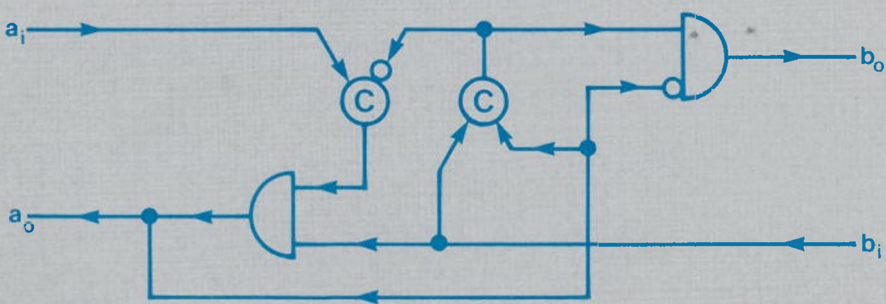
A formalism for concurrent processes

DOI:
10.6100/IR244640

Document status and date:
Published: 01/01/1986

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

A Formalism for Concurrent Processes



Anne Kaldewaij

A formalism for Concurrent Processes

A Formalism
for
Concurrent Processes

PROEFSCHRIFT

TER VERKRIJGING VAN DE GRAAD VAN DOCTOR IN DE TECHNISCHE WETENSCHAPPEN AAN DE TECHNISCHE HOGESCHOOL EINDHOVEN, OP GEZAG VAN DE RECTOR MAGNIFICUS, PROF. DR. F.N. HOOGE, VOOR EEN COMMISSIE AANGEWEZEN DOOR HET COLLEGE VAN DEKANEN IN HET OPENBAAR TE VERDEDIGEN OP DINSDAG 6 MEI 1986 TE 16.00 UUR

DOOR

ANNE KALDEWAIJ

GEBOREN TE EINDHOVEN

dit proefschrift is goedgekeurd
door de promotoren
Prof. dr. M. Rem
en
Prof. dr. F.E.J. Kruseman Aretz

Aan Erna,
Kristien, Renske,
Ernst en Byate

Contents

0 Introduction

0.0 General remarks

The material presented in this thesis has its origin in the research of the Eindhoven VLSI Club.
VLSI is a technique of constructing semiconductor chips containing a large number of active, electronic elements. These elements operate concurrently. The ultimate goal of our research is the construction of a so-called silicon compiler: a mechanical translation of algorithms into chips.

In this monograph we present a general formalism for concurrent processes. We show also how it can be applied to the design of circuits. Such a formalism should satisfy certain requirements:

- it should be a mathematical theory in the sense that concepts are defined rigorously and that assertions are proved;
- it should be close to the objects that are formalized. The distance between formalism and implementation should be relatively small;
- it should be manageable.

The formalism used is known as Trace Theory. To a large extent it has been developed by Martin Rem (cf. [18]) and Jan L.A. van de Snepscheut (cf. [19]). Mazurkiewicz ([14]) was one of the first to describe communicating processes in terms of traces. His traces correspond to equivalence classes over our traces.
This thesis comprises a full and coherent treatment of Trace Theory. The formalism is applied to phenomena like deadlock, livelock, and nondeterminism, and is related to the theory of Communicating Sequential Processes as described by C.A.R. Hoare in [8]. Finally, implementation aspects are discussed.

At the end of most sections we present some exercises. Although this is not of common use in doctoral theses, we have at least two reasons for it:

- it permits the reader to get used to the formalism presented;
- it shows which kind of problems can be solved within the theory.

The exercises do not play any role besides those sketched above. There are no references to them and no proofs of theorems are left as exercises.

0.1 Overview

Chapter 1 contains the prerequisite material for all other chapters. Trace structures and processes are introduced as well as operators on these objects. Processes are related to state graphs. It is shown that processes with the same alphabet form a complete lattice. Monotonicity and continuity of operators are discussed.

In Chapter 2 we present a program notation. The treatment is close to that of [19]. Recursive components are introduced and fixpoint theory is applied to them. Specifications of processes are discussed in Chapter 3. It is shown how program texts may be derived from specifications. These derivations are based on two theorems: the Conjunction-Weave Rule and the Composition Rule. As an example we show how to derive a program that corresponds with the language generated by a given context-free grammar.

Chapter 4 addresses deadlock. Deadlock is defined in terms of trace structures.

In Chapter 5 we discuss livelock and nondeterminism. Nondeterminism arises when a process is projected on a set of events, i.e., when events not in that set are concealed. We define so-called transparent sets of events. If projections are confined to these sets nondeterminism does not occur. In the absence of livelock transparency is closed under intersection. We show the relation between processes in our formalism and those defined by C.A.R. Hoare (cf. [8]).

In Chapter 6 implementation aspects are considered. Parts of it are based on work by Alain J. Martin ([12]). We present some circuits that correspond to given program texts. The circuits derived are delay-insensitive in the sense that their behaviour does not depend on delays in wires and switching elements.

0.2 Some notational conventions

In this monograph a slightly unconventional notation for variable-binding constructs is used. It will be explained here informally. Universal quantification is denoted by

$$(\mathbf{A}\, x : R : E)$$

where **A** is the quantifier, x is a list of bound variables, R is a predicate, and E is the quantified expression. Both R and E will, in general, contain variables from x. R delineates the range of the bound variables. Expression E is defined for values that satisfy R.

Existential quantification is denoted similarly using quantifier **E**.

For expressions E and G, an expression of the form $E \Rightarrow G$ will often be proved in a number of steps by the introduction of intermediate expressions. For instance, we can prove $E \Rightarrow G$ by proving $E \equiv F$ and $F \Rightarrow G$ for some expression F. In order not to be forced to write down expressions like F twice, we record proofs like these as follows.

$$
\begin{array}{ll}
 & E \\
= & \{\text{ hint why } E \equiv F \} \\
 & F \\
\Rightarrow & \{\text{ hint why } F \Rightarrow G \} \\
 & G
\end{array}
$$

These notations have been adopted from [4].

1 Trace structures

1.0 Introduction

In this chapter we define the basic concepts and structures that form the foundation of our treatment of concurrent processes. As an example we consider a one-place buffer which is initially empty. When such a buffer interacts with its environment the following events may be observed.

a : a value enters the buffer
b : a value is retrieved from the buffer

A possible sequence of events is $a\ b\ a\ b\ a$. The set of all possible sequences of events consists of the finite-length alternations of a and b that do not start with b.

In our formalism such a buffer is specified by a pair of sets:

the set of possible events that may occur, and
the set of sequences of events that may be observed.

We define operators on those pairs and we derive algebraic properties thereof.

1.1 Alphabets and trace sets

We assume the existence of a set Ω, the universe. Elements of Ω are called *symbols* . Subsets of Ω are called *alphabets* .
The set of all finite-length sequences of elements of Ω is, as usual, denoted by Ω^*. The empty sequence is denoted by ϵ. For an alphabet A , A^* is defined similarly. Notice that $\varnothing^* = \{\epsilon\}$.
Elements of Ω^* are called *traces* . Subsets of Ω^* are called *trace sets* .

We shall use the following conventions.

Small and capital letters near the beginning of the Latin alphabet denote symbols and alphabets respectively.
Small and capital letters near the end of the Latin alphabet denote traces and trace sets respectively.

The *concatenation* of traces s and t is obtained by placing t to the right of s, and is denoted by st . The set Ω^*, together with the operation concatenation is also known as the free monoid generated by Ω, cf. [5].

The *projection* of a trace t on an alphabet A, denoted by $t\upharpoonright A$, is defined as follows.

$$\begin{aligned} \epsilon\upharpoonright A &= \epsilon \\ (sa)\upharpoonright A &= s\upharpoonright A && \text{if } a \notin A \\ (sa)\upharpoonright A &= (s\upharpoonright A)a && \text{if } a \in A \end{aligned}$$

We write $t\upharpoonright b$ as a shorthand for $t\upharpoonright\{b\}$. In order to save parentheses, we give concatenation the highest priority of all operators.

The projection of a trace set X on an alphabet A, denoted by $X\upharpoonright A$, is the trace set $\{t \mid t \in \Omega^* \wedge (\mathbf{E}\, u : u \in X : t = u\upharpoonright A)\}$.

The length of a trace t, denoted by $l(t)$, is defined by

$$\begin{aligned} l(\epsilon) &= 0 \\ l(sa) &= l(s)+1 \end{aligned}$$

Trace s is called a *prefix* of t, denoted by $s \leqslant t$, if

$$(\mathbf{E}\, u : u \in \Omega^* : su = t)$$

The *prefix closure* of a trace set X, denoted by $pref(X)$, is the trace set consisting of all prefixes of elements of X:

$$pref(X) = \{s \mid s \in \Omega^* \wedge (\mathbf{E}\, t : t \in X : s \leqslant t)\}$$

Trace set X is called *prefix-closed* if $X = pref(X)$.

Example 1.1.0

Let $\Omega = \{a, b, c, d\}$, $A = \{a, b\}$, $s = ba$, $t = bad$, and $X = \{c, dba\}$. Then A is an alphabet, s and t are traces, and X is a trace set.
We have

$s \leqslant t$

$s\upharpoonright A = s \wedge t\upharpoonright A = s$

$s \in A^* \wedge t \notin A^*$

$X\upharpoonright A = \{\epsilon, ba\}$

$pref(X) = \{\epsilon, c, d, db, dba\}$

X is not prefix-closed

$pref(X)$ is prefix-closed.

(End of Example)

We now list a number of properties. According to our notational convention, a and b are symbols, s, t, and u are traces, A and B are alphabets, X and Y are trace sets.

Property 1.1.1 (concatenation)

0 $s\epsilon = \epsilon s = s$

1 $(st)u = s(tu)$

2 $as = bt \;\equiv\; a = b \wedge s = t$
$st = su \;\equiv\; t = u$
$ts = us \;\equiv\; t = u$

3 $s \neq \epsilon \;\equiv\; (\mathbf{E}\, c, v : c \in \Omega \wedge v \in \Omega^* : s = cv)$
$s \neq \epsilon \;\equiv\; (\mathbf{E}\, c, v : c \in \Omega \wedge v \in \Omega^* : s = vc)$

(End of Property)

Property 1.1.2 (projection)

0 $s\restriction A \in A^*$

1 $st\restriction A = (s\restriction A)(t\restriction A)$

2 $s \leqslant t \Rightarrow s\restriction A \leqslant t\restriction A$

3 $s \in A^* \equiv s\restriction A = s$

4 $s\restriction A\restriction B = s\restriction(A \cap B) = s\restriction B\restriction A$

5 $X \subseteq Y \Rightarrow X\restriction A \subseteq Y\restriction A$

6 $s\restriction\varnothing = \epsilon$

(End of Property)

Property 1.1.3 (prefix)

$(\Omega^*, \leqslant)$ is a partially ordered set with least element ϵ :

0 $s \leqslant s$

1 $s \leqslant t \wedge t \leqslant u \Rightarrow s \leqslant u$

2 $s \leqslant t \wedge t \leqslant s \Rightarrow s = t$

3 $\epsilon \leqslant s$

(End of Property)

Property 1.1.4 (prefix-closure)

0 $X \subseteq pref(X)$

1 $X \subseteq Y \Rightarrow pref(X) \subseteq pref(Y)$

2 $pref(pref(X)) = pref(X)$

3 $pref(X\upharpoonright A) = pref(X)\upharpoonright A$

(End of Property)

Property 1.1.5 (length)

0 $l(st) = l(s)+l(t)$

1 $s \leqslant t \Rightarrow l(s) \leqslant l(t)$

2 $l(s\upharpoonright A) \leqslant l(s)$

(End of Property)

As an example we prove Property 1.1.4.2 , $pref(pref(X)) = pref(X)$, which is equivalent to '$pref(X)$ is prefix-closed'.

Proof

For all traces t, we have

$$
\begin{array}{ll}
 & t \in pref(pref(X)) \\
= & \quad \{ \text{ definition of } pref \ \} \\
 & (\mathbf{E}\, u : u \in pref(X) : t \leqslant u) \\
= & \quad \{ \text{ definition of } pref \ \} \\
 & (\mathbf{E}\, u : (\mathbf{E}\, v : v \in X : u \leqslant v) : t \leqslant u) \\
= & \quad \{ \text{ predicate calculus } \} \\
 & (\mathbf{E}\, u : u \in \Omega^* : (\mathbf{E}\, v : v \in X : u \leqslant v \ \wedge\ t \leqslant u)) \\
\Rightarrow & \quad \{ \text{ transitivity of } \leqslant \text{ , Property 1.1.3.1 } \} \\
 & (\mathbf{E}\, u : u \in \Omega^* : (\mathbf{E}\, v : v \in X : t \leqslant v)) \\
= & \quad \{ \text{ predicate calculus } \} \\
 & (\mathbf{E}\, v : v \in X : t \leqslant v) \\
= & \quad \{ \text{ definition of } pref \ \} \\
 & t \in pref(X)
\end{array}
$$

Hence, $pref(pref(X)) \subseteq pref(X)$. Since $pref(X) \subseteq pref(pref(X))$, cf. Property 1.1.4.0, we have $pref(pref(X)) = pref(X)$.

(End of Proof)

Finally, we prove a general theorem on traces.

Theorem 1.1.6 (Lift Theorem)

For all traces s and t, and alphabets A and B, we have

$$s \in A^* \wedge t \in B^* \wedge s\upharpoonright B = t\upharpoonright A \equiv (\mathbf{E}\, u : u \in (A \cup B)^* : u\upharpoonright A = s \wedge u\upharpoonright B = t)$$

Proof

We derive for any trace u

$$\begin{array}{ll} & u\upharpoonright A = s \wedge u\upharpoonright B = t \\ = & \quad \{ \text{property of projection, 1.1.2.0} \} \\ & u\upharpoonright A = s \wedge u\upharpoonright B = t \wedge s \in A^* \wedge t \in B^* \\ \Rightarrow & \quad \{ \text{application of } \upharpoonright A \text{ and } \upharpoonright B \} \\ & u\upharpoonright A\upharpoonright B = s\upharpoonright B \wedge u\upharpoonright B\upharpoonright A = t\upharpoonright A \wedge s \in A^* \wedge t \in B^* \\ \Rightarrow & \quad \{ \text{property of projection, transitivity of } = \} \\ & s\upharpoonright B = t\upharpoonright A \wedge s \in A^* \wedge t \in B^* \end{array}$$

Hence,

$$(\mathbf{E}\, u : u \in (A \cup B)^* : u\upharpoonright A = s \wedge u\upharpoonright B = t) \Rightarrow s \in A^* \wedge t \in B^* \wedge s\upharpoonright B = t\upharpoonright A$$

We prove the converse of the above implication by induction on $l(s) \cdot l(t)$.

Base $l(s) \cdot l(t) = 0$

Then $s = \epsilon \vee t = \epsilon$. For reasons of symmetry we assume $s = \epsilon$, and we derive

$$\begin{array}{ll} & s\upharpoonright B = t\upharpoonright A \wedge t \in B^* \\ = & \quad \{ \text{property of projection, 1.1.2.3} \} \\ & s\upharpoonright B = t\upharpoonright A \wedge t\upharpoonright B = t \\ = & \quad \{ s = \epsilon \text{, definition of projection} \} \\ & s = t\upharpoonright A \wedge t\upharpoonright B = t \\ \Rightarrow & \quad \{ B^* \subseteq (A \cup B)^* \} \\ & (\mathbf{E}\, u : u \in (A \cup B)^* : u\upharpoonright A = s \wedge u\upharpoonright B = t) \end{array}$$

Step $l(s) \cdot l(t) > 0$

Then $s \neq \epsilon \wedge t \neq \epsilon$. By Property 1.1.1.3 we can choose $a \in A$, $b \in B$, $v \in A^*$, and $w \in B^*$ such that $s = av \wedge t = bw$. We consider two cases.

(i) $a \notin B \vee b \notin A$. For reasons of symmetry we assume $a \notin B$, and we derive

$$\begin{array}{ll} & s\upharpoonright B = t\upharpoonright A \wedge s \in A^* \wedge t \in B^* \\ = & \quad \{ s = av,\ a \notin B \} \end{array}$$

$$
\begin{array}{ll}
& v{\upharpoonright}B = t{\upharpoonright}A \ \wedge \ v \in A^* \wedge \ t \in B^* \\
\Rightarrow & \{ \text{ induction hypothesis, } l(v) \cdot l(t) < l(s) \cdot l(t) \ \} \\
& (\mathbf{E}\, u : u \in (A \cup B)^* : u{\upharpoonright}A = v \ \wedge \ u{\upharpoonright}B = t \,) \\
= & \{ \, a \in A \ \wedge \ a \notin B \ \} \\
& (\mathbf{E}\, u : u \in (A \cup B)^* : au{\upharpoonright}A = av \ \wedge \ au{\upharpoonright}B = t \,) \\
\Rightarrow & \{ \text{ renaming the dummy, } s = av \ \} \\
& (\mathbf{E}\, u : u \in (A \cup B)^* : u{\upharpoonright}A = s \ \wedge \ u{\upharpoonright}B = t \,)
\end{array}
$$

(ii) $a \in B \ \wedge \ b \in A$. We derive

$$
\begin{array}{ll}
& s{\upharpoonright}B = t{\upharpoonright}A \ \wedge \ s \in A^* \wedge \ t \in B^* \\
= & \{ \, s = av \ \wedge \ t = bw \ \wedge \ a \in B \ \wedge \ b \in A \ \} \\
& a(v{\upharpoonright}B) = b(w{\upharpoonright}A) \wedge \ v \in A^* \wedge \ w \in B^* \\
= & \{ \text{ property of concatenation, 1.1.1.2 } \} \\
& a = b \ \wedge \ v{\upharpoonright}B = w{\upharpoonright}A \ \wedge \ v \in A^* \wedge \ w \in B^* \\
\Rightarrow & \{ \text{ induction hypothesis, } l(v) \cdot l(w) < l(s) \cdot l(t) \ \} \\
& a = b \ \wedge \ (\mathbf{E}\, u : u \in (A \cup B)^* : u{\upharpoonright}A = v \ \wedge \ u{\upharpoonright}B = w \,) \\
= & \{ \, a \in A \, , \, b \in B \ \} \\
& a = b \ \wedge \ (\mathbf{E}\, u : u \in (A \cup B)^* : au{\upharpoonright}A = av \ \wedge \ bu{\upharpoonright}B = bw \,) \\
\Rightarrow & \{ \text{ substitution } \} \\
& (\mathbf{E}\, u : u \in (A \cup B)^* : au{\upharpoonright}A = s \ \wedge \ au{\upharpoonright}B = t \,) \\
\Rightarrow & \{ \text{ renaming the dummy } \} \\
& (\mathbf{E}\, u : u \in (A \cup B)^* : u{\upharpoonright}A = s \ \wedge \ u{\upharpoonright}B = t \,)
\end{array}
$$

(End of Proof)

Theorem 1.1.6 may be phrased as follows.
The diagram of Figure 1.0 may be lifted up to the commutative diagram of Figure 1.1 .

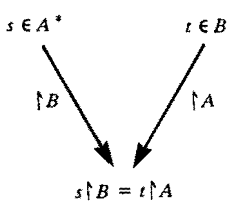


Figure 1.0

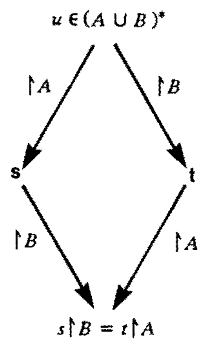


Figure 1.1

Exercises

0. Prove:
   (i) $A^* \cap B^* = (A \cap B)^*$
   (ii) $A^* \cup B^* \subseteq (A \cup B)^*$

1. Prove:
   (i) $pref(X \cup Y) = pref(X) \cup pref(Y)$
   (ii) $pref(X \cap Y) \subseteq pref(X) \cap pref(Y)$

2. Prove: $\epsilon \in pref(X) \equiv X \neq \varnothing$

3. Show that the intersection as well as the union of prefix-closed trace sets are prefix-closed.

4. Prove or disprove:
   (i) $s\lceil A \leqslant t\lceil A \ \wedge\ s\lceil B \leqslant t\lceil B \ \Rightarrow\ s\lceil(A \cup B) \leqslant t\lceil(A \cup B)$
   (ii) $s\lceil A \leqslant t\lceil A \ \vee\ s\lceil B \leqslant t\lceil B \ \Rightarrow\ s\lceil(A \cap B) \leqslant t\lceil(A \cap B)$

5. Prove or disprove:
   (i) $(X \cup Y)\lceil A = (X\lceil A) \cup (Y\lceil A)$
   (ii) $(X \cap Y)\lceil A = (X\lceil A) \cap (Y\lceil A)$

6. Prove:
   $s \in (A \cap C)^* \wedge t \in (B \cap C)^* \wedge s\lceil B = t\lceil A$
   $\Rightarrow (\mathbf{E}\, u : u \in (A \cup B)^* : u\lceil A = s \ \wedge\ u\lceil(B \cap C) = t)$

(End of Exercises)

1.2 Trace structures

A *trace structure* is a pair $<A, X>$ in which A is an alphabet and X is a subset of A^*. We call A the alphabet of the trace structure and we call X the trace set of the trace structure. If T is a trace structure we denote its alphabet by $\mathbf{a}T$ and its trace set by $\mathbf{t}T$, i.e. $T = <\mathbf{a}T, \mathbf{t}T>$.

As a notational convention we shall use capital letters not too far from the end of the Latin alphabet to denote trace structures.

The *prefix closure* of a trace structure T, denoted by $pref(T)$, is the trace structure $<\mathbf{a}T, pref(\mathbf{t}T)>$. T is called *prefix - closed* whenever $\mathbf{t}T$ is prefix-closed. T is called *non-empty* if $\mathbf{t}T \neq \varnothing$.

A non-empty prefix-closed trace structure is also called a *process* . Let T be a process, then T specifies a mechanism in the following way.

> The alphabet of T corresponds to the set of events the mechanism may be involved in. We assume events to be atomic: they have no duration and they do not overlap.
>
> With the mechanism in operation a so-called *trace thus far generated* is associated. Initially, this trace is the empty trace. On the occurrence of an event the trace thus far generated is extended with the symbol associated with that event. At any moment, the trace thus far generated belongs to the trace set of T.
>
> We do not distinguish between events that are initiated by the mechanism and those that are initiated by the environment of the mechanism. If s is the trace thus far generated and $sa \in \mathbf{t}T$ then the event associated with a may happen.

Example 1.2.0

Consider a one-place buffer which is initially empty. We specify the buffer by means of a process T. Possible events are

a : a value enters the buffer
b : a value is retrieved from the buffer

Hence, $\mathbf{a}T = \{a, b\}$.
Let $t \in \mathbf{t}T$. Since values can only be retrieved if they have been entered, we have $l(t\upharpoonright a) - l(t\upharpoonright b) \geqslant 0$. From the fact that the buffer is a one-place buffer we infer $l(t\upharpoonright a) - l(t\upharpoonright b) \leqslant 1$. These restrictions should hold for all t, $t \in \mathbf{t}T$, and their prefixes. Our specification becomes

$$T = <\{a, b\}, \{t \mid t \in \{a, b\}^* \wedge (\mathbf{A}\, s : s \leqslant t : 0 \leqslant l(s\upharpoonright a) - l(s\upharpoonright b) \leqslant 1)\}>$$

T may also be interpreted as the specification of a binary semaphore (cf. [2]), initialized at zero.

The interpretation of the symbols is

a : a V-operation on the semaphore
b : a P-operation on the semaphore

(End of Example)

Example 1.2.1

In the previous example we did not consider the values that are transmitted. In this example we define process U that specifies a *one - place, one - bit* buffer. Possible events are

$a0$: a zero enters the buffer
$a1$: a one enters the buffer
$b0$: a zero is retrieved from the buffer
$b1$: a one is retrieved from the buffer

The same arguments as used in the previous example yield

$$\mathbf{a}U = \{a0, a1, b0, b1\}$$
$$\begin{aligned}\mathbf{t}U = \{t \mid t \in \{a0, a1, b0, b1\}^* \wedge (\mathbf{A}\, s : s \leqslant t : \; & 0 \leqslant l(s\restriction a0) - l(s\restriction b0) \leqslant 1 \\ \wedge \; & 0 \leqslant l(s\restriction a1) - l(s\restriction b1) \leqslant 1 \\ \wedge \; & 0 \leqslant l(s\restriction\{a0, a1\}) - l(s\restriction\{b0, b1\}) \leqslant 1)\}\end{aligned}$$

(End of Example)

There is a one-to-one correspondence between the set of trace structures with alphabet A and $P(A^*)$, the power set of A^*, viz.

$<A, X>$ is a trace structure $\equiv$ $X \subseteq A^*$

According to the structure of $P(A^*)$ we define inclusion, intersection, and union for trace structures *with equal alphabets*, and we denote these with the usual symbols:

$$<A, X> \cup <A, Y> \;=\; <A, X \cup Y>$$
$$<A, X> \cap <A, Y> \;=\; <A, X \cap Y>$$

$$T \subseteq U \quad \equiv \quad \mathbf{a}T = \mathbf{a}U \wedge \mathbf{t}T \subseteq \mathbf{t}U$$

In section 1.3 we have a closer look at the set of processes with alphabet A.

The projection on an alphabet is extended to trace structures by

$$T\restriction A \;=\; <\mathbf{a}T \cap A, \mathbf{t}T\restriction A>$$

Finally, we define the following processes. For an alphabet A the trace structures $STOP(A)$ and $RUN(A)$ are defined by

$$STOP(A) = \langle A, \{\epsilon\} \rangle$$
$$RUN(A) = \langle A, A^* \rangle$$

Process $STOP(\emptyset)$ is also denoted by $STOP$.

For symbols a and b process $SEM_1(a,b)$ is defined by

$$SEM_1(a,b) = \langle \{a,b\}, \{t \mid t \in \{a,b\}^* \wedge (\mathbf{A}\, s : s \leq t : 0 \leq l(s\lceil a) - l(s\lceil b) \leq 1)\} \rangle$$

(cf. Example 1.2.0)

Exercises

0. Give a mechanistic appreciation of $RUN(A)$, $STOP(A)$, and $STOP$.

1. Prove:
   (i) $RUN(A)\lceil B = RUN(A \cap B)$
   (ii) $STOP(A)\lceil B = STOP(A \cap B)$
   (iii) $SEM_1(a,b)\lceil a = RUN(a)$
   (iv) $STOP(A) = RUN(A) \equiv A = \emptyset$

2. Specify a two-place buffer.

3. Specify an unbounded buffer.

4. For trace structure T we define trace structure T° by
   $$T^\circ = \langle \mathbf{a}T, \{t \mid (\mathbf{A}\, s : s \leq t : s \in \mathbf{t}T)\} \rangle$$
   Prove:
   (i) $T^\circ \subseteq T$
   (ii) T° is prefix-closed
   (iii) $T \subseteq U \Rightarrow T^\circ \subseteq U^\circ$
   (iv) T° is the largest prefix-closed trace structure contained in T
   (v) $T = T^\circ \equiv T$ is prefix-closed

5. Prove: T is non-empty $\equiv T\lceil\emptyset = STOP$

6. Specify a binary stack, the depth of which is bounded by two.

(End of Exercises)

1.3 Weaving

Consider two mechanisms P and Q specified by (non-empty prefix-closed) trace structures T and U respectively. The behaviour of the *composite* of P and Q should be in accordance with the behaviour of each of the components:

> if t is the trace thus far generated of the composite then $t\restriction \mathbf{a}T$ will be the trace thus far generated of P and $t\restriction \mathbf{a}U$ will be the trace thus far generated of Q. Hence, extension of the trace thus far generated with a common symbol of $\mathbf{a}T$ and $\mathbf{a}U$ is possible if and only if both P and Q agree upon that symbol. Extension with a non-common symbol depends on one of the components only.

In terms of trace structures this is captured in the following definition.

The *weave* of trace structures T and U, denoted by $T\,\mathbf{w}\,U$, is defined by

$$T\,\mathbf{w}\,U \;=\; <\mathbf{a}T \cup \mathbf{a}U, \{t \mid t \in (\mathbf{a}T \cup \mathbf{a}U)^* \wedge t\restriction \mathbf{a}T \in \mathbf{t}T \wedge t\restriction \mathbf{a}U \in \mathbf{t}U\}>$$

Example 1.3.0

$$\begin{aligned} & <\{a,b\},\{ab\}> \;\mathbf{w}\; <\{c,d\},\{cd\}> \\ = \; & <\{a,b,c,d\},\{abcd, acbd, acdb, cabd, cadb, cdab\}> \end{aligned}$$

$$\begin{aligned} & <\{a,b\},\{b, ba, abb\}> \;\mathbf{w}\; <\{b,c\},\{b, cb\}> \\ = \; & <\{a,b,c\}, \{b, ba, cb, cba\}> \end{aligned}$$

(End of Example)

Example 1.3.1

$SEM_1(a,b) \;=\; <\{a,b\},\{\epsilon, a, ab, aba, \cdots\}>$
$SEM_1(b,c) \;=\; <\{b,c\},\{\epsilon, b, bc, bcb, \cdots\}>$, hence,

$$\begin{aligned} & \mathbf{t}(SEM_1(a,b)\,\mathbf{w}\,SEM_1(b,c)) \\ = \quad & \{\text{ definition of weaving }\} \\ & \{t \mid t \in \{a,b,c\}^* \wedge t\restriction\{a,b\} \in \mathbf{t}SEM_1(a,b) \wedge t\restriction\{b,c\} \in \mathbf{t}SEM_1(b,c)\} \\ = \quad & \{\text{ definition of } SEM_1 \} \\ & \{\epsilon, a, ab, aba, abc, abac, abca, abacb, abcab, \cdots\} \end{aligned}$$

Since $t\restriction\{a,b\} \in \mathbf{t}SEM_1(a,b)$ implies $0 \leqslant l(t\restriction a) - l(t\restriction b) \leqslant 1$
and $t\restriction\{b,c\} \in \mathbf{t}SEM_1(b,c)$ implies $0 \leqslant l(t\restriction b) - l(t\restriction c) \leqslant 1$,
we have

$$0 \leqslant l(t\restriction a) - l(t\restriction c) \leqslant 2$$

(End of Example)

Property 1.3.2

Weaving is symmetric, idempotent, and associative:

0 $T \mathbf{w} U = U \mathbf{w} T$

1 $T \mathbf{w} T = T$

2 $(T \mathbf{w} U) \mathbf{w} V = T \mathbf{w} (U \mathbf{w} V)$

(End of Property)

Property 1.3.3

$$\mathbf{a}U \subseteq \mathbf{a}T \Rightarrow T \mathbf{w} U = \langle \mathbf{a}T, \{t \mid t \in \mathbf{t}T \wedge t{\restriction}\mathbf{a}U \in \mathbf{t}U\}\rangle$$

(End of Property)

Property 1.3.4

0 $T \mathbf{w}\, STOP = T$

1 $T \mathbf{w} (T{\restriction}A) = T$

2 $A \subseteq \mathbf{a}T \Rightarrow T \mathbf{w}\, RUN(A) = T$

3 $\mathbf{a}T \subseteq A \Rightarrow T \mathbf{w} \langle A, \varnothing\rangle = \langle A, \varnothing\rangle$

4 $\mathbf{a}T \subseteq A \wedge \epsilon \in \mathbf{t}T \Rightarrow T \mathbf{w}\, STOP(A) = STOP(A)$

Proof

0. We derive

$$\begin{array}{ll} & T \mathbf{w}\, STOP \\ = & \quad \{ \text{Property 1.3.3, } \mathbf{a}STOP = \varnothing \} \\ & \langle \mathbf{a}T, \{t \mid t \in \mathbf{t}T \wedge t{\restriction}\varnothing \in \mathbf{t}STOP\}\rangle \\ = & \quad \{ \text{Property 1.1.2.6, } \mathbf{t}STOP = \{\epsilon\} \} \\ & T \end{array}$$

1. We derive

$$\begin{array}{ll} & T \mathbf{w} (T{\restriction}A) \\ = & \quad \{ \text{Property 1.3.3, } \mathbf{a}(T{\restriction}A) \subseteq \mathbf{a}T \} \\ & \langle \mathbf{a}T, \{t \mid t \in \mathbf{t}T \wedge t{\restriction}A \in \mathbf{t}(T{\restriction}A)\}\rangle \\ = & \quad \{ \text{definition of projection} \} \\ & T \end{array}$$

2. Assume $A \subseteq \mathbf{a}T$. We derive

$$
\begin{array}{ll}
 & T \mathbf{w} \mathit{RUN}(A) \\
= & \quad \{ \text{Property 1.3.3}, A \subseteq \mathbf{a}T \} \\
 & <\mathbf{a}T, \{t \mid t \in \mathbf{t}T \wedge t\restriction A \in A^*\}> \\
= & \quad \{ \text{Property 1.1.2.0} \} \\
 & T
\end{array}
$$

3. Assume $\mathbf{a}T \subseteq A$. We derive

$$
\begin{array}{ll}
 & T \mathbf{w} <A, \varnothing> \\
= & \quad \{ \text{Property 1.3.3}, \mathbf{a}T \subseteq A \} \\
 & <A, \{t \mid t \in \varnothing \wedge t\restriction \mathbf{a}T \in \mathbf{t}T\}> \\
= & \quad \{ \text{calculus} \} \\
 & <A, \varnothing>
\end{array}
$$

4. Assume $\mathbf{a}T \subseteq A \wedge \epsilon \in \mathbf{t}T$. We derive

$$
\begin{array}{ll}
 & T \mathbf{w} \mathit{STOP}(A) \\
= & \quad \{ \text{Property 1.3.3}, \mathbf{a}T \subseteq A \} \\
 & <A, \{t \mid t \in \mathbf{t}\mathit{STOP}(A) \wedge t\restriction \mathbf{a}T \in \mathbf{t}T\}> \\
= & \quad \{ \mathbf{t}\mathit{STOP}(A) = \{\epsilon\} \text{ and } \epsilon \in \mathbf{t}T \} \\
 & \mathit{STOP}(A)
\end{array}
$$

(End of Proof)

The definition of weaving can be extended to sets of trace structures. Let S be a set of trace structures. The weave of the elements of S, denoted by $(\mathbf{W}\, T : T \in S : T)$ is the trace structure $<A, X>$ where

$$
\begin{aligned}
A &= (\cup\, T : T \in S : \mathbf{a}T) \\
X &= \{t \mid t \in A^* \wedge (\mathbf{A}\, T : T \in S : t\restriction \mathbf{a}T \in \mathbf{t}T)\}
\end{aligned}
$$

By definition we have $(\mathbf{W}\, T : T \in \varnothing : T) = \mathit{STOP}$, the unit element of weaving, cf. Property 1.3.4.0 .

The weave of trace structures expresses a synchronized interleaving. Apparently, the intersection of the alphabets of the trace structures involved plays an important role. This role is made more precise in the following theorems.

Theorem 1.3.5

Let T and U be trace structures and let A be an alphabet, then

$$T \,\mathbf{w}\, (U \upharpoonright A) \supseteq (T \,\mathbf{w}\, U) \upharpoonright (\mathbf{a}T \cup (\mathbf{a}U \cap A))$$

Proof

The alphabets of both sides are equal, viz. $\mathbf{a}T \cup (\mathbf{a}U \cap A)$.

Let $t \in \mathbf{t}(T \,\mathbf{w}\, U) \upharpoonright (\mathbf{a}T \cup (\mathbf{a}U \cap A))$ and let w be such that $w \in \mathbf{t}(T \,\mathbf{w}\, U)$ and $t = w \upharpoonright (\mathbf{a}T \cup (\mathbf{a}U \cap A))$. We derive

$$\begin{array}{ll}
 & t = w \upharpoonright (\mathbf{a}T \cup (\mathbf{a}U \cap A)) \\
\Rightarrow & \{ \text{application of projection} \} \\
 & t \upharpoonright \mathbf{a}T = w \upharpoonright (\mathbf{a}T \cup (\mathbf{a}U \cap A)) \upharpoonright \mathbf{a}T \ \wedge\ t \upharpoonright (\mathbf{a}U \cap A) = w \upharpoonright (\mathbf{a}T \cup (\mathbf{a}U \cap A)) \upharpoonright (\mathbf{a}U \cap A) \\
= & \{ \text{Property 1.1.2.4} \} \\
 & t \upharpoonright \mathbf{a}T = w \upharpoonright \mathbf{a}T \ \wedge\ t \upharpoonright (\mathbf{a}U \cap A) = w \upharpoonright \mathbf{a}U \upharpoonright A \\
\Rightarrow & \{ w \in \mathbf{t}(T \,\mathbf{w}\, U) \} \\
 & t \upharpoonright \mathbf{a}T \in \mathbf{t}T \ \wedge\ t \upharpoonright (\mathbf{a}U \cap A) \in \mathbf{t}U \upharpoonright A \\
= & \{ \text{definition of weaving} \} \\
 & t \in \mathbf{t}(T \,\mathbf{w}\, (U \upharpoonright A))
\end{array}$$

Hence, $\mathbf{t}(T \,\mathbf{w}\, U) \upharpoonright (\mathbf{a}T \cup (\mathbf{a}U \cap A)) \subseteq \mathbf{t}(T \,\mathbf{w}\, (U \upharpoonright A))$

(End of Proof)

Theorem 1.3.6

Let T and U be trace structures, and let A be an alphabet such that $\mathbf{a}T \cap \mathbf{a}U \subseteq A$, then

$$T \,\mathbf{w}\, (U \upharpoonright A) = (T \,\mathbf{w}\, U) \upharpoonright (\mathbf{a}T \cup (\mathbf{a}U \cap A))$$

Proof

As a consequence of Theorem 1.3.5 it suffices to prove

$$\mathbf{t}(T \,\mathbf{w}\, (U \upharpoonright A)) \subseteq \mathbf{t}(T \,\mathbf{w}\, U) \upharpoonright (\mathbf{a}T \cup (\mathbf{a}U \cap A))$$

Let $t \in \mathbf{t}(T \,\mathbf{w}\, (U \upharpoonright A))$, then $t \upharpoonright \mathbf{a}T \in \mathbf{t}T \ \wedge\ t \upharpoonright (\mathbf{a}U \cap A) \in \mathbf{t}U \upharpoonright A$.
Let $v \in \mathbf{t}U$ be such that $t \upharpoonright (\mathbf{a}U \cap A) = v \upharpoonright A$.

We have to show the existence of w, $w \in \mathbf{t}(T \,\mathbf{w}\, U)$, such that $t = w \upharpoonright (\mathbf{a}T \cup (\mathbf{a}U \cap A))$, and we will do so by using the Lift Theorem (1.1.6).

We first derive

$$
\begin{aligned}
& ((\mathbf{a}U \cap A) \cup \mathbf{a}T) \cap \mathbf{a}U \\
= \quad & \{ \text{ set calculus } \} \\
& (\mathbf{a}U \cap A) \cup (\mathbf{a}U \cap \mathbf{a}T) \\
= \quad & \{ \mathbf{a}U \cap \mathbf{a}T \subseteq A \} \\
& \mathbf{a}U \cap A
\end{aligned}
$$

Hence, cf. Figure 1.2,

$$
\begin{aligned}
& v \upharpoonright (\mathbf{a}T \cup (\mathbf{a}U \cap A)) \\
= \quad & \{ v \in \mathbf{t}U \} \\
& v \upharpoonright \mathbf{a}U \upharpoonright (\mathbf{a}T \cup (\mathbf{a}U \cap A)) \\
= \quad & \{ \text{ see above } \} \\
& v \upharpoonright A \upharpoonright \mathbf{a}U \\
= \quad & \{ \text{ definition of } v \} \\
& t \upharpoonright (\mathbf{a}U \cap A) \upharpoonright \mathbf{a}U \\
= \quad & \{ \mathbf{a}U \cap A \subseteq \mathbf{a}U \} \\
& t \upharpoonright (\mathbf{a}U \cap A) \\
= \quad & \{ \text{ see above } \} \\
& t \upharpoonright ((\mathbf{a}U \cap A) \cup \mathbf{a}T) \upharpoonright \mathbf{a}U \\
= \quad & \{ t \in \mathbf{t}(T \mathbin{\mathbf{w}} (U \upharpoonright A)) \} \\
& t \upharpoonright \mathbf{a}U
\end{aligned}
$$

$$
\begin{tikzcd}
& w \arrow[dl, "\upharpoonright(\mathbf{a}T \cup (\mathbf{a}U \cap A))"'] \arrow[dr, "\upharpoonright \mathbf{a}U"] & \\
t \arrow[dr, "\upharpoonright \mathbf{a}U"'] & & v \arrow[dl, "\upharpoonright(\mathbf{a}T \cup (\mathbf{a}U \cap A))"] \\
& t \upharpoonright \mathbf{a}U = v \upharpoonright (\mathbf{a}T \cup (\mathbf{a}U \cap A)) &
\end{tikzcd}
$$

Figure 1.2

Hence, we may apply the Lift Theorem, yielding $w \in (\mathbf{a}T \cup \mathbf{a}U)^*$ such that

$$w \upharpoonright (\mathbf{a}T \cup (\mathbf{a}U \cap A)) = t \quad \text{and} \quad w \upharpoonright \mathbf{a}U = v$$

From

$$
\begin{aligned}
& w \upharpoonright \mathbf{a}T \\
= \quad & \{ \mathbf{a}T \subseteq \mathbf{a}T \cup (\mathbf{a}U \cap A) \} \\
& w \upharpoonright (\mathbf{a}T \cup (\mathbf{a}U \cap A)) \upharpoonright \mathbf{a}T \\
= \quad & \{ \text{ definition of } w \} \\
& t \upharpoonright \mathbf{a}T \\
\in \quad & \{ t \in \mathbf{t}(T \mathbin{\mathbf{w}} (U \upharpoonright A)) \} \\
& \mathbf{t}T
\end{aligned}
$$

and

$$w \upharpoonright \mathbf{a}U = v \in \mathbf{t}U$$

we infer $w \in \mathbf{t}(T \mathbf{w} U)$, and since $t = w \upharpoonright (\mathbf{a}T \cup (\mathbf{a}U \cap A))$, we conclude

$$t \in \mathbf{t}(T \mathbf{w} U) \upharpoonright (\mathbf{a}T \cup (\mathbf{a}U \cap A)).$$

(End of Proof)

Theorem 1.3.7

Let T and U be trace structures and let A be an alphabet such that $\mathbf{a}T \cap \mathbf{a}U \subseteq A$, then

$$(T \mathbf{w} U) \upharpoonright A = (T \upharpoonright A) \mathbf{w} (U \upharpoonright A)$$

Proof

$$
\begin{array}{ll}
 & (T \upharpoonright A) \mathbf{w} (U \upharpoonright A) \\
= & \quad \{ \text{Theorem 1.3.6, } \mathbf{a}T \cap A \cap \mathbf{a}U \subseteq A \} \\
 & ((T \upharpoonright A) \mathbf{w} U) \upharpoonright ((A \cap \mathbf{a}T) \cup (A \cap \mathbf{a}U)) \\
= & \quad \{ \text{set calculus} \} \\
 & ((T \upharpoonright A) \mathbf{w} U) \upharpoonright (A \cap (\mathbf{a}T \cup \mathbf{a}U)) \\
= & \quad \{ \text{Theorem 1.3.6, using the symmetry of weaving} \} \\
 & (T \mathbf{w} U) \upharpoonright (\mathbf{a}U \cup (\mathbf{a}T \cap A)) \upharpoonright (A \cap (\mathbf{a}T \cup \mathbf{a}U)) \\
= & \quad \{ \text{set calculus, property of projection} \} \\
 & (T \mathbf{w} U) \upharpoonright ((\mathbf{a}T \cup \mathbf{a}U) \cap A) \\
= & \quad \{ \mathbf{a}(T \mathbf{w} U) = \mathbf{a}T \cup \mathbf{a}U \} \\
 & (T \mathbf{w} U) \upharpoonright A
\end{array}
$$

(End of Proof)

Theorem 1.3.8

Let T and U be trace structures. Then

0 $\quad pref(T \mathbf{w} U) \subseteq pref(T) \mathbf{w}\, pref(U)$

1 $\quad$ If T and U are processes then $T \mathbf{w} U$ is a process

Proof

0. The alphabets of $pref(T \mathbf{w} U)$ and $pref(T) \mathbf{w}\, pref(U)$ are equal, viz. $\mathbf{a}T \cup \mathbf{a}U$. Let $s \in \mathbf{t}pref(T \mathbf{w} U)$ and let $t \in \mathbf{t}(T \mathbf{w} U)$ be such that $s \leqslant t$. We derive

$t \in \mathbf{t}(T \,\mathbf{w}\, U) \wedge s \leq t$

$=$ { definition of weaving }

$t \upharpoonright \mathbf{a}T \in \mathbf{t}T \wedge t \upharpoonright \mathbf{a}U \in \mathbf{t}U \wedge s \leq t$

$\Rightarrow$ { property of projection, 1.1.2.2 }

$t \upharpoonright \mathbf{a}T \in \mathbf{t}T \wedge t \upharpoonright \mathbf{a}U \in \mathbf{t}U \wedge s \upharpoonright \mathbf{a}T \leq t \upharpoonright \mathbf{a}T \wedge s \upharpoonright \mathbf{a}U \leq t \upharpoonright \mathbf{a}U$

$\Rightarrow$ { definition of *pref* }

$s \upharpoonright \mathbf{a}T \in \mathbf{t}\,pref(T) \wedge s \upharpoonright \mathbf{a}U \in \mathbf{t}\,pref(U)$

$=$ { definition of weaving }

$s \in \mathbf{t}(pref(T) \,\mathbf{w}\, pref(U))$

Hence, $pref(T \,\mathbf{w}\, U) \subseteq pref(T) \,\mathbf{w}\, pref(U)$

1. Assume that T and U are processes. We derive

$pref(T \,\mathbf{w}\, U)$

$\subseteq$ { 0 }

$pref(T) \,\mathbf{w}\, pref(U)$

$=$ { T and U are prefix-closed }

$T \,\mathbf{w}\, U$

$\subseteq$ { property of *pref*, 1.1.4.0 }

$pref(T \,\mathbf{w}\, U)$

from which we infer that $T \,\mathbf{w}\, U$ is prefix-closed. Moreover, we have

$\epsilon \in \mathbf{t}(T \,\mathbf{w}\, U)$

$=$ { definition of weaving }

$\epsilon \in (\mathbf{a}T \cup \mathbf{a}U)^* \wedge \epsilon \upharpoonright \mathbf{a}T \in \mathbf{t}T \wedge \epsilon \upharpoonright \mathbf{a}U \in \mathbf{t}U$

$=$ { definition of projection and of star }

$\epsilon \in \mathbf{t}T \wedge \epsilon \in \mathbf{t}U$

$=$ { T and U are processes }

true

Hence, $T \,\mathbf{w}\, U$ is non-empty and prefix-closed.

(End of Proof)

Theorem 1.3.9

For trace structures T and U such that $\mathbf{a}T \cap \mathbf{a}U = \varnothing$, we have

$$pref(T \mathbin{\mathbf{w}} U) = pref(T) \mathbin{\mathbf{w}} pref(U)$$

Proof

The alphabets are equal. For any t, $t \in (\mathbf{a}T \cup \mathbf{a}U)^*$, we derive

$$\begin{array}{ll}
 & t \in \mathbf{t}(pref(T) \mathbin{\mathbf{w}} pref(U)) \\
= & \quad \{\text{ definition of weaving }\} \\
 & t \restriction \mathbf{a}T \in \mathbf{t}pref(T) \wedge t \restriction \mathbf{a}U \in \mathbf{t}pref(U) \\
= & \quad \{\text{ definition of } pref \ \} \\
 & (\mathbf{E}\, u,v : u \in \mathbf{a}T^* \wedge v \in \mathbf{a}U^* : (t \restriction \mathbf{a}T)u \in \mathbf{t}T \wedge (t \restriction \mathbf{a}U)v \in \mathbf{t}U) \\
= & \quad \{\ \mathbf{a}T \cap \mathbf{a}U = \varnothing\ \} \\
 & (\mathbf{E}\, u,v : u \in \mathbf{a}T^* \wedge v \in \mathbf{a}U^* : tuv \restriction \mathbf{a}T \in \mathbf{t}T \wedge tuv \restriction \mathbf{a}U \in \mathbf{t}U) \\
= & \quad \{\text{ definition of weaving }\} \\
 & (\mathbf{E}\, u,v : u \in \mathbf{a}T^* \wedge v \in \mathbf{a}U^* : tuv \in \mathbf{t}(T \mathbin{\mathbf{w}} U)) \\
\Rightarrow & \quad \{\text{ definition of } pref \ \} \\
 & t \in \mathbf{t}pref(T \mathbin{\mathbf{w}} U)
\end{array}$$

Hence, $pref(T) \mathbin{\mathbf{w}} pref(U) \subseteq pref(T \mathbin{\mathbf{w}} U)$ which yields on account of Theorem 1.3.8.0 $pref(T \mathbin{\mathbf{w}} U) = pref(T) \mathbin{\mathbf{w}} pref(U)$

(End of Proof)

Exercises

0. $T = \langle \{a,b,d,e\}, \{ab, abe, de\} \rangle$, $U = \langle \{b,c,e,f\}, \{bc, bec, fe\} \rangle$, and $V = \langle \{a,b,c\}, \{\epsilon, a, ab, abc\} \rangle$

   Compute $T \mathbin{\mathbf{w}} U$, $T \mathbin{\mathbf{w}} V$, $U \mathbin{\mathbf{w}} V$, and $T \mathbin{\mathbf{w}} U \mathbin{\mathbf{w}} V$.

1. Prove $(T \mathbin{\mathbf{w}} U) \restriction A \subseteq (T \restriction A) \mathbin{\mathbf{w}} (U \restriction A)$ and provide a counterexample for equality.

2. Prove :

   (i) $(T \mathbin{\mathbf{w}} U) \restriction \mathbf{a}T \subseteq T$

   (ii) $T \mathbin{\mathbf{w}} U = ((T \mathbin{\mathbf{w}} U) \restriction \mathbf{a}T) \mathbin{\mathbf{w}} ((T \mathbin{\mathbf{w}} U) \restriction \mathbf{a}U)$

3. For trace structure T we define trace structure T° by

$$T^{\circ} = <\mathbf{a}T, \{t \mid (\mathbf{A}\, s : s \leqslant t : s \in \mathbf{t}T)\}>$$

Prove $(T \mathbf{w} U)^{\circ} = T^{\circ} \mathbf{w} U^{\circ}$

4. Let U and V be trace structures such that $\mathbf{a}U = \mathbf{a}V$. Show that

$$T \mathbf{w} (U \cup V) = (T \mathbf{w} U) \cup (T \mathbf{w} V)$$
$$T \mathbf{w} (U \cap V) = (T \mathbf{w} U) \cap (T \mathbf{w} V)$$

(End of Exercises)

1.4 Blending

The weave of (non-empty prefix-closed) trace structures may be viewed as the specification of the composite of the components they specify. Symbols that belong to more than one of the alphabets of the trace structures are called *internal* symbols.

The other symbols, i.e. those that belong to one of the alphabets only, are called *external* symbols. In the ultimate specification of a composite we want to specify a mechanism without any information about its internal structure:

> in the specification of a four-place buffer we do not want to reflect the fact that it is composed of two two-place buffers, or that it is composed of a one-place buffer and a three-place buffer.

Given a specification of a mechanism, one often tries to decompose that specification in such a way that the mechanism can be obtained by composing simpler mechanisms. In general, there will be interaction between the composing parts. That interaction is, of course, not reflected in the original specification. Hence, we will not specify the composite of a mechanism by the weave of the trace structures involved, but, by the weave followed by projection on the external symbols. This leads to the following definition.

The *blend* of trace structures T and U, denoted by $T \mathbf{b} U$, is defined by

$$T \mathbf{b} U = (T \mathbf{w} U) \restriction (\mathbf{a}T \div \mathbf{a}U)$$

where $\div$ denotes symmetric set difference, i.e. $A \div B = (A \cup B) \setminus (A \cap B)$.

Property 1.4.0

$$\mathbf{a}T \cap \mathbf{a}U = \varnothing \Rightarrow T \mathbf{b} U = T \mathbf{w} U$$

(End of Property)

Property 1.4.1

Blending is symmetric, i.e. $T \mathbf{b} U = U \mathbf{b} T$.

(End of Property)

Property 1.4.2

0 T is non-empty $\Rightarrow$ $T \mathbf{b} T = STOP$

1 $T \mathbf{b} STOP = T$

2 $T \mathbf{b} (T \restriction A) = T \restriction (\mathbf{a}T \setminus A)$

3 $A \subseteq \mathbf{a}T \Rightarrow T \mathbf{b} RUN(A) = T \restriction (\mathbf{a}T \setminus A)$

4 $\epsilon \in \mathbf{t}T \Rightarrow T \mathbf{b} STOP(\mathbf{a}T) = STOP$

Proof

0. Assume T is non-empty. We derive

$T \mathbf{b} T$

= { definition of blending }

$(T \mathbf{w} T) \restriction \varnothing$

= { weaving is idempotent }

$T \restriction \varnothing$

= { T is non-empty, Property 1.1.2.6 }

$STOP$

1. We derive

$T \mathbf{b} STOP$

= { Property 1.4.0, $\mathbf{a}T \cap \mathbf{a}STOP = \varnothing$ }

$T \mathbf{w} STOP$

= { Property 1.3.4.1 }

T

2. We derive

$T \mathbf{b} (T \restriction A)$

= { definition of blending }

$(T \mathbf{w} (T \restriction A)) \restriction (\mathbf{a}T \setminus A)$

= { Property 1.3.4.1 }

$T \restriction (\mathbf{a}T \setminus A)$

3. Assume $A \subseteq \mathbf{a}T$. We derive

$$
\begin{array}{ll}
 & T \ \mathbf{b}\ RUN(A) \\
= & \quad \{ \text{definition of blending}, A \subseteq \mathbf{a}T \} \\
 & (T \ \mathbf{w}\ RUN(A)) \upharpoonright (\mathbf{a}T \setminus A) \\
= & \quad \{ \text{Property 1.3.4.2}, A \subseteq \mathbf{a}T \} \\
 & T \upharpoonright (\mathbf{a}T \setminus A)
\end{array}
$$

4. Assume $\epsilon \in \mathbf{t}T$. We derive

$$
\begin{array}{ll}
 & T \ \mathbf{b}\ STOP(\mathbf{a}T) \\
= & \quad \{ \text{definition of blending} \} \\
 & (T \ \mathbf{w}\ STOP(\mathbf{a}T)) \upharpoonright \varnothing \\
= & \quad \{ \text{Property 1.3.4.4}, \mathbf{a}T \subseteq \mathbf{a}T \text{ and } \epsilon \in \mathbf{t}T \} \\
 & STOP(\mathbf{a}T) \upharpoonright \varnothing \\
= & \quad \{ STOP(\mathbf{a}T) \text{ is non-empty} \} \\
 & STOP
\end{array}
$$

(End of Proof)

From 1.4.2.0 we conclude that blending is not idempotent. The next example shows that blending is not associative.

Example 1.4.3 (blending is not associative)

$$
\begin{array}{ll}
 & (<\{a,b\},\{\epsilon,a,ab\}> \ \mathbf{b}\ <\{b,c\},\{\epsilon,b,bc\}>) \ \mathbf{b}\ <\{b,c\},\{\epsilon,b,bc\}> \\
= & \quad \{ \text{calculus} \} \\
 & <\{a,c\},\{\epsilon,a,ac\}> \ \mathbf{b}\ <\{b,c\},\{\epsilon,b,bc\}> \\
= & \quad \{ \text{calculus} \} \\
 & <\{a,b\},\{\epsilon,a,b,ab,ba\}> \\
\neq & \quad \{ \text{trace sets differ} \} \\
 & <\{a,b\},\{\epsilon,a,ab\}> \\
= & \quad \{ \text{Property 1.4.2.1} \} \\
 & <\{a,b\},\{\epsilon,a,ab\}> \ \mathbf{b}\ STOP \\
= & \quad \{ \text{Property 1.4.2.0} \} \\
 & <\{a,b\},\{\epsilon,a,ab\}> \ \mathbf{b}\ (\{b,c\},\{\epsilon,b,bc\}> \ \mathbf{b}\ <\{b,c\},\{\epsilon,b,bc\}>)
\end{array}
$$

(End of Example)

We do, however, have the following theorem.

Theorem 1.4.4

Under the restriction that each symbol occurs in at most two of the alphabets of the trace structures involved, blending is associative.

Proof

Let T, U, and V be trace structures, such that $\mathbf{a}T \cap \mathbf{a}U \cap \mathbf{a}V = \varnothing$.
From set theory we then have

$$(\mathbf{a}T \cup \mathbf{a}U) \cap \mathbf{a}V \subseteq \mathbf{a}T \div \mathbf{a}U \qquad (*)$$

We derive

$$\begin{array}{ll} & (T \mathbin{\mathbf{b}} U) \mathbin{\mathbf{b}} V \\ = & \quad \{ \text{definition of blending} \} \\ & ((T \mathbin{\mathbf{w}} U) \restriction (\mathbf{a}T \div \mathbf{a}U) \mathbin{\mathbf{w}} V) \restriction ((\mathbf{a}T \div \mathbf{a}U) \div \mathbf{a}V) \\ = & \quad \{ \text{Theorem 1.3.6, using } (*) \} \\ & ((T \mathbin{\mathbf{w}} U) \mathbin{\mathbf{w}} V) \restriction (((\mathbf{a}T \cup \mathbf{a}U) \cap (\mathbf{a}T \div \mathbf{a}U)) \cup \mathbf{a}V) \restriction ((\mathbf{a}T \div \mathbf{a}U) \div \mathbf{a}V) \\ = & \quad \{ \text{set calculus} \} \\ & ((T \mathbin{\mathbf{w}} U) \mathbin{\mathbf{w}} V) \restriction ((\mathbf{a}T \div \mathbf{a}U) \cup \mathbf{a}V) \restriction ((\mathbf{a}T \div \mathbf{a}U) \div \mathbf{a}V) \\ = & \quad \{ \text{Property 1.1.2.4, set calculus} \} \\ & ((T \mathbin{\mathbf{w}} U) \mathbin{\mathbf{w}} V) \restriction ((\mathbf{a}T \div \mathbf{a}U) \div \mathbf{a}V) \\ = & \quad \{ \text{associativity of weaving and of symmetric set difference} \} \\ & (T \mathbin{\mathbf{w}} U \mathbin{\mathbf{w}} V) \restriction (\mathbf{a}T \div \mathbf{a}U \div \mathbf{a}V) \end{array}$$

Since $\mathbf{w}$ as well as $\div$ are symmetric, we conclude

$$(T \mathbin{\mathbf{b}} U) \mathbin{\mathbf{b}} V = T \mathbin{\mathbf{b}} (U \mathbin{\mathbf{b}} V)$$

(End of Proof)

Let X be a finite set of trace structures such that each symbol of $(\cup\, T : T \in X : \mathbf{a}T)$ occurs in alphabets of at most two of the elements of X. Then the blend of the elements of X is well-defined. It is denoted by $(\mathbf{B}\, T : T \in X : T)$. From the proof of Theorem 1.4.4 we infer

$$(\mathbf{B}\, T : T \in X : T) = (\mathbf{W}\, T : T \in X : T) \restriction A$$

where A is the symmetric difference of the alphabets involved.
By definition we have $(\mathbf{B}\, T : T \in \varnothing : T) = \mathit{STOP}$, the unit element of blending.

Whenever we use the blending operation, we shall see to it that each symbol occurs in at most two of the alphabets of the trace structures involved.

From the properties of projection, i.e. 1.1.2.5 and 1.1.4.3, we have the following variant of Theorem 1.3.8 .

Theorem 1.4.5

Let T and U be trace structures. Then

0 $pref(T \mathbf{b} U) \subseteq pref(T)\mathbf{b}\, pref(U)$

1 if T and U are processes then $T \mathbf{b} U$ is a process

(End of Theorem)

Finally, we define a class of trace structures that may be viewed as the specification of a synchronization mechanism. It is a generalization of *SYNC* and *QSYNC* in [19].

Let A and B be alphabets and let p and q be natural numbers. The trace structure $SYNC_{p,q}(A,B)$ is defined as

$$<A \cup B, \{t \mid t \in (A \cup B)^* \wedge (\mathbf{A}\, s : s \leqslant t : -q \leqslant l(s \upharpoonright A) - l(s \upharpoonright B) \leqslant p)\}>$$

In any prefix of a trace of $SYNC_{p,q}(A,B)$ the lead of elements of A over elements of B is at most p, and the lead of elements of B over elements of A is at most q.

Property 1.4.6

0 $SYNC_{p,q}(A,B)$ is a process

1 $SYNC_{0,0}(A,B) = <A \cup B, (A \cap B)^*>$

2 $SYNC_{p,q}(A,B) = SYNC_{q,p}(B,A)$

3 $SYNC_{p,q}(\varnothing,\varnothing) = STOP$

(End of Property)

Note

When using these processes, we usually require that $p+q \geqslant 1$, and that A and B are non-empty and disjoint. However, putting these restrictions in the definitions leads to complicated formulations of properties and theorems.

(End of Note)

The following theorem is useful when calculating the blend of two *SYNC*'s.

Theorem 1.4.7

Let p, q, m, and n be natural numbers such that $p+q \geqslant 1$ and $m+n \geqslant 1$, and let A, B, C, and D be non-empty alphabets such that $A \cap B = \varnothing$, $C \cap D = \varnothing$, $A \cap C = \varnothing, B \cap D = \varnothing, A \cap D \neq \varnothing$, and $B \cap C \neq \varnothing$.

Then

$$
\begin{array}{ll}
 & SYNC_{p,q}(A,B) \ \mathbf{b} \ SYNC_{m,n}(C,D) \\
= & SYNC_{p+m,q+n}((A \cup C)\backslash(B \cup D), (B \cup D)\backslash(A \cup C))
\end{array}
$$

Proof

For the sake of convenience we abbreviate

$SYNC_{p,q}(A,B)$ to $S(A,B)$
$SYNC_{m,n}(C,D)$ to $S(C,D)$
$SYNC_{p+m,q+n}((A \cup C)\backslash(B \cup D), (B \cup D)\backslash(A \cup C))$ to $S(AC\backslash BD, BD\backslash AC)$

and

$A \cup B$ to AB
$A \cup C$ to AC
$C \cup D$ to CD
$B \cup D$ to BD

Due to the restrictions on the alphabets we have

$$
\begin{array}{ll}
 & AB \div CD \\
= & \quad \{\text{ definition of } \div \} \\
 & AB\backslash CD \ \cup \ CD\backslash AB \\
= & \quad \{ A \cap C = \varnothing, B \cap D = \varnothing \} \\
 & A\backslash D \ \cup \ B\backslash C \ \cup \ D\backslash A \ \cup \ C\backslash B \\
= & \quad \{ A \cap B = \varnothing, C \cap D = \varnothing \} \\
 & AC\backslash BD \ \cup \ BD\backslash AC \\
= & \quad \{\text{ definition of } \div \} \\
 & AC \div BD
\end{array}
$$

Hence,

$$AB \div CD \ = \ A\backslash D \ \cup \ B\backslash C \ \cup \ D\backslash A \ \cup \ C\backslash B \ = \ AC \div BD \qquad (*)$$

We derive

$$
\begin{array}{ll}
 & \mathbf{a}(S(A,B)\,\mathbf{b}\,S(C,D)) \\
= & \quad \{\text{ definitions of } SYNC \text{ and blending }\} \\
 & AB \div CD \\
= & \quad \{\,(*)\,\} \\
 & AC \div BD \\
= & \quad \{\text{ definition of } SYNC \text{ }\} \\
 & \mathbf{a}S(AC \setminus BD, BD \setminus AC)
\end{array}
$$

The equality of the trace sets is proved in two steps

(i) $\mathbf{t}(S(A,B)\,\mathbf{b}\,S(C,D)) \subseteq \mathbf{t}S(AC \setminus BD, BD \setminus AC)$

Let $t \in \mathbf{t}(S(A,B)\,\mathbf{b}\,S(C,D))$ and let $s \leqslant t$.
According to Theorem 1.4.5.1 we have $s \in \mathbf{t}(S(A,B)\,\mathbf{b}\,S(C,D))$ as well. Let w be such that $w \in \mathbf{t}(S(A,B)\,\mathbf{w}\,S(C,D))$ and $s = w\restriction(AB \div CD)$.
We derive

$$
\begin{array}{ll}
 & w \in \mathbf{t}(S(A,B)\,\mathbf{w}\,S(C,D)) \\
\Rightarrow & \quad \{\text{ definition of } SYNC \text{ and weaving }\} \\
 & -q \leqslant l(w\restriction A) - l(w\restriction B) \leqslant p \ \wedge\ -n \leqslant l(w\restriction C) - l(w\restriction D) \leqslant m \\
\Rightarrow & \quad \{\text{ calculus }\} \\
 & -(q+n) \leqslant l(w\restriction A) - l(w\restriction B) + l(w\restriction C) - l(w\restriction D) \leqslant p+m \\
= & \quad \{\, A \cap C = \varnothing, B \cap D = \varnothing \,\} \\
 & -(q+n) \leqslant l(w\restriction AC) - l(w\restriction BD) \leqslant p+m \\
= & \quad \{\text{ calculus }\} \\
 & -(q+n) \leqslant l(w\restriction AC \setminus BD) - l(w\restriction BD \setminus AC) \leqslant p+m \\
= & \quad \{\, s = w\restriction(AC \div BD), \text{ cf. } (*) \,\} \\
 & -(q+n) \leqslant l(s\restriction AC \setminus BD) - l(s\restriction BD \setminus AC) \leqslant p+m
\end{array}
$$

Hence,

$$(\mathbf{A}\, s : s \leqslant t : -(q+n) \leqslant l(s\restriction AC \setminus BD) - l(s\restriction BD \setminus AC) \leqslant p+m\,)$$

from which we conclude $t \in \mathbf{t}S(AC \setminus BD, BD \setminus AC)$

(ii) $\mathbf{t}S(AC \setminus BD, BD \setminus AC) \subseteq \mathbf{t}(S(A,B)\,\mathbf{b}\,S(C,D))$

In order to prove (ii) we have to show for each t in the set on the left-hand side, the existence of a trace w, $w \in \mathbf{t}(S(A,B)\,\mathbf{w}\,S(C,D))$, such that $t = w\restriction(AB \div CD)$. We do so by defining a function h.

$h: \ \mathbf{t}S(AC \setminus BD, BD \setminus AC) \rightarrow \mathbf{t}(S(A,B)\,\mathbf{w}\,S(C,D))$

with $h(t)\upharpoonright(AB \div CD) = t$.

We define h by induction on the length of t, which is possible since the domain of h is prefix-closed.

Base $t = \epsilon$

We have $\epsilon \in \mathbf{t}(S(A,B)\,\mathbf{w}\,S(C,D))$ and $\epsilon\upharpoonright(AB \div CD) = \epsilon$. Hence, we define $h(\epsilon) = \epsilon$.

Step $t = sa$ with $a \in AC \div BD$. Let $w = h(s)$.

Due to the symmetry of *SYNC*, cf. Property 1.4.6.2, the symmetry of the theorem to be proved, and (*), it suffices to treat the case $a \in A \setminus D$. We then have

$sa \in \mathbf{t}S(AC \setminus BD, BD \setminus AC) \ \wedge \ a \in A \setminus D$

and the induction hypothesis $(w = h(s))$:

$w \in \mathbf{t}(S(A,B)\,\mathbf{w}\,S(C,D)) \ \wedge \ w\upharpoonright(AB \div CD) = s$

Notice that the first conjunct of the induction hypothesis implies

$(\mathbf{A}\, v : v \leqslant w : -q \leqslant l(v\upharpoonright A) - l(v\upharpoonright B) \leqslant p \ \wedge \ -n \leqslant l(v\upharpoonright C) - l(v\upharpoonright D) \leqslant m)$

We derive

$$
\begin{array}{ll}
 & sa \in \mathbf{t}S(AC \setminus BD, BD \setminus AC) \\
\Rightarrow & \quad \{ \text{definition of } SYNC \} \\
 & l(sa\upharpoonright AC \setminus BD) - l(sa\upharpoonright BD \setminus AC) \leqslant p + m \\
= & \quad \{ a \in A \setminus D, A \cap C = \varnothing, A \cap B = \varnothing \} \\
 & l(s\upharpoonright AC \setminus BD) - l(s\upharpoonright BD \setminus AC) \leqslant p + m - 1 \\
= & \quad \{ \text{induction hypothesis: } s = w\upharpoonright(AC \div BD), \text{ cf. } (*) \} \\
 & l(w\upharpoonright AC \setminus BD) - l(w\upharpoonright BD \setminus AC) \leqslant p + m - 1 \\
= & \quad \{ \text{calculus} \} \\
 & l(w\upharpoonright AC) - l(w\upharpoonright BD) \leqslant p + m - 1 \\
= & \quad \{ A \cap C = \varnothing, B \cap D = \varnothing \} \\
 & l(w\upharpoonright A) + l(w\upharpoonright C) - l(w\upharpoonright B) - l(w\upharpoonright D) \leqslant p + m - 1 \\
\Rightarrow & \quad \{ \text{calculus} \} \\
 & l(w\upharpoonright A) - l(w\upharpoonright B) \leqslant p - 1 \ \vee \ l(w\upharpoonright C) - l(w\upharpoonright D) \leqslant m - 1 \\
= & \quad \{ w\upharpoonright AB \in \mathbf{t}S(A,B), \ w\upharpoonright CD \in \mathbf{t}S(C,D) \}
\end{array}
$$

$$
\begin{array}{cl}
 & -q \leqslant l(w{\restriction}A)-l(w{\restriction}B) \leqslant p-1 \\
 & \quad \vee \ (l(w{\restriction}A)-l(w{\restriction}B)=p \ \wedge \ -n \leqslant l(w{\restriction}C)-l(w{\restriction}D) \leqslant m-1) \\
= & \qquad \{ p+q \geqslant 1, \text{ hence } -q+1 \leqslant p \} \\
 & -q \leqslant l(w{\restriction}A)-l(w{\restriction}B) \leqslant p-1 \\
 & \quad \vee \ (-q+1 \leqslant l(w{\restriction}A)-l(w{\restriction}B)=p \ \wedge \ -n \leqslant l(w{\restriction}C)-l(w{\restriction}D) \leqslant m-1)
\end{array}
$$

Hence, we have two cases :

(0) $-q \leqslant l(w{\restriction}A)-l(w{\restriction}B) \leqslant p-1$

(1) $-q+1 \leqslant l(w{\restriction}A)-l(w{\restriction}B)=p \ \wedge \ -n \leqslant l(w{\restriction}C)-l(w{\restriction}D) \leqslant m-1$

In case (0) we define $h(sa)=wa$, since

$$
\begin{array}{cl}
 & -q \leqslant l(w{\restriction}A)-l(w{\restriction}B) \leqslant p-1 \\
= & \qquad \{ w{\restriction}CD \in \mathbf{t}S(C,D) \} \\
 & -q \leqslant l(w{\restriction}A)-l(w{\restriction}B) \leqslant p-1 \ \wedge \ -n \leqslant l(w{\restriction}C)-l(w{\restriction}D) \leqslant m \\
= & \qquad \{ a \in A \backslash D, A \cap B = \varnothing, A \cap C = \varnothing \} \\
 & -q+1 \leqslant l(wa{\restriction}A)-l(wa{\restriction}B) \leqslant p \ \wedge \ -n \leqslant l(wa{\restriction}C)-l(wa{\restriction}D) \leqslant m \\
\Rightarrow & \qquad \{ \text{induction hypothesis} \} \\
 & wa \in \mathbf{t}(S(A,B)\,\mathbf{w}\,S(C,D))
\end{array}
$$

and

$$
\begin{array}{cl}
 & wa{\restriction}(AB \div CD) \\
= & \qquad \{ a \in A \backslash D, A \cap C = \varnothing \} \\
 & (w{\restriction}(AB \div CD))a \\
= & \qquad \{ \text{induction hypothesis} \} \\
 & sa
\end{array}
$$

In case (1) we define $h(sa)=wba$, where $b \in B \cap C \quad (B \cap C \neq \varnothing)$, since

$$
\begin{array}{cl}
 & -q+1 \leqslant l(w{\restriction}A)-l(w{\restriction}B)=p \ \wedge \ -n \leqslant l(w{\restriction}C)-l(w{\restriction}D) \leqslant m-1 \\
= & \qquad \{ b \in B, B \cap A = \varnothing, B \cap D = \varnothing, b \in C \} \\
 & -q \leqslant l(wb{\restriction}A)-l(wb{\restriction}B)=p-1 \ \wedge \ -n+1 \leqslant l(wb{\restriction}C)-l(wb{\restriction}D) \leqslant m \\
= & \qquad \{ a \in A \backslash D, A \cap B = \varnothing, A \cap C = \varnothing \} \\
 & -q \leqslant l(wb{\restriction}A)-l(wb{\restriction}B)=p-1 \ \wedge \ -n+1 \leqslant l(wb{\restriction}C)-l(wb{\restriction}D) \leqslant m \\
 & \quad \wedge \ -q+1 \leqslant l(wba{\restriction}A)-l(wba{\restriction}B)=p \ \wedge \ -n+1 \leqslant l(wba{\restriction}C)-l(wba{\restriction}D) \leqslant m \\
\Rightarrow & \qquad \{ \text{induction hypothesis} \} \\
 & wba \in \mathbf{t}(S(A,B)\,\mathbf{w}\,S(C,D))
\end{array}
$$

and

$$\begin{array}{ll} & wba\upharpoonleft(AB \div CD) \\ = & \quad \{ b \in B \cap C \} \\ & wa\upharpoonleft(AB \div CD) \\ = & \quad \{ a \in A \setminus D,\ A \cap C = \varnothing \} \\ & (w\upharpoonleft(AB \div CD))a \\ = & \quad \{ \text{induction hypothesis} \} \\ & sa \end{array}$$

(End of Proof)

In the proof of Theorem 1.4.7, viz. in the Step, the fact $B \cap C \neq \varnothing$ is needed if $a \in A\setminus D \cup D\setminus A$ whereas $A \cap D \neq \varnothing$ is needed in the case $a \in B\setminus C \cup C\setminus B$. When $B = C$ the latter does not occur and we have

Theorem 1.4.8

For natural p, q, m, and n such that $p+q \geqslant 1$ and $m+n \geqslant 1$, and non-empty alphabets A, B, and C such that $A \cap B = \varnothing$ and $B \cap C = \varnothing$, we have

$$SYNC_{p,q}(A,B) \mathbf{\ b\ } SYNC_{m,n}(B,C) = SYNC_{p+m,q+n}(A\setminus C, C\setminus A)$$

(End of Theorem)

Corollary 1.4.9

For natural numbers p, q, m, and n such that $p+q \geqslant 1$ and $m+n \geqslant 1$, and mutually disjoint, non-empty alphabets A, B, and C we have

$$SYNC_{p,q}(A,B) \mathbf{\ b\ } SYNC_{m,n}(B,C) = SYNC_{p+m,q+n}(A,C)$$

(End of Corollary)

As a generalization of $SEM_1(a,b)$ we define $SEM_k(A,B)$ for $k \geqslant 0$, and alphabets A and B, by

$$SEM_k(A,B) = SYNC_{k,0}(A,B)$$

We write $SEM_k(a,b)$ and $SYNC_{p,q}(a,b)$ as shorthands for $SEM_k(\{a\},\{b\})$ and

$SYNC_{p,q}(\{a\},\{b\})$ respectively.

Property 1.4.10

0 $\quad SEM_k(A,B)$

$\quad = \langle A \cup B, \{t \mid t \in (A \cup B)^* \wedge (\mathbf{A}\, s : s \leqslant t : 0 \leqslant l(s\upharpoonright A) - l(s\upharpoonright B) \leqslant k)\}\rangle$

1 $\quad SEM_k$ is a process

2 $\quad SEM_0(A,B) = \langle A \cup B, (A \cap B)^* \rangle$

(End of Property)

Theorems 1.4.7 and 1.4.8, as well as Corollary 1.4.9, are easily reformulated for SEM's.

Example 1.4.11

$SYNC_{p,q}(a,b)\ \mathbf{b}\ SYNC_{m,n}(\{x,b\},\{y,a\}) = SYNC_{p+m,q+n}(x,y)$

$SEM_1(\{a0,a1\},c)\ \mathbf{b}\ SEM_2(c,\{a0,a2\}) = SEM_3(a1,a2)$

$SEM_k(a,b)\ \mathbf{b}\ SEM_m(b,c) = SEM_{k+m}(a,c)$

$SYNC_{1,1}(a,b)\ \mathbf{b}\ SEM_1(\{b,x\},\{a,y\}) = SYNC_{2,1}(x,y)$

$\quad SEM_1(a,b)\ \mathbf{b}\ SEM_1(a,c)$

$=\quad \{$ definition of SEM_k $\}$

$\quad SYNC_{1,0}(a,b)\ \mathbf{b}\ SYNC_{1,0}(a,c)$

$=\quad \{$ Property 1.4.6.2 $\}$

$\quad SYNC_{0,1}(b,a)\ \mathbf{b}\ SYNC_{1,0}(a,c)$

$=\quad \{$ Corollary 1.4.9 $\}$

$\quad SYNC_{1,1}(b,c)$

(End of Example)

Exercises

0. Compute:

   $RUN(A)\ \mathbf{b}\ RUN(B)$

   $RUN(A)\ \mathbf{b}\ STOP(B)$

   $STOP(A)\ \mathbf{b}\ STOP(B)$

1. Compute:

   $RUN(A)\ \mathbf{b}\ (\,RUN(A)\ \mathbf{b}\ RUN(A \cup B)\,)$

   $(\,RUN(A)\ \mathbf{b}\ RUN(A)\,)\ \mathbf{b}\ RUN(A \cup B)$

2. Prove: $\mathbf{a}T \cap \mathbf{a}U \subseteq A \;\Rightarrow\; (T\ \mathbf{b}\ U)\upharpoonright A \;=\; (T\upharpoonright A)\ \mathbf{b}\ (U\upharpoonright A)$

3. Prove: $t \in \mathbf{t}(T\ \mathbf{b}\ U) \;\Rightarrow\; t\upharpoonright(\mathbf{a}T \backslash \mathbf{a}U) \in \mathbf{t}T\upharpoonright(\mathbf{a}T \backslash \mathbf{a}U)$

4. Show that $<\{a,b\},\{t \mid t \in \{a,b\}^* \wedge 0 \leqslant l(t\upharpoonright a) - l(t\upharpoonright b) \leqslant 2\}>$ is *not* prefix-closed.

5. Prove: $SYNC_{p,q}(A,B) \;=\; RUN(A \cap B)\ \mathbf{b}\ SYNC_{p,q}(A \backslash B, B \backslash A)$

6. Compute:

   0. $SYNC_{1,1}(\{a,b\},\{c,d\})\ \mathbf{b}\ SYNC_{1,1}(\{d,e\},\{b,f\})$
   1. $SYNC_{1,1}(a,b)\ \mathbf{b}\ SYNC_{1,1}(c,b)$
   2. $SEM_1(\{a0,a1\},a2)\ \mathbf{b}\ SEM_1(\{a2,a3\},a0)$
   3. $SEM_1(\{a,x\},\{b,x\})\ \mathbf{b}\ SEM_1(\{b,y\},\{c,y\})$
   4. $SEM_1(x,y)\ \mathbf{b}\ SEM_1(\{x,a\},\{y,b\})$

7. For distinct symbols a and b we define $SEM(a,b)$ by

   $SEM(a,b) \;=\; <\{a,b\},\{t \mid t \in \{a,b\}^* \wedge (\mathbf{A}\, s : s \leqslant t : 0 \leqslant l(t\upharpoonright a) - l(t\upharpoonright b))\}>$

   Prove:

   0. $SEM(a,b)$ is a process
   1. $SEM(a,b)\ \mathbf{b}\ SEM_1(b,c) \;=\; SEM(a,c)$
   2. $SEM(a,b)\ \mathbf{b}\ SEM(b,c) \;=\; SEM(a,c)$

(End of Exercises)

1.5 States and state graphs

In this section we relate trace structures to labeled directed graphs.

Let T be a trace structure. The binary relation $\underset{T}{\sim}$ on $\mathbf{t}pref(T)$ is defined by

$$s \underset{T}{\sim} t \quad \equiv \quad (\mathbf{A}\, u : u \in \mathbf{a}T^* : su \in \mathbf{t}T \quad \equiv \quad tu \in \mathbf{t}T)$$

Property 1.5.0

0 $\underset{T}{\sim}$ is an equivalence relation:

$$s \underset{T}{\sim} s$$
$$s \underset{T}{\sim} t \quad \equiv \quad t \underset{T}{\sim} s$$
$$s \underset{T}{\sim} t \wedge t \underset{T}{\sim} u \quad \Rightarrow \quad s \underset{T}{\sim} u$$

1 $\underset{T}{\sim}$ is right congruent with respect to concatenation:

$$(\mathbf{A}\, s,t,u : su \in \mathbf{t}pref(T) \wedge tu \in \mathbf{t}pref(T) : s \underset{T}{\sim} t \quad \Rightarrow \quad su \underset{T}{\sim} tu)$$

(End of Property)

The equivalence classes corresponding to $\underset{T}{\sim}$ are called the *states* of T. $[t]_T$ denotes, as usual, the class to which t belongs.
Whenever T is obvious, we omit T in $\underset{T}{\sim}$ and $[t]_T$.

Example 1.5.1

$SEM_1(a, b)$ has two states, viz. $[\epsilon]$ and $[a]$.

(End of Example)

If $[s] = [t]$ and $sa \in \mathbf{t}pref(T)$, we have, due to the fact that $\sim$ is a right congruence, that $[sa] = [ta]$ as well. Hence, we have a relation R on the set of states, viz.

$$[s]\, R\, [t] \quad \equiv \quad (\mathbf{E}\, a : a \in \mathbf{a}T : [sa] = [t])$$

This relation can be represented by a directed labeled graph. The states of T are the nodes of the graph. If $[s]\, R\, [t]$ then there is an arc, labeled a, from $[s]$ to $[t]$ for each symbol $a \in \mathbf{a}T$ such that $[sa] = [t]$.

Example 1.5.2

Let $T = \langle \{a, b\}, \{a, ab, bb\} \rangle$
then $\mathbf{t}\mathit{pref}(T) = \{\epsilon, a, b, ab, bb\}$.

The states are $[\epsilon]$, $[a]$, $[b]$, and $[ab]$.
Notice that traces a and b are *not* equivalent, since $a\epsilon \in \mathbf{t}T$ and $b\epsilon \notin \mathbf{t}T$.
The state graph is shown in Figure 1.3 .

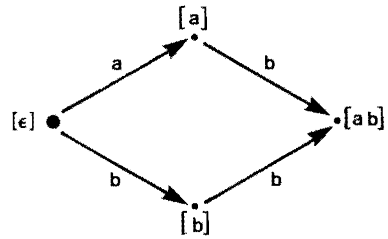


Figure 1.3

(End of Example)

If $\mathbf{t}T$ is empty then the graph is the empty graph. If $\mathbf{t}T$ is non-empty then $[\epsilon]$ is called the *initial state*. In figures of state graphs the initial state is drawn fat. Each path starting in $[\epsilon]$ yields an element of $\mathbf{t}\mathit{pref}(T)$ by recording the labels on that path. If such an element belongs to $\mathbf{t}T$, the endpoint of the path is called a *final state* (all states of a prefix-closed trace structure are final states).
The graph thus obtained is often called the minimal deterministic state graph of T. We call it *the* state graph of T.

Any directed graph with one node as initial node, zero or more nodes as final nodes, and with zero or more arcs labeled with symbols, defines a trace set: each path from the initial state to a final state yields a trace. Such a graph is called *nondeterministic* if there exists a node that has an unlabeled outgoing arc or two or more outgoing arcs with the same label. Otherwise it is called *deterministic*. If it is deterministic and if the number of nodes equals the number of states of the trace set it describes, it is called *minimal*. In a minimal state graph all arcs are labeled.

For a more formal treatment of the above we recommend [9]. A nice algorithm for the transformation of a nondeterministic state graph into a minimal deterministic one can be found in [19].

If T has a finite number of states, T is called a *regular* trace structure. The correspondence between regular trace structures and deterministic finite state machines is described in [19].

Example 1.5.3

Figure 1.4 shows the state graph of $\mathit{SYNC}_{1,1}(a, b)$.
There are three states, viz. $[\epsilon]$, $[a]$, and $[b]$.
Since $\mathit{SYNC}_{1,1}(a, b)$ is a process, every state is a final state.
$\mathit{SYNC}_{1,1}(a, b)$ is a regular process.

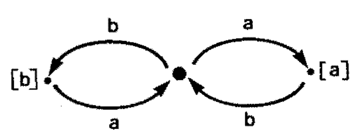


Figure 1.4

(End of Example)

Let B be a subset of $\mathbf{a}T$. A state graph of $T{\restriction}B$ is obtained from a state graph of T by removing all labels not in B. In general this leads to a nondeterministic state graph. Projection may, surprisingly, lead to a trace structure with more states than the original one. This is demonstrated in the next example.

Example 1.5.4

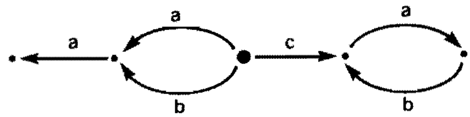


Figure 1.5

Process T is defined by

$\mathbf{a}T = \{a, b, c\}$

$\mathbf{t}T$ is the prefix-closed trace set described by the state graph shown in Figure 1.5 .

The trace set of $T{\restriction}\{a, b\}$ is given by Figure 1.6 .

The minimal deterministic state graph of $T{\restriction}\{a, b\}$ is shown in Figure 1.7 .

Apparently, T has 5 states and $T{\restriction}\{a, b\}$ has 6 states.

(End of Example)

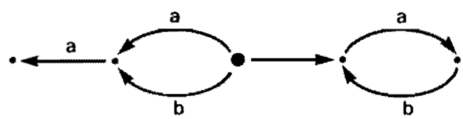


Figure 1.6

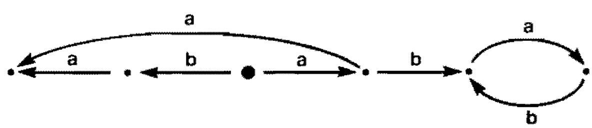


Figure 1.7

From automata theory it is known how a finite nondeterministic state graph can be transformed into a finite deterministic minimal state graph. As a consequence, we have

Property 1.5.5

If T is regular then $T{\restriction}B$ is regular.

(End of Property)

We now consider the relation between the state graph of trace structures T, U, and $T \mathbf{w} U$.

Property 1.5.6

Let s and t be traces of $\mathbf{t}pref(T\ \mathbf{w}\ U)$. Then

$$s\restriction\mathbf{a}T \underset{T}{\sim} t\restriction\mathbf{a}T \ \wedge\ s\restriction\mathbf{a}U \underset{U}{\sim} t\restriction\mathbf{a}U \ \Rightarrow\ s \underset{T\ \mathbf{w}\ U}{\sim} t$$

Proof

Assume $s\restriction\mathbf{a}T \underset{T}{\sim} t\restriction\mathbf{a}T \ \wedge\ s\restriction\mathbf{a}U \underset{U}{\sim} t\restriction\mathbf{a}U$. For any trace u we derive

$$\begin{aligned}
& su \in \mathbf{t}(T\ \mathbf{w}\ U) \\
= & \quad \{\text{ definition of weaving }\} \\
& su \in (\mathbf{a}T \cup \mathbf{a}U)^* \wedge su\restriction\mathbf{a}T \in \mathbf{t}T \wedge su\restriction\mathbf{a}U \in \mathbf{t}U \\
= & \quad \{\text{ property of projection }\} \\
& su \in (\mathbf{a}T \cup \mathbf{a}U)^* \wedge (s\restriction\mathbf{a}T)(u\restriction\mathbf{a}T) \in \mathbf{t}T \wedge (s\restriction\mathbf{a}U)(u\restriction\mathbf{a}U) \in \mathbf{t}U \\
= & \quad \{\ s\restriction\mathbf{a}T \underset{T}{\sim} t\restriction\mathbf{a}T \text{ and } s\restriction\mathbf{a}U \underset{U}{\sim} t\restriction\mathbf{a}U\ \} \\
& tu \in (\mathbf{a}T \cup \mathbf{a}U)^* \wedge (t\restriction\mathbf{a}T)(u\restriction\mathbf{a}T) \in \mathbf{t}T \wedge (t\restriction aU)(u\restriction\mathbf{a}U) \in \mathbf{t}U \\
= & \quad \{\text{ property of projection }\} \\
& tu \in (\mathbf{a}T \cup \mathbf{a}U)^* \wedge tu\restriction\mathbf{a}T \in \mathbf{t}T \wedge tu\restriction\mathbf{a}U \in \mathbf{t}U \\
= & \quad \{\text{ definition of weaving }\} \\
& tu \in \mathbf{t}(T\ \mathbf{w}\ U)
\end{aligned}$$

Hence, $s \underset{T\ \mathbf{w}\ U}{\sim} t$

(End of Proof)

Theorem 1.5.7

The number of states of $T\ \mathbf{w}\ U$ is at most the product of the number of states of T and the number of states of U.

Proof

For all s and t, $s \in \mathbf{t}pref(T\ \mathbf{w}\ U)$ and $t \in \mathbf{t}pref(T\ \mathbf{w}\ U)$, we derive

$$\begin{aligned}
& [s]_{T\ \mathbf{w}\ U} \neq [t]_{T\ \mathbf{w}\ U} \\
\Rightarrow & \quad \{\text{ Property 1.5.6 }\} \\
& [s\restriction\mathbf{a}T]_T \neq [t\restriction\mathbf{a}T]_T \ \vee\ [s\restriction\mathbf{a}U]_U \neq [t\restriction\mathbf{a}U]_U \\
= & \quad \{\text{ definition of equality of pairs }\} \\
& ([s\restriction\mathbf{a}T]_T, [s\restriction\mathbf{a}U]_U) \neq ([t\restriction\mathbf{a}T]_T, [t\restriction\mathbf{a}U]_U)
\end{aligned}$$

and the number of pairs (x, y) where x is a state of T and y is a state of U equals the product of the number of states of T and the number of states of U.

(End of Proof)

Corollary 1.5.8

If T and U are regular trace structures then $T \mathbf{w} U$ is a regular trace structure.

(End of Corollary)

Using Property 1.5.6 we can indeed construct a state graph of $T \mathbf{w} U$ from those of T and U:

Consider all pairs $([p],[q])$ where $[p]$ is a state of T and $[q]$ is a state of U. Take these pairs as nodes. We have an arc with label a from $([p0],[q0])$ to $([p1],[q1])$ in the following cases:

$$a \in \mathbf{a}T \cap \mathbf{a}U \ \wedge\ [p0a] = [p1] \ \wedge\ [q0a] = [q1]$$
$$a \in \mathbf{a}T \backslash \mathbf{a}U \ \wedge\ [p0a] = [p1] \ \wedge\ [q0] = [q1]$$
$$a \in \mathbf{a}U \backslash \mathbf{a}T \ \wedge\ [p0] = [p1] \ \wedge\ [q0a] = [q1]$$

The initial state is the pair of the initial states of T and U, and the final states are all pairs of final states of T and U.
In the resulting graph one may remove all nodes that are not reachable from the initial node, and all nodes from which no final node is reachable.

Example 1.5.9

The state graphs of $SEM_1(a, b)$ and $SEM_1(b, c)$ are shown in Figure 1.8 and Figure 1.9 respectively. Applying the method described above yields Figure 1.10, a state graph of $SEM_1(a, b) \mathbf{w} SEM_1(b, c)$.

Projection on $\{a, c\}$ yields Figure 1.11, the state graph of $SEM_2(a, c)$.

(End of Example)

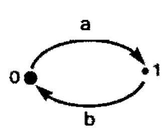


Figure 1.8

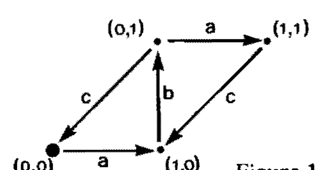


Figure 1.10

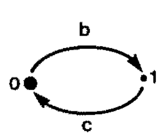


Figure 1.9

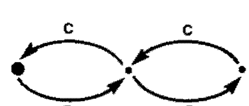


Figure 1.11

From Property 1.5.5 and Corollary 1.5.8 we infer

Theorem 1.5.10

If T and U are regular trace structures then $T \mathbf{b} U$ is a regular trace structure.

(End of Theorem)

Exercises

0. Show that $\sim$ is not left congruent with respect to concatenation.

1. Draw state graphs of the following processes:
$SEM_4(a,b)$, $SYNC_{1,3}(a,b)$, $RUN(\{a,b\})$, $STOP(\{a,b\})$.

2. Let T and U be trace structures. Describe the state graph of $<\mathbf{a}T \cup \mathbf{a}U, \mathbf{t}T \cup \mathbf{t}U>$ in terms of the state graphs of T and U.

3. Describe the state graph of
$<\mathbf{a}T \cup \mathbf{a}U, \{t \mid (\mathbf{E}\, u,v : u \in \mathbf{t}T \ \wedge \ v \in \mathbf{t}U : t = uv)\}>$
in terms of the state graphs of T and U.

4. Compute the number of states of $SYNC_{p,q}(A,B)$.

5. Let $T = <\{b\},\{b\}>$ and let $U = <\{b\},\{bb\}>$. Construct the state graph of $T \mathbf{w} U$ from those of T and U.

6. Process $SEM(a,b)$ is defined as
$<\{a,b\},\{t \mid t \in \{a,b\}^* \ \wedge \ (\mathbf{A}\, s : s \leq t : l(s\restriction a) \geq l(s\restriction b))\}>$
Prove that $SEM(a,b)$ is *not* regular.

7. Prove that for trace structures T and U such that $\mathbf{a}T \cap \mathbf{a}U = \varnothing$:

   the number of states of $T \mathbf{w} U$ equals the product of the number of states of T and the number of states of U.

8. Process T has alphabet $\{a, b, c, d\}$ and state graph as shown in Figure 1.12 . Determine the state graph of $T\upharpoonright\{a, b, c\}$.

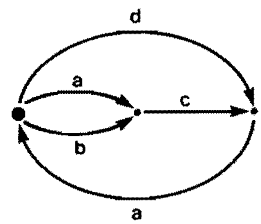


Figure 1.12

(End of Exercises)

1.6 The lattice $\mathcal{T}(A)$

In this section we study the structure of processes in more detail. First we review some concepts of lattice theory. For an introduction to lattice theory we recommend [0].

Let $(S, \leqslant)$ be a partially ordered set and let X be a subset of S. Element a is called the *greatest lower bound* of X if

$$(\mathbf{A}\, x : x \in X : a \leqslant x) \wedge (\mathbf{A}\, b : b \in S \wedge (\mathbf{A}\, x : x \in X : b \leqslant x) : b \leqslant a)$$

It is called the *least upper bound* of X if

$$(\mathbf{A}\, x : x \in X : a \geqslant x) \wedge (\mathbf{A}\, b : b \in S \wedge (\mathbf{A}\, x : x \in X : b \geqslant x) : b \geqslant a)$$

We call $(S, \leqslant)$ a *complete lattice* if each subset of S has a greatest lower bound and a least upper bound. The greatest lower bound and the least upper bound of elements x and y are denoted by x **glb** y and x **lub** y respectively. The greatest lower bound and the least upper bound of X are denoted by $(\mathbf{GLB}\, x : x \in X : x)$ and $(\mathbf{LUB}\, x : x \in X : x)$ respectively.

A complete lattice has a least element and a greatest element, viz. $(\mathbf{LUB}\, x : x \in \varnothing : x)$ and $(\mathbf{GLB}\, x : x \in \varnothing : x)$ respectively.

A sequence $x(i : i \geqslant 0)$ of elements of S is called an *ascending chain* if $(\mathbf{A}\, i : i \geqslant 0 : x(i) \leqslant x(i+1))$.
It is called a *descending chain* if $(\mathbf{A}\, i : i \geqslant 0 : x(i) \geqslant x(i+1))$.

Let $(S, \leqslant)$ and $(T, \leqslant)$ be complete lattices and let f be a function from S to T.

f is called

monotonic if $(\mathbf{A}\, x, y : x \in S \wedge y \in S : x \leqslant y \Rightarrow f(x) \leqslant f(y))$
disjunctive if $(\mathbf{A}\, x, y : x \in S \wedge y \in S : f(x \textbf{ lub } y) = f(x) \textbf{ lub } f(y))$
conjunctive if $(\mathbf{A}\, x, y : x \in S \wedge y \in S : f(x \textbf{ glb } y) = f(x) \textbf{ glb } f(y))$

universally disjunctive if

$$(\mathbf{A}\ X : X \subseteq S : f((\mathbf{LUB}\ x : x \in X : x)) = (\mathbf{LUB}\ x : x \in X : f(x)))$$

universally conjunctive if

$$(\mathbf{A}\ X : X \subseteq S : f((\mathbf{GLB}\ x : x \in X : x)) = (\mathbf{GLB}\ x : x \in X : f(x)))$$

universally disjunctive over non-empty sets if

$$(\mathbf{A}\ X : X \subseteq S \wedge X \neq \varnothing : f((\mathbf{LUB}\ x : x \in X : x)) = (\mathbf{LUB}\ x : x \in X : f(x)))$$

universally conjunctive over non-empty sets if

$$(\mathbf{A}\ X : X \subseteq S \wedge X \neq \varnothing : f((\mathbf{GLB}\ x : x \in X : x)) = (\mathbf{GLB}\ x : x \in X : f(x)))$$

upward continuous if for each ascending chain $x(i : i \geqslant 0)$

$$f((\mathbf{LUB}\ i : i \geqslant 0 : x(i))) = (\mathbf{LUB}\ i : i \geqslant 0 : f(x(i)))$$

downward continuous if for each descending chain $x(i : i \geqslant 0)$

$$f((\mathbf{GLB}\ i : i \geqslant 0 : x(i))) = (\mathbf{GLB}\ i : i \geqslant 0 : f(x(i)))$$

Some of these notions have been adopted from [4].

Example 1.6.0

Let A be a set, then $(P(A), \subseteq)$ is a complete lattice. For a subset Q of $P(A)$ we have

$$(\mathbf{LUB}\ X : X \in Q : X) = (\cup\ X : X \in Q : X)$$

$$(\mathbf{GLB}\ X : X \in Q : X) = (\cap\ X : X \in Q : X)$$

(Taking into account that $(\cup\ X : X \in \varnothing : X) = \varnothing$ and $(\cap\ X : X \in \varnothing : X) = A$)

Let B be a proper subset of A. Consider the function $f : P(A) \rightarrow P(A)$ defined by $f(X) = B \cap X$.
From $B \cap (X \cup Y) = (B \cap X) \cup (B \cap Y)$ we conclude that f is disjunctive.
From $B \cap (X \cap Y) = (B \cap X) \cap (B \cap Y)$ we conclude that f is conjunctive.

Since intersection distributes through any union of sets, f is universally disjunctive as well. Notice, however, that f is not universally conjunctive:

$$\begin{array}{ll} & f((\cap\ X : X \in \varnothing : X)) \\ = & \quad \{\text{definition of } f\} \\ & B \cap (\cap\ X : X \in \varnothing : X) \\ = & \quad \{\text{by definition}\} \\ & B \cap A \\ = & \quad \{B \text{ is a subset of } A\} \\ & B \\ \neq & \quad \{B \text{ is a proper subset of } A\} \end{array}$$

$$\begin{array}{ll} & A \\ = & \quad \{ \text{ by definition } \} \\ & (\cap\ X : X \in \varnothing : B \cap X) \\ = & \quad \{ \text{ definition of } f \ \} \\ & (\cap\ X : X \in \varnothing : f(X)) \end{array}$$

Let $X(i : i \geqslant 0)$ be an descending chain in $P(A)$. For any a, $a \in A$, we have

$$\begin{array}{ll} & a \in B \cap (\cap\ i : i \geqslant 0 : X(i)) \\ = & \quad \{ \text{ set calculus } \} \\ & a \in B \ \wedge\ (\mathbf{A}\, i : i \geqslant 0 : a \in X(i)) \\ = & \quad \{ \text{ predicate calculus } \} \\ & (\mathbf{A}\, i : i \geqslant 0 : a \in B \ \wedge\ a \in X(i)) \\ = & \quad \{ \text{ set calculus } \} \\ & a \in (\cap\ i : i \geqslant 0 : B \cap X(i)) \end{array}$$

from which we infer that f is downward continuous.

In this derivation the hint 'predicate calculus' can be refined to 'conjunction distributes through universal quantification over a non-empty range'. A similar derivation yields that f is universally conjunctive over non-empty sets.

(End of Example)

Without proof we mention the following properties.

Property 1.6.1

Both conjunctivity and disjunctivity imply monotonicity.
Universal conjunctivity over non-empty sets implies downward continuity.
Universal disjunctivity over non-empty sets implies upward continuity.
Both upward and downward continuity imply monotonicity.

(End of Property)

Property 1.6.2

Let S and T be complete lattices. Let f be a function from S to T. Let LS and LT denote the least elements of S and T respectively, and let GS and GT denote the greatest elements of S and T respectively. Then

$$\begin{aligned} & f \text{ is universally disjunctive} \\ \equiv\ & f \text{ is universally disjunctive over non-empty sets} \quad \wedge \quad f(LS) = LT \end{aligned}$$

and

$$\begin{aligned} & f \text{ is universally conjunctive} \\ \equiv\ & f \text{ is universally conjunctive over non-empty sets} \quad \wedge \quad f(GS) = GT \end{aligned}$$

(End of Property)

Let A be an alphabet.

The set of all processes with alphabet A is denoted by $\boldsymbol{T}(A)$.

In Section 1.2 we defined inclusion, intersection, and union for trace structures with equal alphabets, according to their trace sets.

Theorem 1.6.3

$(\boldsymbol{T}(A), \subseteq)$ is a complete lattice with least element $STOP(A)$ and greatest element $RUN(A)$.

Proof

For any non-empty prefix-closed subset X of A^* we have $\epsilon \in X$, hence, $STOP(A) \subseteq T$ for all $T, T \in \boldsymbol{T}(A)$. Moreover, $STOP(A)$ is a process, hence $STOP(A)$ is the least element of $\boldsymbol{T}(A)$.

For any $T, T \in \boldsymbol{T}(A)$, we have $\mathbf{t}T \subseteq A^*$. Since $RUN(A)$ is a process, $RUN(A)$ is the greatest element of $\boldsymbol{T}(A)$.

Let Q be a non-empty set of non-empty prefix-closed subsets of A^*. We have to prove that $(\cup\ X : X \in Q : X)$ and $(\cap\ X : X \in Q : X)$ are non-empty and prefix-closed.

From $Q \neq \varnothing$ and $(\mathbf{A}\ X : X \in Q : \epsilon \in X)$ we infer $\epsilon \in (\cap\ X : X \in Q : X)$ and $\epsilon \in (\cup\ X : X \in Q : X)$, so both are non-empty.

Let s and t be traces such that $s \leqslant t$. We derive

$$\begin{aligned} & t \in (\cup\ X : X \in Q : X) \\ = & \qquad \{ \text{ definition of union } \} \\ & (\mathbf{E}\ X : X \in Q : t \in X) \end{aligned}$$

$\Rightarrow$ { all elements of Q are prefix-closed, $s \leqslant t$ }
$(\mathbf{E}\, X : X \in Q : s \in X)$
$=$ { definition of union }
$s \in (\cup\, X : X \in Q : X)$

Hence, $(\cup\, X : X \in Q : X)$ is prefix-closed.
A similar derivation yields that $(\cap\, X : X \in Q : X)$ is prefix-closed.

(End of Proof)

Note

Since $STOP(A)$ is the least element of $\mathcal{T}(A)$, we have

$(\mathbf{LUB}\, X : X \in \varnothing : X) = STOP(A)$ (in the realm of $\mathcal{T}(A)$).

However, $(\cup\, X : X \in \varnothing : X) = \varnothing$, so we should be careful with the use of $\cup$ instead of **LUB** . A similar remark holds for $\cap$ and **GLB** . We do have:

$(\mathbf{LUB}\, X : X \in \varnothing : X) = STOP(A)$
$(\mathbf{LUB}\, X : X \in Q : X) = (\cup\, X : X \in Q : X)$ if $Q \neq \varnothing$

$(\mathbf{GLB}\, X : X \in \varnothing : X) = RUN(A)$
$(\mathbf{GLB}\, X : X \in Q : X) = (\cap\, X : X \in Q : X)$ if $Q \neq \varnothing$

(End of Note)

Let B be an alphabet. As we have seen in Section 1.2, the projection of trace structure T on B yields a trace structure with alphabet $\mathbf{a}T \cap B$:

$T \restriction B = <\mathbf{a}T \cap B, \mathbf{t}T \restriction B>$

From $pref(X \restriction B) = pref(X) \restriction B$ (Property 1.1.4.3) and $\epsilon \restriction B = \epsilon$ we conclude that $\restriction B$ maps processes onto processes:

$T \rightarrow T \restriction B$ maps $\mathcal{T}(A)$ onto $\mathcal{T}(A \cap B)$

Theorem 1.6.4 (projection is universally disjunctive)

Let B be an alphabet.
The mapping $\restriction B : \mathcal{T}(A) \rightarrow \mathcal{T}(A \cap B)$ is universally disjunctive.

Proof

$STOP(A) \restriction B = STOP(A \cap B)$, and for any non-empty set Q of trace sets we have

$$
\begin{array}{ll}
 & t \in (\cup\ X : X \in Q : X)\restriction B \\
= & \quad \{\text{ definition of projection }\} \\
 & (\mathbf{E}\ u : u \in (\cup\ X : X \in Q : X) : t = u\restriction B) \\
= & \quad \{\text{ definition of union }\} \\
 & (\mathbf{E}\ u : (\mathbf{E}\ X : X \in Q : u \in X) : t = u\restriction B) \\
= & \quad \{\text{ predicate calculus }\} \\
 & (\mathbf{E}\ X : X \in Q : (\mathbf{E}\ u : u \in X : t = u\restriction B)) \\
= & \quad \{\text{ definition of projection }\} \\
 & (\mathbf{E}\ X : X \in Q : t \in X\restriction B) \\
= & \quad \{\text{ definition of union }\} \\
 & t \in (\cup\ X : X \in Q : X\restriction B)
\end{array}
$$

Hence, for any subset Q of $\mathcal{T}(A)$ we have

$$(\mathbf{LUB}\ T : T \in Q : T)\restriction B \ = \ (\mathbf{LUB}\ T : T \in Q : T\restriction B)$$

(End of Proof)

Corollary 1.6.5

Projection is upward continuous and monotonic.

(End of Corollary)

Example 1.6.6 (projection is not downward continuous)

Let $A = \{a, b\}$ and let the descending chain $T(i : i \geqslant 0)$ be given by

$$T(i) = \langle A, \{t \mid (\mathbf{E}\ k : k \geqslant i : t \leqslant a^k b)\}\rangle$$

where $a^0 = \epsilon$ and $a^{k+1} = a^k a$ for $k \geqslant 0$.

Notice that for all i, $i \geqslant 0$, $T(i)$ is a process. We derive

$$
\begin{array}{ll}
 & t \in (\cap\ i : i \geqslant 0 : \mathbf{t}T(i)) \\
= & \quad \{\text{ calculus }\} \\
 & (\mathbf{A}\ i : i \geqslant 0 : (\mathbf{E}\ k : k \geqslant i : t \leqslant a^k b)) \\
= & \quad \{\text{ predicate calculus }\} \\
 & (\mathbf{A}\ i : i \geqslant l(t) : (\mathbf{E}\ k : k \geqslant i : t \leqslant a^k b)) \\
= & \quad \{\text{ calculus }\} \\
 & t \in \{a\}^*
\end{array}
$$

Hence, $(\mathbf{GLB}\ i : i \geqslant 0 : T(i)) = \langle\{a,b\},\{a\}^*\rangle$. Projection on $\{b\}$ yields

$$(\mathbf{GLB}\ i : i \geqslant 0 : T(i))\restriction\{b\} = \langle\{b\},\{\epsilon\}\rangle .$$

On the other hand we have $(\mathbf{A}\ i : i \geqslant 0 : T(i)\restriction\{b\} = \langle\{b\},\{\epsilon,b\}\rangle)$, hence

$$(\mathbf{GLB}\ i : i \geqslant 0 : T(i)\restriction\{b\}) = \langle\{b\},\{\epsilon,b\}\rangle .$$

(End of Example)

Let T be a process. Due to Theorem 1.3.8.1, $T \mathbf{w} V$ is a process for any process V.

Hence, we have a function $f : \boldsymbol{T}(A) \rightarrow \boldsymbol{T}(\mathbf{a}T \cup A)$ defined by $f(V) = T \mathbf{w} V$.

(f is the restriction to $\boldsymbol{T}(A)$ of the weave viewed as a function of its second argument). Since weaving is symmetric, all properties of f are also properties of the weave viewed as a function of its first argument. We simply call these properties 'properties of weaving'.

Theorem 1.6.7

0 Weaving is universally disjunctive over non-empty sets.

1 Weaving is universally conjunctive over non-empty sets.

Proof

0. Let Q be a non-empty subset of $\boldsymbol{T}(A)$. We derive

$$
\begin{array}{cl}
 & t \in \mathbf{t}(T \mathbf{w} (\cup\ V : V \in Q : V)) \\
= & \quad \{\text{ definition of weaving }\} \\
 & t \in (\mathbf{a}T \cup A)^* \wedge t\restriction\mathbf{a}T \in \mathbf{t}T \wedge t\restriction A \in \mathbf{t}(\cup\ V : V \in Q : V) \\
= & \quad \{\text{ definition of union }\} \\
 & t \in (\mathbf{a}T \cup A)^* \wedge t\restriction\mathbf{a}T \in \mathbf{t}T \wedge (\mathbf{E}\ V : V \in Q : t\restriction A \in \mathbf{t}V) \\
= & \quad \{\text{ predicate calculus }\} \\
 & (\mathbf{E}\ V : V \in Q : t \in (\mathbf{a}T \cup A)^* \wedge t\restriction\mathbf{a}T \in \mathbf{t}T \wedge t\restriction A \in \mathbf{t}V) \\
= & \quad \{\text{ definition of weaving }\} \\
 & (\mathbf{E}\ V : V \in Q : t \in \mathbf{t}(T \mathbf{w} V)) \\
= & \quad \{\text{ definition of union }\} \\
 & t \in \mathbf{t}(\cup\ V : V \in Q : T \mathbf{w} V)
\end{array}
$$

Hence, $T \mathbf{w} (\mathbf{LUB}\ V : V \in Q : V) = (\mathbf{LUB}\ V : V \in Q : T \mathbf{w} V)$

1. Similar

(End of Proof)

Note

Due to the non-emptiness of Q we are, in the derivation above, allowed to replace **LUB** by $\cup$.

(End of Note)

The next example shows that weaving is *not* universally disjunctive and *not* universally conjunctive.

Example 1.6.8

Let $A = \{a\}$ and $T = <\{a,b\},\{\epsilon,a,b\}>$ then

T **w** $STOP(a) = <\{a,b\},\{\epsilon,b\}> \neq STOP(\{a,b\})$

T **w** $RUN(a) = <\{a,b\},\{\epsilon,a,b\}> \neq RUN(\{a,b\})$

hence,

T **w** $(\mathbf{LUB}\ V : V \in \varnothing : V) \neq (\mathbf{LUB}\ V : V \in \varnothing : T\ \mathbf{w}\ V)$

T **w** $(\mathbf{GLB}\ V : V \in \varnothing : V) \neq (\mathbf{GLB}\ V : V \in \varnothing : T\ \mathbf{w}\ V)$

(End of Example)

The following corollary is a consequence of Theorem 1.6.7 and Property 1.6.1 .

Corollary 1.6.9

Weaving is conjunctive, disjunctive, and monotonic.
Weaving is upward continuous.
Weaving is downward continuous.

(End of Corollary)

Finally, we consider blending. Let T be a process. From Theorem 1.4.5.1 we conclude that T **b** V is a process for any process V.

Hence, $V \rightarrow T\ \mathbf{b}\ V$ is a mapping from $\mathcal{T}(A)$ to $\mathcal{T}(\mathbf{a}T \div A)$.

This mapping is the composite of $V \rightarrow T\ \mathbf{w}\ V$ and $U \rightarrow U\lceil(\mathbf{a}T \div A)$. Since the composite of two mappings inherits their common junctivity properties, we have on account of 1.6.4, 1.6.5, 1.6.7, and 1.6.9 the following theorem.

Theorem 1.6.10

> Blending is universally disjunctive over non-empty sets.
> Blending is upward continuous and monotonic.

(End of Theorem)

Example 1.6.11 (blending is *not* downward continuous)

Let $A = \{a, b\}$ and let $T = RUN(a)$.
The descending chain $V(i : i \geqslant 0)$ is defined by (cf. Example 1.6.6)
$V(i) = \langle A, \{t \mid (\mathbf{E}\, k : k \geqslant i : t \leqslant a^k b)\}\rangle$
Then

$$
\begin{array}{ll}
 & T\ \mathbf{b}\ (\mathbf{GLB}\, i : i \geqslant 0 : V(i)) \\
= & \quad \{\text{ Property 1.4.2.3, } T = RUN(a)\ \} \\
 & (\mathbf{GLB}\, i : i \geqslant 0 : V(i)) \upharpoonright b \\
= & \quad \{\text{ Example 1.6.6 }\} \\
 & \langle \{b\}, \{\epsilon\}\rangle \\
\neq & \quad \{\text{ trace sets differ }\} \\
 & \langle \{b\}, \{\epsilon, b\}\rangle \\
= & \quad \{\text{ Example 1.6.6 }\} \\
 & (\mathbf{GLB}\, i : i \geqslant 0 : V(i) \upharpoonright b) \\
= & \quad \{\text{ Property 1.4.2.3 }\} \\
 & (\mathbf{GLB}\, i : i \geqslant 0 : T\ \mathbf{b}\ V(i))
\end{array}
$$

(End of Example)

Let A and B be alphabets. The sequence $SEM_k(A, B)$, $k \geqslant 0$, is an ascending chain in $\mathcal{T}(A \cup B)$. Process $SEM(A, B)$ is defined by

$$SEM(A, B) = (\mathbf{LUB}\, k : k \geqslant 0 : SEM_k(A, B))$$

Property 1.6.12

$$SEM(A, B) = \langle A \cup B, \{t \mid t \in (A \cup B)^* \wedge (\mathbf{A}\, s : s \leqslant t : 0 \leqslant l(s \upharpoonright A) - l(s \upharpoonright B))\}\rangle$$

(End of Property)

Property 1.6.13

Let A, B, and C be mutually disjoint, non-empty alphabets. Then

0 $SEM_1(A, B)\,\mathbf{b}\,SEM(B, C) = SEM(A, C)$

1 $SEM(A, B)\,\mathbf{b}\,SEM(B, C) = SEM(A, C)$

Proof

0. We derive

$$\begin{aligned}
& SEM_1(A, B)\,\mathbf{b}\,SEM(B, C) \\
= & \quad \{ \text{definition of } SEM \} \\
& SEM_1(A, B)\,\mathbf{b}\,(\mathbf{LUB}\ k : k \geqslant 0 : SEM_k(B, C)) \\
= & \quad \{ \text{blending is upward continuous} \} \\
& (\mathbf{LUB}\ k : k \geqslant 0 : SEM_1(A, B)\,\mathbf{b}\,SEM_k(B, C)) \\
= & \quad \{ \text{Theorem 1.4.9} \} \\
& (\mathbf{LUB}\ k : k \geqslant 0 : SEM_{k+1}(A, C)) \\
= & \quad \{ SEM_0(A, C) \subseteq SEM_1(A, C) \} \\
& (\mathbf{LUB}\ k : k \geqslant 0 : SEM_k(A, C)) \\
= & \quad \{ \text{definition of } SEM \} \\
& SEM(A, C)
\end{aligned}$$

1. We derive

$$\begin{aligned}
& SEM(A, B)\,\mathbf{b}\,SEM(B, C) \\
= & \quad \{ \text{definition of } SEM \} \\
& SEM(A, B)\,\mathbf{b}\,(\mathbf{LUB}\ k : k \geqslant 0 : SEM_k(B, C)) \\
= & \quad \{ \text{blending is upward continuous} \} \\
& (\mathbf{LUB}\ k : k \geqslant 0 : SEM(A, B)\,\mathbf{b}\,SEM_k(B, C)) \\
= & \quad \{ \text{similar to the proof of part 0} \} \\
& (\mathbf{LUB}\ k : k \geqslant 0 : SEM(A, C)) \\
= & \quad \{ \text{definition of least upper bound} \} \\
& SEM(A, C)
\end{aligned}$$

(End of Proof)

Notice that $SEM(a, b)$ is not regular.
The states of $SEM(a, b)$ are $[a^k]$, $k \geqslant 0$.

We conclude this section with two theorems concerning lattice theory.

Theorem 1.6.14 (Knaster-Tarski)

Let $(S, \leqslant)$ be a complete lattice and let $f : S \to S$ be a *monotonic* function. Then the equation

$$x \in S : f(x) = x \qquad (0)$$

has a least solution, which is also the least solution of

$$x \in S : f(x) \leqslant x \qquad (1)$$

Proof

Notice that the greatest element of S is a solution of (1). Let $m, m \in S$, be defined by $m = (\mathbf{GLB}\ x : f(x) \leqslant x : x)$. We derive

$m = (\mathbf{GLB}\ x : f(x) \leqslant x : x)$

$\Rightarrow$ { definition of greatest lower bound }

$(\mathbf{A}\ x : f(x) \leqslant x : m \leqslant x)$

$\Rightarrow$ { f is monotonic }

$(\mathbf{A}\ x : f(x) \leqslant x : f(m) \leqslant f(x))$

$\Rightarrow$ { transitivity of $\leqslant$ }

$(\mathbf{A}\ x : f(x) \leqslant x : f(m) \leqslant x)$

$\Rightarrow$ { definition of greatest lower bound }

$f(m) \leqslant (\mathbf{GLB}\ x : f(x) \leqslant x : x)$

$=$ { definition of m }

$f(m) \leqslant m$

Hence, m is a solution of (1) and since $m = (\mathbf{GLB}\ x : f(x) \leqslant x : x)$, we have

m is the least solution of (1)

From

$f(m) \leqslant m$

$\Rightarrow$ { f is monotonic }

$f(f(m)) \leqslant f(m)$

$=$ { by definition }

$f(m)$ is a solution of (1)

we infer, since m is the least solution of (1), $m \leqslant f(m)$. Together with $f(m) \leqslant m$ this yields $f(m) = m$.
Hence, m is a solution of (0). Since each solution of (0) is a solution of (1), each solution

of (0) is at least m. We conclude

m is the least solution of (0)

(End of Proof)

Changing $\leqslant$ in $\geqslant$ and **GLB** in **LUB** yields a similar result for the greatest solutions of

$x \in S: \ x = f(x)$ and $x \in S: \ x \leqslant f(x)$

A solution of the equation $x \in S: \ x = f(x)$ is called a *fixpoint* of f. From 1.6.14 we conclude that each monotonic function from S to S has a least fixpoint and a greatest fixpoint.

Theorem 1.6.15

Let $(S, \leqslant)$ be a complete lattice. Let LS and GS denote the least and the greatest elements of S respectively. For a function $f: S \rightarrow S$ we have

0 if f is upward continuous then its least fixpoint equals
$(\mathbf{LUB}\ k : k \geqslant 0 : f^k(LS))$

1 if f is downward continuous then its greatest fixpoint equals
$(\mathbf{GLB}\ k : k \geqslant 0 : f^k(GS))$

Proof

0. Assume f is upward continuous. Then f is monotonic and since $LS \leqslant f(LS)$, $f^k(LS)$, $k \geqslant 0$, is an ascending chain. We derive

$f(\mathbf{LUB}\ k : k \geqslant 0 : f^k(LS))$
$=$ { f is upward continuous }
$(\mathbf{LUB}\ k : k \geqslant 0 : f^{k+1}(LS))$
$=$ { $LS \leqslant f(LS)$ }
$(\mathbf{LUB}\ k : k \geqslant 0 : f^k(LS))$

Hence, $(\mathbf{LUB}\ k : k \geqslant 0 : f^k(LS))$ is a fixpoint of f.

For each fixpoint x of f we have $LS \leqslant x$ and, hence, $f^k(LS) \leqslant f^k(x) = x$ for all k, $k \geqslant 0$. We conclude $(\mathbf{LUB}\ k : k \geqslant 0 : f^k(LS)) \leqslant x$ for each fixpoint x of f.
Hence $(\mathbf{LUB}\ k : k \geqslant 0 : f^k(LS))$ is the least fixpoint of f.

1. Similar

(End of Proof)

Exercises

0. Let $(S, \leqslant)$ be a complete lattice and let x and y be elements of S. Prove
   (i) x **glb** y and x **lub** y are unique.
   (ii) **glb** and **lub** are symmetric and associative.
   (iii) **glb** and **lub** have identity elements.

1. Let $(S, \leqslant)$ be a partially ordered set such that each subset of S has a least upper bound. Prove that each subset of S has a greatest lower bound as well.

2. Disprove
   (i) projection is conjunctive.
   (ii) blending is conjunctive.

3. Let A be an alphabet and let T be an element of $\boldsymbol{T}(A)$. The mappings f and g from $\boldsymbol{T}(A)$ to $\boldsymbol{T}(A)$ are defined by

   $f(V) = T \cup V$ and $g(V) = T \cap V$.

   Find out whether f and g are monotonic, disjunctive, conjunctive, upward continuous, downward continuous, universally disjunctive or universally conjunctive. Determine the fixpoints of f and g.

4. Compute $(\mathbf{LUB}\ i : i \geqslant 0 : SYNC_{i,k}(A, B))$ for fixed natural number k.

5. For natural k trace structure T_k is defined by

   $T_k = <\{a, b\}, \{t \mid (\mathbf{E}\ m, n : 0 \leqslant m \leqslant n \leqslant k : t = a^n b^m)\}>$

   Show that T_k is a process. Draw a state graph of T_3.
   Show that the sequence T_k, $k \geqslant 0$, is ascending. Show that $(\mathbf{LUB}\ k : k \geqslant 0 : T_k)$ is *not* regular.

6. T is a process and $V(i : i \geqslant 0)$ is a descending chain in $\boldsymbol{T}(A)$. Prove:

   $T\ \mathbf{b}\ (\mathbf{GLB}\ i : i \geqslant 0 : V(i)) \subseteq (\mathbf{GLB}\ i : i \geqslant 0 : T\ \mathbf{b}\ V(i))$

7. A is an alphabet. The set of *all* trace structures with alphabet A is denoted by $R(A)$. Prove that $R(A)$ is a complete lattice with least element $<A, \varnothing>$ and greatest element $RUN(A)$. Prove the analogs of the theorems of this section if $\boldsymbol{T}(A)$ is replaced by $R(A)$.

(End of Exercises)

2 A program notation

2.0 Introduction

In this chapter we present a program notation that defines a process. Such a program is also called a *component*. The first class of components we describe yields the set of all regular processes. It is closely related to the field of regular languages and regular expressions, cf. [9]. These components may be implemented as (sequential) finite state machines.

The second class of components still gives rise to regular trace structures, but may be implemented with more concurrency. This class allows components to be composed of - besides a regular expression - a number of subcomponents.

The third class includes recursive components. These can define non-regular processes.

2.1 Commands

From language theory it is known that a regular trace set can be represented by a regular expression. We extend the definition of regular expressions and define so-called *commands*.

Commands are defined inductively by the following six rules. With command S trace structure $TR(S)$ is associated.

(i) ϵ is a command. $TR(\epsilon) = STOP$

(ii) A symbol is a command. $TR(a) = \langle\{a\},\{a\}\rangle$

(iii) If S is a command then S^* is a command.

$TR(S^*) = \langle \mathbf{a}TR(S),(\mathbf{t}TR(S))^* \rangle$

where $(\mathbf{t}TR(S))^*$ denotes the set of finite length sequences of elements of $\mathbf{t}TR(S)$.

If S and T are commands then

(iv) $S \mid T$ is a command.

$TR(S \mid T) = \langle \mathbf{a}TR(S) \cup \mathbf{a}TR(T), \mathbf{t}TR(S) \cup \mathbf{t}TR(T) \rangle$

(v) $S;T$ is a command.

$TR(S;T) = \langle \mathbf{a}TR(S) \cup \mathbf{a}TR(T), \{t \mid (\mathbf{E}\, u,v : u \in \mathbf{t}TR(S) \wedge v \in \mathbf{t}TR(T) : t = uv)\} \rangle$

(vi) S, T is a command.

$$TR(S,T) = TR(S)\,\mathbf{w}\,TR(T)$$

From language theory it is known that the star, the bar, and the semi-colon preserve regularity. Corollary 1.5.8 yields that the comma preserves regularity as well. Since *STOP* and $<\{a\},\{a\}>$ are regular, we have

Property 2.1.0

A command defines a regular trace structure.

(End of Property)

To save parentheses we introduce the following priorities. The star has the highest priority. From the binary operators the comma has the highest priority, followed by the semi-colon and then the bar, i.e. the smaller the symbol the higher its priority.

Example 2.1.1

$$TR((a \mid b)^*) = RUN(a,b)$$
$$TR(a,(a;a)) = <\{a\},\varnothing>$$
$$TR((a;b)^*) = <\{a,b\},\{\epsilon, ab, abab, ababab, \cdots\}>$$

(End of Example)

We now present some algebraic properties of commands. These are expressed as equalities, where $S = T$ means $TR(S) = TR(T)$.

Property 2.1.2

The bar is symmetric, idempotent, and associative:

0 $\quad S0 \mid S1 = S1 \mid S0$

1 $\quad S \mid S = S$

2 $\quad S0 \mid (S1 \mid S2) = (S0 \mid S1) \mid S2$

The comma is symmetric, idempotent, and associative:

3 $\quad S0, S1 = S1, S0$

4 $\quad S, S = S$

5 $\quad S0, (S1, S2) = (S0, S1), S2$

The semicolon is associative:

6 $S0;(S1;S2) = (S0;S1);S2$

(End of Property)

Property 2.1.3

The semicolon distributes through the bar:

0 $S0;(S1 \mid S2) = (S0;S1 \mid S0;S2)$

1 $(S1 \mid S2);S0 = (S1;S0 \mid S2;S0)$

(End of Property)

Note

In some theories, cf. [16], there is a distinction between $a;(b \mid c)$ and $(a;b \mid a;c)$. This distinction arises from an operational point of view :

$a;(b \mid c)$ is interpreted as

'first event a occurs, after which both b and c are possible'

$(a;b \mid a;c)$ is interpreted as

'first event a occurs, after which either b or c is not possible any more'

We do not have this distinction. Both $TR(a;(b \mid c))$ and $TR((a;b \mid a;c))$ are equal to $<\{a,b,c\},\{ab,ac\}>$.
In Chapter 5 we discuss this topic in more detail.

(End of Note)

Property 2.1.4

0 $\epsilon , S = S$

1 $\epsilon ; S = S;\epsilon = S$

(End of Property)

Exercises

0. Draw state graphs of the trace structures defined by the following commands. (Indicate initial states and final states)
   (i) $(a;b)^*$
   (ii) $(a,b)^*$
   (iii) $(a;b)^*,(b;c)^*$
   (iv) $(a0;b0^* \mid a1;b1^*)^*$

1. Prove: $TR(S0,(S1 \mid S2)) \subseteq TR((S0,S1 \mid S0,S2))$
   Disprove: $TR((S0,S1 \mid S0,S2)) \subseteq TR(S0,(S1 \mid S2))$
   $TR((S0;S1),(S0;S2)) \subseteq TR(S0;(S1,S2))$
   $TR(S0;(S1,S2)) \subseteq TR((S0;S1),(S0;S2))$

(End of Exercises)

2.2 Components without subcomponents

The simplest form a component may have, is the following.

com $c(A)$: S **moc**

where c is the *name* of the component, A is a finite alphabet (usually represented by an enumeration of its elements), and S is a command.

With component c process $TR(c)$ is associated, defined by

$$TR(c) = pref(TR(S))$$

We impose the following restrictions on such a program text:

0 $\mathbf{a}TR(S) = A$

1 $\mathbf{t}TR(S) \neq \varnothing$

Due to the last restriction $TR(c)$ is non-empty, hence, $TR(c)$ is a process. From Property 2.1.0 we conclude

Property 2.2.0

A component without subcomponents defines a regular process.

(End of Property)

Example 2.2.1

0	**com** $stop()$: ϵ **moc**	$TR(stop) = STOP$
1	**com** $run(a,b)$: $(a \mid b)^*$ **moc**	$TR(run) = RUN(\{a,b\})$
2	**com** $sem_1(a,b)$: $(a;b)^*$ **moc**	$TR(sem_1) = SEM_1(a,b)$
3	**com** $sync_{1,1}(a,b)$: $(a,b)^*$ **moc**	$TR(sync_{1,1}) = SYNC_{1,1}(a,b)$

(End of Example)

Example 2.2.2

The process of Example 1.2.1, specifying a one-bit one-place buffer, equals $TR(buf_1)$ where buf_1 is defined by

com $buf_1(a0, a1, b0, b1)$: $(a0;b0 \mid a1;b1)^*$ **moc**

(End of Example)

Exercises

0. A binary variable is specified by

   com $var(a0, a1, b0, b1)$: $(a0;b0^* \mid a1;b1^*)^*$ **moc**

   where the following meaning is attached to the symbols.

   $a0$: the value zero is assigned

   $a1$: the value one is assigned

   $b0$: the value zero is inspected

   $b1$: the value one is inspected

   Draw a state graph of $TR(var)$ and interpret it states.

1. Define components for the processes $SEM_2(a,b)$, $STOP(\{a,b\})$, and $SYNC_{1,2}(a,b)$.

2. Give a component that has trace structure $SEM_2(\{a0, a1\}, \{b0, b1\})$.

3. A parity-counter is a mechanism that may be involved in the following events.

   a : a message is accepted

   e : the number of messages thus far accepted is even

o : the number of messages thus far accepted is odd

Give a formal specification of a parity-counter in terms of trace structures. Write a program according to that specification and draw a state graph of the process thus obtained.

4. A full adder is a component that repeatedly accepts three one-bit numbers and generates one two-bit number that equals the sum of the other three. Let the numbers to be added be a, b, and c, satisfying

$0 \leqslant a < 2 \ \wedge \ 0 \leqslant b < 2 \ \wedge \ 0 \leqslant c < 2$

and let the sum be represented by d and e such that

$0 \leqslant d < 2 \ \wedge \ 0 \leqslant e < 2 \ \wedge \ a + b + c = 2 \cdot d + e$

The values of a, b, c, d, and e are encoded by $a0$, $a1$, $b0$, $b1$, etc., where

$a0 \ \equiv \ a = 0, \ \ a1 \ \equiv \ a = 1,$ etc..

Derive a component that specifies the requirements stated above.

5. Figure 2.0 shows the state graph of a one-bit two-place buffer. Write a component buf_2 that specifies this buffer.

(End of Exercises)

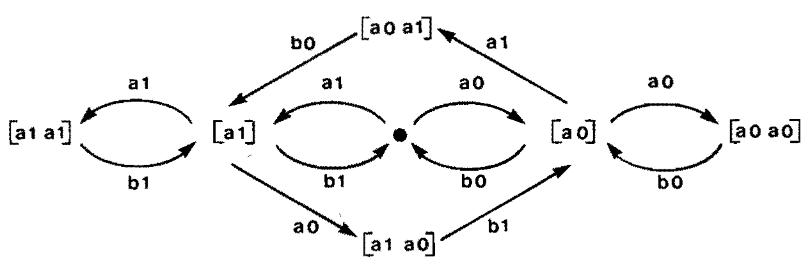


Figure 2.0

2.3 Subcomponents

Before introducing a more general form a component can have, we discuss some new notations.

Up to now we did not discuss the nature of Ω, the universe. As far as our examples are concerned, the set of all small letters and all strings of length two starting with a letter and ending with a digit would have been an appropriate universe. Taking, for example, the natural numbers as a universe, and representing its elements in the usual way, would cause ambiguity when using concatenation.
We tacitly assumed (and we will continue doing so) that the representation of the elements of Ω does not cause such ambiguities. Furthermore, we identify the elements of Ω with their representations.

Let A be an alphabet and let p be a symbol, then $p \cdot A$ denotes the set that is obtained by replacing each symbol a in A by $p \cdot a$. If X is a trace set then $p \cdot X$ denotes the set of sequences obtained by replacing in each trace of X each symbol a by $p \cdot a$. For trace structure T we define $p \cdot T$ by

$$p \cdot T \ = \ < p \cdot \mathbf{a}T, p \cdot \mathbf{t}T >$$

Example 2.3.0

Let $T \ = \ <\{a, b\}, \{\epsilon, a, ab, aba\}>$ then

$$p \cdot T \ = \ <\{p \cdot a, p \cdot b\}, \{\epsilon, p \cdot a, p \cdot a \ p \cdot b, p \cdot a \ p \cdot b \ p \cdot a\}>$$

(End of Example)

To avoid name clashes we require that no symbol in Ω contains a dot. The set $\Omega \cdot \Omega$ is defined by $\Omega \cdot \Omega = (\cup \ p : p \in \Omega : p \cdot \Omega)$. Elements of $\Omega \cdot \Omega$ are called *compound* symbols. Elements of Ω are called *simple* symbols. Due to our requirement $\Omega \cap \Omega \cdot \Omega \ = \ \emptyset$.

Our new universe is $\Omega \cup \Omega \cdot \Omega$. We shall see to it that the transformation of T into $p \cdot T$ is only applied if $\mathbf{a}T$ consists of simple symbols. The alphabets of components consist of simple symbols only. Compound symbols are used in program texts.

A more general form of a component is the following.

> **com** $c(A)$:
>   **sub** $p_0 : c_0, \ldots, p_{n-1} : c_{n-1}$ **bus**
> S
> **moc**

where $c_0, \ldots, c_{n-1}$ are previously defined components, called the *subcomponents* of c, with names $p_0, \ldots, p_{n-1}$ respectively. S is a command.

With subcomponent p_i process $p_i{\cdot}TR(c_i)$ is associated. We impose the following restrictions on such a program text:

0 The names p_i, $0 \leqslant i < n$, are distinct;

1 Alphabet A consists of simple symbols and

$\mathbf{a}TR(S) = A \cup (\cup\, i : 0 \leqslant i < n : p_i{\cdot}\mathbf{a}TR(c_i))$;

2 $TR(S)$ is non-empty.

From restrictions 0 and 1 we infer that each compound symbol occurs in exactly two alphabets of $p_0{\cdot}\mathbf{a}TR(c_0), \ldots, p_{n-1}{\cdot}\mathbf{a}TR(c_{n-1})$ and $\mathbf{a}TR(S)$.
Hence, blending of $p_0{\cdot}TR(c_0), \ldots, p_{n-1}{\cdot}TR(c_{n-1})$ and $pref(TR(S))$ is associative and yields a process with alphabet A.

The trace structure of component c is given by

$$TR(c) = (\mathbf{B}\, i : 0 \leqslant i < n : p_i{\cdot}TR(c_i))\ \mathbf{b}\ pref(TR(S))$$

Due to our syntactic restrictions $TR(c)$ is well-defined and (cf. Theorem 1.4.4) we have

$$TR(c) = ((\mathbf{W}\, i : 0 \leqslant i < n : p_i{\cdot}TR(c_i))\ \mathbf{w}\ pref(TR(S)))\restriction A$$

Because subcomponents have to be defined in advance, we call such a component a *non-recursive* component. Application of Property 2.1.0, Property 2.2.0, and Theorem 1.5.10, using induction over the syntax of components, yields

Property 2.3.1

A non-recursive component defines a regular process.

(End of Property)

Example 2.3.2

Component sem_1 is defined by **com** $sem_1(a,b)$: $(a;b)^*$ **moc**

Component sem_3 is defined by

com $sem_3(a,b)$:
  sub $p: sem_1$ **bus**
    $(a;p{\cdot}a)^*, (p{\cdot}b;b)^*$
moc

We derive

$$p{\cdot}SEM_1(a,b)\ \mathbf{b}\ pref(TR((a;p{\cdot}a)^*, (p{\cdot}b;b)^*))$$

$=$ { definition of comma }

$p{\cdot}SEM_1(a,b)$ **b** $pref(TR((a;p{\cdot}a)^*)$ **w** $TR((p{\cdot}b;b)^*))$

$=$ { Theorem 1.3.9, alphabets are disjoint }

$p{\cdot}SEM_1(a,b)$ **b** $(pref(TR(a;p{\cdot}a)^*)$ **w** $pref(TR(p{\cdot}b;b)^*))$

$=$ { definition of SEM_1 }

$p{\cdot}SEM_1(a,b)$ **b** $(SEM_1(a,p{\cdot}a)$ **w** $SEM_1(p{\cdot}b,b))$

$=$ { Property 1.4.0, alphabets are disjoint }

$p{\cdot}SEM_1(a,b)$ **b** $(SEM_1(a,p{\cdot}a)$ **b** $SEM_1(p{\cdot}b,b))$

$=$ { definition of $p{\cdot}$ }

$SEM_1(p{\cdot}a,p{\cdot}b)$ **b** $(SEM_1(a,p{\cdot}a)$ **b** $SEM_1(p{\cdot}b,b))$

$=$ { no symbol occurs in more than two alphabets }

$SEM_1(p{\cdot}a,p{\cdot}b)$ **b** $SEM_1(a,p{\cdot}a)$ **b** $SEM_1(p{\cdot}b,b)$

$=$ { Corollary 1.4.9 }

$SEM_3(a,b)$

Hence, $TR(sem_3) = SEM_3(a,b)$

(End of Example)

Example 2.3.3

Component sem_k, $k \geq 1$, with $TR(sem_k) = SEM_k(a,b)$ is defined inductively by

com $sem_1(a,b)$: $(a;b)^*$ **moc**, and for $k \geq 2$:

com $sem_k(a,b)$:
  sub $p : sem_{k-1}$ **bus**
    $((a \mid p{\cdot}b);(b \mid p{\cdot}a))^*$
moc

since

$SEM_{k-1}(p{\cdot}a,p{\cdot}b)$ **b** $pref(TR((a \mid p{\cdot}b);(b \mid p{\cdot}a))^*)$

$=$ { definition of SEM_1 }

$SEM_{k-1}(p{\cdot}a,p{\cdot}b)$ **b** $SEM_1(\{a,p{\cdot}b\},\{b,p{\cdot}a\})$

$=$ { Theorem 1.4.7 }

$SEM_k(a,b)$

(End of Example)

The last extension of our program notation is the following.

com $c(A)$:
  sub $p_0 : c_0, \ldots, p_{n-1} : c_{n-1}$ **bus**
  $[x_0 = y_0, \ldots, x_{m-1} = y_{m-1}]$
    S
moc

The equalities represent relations (connections);
x_0 through x_{m-1} are compound symbols and y_0 through y_{m-1} are simple or compound symbols. We are interested in the same blend as before, viz.

$$(\mathbf{B}\, i : 0 \leqslant i < n : p_i \cdot TR(c_i)) \ \mathbf{b} \ pref(TR(S))$$

but before computing this blend we carry out a substitution according to the equalities.

Each symbol at the left hand side of an equality is replaced by the symbol to which it is equated, both in the alphabet and in the trace set of the trace structure to which it belongs. After having carried out this substitution the blend is computed, i.e.

$$(\mathbf{B}\, i : 0 \leqslant i < n : (p_i \cdot TR(c_i))^{x_0, \ldots, x_{m-1}}_{y_0, \ldots, y_{m-1}}) \ \mathbf{b} \ pref(TR(S))$$

We impose the following restrictions.

0 The names p_i, $0 \leqslant i < n$, are distinct;

1 for all j, $0 \leqslant j < m$,
  x_j is an element of $(\cup\, i : 0 \leqslant i < n : p_i \cdot \mathbf{a}TR(c_i))$,
  y_j is an element of $(\cup\, i : 0 \leqslant i < n : p_i \cdot \mathbf{a}TR(c_i)) \cup A$,
  x_j and y_j belong to two different (of the $n+1$) alphabets,
  each symbol of $(\cup\, i : 0 \leqslant i < n : p_i \cdot \mathbf{a}TR(c_i)) \cup A$ occurs in at most one equality;

2 alphabet A consists of simple symbols and
$\mathbf{a}TR(S) = ((\cup\, i : 0 \leqslant i < n : p_i \cdot \mathbf{a}TR(c_i)) \cup A) \backslash (\cup\, j : 0 \leqslant j < m : \{x_j, y_j\})$;

3 $TR(S)$ is non-empty.

Due to these restrictions we have (after having carried out the substitution):

a compound symbol occurs in zero alphabets since it has been replaced *or*

a compound symbol occurs in two subcomponent-alphabets and not in $\mathbf{a}TR(S)$, since it occurred at the right hand side of an equality *or*

a compound symbol does not occur in any equality and then occurs in the alphabet of its subcomponent and in $\mathbf{a}TR(S)$.

Hence (Theorem 1.4.4), associativity of the blending operator is guaranteed, and

$$TR(c) \;=\; ((\mathbf{W}\; i : 0 \leqslant i < n : (p_i{\cdot}TR(c_i))^{x_0,\ldots,x_{m-1}}_{y_0,\ldots,y_{m-1}}) \;\mathbf{w}\; pref(TR(S))) \upharpoonright A$$

Example 2.3.4

com $sem_2(a, b)$:
  sub $p : sem_1$ **bus**
  $[\, p{\cdot}b = b \,]$
    $(a ; p{\cdot}a)^*$
moc

We derive

$(SEM_1(p{\cdot}a, p{\cdot}b))^{p{\cdot}b}_{b}$ **b** $pref(TR((a ; p{\cdot}a)^*))$
= { substitution }
$SEM_1(p{\cdot}a, b)$ **b** $pref(TR((a ; p{\cdot}a)^*))$
= { definition of SEM_1 }
$SEM_1(p{\cdot}a, b)$ **b** $SEM_1(a, p{\cdot}a)$
= { Corollary 1.4.9 }
$SEM_2(a, b)$

Hence, $TR(sem_2) = SEM_2(a, b)$

(End of Example)

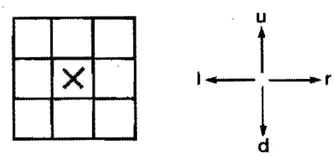


Figure 2.1

Example 2.3.5

See Figure 2.1 .

A pebble is placed in the middle of a 3×3 checker board. It may move up, down, left, and right but it is not allowed to leave the board. We derive a component that describes the behaviour of the pebble.

Possible events are u, d, l, and r meaning up, down, left, and right respectively. From the initial state a lead of two or more r's over l's violates the restriction on the pebble. A lead of one does not harm. For reasons of symmetry the same holds for leads of l over r, u over d, and d over u. This yields

$-1 \leqslant l(t \upharpoonright r) - l(t \upharpoonright l) \leqslant 1$ and
$-1 \leqslant l(t \upharpoonright u) - l(t \upharpoonright d) \leqslant 1$

for each t, $t \in \{u, d, l, r\}^*$, that describes a pattern of the pebble.

Since this should hold for all prefixes of these traces as well, we have

$$T = <\{u,d,l,r\}, \{t \mid t \in \{u,d,l,r\}^* \wedge (\mathbf{A}\, s : s \leqslant t : -1 \leqslant l(s\lceil r) - l(s\lceil l) \leqslant 1)$$
$$\wedge\ (\mathbf{A}\, s : s \leqslant t : -1 \leqslant l(s\lceil u) - l(s\lceil d) \leqslant 1)\}>$$

is a specification of the pebble.

Evidently, $T = SYNC_{1,1}(r,l)\ \mathbf{w}\ SYNC_{1,1}(u,d)$.

Since the alphabets are disjoint, this weave equals the blend. A component with process T is given by

com $pebble(u,d,l,r)$:
  sub $p,q : sync_{1,1}$ **bus**
  $[\,p{\cdot}a = u,\ p{\cdot}b = d,\ q{\cdot}a = l,\ q{\cdot}b = r\,]$
    ϵ
moc

(For $sync_{1,1}$ we refer to Example 2.2.1 .)

The text $p,q : sync_{1,1}$ is short for $p : sync_{1,1},\ q : sync_{1,1}$.

(End of Example)

Example 2.3.6

Component sem_k, $k \geqslant 1$, with $TR(sem_k) = SEM_k(a,b)$ can be defined inductively by

com $sem_1(a,b)$: $(a;b)^*$ **moc** , and for $k \geqslant 2$:

com $sem_k(a,b)$:
  sub $p : sem_1,\ q : sem_{k-1}$ **bus**
  $[\,p{\cdot}a = a,\ p{\cdot}b = q{\cdot}a,\ q{\cdot}b = b\,]$
    ϵ
moc

since

$SEM_1(p{\cdot}a,p{\cdot}b)^{p\cdot a,\,p\cdot b}_{a,\ q\cdot a}\ \mathbf{b}\ SEM_{k-1}(q{\cdot}a,q{\cdot}b)^{q\cdot b}_{b}\ \mathbf{b}\ STOP$

$=$ { substitution, $STOP$ is the unit element of blending }

$SEM_1(a,q{\cdot}a)\ \mathbf{b}\ SEM_{k-1}(q{\cdot}a,b)$

$=$ { Corollary 1.4.9 }

$SEM_k(a,b)$

(End of Example)

Example 2.3.7

In the previous example we presented component sem_k. Component $csem_k$ is defined by

> **com** $csem_k(a0, a1, b0, b1)$:
> **sub** $p : sem_k$ **bus**
> $(p{\cdot}a;(a0 \mid a1) \mid p{\cdot}b;(b0 \mid b1))^*$
> **moc**

Let S denote the command of $csem_k$. We derive

$$
\begin{array}{ll}
 & TR(csem_k) \\
= & \quad \{\text{ definition of a component }\} \\
 & (SEM_k(p{\cdot}a, p{\cdot}b) \ \mathbf{w} \ pref(TR(S)))\restriction\{a0, a1, b0, b1\} \\
= & \quad \{\text{ structure of } S\text{, compound symbols are removed by } \restriction\{a0, a1, b0, b1\}\ \} \\
 & (SEM_k(p{\cdot}a, p{\cdot}b) \ \mathbf{w} \ TR(S))\restriction\{a0, a1, b0, b1\}
\end{array}
$$

and for any trace t

$$
\begin{array}{ll}
 & t \in \mathbf{t}(SEM_k(p{\cdot}a, p{\cdot}b) \ \mathbf{w} \ TR(S)) \\
\Rightarrow & \quad \{\text{ definition of weaving and of } SEM_k\ \} \\
 & 0 \leq l(t\restriction p{\cdot}a) - l(t\restriction p{\cdot}b) \leq k \ \wedge \ t \in \mathbf{t}TR(S) \\
\Rightarrow & \quad \{\text{ structure of } S\ \} \\
 & 0 \leq l(t\restriction p{\cdot}a) - l(t\restriction p{\cdot}b) \leq k \ \wedge \ l(t\restriction p{\cdot}a) = l(t\restriction\{a0, a1\}) \ \wedge \ l(t\restriction p{\cdot}b) = l(t\restriction\{b0, b1\}) \\
\Rightarrow & \quad \{\text{ calculus }\} \\
 & 0 \leq l(t\restriction\{a0, a1\}) - l(t\restriction\{b0, b1\}) \leq k
\end{array}
$$

Hence, $TR(csem_k) \subseteq SEM_k(\{a0, a1\}, \{b0, b1\})$.

We prove $SEM_k(\{a0, a1\}, \{b0, b1\}) \subseteq TR(csem_k)$ by constructing a function

$$h : \mathbf{t}SEM_k(\{a0, a1\}, \{b0, b1\}) \rightarrow \mathbf{t}(SEM_k(p{\cdot}a, p{\cdot}b) \ \mathbf{w} \ TR(S))$$

such that $h(t)\restriction\{a0, a1, b0, b1\} = t$.

h is defined inductively by:

$$
\begin{array}{lll}
h(\epsilon) = \epsilon & h(ta0) = h(t)\, p{\cdot}a\ a0 & h(ta1) = h(t)\, p{\cdot}a\ a1 \\
 & h(tb0) = h(t)\, p{\cdot}b\ b0 & h(tb1) = h(t)\, p{\cdot}b\ b1
\end{array}
$$

Then, evidently, $h(t) \in \mathbf{t}(SEM_k(p{\cdot}a, p{\cdot}b) \ \mathbf{w} \ TR(S))$ and $h(t)\restriction\{a0, a1, b0, b1\} = t$.

We conclude $TR(csem_k) = SEM_k(\{a0, a1\}, \{b0, b1\})$.

(End of Example)

A component can always be transformed into a component with equalities only (i.e. with command ϵ) by adding a subcomponent of the type described in Section 2.2 . Hence, the components described in this section could also have been introduced without the command S. The transformation is as follows.

Let c be the component defined by

com $c(A)$:
  sub $p_0 : c_0, \ldots, p_{n-1} : c_{n-1}$ **bus**
  $[x_0 = y_0, \ldots, x_{m-1} = y_{m-1}]$
   S
moc

Define a one-to-one function $\phi : \mathbf{a}TR(S) \to \Omega$, a renaming function used to get rid of compound symbols. Define component c_n by

com $c_n(\phi(\mathbf{a}TR(S)))$: $\phi(S)$ **moc**

where $\phi(S)$ is obtained from S by changing each symbol a in S into $\phi(a)$.
Then $TR(c_n) = pref(TR(\phi(S)))$.

Component d is defined by

com $d(A)$:
  sub $p_0 : c_0, \ldots, p_{n-1} : c_{n-1}, p_n : c_n$ **bus**
  $[x_0 = y_0, \ldots, x_{m-1} = y_{m-1}, p_n \cdot \phi(z_0) = z_0, \ldots, p_n \cdot \phi(z_{k-1}) = z_{k-1}]$
   ϵ
moc

where $\{z_0, \ldots, z_{k-1}\} = \mathbf{a}TR(S)$.
Notice that d satisfies the restrictions imposed on program texts.

We then have

$TR(d)$
$=$ { definition of a component }
$(\mathbf{B}\, i : 0 \leqslant i < n : (p_i \cdot TR(c_i))^{x_0, \ldots, x_{m-1}}_{y_0, \ldots, y_{m-1}})\ \mathbf{b}\ (p_n \cdot TR(c_n))^{p_n \cdot \phi(z_0), \ldots, p_n \cdot \phi(z_{k-1})}_{z_0, \ldots, z_{k-1}}$
$=$ { substitution }
$(\mathbf{B}\, i : 0 \leqslant i < n : (p_i \cdot TR(c_i))^{x_0, \ldots, x_{m-1}}_{y_0, \ldots, y_{m-1}})\ \mathbf{b}\ pref(TR(S))$
$=$ { definition of c }
$TR(c)$

Example 2.3.8

We transform component sem_2 of Example 2.3.4 :

> **com** $sem_2(a, b)$:
> **sub** $p : sem_1$ **bus**
> $[\, p \cdot b = b \,]$
> $(a ; p \cdot a)^*$
> **moc**

Function ϕ is defined by $\phi(a) = x$ and $\phi(p \cdot a) = y$, and component c_1 is defined by

> **com** $c_1(x, y)$: $(x ; y)^*$ **moc**

The transformation yields

> **com** $d(a, b)$:
> **sub** $p : sem_1, q : c_1$ **bus**
> $[\, p \cdot b = b, q \cdot x = a, q \cdot y = p \cdot a \,]$
> ϵ
> **moc**

And, indeed (cf. Example 2.3.6), we have $TR(d) = SEM_2(a, b)$.

(End of Example)

We may, on the other hand, transform a component with equalities only (i.e. with command ϵ) into a component without equalities.
Let component c be defined by

> **com** $c(A)$:
> **sub** $p_0 : c_0, \ldots, p_{n-1} : c_{n-1}$ **bus**
> $[\, x_0 = y_0, \ldots, x_{m-1} = y_{m-1}]$
> ϵ
> **moc**

Due to our restrictions, each symbol of $A \cup (\cup\, i : 0 \leqslant i < n : p_i \cdot \mathbf{a}TR(c_i))$ occurs exactly once in the equalities. Moreover, x_0 through x_{m-1} are compound symbols.

Component d is defined by

com $d(A)$:
  sub $p_0 : c_0, \ldots, p_{n-1} : c_{n-1}$ **bus**
   $(x_0; y_0 \mid \cdots \mid x_{m-1}; y_{m-1})^*$
moc

Let S denote the command of d. Define T and U by

$$T = (\mathbf{W}\, i : 0 \leqslant i < n : (p_i \cdot TR(c_i))_{y_0, \ldots, y_{m-1}}^{x_0, \ldots, x_{m-1}})$$

$$U = (\mathbf{W}\, i : 0 \leqslant i < n : p_i \cdot TR(c_i))\ \mathbf{w}\ TR(S)$$

Then $TR(c) = T \restriction A$. Since each trace of $\mathbf{t}pref(TR(S)) \backslash \mathbf{t}TR(S)$ is the concatenation of a trace of $\mathbf{t}TR(S)$ and a compound symbol (x_0 through x_{m-1} are compound symbols), we have $TR(d) = U \restriction A$.

Let $f : \mathbf{t}T \rightarrow \mathbf{t}U$ be defined by

$f(\epsilon) = \epsilon$

$f(ty_k) = f(t)x_k y_k \quad (0 \leqslant k < m)$

Then $f(t) \restriction A = t \restriction A$. Furthermore, f has inverse g defined by

$g(\epsilon) = \epsilon$

$g(tx_k y_k) = g(t) y_k \quad (0 \leqslant k < m)$

We conclude $\mathbf{t}T \restriction A = f(\mathbf{t}T) \restriction A = \mathbf{t}U \restriction A$, and, hence, $TR(c) = TR(d)$.

Example 2.3.9

Component sem_2 (cf. Example 2.3.6) is defined by

com $sem_2(a, b)$:
  sub $p, q : sem_1$ **bus**
  $[p \cdot a = a, p \cdot b = q \cdot a, q \cdot b = b]$
   ϵ
moc

The transformation as described above yields

com $d(a, b)$:
  sub $p, q : sem_1$ **bus**
   $(p \cdot a ; a \mid p \cdot b ; q \cdot a \mid q \cdot b ; b)^*$
moc

(End of Example)

Exercises

0. Prove that the pebble of Example 2.3.5 is also specified by

   com $peb(u,d,l,r)$: $(u,d)^*,(l,r)^*$ **moc**

   Draw a state graph of $TR(peb)$.

1. Define inductively for $k \geqslant 1$ and $l \geqslant 1$, component $sync_{k,l}$ such that

   $TR(sync_{k,l}) = SYNC_{k,l}(a,b)$

2. Let sem_1 be defined by **com** $sem_1(a,b)$: $(a;b)^*$ **moc**.

   For i, $i \geqslant 1$, sem_{2i+1} and sem_{2i} are defined by

   com $sem_{2i+1}(a,b)$:
     sub $p : sem_i$ **bus**
     $((a \mid p{\cdot}b;b);(p{\cdot}a;a \mid b))^*$
   moc

   com $sem_{2i}(a,b)$:
     sub $p : sem_{2i-1}$ **bus**
     $[p{\cdot}b = b]$
     $(a;p{\cdot}a)^*$
   moc

   Prove $TR(sem_i) = SEM_i(a,b)$ for all i, $i \geqslant 1$.

3. Component sem_2 has $TR(sem_2) = SEM_2(a,b)$. Compute the processes of the following components.

   (i) **com** $c(a,b)$:
     sub $p,q : sem_2$ **bus**
     $[p{\cdot}a = a,\ p{\cdot}b = q{\cdot}a,\ q{\cdot}b = b]$
     ϵ
   moc

   (ii) **com** $d(a,b)$:
     sub $p,q : sem_2$ **bus**
     $(a;p{\cdot}a)^*,(p{\cdot}b;q{\cdot}a)^*,(q{\cdot}b;b)^*$
   moc

(End of Exercises)

2.4 Recursive components

In this section we drop the rule that components should have been defined before they are used as subcomponents.

We say that component d *occurs* in component c if d is a subcomponent of c or if d occurs in a subcomponent of c.

Component c is called *recursive* if c occurs in c. In the sequel we consider component c defined by

com $c(A)$:
  sub $p : c$ **bus**
    S
moc

where A is an alphabet of simple symbols, $TR(S)$ is non-empty, and $\mathbf{a}TR(S) = A \cup p{\cdot}A$.

A component of this form is called *directly* recursive. If we stick to the definition of the process associated with c, we have $TR(c) = p{\cdot}TR(c)\ \mathbf{b}\ pref(TR(S))$.

This means that $TR(c)$ is a solution of the equation

$$T \in \boldsymbol{T}(A):\ T = p{\cdot}T\ \mathbf{b}\ pref(TR(S))$$

or, phrased differently, $TR(c)$ is a fixpoint of the function

$$f : \boldsymbol{T}(A) \rightarrow \boldsymbol{T}(A) \text{ defined by } f(T) = p{\cdot}T\ \mathbf{b}\ pref(TR(S))$$

We investigate some properties of this function f.

Property 2.4.0

f is upward continuous and (hence) monotonic.

Proof

f is the composite of the functions $g : \boldsymbol{T}(A) \rightarrow \boldsymbol{T}(p{\cdot}A)$ defined by $g(T) = p{\cdot}T$ and $h : \boldsymbol{T}(p{\cdot}A) \rightarrow \boldsymbol{T}(A)$ defined by $h(U) = U\ \mathbf{b}\ pref(TR(S))$. Function g is just a renaming. It is a lattice isomorphism and has all junctivity properties.

From Theorem 1.6.10 we have that h is upward continuous. Hence, f is upward continuous.

(End of Proof)

From Property 2.4.0, Theorem 1.6.14 (Knaster-Tarski) and Theorem 1.6.15 we infer

Property 2.4.1

f has a least fixpoint and a greatest fixpoint.

The least fixpoint of f equals $(\mathbf{LUB}\, i : i \geqslant 0 : f^{\,i}(STOP(A)))$.

(End of Property)

The process of component c is defined as the least fixpoint of f, i.e.

$$TR(c) = (\mathbf{LUB}\, i : i \geqslant 0 : f^{\,i}(STOP(A)))$$

The following property is useful in calculating the least fixpoint of f.

Property 2.4.2

$$f(T) = \langle A, \{t \mid t \in \mathbf{t}pref(TR(S)) \wedge t\restriction p{\cdot}A \in p{\cdot}\mathbf{t}T\}\restriction A \rangle$$

Proof

We derive

$f(T)$

$=$ { definition of f }

$p{\cdot}T \;\mathbf{b}\; pref(TR(S))$

$=$ { definition of blending }

$(p{\cdot}T \;\mathbf{w}\; pref(TR(S)))\restriction A$

$=$ { Property 1.3.3, $\mathbf{a}p{\cdot}T \subseteq \mathbf{a}pref(TR(S))$ }

$\langle A \cup p{\cdot}A, \{t \mid t \in \mathbf{t}pref(TR(S)) \wedge t\restriction p{\cdot}A \in p{\cdot}\mathbf{t}T\}\rangle \restriction A$

$=$ { definition of projection }

$\langle A, \{t \mid t \in \mathbf{t}pref(TR(S)) \wedge t\restriction p{\cdot}A \in p{\cdot}\mathbf{t}T\}\restriction A \rangle$

(End of Proof)

Example 2.4.3

Component *sem* is defined by

com *sem* (a, b):
  sub p : *sem* **bus**
  $((a \mid p{\cdot}b); (p{\cdot}a \mid b))^*$
moc

We derive

$$
\begin{aligned}
& f(STOP(\{a, b\})) \\
= & \quad \{\text{ definition of } f \ \} \\
& STOP(\{p{\cdot}a, p{\cdot}b\}) \ \mathbf{b} \ pref(TR((a \mid p{\cdot}b); (p{\cdot}a \mid b))^*) \\
= & \quad \{\text{ definition of } SEM_1 \} \\
& STOP(\{p{\cdot}a, p{\cdot}b\}) \ \mathbf{b} \ SEM_1(\{a, p{\cdot}b\}, \{p{\cdot}a, b\}) \\
= & \quad \{\text{ calculus }\} \\
& SEM_1(a, b)
\end{aligned}
$$

and for k, $k \geqslant 1$,

$$
\begin{aligned}
& f(SEM_k(a, b)) \\
= & \quad \{\text{ definition of } f \ \} \\
& SEM_k(p{\cdot}a, p{\cdot}b) \ \mathbf{b} \ SEM_1(\{a, p{\cdot}b\}, \{p{\cdot}a, b\}) \\
= & \quad \{\text{ Theorem 1.4.7 }\} \\
& SEM_{k+1}(a, b)
\end{aligned}
$$

Hence, $TR(sem) = (\mathbf{LUB}\ k : k \geqslant 0 : SEM_k(a, b)) = SEM(a, b)$.

Using the distribution of the semicolon through the bar (Property 2.1.3) one may rewrite the command of *sem*, yielding

$(a; b \mid a; p{\cdot}a \mid p{\cdot}b; b \mid p{\cdot}b; p{\cdot}a)^*$

Denoting this command by S, we have

$u\ p{\cdot}b\ p{\cdot}a\ v \in \mathbf{t}TR(S) \Rightarrow uv \in \mathbf{t}TR(S)$ for any traces u and v,

and also

$u\ p{\cdot}b\ p{\cdot}a\ v \in \mathbf{t}p{\cdot}SEM_k(a, b) \Rightarrow uv \in \mathbf{t}p{\cdot}SEM_k(a, b)$

From these relations and the fact that $p{\cdot}b$ and $p{\cdot}a$ are compound symbols (removed under blending), we infer that the alternative $p{\cdot}b; p{\cdot}a$ of command S can be omitted.

This yields

> **com** $sem(a,b)$:
> **sub** $p : sem$ **bus**
> $(a;b \mid a;p{\cdot}a \mid p{\cdot}b;b)^*$
> **moc**

Command $S0$ of this program has the property that for any t, $t \in \mathbf{t}pref(TR(S0))$, the number of consecutive compound symbols in t is bounded (by two), whereas in the previous command there is no upper bound. In Chapter 5 we discuss such distinctions in more detail.

(End of Example)

Example 2.4.4

We change component sem (cf. Example 2.4.3) into component $zsem$ that has alphabet $\{a,b,z\}$ where z indicates that the lead of a's over b's equals zero. Hence, $TR(zsem)$ should satisfy $TR(zsem) = T$, where T is defined by

$$\mathbf{a}T = \{a,b,z\}$$
$$\mathbf{t}T = \{t \mid t \in \{a,b,z\}^* \wedge t\restriction\{a,b\} \in \mathbf{t}SEM(a,b) \wedge (\mathbf{A}\, s : sz \leqslant t : l(s\restriction a) - l(s\restriction b) = 0)\}$$

We propose a component of the form

> **com** $zsem(a,b,z)$:
> **sub** $p : zsem$ **bus**
> S
> **moc**

We first consider command $S0$ of sem: $S0 = (a;b \mid a;p{\cdot}a \mid p{\cdot}b;b)^*$

For $S0$ the following relations hold.

$$l(t\restriction a) - l(t\restriction b) = l(t\restriction p{\cdot}a) - l(t\restriction p{\cdot}b) \qquad \text{if } t \in \mathbf{t}TR(S0)$$
$$l(t\restriction a) - l(t\restriction b) = l(t\restriction p{\cdot}a) - l(t\restriction p{\cdot}b) + 1 \qquad \text{if } t \in \mathbf{t}pref(TR(S0)) \setminus \mathbf{t}TR(S0)$$

Inspired by these relations (and noticing that $l(\epsilon\restriction a) - l(\epsilon\restriction b) = 0$) we propose

$$S = z^*;(a;b \mid a;p{\cdot}a \mid p{\cdot}b;b \mid p{\cdot}z;z)^*$$

Computation of $f(STOP(\{a,b,z\}))$, where f is the function associated with $zsem$, yields $pref(TR(z^*;(a;b)^*))$.

Some more calculations give rise to the conjecture

$$\mathbf{t}f^{k}(STOP(\{a,b,z\})) =$$
$$\{t \mid t \in \{a,b,z\}^* \wedge t\lceil\{a,b\} \in \mathbf{t}SEM_k(a,b)$$
$$\wedge\ (\mathbf{A}\ s : sz \leqslant t : s\lceil\{a,b\} \in \mathbf{t}SEM_{k-1}(a,b) \wedge l(s\lceil a) - l(s\lceil b) = 0)\}$$

In view of our computation of $TR(sem)$, the conjunct $t\lceil\{a,b\} \in \mathbf{t}SEM_k(a,b)$ is not surprising. The last conjunct, however, is complicated and we use a different way to compute $(\mathbf{LUB}\ k : k \geqslant 0 : f^{k}(STOP(\{a,b,z\})))$.

For k, $k \geqslant 0$, process T_k is defined by

$$\mathbf{a}T_k = \{a,b,z\}$$
$$\mathbf{t}T_k = \{t \mid t \in \{a,b,z\}^* \wedge t\lceil\{a,b\} \in \mathbf{t}SEM_k(a,b) \wedge (\mathbf{A}\ s : sz \leqslant t : l(s\lceil a) - l(s\lceil b) = 0)\}$$

Then $T_0 = \langle\{a,b,z\},\{z\}^*\rangle$ and $T_1 \supseteq pref(TR(z^*;(a;b)^*;z^*))$, hence,

$$T_0 \subseteq f(STOP(\{a,b,z\})) \subseteq T_1$$

Moreover,

$$T = (\mathbf{LUB}\ k : k \geqslant 0 : T_k)$$

We prove $f(T_k) = T_{k+1}$.
Let $k \geqslant 0$.

Due to the similarity of *sem* and *zsem*, we prove only that for all s, $s \in \mathbf{t}f(T_k)$, we have $l(s\lceil a) - l(s\lceil b) = 0 \equiv sz \in \mathbf{t}f(T_k)$

Let $s \in \mathbf{t}f(T_k)$.
Since $f(T_k) = p{\cdot}T_k\ \mathbf{b}\ pref(TR(S))$ we may take w, $w \in \mathbf{t}(p{\cdot}T_k\ \mathbf{w}\ pref(TR(S)))$, such that $s = w\lceil\{a,b,z\}$.
Since $p{\cdot}T_k$ and $pref(TR(S))$ are prefix-closed we assume that w does not end on a compound symbol. Notice that (Property 1.3.3)

$$w \in \mathbf{t}(p{\cdot}T_k\ \mathbf{w}\ pref(TR(S))) \equiv w \in \mathbf{t}pref(TR(S)) \wedge w\lceil\{p{\cdot}a, p{\cdot}b, p{\cdot}z\} \in \mathbf{t}p{\cdot}T_k$$

We derive

$$\begin{array}{ll}
 & l(s\lceil a) - l(s\lceil b) = 0 \\
= & \quad \{ s = w\lceil\{a,b,z\} \} \\
 & l(w\lceil a) - l(w\lceil b) = 0 \\
= & \quad \{ w \in \mathbf{t}pref(TR(S)), \text{hence } l(w\lceil p{\cdot}a) - l(w\lceil p{\cdot}b) \leqslant l(w\lceil a) - l(w\lceil b) \} \\
 & l(w\lceil p{\cdot}a) - l(w\lceil p{\cdot}b) \leqslant 0 \wedge l(w\lceil a) - l(w\lceil b) = 0 \\
= & \quad \{ w\lceil\{p{\cdot}a, p{\cdot}b\} \in \mathbf{t}SEM_k(p{\cdot}a, p{\cdot}b) \} \\
 & l(w\lceil p{\cdot}a) - l(w\lceil p{\cdot}b) = 0 \wedge l(w\lceil a) - l(w\lceil b) = 0
\end{array}$$

$$
\begin{aligned}
= &\quad \{ \text{structure of } S, w \text{ does not end in } p{\cdot}z \} \\
& l(w \restriction p{\cdot}a) - l(w \restriction p{\cdot}b) = 0 \wedge w \in \mathbf{t}TR(S) \\
= &\quad \{ w \restriction \{p{\cdot}a, p{\cdot}b, p{\cdot}z\} \in \mathbf{t}p{\cdot}T_k, \text{ definition of } T_k \} \\
& (w \restriction \{p{\cdot}a, p{\cdot}b, p{\cdot}z\})\, p{\cdot}z \in \mathbf{t}p{\cdot}T_k \wedge w \in \mathbf{t}TR(S) \\
= &\quad \{ \text{definition of projection, structure of } S \} \\
& w\ p{\cdot}z\ z \restriction \{p{\cdot}a, p{\cdot}b, p{\cdot}z\} \in \mathbf{t}p{\cdot}T_k \wedge w\ p{\cdot}z\ z \in \mathbf{t}\mathit{pref}(TR(S)) \\
= &\quad \{ \text{definition of weaving} \} \\
& w\ p{\cdot}z\ z \in \mathbf{t}(p{\cdot}T_k\ \mathbf{w}\ \mathit{pref}(TR(S))) \\
= &\quad \{ w \restriction \{a, b, z\} = s, \text{ structure of } S, w \text{ does not end in a compound symbol} \} \\
& sz \in \mathbf{t}(p{\cdot}T_k\ \mathbf{b}\ \mathit{pref}(TR(S))) \\
= &\quad \{ \text{definition of } f \} \\
& sz \in \mathbf{t}f(T_k)
\end{aligned}
$$

Hence, $f(T_k) = T_{k+1}$

Finally, we derive

$$
\begin{aligned}
& T_0 \subseteq f(\mathit{STOP}(\{a,b,z\})) \subseteq T_1 \\
\Rightarrow &\quad \{ f \text{ is monotonic} \} \\
& (\mathbf{LUB}\ k : k \geq 0 : f^k(T_0)) \subseteq (\mathbf{LUB}\ k : k \geq 1 : f^k(\mathit{STOP}(\{a,b,z\}))) \\
& \qquad \subseteq (\mathbf{LUB}\ k : k \geq 0 : f^k(T_1)) \\
= &\quad \{ f(T_k) = T_{k+1}, \text{ definition of } \mathit{zsem} \} \\
& (\mathbf{LUB}\ k : k \geq 0 : T_k) \subseteq TR(\mathit{zsem}) \subseteq (\mathbf{LUB}\ k : k \geq 1 : T_k) \\
= &\quad \{ \text{definition of } T \} \\
& T \subseteq TR(\mathit{zsem}) \subseteq T \\
= &\quad \{ \text{antisymmetry of } \subseteq \} \\
& TR(\mathit{zsem}) = T
\end{aligned}
$$

(End of Example)

The theory of this section is easily extended to components with more than one subcomponent of the same type and to components with previously defined subcomponents as well.

E.g., component c defined by

com $c(A)$:
  sub $p, q : c, r : d$ **bus**
    S
moc

has trace structure $(\mathbf{LUB}\ k : k \geqslant 0 : f^k(STOP(A)))$ where $f : \mathcal{T}(A) \to \mathcal{T}(A)$ is defined by

$$f(T) = p{\cdot}T \ \mathbf{b}\ q{\cdot}T \ \mathbf{b}\ r{\cdot}TR(d) \ \mathbf{b}\ pref(TR(S))$$

Exercises

0. Determine the process of component *cat* defined by

   com $cat(a, b)$:
     sub $p : cat$ **bus**
       $(a; p{\cdot}a)^*; a; b; (p{\cdot}b; b)^*$
   moc

1. Derive a component that represents an integer value. The initial value is zero. The alphabet is $\{a, b, z\}$ where a denotes an increment by one, b denotes a decrement by one, and z denotes 'the value equals zero'.

2. A binary bag is a component that accepts zeroes and ones. A previously stored zero or one may be retrieved. Give a formal specification of a process that specifies such a bag, and derive a component according to that specification.

3. Prove that component *sem* defined by

   com $sem(a, b)$:
     sub $p : sem$ **bus**
       $a; ((p{\cdot}a; a \mid b); (a \mid p{\cdot}b; b))^*$
   moc

   has trace structure $SEM(a, b)$.

4. Determine the least and the greatest fixpoints of the functions associated with **com** $c(a, b)$: **sub** $p : c$ **bus** S **moc**

   where S is given by

   (i) $(a; p{\cdot}a; p{\cdot}b; b)^*$

(ii) $(a;p\cdot a;b;p\cdot b)^*$

(iii) $(p\cdot a;a;b;p\cdot b)^*$

(iv) $(a;b;p\cdot a;p\cdot b)^*$

(v) $(a;b;p\cdot b;p\cdot a)^*$

(End of Exercises)

2.5 Unique fixpoints of recursive components

In this section we take a closer look at the fixpoints of the function associated with a directly recursive component. A generalization of the theory of this section can be found in [11]. There is, however, a difference in the lattices that are considered. Let component c be defined by

com $c(A)$:
  sub $p : c$ **bus**
   S
moc

and let $f : \boldsymbol{T}(A) \rightarrow \boldsymbol{T}(A)$ be the associated function, i.e.

$$f(T) = p\cdot T \;\mathbf{b}\; pref(TR(S)).$$

We study conditions under which f has exactly one fixpoint.

First we switch from processes to trace sets. For the sake of brevity we define trace set U by $U = \mathbf{t}pref(TR(S))$.

For a fixpoint $<A,V>$ of f we derive

$$
\begin{array}{ll}
 & <A,V> \text{ is a fixpoint of } f \\
= & \quad \{ \text{definition of } f \} \\
 & <A,V> \;=\; <p\cdot A, p\cdot V> \;\mathbf{b}\; pref(TR(S)) \\
= & \quad \{ \text{Property 2.4.2, definition of } U \} \\
 & <A,V> \;=\; <A, \{t \mid t \in U \;\wedge\; t\lceil p\cdot A \in p\cdot V\}\lceil A> \\
= & \quad \{ \text{set calculus} \} \\
 & <A,V> \;=\; <A, \{t \mid t \in U \;\wedge\; (\mathbf{E}\, v : v \in V : t\lceil p\cdot A = p\cdot v)\}\lceil A> \\
= & \quad \{ A = A \} \\
 & V \;=\; \{t \mid t \in U \;\wedge\; (\mathbf{E}\, v : v \in V : t\lceil p\cdot A = p\cdot v)\}\lceil A
\end{array}
$$

For a non-empty prefix-closed trace set Y we define $Q(Y)$ by

$$Q(Y) = \{X \mid X \subseteq Y \wedge X \text{ is non-empty and prefix-closed }\}$$

$(Q(Y), \subseteq)$ is a complete lattice with least element $\{\epsilon\}$ and greatest element Y.

In view of the derivation above we define $g : Q(A^*) \to Q(A^*)$ by

$$g(V) = \{t \mid t \in U \wedge (\mathbf{E}\, v : v \in V : t{\upharpoonright}p{\cdot}A = p{\cdot}v)\}{\upharpoonright}A$$

Then the following property holds.

Property 2.5.0

$$<A, V> \text{ is a fixpoint of } f \quad \equiv \quad V \text{ is a fixpoint of } g$$

(End of Property)

Inspired by the computations of least fixpoints (cf. Section 2.4) we define another function $h : Q(U) \to Q(U)$, which is closely related to f, by

$$h(W) = \{t \mid t \in U \wedge (\mathbf{E}\, w : w \in W : t{\upharpoonright}p{\cdot}A = p{\cdot}(w{\upharpoonright}A))\}$$

Finally, we define two functions G and H that relate $g, h, Q(A^*)$, and $Q(U)$:

$G : Q(A^*) \to Q(U)$ with $G(V) = \{t \mid t \in U \wedge (\mathbf{E}\, v : v \in V : t{\upharpoonright}p{\cdot}A = p{\cdot}v)\}$

$H : Q(U) \to Q(A^*)$ with $H(W) = W{\upharpoonright}A$

We then have (cf. Figure 2.2)

$$\begin{tikzcd}
Q(A^*) \arrow[r, "g"] \arrow[dr, "G"] & Q(A^*) \\
Q(U) \arrow[u, "H"] \arrow[r, "h"'] & Q(U) \arrow[u, "H"']
\end{tikzcd}$$

Figure 2.2

Property 2.5.1

$$g = H \circ G \quad \text{and} \quad h = G \circ H$$

(End of Property)

Property 2.5.2

G and H are upward continuous.

Proof

Let $V(i : i \geqslant 0)$ be an ascending chain in $Q(A^*)$. We derive

$$\begin{array}{cl}
 & t \in G((\cup\, i : i \geqslant 0 : V(i))) \\
= & \quad \{ \text{definition of } G \} \\
 & t \in U \wedge (\mathbf{E}\, v : v \in (\cup\, i : i \geqslant 0 : V(i)) : t{\upharpoonright}p{\cdot}A = p{\cdot}v)
\end{array}$$

$=$ { definition of union }

$t \in U \ \wedge \ (\mathbf{E}\, v : (\mathbf{E}\, i : i \geqslant 0 : v \in V(i)) : t{\restriction}p{\cdot}A = p{\cdot}v)$

$=$ { predicate calculus }

$(\mathbf{E}\, i : i \geqslant 0 : t \in U \ \wedge \ (\mathbf{E}\, v : v \in V(i) : t{\restriction}p{\cdot}A = p{\cdot}v))$

$=$ { definition of G }

$(\mathbf{E}\, i : i \geqslant 0 : t \in G(V(i)))$

$=$ { set calculus }

$t \in (\cup\, i : i \geqslant 0 : G(V(i)))$

Hence, $G((\cup\, i : i \geqslant 0 : V(i))) = (\cup\, i : i \geqslant 0 : G(V(i)))$.

The upward continuity of H is a consequence of Corollary 1.6.5 (projection is upward continuous).

(End of Proof)

From Property 2.5.1 and Property 2.5.2 we deduce

Property 2.5.3

g and h are upward continuous

(End of Property)

The following theorem shows how the fixpoints of g and h are related.

Theorem 2.5.4

0 V is a fixpoint of $g \Rightarrow G(V)$ is a fixpoint of h

1 W is a fixpoint of $h \Rightarrow H(W)$ is a fixpoint of g

2 The poset of fixpoints of g is isomorphic to the poset of fixpoints of h

Proof

0. Assume V is a fixpoint of g. We derive

$h(G(V))$

$=$ { Property 2.5.1 }

$G \circ H(G(V))$

$=$ { Property 2.5.1 }

$G(g(V))$

$=$ { V is a fixpoint of g }

$G(V)$

1. Similar

2. For a fixpoint V of g we have $H \circ G(V) = g(V) = V$.

   For a fixpoint W of h we have $G \circ H(W) = h(W) = W$.

   Hence, G and H are bijections between the fixpoints of g and those of h, the one being the inverse of the other.

   Furthermore (Property 2.5.2), both G and H are monotonic.

(End of Proof)

Since g and h are upward continuous, both have a least fixpoint and a greatest fixpoint (Knaster-Tarski).

The least fixpoint of h equals $(\cup\ i : i \geq 0 : h^i(\{\epsilon\}))$ and is denoted by μh. The greatest fixpoint of h is denoted by νh. Since μh is the greatest lower bound of *all* fixpoints of h, we have $\mu h \subseteq \nu h$.

Application of Property 2.5.0 and Theorem 2.5.4 yields

Property 2.5.5

$$f \text{ has one fixpoint} \equiv \nu h \subseteq \mu h$$

(End of Property)

We have now obtained a very nice result. From the text of component c it is clear that only the structure of command S can play a role. And indeed, we have shown that all information is in the function h which was defined by (replacing U by $\mathbf{t}pref(TR(S))$)

$$h : Q(\mathbf{t}pref(TR(S))) \to Q(\mathbf{t}pref(TR(S)))$$

$$h(W) = \{t \mid t \in \mathbf{t}pref(TR(S)) \wedge (\mathbf{E}\, w : w \in W : t\restriction p \cdot A = p \cdot (w \restriction A))\}$$

The following theorem has also been proved in [11].

Theorem 2.5.6

If $(\mathbf{A}\, u : u \in \nu h \wedge u{\restriction}A \neq \epsilon : (\mathbf{E}\, w : w \in \nu h : u{\restriction}A = w{\restriction}A \wedge l(w{\restriction}p{\cdot}A) < l(w{\restriction}A)))$
then f has exactly one fixpoint.

Proof

Assume that the given condition holds (referred to as 'assumption').
We prove by induction on $l(u{\restriction}p{\cdot}A)$ that $u \in \nu h \Rightarrow u \in \mu h$.
The theorem then follows from Property 2.5.5 .

Base $u{\restriction}p{\cdot}A = \epsilon$

From $h(\{\epsilon\}) = \{t \mid t \in \mathbf{t}pref(TR(S)) \wedge t{\restriction}p{\cdot}A = \epsilon\}$ we conclude $u \in h(\{\epsilon\})$, and hence,
$u \in (\cup\, i : i \geq 0 : h^i(\{\epsilon\})) = \mu h$

Step $l(u{\restriction}p{\cdot}A) > 0$

We derive

$$
\begin{array}{ll}
 & u \in \nu h \\
= & \quad \{ \nu h \text{ is a fixpoint of } h \} \\
 & u \in h(\nu h) \\
= & \quad \{ \text{definition of } h,\, u \in \nu h \subseteq \mathbf{t}pref(TR(S)) \} \\
 & (\mathbf{E}\, v : v \in \nu h : u{\restriction}p{\cdot}A = p{\cdot}(v{\restriction}A)) \\
= & \quad \{ \text{assumption, } l(u{\restriction}p{\cdot}A) > 0 \text{, hence } v{\restriction}A \neq \epsilon \} \\
 & (\mathbf{E}\, v : v \in \nu h : u{\restriction}p{\cdot}A = p{\cdot}(v{\restriction}A) \wedge (\mathbf{E}\, w : w \in \nu h : v{\restriction}A = w{\restriction}A \wedge l(w{\restriction}p{\cdot}A) < l(w{\restriction}A))) \\
\Rightarrow & \quad \{ \text{predicate calculus} \} \\
 & (\mathbf{E}\, w : w \in \nu h : u{\restriction}p{\cdot}A = p{\cdot}(w{\restriction}A) \wedge l(w{\restriction}p{\cdot}A) < l(u{\restriction}p{\cdot}A)) \\
\Rightarrow & \quad \{ \text{induction hypothesis} \} \\
 & (\mathbf{E}\, w : w \in \nu h : u{\restriction}p{\cdot}A = p{\cdot}(w{\restriction}A) \wedge w \in \mu h) \\
= & \quad \{ \mu h \subseteq \nu h \} \\
 & (\mathbf{E}\, w : w \in \mu h : u{\restriction}p{\cdot}A = p{\cdot}(w{\restriction}A)) \\
= & \quad \{ \text{definition of } h \} \\
 & u \in \mu h
\end{array}
$$

(End of Proof)

It is, in general, not easy to compute νh. We weaken Theorem 2.5.6 to a theorem that is more easily applied. The next theorem can also be found in [20].

Theorem 2.5.7

If $(\mathbf{A}\, u : u \in \mathbf{t}\, pref(TR(S)) : l(u \upharpoonright p \cdot A) \leq l(u \upharpoonright A))$ then f has exactly one fixpoint.

Proof

Assume that the given condition holds. We show that the condition of Theorem 2.5.6 holds as well. We derive

$$
\begin{array}{ll}
 & u \in \nu h \ \wedge \ u \upharpoonright A \neq \epsilon \\
= & \qquad \{\text{ calculus }\} \\
 & u \in \nu h \ \wedge \ (\mathbf{E}\, s, t, a : a \in A : u = sat \ \wedge \ t \upharpoonright A = \epsilon) \\
\Rightarrow & \qquad \{\ \nu h \text{ is prefix-closed }\} \\
 & (\mathbf{E}\, s, a : a \in A : u \upharpoonright A = sa \upharpoonright A \ \wedge \ sa \in \nu h \ \wedge \ s \in \nu h) \\
\Rightarrow & \qquad \{\text{ assumption applied to } s\ \} \\
 & (\mathbf{E}\, s, a : a \in A : u \upharpoonright A = sa \upharpoonright A \ \wedge \ sa \in \nu h \ \wedge \ l(s \upharpoonright p \cdot A) \leq l(s \upharpoonright A)) \\
= & \qquad \{\text{ property of projection and length }\} \\
 & (\mathbf{E}\, s, a : a \in A : u \upharpoonright A = sa \upharpoonright A \ \wedge \ sa \in \nu h \ \wedge \ l(sa \upharpoonright p \cdot A) < l(sa \upharpoonright A)) \\
\Rightarrow & \qquad \{\text{ predicate calculus }\} \\
 & (\mathbf{E}\, w : w \in \nu h : u \upharpoonright A = w \upharpoonright A \ \wedge \ l(w \upharpoonright p \cdot A) < l(w \upharpoonright A))
\end{array}
$$

(End of Proof)

Example 2.5.8

Component *ex* is defined by

```
com ex(a):
  sub p : ex  bus
   (a ; p·a)*
moc
```

Component *ex* satisfies the condition of Theorem 2.5.7 .

Hence, any solution of $T = p \cdot T\ \mathbf{b}\ pref(TR((a\,;p \cdot a)^*))$ is the least solution of it. We show that $RUN(a)$ is a solution.

$$
\begin{array}{ll}
 & RUN(p \cdot a)\ \mathbf{b}\ pref(TR((a\,;p \cdot a)^*)) \\
= & \qquad \{\text{ Property 1.4.2.3 }\} \\
 & pref(TR((a\,;p \cdot a)^*)) \upharpoonright \{a\} \\
= & \qquad \{\text{ calculus }\} \\
 & RUN(a)
\end{array}
$$

Hence, $TR(ex) = RUN(a)$

(End of Example)

Example 2.5.9

This example demonstrates that the condition of Theorem 2.5.7 is not necessary.

com $ex(a,b)$:
  sub $p : ex$ **bus**
  $(a ; p{\cdot}b \mid p{\cdot}a ; b)$
moc

Trace $p{\cdot}a\ b$ does not satisfy the requirement of Theorem 2.5.7 .
However, $h(\{\epsilon\}) = \{\epsilon, a\}$, $h(\{\epsilon, a\}) = \{\epsilon, a, p{\cdot}a, p{\cdot}a\ b\}$ and
$h(\{\epsilon, a, p{\cdot}a, p{\cdot}a\ b\}) = \{\epsilon, a, p{\cdot}a, p{\cdot}a\ b, a\ p{\cdot}b\} = \mathbf{t}pref(TR(S))$

Hence, $\mu h = \nu h = \mathbf{t}pref(TR(S))$: we have exactly one fixpoint.

(End of Example)

Example 2.5.10

This example demonstrates that Theorem 2.5.7 is indeed weaker than Theorem 2.5.6.

com $ex(a)$:
  sub $: ex$ **bus**
  $(a \mid p{\cdot}a)^*$
moc

The condition of Theorem 2.5.7 is not satisfied: the lead of compound symbols over simple symbols is unbounded. The greatest fixpoint of h equals $RUN(a, p{\cdot}a)$ and for all $t, t \in \mathbf{t}pref(TR(S))$, we have $t \restriction \{a\} \in \mathbf{t}pref(TR(S))$. Hence, the condition of Theorem 2.5.6 is satisfied and $TR(ex) = RUN(a)$.

(End of Example)

For a discussion of other forms of recursion we recommend [11].

Exercises

0. Prove that for g, h, G, and H as defined in this section

   (i) $h(W)\upharpoonright A \;=\; g(W\upharpoonright A)$

   (ii) $G \circ g \;=\; h \circ G$

1. Prove that component *cat* defined by

   com *cat*(a,b):
     sub p : *cat* **bus**
     $(a\,;p{\cdot}a)^*;a\,;b\,;(p{\cdot}b\,;b)^*$
   moc

   has a unique fixpoint, viz. $<\{a,b\},\{t \mid (\mathbf{E}\, m,n : 0 \leqslant m \leqslant n : t = a^n b^m)\}>$.

2. Determine the process of

   com *rem*(a,b):
     sub p : *rem* **bus**
     $a\,;((p{\cdot}a\,;a \mid b);(a \mid p{\cdot}b\,;b))^*$
   moc

3. Let $f\,:\,\boldsymbol{T}(A) \rightarrow \boldsymbol{T}(A)$ be upward continuous and let $T, T \in \boldsymbol{T}(A)$, be a fixpoint of f such that

   $(\mathbf{A}\, t : t \in \mathbf{t}T \;\wedge\; t \neq \epsilon : (\mathbf{E}\, s : s \in \mathbf{t}T : l(s) < l(t) \;\wedge\; t \in \mathbf{t}f(<A, pref(\{s\})>))$

   Prove that T is the least fixpoint of f . Apply the above to the recursive component with alphabet $\{a,b\}$ and command $(a\,;p{\cdot}a \mid a\,;b \mid p{\cdot}b\,;b)^*$.

(End of Exercises)

3 From specification to program text

3.0 Introduction

As we have seen in the previous chapters there are many ways in which a process may be specified. One may use enumeration, a state graph, a program text or a predicate. A predicate specifies what traces do belong to the trace set of a process, whereas a program text suggests how the traces of the process may be generated.
In this chapter we formalize the notion of a specification. Furthermore we present some theorems that are useful in the derivation of a program text from such a specification.

3.1 Specifications

A *specification* of a process is a pair $<A, P>$, where A is an alphabet and P is a predicate on A^* such that $P(\epsilon)$ holds.
Specification $<A, P>$ specifies the process

$$<A, \{t \mid t \in A^* \wedge (\mathbf{A}\, s : s \leqslant t : P(s))\}>$$

Note

Let $<A, P>$ specify T. From $P(\epsilon)$ we infer $\epsilon \in \mathbf{t}T$. For any $t, t \in A^*$, we have

$$\begin{array}{ll} & (\mathbf{A}\, s : s \leqslant t : P(s)) \\ = & \quad \{ t \in A^* \} \\ & (\mathbf{A}\, s : s \leqslant t : s \in A^* \wedge P(s)) \\ = & \quad \{ \text{calculus, transitivity of } \leqslant \} \\ & (\mathbf{A}\, s : s \leqslant t : s \in A^* \wedge (\mathbf{A}\, v : v \leqslant s : P(v))) \end{array}$$

Hence, $<A, P>$ specifies a process, i.e. a non-empty prefix-closed trace structure.

(End of Note)

Instead of using a lambda-notation like $<A, (\lambda t : t \in A^* : P(t))>$ we use the notation $<A, t : P(t)>$.

Example 3.1.0

$SEM_1(a,b)$ is specified by $<\{a,b\}, t : 0 \leqslant l(t \restriction a) - l(t \restriction b) \leqslant 1>$
$SEM(a,b)$ is specified by $<\{a,b\}, t : 0 \leqslant l(t \restriction a) - l(t \restriction b)>$

$<\{a\}, t : l(t)$ is even $>$ specifies $STOP(a)$

(End of Example)

The following property is useful when one wants to enumerate the traces of a process given by a specification, up to some fixed length.

Property 3.1.1

Let $<A, P>$ be a specification of T. Then T is the least solution of

$$U \in \mathcal{T}(A): \quad (\mathbf{A}\, t, a : t \in \mathbf{t}U \wedge a \in A : P(ta) \Rightarrow ta \in \mathbf{t}U)$$

Proof

For any $t, t \in \mathbf{t}T$, and symbol $a, a \in A$, we have

$t \in \mathbf{t}T \wedge a \in A \wedge P(ta)$
$=$ $\{ <A, P>$ specifies $T \}$
$(\mathbf{A}\, s : s \leqslant t : P(s)) \wedge t \in A^* \wedge a \in A \wedge P(ta)$
$=$ $\{$ calculus $\}$
$(\mathbf{A}\, s : s \leqslant ta : P(s)) \wedge ta \in A^*$
$=$ $\{ <A, P>$ specifies $T \}$
$ta \in \mathbf{t}T$

Hence, T is a solution of the equation. Let $U, U \in \mathcal{T}(A)$, be a solution. By induction on the length of t we prove that for all $t, t \in \mathbf{t}T$, we have $t \in \mathbf{t}U$.

Base $t = \epsilon$. Since $U \in \mathcal{T}(A)$, we have $\epsilon \in \mathbf{t}U$.

Step $t = sa$ with $a \in A$. We derive

$sa \in \mathbf{t}T$
$=$ $\{ T$ is prefix-closed $\}$
$s \in \mathbf{t}T \wedge a \in A \wedge sa \in \mathbf{t}T$
$\Rightarrow$ $\{$ induction hypothesis $\}$
$s \in \mathbf{t}U \wedge a \in A \wedge sa \in \mathbf{t}T$

$\Rightarrow \quad \{ <A\,,P> \text{ specifies } T \}$
$s \in \mathbf{t}U \wedge a \in A \wedge P(sa)$
$\Rightarrow \quad \{ U \text{ is a solution of the equation} \}$
$sa \in \mathbf{t}U$

(End of Proof)

Let T be specified by $<A\,,P>$. From the property above we infer that $\mathbf{t}T$ is determined by the following rules.

(i) $\epsilon \in \mathbf{t}T$

(ii) $t \in \mathbf{t}T \wedge a \in A \wedge P(ta) \Rightarrow ta \in \mathbf{t}T$

(iii) $\mathbf{t}T$ contains no other traces then those that belong to it on account of (i) and (ii).

Example 3.1.2

The traces of length at most three of the process specified by

$$<\{a\,,b\},\, t : 2 \cdot l(t\restriction a) - l(t\restriction b) < 2>$$

are $\epsilon, b\,, ba\,, bb\,, bab\,, bba$ and bbb.

(End of Example)

Property 3.1.3

Let $<A\,,P>$ be a specification of T. Then T is the greatest solution of

$$U \in \mathcal{T}(A): \quad (\mathbf{A}\, t : t \in \mathbf{t}U : P(t))$$

Proof

We derive

$<A\,,P>$ specifies T
$\Rightarrow \quad \{ \text{definition of 'specifies'} \}$
$(\mathbf{A}\, t : t \in \mathbf{t}T : P(t))$

Hence, T is a solution. For any $U, U \in \mathcal{T}(A)$, we have

$(\mathbf{A}\, t : t \in \mathbf{t}U : P(t))$
$= \quad \{ U \text{ is prefix-closed, } \mathbf{a}U = A \}$

$$
\begin{array}{ll}
 & (\mathbf{A}\, t : t \in \mathbf{t}U : t \in A^* \wedge (\mathbf{A}\, s : s \leqslant t : P(s))) \\
= & \qquad \{ <A, P> \text{ specifies } T \} \\
 & (\mathbf{A}\, t : t \in \mathbf{t}U : t \in \mathbf{t}T) \\
= & \qquad \{ \mathbf{a}T = \mathbf{a}U \} \\
 & U \subseteq T
\end{array}
$$

(End of Proof)

We now list some examples that are used in the next sections. All these examples involve storage and retrieval of zeroes and ones. We use the following symbols.

$a0$: a zero is stored

$a1$: a one is stored

$b0$: a zero is retrieved

$b1$: a one is retrieved

Example 3.1.4 (bounded bag)

For natural number k a k-bounded bag is specified by

$$
\begin{array}{ll}
<\{a0, a1, b0, b1\}, t : & l(t \upharpoonright b0) \leqslant l(t \upharpoonright a0) \\
 & \wedge\ l(t \upharpoonright b1) \leqslant l(t \upharpoonright a1) \\
 & \wedge\ 0 \leqslant l(t \upharpoonright \{a0, a1\}) - l(t \upharpoonright \{b0, b1\}) \leqslant k \\
>
\end{array}
$$

(End of Example)

Example 3.1.5 (unbounded bag)

An unbounded bag is specified by

$$<\{a0, a1, b0, b1\}, t : l(t \upharpoonright b0) \leqslant l(t \upharpoonright a0) \wedge\ l(t \upharpoonright b1) \leqslant l(t \upharpoonright a1)>$$

(End of Example)

Example 3.1.6 (unbounded sorter)

An unbounded sorter is specified by

$$< \{a0, a1, b0, b1\}, t:\ l(t{\restriction}b0) \leq l(t{\restriction}a0) \wedge l(t{\restriction}b1) \leq l(t{\restriction}a1)$$
$$\wedge\ (\mathbf{A}\, s : t = sb1 : l(s{\restriction}a0) = l(s{\restriction}b0))$$
$$>$$

(End of Example)

Exercises

0. Let $<A, P>$ be a specification. Show that $RUN(A)$ is the greatest solution of

$$U \in \mathcal{T}(A):\quad (\mathbf{A}\, t, a : t \in \mathbf{t}U \wedge a \in A : P(ta) \Rightarrow ta \in \mathbf{t}U)$$

Show that $STOP(A)$ is the least solution of

$$U \in \mathcal{T}(A):\quad (\mathbf{A}\, t : t \in \mathbf{t}U : P(t))$$

1. Specify a k-bounded sorter (cf. Example 3.1.6).

2. Extend the specification of a bag (Example 3.1.5) such that symbol e corresponds to 'the bag is empty'.

3. Give a specification of the following mechanisms.
   (i) A binary first-in first-out queue.
   (ii) The mechanism accepts a series of zeroes followed by a one, after which it delivers the same number of zeroes followed by a one.
   (iii) The mechanism generates any sequence of a's, b's, and c's in which no two adjacent symbols are equal.
   (iv) The mechanism generates the sequence of positive numbers as follows. First one a is generated, then two a's are generated, and so on. Between each sequence of a's a b is generated. Typical traces are a, aba, and $abaabaaaba$.
   (v) The mechanism represents a natural number, initially zero. Possible events are
   u : increment value by one (up)
   d : decrement value by one (down)
   z : the value equals zero (zero)
   (vi) The same as (v) but now negative values are allowed: the mechanism represents an integer.

(End of Exercises)

3.2 The Conjunction-Weave Rule

In this section we investigate the relation between processes and their weave, in terms of specifications. Our first theorem is called the Conjunction-Weave Rule, abbreviated as CW-rule.

Theorem 3.2.0 (CW-rule)

Let $<A, P>$ and $<B, Q>$ be specifications of processes T and U respectively. Then $T \mathbf{w} U$ is specified by

$$<A \cup B, t : P(t \upharpoonright A) \wedge Q(t \upharpoonright B)>$$

Proof

$<A \cup B, t : P(t \upharpoonright A) \wedge Q(t \upharpoonright B)>$ is a specification, since

$$\begin{array}{ll} & P(\epsilon \upharpoonright A) \wedge Q(\epsilon \upharpoonright B) \\ = & \quad \{ \text{definition of projection} \} \\ & P(\epsilon) \wedge Q(\epsilon) \\ = & \quad \{ <A, P> \text{ and } <B, Q> \text{ are specifications} \} \\ & \text{true} \end{array}$$

Furthermore, $\mathbf{a}(T \mathbf{w} U) = A \cup B$ and for any t, $t \in (A \cup B)^*$, we have

$$\begin{array}{ll} & t \in \mathbf{t}(T \mathbf{w} U) \\ = & \quad \{ \text{definition of weaving} \} \\ & t \upharpoonright A \in \mathbf{t}T \wedge t \upharpoonright B \in \mathbf{t}U \\ = & \quad \{ <A, P> \text{ specifies } T \text{ and } <B, Q> \text{ specifies } U \} \\ & (\mathbf{A}\, s : s \leqslant t \upharpoonright A : P(s)) \wedge (\mathbf{A}\, s : s \leqslant t \upharpoonright B : Q(s)) \\ = & \quad \{ \text{Property 1.1.4.3} \} \\ & (\mathbf{A}\, s : s \leqslant t : P(s \upharpoonright A)) \wedge (\mathbf{A}\, s : s \leqslant t : Q(s \upharpoonright B)) \\ = & \quad \{ \text{predicate calculus} \} \\ & (\mathbf{A}\, s : s \leqslant t : P(s \upharpoonright A) \wedge Q(s \upharpoonright B)) \end{array}$$

(End of Proof)

Example 3.2.1

An unbounded bag is specified by

$$<\{a0, a1, b0, b1\}, t : l(t \upharpoonright b0) \leqslant l(t \upharpoonright a0) \wedge l(t \upharpoonright b1) \leqslant l(t \upharpoonright a1)>$$

We derive component *bag* such that $TR(bag)$ is the process specified above. From the specification of $SEM(a,b)$, viz. $<\{a,b\}, t: l(t\upharpoonright b) \leq l(t\upharpoonright a)>$, and the CW-rule we infer

$$TR(bag) = SEM(a0,b0) \mathbf{\ w\ } SEM(a1,b1)$$

Since $\{a0,b0\} \cap \{a1,b1\} = \varnothing$, we may replace this weave by a blend:

$$TR(bag) = SEM(a0,b0) \mathbf{\ b\ } SEM(a1,b1)$$

These observations lead to the following solution in which *sem* is the component of Example 2.4.3 .

```
com bag(a0,a1,b0,b1):
  sub p,q : sem bus
  [p·a = a0, p·b = b0, q·a = a1, q·b = b1]
   ε
moc
```

(End of Example)

In the example above we replaced a weave by a blend which is allowed on account of Property 1.4.0 . The following theorem, also called the Composition Rule, shows a more general method.

Theorem 3.2.2 (Composition Rule)

Let c and d be components with alphabets A and B respectively.
Let $A \cup B = \{x_0, \ldots, x_{n-1}\}$ and let component cd be defined by

```
com cd(A ∪ B):
  sub p : c, q : d bus
  (S_0 | ··· | S_{n-1})*
moc
```

where for $i, 0 \leq i < n$,

$$\begin{array}{lll} S_i = p\cdot x_i ; x_i & \text{if } x_i \in A \setminus B \\ S_i = q\cdot x_i ; x_i & \text{if } x_i \in B \setminus A \\ S_i = p\cdot x_i , q\cdot x_i ; x_i & \text{if } x_i \in A \cap B \end{array}$$

Then $TR(cd) = TR(c) \mathbf{\ w\ } TR(d)$

Proof

The alphabets of $TR(cd)$ and $TR(c) \mathbf{\ w\ } TR(d)$ are equal, viz. $A \cup B$. Let S denote the command of cd. We compute

$TR(cd)$

$=$ { definition of the process of a component }

$(p\cdot TR(c)\,\mathbf{w}\,q\cdot TR(d)\,\mathbf{w}\,pref(TR(S)))\upharpoonright(A\cup B)$

$=$ { structure of S, compound symbols are removed by $\upharpoonright(A\cup B)$ }

$(p\cdot TR(c)\,\mathbf{w}\,q\cdot TR(d)\,\mathbf{w}\,TR(S))\upharpoonright(A\cup B)$

From the structure of S we infer

$$t\in\mathbf{t}TR(S)\ \Rightarrow\ t\upharpoonright p\cdot A = p\cdot(t\upharpoonright A)\wedge t\upharpoonright q\cdot B = q\cdot(t\upharpoonright B)$$

Hence,

$t\in\mathbf{t}(p\cdot TR(c)\,\mathbf{w}\,q\cdot TR(d)\,\mathbf{w}\,TR(S))$

$\Rightarrow$ { definition of weaving }

$t\upharpoonright p\cdot A\in\mathbf{t}p\cdot TR(c)\wedge t\upharpoonright q\cdot B\in\mathbf{t}q\cdot TR(d)\wedge t\in\mathbf{t}TR(S)$

$\Rightarrow$ { structure of S, see above }

$t\upharpoonright p\cdot A\in\mathbf{t}p\cdot TR(c)\wedge t\upharpoonright q\cdot B\in\mathbf{t}q\cdot TR(d)\wedge t\upharpoonright p\cdot A = p\cdot(t\upharpoonright A)\wedge t\upharpoonright q\cdot B = q\cdot(t\upharpoonright B)$

$\Rightarrow$ { substitution }

$p\cdot(t\upharpoonright A)\in p\cdot\mathbf{t}TR(c)\wedge q\cdot(t\upharpoonright B)\in q\cdot\mathbf{t}TR(d)$

$=$ { definition of $p\cdot$ }

$t\upharpoonright A\in\mathbf{t}TR(c)\wedge t\upharpoonright B\in\mathbf{t}TR(d)$

Together with our computation of $TR(cd)$ this yields

$$TR(cd)\sqsubseteq TR(c)\,\mathbf{w}\,TR(d)$$

We are left with the proof obligation $\mathbf{t}(TR(c)\,\mathbf{w}\,TR(d))\subseteq\mathbf{t}TR(cd)$.
This is done by constructing a function h from $\mathbf{t}(TR(c)\,\mathbf{w}\,TR(d))$ into $\mathbf{t}(p\cdot TR(c)\,\mathbf{w}\,q\cdot TR(d)\,\mathbf{w}\,pref(TR(S)))$ such that $h(t)\upharpoonright(A\cup B) = t$

h is defined inductively by

$$\begin{array}{lll} h(\epsilon) = \epsilon & & \\ h(ta) = h(t)\,p\cdot a\ a & \text{if } a\in A\backslash B \\ h(ta) = h(t)\,q\cdot a\ a & \text{if } a\in B\backslash A \\ h(ta) = h(t)\,p\cdot a\ q\cdot a\ a & \text{if } a\in A\cap B \end{array}$$

Then, evidently, $h(t)\in\mathbf{t}(p\cdot TR(c)\,\mathbf{w}\,q\cdot TR(d)\,\mathbf{w}\,pref(TR(S)))$ and $h(t)\upharpoonright(A\cup B) = t$.

(End of Proof)

Example 3.2.3

A k-bounded bag is specified by

$$\begin{aligned} <\{a0, a1, b0, b1\}, t:\ & 0 \leq l(t\restriction a0) - l(t\restriction b0) \leq k \\ & \wedge\ 0 \leq l(t\restriction a1) - l(t\restriction b1) \leq k \\ & \wedge\ 0 \leq l(t\restriction\{a0, a1\}) - l(t\restriction\{b0, b1\}) \leq k \\ > & \end{aligned}$$

For component bag_k that satisfies this specification, we have, according to the CW-rule

$$TR(bag_k) = SEM_k(a0, b0)\ \mathbf{w}\ SEM_k(a1, b1)\ \mathbf{w}\ SEM_k(\{a0, a1\}, \{b0, b1\})$$

A component for $SEM_k(a, b)$ is given by sem_k (Example 2.3.6).
A component for $SEM_k(\{a0, a1\}, \{b0, b1\})$ is given by $csem_k$ (Example 2.3.7).

Application of the Composition Rule yields a program with 3 subcomponents:

com $bag_k(a0, a1, b0, b1)$:
  sub $p, q : sem_k,\ r : csem_k$ **bus**
  $(p{\cdot}a, r{\cdot}a0 ; a0 \mid p{\cdot}b, r{\cdot}b0 ; b0 \mid q{\cdot}a, r{\cdot}a1 ; a1 \mid q{\cdot}b, r{\cdot}b1 ; b1)^*$
moc

From the text of $csem_k$ in Example 2.3.7 we infer that the following component satisfies the specification as well.

com $bag_k(a0, a1, b0, b1)$:
  sub $p, q, r : sem_k$ **bus**
  $(p{\cdot}a, r{\cdot}a ; a0 \mid p{\cdot}b, r{\cdot}b ; b0 \mid q{\cdot}a, r{\cdot}a ; a1 \mid q{\cdot}b, r{\cdot}b ; b1)^*$
moc

(End of Example)

Example 3.2.4

A *sorter* is specified by

$$\begin{aligned} <\{a0, a1, b0, b1\}, t:\ & 0 \leq l(t\restriction a0) - l(t\restriction b0) \\ & \wedge\ 0 \leq l(t\restriction a1) - l(t\restriction b1) \\ & \wedge\ (\mathbf{A}\ s : t = s b1 : l(s\restriction a0) = l(s\restriction b0)) \\ > & \end{aligned}$$

The last conjunct expresses that a one may be retrieved only if there is no zero to be retrieved.

In Example 2.4.4 we derived component *zsem*, that has specification:

$<\{a,b,z\}, t : 0 \leqslant l(t\restriction a) - l(t\restriction b) \wedge (\mathbf{A}\, s : t = sz : l(s\restriction a) - l(s\restriction b) = 0)>$

which is - apart from renaming - expressed by the first and the third conjunct of the specification of *sorter*.

The second conjunct specifies $SEM(a1, b1)$. This yields component *sorter* given by

com *sorter*$(a0, a1, b0, b1)$:
  sub p: *zsem*, q : *sem* **bus**
  $(p{\cdot}a ; a0 \mid p{\cdot}b ; b0 \mid p{\cdot}z, q{\cdot}b ; b1 \mid q{\cdot}a ; a1)^*$
moc

(End of Example)

Exercises

0. Derive component *nodup* specified by

   $<\{a,b,c\}, t : (\mathbf{A}\, u, v : t = uava \vee t = ubvb \vee t = ucvc : v \neq \epsilon)>$

1. Component *ebag* is an unbounded bag that has additional symbol *e* to denote the emptiness of the bag. Give a specification for *ebag* and derive a program text from that specification.

2. Construct components that have $\{a,b\}$ as their alphabet and that have as specification predicate:

   (i) $l(t\restriction a) = 0$
   (ii) $l(t\restriction a) = l(t\restriction b)$
   (iii) $l(t\restriction a) = 0 \vee l(t\restriction b) = 0$
   (iv) $l(t\restriction a) \leqslant l(t\restriction b) \leqslant 5$
   (v) $l(t\restriction a) \cdot l(t\restriction b) \leqslant 9$
   (vi) $l(t\restriction a) + l(t\restriction b) \leqslant 5$

3. Derive a program for a bounded sorter.

4. Component *c* has alphabet A, $A = \{a_0, \ldots, a_{m-1}\}$, and component *d* has alphabet B, $B = \{b_0, \ldots, b_{n-1}\}$.

Component cd is defined by

com $cd(A \cup B)$:
  sub $p : c, q : d$ **bus**
  $(p \cdot a_0; a_0 \mid \cdots \mid p \cdot a_{m-1}; a_{m-1})^*$
  $,(q \cdot b_0; b_0 \mid \cdots \mid q \cdot b_{n-1}; b_{n-1})^*$
moc

($A \cap B$ is not necessarily empty)

Prove that $TR(cd) = TR(c) \mathbf{w} TR(d)$. Compute $TR(cd)$ if the comma in the command is replaced by a bar. Compute $TR(cd)$ if the comma in the command is replaced by a semicolon.

(End of Exercises)

3.3 The Conjunction-Blend Rule

In this section we consider an analogue of the CW-rule for the blending operator. It will turn out that the analogue is too complicated to be useful. The main purpose of this section is to show the source of the complications. Since blending equals weaving followed by projection, we first consider the projection operator.

Property 3.3.0

Let $<A, P>$ be a specification of process T, and let B be an alphabet. Then

$$<A \cap B, t: (\mathbf{E}\, u : u \in A^* \wedge (\mathbf{A}\, v : v \leqslant u : P(v)) : t = u{\restriction}B)>$$

specifies $T{\restriction}B$.

Proof

$\mathbf{a}(T{\restriction}B) = A \cap B$, and for any t, $t \in (A \cap B)^*$, we have

$t \in \mathbf{t}T{\restriction}B$

$=$ { $T{\restriction}B$ is prefix-closed }

$(\mathbf{A}\, s : s \leqslant t : s \in \mathbf{t}T{\restriction}B)$

$=$ { definition of projection }

$(\mathbf{A}\, s : s \leqslant t : (\mathbf{E}\, u : u \in \mathbf{t}T : s = u{\restriction}B))$

$=$ { $<A, P>$ specifies T }

$$(\mathbf{A}\, s : s \leqslant t : (\mathbf{E}\, u : u \in A^* \wedge (\mathbf{A}\, v : v \leqslant u : P(v)) : s = u\restriction B))$$

(End of Proof)

The predicate $t : (\mathbf{A}\, s : s \leqslant t : (\mathbf{E}\, u : u \in A^* \wedge (\mathbf{A}\, v : v \leqslant u : P(v)) : s = u\restriction B))$ does not look very attractive. One might hope that the simpler one given by

$$t : (\mathbf{A}\, s : s \leqslant t : (\mathbf{E}\, u : u \in A^* \wedge P(u) : s = u\restriction B))$$

would describe $\mathbf{t}T\restriction B$ as well. Unfortunately, this is in general not true as the following example demonstrates.

Example 3.3.1

Process T is specified by $<\{a,b\},P>$ where

$$P(t) \equiv t = \epsilon \vee (\mathbf{E}\, w : w \in \{a,b\}^* : t = wa).$$

Then $T = <\{a,b\},\{a\}^*>$ and $T\restriction b = <\{b\},\{\epsilon\}>$

For any n, $n \geqslant 0$, we have $b^n = (b^n a)\restriction b$. Hence,
$<\{b\}, t : (\mathbf{E}\, u : u \in \{a,b\}^* \wedge P(u) : t = u\restriction b)>$ specifies $<\{b\},\{b\}^*>$.

We conclude that the specifications

$$<\{b\}, t : (\mathbf{E}\, u : u \in \{a,b\}^* \wedge (\mathbf{A}\, v : v \leqslant u : P(v)) : t = u\restriction B)> \text{ and}$$
$$<\{b\}, t : (\mathbf{E}\, u : u \in \{a,b\}^* \wedge P(u) : t = u\restriction B)>$$

specify different processes.

(End of Example)

Combining Property 3.3.0 and the CW-rule yields

Theorem 3.3.2 (CB-rule)

Let $<A,P>$ and $<B,Q>$ be specifications of T and U respectively. Then

$<A \div B, t : (\mathbf{E}\, u : u \in (A \cup B)^* \wedge (\mathbf{A}\, v : v \leqslant u : P(v\restriction A) \wedge Q(v\restriction B)) : t = u\restriction(A \div B))>$

specifies $T\, \mathbf{b}\, U$.

(End of Theorem)

The CB-rule is not useful when deriving programs from a specification. It shows how difficult the relation between components and their blend can be. Moreover, it is the projection that should be blamed for it.

Exercises

0. Determine a specification of $SEM_2(a,b)$ using the CB-rule and
$SEM_2(a,b) = SEM_1(a,c)\ \mathbf{b}\ SEM_1(c,b)$.

1. For symbols $a0, a1, b0, b1$ and for integer $k, k \geqslant 1$, process $BAG_k(a0, a1, b0, b1)$ has alphabet $\{a0, a1, b0, b1\}$ and trace set

$$\{t \mid t \in \{a0,a1,b0,b1\}^* \wedge (\mathbf{A}\, s : s \leqslant t :\ 0 \leqslant l(s\restriction a0) - l(s\restriction b0) \leqslant k$$
$$\wedge\ 0 \leqslant l(s\restriction a1) - l(s\restriction b1) \leqslant k$$
$$\wedge\ 0 \leqslant l(s\restriction\{a0,a1\}) - l(s\restriction\{b0,b1\}) \leqslant k\ )\}$$

Prove that for all k and $n, k \geqslant 1 \wedge n \geqslant 1$:

$BAG_k(a0,a1,b0,b1)\ \mathbf{b}\ BAG_n(b0,b1,c0,c1) \neq BAG_{k+n}(a0,a1,c0,c1)$

(End of Exercises)

3.4 Context-free grammars

A trace structure T may be viewed as a language $\mathbf{t}T$ over alphabet $\mathbf{a}T$, cf. [6]. One may wonder what kind of languages are generated by components. Some research on this topic can be found in [11]. In this section we show how a component can be constructed whose trace set corresponds to a language given by a context-free grammar. We first give an informal introduction to context-free grammars. For a detailed treatment we recommend [6].

A context-free grammar G is a quadruple, $G = \langle A, N, S, P \rangle$, where

A is an alphabet, the set of *terminals*,

N is an alphabet, the set of *non-terminals*, $A \cap N = \varnothing$,

S is an element of N, the *start symbol*,

P is a finite subset of $N \times (A \cup N)^*$, the set of *production rules*

The relation $\rightarrow$ on $(A \cup N)^*$ is defined by $(\alpha, \beta, \nu \in (A \cup N)^*$ and $X \in N)$

$\alpha X \beta \rightarrow \alpha \nu \beta$ if $(X, \nu) \in P$

The k-fold composition of $\rightarrow$ is denoted by $\xrightarrow{k}$.
The reflexive and transitive closure of $\rightarrow$ is denoted by $\xrightarrow{*}$.

The language of grammar G, denoted by $L(G)$, is defined by

$$L(G) = \{v \mid v \in A^* \wedge S \xrightarrow{*} v\}$$

Informally, a trace $t \in A^*$ is an element of $L(G)$ if it can be obtained by means of a systematic rewriting process on elements of $(A \cup N)^*$ that begins with start symbol S and in which repeatedly a left-hand part of a production rule is replaced by a right-hand part until no non-terminal remains.

Let G, $G = \langle A, N, S, P \rangle$, be a context-free grammar.

In general, $L(G)$ is not prefix-closed.
We construct component g with alphabet $A \cup \{\surd\}$, where $\surd$ (pronounced as 'tick') is a fresh symbol, and

$$t \in L(G) \equiv t\surd \in \mathbf{t}TR(g)$$

Since $L(G) = \{t \mid t \in (A \cup N)^* \wedge S \xrightarrow{*} t\} \cap A^*$ we first construct component h, which has trace structure

$$\langle A \cup N \cup \{\surd\}, pref(\{t \mid (\mathbf{E}\, u : u \in (A \cup N)^* \wedge S \xrightarrow{*} u : t = u\surd)\}) \rangle$$

Then g with $\mathbf{a}TR(g) = A \cup \{\surd\}$ and $\mathbf{t}TR(g) = \mathbf{t}TR(h) \cap (A \cup \{\surd\})^*$ satisfies our requirements.
Since the intersection of processes with equal alphabets is equal to the weave of these processes, we have

$$TR(g) = (TR(h)\,\mathbf{w}\,\langle A \cup N \cup \{\surd\}, (A \cup \{\surd\})^* \rangle) \restriction (A \cup \{\surd\})$$

The projection on $A \cup \{\surd\}$ is needed to get rid of the non-terminals in the alphabet of the weave. This projection can be obtained (Property 1.4.2.3) by a blend :

$$TR(g) = (TR(h)\,\mathbf{w}\,\langle A \cup N \cup \{\surd\}, (A \cup \{\surd\})^* \rangle)\,\mathbf{b}\,RUN(N)$$

Let $A = \{a_0, \ldots, a_{m-1}\}$, $N = \{X_0, \ldots, X_{n-1}\}$, and $P = \{P_0, \ldots, P_{k-1}\}$.

Component h is given by

com $h(A \cup N \cup \{\surd\})$:
  sub $p : h$ **bus**
  $S; \surd \mid (C_0 \mid \cdots \mid C_{k-1} \mid D_0 \mid \cdots \mid D_{m-1} \mid E_0 \mid \cdots \mid E_{n-1})^*; p \cdot \surd; \surd$
moc

where S is the start symbol of G,
C_i, $0 \leqslant i < k$, corresponds to P_i : if $P_i = (X, \alpha_0 \cdots \alpha_{r-1})$ then
$C_i = p \cdot X : \alpha_0; \cdots ; \alpha_{r-1}$,
$D_i = p \cdot a_i ; a_i$ for $0 \leqslant i < m$,
$E_i = p \cdot X_i ; X_i$ for $0 \leqslant i < n$.

Command C_i corresponds to the application of production P_i.
Command D_i corresponds to copying terminal a_i.
Command E_i corresponds to copying nonterminal X_i.

Let f denote the function associated with component h. Then
$\mathbf{t} f(STOP(A \cup N \cup \{\surd\})) = \{\epsilon, S, S\surd\}$, and for k, $k \geqslant 0$,

$pref(\{t \mid (\mathbf{E}\, u : u \in (A \cup N)^* \wedge S \xrightarrow{k} u : t = u\surd)\}) \subseteq \mathbf{t} f^{k+1}(STOP(A \cup N \cup \{\surd\}))$ and

$\mathbf{t} f^{k+1}(STOP(A \cup N \cup \{\surd\})) \subseteq pref(\{t \mid (\mathbf{E}\, u : u \in (A \cup N)^* \wedge S \xrightarrow{*} u : t = u\surd)\})$

Hence,

$$TR(h) = <A \cup N \cup \{\surd\}, pref(\{t \mid (\mathbf{E}\, u : u \in (A \cup N)^* \wedge S \xrightarrow{*} u : t = u\surd)\})>$$

We now have to realize components c and d such that

$TR(c) = <A \cup N \cup \{\surd\}, (A \cup \{\surd\})^*>$ and $TR(d) = RUN(N)$.

Component d is defined by

com $d(N)$: $(X_0 \mid \cdots \mid X_{n-1})^*$ **moc**

Component c is defined by

com $c(A \cup N \cup \{\surd\})$:
$(a_0 \mid \cdots \mid a_{m-1} \mid \surd)^*$
$\mid (X_0; X_0), X_0, \cdots, (X_{n-1}; X_{n-1}), X_{n-1}$
moc

The part $(X_0; X_0), X_0, \cdots, (X_{n-1}; X_{n-1}), X_{n-1}$ has been added to include N in the alphabet of c. (It may be omitted if the requirement 'the alphabet of the command equals the alphabet of the component' is weakened to 'the alphabet of the command is a subset of the alphabet of the component')

We now have components h, c, and d, such that

$$TR(g) = (TR(h)\ \mathbf{w}\ TR(c))\ \mathbf{b}\ TR(d)$$

Using the Composition Rule one can easily construct g.

Example 3.4.0

Let $G = \langle \{a,b\}, \{S\}, S, \{S \to a,\ S \to bSS\} \rangle$. Then h is given by

com $h(a,b,S,\surd)$:
  sub $p : h$ **bus**
  $S ; \surd \mid (p{\cdot}S ; a \mid p{\cdot}S ; b ; S ; S \mid p{\cdot}a ; a \mid p{\cdot}b ; b \mid p{\cdot}S ; S)^* ; p{\cdot}\surd ; \surd$
moc

Components c and d are given by

com $c(a,b,\surd,S)$: $(a \mid b \mid \surd)^* \mid (S ; S), S$ **moc**
com $d(S)$: S^* **moc**

Component hc is defined as

com $hc(a,b,\surd,S)$:
  sub $p : h,\ q : c$ **bus**
  $(p{\cdot}a, q{\cdot}a ; a \mid p{\cdot}b, q{\cdot}b ; b \mid p{\cdot}\surd, q{\cdot}\surd ; \surd \mid p{\cdot}S, q{\cdot}S ; S)^*$
moc

According to the Composition Rule we have $TR(hc) = TR(h)\ \mathbf{w}\ TR(c)$.

The ultimate component g is given by

com $g(a,b,\surd)$:
  sub $p : hc,\ q : d$ **bus**
  $[p{\cdot}a = a,\ p{\cdot}b = b,\ p{\cdot}\surd = \surd,\ p{\cdot}S = q{\cdot}S]$
moc

(End of Example)

Exercises

0. Let $G = \langle \{a, b\}, \{S\}, S, \{S \rightarrow a, S \rightarrow bSS\} \rangle$, cf. Example 3.4.0 . Consider component g defined by

com $g(a, b, \surd)$:
  sub $p : g$ **bus**
    $a ; \surd$
  $\mid (p{\cdot}a ; a \mid p{\cdot}a ; b ; a ; a \mid p{\cdot}b ; b)^* ; p{\cdot}\surd ; \surd$
moc

Show that g corresponds to $L(G)$.

1. An unbounded stack of binary values can be specified by grammar G with

alphabet : $\{a0, a1, b0, b1\}$;
N : $\{S\}$;
P : $\{S \rightarrow \epsilon, S \rightarrow a0\ S\ b0\ S, S \rightarrow a1\ S\ b1\ S\}$;
start symbol : S

Derive a component that corresponds to $L(G)$.

(End of Exercises)

4 Deadlock

4.0 Introduction

Deadlock is a well-known phenomenon in the domain of concurrent processes, cf. [2]. It is usually explained in terms of shared resources. We illustrate deadlock by the following example.

Component $c0$ has alphabet $\{a0, b0, p0, q0, e0, f0\}$. The meaning of the symbols is as follows.

$a0$: acquire resource A

$b0$: release resource A

$p0$: acquire resource P

$q0$: release resource P

$e0$: initiation of a computation using resources A and P

$f0$: termination of the computation initiated by $e0$

Component $c1$ has alphabet $\{a1, b1, p1, q1, e1, f1\}$. The symbols have the same meaning as the corresponding symbols of component $c0$.
Furthermore, we have components exA and exP that guarantee mutual exclusion in the use of A and P respectively. The components are given by

com $c0(a0, b0, p0, q0, e0, f0)$: $(a0 ; p0 ; e0 ; f0 ; b0 ; q0)^*$ **moc**

com $c1(a1, b1, p1, q1, e1, f1)$: $(p1 ; a1 ; e1 ; f1 ; q1 ; b1)^*$ **moc**

com $exA(a0, a1, b0, b1)$: $(a0 ; b0 \mid a1 ; b1)^*$ **moc**

com $exP(p0, p1, q0, q1)$: $(p0 ; q0 \mid p1 ; q1)^*$ **moc**

We consider the composite U of these components:

$$U = TR(c0) \mathbf{w} TR(c1) \mathbf{w} TR(exA) \mathbf{w} TR(exP)$$

Typical traces of $\mathbf{t}U$ are

$a0\ p0\ e0\ f0\ b0\ q0\ p1\ a1\ e1\ f1\ q1\ b1$

$a0\ p0\ e0\ f0\ b0\ q0\ a0\ p0\ e0$

$a0\ p1$

The last one, $a0p1$, has no extension in $\mathbf{t}U$: from the command of $c0$ we infer that $p0$ is the only candidate of the set $\{a0, b0, p0, q0, e0, f0\}$ and extension with $p0$ is not in accordance with exP. A similar argument shows that none of the elements of $\{a1, b1, p1, q1, e1, f1\}$ are possible as an extension of $a0p1$.
For each component, however, the projection of $a0p1$ on the alphabet of that component may be extended (with respect to that component). Phrased differently : the composite has terminated whereas none of the subcomponents have terminated. We say that the system is in a deadlock.

In the next sections we give a formalization of deadlock and we derive properties thereof.

4.1 Lock

In Section 1.2 we discussed a mechanistic appreciation of processes. For a set X of processes we have

t is the trace thus far generated with respect to $(\mathbf{W}\, T : T \in X : T)$

$\equiv$

$(\mathbf{A}\, T : T \in X : t \restriction \mathbf{a}T$ is the trace thus far generated with respect to T)

For process T and trace t, $t \in \mathbf{t}T$, we define the *successor set* of t with respect to T, denoted by $S(t, T)$, by

$$S(t, T) = \{a \mid a \in \mathbf{a}T \wedge ta \in \mathbf{t}T\}$$

Let T be a process and let t be the trace thus far generated with respect to the mechanism described by T. If $S(t, T) = \varnothing$, we say that the mechanism has terminated. If $S(t, T) \neq \varnothing$, the mechanism eventually gets involved in a next event thereby extending t with the symbol associated with that event.

We call T a *non-terminating* process if

$$(\mathbf{A}\, t : t \in \mathbf{t}T : S(t, T) \neq \varnothing)$$

Notice that the negation of non-terminating is 'may terminate'.

Property 4.1.0

Let T be a process and let s and t be elements of $\mathbf{t}T$. Then

$$s \sim t \Rightarrow S(s, T) = S(t, T)$$

(End of Property)

Due to the last property we may extend the notion of successor set from traces to states. Then $S([t], T) = S(t, T)$ for all t, $t \in \mathbf{t}T$.

Property 4.1.1

Let T be a process. T is a non-terminating process if and only if each node of the state graph of T has an outgoing arc.

(End of Property)

In the sequel X is a set of processes and $U = (\mathbf{W}\ T : T \in X : T)$.

Property 4.1.2

$$(\mathbf{A}\ T : T \in X : S(t{\restriction}\mathbf{a}T, T) = \varnothing) \;\Rightarrow\; S(t, U) = \varnothing$$

Proof

We derive

$S(t, U) \neq \varnothing$

$=$ { definition of successor set }

$(\mathbf{E}\ a : a \in \mathbf{a}U : ta \in \mathbf{t}U)$

$=$ { predicate calculus, $\mathbf{a}U = (\cup\ T : T \in X : \mathbf{a}T)$ }

$(\mathbf{E}\ T : T \in X : (\mathbf{E}\ a : a \in \mathbf{a}T : ta \in \mathbf{t}U))$

$\Rightarrow$ { $U = (\mathbf{W}\ T : T \in X : T)$ }

$(\mathbf{E}\ T : T \in X : (\mathbf{E}\ a : a \in \mathbf{a}T : ta{\restriction}\mathbf{a}T \in \mathbf{t}T))$

$=$ { definition of projection }

$(\mathbf{E}\ T : T \in X : (\mathbf{E}\ a : a \in \mathbf{a}T : (t{\restriction}\mathbf{a}T)a \in \mathbf{t}T))$

$=$ { definition of successor set }

$(\mathbf{E}\ T : T \in X : S(t{\restriction}\mathbf{a}T, T) \neq \varnothing)$

(End of Proof)

For t, $t \in \mathbf{t}U$, we define $lock(t, X)$ by

$$lock(t, X) \equiv S(t, U) = \varnothing \wedge (\mathbf{E}\ T : T \in X : S(t{\restriction}\mathbf{a}T, T) \neq \varnothing)$$

$lockfree(X)$ is defined by

$$lockfree(X) \equiv (\mathbf{A}\ t : t \in \mathbf{t}U : \neg lock(t, X))$$

If $\neg lockfree(X)$ holds, we say that X has danger of lock.

Property 4.1.3

0 $lockfree(\varnothing)$

1 $lockfree(\{T\})$ for any process T

(End of Property)

Property 4.1.4

$lockfree(X) \equiv (\mathbf{A}\, t : t \in \mathbf{t}U : S(t, U) = \varnothing \equiv (\mathbf{A}\, T : T \in X : S(t \restriction \mathbf{a}T, T) = \varnothing))$

Proof

We derive

$$
\begin{array}{cl}
 & lockfree(X) \\
= & \{ \text{ definition of } lockfree \} \\
 & (\mathbf{A}\, t : t \in \mathbf{t}U : \neg lock(t, X)) \\
= & \{ \text{ definition of } lock \text{, predicate calculus } \} \\
 & (\mathbf{A}\, t : t \in \mathbf{t}U : S(t, U) \neq \varnothing \vee (\mathbf{A}\, T : T \in X : S(t \restriction \mathbf{a}T, T) = \varnothing)) \\
= & \{ \text{ predicate calculus } \} \\
 & (\mathbf{A}\, t : t \in \mathbf{t}U : S(t, U) = \varnothing \Rightarrow (\mathbf{A}\, T : T \in X : S(t \restriction \mathbf{a}T, T) = \varnothing)) \\
= & \{ \text{ Property 4.1.2 } \} \\
 & (\mathbf{A}\, t : t \in \mathbf{t}U : S(t, U) = \varnothing \equiv (\mathbf{A}\, T : T \in X : S(t \restriction \mathbf{a}T, T) = \varnothing))
\end{array}
$$

(End of Proof)

Property 4.1.4 may be phrased as
'The composite of a set of mechanisms has no danger of lock' and
'The composite has terminated if and only if all composing parts have terminated'
are equivalent.

Theorem 4.1.5

Let X be a set of processes and let $U = (\mathbf{W}\, T : T \in X : T)$.
Let for $V, V \in X$, $\hat{V}$ denotes the process $(\mathbf{W}\, T : T \in X \wedge T \neq V : T)$.
Then

$$lockfree(X) \equiv (\mathbf{A}\, T : T \in X : lockfree(\{T, \hat{T}\}))$$

Proof

For any $T, T \in X$, and $t, t \in \mathbf{t}U$, we have

$(\mathbf{A}\, V : V \in X : S(t{\restriction}\mathbf{a}V, V) = \varnothing)$

$\Rightarrow$ { predicate calculus }

$(\mathbf{A}\, V : V \in X \ \wedge\ V \neq T : S(t{\restriction}\mathbf{a}V, V) = \varnothing)$

$\Rightarrow$ { Property 4.1.2 with U replaced by $\hat{T}$ and X replaced by $X \backslash \{T\}$ }

$S(t{\restriction}\mathbf{a}\hat{T}, \hat{T}) = \varnothing$

Hence,

$$(\mathbf{A}\, T : T \in X : S(t{\restriction}\mathbf{a}T, T) = \varnothing) \Rightarrow (\mathbf{A}\, T : T \in X : S(t{\restriction}\mathbf{a}\hat{T}, \hat{T}) = \varnothing) \qquad (*)$$

We derive

$(\mathbf{A}\, T : T \in X : \mathit{lockfree}(\{T, \hat{T}\}))$

$=$ { Property 4.1.4 applied to $\{T, \hat{T}\}$ }

$(\mathbf{A}\, T : T \in X : (\mathbf{A}\, t : t \in \mathbf{t}U : S(t, U) = \varnothing \ \equiv\ S(t{\restriction}\mathbf{a}T, T) = \varnothing \wedge S(t{\restriction}\mathbf{a}\hat{T}, \hat{T}) = \varnothing))$

$=$ { Property 4.1.2 }

$(\mathbf{A}\, T : T \in X : (\mathbf{A}\, t : t \in \mathbf{t}U : S(t, U) = \varnothing \ \Rightarrow\ S(t{\restriction}\mathbf{a}T, T) = \varnothing \wedge S(t{\restriction}\mathbf{a}\hat{T}, \hat{T}) = \varnothing))$

$=$ { predicate calculus }

$(\mathbf{A}\, t : t \in \mathbf{t}U : S(t, U) = \varnothing \ \Rightarrow\ (\mathbf{A}\, T : T \in X : S(t{\restriction}\mathbf{a}T, T) = \varnothing)$
$\qquad \wedge\ (\mathbf{A}\, T : T \in X : S(t{\restriction}\mathbf{a}\hat{T}, \hat{T}) = \varnothing))$

$=$ { (*) }

$(\mathbf{A}\, t : t \in \mathbf{t}U : S(t, U) = \varnothing \ \Rightarrow\ (\mathbf{A}\, T : T \in X : S(t{\restriction}\mathbf{a}T, T) = \varnothing))$

$=$ { Property 4.1.2 }

$(\mathbf{A}\, t : t \in \mathbf{t}U : S(t, U) = \varnothing \ \equiv\ (\mathbf{A}\, T : T \in X : S(t{\restriction}\mathbf{a}T, T) = \varnothing))$

$=$ { Property 4.1.4 }

$\mathit{lockfree}(X)$

(End of Proof)

A consequence of Theorem 4.1.5 is :

> a system that has danger of lock with respect to its components can always be cut into two parts such that the system has danger of lock with respect to these two parts.

Example 4.1.6

Consider components $c0$, $c1$, exA, and exP that were introduced in Section 4.0 :

$\mathbf{com}\ c0(a0, b0, p0, q0, e0, f0):\ (a0 ; p0 ; e0 ; f0 ; b0 ; q0)^{*}\ \mathbf{moc}$

com $c1(a1,b1,p1,q1,e1,f1)$: $(p1;a1;e1;f1;q1;b1)^*$ **moc**

com $exA(a0,a1,b0,b1)$: $(a0;b0 \mid a1;b1)^*$ **moc**

com $exP(p0,p1,q0,q1)$: $(p0;q0 \mid p1;q1)^*$ **moc**

Let $X = \{TR(c0), TR(c1), TR(exA), TR(exP)\}$ and let $U = (\mathbf{W}\ T : T \in X : T)$.
Then $a0p1 \in \mathbf{t}U$, $S(a0p1, U) = \varnothing$ and $S(a0p1 \restriction \mathbf{a}TR(c0), TR(c0)) \neq \varnothing$.
Hence, $lock(a0p1, X)$ and also
$lock(a0p1, \{TR(c0), TR(c1)\ \mathbf{w}\ TR(exA)\ \mathbf{w}\ TR(exP)\})$.

Notice, that $lockfree(\{TR(c1), TR(exA), TR(exP)\})$.

(End of Example)

The next theorem shows how larger lockfree systems can be built from smaller ones.

Theorem 4.1.7

Let X and Y be sets of processes. Let $U = (\mathbf{W}\ T : T \in X : T)$ and let $V = (\mathbf{W}\ T : T \in Y : T)$. Then

$$lockfree(X) \wedge lockfree(Y) \wedge lockfree(\{U, V\}) \Rightarrow lockfree(X \cup Y)$$

Proof

Assume $lockfree(X) \wedge lockfree(Y) \wedge lockfree(\{U, V\})$. For any t, $t \in \mathbf{t}(U\ \mathbf{w}\ V)$, we derive

$$
\begin{array}{ll}
 & S(t, U\ \mathbf{w}\ V) = \varnothing \\
= & \quad \{\text{ Property 4.1.4, } lockfree(\{U, V\}) \} \\
 & S(t \restriction \mathbf{a}U, U) = \varnothing \wedge S(t \restriction \mathbf{a}V, V) = \varnothing \\
= & \quad \{\text{ Property 4.1.4, } lockfree(X) \text{ and } lockfree(Y) \} \\
 & (\mathbf{A}\ T : T \in X : S(t \restriction \mathbf{a}U \restriction \mathbf{a}T, T) = \varnothing) \wedge (\mathbf{A}\ T : T \in Y : S(t \restriction \mathbf{a}V \restriction \mathbf{a}T, T) = \varnothing) \\
= & \quad \{ T \in X \text{ implies } \mathbf{a}T \subseteq \mathbf{a}U, T \in Y \text{ implies } \mathbf{a}T \subseteq \mathbf{a}V \} \\
 & (\mathbf{A}\ T : T \in X : S(t \restriction \mathbf{a}T, T) = \varnothing) \wedge (\mathbf{A}\ T : T \in Y : S(t \restriction \mathbf{a}T, T) = \varnothing) \\
= & \quad \{\text{ predicate calculus }\} \\
 & (\mathbf{A}\ T : T \in X \cup Y : S(t \restriction \mathbf{a}T, T) = \varnothing)
\end{array}
$$

Application of Property 4.1.4 yields $lockfree(X \cup Y)$

(End of Proof)

In general, the converse of Theorem 4.1.7 does not hold, as the following example shows.

Example 4.1.8

Components $c0$, $c1$ and $c2$ are defined by

com $c0(a,b)$: $(a^* \mid b^*)$ **moc**

com $c1(a,b)$: $(a^* \mid b\ )$ **moc**

com $c2(a,b)$: $a^* \mid b,(b;b)$ **moc** $\quad (TR(c2) = <\{a,b\},\{a\}^*>)$

Then $TR(c0) \mathbf{w} TR(c1) \mathbf{w} TR(c2) = <\{a,b\},\{a\}^*>$. For all t, $t \in \{a\}^*$, we have

$S(t, <\{a,b\},\{a\}^*>) = \{a\} \neq \varnothing$.

Hence, $lockfree(\{TR(c0), TR(c1), TR(c2)\})$.

We have $b \in \mathbf{t}(TR(c0) \mathbf{w} TR(c1))$, $S(b, TR(c0) \mathbf{w} TR(c1)) = \varnothing$,
and $S(b, TR(c0)) = \{b\} \neq \varnothing$. Hence, $\neg lockfree(\{TR(c0), TR(c1)\})$.

We conclude

$lockfree(\{TR(c0), TR(c1), TR(c2)\}) \ \wedge\ \neg lockfree(\{TR(c0), TR(c1)\})$

(End of Example)

Most mechanisms, such as bags, queues, and stacks, correspond to non-terminating processes. In general, we are not interested in mechanisms that may terminate. Notice that for $U = (\mathbf{W}\ T : T \in X : T)$, we have

U is non-terminating $\Rightarrow$ $lockfree(X)$

In the next section we define deadlock. In general, the implication above does not hold for deadlock.

Exercises

0. T and U are non-terminating processes such that $\mathbf{a}T \cap \mathbf{a}U$ contains at most one element. Prove that $T \mathbf{w} U$ is also non-terminating.

1. T is a process. Prove $lockfree(\{T, STOP\})$.

2. Prove that for any set X of processes:

$$lockfree(X)$$
$$\equiv \quad (\mathbf{A}\, Y : Y \subseteq X : lockfree(\{(\mathbf{W}\, T : T \in Y : T), (\mathbf{W}\, T : T \in X \backslash Y : T)\}))$$

3. T is a process and a is a symbol. Prove

$$lockfree(\{T, STOP(a)\}) \quad \equiv \quad (\mathbf{A}\, t : t \in \mathbf{t}T : S(t, T) \neq \{a\})$$

4. Components $c0$, $c1$, exA, and exP are defined as in Example 4.1.6 . Let $T = TR(c0)\, \mathbf{b}\, TR(exA)\, \mathbf{b}\, TR(exP)$. Determine the state graph of T. Determine also the state graph of $T\, \mathbf{b}\, TR(c1)$. What are your conclusions ?

5. Component ex is defined by

   com $ex(a, b, c)$:
   sub $p, q : sem_1$ **bus**
   $(p{\cdot}a ; a \mid p{\cdot}b, q{\cdot}a ; b \mid q{\cdot}b ; c)^*$
   moc

   Show that $TR(ex) = SEM_1(a, b)\, \mathbf{w}\, SEM_1(b, c)$. Let S denote the command of ex.

   Determine $lockfree(\{p{\cdot}SEM_1(a, b), q{\cdot}SEM_1(b, c), pref(TR(S))\})$

(End of Exercises)

4.2 Deadlock

In Section 4.0 we considered components $c0$, $c1$, exA, and exP, defined by

com $c0(a0, b0, p0, q0, e0, f0)$: $(a0 ; p0 ; e0 ; f0 ; b0 ; q0)^*$ **moc**
com $c1(a1, b1, p1, q1, e1, f1)$: $(p1 ; a1 ; e1 ; f1 ; q1 ; b1)^*$ **moc**
com $exA(a0, a1, b0, b1)$: $(a0 ; b0 \mid a1 ; b1)^*$ **moc**
com $exP(p0, p1, q0, q1)$: $(p0 ; q0 \mid p1 ; q1)^*$ **moc**

Let $X = \{TR(c0), TR(c1), TR(exA), TR(exP)\}$ and let $U = (\mathbf{W}\, T : T \in X : T)$. Then $\neg lockfree(X)$.

Let y be a symbol. We add $RUN(y)$ to these processes. For any t, $t \in \mathbf{t}(U \mathbf{w} RUN(y))$, we have $ty \in \mathbf{t}(U \mathbf{w} RUN(y))$. Hence, $U \mathbf{w} RUN(y)$ is a non-terminating process, from which we infer

$$lockfree(X \cup \{RUN(y)\})$$

Nevertheless, the mechanism described by the weave of these five processes has danger of deadlock in the usual sense of the word. These observations lead to the following definition.

For a set X of processes $deadlockfree(X)$ is defined by

$$deadlockfree(X) \equiv (\mathbf{A}\, Y : Y \subseteq X : lockfree(Y))$$

If $\neg deadlockfree(X)$ holds, we say that X has danger of deadlock.

Property 4.2.0

$$deadlockfree(X) \equiv (\mathbf{A}\, Y : Y \subseteq X : deadlockfree(Y))$$

(End of Property)

Property 4.2.1

For processes T and U we have

0 $\quad deadlockfree(\{T, U\}) \equiv lockfree(\{T, U\})$

1 $\quad T \mathbf{w} U$ is non-terminating $\Rightarrow deadlockfree(\{T, U\})$

(End of Property)

Property 4.2.2

Let T and U be non-terminating processes such that $\mathbf{a}T \cap \mathbf{a}U$ contains at most one element. Then $deadlockfree(\{T, U\})$.

Proof

Let $t \in \mathbf{t}(T \mathbf{w} U)$, then $t \restriction \mathbf{a}T \in \mathbf{t}T$ and $t \restriction \mathbf{a}U \in \mathbf{t}U$. Since T and U are non-terminating, we can choose a, $a \in \mathbf{a}T$, and b, $b \in \mathbf{a}U$, such that $(t \restriction \mathbf{a}T)a \in \mathbf{t}T$ and $(t \restriction \mathbf{a}U)b \in \mathbf{t}U$. Since $\mathbf{a}T \cap \mathbf{a}U$ contains at most one element, we have three cases.

(i) $a = b$

(ii) $a \notin \mathbf{a}U$

(iii) $b \notin \mathbf{a}T$

In case (i) we have $ta \in \mathbf{t}(T \mathbf{\ w\ } U)$, in case (ii) we have $ta \in \mathbf{t}(T \mathbf{\ w\ } U)$, and in case (iii) we have $tb \in \mathbf{t}(T \mathbf{\ w\ } U)$.

Hence, $T \mathbf{\ w\ } U$ is non-terminating.

Application of Property 4.2.1.1 yields *deadlockfree* $(\{T, U\})$.

(End of Proof)

We do not have an analogue of Theorem 4.1.5 that holds for *deadlockfree*. If X has n elements, then $P(X)$ has 2^n elements, and to assure *deadlockfree* (X) requires in the worst case 2^n investigations. Notice that *lockfree* (X) requires only n investigations.
A similar conclusion holds for Theorem 4.1.7 .

The best we can prove is the following.

Theorem 4.2.3

Let X be a set of processes such that *deadlockfree* (X) holds, and let V be a process. Then

$$(\mathbf{A}\, Y : Y \subseteq X : \mathit{lockfree}(\{V, (\mathbf{W}\, T : T \in Y : T)\})) \Rightarrow \mathit{deadlockfree}(X \cup \{V\})$$

Proof

We derive

$$
\begin{array}{ll}
 & (\mathbf{A}\, Y : Y \subseteq X : \mathit{lockfree}(\{V, (\mathbf{W}\, T : T \in Y : T)\})) \\
= & \quad \{ \text{Property 4.1.3.1} \} \\
 & (\mathbf{A}\, Y : Y \subseteq X : \mathit{lockfree}(\{V\}) \wedge \mathit{lockfree}(\{V, (\mathbf{W}\, T : T \in Y : T)\})) \\
= & \quad \{ \mathit{deadlockfree}(X) \} \\
 & (\mathbf{A}\, Y : Y \subseteq X : \mathit{lockfree}(Y) \wedge \mathit{lockfree}(\{V\}) \wedge \mathit{lockfree}(\{V, (\mathbf{W}\, T : T \in Y : T)\})) \\
\Rightarrow & \quad \{ \text{Theorem 4.1.7} \} \\
 & (\mathbf{A}\, Y : Y \subseteq X : \mathit{lockfree}(Y \cup \{V\})) \\
= & \quad \{ \mathit{deadlockfree}(X) \} \\
 & (\mathbf{A}\, Y : Y \subseteq X \cup \{V\} : \mathit{lockfree}(Y)) \\
= & \quad \{ \text{definition of } \mathit{deadlockfree} \} \\
 & \mathit{deadlockfree}(X \cup \{V\})
\end{array}
$$

(End of Proof)

An even more serious problem is the following. We consider again components $c0$, $c1$, exA, and exP.

Figure 4.0 shows the state graph of $TR(c0)\,\mathbf{w}\,TR(exA)\,\mathbf{w}\,TR(exP)$. Projection on $\{e0, f0, a1, b1, p1, q1\}$ yields the blend, the state graph of which is shown in Figure 4.1 .

Apparently, we have $deadlockfree(\{TR(c0)\,\mathbf{b}\,TR(exA)\,\mathbf{b}\,TR(exP), TR(c1)\})$. The state graph of the blend of the four processes is shown in Figure 4.2 . Evidently, all information about deadlock has disappeared.

From Figure 4.2 one concludes that initially $e0$ is possible. However, internal events $a0$ and $p1$ bring the system to a grinding halt (as explained in Section 4.0).

It looks as if we have lost our hierarchical way of composing. It seems that, in order to avoid deadlock, one has to keep track of the internal structure of the components.

In Chapter 5, we cope with problems like these. We give conditions under which the situation described above does not occur.

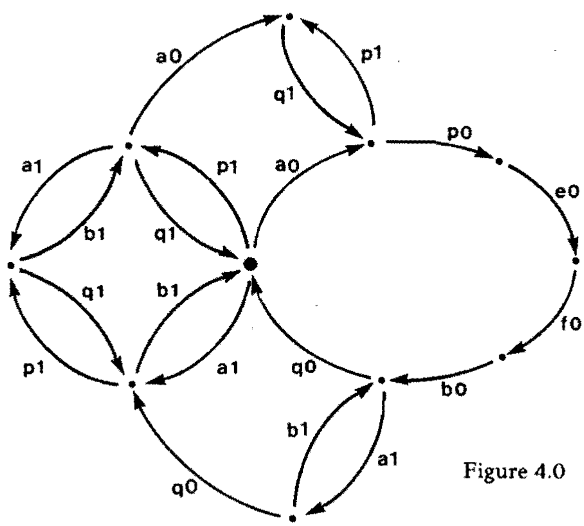


Figure 4.0

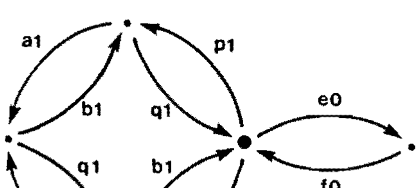


Figure 4.1

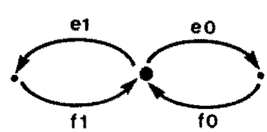


Figure 4.2

Finally we define deadlockfree for components. Let component c be defined by

> **com** $c(A)$:
> **sub** $p_0 : c_0, \ldots, p_{n-1} : c_{n-1}$ **bus**
> $[x_0 = y_0, \ldots, x_{m-1} = y_{m-1}]$
> S
> **moc**

Then $TR(c) = T \lceil A$ where

$$T = (\mathbf{W}\ i : 0 \leq i < n : (p_i \cdot TR(c_i))^{x_0, \ldots, x_{m-1}}_{y_0, \ldots, y_{m-1}})\ \mathbf{w}\ pref(TR(S))$$

Let X be the set consisting of all $(p_i \cdot TR(c_i))^{x_0, \ldots, x_{m-1}}_{y_0, \ldots, y_{m-1}}$, $0 \leq i < n$, and $pref(TR(S))$.
We call c *deadlockfree* if $deadlockfree(X)$ holds.

Exercises

0. Determine which of the following sets of processes are deadlockfree.
   (i) $\{SEM_1(a,b), SEM_1(b,c), SEM_1(c,d)\}$
   (ii) $\{SEM_1(a,b), SEM_1(b,c), SEM_1(c,a)\}$
   (iii) $\{SYNC_{1,1}(a,b), SEM_2(a,b), STOP\}$
   (iv) $\{STOP(a), STOP(b), RUN(a,c)\}$

1. Provide counterexamples for the following conjectures.
   (i) $deadlockfree(X) \wedge deadlockfree(\{V, (\mathbf{W}\ T : T \in X : T)\})$
   $\quad \Rightarrow deadlockfree(X \cup \{V\})$
   (ii) $deadlockfree(X) \wedge (\mathbf{A}\ T : T \in X : deadlockfree(\{T, V\}))$
   $\quad \Rightarrow deadlockfree(X \cup \{V\})$

2. X is a set of processes. For $T \in X$ we define $disabled(T, X)$ by
   $disabled(T, X) \equiv (\mathbf{E}\ t : t \in \mathbf{t}U : S(t\lceil \mathbf{a}T, T) \neq \varnothing \wedge (\mathbf{A}\ s : ts \in \mathbf{t}U : S(ts, U) \cap \mathbf{a}T = \varnothing))$
   where $U = (\mathbf{W}\ T : T \in X : T)$.
   $disablefree(X)$ is defined by $disablefree(X) \equiv (\mathbf{A}\ T : T \in X : \neg disabled(T, X))$
   Derive properties of *disablefree* that are similar to the properties of *lockfree* and *deadlockfree*.

(End of Exercises)

5 Livelock and nondeterminism

5.0 Introduction

Let T and U be processes. As explained in Section 1.4 we view $T\,\mathbf{b}\,U$ as the specification of the composite of the mechanisms specified by T and U respectively. This composite behaves according to $T\,\mathbf{w}\,U$.
Symbols of $\mathbf{a}T \cap \mathbf{a}U$ are called *internal* symbols. They correspond to *internal* events.
Symbols of $\mathbf{a}T \div \mathbf{a}U$ are called *external* symbols. They correspond to *external* events.
The blend of T and U does not contain any information about the internal events.

In this chapter we have the following assumption about the behaviour of a composite.

> 'The internal events occur automatically and instantaneously as soon as they can, without being observed or controlled by the environment of the process.'
>
> C.A.R. Hoare [8, section 3.5]

Consider processes T and U defined by

$$T = pref(TR((a ; x \mid b ; y)^*)) \quad \text{and} \quad U = pref(TR((a ; x)^*))$$

Then $T\,\mathbf{b}\,U \;=\; pref(TR((b ; y)^*))$

From $T\,\mathbf{b}\,U$ one concludes that the composite eventually gets involved in event b. However, whenever event b can occur, internal events are possible and, according to our assumption, an internal event will happen. It is not guaranteed that b will ever happen.
This phenomenon is called *livelock*. (It is also known as *infinite chatter* or as *divergence*). The behaviour of the composite is not in accordance with $T\,\mathbf{b}\,U$.

The phenomenon of *nondeterminism* is illustrated by the following example.

Let $T = \langle\{a,b,x,y\},\{\epsilon,x,xa,y,yb\}\rangle$. Then $T\,\mathbf{b}\,RUN(x,y) = \langle\{a,b\},\{\epsilon,a,b\}\rangle$.
From $T\,\mathbf{b}\,RUN(x,y)$ one may infer that either a or b may happen initially. From our assumption, however, we conclude that either x or y will occur instantaneously after which a is not possible any more or b is not possible any more. This is not reflected in $\langle\{a,b\},\{\epsilon,a,b\}\rangle$. We say that the composite of T and $RUN(x,y)$ has (internal) nondeterminism.

In the sequel we study conditions under which we can guarantee the blend of processes to be a proper specification of the composite of the corresponding mechanisms.

Since livelock and nondeterminism arise when certain events are concealed, we first study the relation between the mechanism described by a process T and the mechanism described by $T \restriction B$, where B is a subset of $\mathbf{a}T$. Notice that $T \restriction B = T \ \mathbf{b}\ RUN(\mathbf{a}T \setminus B)$.

In examples a process T is sometimes specified by a command S.
Then $T = pref(TR(S))$.

Processes are also specified by state graphs. Unless stated otherwise, the alphabet of the corresponding process consists of all labels that occur in the state graph.

5.1 Livelock

For process T and subset B of $\mathbf{a}T$ we define $livelock(B,T)$ and $livelockfree(B,T)$ by

$livelock(B,T) \equiv (\mathbf{E}\, t : t \in \mathbf{t}T : (\mathbf{A}\, n : n \geqslant 0 : (\mathbf{E}\, u : u \in B^* \wedge tu \in \mathbf{t}T : l(u) > n)))$

$livelockfree(B,T) \equiv \neg livelock(B,T)$

If T is obvious from the context we omit T and write $livelock(B)$ and $livelockfree(B)$.

Applying König's Lemma (cf. [10]) yields

Property 5.1.0

Let T be a process with a finite alphabet, and let B be a subset of $\mathbf{a}T$ such that $livelock(B)$ holds.

Then there exists a trace t, $t \in \mathbf{t}T$, and an infinite sequence $b(i : i \geqslant 0)$ such that $(\mathbf{A}\, i : i \geqslant 0 : b(i) \in B)$ and such that all finite prefixes of $tb(i : i \geqslant 0)$ belong to $\mathbf{t}T$.

(End of Property)

Property 5.1.1

For process T and subsets A and B of $\mathbf{a}T$ such that $A \subseteq B$, we have

$livelock(A) \Rightarrow livelock(B)$

$livelockfree(B) \Rightarrow livelockfree(A)$

(End of Property)

Exercises

0. Disprove: $livelockfree(A, T) \wedge livelockfree(B, T) \Rightarrow livelockfree(A \cup B, T)$

1. Prove
   (i) $livelockfree(\varnothing, T)$
   (ii) T is non-terminating $\Rightarrow$ $livelock(\mathbf{a}T, T)$

2. Disprove: $livelock(\mathbf{a}T, T) \Rightarrow T$ is non-terminating

3. Prove that for processes T and U:
   $livelockfree(\mathbf{a}T \cap \mathbf{a}U, T) \vee livelockfree(\mathbf{a}T \cap \mathbf{a}U, U)$
   $\Rightarrow livelockfree(\mathbf{a}T \cap \mathbf{a}U, T \mathbin{\mathbf{w}} U)$

4. Prove
   (i) $(\mathbf{A}\, n : n \geqslant 0 : (\mathbf{E}\, t : t \in \mathbf{t}SEM(a, b) : (\mathbf{E}\, u : u \in \{b\}^* \wedge tu \in \mathbf{t}SEM(a, b) : l(u) > n)))$
   (ii) $livelockfree(\{b\}, SEM(a, b))$
   (iii) $\neg livelockfree(\{a\}, SEM(a, b))$

(End of Exercises)

5.2 Independence and Transparency

Let T be a process and let B be a subset of $\mathbf{a}T$. The complement of B with respect to $\mathbf{a}T$, i.e. $\mathbf{a}T \setminus B$, is denoted by $\overline{B}$. According to Property 1.4.2.3 we have

$$T\restriction B \;=\; T \,\mathbf{b}\, RUN(\overline{B})$$

In the sequel we investigate conditions under which $T\restriction B$ is a proper description of the composite of the mechanisms associated with T and $RUN(\overline{B})$.

To that end we define four types of independence, each type being more restrictive than the previous ones. We give a mechanistic appreciation of each type.

For $j, 0 \leq j < 4$, $I_j(B,T)$, pronounced as 'B is j-independent with respect to T', is defined by

$$I_0(B,T) \;\equiv\; (\mathbf{A}\, t : t \in \mathbf{t}T : S(t,T) \subseteq B \;\Rightarrow\; S(t,T) = S(t\restriction B, T\restriction B))$$

$$I_1(B,T) \;\equiv\; (\mathbf{A}\, s,t : s \in \mathbf{t}T \wedge t \in \mathbf{t}T : s\restriction B \leq t\restriction B \;\Rightarrow\; (\mathbf{E}\, u : su \in \mathbf{t}T : su\restriction B = t\restriction B))$$

$$I_2(B,T) \;\equiv\; (\mathbf{A}\, t : t \in \mathbf{t}T : (\mathbf{E}\, u : u \in (\overline{B})^* \wedge tu \in \mathbf{t}T : S(tu,T) = S(t\restriction B, T\restriction B)))$$

$$I_3(B,T) \;\equiv\; I_0(B,T) \wedge \mathit{livelockfree}(\overline{B},T)$$

An appreciation in terms of the mechanism specified by T is the following.

$I_0(B,T)$: If the mechanism enters a state in which only events of B are possible, then the mechanism behaves according to $T\restriction B$.

$I_1(B,T)$: From each state of the mechanism it is possible to continue such that the behaviour of the mechanism is as expected from $T\restriction B$.

$I_2(B,T)$: From each state of the mechanism it is possible to enter a state (via events of $\overline{B}$) such that only events of B are possible. The mechanism behaves in that state according to $T\restriction B$.

$I_3(B,T)$: For each state of the mechanism it is guaranteed that performing internal events (events of $\overline{B}$) will terminate in a state in which only events of B are possible. The mechanism behaves in that state according to $T\restriction B$.

If $I_3(B,T)$ holds we say that B is *transparent* with respect to T.

In the sequel T is a fixed process and B is a subset of $\mathbf{a}T$.
We write $I_j(B)$ instead of $I_j(B,T)$.

Property 5.2.0

For any $t, t \in \mathbf{t}T$, we have $S(t,T) \cap B \subseteq S(t{\restriction}B, T{\restriction}B)$

(End of Property)

Lemma 5.2.1

$$I_1(B) \equiv (\mathbf{A}\, s,v : s \in \mathbf{t}T \wedge (s{\restriction}B)v \in \mathbf{t}T{\restriction}B : (\mathbf{E}\, u : su \in \mathbf{t}T : su{\restriction}B = (s{\restriction}B)v))$$

Proof

We derive

$(\mathbf{A}\, s,v : s \in \mathbf{t}T \wedge (s{\restriction}B)v \in \mathbf{t}T{\restriction}B : (\mathbf{E}\, u : su \in \mathbf{t}T : su{\restriction}B = (s{\restriction}B)v))$

$=$ { definition of projection }

$(\mathbf{A}\, s,v,t : s \in \mathbf{t}T \wedge t \in \mathbf{t}T \wedge (s{\restriction}B)v = t{\restriction}B : (\mathbf{E}\, u : su \in \mathbf{t}T : su{\restriction}B = (s{\restriction}B)v))$

$=$ { substitution }

$(\mathbf{A}\, s,v,t : s \in \mathbf{t}T \wedge t \in \mathbf{t}T \wedge (s{\restriction}B)v = t{\restriction}B : (\mathbf{E}\, u : su \in \mathbf{t}T : su{\restriction}B = t{\restriction}B))$

$=$ { definition of prefix }

$(\mathbf{A}\, s,t : s \in \mathbf{t}T \wedge t \in \mathbf{t}T \wedge s{\restriction}B \leqslant t{\restriction}B : (\mathbf{E}\, u : su \in \mathbf{t}T : su{\restriction}B = t{\restriction}B))$

$=$ { definition of I_1 }

$I_1(B)$

(End of Proof)

Theorem 5.2.2

0 $\quad I_1(B) \Rightarrow I_0(B)$

1 $\quad I_2(B) \Rightarrow I_1(B)$

2 $\quad I_3(B) \Rightarrow I_2(B)$

Proof

0. Assume $I_1(B)$. Let $t, t \in \mathbf{t}T$, be such that $S(t,T) \subseteq B$. We have to prove $S(t,T) = S(t{\restriction}B, T{\restriction}B)$. We derive

$a \in S(t{\restriction}B, T{\restriction}B)$

$=$ { definition of successor set }

$(t{\restriction}B)a \in \mathbf{t}T{\restriction}B$

$=$ { Lemma 5.2.1 }

$(\mathbf{E}\, u : tu \in \mathbf{t}T : tu\upharpoonright B = (t\upharpoonright B)a)$

$= \quad \{ S(t,T) \subseteq B \}$

$a \in S(t,T)$

Hence, $S(t,T) = S(t\upharpoonright B, T\upharpoonright B)$

1. Assume $I_2(B)$. Let s and t be such that $s \in \mathbf{t}T \wedge t \in \mathbf{t}T \wedge s\upharpoonright B \leqslant t\upharpoonright B$.
   We prove the existence of v such that $sv \in \mathbf{t}T \wedge sv\upharpoonright B = t\upharpoonright B$ by the following program. (The notation is from [3])

   $v := \epsilon$ {invariant $sv \in \mathbf{t}T \wedge sv\upharpoonright B \leqslant t\upharpoonright B$, variant function $l(t\upharpoonright B) - l(v\upharpoonright B)$ }

   ;**do** $sv\upharpoonright B \neq t\upharpoonright B$

   $\rightarrow$ $\{ sv\upharpoonright B < t\upharpoonright B \}$

   let b be such that $(sv\upharpoonright B)b \leqslant t\upharpoonright B$, then $b \in S(sv\upharpoonright B, T\upharpoonright B)$

   ; let u, $u \in (\bar{B})^*$, be such that $S(svu, T) = S(sv\upharpoonright B, T\upharpoonright B)$, u exists due to $I_2(B)$

   $\{ svub \in \mathbf{t}T \wedge svub\upharpoonright B = svb\upharpoonright B \leqslant t\upharpoonright B \wedge b \in B \}$

   ; $v := vub$

   od

   $\{ sv \in \mathbf{t}T \wedge sv\upharpoonright B = t\upharpoonright B \}$

2. Assume $I_3(B)$. Let $t \in \mathbf{t}T$. We have to prove the existence of v, $v \in (\bar{B})^*$, such that $tv \in \mathbf{t}T \wedge S(tv, T) = S(t\upharpoonright B, T\upharpoonright B)$.

   Consider the following program.

   $v := \epsilon$

   { invariant $tv \in \mathbf{t}T \wedge v \in (\bar{B})^*$

   variant function $l(v)$, bounded since *livelockfree* $(\bar{B})$ }

   ;**do** $S(tv, T) \cap \bar{B} \neq \varnothing$

   $\rightarrow$ let b be such that $b \in S(tv, T) \cap \bar{B}$

   $\{ tvb \in \mathbf{t}T \wedge vb \in (\bar{B})^* \}$

   ; $v := vb$

   od

   $\{ S(tv, T) \cap \bar{B} = \varnothing \wedge v \in (\bar{B})^*$, hence, $S(tv, T) \subseteq B \wedge tv\upharpoonright B = t\upharpoonright B$.

   From $I_0(B)$ we infer $S(tv, T) = S(tv\upharpoonright B, T\upharpoonright B) = S(t\upharpoonright B, T\upharpoonright B) \}$

We conclude $(\mathbf{E}\, v : v \in (\bar{B})^* \wedge tv \in \mathbf{t}T : S(tv, T) = S(t\upharpoonright B, T\upharpoonright B))$

(End of Proof)

Corollary 5.2.3

If $livelockfree(\overline{B})$ then $I_0(B)$, $I_1(B)$, $I_2(B)$, and $I_3(B)$ are equivalent.

(End of Corollary)

The following example shows that the implications in Theorem 5.2.2 are proper implications : the converses of these implications do not hold in general.

Example 5.2.4

0. Process T is defined by command $a;a^* \mid b$.
   For $t \in \mathbf{t}T$, not ending in symbol b we have $\neg(S(t,T) \subseteq \{b\})$.
   For $t \in \mathbf{t}T$, ending in symbol b we have $S(t,T) = \emptyset = S(t\restriction b, T\restriction b)$.
   Hence, $I_0(\{b\})$.
   Since $a\restriction b \leqslant b\restriction b \ \wedge \ (\mathbf{A}\, u : au \in \mathbf{t}T : au\restriction b \neq b\restriction b)$, we have $\neg I_1(\{b\})$.

   We conclude $I_0(\{b\}) \ \wedge \ \neg I_1(\{b\})$.

1. Process T is defined by $(a \mid b)^*$.
   From $t \in \mathbf{t}T \ \Rightarrow \ tb \in \mathbf{t}T$ we infer $I_1(\{b\})$.
   On the other hand we have $(\mathbf{A}\, t : t \in \mathbf{t}T : S(t,T) = \{a,b\} \neq \{b\})$, which implies $\neg I_2(\{b\})$.

   We conclude $I_1(\{b\}) \ \wedge \ \neg I_2(\{b\})$.

2. Process T is defined by $a^*;b;c$.
   For $t \in \mathbf{t}T$, ending in b or c we have $S(t,T) = S(t\restriction c, T\restriction c)$.
   For $t \in \mathbf{t}T$, not ending in b or c we have $S(tb,T) = S(t\restriction c, T\restriction c)$.
   Hence, $I_2(\{c\})$.
   Since for all n, $n \geqslant 0$, $a^n \in \mathbf{t}T$ we have $\neg I_3(\{c\})$.

   We conclude $I_2(\{c\}) \ \wedge \ \neg I_3(\{c\})$.

(End of Example)

The next theorem relates independence to state graphs.

Theorem 5.2.5

$$I_1(B) \ \Rightarrow \ (\mathbf{A}\, s,t : s \in \mathbf{t}T \ \wedge \ t \in \mathbf{t}T : s \sim t \ \Rightarrow \ s\restriction B \sim t\restriction B)$$

Proof

Assume $I_1(B)$. Let s and t be elements of $\mathbf{t}T$ such that $s \sim t$.
Let $v \in B^*$ be such that $(s\restriction B)v \in \mathbf{t}T\restriction B$. We derive

$$
\begin{array}{ll}
 & \text{true} \\
= & \quad \{ \text{definition of prefix} \} \\
 & s\restriction B \leqslant (s\restriction B)v \\
= & \quad \{ \text{Lemma 5.2.1}, (s\restriction B)v \in \mathbf{t}T\restriction B \} \\
 & (\mathbf{E}\, u : su \in \mathbf{t}T : su\restriction B = (s\restriction B)v) \\
= & \quad \{ \text{property of projection} \} \\
 & (\mathbf{E}\, u : su \in \mathbf{t}T : (s\restriction B)(u\restriction B) = (s\restriction B)v) \\
= & \quad \{ \text{property of concatenation} \} \\
 & (\mathbf{E}\, u : su \in \mathbf{t}T : u\restriction B = v) \\
= & \quad \{ s \sim t \} \\
 & (\mathbf{E}\, u : tu \in \mathbf{t}T : u\restriction B = v) \\
= & \quad \{ \text{properties of concatenation and projection} \} \\
 & (\mathbf{E}\, u : tu \in \mathbf{t}T : tu\restriction B = (t\restriction B)v) \\
\Rightarrow & \quad \{ \text{definition of projection} \} \\
 & (t\restriction B)v \in \mathbf{t}T\restriction B
\end{array}
$$

For reasons of symmetry we conclude

$$(\mathbf{A}\, v : v \in B^* : (s\restriction B)v \in \mathbf{t}T\restriction B \equiv (t\restriction B)v \in \mathbf{t}T\restriction B)$$

Hence, $s\restriction B \sim t\restriction B$

(End of Proof)

Corollary 5.2.6

If B is transparent with respect to T then the number of states of $T\restriction B$ is at most the number of states of T.

(End of Corollary)

The following theorem is another consequence of Theorem 5.2.5 .

Theorem 5.2.7

Let B be transparent with respect to T. A state graph of $T\restriction B$ is obtained from the state graph of T by removing all arcs labeled with symbols of $\overline{B}$ thereby identifying the states connected by these.

Proof

We have to show that for B transparent, $s \in \mathbf{t}T$, $c \in \overline{B}$, and $sc \in \mathbf{t}T$:

$$(\mathbf{A}\, t,u : t \in [s]_T \wedge u \in [sc]_T : t\restriction B \sim u\restriction B)$$

Let B be transparent with respect to T. Then $I_1(B,T)$ (cf. Theorem 5.2.2).
Let $s \in \mathbf{t}T$, $c \in \overline{B}$ and $sc \in \mathbf{t}T$. We derive

$$\begin{array}{ll} & t \in [s]_T \wedge u \in [sc]_T \\ = & \quad \{\text{ definition of equivalence class }\} \\ & t \sim s \wedge u \sim sc \\ \Rightarrow & \quad \{\text{ Theorem 5.2.5 }\} \\ & t\restriction B \sim s\restriction B \wedge u\restriction B \sim sc\restriction B \\ = & \quad \{ c \in \overline{B} \} \\ & t\restriction B \sim s\restriction B \wedge u\restriction B \sim s\restriction B \\ \Rightarrow & \quad \{\text{ transitivity of } \sim \} \\ & t\restriction B \sim u\restriction B \end{array}$$

(End of Proof)

Notice that $I_0(B,T)$ can be expressed in terms of states:

$$I_0(B,T) \equiv (\mathbf{A}\, t : t \in \mathbf{t}T : S([t],T) \subseteq B \Rightarrow S([t],T) = S([t\restriction B], T\restriction B)),$$

since the extension of successor set from traces to states yields

$$S(t,T) = S([t],T) \text{ for all } t, t \in \mathbf{t}T.$$

Theorem 5.2.8

None of the independencies are closed under union.

Proof

Process T is defined by command $a;b \mid x;b;a$. Then $I_3(\{a\})$ and $I_3(\{b\})$.
From $S(x,T) = \{b\} \neq \{a,b\} = S(x\restriction\{a,b\}, T\restriction\{a,b\})$ we conclude $\neg I_0(\{a,b\})$.

Theorem 5.2.2 yields that for any j, $0 \leqslant j < 4$,

$$I_j(\{a\}) \wedge I_j(\{b\}) \wedge \neg I_j(\{a,b\})$$

(End of Proof)

Let B be a 1-independent subset of $\mathbf{a}T$ and let $t \in \mathbf{t}T, b \in B, c \in \overline{B}$ such that $tc \in \mathbf{t}T \wedge tb \in \mathbf{t}T$.

Then $tc\upharpoonright B = t\upharpoonright B \leqslant tb\upharpoonright B$. Since B is 1-independent, there exists a u, $u \in (\overline{B})^*$, such that $tcub \in \mathbf{t}T$. Hence, the choice of c instead of b does not disable b.

Phrased differently: symbols of B cannot be disabled by symbols of $\overline{B}$.

Suppose that B and C are 1-independent subsets of $\mathbf{a}T$. A symbol of $B \cap C$ cannot be disabled by a symbol not in B nor by a symbol not in C, hence, only by symbols in $B \cap C$.

This observation might lead to the conjecture that 1-independence is closed under intersection. The next theorem, however, shows that none of the independencies are closed under intersection.

Theorem 5.2.9

None of the independencies are closed under intersection.

Proof

By the following counterexamples.

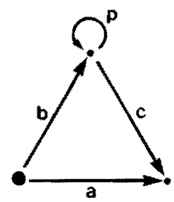


Figure 5.0

0. Process T is defined by the state graph of Figure 5.0 .
   We have $I_0(\{a,b\}) \wedge I_0(\{a,c\})$
   From $S(bc,T) = \varnothing \subseteq \{a\}$ and $S(bc\upharpoonright a, T\upharpoonright a) = \{a\}$ we infer $\neg I_0(\{a\})$.

1. Process T is defined by the state graph of Figure 5.1 .

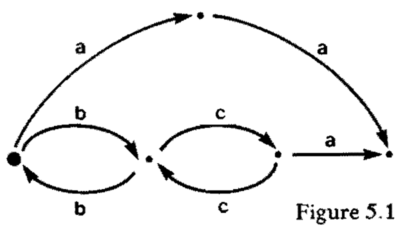


Figure 5.1

Trace t, $t \in \mathbf{t}T$, consisting of b's and c's may be extended with aa if and only if the number of b's is even and the number of c's is even. It may be extended with a single a only, if the number of b's and the number of c's are both odd.

We have $I_1(\{a,b\})$ and $I_1(\{a,c\})$.

Examination of traces bca and aa yields $\neg I_1(\{a\})$.

2. Process T is given by the state graph of Figure 5.2 .

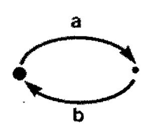


Figure 5.2

We have $I_2(\{a\})$, $I_2(\{b\})$, $I_3(\{a\})$, and $I_3(\{b\})$.

Since for all t, $t \in \mathbf{t}T$, the successor set $S(t,T)$ is non-empty, we have $\neg I_2(\varnothing)$ and, hence, $\neg I_3(\varnothing)$.

(End of Proof)

After having studied the counterexamples of the previous theorem one could hope for

$$I_3(B) \wedge I_3(C) \Rightarrow I_0(B \cap C)$$

We supply a counterexample for this implication as well. Notice that such a counterexample provides a proof of Theorem 5.2.9 as well. The example provides insight into conditions under which one may hope for more positive results. Since the example is a nice illustration of the theory, we devote a theorem to it.

Theorem 5.2.10

There exists a process T and there exists subsets B and C of $\mathbf{a}T$ such that

$$I_3(B) \wedge I_3(C) \wedge \neg I_0(B \cap C).$$

Proof

We construct process T that has alphabet $\{a,b,x\}$, such that $I_3(\{a,x\})$, $I_3(\{b,x\})$, and $\neg I_0(\{x\})$ hold.

Evidently, *livelockfree* $(\{a\})$ and *livelockfree* $(\{b\})$ have to hold.

Process T will be symmetric with respect to a and b.

Figure 5.3 shows the state graph of T.

$T\restriction\{a,b\} = SYNC_{2,2}(a,b)$, hence *livelockfree* $(\{a\})$ and *livelockfree* $(\{b\})$.

$T\restriction\{a,x\}$ corresponds to the command $(a;a)^*;(a;x \mid x;x)$.

$T\restriction\{b,x\}$ corresponds to the command $(b;b)^*;(b;x \mid x;x)$.

xx is possible if and only if the parity of the a's is even and the parity of the b's is even; a single x is possible if the parity of a's is odd and the parity of the b's is odd.

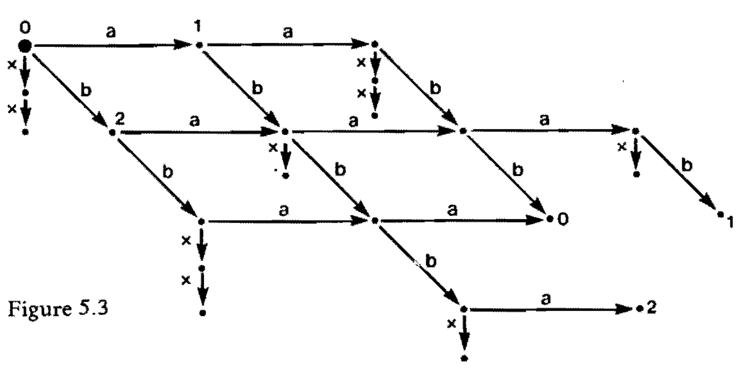


Figure 5.3

We have $I_0(\{a,x\}) \wedge livelockfree(\{b\})$. hence $I_3(\{a,x\})$. For reasons of symmetry $I_3(\{b,x\})$ holds as well.

From $S(abx,T)=\varnothing \wedge S(abx\upharpoonright x, T\upharpoonright x)=\{x\}$ we infer $\neg I_0(\{x\})$.

(End of Proof)

In the trace structure that was exhibited in the proof of Theorem 5.2.10 we have $livelock(\{a,b\})$. From the definition of 3-independence it is evident that $I_3(B)$ cannot be proved in the presence of $livelock(\bar{B})$.

The next theorem is the main theorem of this section.

Theorem 5.2.11

Let T be a process and let B and C be subsets of $\mathbf{a}T$ such that T does not have livelock with respect to $\mathbf{a}T\setminus(B\cap C)$. Then

$$\begin{array}{ll} & B \text{ and } C \text{ are transparent with respect to } T \\ \Rightarrow & B\cap C \text{ is transparent with respect to } T \end{array}$$

(End of Theorem)

This theorem may also be phrased as

'in the absence of livelock, transparency is closed under intersection'

For a proof of Theorem 5.2.11 we first derive some lemmata. T is a process and B and C are subsets of $\mathbf{a}T$.

Lemma 5.2.12

$$\begin{array}{ll} & I_3(B) \wedge I_3(C) \wedge b\in\overline{B\cap C} \\ \Rightarrow & (\mathbf{A}\, s,t : sb\in\mathbf{t}T \wedge st\in\mathbf{t}T : (\mathbf{E}\, u : sbu\in\mathbf{t}T : sbu\upharpoonright(B\cap C)=st\upharpoonright(B\cap C))) \end{array}$$

Proof

Assume $I_3(B) \wedge I_3(C) \wedge b\in\overline{B\cap C}$.
For reasons of symmetry we assume $b\notin C$.

Let s and t be such that $sb\in\mathbf{t}T \wedge st\in\mathbf{t}T$. We derive

$$\begin{array}{ll} & sb\upharpoonright C = s\upharpoonright C \leqslant st\upharpoonright C \\ \Rightarrow & \quad \{ I_1(C).\ sb\in\mathbf{t}T \wedge st\in\mathbf{t}T \} \\ & (\mathbf{E}\, u : sbu\in\mathbf{t}T : sbu\upharpoonright C = st\upharpoonright C) \end{array}$$

$\Rightarrow$ { application of projection on B }

$(\mathbf{E}\, u : sbu \in \mathbf{t}T : sbu{\restriction}(B \cap C) = st{\restriction}(B \cap C))$

(End of Proof)

Lemma 5.2.13

If $livelockfree(\overline{B \cap C})$ then

$$\begin{array}{l} I_3(B) \wedge I_3(C) \wedge b \in B \cap C \\ \Rightarrow (\mathbf{A}\, s, t : sb \in \mathbf{t}T \wedge st \in \mathbf{t}T \wedge sb{\restriction}(B \cap C) \leqslant st{\restriction}(B \cap C) : \\ \qquad\qquad (\mathbf{E}\, u : sbu \in \mathbf{t}T : sbu{\restriction}(B \cap C) = st{\restriction}(B \cap C))) \end{array}$$

Proof

Assume $livelockfree(\overline{B \cap C}) \wedge I_3(B) \wedge I_3(C) \wedge b \in B \cap C$.
From $livelockfree(\overline{B \cap C})$ we infer that the function f given by

$$f(t) = (\mathbf{MAX}\ u : tu \in \mathbf{t}T \wedge u \in (\overline{B \cap C})^* : l(u))$$

is well-defined.
By induction on $f(s)$ we prove that for all s, $s \in \mathbf{t}T \wedge sb \in \mathbf{t}T$:

$(\mathbf{A}\, t : st \in \mathbf{t}T \wedge sb{\restriction}(B \cap C) \leqslant st{\restriction}(B \cap C) : (\mathbf{E}\, u : sbu \in \mathbf{t}T : sbu{\restriction}(B \cap C) = st{\restriction}(B \cap C)))$

Base

Let s be such that $s \in \mathbf{t}T \wedge sb \in \mathbf{t}T \wedge f(s) = 0$.
Let t be such that $st \in \mathbf{t}T \wedge sb{\restriction}(B \cap C) \leqslant st{\restriction}(B \cap C)$. We derive

$sb{\restriction}(B \cap C) \leqslant st{\restriction}(B \cap C)$

$=$ { $b \in B \cap C$ }

$(\mathbf{E}\, u, v : v \in (\overline{B \cap C})^* \wedge u \in \mathbf{a}T^* : st = svbu)$

$=$ { $f(s) = 0$ }

$(\mathbf{E}\, u : u \in \mathbf{a}T^* : st = sbu)$

$\Rightarrow$ { $st \in \mathbf{t}T$, application of projection }

$(\mathbf{E}\, u : sbu \in \mathbf{t}T : st{\restriction}(B \cap C) = sbu{\restriction}(B \cap C))$

Step

Let s be such that $s \in \mathbf{t}T \wedge sb \in \mathbf{t}T \wedge f(s) > 0$.
Let t be such that $st \in \mathbf{t}T \wedge sb{\restriction}(B \cap C) \leqslant st{\restriction}(B \cap C)$
Then $t = vbu_0$ for some $v \in (\overline{B \cap C})^*$ and $u_0 \in \mathbf{a}T^*$ (cf. Base)

We distinguish two cases:

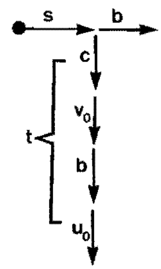


Figure 5.4

(i) $v = \epsilon$.

Then $t = bu_0$, hence, $(\mathbf{E}\, u : sbu \in \mathbf{t}T : sbu\restriction(B \cap C) = st\restriction(B \cap C))$

(ii) $v = cv_0$ with $c \in \overline{B \cap C}$

For reasons of symmetry we assume $c \notin B$.
We then have

$$st = scv_0bu_0 \wedge c \notin B \wedge v_0 \in (\overline{B \cap C})^* \quad \text{(cf. Figure 5.4).}$$

We derive

$$\begin{array}{ll} & sb \in \mathbf{t}T \wedge scv_0bu_0 \in \mathbf{t}T \wedge c \notin B \\ \Rightarrow & \{ sc\restriction B = s\restriction B \text{, } \mathbf{t}T \text{ is prefix-closed} \} \\ & S(sc\restriction B, T\restriction B) = S(s\restriction B, T\restriction B) \wedge sb \in \mathbf{t}T \wedge sc \in \mathbf{t}T \\ \Rightarrow & \{ b \in B \cap C \text{, hence } (s\restriction B)b \in \mathbf{t}T\restriction B \} \\ & b \in S(sc\restriction B, T\restriction B) \wedge sc \in \mathbf{t}T \\ \Rightarrow & \{ I_2(B) \} \\ & (\mathbf{E}\, u : u \in (\overline{B})^* \wedge scu \in \mathbf{t}T : scub \in tT) \end{array}$$

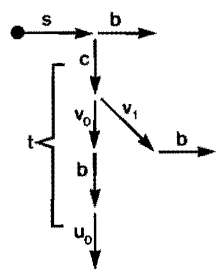


Figure 5.5

Choose $v_1 \in (\overline{B})^*$ such that $scv_1b \in \mathbf{t}T$ (cf. Figure 5.5)
We derive

$$\begin{array}{ll} & scv_1\restriction B = s\restriction B \leqslant st\restriction B \wedge scv_1 \in \mathbf{t}T \wedge st \in \mathbf{t}T \\ \Rightarrow & \{ I_1(B) \} \\ & (\mathbf{E}\, u : scv_1u \in \mathbf{t}T : scv_1u\restriction B = st\restriction B) \\ \Rightarrow & \{ \text{application of projection on } C \} \\ & (\mathbf{E}\, u : scv_1u \in \mathbf{t}T : scv_1u\restriction(B \cap C) = st\restriction(B \cap C)) \end{array}$$

Choose u_1 such that $scv_1u_1 \in \mathbf{t}T \wedge scv_1u_1\restriction(B \cap C) = st\restriction(B \cap C)$
(cf. Figure 5.6)

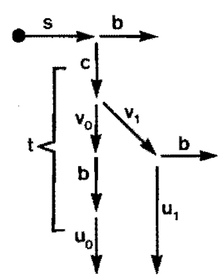


Figure 5.6

From $c \notin B \cap C$ and $v_1 \in (\overline{B})^*$ we infer

$$f(scv_1) \leqslant f(s) - 1 < f(s)$$

Furthermore we have

$$scv_1b\restriction(B \cap C) = sb\restriction(B \cap C) \leqslant st\restriction(B \cap C) = scv_1u_1\restriction(B \cap C)$$
$$\wedge\ scv_1u_1 \in \mathbf{t}T \wedge scv_1b \in \mathbf{t}T \wedge scv_1 \in \mathbf{t}T$$

Hence, we may apply the induction hypothesis with s replaced by scv_1 and t replaced by u_1.

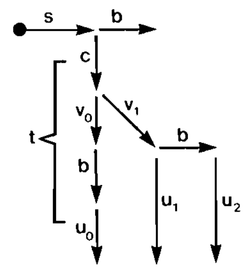


Figure 5.7

This yields u_2 such that

$$scv_1bu_2 \in \mathbf{t}T \ \wedge \ scv_1bu_2 \upharpoonright (B \cap C) = scv_1u_1 \upharpoonright (B \cap C)$$

(cf. Figure 5.7)

Our last step is the derivation

$$\begin{array}{ll} & sb \upharpoonright B \leqslant scv_1bu_2 \upharpoonright B \\ \Rightarrow & \{ I_1(B),\ sb \in \mathbf{t}T \ \wedge \ scv_1bu_2 \in \mathbf{t}T \ \} \\ & (\mathbf{E}\, u : sbu \in \mathbf{t}T : sbu \upharpoonright B = scv_1bu_2 \upharpoonright B) \\ \Rightarrow & \{ \text{application of projection on } C \ \} \\ & (\mathbf{E}\, u : sbu \in \mathbf{t}T : sbu \upharpoonright (B \cap C) = scv_1bu_2 \upharpoonright (B \cap C)) \\ = & \{ scv_1bu_2 \upharpoonright (B \cap C) = st \upharpoonright (B \cap C) \ \} \\ & (\mathbf{E}\, u : sbu \in \mathbf{t}T : sbu \upharpoonright (B \cap C) = st \upharpoonright (B \cap C)) \end{array}$$

(End of Proof)

Combining Lemma 5.2.12 and Lemma 5.2.13 yields

Lemma 5.2.14

If *livelockfree* $(\overline{B \cap C})$ then

$$\begin{array}{ll} & I_3(B) \ \wedge \ I_3(C) \ \wedge \ b \in \mathbf{a}T \\ \Rightarrow & (\mathbf{A}\, s, t : sb \in \mathbf{t}T \ \wedge \ st \in \mathbf{t}T \ \wedge \ sb \upharpoonright (B \cap C) \leqslant st \upharpoonright (B \cap C) : \\ & \qquad (\mathbf{E}\, u : sbu \in \mathbf{t}T : sbu \upharpoonright (B \cap C) = st \upharpoonright (B \cap C))) \end{array}$$

(End of Lemma)

We now prove Theorem 5.2.11 .

Proof

Let T be a process and let B and C be subsets of $\mathbf{a}T$ such that $I_3(B)$, $I_3(C)$, and *livelockfree* $(\overline{B \cap C})$ hold.

We have to prove $I_3(B \cap C)$.

Since *livelockfree* $(\overline{B \cap C})$ holds it suffices (Corollary 5.2.3) to prove $I_1(B \cap C)$.

Let $s_0 \in \mathbf{t}T$ and $s_1 \in \mathbf{t}T$ such that $s_0 \upharpoonright (B \cap C) \leqslant s_1 \upharpoonright (B \cap C)$.

The following program shows the existence of t, $t \in \mathbf{a}T^*$, such that

$$s_0t \in \mathbf{t}T \ \wedge \ s_0t \upharpoonright (B \cap C) = s_1 \upharpoonright (B \cap C).$$

$s := \epsilon \; ; t := s_1$
{ invariant $st \in \mathbf{t}T \wedge s \leqslant s_0 \wedge st{\restriction}(B \cap C) = s_1{\restriction}(B \cap C)$
variant function $l(s_0) - l(s)$ }
; **do** $s \neq s_0$
$\rightarrow$ { $s < s_0$ }
let b be such that $sb \leqslant s_0$
{ $sb{\restriction}(B \cap C) \leqslant s_0{\restriction}(B \cap C) \leqslant s_1{\restriction}(B \cap C) = st{\restriction}(B \cap C)$, apply Lemma 5.2.14 }
; choose u such that $sbu \in \mathbf{t}T \wedge sbu{\restriction}(B \cap C) = st{\restriction}(B \cap C)$
{ $sbu \in \mathbf{t}T \wedge sb \leqslant s_0 \wedge sbu{\restriction}(B \cap C) = st{\restriction}(B \cap C) = s_1{\restriction}(B \cap C)$ }
; $s := sb \; ; t := u$
od
{ $s = s_0$, hence $s_0 t \in \mathbf{t}T \wedge s_0 t{\restriction}(B \cap C) = s_1{\restriction}(B \cap C)$ }

(End of Proof)

As a consequence of Theorem 5.2.11 we have

Theorem 5.2.15

Let T be a process with a finite alphabet, and let A be a subset of $\mathbf{a}T$ such that T does not have livelock with respect to $\mathbf{a}T \backslash A$. Then there exists a smallest set B, $A \subseteq B \subseteq \mathbf{a}T$, that is transparent with respect to T.

Proof

From Property 5.1.1 we infer *livelockfree*$(\overline{B})$, for any B, $A \subseteq B \subseteq \mathbf{a}T$. According to Theorem 5.2.11 we then have $I_3(B \cap C)$ for any transparent B and C with $A \subseteq B \subseteq \mathbf{a}T$ and $A \subseteq C \subseteq \mathbf{a}T$. Moreover, $\mathbf{a}T$ is transparent with respect to T, hence, the intersection of all transparent subsets of $\mathbf{a}T$ containing A equals the smallest transparent subset of $\mathbf{a}T$ containing A.

(End of Proof)

In the next section we relate transparency to (internal) nondeterminism.

Exercises

0. Prove $I_0(\varnothing, T)$ and $I_1(\varnothing, T)$

   Disprove $I_2(\varnothing, T)$ and $I_3(\varnothing, T)$

1. Prove

   $I_1(B, T) \equiv (\mathbf{A}\, s, b : s \in \mathbf{t}T \wedge b \in B \wedge (s{\restriction}B)b \in \mathbf{t}T{\restriction}B : (\mathbf{E}\, u : u \in (\overline{B})^* : sub \in \mathbf{t}T))$

2. Determine *livelockfree*$(\{a\})$ and $I_0(\{b\})$ for the processes defined by the following commands.

   (i) $(a ; b)^*$

   (ii) $(a \mid b)^*$

   (iii) $(a \mid a ; b)^*$

   (iv) $a^* \mid b^*$

   (v) $a \mid b^*$

   (vi) $(a ; b \mid a ; a ; b)^*$

   (vii) $(a \mid b ; a^* ; b^*)$

3. Let p and q be positive integers. Prove that $\{a, c\}$ is transparent with respect to $SEM_p(a, b)\, \mathbf{w}\, SEM_q(b, c)$

4. T and U are processes. $\mathbf{a}T \cap \mathbf{a}U$ contains at most one element. $\mathbf{a}T \setminus \mathbf{a}U$ is transparent with respect to T and $\mathbf{a}U \setminus \mathbf{a}T$ is transparent with respect to U.

   Show that $\mathbf{a}T \div \mathbf{a}U$ is transparent with respect to $T\, \mathbf{w}\, U$.

5. Let S denote the command $(p{\cdot}a ; a \mid p{\cdot}b, q{\cdot}a ; b \mid q{\cdot}b ; c)^*$, and let

   $T = SEM_1(p{\cdot}a, p{\cdot}b)\, \mathbf{w}\, SEM_1(q{\cdot}a, q{\cdot}b)\, \mathbf{w}\, pref(TR(S))$

   (i) Show that $T{\restriction}\{a, b, c\} = SEM_1(a, b)\, \mathbf{w}\, SEM_1(b, c)$

   (ii) Determine *livelockfree*$(\overline{\{a, b, c\}}, T)$ and $I_0(\{a, b, c\}, T)$

6. T is a process with a finite alphabet. Subset B of $\mathbf{a}T$ is called *strongly independent* if $(\mathbf{A}\, t : t \in \mathbf{t}T : S(t{\restriction}B, T{\restriction}B) \subseteq S(t, T))$.

   Prove

   (i) $\varnothing$ and $\mathbf{a}T$ are strongly independent.

   (ii) The strongly independent subsets of $\mathbf{a}T$ form a complete lattice.

   (iii) B and $\overline{B}$ are strongly independent $\equiv$ $T = (T{\restriction}B)\, \mathbf{w}\, (T{\restriction}\overline{B})$

For A, $A \subseteq \mathbf{a}T$, $m(A)$ denotes the smallest strongly independent subset of $\mathbf{a}T$ containing A.

(iv) Show that $m(A)$ is well-defined and prove $T = (\mathbf{W}\ a : a \in \mathbf{a}T : m(\{a\}))$

(End of Exercises)

5.3 Transparency and nondeterminism

In this section we relate the theory developed in the previous sections to the theory of CSP in [8]. We present a short introduction to the model defined in [8]. For a more detailed treatment we also recommend [1].

A *CSP-process* is defined as a triple $<A, F, D>$ where

A is an alphabet

F is a set of pairs (t, X) where $t \in A^*$ and $X \subseteq A$

D is a subset of A^*

Let P, $P = <A, F, D>$, be a CSP-process. The set F is called the *failure set* of P. It consists of pairs (t, X) where t is a trace, $t \in A^*$, and X is a so-called *refusal set* of t. F is used to model nondeterminism.

Set D is called the *set of divergences* of P and consists of 'all traces of P after which P behaves chaotically'.

The triple $<A, F, D>$ should satisfy the following conditions (cf. [8, 3.9])

C0 $(\epsilon, \varnothing) \in F$

C1 $(tu, X) \in F \Rightarrow (t, \varnothing) \in F$

C2 $(t, X) \in F \wedge Y \subseteq X \Rightarrow (t, Y) \in F$

C3 $(t, X) \in F \wedge a \in A \Rightarrow (t, X \cup \{a\}) \in F \vee (ta, \varnothing) \in F$

C4 $D \subseteq \{t \mid (t, \varnothing) \in F\}$

C5 $t \in D \wedge u \in A^* \Rightarrow tu \in D$

C6 $t \in D \wedge X \subseteq A \Rightarrow (t, X) \in F$

The alphabet of P is denoted by $\mathbf{a}P$. The set $\{t \mid (t, \varnothing) \in F\}$ is called the trace set of P and is denoted by $\mathbf{t}P$.

From conditions C0 and C1 we infer that $\mathbf{t}P$ is non-empty and prefix-closed.

From C2 we conclude that the refusal sets of a trace t, $t \in \mathbf{t}P$, are determined by the maximal refusal sets of t.

A mechanistic appreciation of P is the following.

With P a mechanism is associated. With that mechanism in operation a so-called *trace thus far generated*, say t, is associated. Initially $t = \epsilon$. At any moment $t \in \mathbf{t}P$.

If t is an element of D anything may happen: the mechanism may refuse any event or may get involved in any event. This is expressed by conditions C5 and C6. The mechanism behaves chaotically.

If t is not an element of D, we consider the set $\{a \mid (ta, \varnothing) \in F\}$. For an element b of that set there exist two possibilities.

(i) $(\mathbf{E}\, X : (t, X) \in F : b \in X)$. Then b may happen but b may also be refused ('depending on some internal event, b may get disabled').

(ii) $(\mathbf{A}\, X : (t, X) \in F : b \notin X)$. Then b may happen, either since the environment initiates b or since the mechanism does so.

With CSP-process P we associate the process (i.e. the non-empty prefix-closed trace structure) $<\mathbf{a}P, \mathbf{t}P>$.

For $t, t \in \mathbf{t}P$, the successor set $S(t, <\mathbf{a}P, \mathbf{t}P>)$ is also denoted by $S(t, P)$. Notice that $S(t, P) = \{a \mid (ta, \varnothing) \in F\}$. From condition C3 we infer

Property 5.3.0

Let $P, P = <A, F, D>$ be a CSP-process and let $t \in \mathbf{t}P$. Then

$$(\mathbf{A}\, X : X \subseteq A \backslash S(t, P) : (t, X) \in F)$$

(End of Property)

The set of all CSP-processes is denoted by H, and the set of all non-empty prefix-closed trace structures is denoted by K (both sets with respect to the same universe Ω). We then have the following mappings.

$$tr : H \to K \text{ defined by } tr(P) = <\mathbf{a}P, \mathbf{t}P>$$

$$pr : K \to H \text{ defined by } pr(T) = <\mathbf{a}T, \{(t, X) \mid t \in \mathbf{t}T \wedge X \subseteq \mathbf{a}T \backslash S(t, T)\}, \varnothing>$$

Property 5.3.1

$$tr(pr(T)) = T \text{ for all } T, T \in K$$

Proof

For any $T, T \in K$, we have

$tr\,(pr\,(T\,))$
$=$ { definition of pr }
$tr\,(<\mathbf{a}T\,,\{(t\,,X\,)\mid t\in\mathbf{t}T\ \wedge\ X\subseteq \mathbf{a}T\backslash S(t\,,T\,)\},\varnothing>)$
$=$ { definition of tr and $\mathbf{t}P$ }
$<\mathbf{a}T\,,\{t\mid t\in\mathbf{t}T\ \wedge\ \varnothing\subseteq \mathbf{a}T\backslash S(t\,,T\,)\}>$
$=$ { calculus }
T

(End of Proof)

Property 5.3.2

For all $P\,,P=\,<A\,,F\,,D>$,

$pr\,(tr\,(P\,))=\,<A\,,\{(t\,,X\,)\mid(t\,,X\,)\in F\ \wedge\ X\subseteq A\backslash S(t\,,P\,)\},\varnothing>$

Proof

For any $P\,,P=\,<A\,,F\,,D>$, we derive

$pr\,(tr\,(P\,))$
$=$ { definition of tr }
$pr\,(<A\,,\{t\mid(t\,,\varnothing)\in F\,\}>)$
$=$ { definition of pr and successor set }
$<A\,,\{(t\,,X\,)\mid(t\,,\varnothing)\in F\ \wedge\ X\subseteq A\backslash S(t\,,P\,)\},\varnothing>$
$=$ { Property 5.3.0 and condition C2 }
$<A\,,\{(t\,,X\,)\mid(t\,,X\,)\in F\ \wedge\ X\subseteq A\backslash S(t\,,P\,)\},\varnothing>$

(End of Proof)

Let $P\,,P=\,<A\,,F\,,D>$, be a CSP-process and let B be a subset of $\mathbf{a}P$.
The *projection of P on B*, denoted by $P\restriction B$, is the CSP-process $<A_B\,,F_B\,,D_B>$ where

$A_B=B$
$F_B=\{(t\,,X\,)\mid X\subseteq B\ \wedge\ (t\in D_B\ \vee\ (\mathbf{E}\,u:(u\,,X\cup\overline{B}\,)\in F:t=u\restriction B\,))\}$
$D_B=\{t\mid(\mathbf{E}\,u\,,v:v\in B^*\wedge\ u\in D:t=(u\restriction B\,)v\,)\}$
$\quad\cup\{t\mid(\mathbf{E}\,u\,,v:v\in B^*\wedge\ (\mathbf{A}\,n:n\geqslant 0:(\mathbf{E}\,s:s\in(\overline{B}\,)^*\wedge\ l(s\,)>n:us\in\mathbf{t}P\,)):t=(u\restriction B\,)v\,)\}$

($\overline{B}$ denotes the complement of B with respect to $\mathbf{a}P$)

Example 5.3.3

T is defined by $T = pref(TR(b;x \mid c;y))$.

Let $P = pr(T)$, then $P = <A, F, D>$ where

$A = \{b, c, x, y\}$;

the set of (t, X) in F such that X is maximal equals

$\{(\epsilon, \{x, y\}), (b, \{b, c, y\}), (c, \{b, c, x\}), (bx, \{b, c, x, y\}), (cy, \{b, c, x, y\})\}$;

$D = \emptyset$

Let $B = \{x, y\}$, then $P \restriction B = <A_B, F_B, D_B>$ where

$A_B = \{x, y\}$;

$F_B = \{(\epsilon, \{x\}), (\epsilon, \{y\}), (\epsilon, \emptyset)$
$, (x, \{x, y\}), (x, \{x\}), (x, \{y\}), (x, \emptyset)$
$, (y, \{x, y\}), (y, \{x\}), (y, \{y\}), (y, \emptyset)$
$\}$;

$D_B = \emptyset$

Notice that $P \restriction B$ may refuse x as well as y initially, but not both.

$tr(P \restriction B) = <\{x, y\}, \{\epsilon, x, y\}>$ and $pr(tr(P \restriction B)) = <A', F', D'>$ where

$A' = \{x, y\}$;

$F' = \{(\epsilon, \emptyset)$
$, (x, \{x, y\}), (x, \{x\}), (x, \{y\}), (x, \emptyset)$
$, (y, \{x, y\}), (y, \{x\}), (y, \{y\}), (y, \emptyset)$
$\}$;

$D' = \emptyset$

Notice that $pr(tr(P \restriction B)) \neq P \restriction B$

(End of Example)

For an informal definition of *determinism* we quote C.A.R. Hoare [8].

> 'whenever there is more than one event possible, the choice between them is determined externally by the environment of the process. It is determined either in the sense that the environment can actually make the choice, or in the weaker sense that the environment can observe which choice has been made at the very moment of that choice.'

A formal definition is given by

$$P \text{ is deterministic } \equiv \quad pr(tr(P)) = P$$

Application of Property 5.3.2 yields

Theorem 5.3.4

$$P,\ P = \langle A, F, D\rangle, \text{ is deterministic}$$
$$\equiv \quad D = \emptyset \ \wedge\ (\mathbf{A}\, t, X : (t, X) \in F : X \subseteq A \setminus S(t, P))$$

(End of Theorem)

In the sequel $P, P = \langle A, F, D\rangle$, is a CSP-process and B is a subset of A. Furthermore, $\langle A_B, F_B, D_B\rangle$ denotes the CSP-process $P \upharpoonright B$.

livelockfree $(\bar{B}, \langle \mathbf{a}P, \mathbf{t}P\rangle)$ and $I_0(B, \langle \mathbf{a}P, \mathbf{t}P\rangle)$ are abbreviated to *livelockfree* $(\bar{B})$ and $I_0(B)$ respectively.

B is called transparent with respect to P if B is transparent with respect to $\langle \mathbf{a}P, \mathbf{t}P\rangle$, i.e. if $I_0(B) \wedge$ *livelockfree* $(\bar{B})$ holds.

Property 5.3.5

$$(\mathbf{t}P)\upharpoonright B \subseteq \mathbf{t}(P\upharpoonright B)$$

Proof

For any $t, t \in B^*$, we derive

$$t \in (\mathbf{t}P)\upharpoonright B$$
$$= \quad \{ \text{definition of projection} \}$$
$$(\mathbf{E}\, u : u \in \mathbf{t}P : t = u\upharpoonright B)$$
$$= \quad \{ \text{definition of } \mathbf{t}P \}$$
$$(\mathbf{E}\, u : (u, \emptyset) \in F : t = u\upharpoonright B)$$
$$= \quad \{ \text{predicate calculus} \}$$
$$(\mathbf{E}\, u : (u, \emptyset) \in F : t = u\upharpoonright B \ \wedge\ (\ \ (\mathbf{A}\, s : s \in (\bar{B})^* \wedge us \in \mathbf{t}P : S(us, P) \cap \bar{B} \neq \emptyset)$$
$$\vee\ (\mathbf{E}\, s : s \in (\bar{B})^* \wedge us \in \mathbf{t}P : S(us, P) \subseteq B)))$$
$$\Rightarrow \quad \{ \text{definition of } D_B \}$$
$$(\mathbf{E}\, u : (u, \emptyset) \in F : t = u\upharpoonright B \ \wedge\ (u\upharpoonright B \in D_B \ \vee\ (\mathbf{E}\, s : s \in (\bar{B})^* \wedge us \in \mathbf{t}P : S(us, P) \subseteq B)))$$

$\Rightarrow$ { Property 5.3.0 }

$(\mathbf{E}\,u : (u,\varnothing)\in F : t = u{\restriction}B \ \wedge\ (u{\restriction}B \in D_B \ \vee\ (\mathbf{E}\,s : s\in(\overline{B})^* : (us,\overline{B})\in F)))$

$\Rightarrow$ { predicate calculus, definition of projection }

$(\mathbf{E}\,u : (u,\varnothing)\in F : (t = u{\restriction}B \ \wedge\ u{\restriction}B \in D_B) \ \vee\ (\mathbf{E}\,s : s\in(\overline{B})^* : (us,\overline{B})\in F \ \wedge\ t = us{\restriction}B))$

$\Rightarrow$ { calculus }

$(\mathbf{E}\,u : (u,\varnothing)\in F : t\in D_B \ \vee\ (\mathbf{E}\,v : (v,\overline{B})\in F : t = v{\restriction}B))$

$=$ { F is non-empty, $(\epsilon,\varnothing)\in F$ }

$t\in D_B \ \vee\ (\mathbf{E}\,v : (v,\overline{B})\in F : t = v{\restriction}B)$

$=$ { definition of F_B }

$(t,\varnothing)\in F_B$

$=$ { definition of $\mathbf{t}(P{\restriction}B)$ }

$t\in\mathbf{t}(P{\restriction}B)$

(End of Proof)

Property 5.3.6

$livelockfree(\overline{B}) \ \Rightarrow\ (\mathbf{t}P){\restriction}B = \mathbf{t}(P{\restriction}B)$

Proof

Assume $livelockfree(\overline{B})$. For any t, $t\in B^*$, we have

$t\in\mathbf{t}(P{\restriction}B)$

$=$ { definition of $P{\restriction}B$ }

$(t,\varnothing)\in F_B$

$=$ { definition of F_B }

$t\in D_B \ \vee\ (\mathbf{E}\,u : (u,\overline{B})\in F : t = u{\restriction}B)$

$=$ { definition of D_B, $livelockfree(\overline{B})$ }

$(\mathbf{E}\,u,v : u\in D \ \wedge\ v\in B^* : t = (u{\restriction}B)v) \ \vee\ (\mathbf{E}\,u : (u,\overline{B})\in F : t = u{\restriction}B)$

$\Rightarrow$ { condition C5, $B^*\subseteq A^*$ }

$(\mathbf{E}\,u : u\in D : t = u{\restriction}B) \vee (\mathbf{E}\,u : (u,\overline{B})\in F : t = u{\restriction}B)$

$\Rightarrow$ { conditions C4 an C1 }

$(\mathbf{E}\,u : (u,\varnothing)\in F : t = u{\restriction}B) \vee (\mathbf{E}\,u : (u,\varnothing)\in F : t = u{\restriction}B)$

$=$ { definition of $\mathbf{t}P$ }

$t\in(\mathbf{t}P){\restriction}B$

Hence, $\mathbf{t}(P{\restriction}B)\subseteq(\mathbf{t}P){\restriction}B$. Combining this with Property 5.3.4 yields

$\mathbf{t}(P\restriction B) = (\mathbf{t}P)\restriction B$.

(End of Proof)

Property 5.3.7

If P is deterministic then $livelockfree(\overline{B}) \equiv D_B = \varnothing$

Proof

If P is deterministic then $D = \varnothing$ (Theorem 5.3.4). We derive

$livelockfree(\overline{B})$

$=$ { definition of livelockfree }

$(\mathbf{A}\, t : t \in \mathbf{t}P : (\mathbf{E}\, n : n \geqslant 0 : (\mathbf{A}\, u : u \in (\overline{B})^* \wedge l(u) > n : tu \notin \mathbf{t}P)))$

$=$ { definition of D_B, $D = \varnothing$ }

$D_B = \varnothing$

(End of Proof)

We are now ready for the main theorem of this section.

Theorem 5.3.8

Let P be a *deterministic* CSP-process and let B be a subset of the alphabet of P. Then

$P\restriction B$ is deterministic $\equiv$ B is transparent with respect to P

Proof

(i) Assume $P\restriction B$ is deterministic. We derive

$P\restriction B$ is deterministic

$\Rightarrow$ { Theorem 5.3.2 }

$D_B = \varnothing$

$=$ { P is deterministic, Property 5.3.7 }

$livelockfree(\overline{B})$

For any t, $t \in \mathbf{t}P$, such that $S(t, P) \subseteq B$, we have

$t \in \mathbf{t}P \wedge S(t, P) \subseteq B$

$=$ { Property 5.3.0 }

$(t, \mathbf{a}P \backslash S(t,P)) \in F \ \wedge \ S(t,P) \subseteq B$

$\Rightarrow \quad \{ S(t,P) \subseteq B \ \equiv \ \overline{B} \subseteq \mathbf{a}P \backslash S(t,P) \}$

$(t, \overline{B} \cup B \backslash S(t,P)) \in F$

$\Rightarrow \quad \{$ definition of F_B $\}$

$(t \upharpoonright B, B \backslash S(t,P)) \in F_B$

$\Rightarrow \quad \{ P \upharpoonright B$ is deterministic, Theorem 5.3.4 $\}$

$B \backslash S(t,P) \subseteq B \backslash S(t \upharpoonright B, P \upharpoonright B)$

$= \quad \{$ set calculus, $S(t,P) \subseteq B$ $\}$

$S(t \upharpoonright B, P \upharpoonright B) \subseteq S(t,P)$

$= \quad \{$ Property 5.2.0 $\}$

$S(t \upharpoonright B, P \upharpoonright B) = S(t,P)$

$= \quad \{$ *livelockfree* $(\overline{B})$, Property 5.3.6 $\}$

$S(t \upharpoonright B, <\mathbf{a}P, \mathbf{t}P> \upharpoonright B) = S(t, <\mathbf{a}P, \mathbf{t}P>)$

Hence, *livelockfree* $(\overline{B}) \wedge I_0(B)$ which is equivalent to B is transparent with respect to P.

(ii) Assume B is transparent with respect to P. Then *livelockfree* $(\overline{B}) \wedge I_0(B)$. From *livelockfree* $(\overline{B})$ and P is deterministic we infer $D_B = \varnothing$ (Property 5.3.7). We derive

livelockfree $(\overline{B})$

$\Rightarrow \quad \{$ definition of F_B, $D_B = \varnothing$ $\}$

$F_B = \{ (t, X) \mid X \subseteq B \ \wedge \ (\mathbf{E}\, u : (u, X \cup \overline{B}) \in F : t = u \upharpoonright B) \}$

$= \quad \{ P$ is deterministic, Theorem 5.3.4 $\}$

$F_B = \{ (t, X) \mid X \subseteq B \ \wedge \ (\mathbf{E}\, u : (u, X \cup \overline{B}) \in F \ \wedge \ X \cup \overline{B} \subseteq \mathbf{a}P \backslash S(u,P) : t = u \upharpoonright B) \}$

For any X, $X \subseteq B$, and u, $u \in \mathbf{a}P^*$, we have

$X \subseteq B \ \wedge \ (u, X \cup \overline{B}) \in F \ \wedge \ X \cup \overline{B} \subseteq \mathbf{a}P \backslash S(u,P)$

$= \quad \{$ set calculus $\}$

$X \subseteq B \ \wedge \ (u, X \cup \overline{B}) \in F \ \wedge \ S(u,P) \subseteq B \ \wedge \ X \cup \overline{B} \subseteq \mathbf{a}P \backslash S(u,P)$

$\Rightarrow \quad \{ I_0(B)$ and $\mathbf{t}(P \upharpoonright B) = (\mathbf{t}P) \upharpoonright B$ $\}$

$X \subseteq B \ \wedge \ (u, X \cup \overline{B}) \in F \ \wedge \ X \cup \overline{B} \subseteq \mathbf{a}P \backslash S(u \upharpoonright B, P \upharpoonright B)$

$\Rightarrow \quad \{$ definition of F_B, set calculus $\}$

$(u \upharpoonright B, X) \in F_B \ \wedge \ X \subseteq B \backslash S(u \upharpoonright B, P \upharpoonright B)$

Hence,

$(t, X) \in F_B$

$\Rightarrow$ { previous derivation }

$(\mathbf{E}\, u : (u, X \cup \bar{B}) \in F \ \wedge\ X \subseteq B \ \wedge\ X \cup \bar{B} \subseteq \mathbf{a}P \backslash S(t, P) : t = u \restriction B)$

$\Rightarrow$ { derivation above }

$X \subseteq B \backslash S(t, P \restriction B)$

Furthermore, we have $D_B = \varnothing$. Application of Theorem 5.3.4 yields $P \restriction B$ is deterministic.

(End of Proof)

For a deterministic CSP-process P we have, by definition, $pr(tr(P)) = P$. We have also, cf. Property 5.3.1, $tr(pr(T)) = T$ for $T \in K$. Hence, K may be identified with the set of deterministic CSP-processes. Theorem 5.3.8 expresses that this set is closed under projection on transparent alphabets.

We conclude that mechanisms that have (internal) nondeterminism cannot be described in terms of trace structures. That does not bother us, since we are not interested in mechanisms that have (internal) nondeterminism.

We shall avoid internal nondeterminism, either by guaranteeing that projection is done on a transparent alphabet or by implementing processes in such a way that internal events do *not* occur automatically and instantaneously.
We discuss such implementations in Chapter 6.

This concludes our discussion of CSP-processes.

5.4 Transparent components

In this section we apply the theory of Section 5.2 to components. Let component c be defined by

com $c(A)$:
  sub $p_0 : c_0, \ldots, p_{n-1} : c_{n-1}$ **bus**
  $[x_0 = y_0, \ldots, x_{m-1} = y_{m-1}]$
  S
moc

Then $TR(c) = T \upharpoonright A$ where

$$T = (\mathbf{W}\ i : 0 \leqslant i < n : (p_i{\cdot}TR(c_i))^{x_0, \ldots, x_{m-1}}_{y_0, \ldots, y_{m-1}})\ \mathbf{w}\ pref\ (TR(S))$$

In view of the theory developed in the previous sections we call c *livelockfree* if $livelockfree(\mathbf{a}T \backslash A, T)$ holds.

We call c *transparent* if A is transparent with respect to T.

Since for any process T, $\mathbf{a}T$ is transparent with respect to T, we have

Property 5.4.0

A component without subcomponents is transparent.

(End of Property)

Implementing a transparent component (i.e. constructing a mechanism that behaves according to its trace structure) is relatively easy since it does not matter how fast and in which order internal events will happen.

If component c is not transparent we implement the command of c in such a way that the nondeterminism of c is resolved without affecting $TR(c)$.

Example 5.4.1 (cf. Example 2.3.6)

Component sem_1 with $TR(sem_1) = SEM_1(a,b)$ is defined by

com $sem_1(a,b)$: $(a;b)^*$ **moc**

Component sem_2 with $TR(sem_2) = SEM_2(a,b)$ is defined by

com $sem_2(a, b)$:
  sub p, q : sem_1 **bus**
  $[\, p{\cdot}a = a,\ p{\cdot}b = q{\cdot}a,\ q{\cdot}b = b \,]$
  ϵ
moc

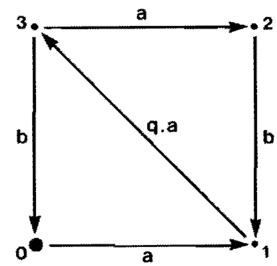


Figure 5.8

Let $T = SEM_1(a, q{\cdot}a) \mathbf{w}\ SEM_1(q{\cdot}a, b)$.
Then $TR(sem_2) = T \restriction \{a, b\}$.

The state graph of T is shown in Figure 5.8 . It does not contain any cycle of compound symbols. Hence, *livelockfree*$(\{q{\cdot}a\}, T)$ holds. In states 0, 2, and 3 the successor sets are subsets of $\{a, b\}$. They equal the successor sets obtained by projection on $\{a, b\}$. Hence, $I_0(\{a, b\}, T)$.

We conclude that sem_2 is transparent.

(End of Example)

Example 5.4.2

We transform component sem_2 of the previous example into component $asem_2$ by removing the equalities:

com $asem_2(a, b)$:
  sub p, q : sem_1 **bus**
  $(p{\cdot}a ; a \mid p{\cdot}b ; q{\cdot}a \mid q{\cdot}b ; b)^*$
moc

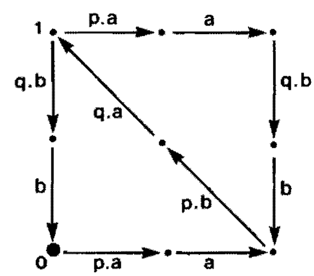


Figure 5.9

Let $T = SEM_1(p{\cdot}a, p{\cdot}b) \mathbf{w}\ SEM_1(q{\cdot}a, q{\cdot}b) \mathbf{w}\ pref(TR(S))$ where S denotes the command of $asem_2$. The state graph of T is shown in Figure 5.9 . Since there is no cycle of compound symbols, $asem_2$ is livelockfree.

From $S(p{\cdot}a\ a\ p{\cdot}b\ q{\cdot}a\ p{\cdot}a, T) = \{a\}$ and $S(a, T \restriction \{a, b\}) = \{a, b\}$ we infer
$\neg I_0(\{a, b\}, T)$.

Hence, $asem_2$ is not transparent.

If a and b are events that are initiated by the environment we implement S in such a way that the choice between $p{\cdot}a$ and $q{\cdot}b$ (state 1) is postponed until the environment has initiated event a or b.

(End of Example)

Example 5.4.3

We construct component $wsem_2$ that has trace structure $SEM_1(a,b)\ \mathbf{w}\ SEM_1(b,c)$:

> **com** $wsem_2(a,b,c)$:
> **sub** p,q : sem_1 **bus**
> $(p{\cdot}a\,;a \mid p{\cdot}b\,,q{\cdot}a\,;b \mid q{\cdot}b\,;c)^*$
> **moc**

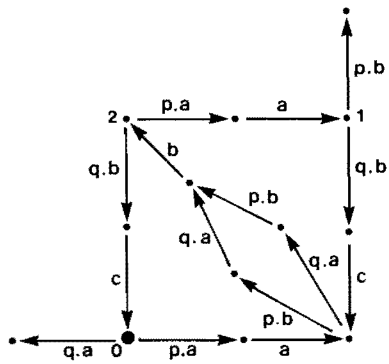


Figure 5.10

Let $T = SEM_1(p{\cdot}a\,,p{\cdot}b)\ \mathbf{w}\ SEM_1(q{\cdot}a\,,q{\cdot}b)\ \mathbf{w}\ pref(TR(S))$.

The state graph of T is shown in Figure 5.10 . Since $S(q{\cdot}a\,,T) = \emptyset$ and $S(q{\cdot}a\,,SEM_1(q{\cdot}a\,,q{\cdot}b)) \neq \emptyset$, $wsem_2$ is not deadlockfree. The number of consecutive compound symbols is bounded by 2. Hence, $wsem_2$ is livelockfree.

Command S should be implemented in such a way that $p{\cdot}b$ or $q{\cdot}a$ will happen only if the environment initiates b. As in Example 5.4.2, the choice (state 2) should be postponed.

If events a and c are initiated by the implementation we regard the implementation of $pref(TR(a\,;(b\,;a\,;c)^*)$ as a valid one.

(End of Example)

Example 5.4.4

In Example 2.3.3 we derived (recursive) component sem with $TR(sem) = SEM(a,b)$:

> **com** $sem(a,b)$:
> **sub** p : sem **bus**
> $((a \mid p{\cdot}b)\,;(p{\cdot}a \mid b))^*$
> **moc**

Let $T = SEM(p{\cdot}a\,,p{\cdot}b)\ \mathbf{w}\ pref(TR(S))$ where S denotes the command of sem.

For any $n\,, n \geqslant 0$, we have

$(p{\cdot}a\ p{\cdot}b)^n \in \mathbf{t}SEM(p{\cdot}a\,,p{\cdot}b)$ and $a(p{\cdot}a\ p{\cdot}b)^n \in \mathbf{t}pref(TR(S))$.

Hence, $a(p{\cdot}a\ p{\cdot}b)^n \in \mathbf{t}T$. We conclude that sem is not livelockfree.

In Example 2.3.3 we showed that S may be replaced by S' where $S' = (a\,;p{\cdot}a \mid a\,;b \mid p{\cdot}b\,;b)^*$, without affecting $TR(sem)$.

Choosing S' instead of S yields a component that is livelockfree. However, since $S(a\ p{\cdot}a\ p{\cdot}b\ ,T) = \{b\}$ and $S(a\ ,T\lceil\{a\ ,b\}) = \{a\ ,b\}$, it is not transparent.

The choice of $p{\cdot}b$ should be postponed until the environment initiates b.

(End of Example)

One may wonder why we did not choose the name *deterministic* instead of transparent. The reason is that there exists another form of nondeterminism that has not been discussed yet. It is the choice between (external) events that are initiated by a component.

Consider component *guess* defined by

com $guess(a\ ,b\ ,x\ ,y\ )\colon\ (a\ ;x\ \mid b\ ;y\ )^*$ **moc**

Suppose events a and b are to be initiated by the component, and events x and y are to be initiated by the environment. In [8] component *guess* is considered deterministic since the choice between a and b can be *observed* by the environment. We do, however, consider guess as a nondeterministic process, since some internal choice has to be made between a and b, and the environment does not have any knowledge about the way in which this choice is made.

Exercises

0. Determine which of the following components are transparent.

(1) **com** $sem_2(a\ ,b\ )\colon\ a\ ;(a\ ,b\ )^*$ **moc**

(2) **com** $sem_3(a\ ,b\ )$:

sub p : sem_1 **bus**

$(a\ ;p{\cdot}a\ )^*,(p{\cdot}b\ ;b\ )^*$

moc

(3) **com** $ex(a\ ,b\ )$:

sub $p\ ,q$: sem_1 **bus**

$(a\ ;(p{\cdot}a \mid q{\cdot}a\ ) \mid b\ ;(p{\cdot}b \mid q{\cdot}b\ ))^*$

moc

(4) **com** $sem_2(a,b)$:
sub p : sem_1 **bus**
$[p{\cdot}b = b]$ $(a;p{\cdot}a)^*$
moc

(5) **com** $sem(a,b)$:
sub p : sem **bus**
$a;(a;p{\cdot}a \mid b;a \mid b;p{\cdot}b)^*$
moc

(6) **com** $sem(a,b)$:
sub p : sem **bus**
$a;((b \mid p{\cdot}a;a);(p{\cdot}b;b \mid a))^*$
moc

(7) **com** $ex(a,b)$:
sub p : ex **bus**
$a;p{\cdot}b \mid p{\cdot}a;b$
moc

(8) **com** $run(a,b)$:
sub p : run **bus**
$(a;p{\cdot}a)^*$
moc

(End of Exercises)

6 Implementation Aspects

6.0 Introduction

In this chapter we discuss implementations of processes. Although we implement processes as (electrical) circuits, most concepts introduced do not depend on this choice. To a great extent we have been inspired by the work of Alain J. Martin ([12]).

This chapter differs from the previous ones: it is less formal and we do not provide proofs. We just present some ideas about implementations. Many of these still require further research.

The synchronization of events is solved by a so-called *four-phase handshaking protocol*. We do not distinguish between 'input symbols' and 'output symbols'. We do, however, make a distinction between events that are initiated by a component and those that are initiated by the environment of that component. It will turn out that the difference between these types of events is very small.

The circuits we derive are delay-insensitive in the sense that their behaviour does not depend on delays in wires and switching elements. We do not prove their delay-insensitivity formally.

6.1 Notations

For sequential programs we use the guarded command language with CSP-syntax (cf. [7]) :

$[\;\cdots\;]$ instead of **if** $\cdots$ **fi**

$*[\;\cdots\;]$ instead of **do** $\cdots$ **od**

Execution of an if-statement amounts to suspension of the program until one or more of the guards evaluate to true, after which a statement of which the guard is true is selected.

$*[\text{true} \rightarrow S]$ is abbreviated to $*[S]$ ('do S forever')

$[B \rightarrow \textbf{skip}\,]$ is abbreviated to $[B]$ ('wait until B')

With symbol a we associate a pair (a_o, a_i) of boolean variables. One may associate an 'output wire' with a_o and an 'input wire' with a_i. The value true will correspond to a high level voltage on the associated wire and the value false will correspond to a low level voltage on the associated wire. If x is such a boolean variable then

$x\uparrow$ means $x := \text{true}$ ('set x to a high level voltage')

$x\downarrow$ means $x := \text{false}$ ('set x to a low level voltage')

$[x]$ may be interpreted as 'wait until x has a high level voltage'.

$[\neg x]$ may be interpreted as 'wait until x has a low level voltage'.

Events are either *passive* or *active*. Active events are initiated by the mechanism, whereas passive events are initiated by the environment of the mechanism. Notice that the environment of a mechanism may be a mechanism as well.

Let a be a symbol. The occurrence of a in a process in which a is *passive* corresponds to the following sequence of actions in the implementation

$[a_i]\ ;\ a_o\uparrow\ ;\ [\neg a_i]\ ;\ a_o\downarrow$ (a passive)

After execution of $[a_i]\ ;\ a_o\uparrow$ event a 'has happened'.

The sequence $[\neg a_i]\ ;\ a_o\downarrow$ is used to return to the state $\neg a_o \wedge \neg a_i$.

The environment of the implementation performs a by the sequence

$a_i\uparrow\ ;\ [a_o]\ ;\ a_i\downarrow\ ;\ [\neg a_o]$ (environment of passive a)

The occurrence of a in a process in which a is *active* corresponds to

$a_o\uparrow\ ;\ [a_i]\ ;\ a_o\downarrow\ ;\ [\neg a_i]$ (a active)

After execution of $a_o\uparrow\ ;\ [a_i]$ event a 'has happened'.

The environment performs a by the sequence

$[a_o]\ ;\ a_i\uparrow\ ;\ [\neg a_o]\ ;\ a_i\downarrow$ (environment of active a)

Apparently,

> the pair (a_o, a_i) of a mechanism corresponds to the pair (a_i, a_o) of the environment. If (a_o, a_i) is active then (a_i, a_o) is passive and vice versa.

The synchronization thus obtained is called *four-phase handshaking*. For a synchronization protocol in which both mechanism and environment may initiate a we refer to [13].

The transformation of symbol a into such a sequence is called *handshaking expansion*.

Example 6.1.0

Consider component sem_1 defined by **com** $sem_1(a, b)$: $(a\ ;b)^*$ **moc**. If a is passive and b is active, handshaking expansion yields

$$*[[a_i] ; a_o\uparrow ; [\neg a_i] ; a_o\downarrow ; b_o\uparrow ; [b_i] ; b_o\downarrow ; [\neg b_i]]$$

If both a and b are active, we have

$$*[a_o\uparrow ; [a_i] ; a_o\downarrow ; [\neg a_i] ; b_o\uparrow ; [b_i] ; b_o\downarrow ; [\neg b_i]]$$

These programs express the behaviour of mechanisms with respect to a_i, a_o, b_i, and b_o. In the next section we realize such a mechanism.

6.2 Circuits

For the construction of our circuits we assume the existence of the following basic elements.

An *And-element* has two inputs and one output. If both inputs are true the output will be true, otherwise the output will be false. If x and y are inputs and z is output this is expressed by

$$x \wedge y \rightarrow z\uparrow$$
$$\neg x \vee \neg y \rightarrow z\downarrow$$

A *C-element*, cf. [15], has two inputs and one output. If the inputs have the same value then the output will also receive that value, otherwise the output does not change its value. This is expressed by

$$x \wedge y \rightarrow z\uparrow$$
$$\neg x \wedge \neg y \rightarrow z\downarrow$$

An *Inverter* has one input and one output. The output receives as its value the negation of the value of the input. It is expressed by

$$x \rightarrow z\downarrow$$
$$\neg x \rightarrow z\uparrow$$

Figure 6.0 shows how these basic elements are represented in pictures of circuits.

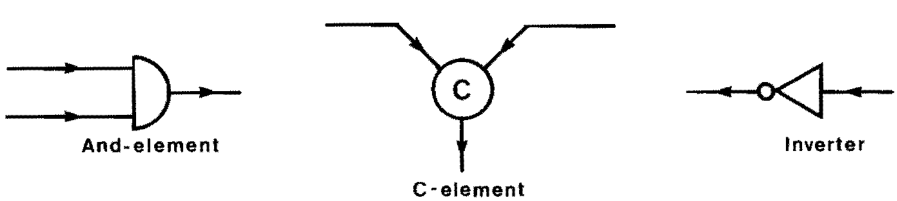


Figure 6.0

An Inverter in front of an And-element or C-element may be incorporated in that element, thus yielding a new basic element. The Inverter is drawn as a circle attached to the element. As an example, consider the specification

$$x \wedge \neg y \rightarrow z\uparrow$$
$$\neg x \wedge y \rightarrow z\downarrow$$

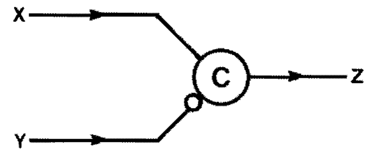


Figure 6.1

This denotes a C-element with inputs x and $\neg y$, and output z.
The corresponding circuit is shown in Figure 6.1

Example 6.2.0

We show an implementation of $SEM_1(a, b)$ where a is passive and b is active (cf. Example 6.1.0). Handshaking expansion yields

$$*[[a_i] ; a_o\uparrow ; [\neg a_i] ; a_o\downarrow ; b_o\uparrow ; [b_i] ; b_o\downarrow ; [\neg b_i]]$$

Initially we have $\neg a_o \wedge \neg a_i \wedge \neg b_o \wedge \neg b_i$. This state equals the state after $a_o\downarrow$. Hence, we need an additional variable, say x, to be able to trigger $b_o\uparrow$. Initially $\neg x$ holds. We propose

$$*[[a_i] ; a_o\uparrow ; x\uparrow ; [\neg a_i \wedge x] ; a_o\downarrow ; b_o\uparrow ; [b_i] ; x\downarrow ; [\neg x] ; b_o\downarrow ; [\neg b_i]]$$

We then have

(0) $a_i \wedge \neg x \wedge \neg b_i \rightarrow a_o\uparrow$
$\neg a_i \wedge x \rightarrow a_o\downarrow$

(1) $a_o \rightarrow x\uparrow$
$b_i \rightarrow x\downarrow$

(2) $\neg a_o \wedge x \rightarrow b_o\uparrow$
$\neg x \rightarrow b_o\downarrow$

Since in the period from $a_o\uparrow$ until $a_o\downarrow$ we have $\neg b_i$, we may transform (0) into

(0') $(a_i \wedge \neg x) \wedge \neg b_i \rightarrow a_o\uparrow$
$(\neg a_i \wedge x) \vee b_i \rightarrow a_o\downarrow$

This is a combination of a C-element and an And-element :

$$a_i \wedge \neg x \rightarrow y\uparrow$$
$$\neg a_i \wedge x \rightarrow y\downarrow$$

$$y \wedge \neg b_i \rightarrow a_o\uparrow$$
$$\neg y \vee b_i \rightarrow a_o\downarrow$$

Initially $\neg y$ holds.

A similar reasoning yields for (1) and (2)

(1') $\neg b_i \wedge a_o \rightarrow x\uparrow$ a C-element

$b_i \wedge \neg a_o \rightarrow x\downarrow$

(2') $\neg a_o \wedge x \rightarrow b_o\uparrow$ an And-element

$a_o \vee \neg x \rightarrow b_o\downarrow$

The ultimate circuit is shown in Figure 6.2 . The fat dots denote so-called *internal forks*. As in [12], we assume that the propagation delay in a forked wire is short compared to the delays in the basic elements.

Exercises

0. Consider the circuit shown in Figure 6.2 . What happens if the environment executes $b_i\uparrow ; a_i\uparrow$?

1. Derive a circuit that is an implementation of $SEM_1(a, b)$ with a and b active. Use
$*[a_o\uparrow ; [a_i] ; x\uparrow ; [x] ; a_o\downarrow ; [\neg a_i] ; b_o\uparrow ; [b_i] ; x\downarrow ; [\neg x] ; b_o\downarrow ; [\neg b_i]]$.
Derive from the resulting circuit an implementation with a passive and b active.

(End of Exercises)

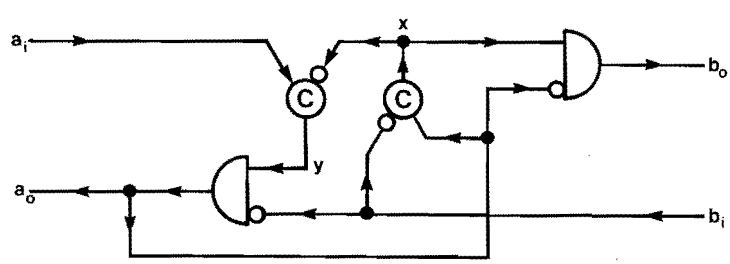


Figure 6.2

6.3 Active and Passive

Suppose we have a circuit corresponding to a process with passive a. We wish to connect an 'activator' (cf. Figure 6.3) to a_i and a_o such that its other two wires, p_i and p_o, yield an active version of a.

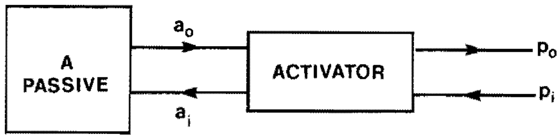


Figure 6.3

Action $p_o\uparrow$ is to be executed as soon as the original circuit is willing to acknowledge a_i. This yields

$$*[\,a_i\uparrow\,;[\,a_o\,]\,;p_o\uparrow\,;[\,p_i\,]\,;a_i\downarrow\,;[\,\neg a_o\,]\,;p_o\downarrow\,;[\,\neg p_i\,]\,]$$

Notice that the 'return to zero phase' has been moved to the right. We have

$$\neg p_i \rightarrow a_i\uparrow \qquad \text{(an Inverter)}$$
$$p_i \rightarrow a_i\downarrow$$

and

$$a_o \rightarrow p_o\uparrow \qquad \text{(a wire)}$$
$$\neg a_o \rightarrow p_o\downarrow$$

We conclude (cf. Figure 6.4)

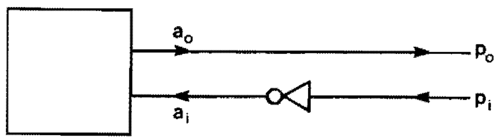


Figure 6.4

Theorem 6.3.0 (From passive to active)

If event a has been implemented as passive, by the pair (a_o, a_i), then an implementation with a active is obtained by placing an Inverter in front of a_i.

(End of Theorem)

Warning 6.3.1

Transforming a passive event into an active event in the way described above may introduce nondeterminism. This is shown by the following example.

Component *select* is defined by **com** *select*(a,b,x,y): $(a;x\mid b;y)^*$ **moc**. It is implemented such that a and b are passive, and x and y are active. The state graph of the implementation is shown in Figure 6.5 . We did not label all arcs : opposite sides of squares have the same label.

Notice that in state 4 a choice is made between $a_o\uparrow$ and $b_o\uparrow$. To implement this choice a new basic element, an Arbiter-element , is needed. We do not discuss nor introduce such an element. We assume that this implementation of *select* exists.

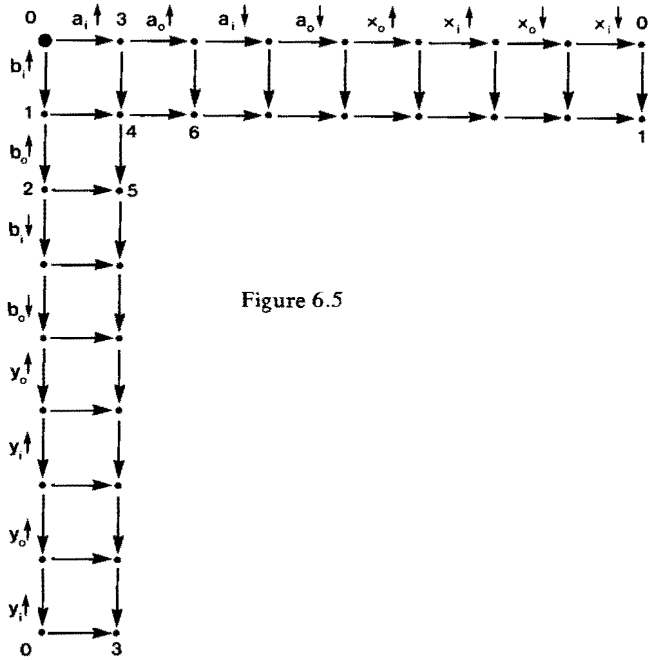


Figure 6.5

The environment behaves according to **com** *env*(a,b,x,y): $(a;x;b;y)^*$ **moc** with a and b active and x and y passive. The implementation never enters state 4 and the communication between environment and component behaves as expected.

If b is transformed into an active event, however, the following may happen. The inverter will cause $b_i\uparrow$ and the implementation will react with b_o. The environment will cause $a_i\uparrow$ and the mechanism enters state 5. The mechanism is suspended until $b_i\downarrow$ happens and the environment is suspended until $a_o\uparrow$ happens. Both events will not occur: the system is in a deadlock.

If both a and b are activated the situation is even worse. Depending on the speed of the inverters used, the mechanism will enter state 5 or state 6.

Activating a or b transforms the implementation into a nondeterministic mechanism in the sense that events may be initiated by the mechanism based on some decision unknown to the environment.

We conclude that activating a passive event is not allowed if the implementation makes a choice between the acceptance of that event and the acceptance of other events.

In this monograph we restrict ourselves to components that do not require the use of Arbiter-elements.

(End of Warning)

Suppose event a has been implemented as an active event. We wish to connect a 'passivator' (cf. Figure 6.6) to a_i and a_o such that its other two wires, p_i and p_o, yield a passive version of a.

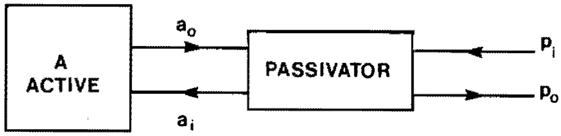


Figure 6.6

The original circuit may initiate a (by $a_o\uparrow$) as soon as that is possible. It is not acknowledged until the environment initiates a (by $p_i\uparrow$) as well. This yields

$[[a_o \wedge p_i] ; a_i\uparrow , p_o\uparrow ; [\neg a_o \wedge \neg p_i] ; a_i\downarrow , p_o\downarrow]$

which gives rise to a C-element.

We conclude (cf. Figure 6.7)

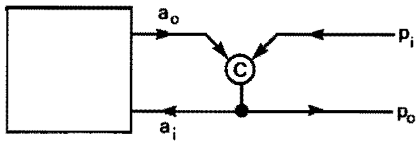


Figure 6.7

Theorem 6.3.2 (from active to passive)

> If event a has been implemented active by the pair (a_o , a_i) then an implementation with a passive is obtained by using a C-element as shown in Figure 6.7.

(End of Theorem)

(Transforming an active event into a passive event does not introduce nondeterminism.)

Consider the circuit shown in Figure 6.8. It consists of a C-element and a part called M. The occurrence of event a corresponds to the sequence

$a_i\uparrow , p_o\uparrow ; (a_o\uparrow ; a_i\downarrow) , (p_i\uparrow ; p_o\downarrow) ; a_o\downarrow , p_i\downarrow$

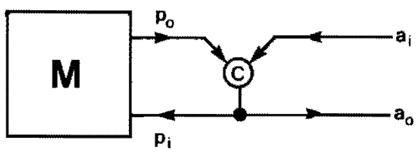


Figure 6.8

Projection on $\{a_o , a_i\}$ and $\{p_o , p_i\}$ yields respectively

$a_i\uparrow ; a_o\uparrow ; a_i\downarrow ; a_o\downarrow$ (passive)

$p_o\uparrow ; p_i\uparrow ; p_o\downarrow ; p_i\downarrow$ (active)

We conclude that removing the C-element transforms event a from passive into active. This transformation does not introduce nondeterminism.

In general we cannot transform an active event into a passive event by removing an Inverter. This is demonstrated by the following example.

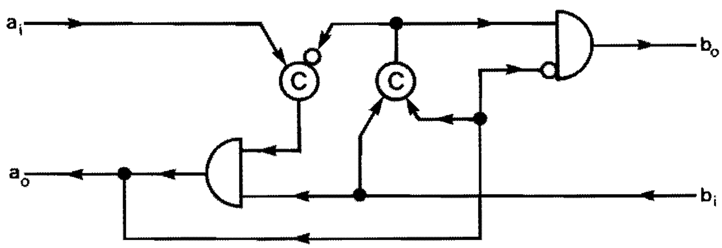


Figure 6.9

Example 6.3.3

In Section 6.2 we derived a circuit for $SEM_1(a,b)$ in which a is passive and b is active (cf. Figure 6.2). Removing the inverters to which b_i is connected yields the circuit shown in Figure 6.9 .

After $a_i\uparrow$ nothing will happen until $b_i\uparrow$ has occurred. This is not a valid implementation of $SEM_1(a,b)$.

(End of Example)

There is another remark on the difference between activators and passivators. In the next section we show how the composite of processes may be obtained by connecting wires that correspond to the same symbol. In view of the handshaking protocol we will connect events of different types only. If both implementations are active then a C-element is used (cf. Figure 6.10). Notice the symmetry of the connection (it is not known which of the implementations is turned into a passive one).

If both implementations are passive then a choice can be made (cf. Figure 6.11). This choice should be such that no nondeterminism is introduced.

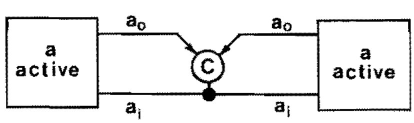


Figure 6.10

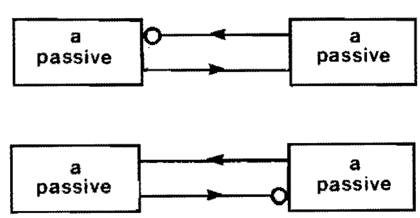


Figure 6.11

Exercises

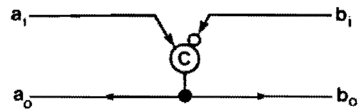


Figure 6.12

0. An implementation of $SEM_1(a, b)$ with a passive and b active may be obtained by implementing

$$*[[a_i] ; a_o\uparrow, b_o\uparrow ; [\neg a_i \wedge b_i] ; a_o\downarrow, b_o\downarrow ; [\neg b_i]]$$

which is obtained from the handshaking expansion and postponing of the second half of the expansion of a.

Show that this program yields the circuit shown in Figure 6.12 .

Should it be regarded as a valid implementation ?

Transform the circuit such that both a and b are active.

Transform the circuit such that both a and b are passive.

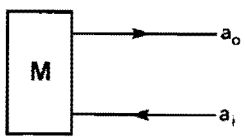


Figure 6.13

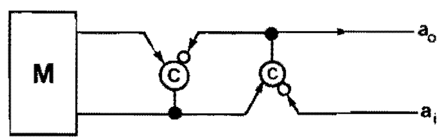


Figure 6.14

1. In Figure 6.13 the event corresponding to (a_o, a_i) is active.

The circuit of Figure 6.14 is obtained by subsequently passivating, activating, passivating, and activating (a_o, a_i).

Show that the two circuits are equivalent.

2. Two active events may be connected using a passivator. Is the (symmetric) circuit shown in Figure 6.15 an appropriate connection between passive events ?

(End of Exercises)

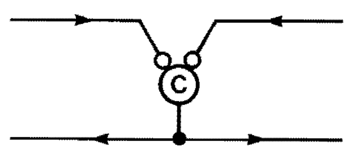


Figure 6.15

6.4 Components with subcomponents

In this section we discuss implementations of components that have subcomponents. We first consider components with command ϵ :

com $c(A)$:
  sub $p_0 : c_0, \ldots, p_{n-1} : c_{n-1}$ **bus**
  $[x_0 = y_0, \ldots, x_{m-1} = y_{m-1}]$
   ϵ
moc

Due to the restrictions imposed on program texts, each compound symbol occurs exactly once in the equalities. We assume that the subcomponents have already been implemented. Furthermore we assume that c is transparent.

With an element a of A two wires a_o and a_i are associated. Each element of A occurs in an equality. We connect the output wire of the symbol to which a is equated with a_o, and we connect its input wire to a_i.

Each equality between compound symbols yields a connection in the way described in Section 6.3 :

If the events have different types the connection is straightforward. If the events are both active a passivator is used. If the events are both passive one of these is activated.

In the last case one of the events should allow activation, i.e. activation should not cause nondeterminism. If such a choice is not possible we do not implement c (we consider the program as being wrong). Notice that activating may also be done by removing a passivator.

Finally, we may activate or passivate the implementation of the elements of A.

Example 6.4.0

Component run_1 is defined by **com** $run_1(a)$: a^* **moc**. Then $TR(run_1) = RUN(a)$.

With a passive, handshaking expansion yields

  $*[[a_i] ; a_o\uparrow ; [\neg a_i] ; a_o\downarrow]$ which is just a wire.

Component run_2, with $TR(run_2) = RUN(a, b)$, is defined by

com $run_2(a, b)$:
  sub p, q : run_1 **bus**
  $[p{\cdot}a = a, q{\cdot}a = b]$
   ϵ
moc

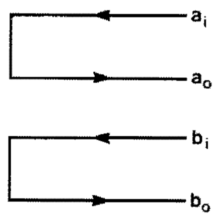


Figure 6.16

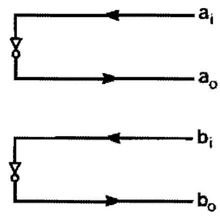


Figure 6.17

The method described above yields the circuit of Figure 6.16 .

An implementation with both a and b active is obtained by adding inverters, and is shown in Figure 6.17 .

(End of Example)

Example 6.4.1

An implementation of $SEM_2(a,b)$ with a passive and b active can be obtained from implementations of $SEM_1(a,b)$ with a passive and b active (cf. Example 5.4.1 and Example 6.2.0). It is based on the program

com $sem_2(a,b)$:
  sub p,q : sem_1 **bus**
  $[\,p{\cdot}a = a\,,\, p{\cdot}b = q{\cdot}a\,,\, q{\cdot}b = b\,]$
  ϵ
moc

The circuit is shown in Figure 6.18 .

(End of Example)

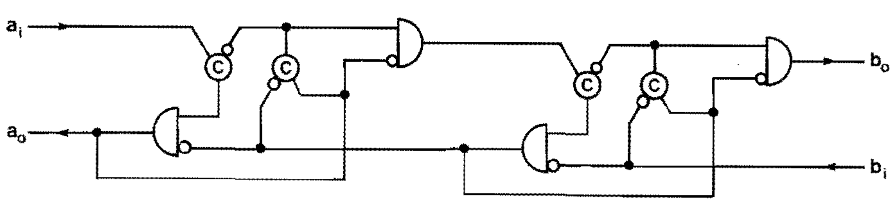


Figure 6.18

Finally, we consider components without equalities. Let c be defined by

> **com** $c(A)$:
> **sub** $p_0 : c_0, \ldots, p_{n-1} : c_{n-1}$ **bus**
> S
> **moc**

We assume that c is livelockfree. Notice that the subcomponents have only elements in common with S.

The implementation of $pref(TR(S))$ should be such that no (internal) nondeterminism arises (cf. Section 5.4). It turns out that the handshaking protocol often guarantees the absence of nondeterminism, as shown in the following example.

Example 6.4.2

We implement (cf. Example 5.4.3) component $wsem_2$ defined by

> **com** $wsem_2(a, b, c)$:
> **sub** $p, q : sem_1$ **bus**
> $(p{\cdot}a ; a \mid p{\cdot}b, q{\cdot}a ; b \mid q{\cdot}b ; c)^*$
> **moc**

with $TR(wsem_2) = SEM_1(a, b) \mathbf{w} SEM_1(b, c)$.

We assume that subcomponents $p{\cdot}sem_1$ and $q{\cdot}sem_1$ have been implemented with all events active.

We implement $wsem_2$ with a, b, and c passive. In accordance with the strategy explained in Example 5.4.3 an alternative of the command is executed if both subcomponent and environment initiate that alternative.

This yields the following expansions (output of a subcomponent is treated as input of the circuit corresponding to the command, and vice versa) :

$$*[[p{\cdot}a_o \wedge a_i \rightarrow p{\cdot}a_i\uparrow, a_o\uparrow ; [\neg p{\cdot}a_o \wedge \neg a_i] ; p{\cdot}a_i\downarrow, a_o\downarrow]]$$

$$*[[p{\cdot}b_o \wedge q{\cdot}a_o \wedge b_i \rightarrow p{\cdot}b_i\uparrow, q{\cdot}a_i\uparrow, b_o\uparrow ; [\neg p{\cdot}b_o \wedge \neg q{\cdot}a_o \wedge \neg b_i] ; p{\cdot}b_i\downarrow, q{\cdot}a_i\downarrow, b_o\downarrow]]$$

$$*[[q{\cdot}b_o \wedge c_i \rightarrow q{\cdot}b_i\uparrow, c_o\uparrow ; [\neg q{\cdot}b_o \wedge \neg c_i] ; q{\cdot}b_i\downarrow, c_o\downarrow]]$$

The first and the last one give rise to a C-element (with forked output). The middle one yields two C-elements.

The circuit is shown in Figure 6.19 .

We can activate a, b, and c by removing three C-elements (passivators). This yields Figure 6.20 .

Composing $wsem_2$ with $RUN(b)$, i.e. connecting b_i and b_o yields $SEM_2(a, c)$. This circuit, shown in Figure 6.21, is also obtained when connecting implementations of $SEM_1(a, b)$ in which both a and b are active.

(End of Example)

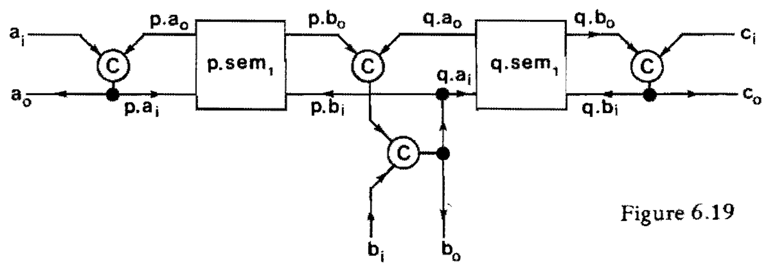


Figure 6.19

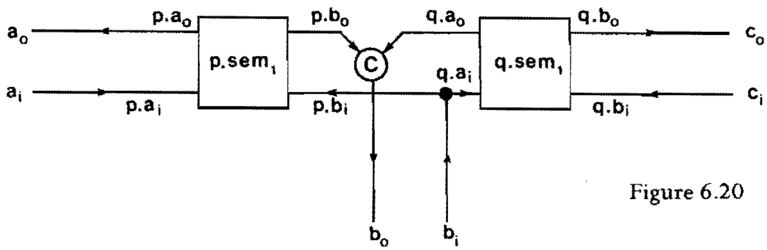


Figure 6.20

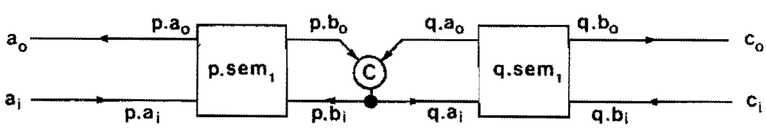


Figure 6.21

6.5 Final Remarks

We have shown how a certain class of processes can be implemented as delay-insensitive circuits. Nondeterminism did not play a role in the examples, due to the fact that we did not use Arbiter-elements. A treatment of these elements falls beyond the scope of this monograph. A typical process for which an Arbiter-element is needed is $SEM_1(a, \{b, c\})$ in which a, b, and c are passive.

A general method for the translation of commands into circuits has to be investigated. Since processes correspond to minimal deterministic state graphs, it is worthwhile to consider the translation from state graphs into circuits as well. For suggestions we refer to [17] and [19].

The concepts active and passive and the relations between these are very useful. The concepts 'input' and 'output' should be reserved for the description of processes on a higher level.

We have claimed that the circuits we derive are delay-insensitive in the sense that their behaviour does not depend on delays in wires and switching elements. A proof of such a claim must be based on a formalization of delay-insensitivity. In [21] delay-insensitivity is formalized and a classification of delay-insensitive processe is given.

7 Conclusions

In the preceding chapters we discussed several aspects of concurrent processes. The algebraic structure underlying these processes is relatively simple. Properties of operators like projection, weaving, and blending are easily formulated in terms of lattice theory.

Program texts provide a neat and concise way for the representation of processes. Moreover, the use of subcomponents admits a hierarchical way of constructing processes.

Phenomena like deadlock, livelock and nondeterminism have been succinctly expressed in terms of trace structures. This enabled us to formulate and prove many properties and theorems related to these concepts.

We conclude that Trace Theory is an adequate formalism for the description of concurrent processes.

Compared to other formalisms trace structures form a subclass of all possible processes. That subclass, however, is the class of mechanisms in which we are interested. We do not implement nondeterministic processes. On the other hand we do allow environments of processes to behave nondeterministically. In our formalism nondeterminism is captured by transparency. We showed that in the absence of livelock transparency is closed under intersection.

Due to the Conjunction-Weave Rule and the Composition Rule the derivation of programs from specifications is often straightforward. Our program notation is close to implementations. We showed examples of circuits that correspond to program texts. Again, the hierarchical structure of components plays an important role.

The derivation of circuits is based on four-phase handshaking and the notions of passive and active events. It turns out that nontransparency (i.e. internal nondeterminism) does not play an important role in these derivations.

External nondeterminism, however, cannot be resolved that easily. This form of nondeterminism is caused by transforming passive events into active events.

The derivation of circuits from programs requires further research.

Another topic that deserves further research is the communication of values.

Consider a mechanism that repeatedly inputs a value, say x, via channel a after which it outputs the value $2 \cdot x$ via channel b. The events the mechanism may be involved in are pairs (c, v) where c is the name of a channel and v is an integer value. If we do not take the values into account, the mechanism is specified by $SEM_1(a, b)$.

The trace structure $SEM_1(a, b)$ is called the *communication structure* of this mechanism. Besides the communication structure we have a *predicate* that relates the sequences of values transmitted via b to the sequence of values transmitted via a.

When the mechanism described above is composed with a mechanism that repeatedly inputs a value, say y, via channel b after which it outputs the value $3 \cdot y$ via channel c, we expect a mechanism that inputs via a a value x after which it outputs via c the value $6 \cdot x$. The communication structure of this composite will be $SEM_2(a, c)$, the composite of $SEM_1(a, b)$ and $SEM_1(b, c)$.

A theory needs to be developed that supports the reasoning above. Since output values have to be computed whereas input values have to be accepted only, we expect that in this theory a distinction between input and output will have to be made.

In this thesis we did not distinguish between input and output. Such a distinction would have complicated the theory needlessly. Notice that we postponed the introduction of 'active' and 'passive' until implementation aspects were considered.

Finally, it has been a pleasure to write this monograph :

a pleasure to build up the theory of the first chapters and a pleasure to apply it in the subsequent chapters.

We enjoyed the development of programs as well as the development of circuits. Actually, these activities turned out to be -in essence- very similar.

8 References

[0] Birkhoff, G.
Lattice Theory.
American Mathematical Society, Providence, 1967.
(AMS Colloquium Publications: vol. 25).

[1] Brookes, S.D. and A.W. Roscoe
An Improved Failures Model for Communicating Processes.
Seminar on Concurrency; ed. S.D. Brookes, A.W. Roscoe, G. Winskel.
Springer, Berlin, 1985.
(Lecture Notes in Computer Science: 197); pp. 281-305.

[2] Dijkstra, Edsger W.
Cooperating Sequential Processes.
Programming Languages; ed. F. Genuys.
Academic Press, New York, 1968; pp. 43-112.

[3] Dijkstra, Edsger W.
A Discipline of Programming.
Prentice-Hall, New York, 1976.

[4] Dijkstra, Edsger W.
Lecture Notes "Predicate transformers". (Draft).
Eindhoven University of Technology, 1982.
(EWD 835).

[5] Gilbert, William J.
Modern Algebra with Applications.
John Wiley & Sons, New York, 1976.

[6] Ginsburg, Seymour
The Mathematical Theory of Context-free Languages.
Mc Graw-Hill, New York, 1966.

[7] Hoare, C.A.R.
Communicating Sequential Processes.
Communications of the ACM **21** (1978); pp. 666-677.

[8] Hoare, C.A.R.
Communicating Sequential Processes.
Prentice Hall, New York, 1985.

[9] Hopcroft, J.E. and J.D. Ullman
Formal Languages and their Relation to Automata.
Addison-Wesley, New York, 1969.

[10] König, D.
Theorie der endlichen und unendlichen Graphen.
Chelsea, New York, 1950.

[11] Kuijpers, Evert P.J.
Recursive Components.
Eindhoven University of Technology, 1985.
(Master's Thesis).

[12] Martin, Alain J.
The Design of a Self-Timed Circuit for Distributed Mutual Exclusion.
Proceedings 1985 Chapel Hill Conference on VLSI; ed H. Fuchs.
Computer Science Press, Rockville, 1985; pp. 247-260.

[13] Martin, Alain J.
The Probe: an Addition to Communication Primitives.
Information Processing Letters **20** (1985); pp. 125-130.

[14] Mazurkiewicz, A.
Concurrent Program Schemes and Their Interpretation.
Report DAIMI, PB-78.
Aarhus University, 1977.

[15] Miller, R.E.
Switching Theory. Vol 2: Sequential Circuits and Machines; chapter 10.
John Wiley & Sons, New York, 1965.

[16] Milner, Robin
A Calculus of Communicating Systems.
Springer, Berlin, 1980.
(Lecture Notes in Computer Science: 92).

[17] Molnar, C.E. , T.P. Fang and F.U. Rosenberger
Synthesis of Delay-Insensitive Modules.
Proceedings 1985 Chapel Hill Conference on VLSI; ed. H.Fuchs.
Computer Science Press, Rockville, 1985; pp. 67-86.

[18] Rem, Martin
Concurrent Computations and VLSI Circuits.
Control Flow and Data Flow: Concepts of Distributed Programming; ed. M. Broy.
Springer, Berlin, 1985; pp. 399-437.

[19] Snepscheut, Jan L.A. van de
Trace Theory and VLSI Design.
Springer, Berlin, 1985.
(Lecture Notes in Computer Science: 200).

[20] Udding, Jan Tijmen
On recursively defined sets of traces.
Eindhoven University of Technology, 1983.
(THE Memorandum JTU0a).

[21] Udding, Jan Tijmen
Classification and composition of delay-insensitive circuits.
Ph.D.-thesis.
Eindhoven University of Technology, 1984.

Index

Samenvatting

Dit proefschrift bestaat uit twee delen. In het eerste deel (de hoofdstukken 1, 2 en 3) wordt een formalisme behandeld. In het tweede deel (de hoofdstukken 4, 5 en 6) wordt de ontwikkelde theorie toegepast.

De theorie, bekend onder de naam tracetheorie, levert een model voor het gedrag van een aantal samenwerkende mechanismen die gelijktijdig actief zijn. Een mechanisme wordt gekarakteriseerd door een paar:

de verzameling van mogelijke gebeurtenissen waarbij het mechanisme betrokken is en

de verzameling van mogelijke opeenvolgingen van dergelijke gebeurtenissen.

Gebeurtenissen worden voorgesteld door symbolen en de mogelijke opeenvolgingen worden voorgesteld door symboolrijen (traces). Een aldus verkregen paar heet een proces. Op de collectie van processen worden relaties en operaties gedefinieerd. Deze komen overeen met relaties tussen de corresponderende mechanismen en met, bijvoorbeeld, het samenstellen van mechanismen.

De verzameling van processen vormt een volledig tralie. Eigenschappen van de operaties worden beschreven in termen van tralietheorie.

Een proces kan worden weergegeven met een programmatekst. Een programma is niet alleen een compacte beschrijving van een proces maar geeft ook een idee over mogelijke implementaties. De afgeleide tralie-eigenschappen vormen een basis voor de behandeling van recursieve programma's.
Er worden regels gegeven waarmee het afleiden van een programma uit een gegeven specificatie wordt vergemakkelijkt. Als voorbeeld laten wij zien hoe een programma kan worden afgeleid dat past bij een gegeven contextvrije grammatica.

In hoofdstuk 4 komt het begrip deadlock aan de orde. Deadlock wordt gedefinieerd in termen van processen.

Het samenstellen van een aantal processen levert een nieuw proces. Bij dit proces onderscheiden we twee soorten symbolen:

interne symbolen die de onderlinge samenwerking van de delen betreffen

externe symbolen die de communicatie met de buitenwereld betreffen.

De uiteindelijke beschrijving van een mechanisme bevat geen informatie over de wisselwerking tussen de delen waaruit het mechanisme is opgebouwd. Deze beschrijving wordt verkregen door het proces te projecteren op de collectie externe symbolen. Bij projectie kan (intern) nondeterminisme ontstaan. In hoofdstuk 5 wordt het begrip transparantie gedefinieerd. Intern nondeterminisme treedt niet op indien men zich beperkt tot projectie op transparante verzamelingen. Het begrip livelock speelt hierbij een verrassende

rol. In de afwezigheid van livelock is transparantie gesloten onder doorsnede.
In hoofdstuk 5 wordt tevens aandacht geschonken aan de relatie tussen processen in ons formalisme en processen zoals deze zijn gedefinieerd door C.A.R. Hoare.

In hoofdstuk 6 beschouwen we implementaties. Deze zijn gebaseerd op een zogeheten 'four phase handshaking protocol'. Symbolen zijn actief dan wel passief. Actieve symbolen worden geïnitieerd door het mechanisme en passieve symbolen worden geïnitieerd door de omgeving. Het omzetten van actief naar passief en vice versa is relatief eenvoudig. Activeren van een passief symbool kan leiden tot nondeterminisme.
De schakelingen die worden afgeleid zijn vertragingsongevoelig in de zin dat hun gedrag niet afhangt van vertragingen in draden en schakelelementen.

Curriculum vitae

Op 3 september 1947 werd ik geboren te Eindhoven. Na het behalen van de diploma's MULO-B en HBS-B volgde de militaire dienst.

In 1969 begon de studie Wis - en Natuurkunde aan de universiteit van Utrecht. Het afstuderen vond plaats onder leiding van dr. J.D. Stegeman met als onderwerp Fouriertransformaties op locaal compacte groepen. In 1973 studeerde ik met lof af.

Na deze studie ben ik tot 1976 als wetenschappelijk medewerker werkzaam geweest bij de N.V. Philips, met als taak het ontwikkelen en onderhouden van IBM systeemsoftware. In deze periode ontstond mijn belangstelling voor formele methoden als gereedschap bij het programmeren.

Van 1976 tot 1982 was ik verbonden aan het Instituut voor Hoger Beroepsonderwijs te Eindhoven, eerst als docent wiskunde, later ook als docent informatica. Vanaf 1979 was ik belast met de leiding van de afdeling Informatica van de avond-HTS. In dezelfde periode werd samen met dr. J. van Tiel een aanvang gemaakt met de serie Voortgezette Wiskunde.

In de tussentijd volgde ik informaticacolleges aan de Technische Hogeschool Eindhoven. Dit resulteerde niet alleen in het behalen van de lesbevoegdheid Informatica maar ook in een hernieuwde wetenschappelijke belangstelling voor dit vakgebied. In 1982 werd ik benoemd tot wetenschappelijk medewerker in de vakgroep Informatica van de THE. Sindsdien heb ik me bezig gehouden met onderzoek en onderwijs op de gebieden programmeren, didactiek van het programmeren en parallellisme. Het laatste onderzoeksgebied heeft onder leiding van prof. dr. M. Rem geleid tot deze dissertatie.

STELLINGEN

behorend bij het proefschrift

A Formalism
for
Concurrent Processes

van

Anne Kaldewaij

Eindhoven,
6 mei 1986

1. Trace theorie is een adequaat formalisme voor het beschrijven van parallelle processen.

2. Met de in dit proefschrift gebruikte programmanotatie kan elke contextvrije taal worden beschreven.

3. De trace structuur $SEM_3(a, b)$ bevat F_n traces ter lengte n, waarbij F_n het n^{de} getal van Fibonacci is : $F_0 = 1$, $F_1 = 1$ en $F_{n+2} = F_n + F_{n+1}$.

4. Zij X een Hausdorff-ruimte en zij Y, $Y \subseteq X$, voorzien van de door X geïnduceerde topologie locaal compact. Dan geldt

   $\bar{Y} = X \Rightarrow Y$ is open in X

   Met behulp hiervan kan het bewijs van de Pontryagin dualiteitsstelling in [0] gecorrigeerd worden.

   [0] Walter Rudin, Fourier Analysis on Groups, Interscience Publishers, John Wiley & Sons, 1967.

5. Zij X een rij gehele getallen. Het minimum aantal stijgende deelrijen dat een partitie van X vormt is gelijk aan de maximale lengte van enige niet-stijgende deelrij van X.

   Lit. Anne Kaldewaij, On the decomposition of sequences into ascending subsequences, in Information Processing Letters, 21 (1985), p 69.

6. Intern nondeterminisme zoals beschreven in [1], speelt bij het implementeren van processen een geringe rol.

   [1] Hoare C.A.R., Communicating Sequential Processes,

7. Naast de zeven beperkt transponeerbare reeksen (les sept modes transpositions limitées) van de componist Olivier Messiaen bestaan er, afgezien van de chromatische reeks, nog precies drie, te weten

   C - Es - E - G - As - B - C

   C - Es - F - Fis - A - B - C

   C - D - F - Fis - Gis - B - C

   Lit. Olivier Messiaen, Techniques de mon langage musical, Alphonse Leduc, Paris

   Sietze Kaldewaij, Analyse van *Dieu parmi nous*, Utrechts Conservatorium, Mariaplaats 28 Utrecht, 1982.

8. Bij het informaticaonderwijs op middelbare scholen en in het hoger beroepsonderwijs dient men zich meer toe te leggen op het overdragen van inzichten. Apparatuur speelt daarbij een verwaarloosbare rol.

9. Het beoefenen van informatica vereist een groot abstractievermogen. Dit dient tot uiting te komen in de eerstejaars curricula van de universitaire informatica-opleidingen.

10. Bij het huidige wetenschapsbeleid trooste men zich met het gezegde 'sterke snoei geeft grote bloei'.