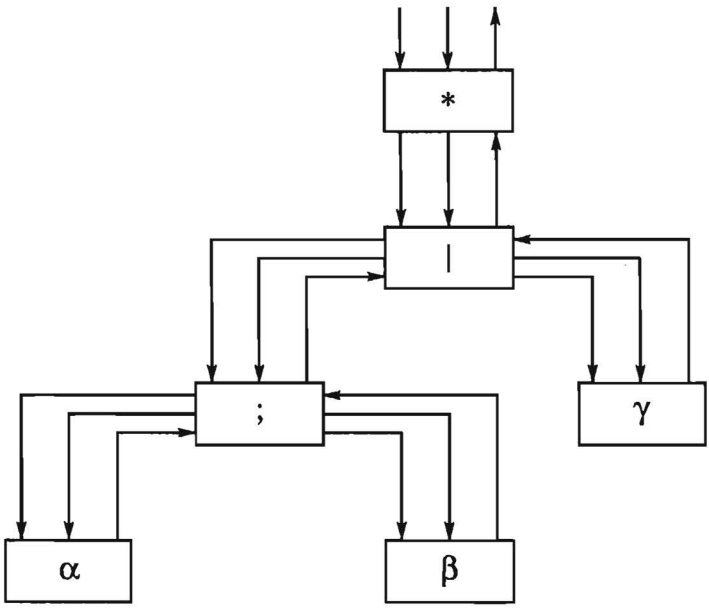
Parallel computations

Parallel Computations



Gerard Zwaan

Parallel Computations

Parallel Computations

PROEFSCHRIFT

TER VERKRIJGING VAN DE GRAAD VAN DOCTOR
AAN DE TECHNISCHE UNIVERSITEIT EINDHOVEN,
OP GEZAG VAN DE RECTOR MAGNIFICUS, PROF. IR. M. TELS,
VOOR EEN COMMISSIE AANGEWEZEN DOOR HET COLLEGE VAN DEKANEN
IN HET OPENBAAR TE VERDEDIGEN OP
VRIJDAG 20 JANUARI 1989 TE 16.00 UUR

DOOR

GERARD ZWAAN

GEBOREN TE BREDA

Dit proefschrift is goedgekeurd door
de promotor

prof. dr. M. Rem

en de copromotor

dr. A. Kaldewaij

Aan mijn ouders en Hanny

Author's present adress

Department of Mathematics and Computing Science
Eindhoven University of Technology
P.O. Box 513
5300 MB Eindhoven
The Netherlands

wsinzwaan@heitue5.bitnet (EARN, BITNET)
mcvax!eutws1!wsinswan (UUCP)

Contents

0 Introduction

0.0 Communication of values and parallel computations

In this thesis we discuss the description of the communication of values between mechanisms, and parallel computations. This is done using trace theory, a formalism for concurrent processes developed by Martin Rem ([Re85]), Jan L.A. van de Snepscheut ([Sn]), and Anne Kaldewaij ([Ka]).

In trace theory mechanisms are described by processes. Interaction of a mechanism with its environment is described by the occurrence of events. In order to describe communication of values we introduce events that are pairs consisting of a channel name and a message (value). This is done also by C.A.R. Hoare in Communicating Sequential Processes ([Ho]). Occurrence of pair $\langle c, m \rangle$ is interpreted as the passing of message m via channel c. By introducing these pairs one can fully describe communication of values within trace theory.

An important aspect is the extent to which the values that are communicated determine the communication behaviours of the processes. A process is said to be data independent if its communication behaviour is independent of the values that it sends and receives.

Consider the following program

com *adder*(**in** a, b : int, **out** c : int) :
    var x, y : int **rav**
    $(a?x\ ,\ b?y\ ;\ c!(x+y))^*$
moc

where $a?x$ denotes the receiving of a value via input channel a and the assignment of that value to local variable x, and $c!(x+y)$ denotes the sending via output channel c of the sum of the values of the variables x and y (the program notation used here is introduced in chapters 1 and 4). Program *adder* defines a data independent process that describes a mechanism that repeatedly computes the sum of pairs of integer values. Its communication behaviour is described by program

com $adder_{comm}(a, b, c) : (a\ ,\ b\ ;\ c)^*$ **moc**

Let $a(i)$ denote the i-th value that is communicated via channel a ($i \geq 0$). Let $b(i)$ and $c(i)$ be defined analogously. We then have for i, $i \geq 0$,

$$c(i) = a(i) + b(i)$$

This relation is called the input/output relation. Observe that the communication behaviour and the input/output relation are independent.
Next, consider a mechanism that filters negative values from an incoming stream of integer numbers, and that is described by the following program

com *filter*(**in** a : int, **out** b : int) :
  var x : int **rav**
  $(a?x\,;\ \textbf{if}\ x \geq 0 \rightarrow b!x \,[\!]\, x < 0 \rightarrow \varepsilon\ \textbf{fi})^*$
moc

The process defined by the above program is not data independent. Its communication behaviour cannot be described independently of the values that it sends and receives. After the occurrence of $\langle a, m \rangle$ for some $m < 0$ no communication via channel b can follow directly.
Data independence of processes allows us to express phenomena like deadlock and divergence ([Ka],[Ho]) in terms of the communication behaviours the of processes instead of in terms of the processes themselves.
Parallel computations are networks of processors or cells that can be described by processes. We are mainly interested in networks that can be characterized as follows

- the network is a regular arrangement of cells (for instance, a rectangular grid or a tree)
- communication between cells in the network and between cells and the environment of the network takes place via unidirectional channels
- cells are simple and communicate via a fixed number of channels with neighbour cells and/or the environment of the network (fixed means independent of the total number of cells)
- the communication behaviours of the cells are independent of the values that they send and receive, i.e. their processes are data independent
- cells synchronize by message passing only

Networks that satisfy the first four conditions are often referred to as systolic arrays ([Ku]). Systolic arrays usually have a global clock to synchronize the cells and, therefore, do not satisfy the fifth condition.

In this thesis we discuss a programming method with which one can derive programs from specifications that describe data independent processes ([Re87]). Data independence plays an important role in this method since it allows us to treat communication behaviours and input/output relations in isolation. The programs that are derived define networks of processes that satisfy the above conditions. The derived programs are formally proved to satisfy the given specifications and to have no divergence or deadlock.

0.1 Overview

In chapter 1 we give an overview of trace theory. Some new concepts are introduced, among them the notion of systems. A system describes a network of processes. Programs as defined in [Sn] and [Ka] denote a special class of systems. A recursive program defines a system consisting of an infinite number of processes. The process of a program is defined to be the process of the system specified by the program. As a result of this definition the process of a recursive program is equal to the least fixpoint of a recursive equation defined by the program, which was the definition in [Sn] and [Ka].

In chapter 2 we first discuss nondeterminism and divergence. The concepts of non-disabling and transparent sets of events (alphabets) are introduced (non-disabling corresponds to I_1 in [Ka]). Absence of divergence is characterized in several ways. A number of results is presented on non-disabling, non-divergent or transparent alphabets after composition and projection.
Secondly, we discuss termination and deadlock ([Ka]). If one wants to investigate the absence or presence of deadlock one may project on transparent alphabets that contain the common symbols.
Finally, we introduce the class of conservative processes and the class of cubic processes ([Ve86]). The latter is a subclass of the former. A process is conservative if its future behaviour depends only on the numbers of past events and not on their order. Cubic processes are the processes that can be described by partial orders on occurrences of events ([Ve86]). These classes are closed under composition and projection. Each subset of the alphabet of a conservative process is non-disabling. For cubic processes so-called sequence functions are introduced. A sequence function for a cubic process defines a subprocess that is cubic and that may be interpreted as a restricted (clocked) behaviour of the original process. Existence of a sequence function for a system of cubic processes implies the absence of deadlock. The notion of constant response time is defined in terms of sequence functions.

In chapter 3 we show how to model communication of values in terms of trace theory. Data independence is defined and is shown to be expressible in terms of transparence. Data independence is preserved by projection on alphabets that are non-disabling with

respect to the communication behaviour. Conditions are given such that composition of data independent processes yields a data independent process. Split specifications are introduced. The class of processes described by split specifications equals the class of data independent processes. In case of data independence phenomena like divergence and deadlock can be expressed in terms of communication behaviours only. Finally, we introduce channel order independence, expressing that the future behaviour of a process does not depend on the order in which the channels were used by the process. Its definition closely resembles that of conservative processes.

In chapter 4 we extend the program notation introduced in chapter 1 in order to express communication of values within programs. The notations have partly been adopted from CSP ([Ho]). In this chapter we make a distinction between input and output.

In chapter 5 we introduce, by means of examples, the programming method that was mentioned in the previous section.

0.2 Notation

We conclude with some remarks concerning notations that are used in this thesis.

Universal quantification is denoted by

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

where $\mathbf{A}$ is the quantifier, l is a list of bound variables, R is a predicate, and E is the quantified expression. Both R and E, in general, contain variables from l. Predicate R delineates the range of the bound variables and expression E is defined for values in that range. Likewise, we denote existential quantification, summation, union, and intersection using quantifiers $\mathbf{E}$, $\mathbf{S}$, $\cup$, and $\cap$, respectively.

For expressions E and G, an expression of the form $E \Rightarrow G$ will often be proved in a number of steps by introduction of intermediate expressions. For instance, if we can prove that $E \Rightarrow G$ by proving $E \equiv F$ and $F \Rightarrow G$, we record this proof as follows

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

In this way we avoid writing down intermediate expressions like F twice. These notations have been adopted from [Dij].

With $\mathcal{N}$ and $\mathcal{Z}$ we denote the set of natural numbers and the set of integer numbers, respectively.

1 Trace theory

1.0 Introduction

In this chapter we present an overview of trace theory. Most of the subjects that are also treated elsewhere ([Ka],[Sn]) will be described only briefly. The material presented in this chapter forms the basis for the rest of this thesis.

The basic notion of trace theory is that of processes. A process is a mathematical model of a mechanism. For instance, a variable that has no initial value interacts with its environment through two kinds of events, namely

a : a value is assigned to the variable

b : the variable returns its value

Sequences of a's and b's describing possible behaviours of the variable are $a\,b\,a\,b$, $a\,b\,b\,a\,a$, $a\,a\,b$ etc.. The sequence b is not a behaviour of the variable.

Finite sequences of events are called traces. A process describing a mechanism consists of the set of relevant events and the set of all possible traces. These sets are called the alphabet and trace set of the process, respectively. The process describing the variable has $\{a, b\}$ as its alphabet and the set of all sequences of a's and b's not starting with a b as its trace set.

It is clear that for every trace all initial parts thereof should also be allowed. Furthermore, the empty trace — meaning that the mechanism has not yet engaged in any event — should always be in the trace set. These two properties characterize processes.

The alphabet of a process contains the events that we are interested in. Depending on the aspects that are considered a mechanism may be described by different processes. If we are interested in the values that are assigned to the variable we might have chosen $\{a, b\} \times \mathcal{Z}$ ($\mathcal{Z}$ denotes the set of integer numbers) for the alphabet - assuming that the variable can store only integer values - where for $n \in \mathcal{Z}$

$\langle a, n\rangle$: the value n is assigned to the variable

$\langle b, n\rangle$: the variable returns the value n

Typical traces are $\langle a,4\rangle\,\langle b,4\rangle\,\langle b,4\rangle\,\langle a,-1\rangle\,\langle b,-1\rangle$, and $\langle a,0\rangle\,\langle a,1\rangle\,\langle b,1\rangle\,\langle a,0\rangle\,\langle b,0\rangle$.

In trace theory neither time nor speed plays a role. Events do not occur at a certain speed. Events are assumed to be atomic : they have no duration, they happen instantaneously, and they do not overlap.

Composition of mechanisms is represented by composition of corresponding processes. Interaction between mechanisms is assumed to be instantaneous. A common event takes place only if all processes having the event in their alphabets are able to engage in the event.

1.1 Trace calculus

With every kind of event a name (a symbol) is associated. We assume the existence of a set Ω of names. An element of Ω is called a *symbol.* A subset of Ω is called an *alphabet.*

The set of all finite-length sequences of symbols is denoted by Ω^*. The empty sequence ε is an element of Ω^*. An element of Ω^* is called a *trace.* A subset of Ω^* is called a *trace set.* For an alphabet A, set A^* is defined similarly. Notice that $\varnothing^* = \{\varepsilon\}$.

In our notation we employ 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 *length* of trace t, denoted by $\ell(t)$, is defined by

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

The *concatenation* of traces s and t is denoted by st. In order to save parentheses, concatenation is given the highest priority of all operators.

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

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

The *prefix closure* of a trace set X, denoted by $\mathrm{PREF}(X)$, is the trace set consisting of all prefixes of elements of X.

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

Trace set X is called *prefix-closed* if $X = \mathrm{PREF}(X)$.

The *projection* of trace t on alphabet A, denoted by $t\restriction A$, is obtained by removing from t all symbols that are not in A. It is defined as follows

$$\begin{array}{lcll} \varepsilon\restriction A & = & \varepsilon & \\ sa\restriction A & = & s\restriction A & a \notin A \\ sa\restriction A & = & (s\restriction A)a & a \in A \end{array}$$

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

A *trace structure* T is a pair $\langle A, X\rangle$ where A is an alphabet and X is a trace set such that $X \subseteq A^*$. We call A the alphabet of the trace structure and X the trace set of the trace structure.

The alphabet of a trace structure T is denoted by $\mathbf{a}T$ and its trace set by $\mathbf{t}T$. Notice that for a trace structure T we have $T = \langle \mathbf{a}T, \mathbf{t}T\rangle$.

We will denote trace structures using capital letters near the end of the Latin alphabet.

The *projection* of trace structure T on alphabet A, denoted by $T\restriction A$, is defined by

$$T\restriction A = \langle \mathbf{a}T \cap A, \mathbf{t}T\restriction A\rangle$$

The *prefix closure* of trace structure T, denoted by $\mathrm{PREF}(T)$, is defined by

$$\mathrm{PREF}(T) = \langle \mathbf{a}T, \mathrm{PREF}(\mathbf{t}T)\rangle$$

Trace structure T is called *prefix-closed* if $\mathbf{t}T$ is prefix-closed. Trace structure T is called *nonempty* if $\mathbf{t}T \neq \varnothing$.

A *process* is a nonempty prefix-closed trace structure. A process T is thought of as an abstraction of a mechanism. The alphabet of T is the set of relevant events the mechanism may engage in. It is assumed that events have no duration and that they do not overlap (events are said to be atomic). The state of the mechanism is described by the so called *current trace* being the sequence of events the mechanism has participated in. The behaviour of the mechanism in operation is described as follows. Initially, the current trace is empty. On occurrence of an event the current trace is extended with the symbol associated with that event. Clearly, the current trace should, at any moment, belong to the trace set of T. Moreover, if s is the current trace and $sa \in \mathbf{t}T$ then the event associated with a may happen. Notice that we do not make a distinction between events initiated by the mechanism and events initiated by the environment of the mechanism.

Example 1.1.0

A variable that has no initial value may be specified by a process T in the following way. The relevant events are

a : a value is assigned to the variable

b : the variable returns its value

Therefore, we choose $\mathbf{a}T = \{a, b\}$.
Any trace in $\{a, b\}^*$ that does not start with a b is a trace of T or, equivalently, any trace in $\{a, b\}^*$ that is either empty or starts with an a is a trace of T. The variable can be specified by

$$T = \langle\, \{a, b\}, \{\, t \mid t \in \{a, b\}^* \wedge (t = \varepsilon \vee a \leq t)\}\,\rangle$$

Notice that T is a process, i.e. a nonempty prefix-closed trace structure.

(End of Example)

Property 1.1.1

If T is a process and A is an alphabet then $T{\restriction}A$ is a process.

(End of Property)

We now define some special processes that play an important role. For alphabet A processes STOP(A) and RUN(A) are defined by

$$\begin{aligned} \text{STOP}(A) &= \langle A, \{\varepsilon\}\rangle \\ \text{RUN}(A) &= \langle A, A^*\rangle \end{aligned}$$

Process STOP($\varnothing$) (= RUN($\varnothing$)) is also denoted by STOP .
Let A and B be alphabets and let p and q be natural numbers. Process $\text{SYNC}_{p,q}(A, B)$ is defined by

$$\begin{aligned} &\text{SYNC}_{p,q}(A, B) \\ &= \langle\, A \cup B, \{\, t \mid t \in (A \cup B)^* \wedge (\mathbf{A}\, s : s \leq t : -q \leq \ell(s{\restriction}A) - \ell(s{\restriction}B) \leq p)\}\,\rangle \end{aligned}$$

Let k be a natural number. Process $\text{SEM}_k(A, B)$ is defined by

$$\text{SEM}_k(A, B) = \text{SYNC}_{k,0}(A, B)$$

We will often write $\text{SEM}_k(a, b)$ instead of $\text{SEM}_k(\{a\}, \{b\})$
and $\text{SYNC}_{p,q}(a, b)$ instead of $\text{SYNC}_{p,q}(\{a\}, \{b\})$.

Intersection, union, and inclusion are defined for trace structures having equal alphabets. Let $\langle A, X\rangle$ and $\langle A, Y\rangle$ be trace structures.

$$\langle A, X\rangle \subseteq \langle A, Y\rangle \;\equiv\; X \subseteq Y$$

Let X be a nonempty set of trace structures all having alphabet A. We define

$$(\cup T : T \in X : T) = \langle A, (\cup T : T \in X : \mathbf{t}T)\rangle$$

and

$$(\cap T : T \in X : T) = \langle A, (\cap T : T \in X : \mathbf{t}T)\rangle$$

The set of all processes with alphabet A is denoted by $\mathcal{T}(A)$. We have that $\langle \mathcal{T}(A), \subseteq\rangle$ is a complete lattice ([Bi]) with least element STOP(A) and greatest element RUN(A).

Theorem 1.1.2

If X is a nonempty set of processes from $\mathcal{T}(A)$,
then $(\cup T : T \in X : T)$ and $(\cap T : T \in X : T)$ are processes in $\mathcal{T}(A)$.

(End of Theorem)

Let T be a process and let $t \in \mathbf{t}T$. Process $after(t, T)$ is defined by

$$after(t, T) = \langle\, \mathbf{a}T, \{\, u \mid u \in \mathbf{a}T^* \wedge tu \in \mathbf{t}T\}\,\rangle$$

The *successor set* of trace t, denoted by $suc(t, T)$, is the set of all symbols that may follow t in $\mathbf{t}T$, i.e.

$$suc(t, T) = \{\, a \mid a \in \mathbf{a}T \wedge ta \in \mathbf{t}T\}$$

Property 1.1.3

Let T and U be processes with equal alphabets. Let $t \in \mathbf{t}T$ and $A \subseteq \mathbf{a}T$.

0 $a \in suc(t, T) \;\equiv\; a \in \mathbf{t}after(t, T)$

1 $suc(t, T) = suc(\varepsilon, after(t, T))$

2 $suc(t, T) \cap A \subseteq suc(t\restriction A, T\restriction A)$
$suc(t\restriction A, T\restriction A) = (\cup s : s \in \mathbf{t}T \wedge s\restriction A = t\restriction A : suc(s, T) \cap A)$

3 $after(t, T)\restriction A \subseteq after(t\restriction A, T\restriction A)$
$after(t\restriction A, T\restriction A) = (\cup s : s \in \mathbf{t}T \wedge s\restriction A = t\restriction A : after(s, T)\restriction A)$

4 $T \subseteq U \;\Rightarrow\; suc(t, T) \subseteq suc(t, U) \wedge after(t, T) \subseteq after(t, U)$

Let X be a nonempty set of processes with equal alphabets.
Let $u \in \mathbf{t}(\cup T : T \in X : T)$ and $v \in \mathbf{t}(\cap T : T \in X : T)$.

5 $suc(u, (\cup T : T \in X : T)) = (\cup T : T \in X \wedge u \in \mathbf{t}T : suc(u, T))$
$suc(v, (\cap T : T \in X : T)) = (\cap T : T \in X : suc(v, T))$
$after(u, (\cup T : T \in X : T)) = (\cup T : T \in X \wedge u \in \mathbf{t}T : after(u, T))$
$after(v, (\cap T : T \in X : T)) = (\cap T : T \in X : after(v, T))$

(End of Property)

If s and t are traces of process T such that $after(s, T) = after(t, T)$ we say that s and t belong to the same state of T. More formally the *states* of T are defined to be the equivalence classes of the equivalence relation $\overset{T}{\sim}$ defined on $\mathbf{t}T$ by

$$s \overset{T}{\sim} t \equiv after(s, T) = after(t, T)$$

With $[t]_T$ we denote the class to which t belongs. The set of all classes (states) of T is denoted by $[T]$.

Property 1.1.4

Let T be a process. Let $s, t \in \mathbf{t}T$.

0 $s \overset{T}{\sim} t \Rightarrow suc(s, T) = suc(t, T)$
1 $s \overset{T}{\sim} t \equiv (\mathbf{A}\, u : u \in \mathbf{a}T^* \wedge su \in \mathbf{t}T \wedge tu \in \mathbf{t}T : suc(su, T) = suc(tu, T))$

(End of Property)

The definitions of *after* and *suc* may be extended to the states of T:

$$after([t]_T, T) = after(t, T) \qquad \text{for } t \in \mathbf{t}T$$
$$suc([t]_T, T) = suc(t, T) \qquad \text{for } t \in \mathbf{t}T$$

We then have

$$after(\alpha, T) = after(\beta, T) \equiv \alpha = \beta \qquad \text{for } \alpha, \beta \in [T]$$

If T has a finite number of states, then T is called *regular*.

Theorem 1.1.5

Let T be a process and A be an alphabet. If T is regular then $T \restriction A$ is regular.

(End of Theorem)

Parallel composition of mechanisms is described in terms of the composition of the associated processes. Assume mechanism P is specified by process T and mechanism Q by process U. We specify the mechanism that is obtained by composing P and Q by a process V defined in terms of T and U. The alphabet of V is $\mathbf{a}T \cup \mathbf{a}U$. Let t be the current trace of the composite. Then $t\restriction\mathbf{a}T$ is the current trace of P and $t\restriction\mathbf{a}U$ the current trace of Q. Trace t can be extended with a symbol from $\mathbf{a}T \cap \mathbf{a}U$ if both P and Q can engage in the event associated with the symbol, i.e. both the current trace of P and the current trace of Q may be extended with the symbol. Trace t can be extended with a symbol from $\mathbf{a}T \div \mathbf{a}U$ if the mechanism having that symbol in its alphabet can engage in the associated event. From the above we infer that t is a trace of V if and only if $t\restriction\mathbf{a}T$ is a trace of T and $t\restriction\mathbf{a}U$ is a trace of U. This leads to the following definition.

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

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

It is easily shown that $T \mathbf{w} U$ is a process. Weaving is interleaving with synchronization on common symbols. Therefore, weaving of processes with disjoint alphabets amounts to just interleaving.

Example 1.1.6

$$\begin{aligned}
&\mathrm{RUN}(A) \mathbf{w} \mathrm{RUN}(B) = \mathrm{RUN}(A \cup B) \\
&\mathrm{SEM}_1(a,b) \mathbf{w} \mathrm{RUN}(\{c\}) = \mathrm{SEM}_1(\{a,c\},\{b,c\}) \\
&\mathrm{SYNC}_{p,q}(A,B) \mathbf{w} \mathrm{RUN}(C) = \mathrm{SYNC}_{p,q}(A \cup C \setminus B, B \cup C \setminus A) \\
&\mathrm{SEM}_1(a,b) \mathbf{w} \mathrm{SEM}_1(b,a) = \mathrm{STOP}(\{a,b\}) \\
&\mathrm{RUN}(A) \mathbf{w} \mathrm{STOP}(B) = \mathrm{RUN}(A \setminus B) \mathbf{w} \mathrm{STOP}(B) \\
&\qquad\qquad = \langle A \cup B, (A \setminus B)^* \rangle
\end{aligned}$$

(End of Example)

The following property shows that weaving is symmetric, idempotent, associative, and monotonic. Its unit element is STOP and its zero element is STOP(Ω).

Property 1.1.7

Let T, U, and V be processes. Let A be an alphabet.

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

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

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

3 $T \mathbf{w} \mathrm{STOP} = T$
$\mathbf{a}T \subseteq A \Rightarrow T \mathbf{w} \mathrm{STOP}(A) = \mathrm{STOP}(A)$
$T \mathbf{w} \mathrm{STOP}(\Omega) = \mathrm{STOP}(\Omega)$

4 $T \mathbf{w} T\restriction A = T$
$T \mathbf{w} U\restriction \mathbf{a}T \subseteq T$
$T \subseteq U \Rightarrow T \mathbf{w} U = T$

5 $T \mathbf{w} \mathrm{RUN}(A) = T \mathbf{w} \mathrm{RUN}(A \setminus \mathbf{a}T)$

6 $T \subseteq U \Rightarrow T \mathbf{w} V \subseteq U \mathbf{w} V$

(End of Property)

In view of the above property we can generalize the definition of weaving to arbitrary sets of processes. Let X be a set of processes. The weave of the processes in X, denoted by $(\mathbf{W}\, T : T \in X : T)$, is defined to be process

$$\langle (\cup T : T \in X : \mathbf{a}T)$$
$$, \{\, t \mid t \in (\cup T : T \in X : \mathbf{a}T)^* \wedge (\mathbf{A}\, T : T \in X : t\restriction \mathbf{a}T \in \mathbf{t}T) \,\} \rangle$$

Instead of $(\mathbf{W}\, T : T \in X : T)$ we also write $\mathbf{W}(X)$. The next property shows $\mathbf{W}$ to be a generalization of $\mathbf{w}$.

Property 1.1.8

Let X and Y be sets of processes. Let U be a process. Let A be an alphabet.

0 $\mathbf{W}(\varnothing) = \mathrm{STOP}$

1 $\mathbf{W}(\{U\}) = U$

2 $\mathbf{W}(X \cup Y) = \mathbf{W}(X) \mathbf{w} \mathbf{W}(Y)$

3 $X \subseteq Y \wedge A \subseteq \mathbf{a}\mathbf{W}(X) \Rightarrow \mathbf{W}(X)\restriction A \supseteq \mathbf{W}(Y)\restriction A$

(End of Property)

The following results show the relation between weaving and projection. Observe the important role played by the intersection of the alphabets. In the sequel T and U are processes and A and B are alphabets.

Property 1.1.9

0 $(T \mathbf{w} U)\restriction (A \cup \mathbf{a}T) \subseteq T \mathbf{w} U\restriction A$

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

(End of Property)

Theorem 1.1.10

0 $\mathbf{a}T \cap \mathbf{a}U \subseteq A \Rightarrow (T \mathbf{w} U)\restriction(A \cup \mathbf{a}T) = T \mathbf{w} U\restriction A$
1 $\mathbf{a}T \cap \mathbf{a}U \subseteq A \Rightarrow (T \mathbf{w} U)\restriction A = T\restriction A \mathbf{w} U\restriction A$

(End of Theorem)

Theorem 1.1.11

Let $A \subseteq \mathbf{a}T$, $B \subseteq \mathbf{a}U$, and $\mathbf{a}T \cap \mathbf{a}U = A \cap B$. Then

$$(T \mathbf{w} U)\restriction(A \cup B) = T\restriction A \mathbf{w} U\restriction B$$

Proof

We derive

$$\begin{array}{ll} & \mathbf{a}T \cap (A \cup B) \\ = & \quad \{ A \subseteq \mathbf{a}T \} \\ & A \cup (\mathbf{a}T \cap B) \\ = & \quad \{ \mathbf{a}T \cap B \subseteq \mathbf{a}T \cap \mathbf{a}U = A \cap B \} \\ & A \end{array}$$

Likewise, one can derive $\mathbf{a}U \cap (A \cup B) = B$.

$$\begin{array}{ll} & (T \mathbf{w} U)\restriction(A \cup B) \\ = & \quad \{ \mathbf{a}T \cap \mathbf{a}U = A \cap B, \text{theorem } 1.1.10.1 \} \\ & T\restriction(A \cup B) \mathbf{w} U\restriction(A \cup B) \\ = & \quad \{ T\restriction \mathbf{a}T = T, U\restriction \mathbf{a}U = U, \text{property projection, above derivation} \} \\ & T\restriction A \mathbf{w} U\restriction B \end{array}$$

(End of Proof)

Corollary 1.1.12

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

(End of Corollary)

The continuation of process $T \mathbf{w} U$ after trace t, process $after(t, T \mathbf{w} U)$, equals the weave of $after(t\restriction \mathbf{a}T, T)$ and $after(t\restriction \mathbf{a}U, U)$ as the following theorem expresses.

Theorem 1.1.13

Let T and U be processes. For every $t \in \mathbf{t}(T \mathbf{w} U)$ we have

$$after(t, T \mathbf{w} U) = after(t\restriction \mathbf{a}T, T) \mathbf{w} after(t\restriction \mathbf{a}U, U)$$

Proof

Let $t \in \mathbf{t}(T \mathbf{w} U)$. For trace u we have

$$\begin{array}{ll} & u \in \mathbf{t} after(t, T \mathbf{w} U) \\ = & \quad \{ \text{definition } after \} \\ & u \in \mathbf{a}(T \mathbf{w} U)^* \wedge tu \in \mathbf{t}(T \mathbf{w} U) \\ = & \quad \{ \text{definition weave, property projection} \} \\ & u \in (\mathbf{a}T \cup \mathbf{a}U)^* \wedge (t\restriction \mathbf{a}T)(u\restriction \mathbf{a}T) \in \mathbf{t}T \wedge (t\restriction \mathbf{a}U)(u\restriction \mathbf{a}U) \in \mathbf{t}U \\ = & \quad \{ \text{definition } after,\ t\restriction \mathbf{a}T \in \mathbf{t}T,\ t\restriction \mathbf{a}U \in \mathbf{t}U \} \\ & u \in (\mathbf{a}T \cup \mathbf{a}U)^* \wedge u\restriction \mathbf{a}T \in \mathbf{t} after(t\restriction \mathbf{a}T, T) \wedge u\restriction \mathbf{a}U \in \mathbf{t} after(t\restriction \mathbf{a}U, U) \\ = & \quad \{ \text{definition weave} \} \\ & u \in \mathbf{t}(after(t\restriction \mathbf{a}T, T) \mathbf{w} after(t\restriction \mathbf{a}U, U)) \end{array}$$

(End of Proof)

A direct consequence of the above theorem is

Theorem 1.1.14

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

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

(End of Theorem)

From theorem 1.1.14 we infer

Theorem 1.1.15

Let T and U be processes. 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.

(End of Theorem)

Corollary 1.1.16

If T and U are regular processes, then $T \mathbf{w} U$ is a regular process.

(End of Corollary)

Successor sets of $T \mathbf{w} U$ can be expressed in terms of successor sets of T and U.

Theorem 1.1.17

Let T and U be processes. For all $t \in \mathbf{t}(T \mathbf{w} U)$ we have

$$\begin{aligned} suc(t, T \mathbf{w} U) = \; & (suc(t \restriction \mathbf{a}T, T) \cap suc(t \restriction \mathbf{a}U, U)) \\ & \cup suc(t \restriction \mathbf{a}T, T) \setminus \mathbf{a}U \cup suc(t \restriction \mathbf{a}U, U) \setminus \mathbf{a}T \end{aligned}$$

Proof

Let $t \in \mathbf{t}(T \mathbf{w} U)$. We have

$$\begin{aligned} & suc(t, T \mathbf{w} U) \\ = \; & \quad \{ \text{definition successor set and weaving} \} \\ & \{ a \mid a \in \mathbf{a}T \cup \mathbf{a}U \wedge (ta) \restriction \mathbf{a}T \in \mathbf{t}T \wedge (ta) \restriction \mathbf{a}U \in \mathbf{t}U \} \\ = \; & \quad \{ \text{definition projection, } t \in \mathbf{t}(T \mathbf{w} U), \text{ set calculus} \} \\ & \{ a \mid a \in \mathbf{a}T \cap \mathbf{a}U \wedge (t \restriction \mathbf{a}T)a \in \mathbf{t}T \wedge (t \restriction \mathbf{a}U)a \in \mathbf{t}U \} \\ & \cup \{ a \mid a \in \mathbf{a}T \setminus \mathbf{a}U \wedge (t \restriction \mathbf{a}T)a \in \mathbf{t}T \} \cup \{ a \mid a \in \mathbf{a}U \setminus \mathbf{a}T \wedge (t \restriction \mathbf{a}U)a \in \mathbf{t}U \} \\ = \; & \quad \{ \text{definition successor set, set calculus} \} \\ & (suc(t \restriction \mathbf{a}T, T) \cap suc(t \restriction \mathbf{a}U, U)) \cup suc(t \restriction \mathbf{a}T, T) \setminus \mathbf{a}U \cup suc(t \restriction \mathbf{a}U, U) \setminus \mathbf{a}T \end{aligned}$$

(End of Proof)

Theorem 1.1.18

Let T and U be processes. Let A be an alphabet. If $\mathbf{a}T \cap \mathbf{a}U \subseteq A$ then for all $t \in \mathbf{t}(T \mathbf{w} U)$ we have

$$suc(t, T \mathbf{w} U) \subseteq A \equiv suc(t \restriction \mathbf{a}T, T) \cup suc(t \restriction \mathbf{a}U, U) \subseteq A$$

Proof

Assume $\mathbf{a}T \cap \mathbf{a}U \subseteq A$. Let $t \in \mathbf{t}(T \mathbf{w} U)$. We have

$$\begin{aligned} & suc(t, T \mathbf{w} U) \subseteq A \\ = \; & \quad \{ \text{theorem 1.1.17} \} \end{aligned}$$

$$
\begin{array}{ll}
 & (suc(t\restriction \mathbf{a}T, T) \cap suc(t\restriction \mathbf{a}U, U)) \cup suc(t\restriction \mathbf{a}T, T) \setminus \mathbf{a}U \cup suc(t\restriction \mathbf{a}U, U) \setminus \mathbf{a}T \subseteq A \\
= & \quad \{\, \mathbf{a}T \cap \mathbf{a}U \subseteq A \,\} \\
 & suc(t\restriction \mathbf{a}T, T) \setminus \mathbf{a}U \cup suc(t\restriction \mathbf{a}U, U) \setminus \mathbf{a}T \subseteq A \\
= & \quad \{\, suc(t\restriction \mathbf{a}T, T) \subseteq \mathbf{a}T,\ suc(t\restriction \mathbf{a}U, U) \subseteq \mathbf{a}U, \text{ set calculus} \,\} \\
 & (suc(t\restriction \mathbf{a}T, T) \cup suc(t\restriction \mathbf{a}U, U)) \setminus (\mathbf{a}T \cap \mathbf{a}U) \subseteq A \\
= & \quad \{\, \mathbf{a}T \cap \mathbf{a}U \subseteq A, \text{ set calculus} \,\} \\
 & suc(t\restriction \mathbf{a}T, T) \cup suc(t\restriction \mathbf{a}U, U) \subseteq A
\end{array}
$$

(End of Proof)

Corollary 1.1.19

Let T and U be processes. Let A be an alphabet. For all $t \in \mathbf{t}(T\,\mathbf{w}\,U)$ we have

$$suc(t, T\,\mathbf{w}\,U) \subseteq A \;\Rightarrow\; suc(t\restriction \mathbf{a}T, T) \cup suc(t\restriction \mathbf{a}U, U) \subseteq A \cup (\mathbf{a}T \cap \mathbf{a}U)$$

(End of Corollary)

1.2 Description of processes

In this section we present two ways in which processes may be described, by specifications and by a generalized form of regular expressions.

A *specification* of a process is a pair $\langle A, P\rangle$ where A is an alphabet and P is a predicate on A^* such that $P(\varepsilon)$ holds. The process specified by specification $\langle A, P\rangle$ is

$$\langle\, A, \{\, t \mid t \in A^* \wedge (\mathbf{A}\, s : s \leq t : P(s)) \,\}\,\rangle$$

It is easily shown that this trace structure is indeed a process. A specification will usually be written as $\langle\, A\,,\ t : P(t)\,\rangle$.

Example 1.2.0

0 Process $\mathrm{SEM}_1(a, b)$ is specified by
$\langle\, \{a, b\}\,,\ t : 0 \leq \ell(t\restriction a) - \ell(t\restriction b) \leq 1\,\rangle$

1 The process describing the variable in section 1.0 is specified by
$\langle\, \{a, b\}\,,\ t : t = \varepsilon \vee a \leq t\,\rangle$

(End of Example)

If $\langle A, P\rangle$ specifies process T, then $\mathbf{t}T$ is determined by

(0) $\varepsilon \in \mathbf{t}T$

(1) $t \in \mathbf{t}T \wedge a \in A \wedge P(ta) \Rightarrow ta \in \mathbf{t}T$

(2) $\mathbf{t}T$ contains no other traces than those that belong to it on account of (0) and (1)

The following theorem, called the Conjunction-Weave Rule (CW-Rule), shows the relation between the specifications of two processes and the specification of the weave of the two processes.

Theorem 1.2.1 Conjunction-Weave Rule

Let $\langle A, P\rangle$ and $\langle B, Q\rangle$ specify processes T and U respectively. Then

$$\langle\, A \cup B\,,\, t : P(t\restriction A) \wedge Q(t\restriction B)\,\rangle$$

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

(End of Theorem)

Commands form an extension of the notion of regular expressions. With each command S a trace structure $\mathrm{TR}(S)$ is associated. Commands and associated trace structures are defined inductively by the following rules.

- ε is a command and $\mathrm{TR}(\varepsilon) = \mathrm{STOP}$
- a is a command and $\mathrm{TR}(a) = \langle\{a\}, \{a\}\rangle$ for all symbols a
- if S is a command then S^* is a command and
  $\mathrm{TR}(S^*) = \langle\, \mathbf{a}\mathrm{TR}(S), (\mathbf{t}\mathrm{TR}(S))^*\rangle$
- if S and T are commands then $S \mid T$ is a command and
  $\mathrm{TR}(S \mid T) = \langle\, \mathbf{a}\mathrm{TR}(S) \cup \mathbf{a}\mathrm{TR}(T), \mathbf{t}\mathrm{TR}(S) \cup \mathbf{t}\mathrm{TR}(T)\,\rangle$
- if S and T are commands then $S\,;\,T$ is a command and
  $\mathrm{TR}(S\,;\,T) = \langle\, \mathbf{a}\mathrm{TR}(S) \cup \mathbf{a}\mathrm{TR}(T), \{\, uv \mid u \in \mathbf{t}\mathrm{TR}(S) \wedge v \in \mathbf{t}\mathrm{TR}(T)\}\,\rangle$
- if S and T are commands such that $\mathbf{a}\mathrm{TR}(S) \cap \mathbf{a}\mathrm{TR}(T) = \varnothing$ then $S\,,\,T$ is a command and
  $$\begin{aligned}\mathrm{TR}(S\,,\,T) = \langle\, & \mathbf{a}\mathrm{TR}(S) \cup \mathbf{a}\mathrm{TR}(T) \\ & ,\{\, t \mid\ t \in (\mathbf{a}\mathrm{TR}(S) \cup \mathbf{a}\mathrm{TR}(T))^* \\ & \qquad \wedge\ t\restriction \mathbf{a}\mathrm{TR}(S) \in \mathbf{t}\mathrm{TR}(S) \wedge t\restriction \mathbf{a}\mathrm{TR}(T) \in \mathbf{t}\mathrm{TR}(T)\}\,\rangle\end{aligned}$$
- if S is a command then S^0 is a command and $\mathrm{TR}(S^0) = \mathrm{STOP}(\mathbf{a}\mathrm{TR}(S))$

Observe that definition of $\mathrm{TR}(S\,,\,T)$ resembles the definition of the weave of two processes. Moreover, it differs from the definition in [Sn] and [Ka] where the condition

$\mathrm{aTR}(S) \cap \mathrm{aTR}(T) = \varnothing$ is not imposed. Listed in order of decreasing priority the operators are the star, the zero, the comma, the semicolon, and the bar. Commands are said to be equivalent ($S = T$) if and only if their trace structures coincide ($\mathrm{TR}(S) = \mathrm{TR}(T)$). Observe that for all commands S trace structure $\mathrm{TR}(S)$ is nonempty. Therefore $\mathrm{PREF}(\mathrm{TR}(S))$ is a process. The process $\mathrm{PR}(S)$ associated with command S is defined by

$$\mathrm{PR}(S) = \mathrm{PREF}(\mathrm{TR}(S))$$

Theorem 1.2.2

0 If S is a command, then $\mathrm{PR}(S)$ is a regular process.

1 If T is a regular process, then there exists a command S such that $T = \mathrm{PR}(S)$

(End of Theorem)

As a useful abbreviation we introduce for all commands S and all n, $n > 0$, the command S^n being the concatenation of n times the command S. More formally

$$\begin{aligned} S^1 &= S \\ S^{n+1} &= S\,;\, S^n \qquad\qquad n > 0 \end{aligned}$$

Example 1.2.3

$$\begin{aligned} \mathrm{SEM}_2(a,b) &= \mathrm{PR}(a\,;\,(a\,,\,b)^*) \\ \mathrm{SEM}_2(a,b) &= \mathrm{PR}((a\,;\,(a\,;\,b)^*\,;\,b)^*) \\ \mathrm{STOP}(\{a\}) &= \mathrm{PR}(a^0) \\ \mathrm{SYNC}_{1,1}(a,b) &= \mathrm{PR}((a\,,\,b)^*) \end{aligned}$$

(End of Example)

1.3 Systems

The composite of mechanisms can be described by the weave of the processes corresponding to these mechanisms. Sometimes, however, we want to retain the information on the partition into submechanisms. This can be done by describing the composite by a so-called *system* being a pair consisting of an alphabet and a set of processes. The set of processes consists of the processes corresponding to the (sub)mechanisms. The alphabet consists of the symbols that represent the *external events* of the composite.

This reflects that the other events of the composite are not observable from the outside. These events are called *internal events.*

More formally, a *system* S is a pair $\langle A, X\rangle$ where A is an alphabet and X is a set of processes such that $A \subseteq \mathbf{aW}(X)$. Alphabet A is called the (external) alphabet of the system and set X is called the set of processes, or *process set*, of the system.

Let S be a system. The external alphabet of S is denoted by $\mathbf{e}S$ and its process set is denoted by $\mathbf{p}S$. The condition imposed on the alphabets now reads $\mathbf{e}S \subseteq \mathbf{aW}(\mathbf{p}S)$. The (external) process of S, denoted by $\mathrm{PR}(S)$, is defined by $\mathrm{PR}(S) = \mathbf{W}(\mathbf{p}S)\restriction\mathbf{e}S$.

The external process of certain systems is given in the following theorem.

Theorem 1.3.0 ([Ka])

Let p, q, m, and n be natural numbers such that $p + q \geq 1$ and $m + n \geq 1$. Let A, B, and C be nonempty alphabets such that $A \cap B = \varnothing$ and $B \cap C = \varnothing$. Then

$$\mathrm{PR}(\langle A \div C, \{\mathrm{SYNC}_{p,q}(A,B), \mathrm{SYNC}_{m,n}(B,C)\}\rangle) = \mathrm{SYNC}_{p+m,q+n}(A \setminus C, C \setminus A)$$

(End of Theorem)

Corollary 1.3.1 ([Ka])

Let p and q be natural numbers such that $p + q \geq 1$. Let A, B, and C be nonempty alphabets that are mutually disjoint. Then

$$\mathrm{PR}(\langle A \cup C, \{\mathrm{SEM}_p(A,B), \mathrm{SEM}_q(B,C)\}\rangle) = \mathrm{SEM}_{p+q}(A,C)$$

(End of Corollary)

The set of all systems having external alphabet A is denoted by $\Sigma(A)$.

External symbols of a system can be hidden by projection of the system on an alphabet. They then become internal symbols. Projection has no effect on the process set of the system. The *projection* of system S on alphabet A, denoted by $S\restriction A$, is defined by

$$S\restriction A = \langle \mathbf{e}S \cap A, \mathbf{p}S\rangle$$

Notice that $\mathbf{p}(S\restriction A) = \mathbf{p}S$ and, hence, $\mathbf{aW}(\mathbf{p}S) = \mathbf{aW}(\mathbf{p}(S\restriction A))$.

Let S and T be systems. Then S and T describe networks of processes with external alphabets $\mathbf{e}S$ and $\mathbf{e}T$, respectively. Composition of systems S and T should reflect the composition of these networks of processes. Obviously, the only synchronization between both networks should be done on common external symbols. This implies

that common symbols should be external symbols of both networks. More formally, $\mathbf{aW}(\mathbf{p}S) \cap \mathbf{aW}(\mathbf{p}T) = \mathbf{e}S \cap \mathbf{e}T$. The external alphabet of the composite consists of the external symbols of both S and T. Furthermore, the network of the composite consists of both the processes of S and the processes of T, i.e. the process set of the composite of S and T is $\mathbf{p}S \cup \mathbf{p}T$. Therefore, for systems S and T satisfying $\mathbf{aW}(\mathbf{p}S) \cap \mathbf{aW}(\mathbf{p}T) = \mathbf{e}S \cap \mathbf{e}T$ the composite of S and T, denoted by $S \parallel T$ (read "S parallel T"), is defined by

$$S \parallel T = \langle \mathbf{e}S \cup \mathbf{e}T, \mathbf{p}S \cup \mathbf{p}T \rangle$$

Example 1.3.2

0 $\langle \varnothing, \varnothing \rangle$ is a system
$\mathbf{p}\langle \varnothing, \varnothing \rangle = \varnothing$
$\mathbf{e}\langle \varnothing, \varnothing \rangle = \varnothing$
$\mathrm{PR}(\langle \varnothing, \varnothing \rangle) = \mathbf{W}(\varnothing)\restriction\varnothing = \mathrm{STOP}$

1 Let $S = \langle \{a, b\}, \{\mathrm{SEM}_1(a, b)\} \rangle$ and $T = \langle \{b, c\}, \{\mathrm{SEM}_1(b, c)\} \rangle$.
We have
$S \parallel T = \langle \{a, b, c\}, \{\mathrm{SEM}_1(a, b), \mathrm{SEM}_1(b, c)\} \rangle$,
$(S \parallel T)\restriction\{a, c\} = \langle \{a, c\}, \{\mathrm{SEM}_1(a, b), \mathrm{SEM}_1(b, c)\} \rangle$,
and, by corollary 1.3.1,
$\mathrm{PR}((S \parallel T)\restriction\{a, c\}) = \mathrm{SEM}_2(a, c)$

(End of Example)

Below we list a number of properties of systems and their processes.

Property 1.3.3

Let R,S, and T be systems such that $\mathbf{aW}(\mathbf{p}R) \cap \mathbf{aW}(\mathbf{p}S) = \mathbf{e}R \cap \mathbf{e}S$, $\mathbf{aW}(\mathbf{p}R) \cap \mathbf{aW}(\mathbf{p}T) = \mathbf{e}R \cap \mathbf{e}T$, and $\mathbf{aW}(\mathbf{p}S) \cap \mathbf{aW}(\mathbf{p}T) = \mathbf{e}S \cap \mathbf{e}T$. Let A be an alphabet.

0 $\langle \varnothing, \varnothing \rangle \parallel R = R$

1 $R \parallel S = S \parallel R$

2 $(R \parallel S) \parallel T = R \parallel (S \parallel T)$

3 $\mathrm{PR}(S\restriction\varnothing) = \mathrm{STOP}$

4 $\mathbf{e}S = \mathbf{e}T \wedge \mathbf{p}S \subseteq \mathbf{p}T \Rightarrow \mathrm{PR}(S) \supseteq \mathrm{PR}(T)$

5 $\mathrm{PR}(S\restriction A) = \mathrm{PR}(S)\restriction A$

6 $\mathrm{PR}(S \parallel T) = \mathrm{PR}(S)\ \mathbf{w}\ \mathrm{PR}(T)$

7 $\mathbf{e}S \cap \mathbf{e}T \subseteq A \ \Rightarrow\ (S \parallel T)\restriction A = (S\restriction A) \parallel (T\restriction A)$

Proof

4 See property 1.1.8.3

5

$$
\begin{array}{ll}
 & \mathrm{PR}(S\restriction A) \\
= & \quad \{\text{ definition PR }\} \\
 & \mathbf{W}(\mathbf{p}(S\restriction A))\restriction\mathbf{e}(S\restriction A) \\
= & \quad \{\text{ definition projection }\} \\
 & \mathbf{W}(\mathbf{p}S)\restriction(\mathbf{e}S \cap A) \\
= & \quad \{\text{ property projection, definition PR }\} \\
 & \mathrm{PR}(S)\restriction A
\end{array}
$$

6

$$
\begin{array}{ll}
 & \mathrm{PR}(S \parallel T) \\
= & \quad \{\text{ definition PR }\} \\
 & \mathbf{W}(\mathbf{p}(S \parallel T))\restriction\mathbf{e}(S \parallel T) \\
= & \quad \{\text{ definition composition }\} \\
 & \mathbf{W}(\mathbf{p}S \cup \mathbf{p}T)\restriction(\mathbf{e}S \cup \mathbf{e}T) \\
= & \quad \{\text{ property 1.1.8.2 }\} \\
 & (\mathbf{W}(\mathbf{p}S)\ \mathbf{w}\ \mathbf{W}(\mathbf{p}T))\restriction(\mathbf{e}S \cup \mathbf{e}T) \\
= & \quad \{\ \mathbf{aW}(\mathbf{p}S) \cap \mathbf{aW}(\mathbf{p}T) = \mathbf{e}S \cap \mathbf{e}T\text{, theorem 1.1.11 }\} \\
 & \mathbf{W}(\mathbf{p}S)\restriction\mathbf{e}S\ \mathbf{w}\ \mathbf{W}(\mathbf{p}T)\restriction\mathbf{e}T \\
= & \quad \{\text{ definition PR }\} \\
 & \mathrm{PR}(S)\ \mathbf{w}\ \mathrm{PR}(T)
\end{array}
$$

7 Assume $\mathbf{e}S \cap \mathbf{e}T \subseteq A$. We derive

$$
\begin{array}{ll}
 & \mathbf{aW}(\mathbf{p}(S\restriction A)) \cap \mathbf{aW}(\mathbf{p}(T\restriction A)) = \mathbf{e}(S\restriction A) \cap \mathbf{e}(T\restriction A) \\
= & \quad \{\text{ definition projection }\} \\
 & \mathbf{aW}(\mathbf{p}S) \cap \mathbf{aW}(\mathbf{p}T) = \mathbf{e}S \cap A \cap \mathbf{e}T \cap A \\
= & \quad \{\text{ assumption }\} \\
 & \mathbf{e}S \cap \mathbf{e}T = \mathbf{e}S \cap \mathbf{e}T \cap A \\
= & \quad \{\text{ set calculus }\} \\
 & \mathbf{e}S \cap \mathbf{e}T \subseteq A
\end{array}
$$

Hence, systems $S\restriction A$ and $T\restriction A$ have only external symbols in common and can be composed. The equality between $(S \parallel T)\restriction A$ and $(S\restriction A) \parallel (T\restriction A)$ follows immediately.

(End of Proof)

We conclude with two definitions. If T is a process then the system corresponding to T, denoted by $sys(T)$, is defined by $sys(T) = \langle \mathbf{a}T, \{T\}\rangle$. Notice that $\mathrm{PR}(sys(T)) = T$. If S is a command then the system corresponding to S, denoted by $sys(S)$, is defined by $sys(S) = \langle \mathbf{a}\mathrm{PR}(S), \{\mathrm{PR}(S)\}\rangle$. Notice that $\mathrm{PR}(sys(S)) = \mathrm{PR}(S)$.

1.4 A program notation

In this section we introduce a program notation similar to the one in [Ka]. Here, however, a program — also called a component — defines a system. The process of a component will be defined to be the process of the corresponding system. This results in a process equal to the one obtained by applying the definition from [Ka].

Before introducing the program notation we have to say somewhat more on the nature of the set of symbols Ω. We assume the existence of a set Ω_s. An element of Ω_s is called a *simple* symbol. For $n > 0$ the set Ω_s^n is defined to be the set of all n-tuples of symbols in Ω_s. We assume that

$$\Omega = (\cup\, n : n > 0 : \Omega_s^n)$$

An element of $\Omega \setminus \Omega_s$ is called a *compound* symbol. Element $(a_0, a_1, \ldots, a_{n-1})$ of Ω is denoted by $a_0{\cdot}a_1{\cdot}\ldots{\cdot}a_{n-1}$. If a and b are symbols then $a{\cdot}b$ is a symbol as well. Let p be a symbol. With p we can associate a function in $\Omega \to \Omega$ that maps each symbol a onto symbol $p{\cdot}a$. This function is denoted by $p{\cdot}$. Notice that function $p{\cdot}$ is injective. For $n \geq 0$ function $(p{\cdot})^n$ is defined inductively by

$$\begin{array}{ll} (p{\cdot})^0 a = a & \text{for } a \in \Omega \\ (p{\cdot})^{i+1} a = p{\cdot}(p{\cdot})^i a & \text{for } a \in \Omega,\ i \geq 0 \end{array}$$

Furthermore, we define

$$\begin{array}{ll} p{\cdot}A = \{p{\cdot}a \mid a \in A\} & \text{for } A \subseteq \Omega \\ p{\cdot}\varepsilon = \varepsilon & \\ p{\cdot}(ta) = (p{\cdot}t)p{\cdot}a & \text{for } t \in \Omega^*, a \in \Omega \\ p{\cdot}X = \{p{\cdot}t \mid t \in X\} & \text{for } X \subseteq \Omega^* \\ p{\cdot}T = \langle p{\cdot}\mathbf{a}T, p{\cdot}\mathbf{t}T\rangle & \text{for } T \text{ a trace structure} \\ p{\cdot}S = \langle p{\cdot}\mathbf{e}S, \{p{\cdot}T \mid T \in \mathbf{p}S\}\rangle & \text{for } S \text{ a system} \end{array}$$

The program

$$\textbf{com } c(A) : S \textbf{ moc}$$

denotes a component without subcomponents, where c is the name of the component, A is a finite alphabet, the external alphabet of the component, and S is a command. The only restrictions imposed on such a program text are $A = \mathbf{a}\mathrm{PR}(S)$ and A consists of simple symbols only. The system of component c, denoted by $sys(c)$, is defined by

$$sys(c) = sys(S)\restriction A$$

Notice that $sys(c) = \langle A, \{\mathrm{PR}(S)\}\rangle = sys(S)$. The process of component c, denoted by $\mathrm{PR}(c)$, is defined by $\mathrm{PR}(c) = \mathrm{PR}(sys(c))$. Notice that $\mathrm{PR}(c) = \mathrm{PR}(S)$.
A component with subcomponents is denoted by the program

$$\begin{array}{l}\textbf{com } c(A) : \\ \quad \textbf{sub } p_0 : c_0, p_1 : c_1, \ldots, p_{n-1} : c_{n-1} \textbf{ bus} \\ \quad [x_0 = y_0, x_1 = y_1, \ldots, x_{m-1} = y_{m-1}] \\ \quad S \\ \textbf{moc}\end{array}$$

where c is the name of the component, A is a finite alphabet, the external alphabet of the component, S is a command, and c_0, c_1, ..., and c_{n-1} are previously defined components, called the subcomponents of c and having names p_0, p_1, ..., and p_{n-1} respectively. We require that A contains simple symbols only and that $p_0, p_1, \ldots, p_{n-1}$ are n distinct, simple symbols. With subcomponent p_i system $p_i{\cdot}sys(c_i)$ is associated. The set

$$B = (\cup i : 0 \leq i < n : \mathbf{e}p_i{\cdot}sys(c_i))$$

is called the set of internal symbols of component c (notice that B consists of the external symbols of all subcomponents). The equalities represent (internal) connections. An internal connection links two subcomponents or a subcomponent and the external alphabet. Since we do not want external symbols of the same subcomponent to be connected either directly or indirectly we impose some restrictions. First, we define

$$\mathcal{C} = \{A\} \cup \{\, \mathbf{e}p_i{\cdot}sys(c_i) \mid 0 \leq i < n\}$$

and observe that $\mathcal{C}$ is collection of $n + 1$ mutually disjoint alphabets. The restrictions are as follows

- $(\mathbf{A}\, j : 0 \leq j < m : x_j \in B)$
- $(\mathbf{A}\, j : 0 \leq j < m : y_j \in B \cup A)$
- $|\{\, x_i \mid 0 \leq i < m \,\}| = m$
- $\{\, x_i \mid 0 \leq i < m \,\} \cap \{\, y_j \mid 0 \leq j < m \,\} = \varnothing$
- for all j, $0 \leq j < m$, symbols x_j and y_j belong to two different alphabets in $\mathcal{C}$
- for all i and j, $0 \leq i < j < m$, such that $y_i = y_j$ symbols x_i and x_j belong to two different alphabets in $\mathcal{C}$

Furthermore, we require that every external symbol appears in the command S or is connected to an internal symbol

$$A \subseteq \mathbf{a}\mathrm{PR}(S) \cup \{\, y_j \mid 0 \leq j < m \}$$

The alphabet of command S should consist of external symbols and internal symbols not in $\{\, x_j \mid 0 \leq j < m \,\}$, i.e.

$$\mathbf{a}\mathrm{PR}(S) \subseteq A \cup B \setminus \{\, x_j \mid 0 \leq j < m \,\}$$

We now define the system of component c, denoted by $sys(c)$, by

$$sys(c) = (\, (\,\|\ i : 0 \leq i < n : (p_i{\cdot}sys(c_i))^{x_0,x_1,\ldots,x_{m-1}}_{y_0,y_1,\ldots,y_{m-1}}) \parallel sys(S)\,) \upharpoonright A$$

Notice that due the above restrictions $sys(c)$ is well defined. With $(p_i{\cdot}sys(c_i))^{x_0,x_1,\ldots,x_{m-1}}_{y_0,y_1,\ldots,y_{m-1}}$ we denote system $p_i{\cdot}sys(c_i)$ in which every occurrence of symbol x_j has been substituted by symbol y_j for all j, $0 \leq j < m$. In this way we will, in general, denote substitution (renaming). The process of component c, denoted by $\mathrm{PR}(c)$, is defined by $\mathrm{PR}(c) = \mathrm{PR}(sys(c))$. Notice that $\mathbf{e}sys(c) = A$ and $\mathbf{a}\mathrm{PR}(c) = A$. We derive

$$\begin{array}{ll} & \mathrm{PR}(c) \\ = & \quad \{\, \text{definition} \,\} \\ & \mathrm{PR}(\,(\,\|\ i : 0 \leq i < n : (p_i{\cdot}sys(c_i))^{x_0,x_1,\ldots,x_{m-1}}_{y_0,y_1,\ldots,y_{m-1}}) \parallel sys(S)\,) \upharpoonright A\,) \\ = & \quad \{\, \text{property 1.3.3} \,\} \\ & (\,(\mathbf{W}\, i : 0 \leq i < n : \mathrm{PR}((p_i{\cdot}sys(c_i))^{x_0,x_1,\ldots,x_{m-1}}_{y_0,y_1,\ldots,y_{m-1}}))\ \mathbf{w}\ \mathrm{PR}(S)\,) \upharpoonright A \\ = & \quad \{\, \text{note 1.4.0} \,\} \\ & (\,(\mathbf{W}\, i : 0 \leq i < n : (p_i{\cdot}\mathrm{PR}(c_i))^{x_0,x_1,\ldots,x_{m-1}}_{y_0,y_1,\ldots,y_{m-1}})\ \mathbf{w}\ \mathrm{PR}(S)\,) \upharpoonright A \end{array}$$

Note 1.4.0

In this note we show that weaving and substitution commute and that projection and substitution commute due to the restrictions imposed on the component. Let $0 \leq i < n$. Let $i_0, i_1, \ldots, i_{k-1}$ be the subsequence of $0, 1, \ldots, m-1$ such that

$$\{ x_{i_s} \mid 0 \leq s < k \} = \{ x_j \mid 0 \leq j < m \wedge x_j \in \mathbf{e}(p_i{\cdot}sys(c_i)) \}$$

Notice that

$$(*) \quad (\mathbf{A}\, s,t : 0 \leq s < t < k : y_{i_s} \neq y_{i_t}) \wedge (\mathbf{A}\, s : 0 \leq s < k : y_{i_s} \notin \mathbf{e}(p_i{\cdot}sys(c_i)))$$

We derive

$$\begin{array}{ll}
 & \mathrm{PR}(\, (p_i{\cdot}sys(c_i))^{x_0,x_1,\ldots,x_{m-1}}_{y_0,y_1,\ldots,y_{m-1}}\,) \\
= & \quad \{ \text{definition process of a system, definition of } i_0, i_1, \ldots, i_{k-1} \} \\
 & (\mathbf{W}\, T : T \in \mathbf{p}(p_i{\cdot}sys(c_i))^{x_{i_0},x_{i_1},\ldots,x_{i_{k-1}}}_{y_{i_0},y_{i_1},\ldots,y_{i_{k-1}}} : T) \restriction \mathbf{e}(p_i{\cdot}sys(c_i))^{x_{i_0},x_{i_1},\ldots,x_{i_{k-1}}}_{y_{i_0},y_{i_1},\ldots,y_{i_{k-1}}} \\
= & \quad \{ \text{property substitution, property dot} \} \\
 & (\mathbf{W}\, T : T \in \mathbf{p}sys(c_i) : (p_i{\cdot}T)^{x_{i_0},x_{i_1},\ldots,x_{i_{k-1}}}_{y_{i_0},y_{i_1},\ldots,y_{i_{k-1}}}) \restriction (p_i{\cdot}\mathbf{e}sys(c_i))^{x_{i_0},x_{i_1},\ldots,x_{i_{k-1}}}_{y_{i_0},y_{i_1},\ldots,y_{i_{k-1}}} \\
= & \quad \{ (*), \text{restrictions on component} \} \\
 & (p_i{\cdot}(\mathbf{W}\, T : T \in \mathbf{p}sys(c_i) : T))^{x_{i_0},x_{i_1},\ldots,x_{i_{k-1}}}_{y_{i_0},y_{i_1},\ldots,y_{i_{k-1}}} \restriction (p_i{\cdot}\mathbf{e}sys(c_i))^{x_{i_0},x_{i_1},\ldots,x_{i_{k-1}}}_{y_{i_0},y_{i_1},\ldots,y_{i_{k-1}}} \\
= & \quad \{ (*), \text{restriction on component} \} \\
 & (\, p_i{\cdot}(\, (\mathbf{W}\, T : T \in \mathbf{p}sys(c_i) : T) \restriction \mathbf{e}sys(c_i)\,)\,)^{x_{i_0},x_{i_1},\ldots,x_{i_{k-1}}}_{y_{i_0},y_{i_1},\ldots,y_{i_{k-1}}} \\
= & \quad \{ \text{definition process of a system, definition } i_0, i_1, \ldots, i_{k-1} \} \\
 & (p_i{\cdot}\mathrm{PR}(c_i))^{x_0,x_1,\ldots,x_{m-1}}_{y_0,y_1,\ldots,y_{m-1}}
\end{array}$$

(End of Note)

Example 1.4.1

Component sem_1 is defined by

$$\mathbf{com}\ sem_1(a,b) : (a\ ;\ b)^*\ \mathbf{moc}$$

We then have

$$sys(sem_1) = \langle \{a,b\}, \{\mathrm{SEM}_1(a,b)\} \rangle$$

and

$$\mathrm{PR}(sem_1) = \mathrm{SEM}_1(a,b)$$

Component sem_3 is defined by

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

We then have

$$sys(sem_3) = \langle\, \{a,b\}, \{\mathrm{SEM}_1(a,p{\cdot}b), \mathrm{SEM}_1(p{\cdot}b,q{\cdot}a), \mathrm{SEM}_1(q{\cdot}a,b)\}\,\rangle$$

From corollary 1.3.1 we infer

$$\mathrm{PR}(sem_3) = \mathrm{SEM}_3(a,b)$$

(End of Example)

We now drop the requirement that the subcomponents of a component are previously defined components. 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. A component is called *recursive* if it occurs in itself. Here, we will restrict ourselves to the most simple form of recursion. Let component c be defined by

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

where A is a finite alphabet of simple symbols, p is a simple symbol, and $\mathbf{a}\mathrm{PR}(S) = A \cup p{\cdot}A$. Applying the previous definition of a component yields $sys(c) = (p{\cdot}sys(c) \parallel sys(S))\restriction A$, in other words $sys(c)$ is a solution of

$$R \in \Sigma(A) : R = (p{\cdot}R \parallel sys(S))\restriction A$$

or, using $A \subseteq \mathbf{a}\mathrm{PR}(S)$,

$$R \in \Sigma(A) : \mathbf{p}R = \mathbf{p}(p{\cdot}R) \cup \{\mathrm{PR}(S)\}$$

From lattice theory ([Bi]) it is known that this equation has a least fixpoint, namely

$$\langle\, A, \{\, (p{\cdot})^i \mathrm{PR}(S) \mid i \geq 0\}\,\rangle$$

Therefore, we define

$$sys(c) = \langle A, \{(p\cdot)^i \mathrm{PR}(S) \mid i \geq 0\}\rangle$$

The process of component c, denoted by $\mathrm{PR}(c)$, is defined by $\mathrm{PR}(c) = \mathrm{PR}(sys(c))$. In [Ka] the process of component c is defined to be the least fixpoint of $f : \mathcal{T}(A) \to \mathcal{T}(A)$ where $f(T) = (p\cdot T \,\mathbf{w}\, \mathrm{PR}(S))\restriction A$ for all $T \in \mathcal{T}(A)$. This least fixpoint equals

$$(\cup i : i \geq 0 : f^i(\mathrm{STOP}(A)))$$

We will prove that $\mathrm{PR}(c)$ equals this fixpoint, thereby showing that the choice in [Ka] is the right one.

Theorem 1.4.2

$$\mathrm{PR}(c) = (\cup i : i \geq 0 : f^i(\mathrm{STOP}(A)))$$

Proof

We have

$$\mathrm{PR}(c) = (\mathbf{W}\, i : i \geq 0 : (p\cdot)^i \mathrm{PR}(S))\restriction A$$

It is easily seen that $\mathrm{PR}(c)$ is indeed a fixpoint of f. Define for $j \geq 0$

$$T_j = (\mathbf{W}\, i : 0 \leq i < j : (p\cdot)^i \mathrm{PR}(S)) \,\mathbf{w}\, (\mathbf{W}\, i : i \geq j : (p\cdot)^i \mathrm{STOP}(A))$$

We observe that

(0) $T_0 = \mathrm{STOP}(\,(\cup i : i \geq 0 : (p\cdot)^i A)\,)$
(1) $(\mathbf{A}\, j : j \geq 0 : p\cdot T_j \,\mathbf{w}\, \mathrm{PR}(S) = T_{j+1})$
(2) $(\mathbf{A}\, j : j \geq 0 : T_j \subseteq T_{j+1})$
(3) $(\cup j : j \geq 0 : T_j) = (\mathbf{W}\, i : i \geq 0 : (p\cdot)^i \mathrm{PR}(S))$

By induction we show that for $j \geq 0$

(4) $T_j\restriction A = f^j(\mathrm{STOP}(A))$

base

$$\begin{array}{ll} & T_0\restriction A \\ = & \quad \{\, \mathrm{STOP}(B)\restriction C = \mathrm{STOP}(B \cap C),\ (0)\,\} \\ & \mathrm{STOP}(A) \\ = & \quad \{\, \text{definition } f^0\,\} \\ & f^0(\mathrm{STOP}(A)) \end{array}$$

step

Let $k \geq 0$. Suppose $T_k \restriction A = f^k(\mathrm{STOP}(A))$. We derive

$$
\begin{array}{ll}
 & f^{k+1}(\mathrm{STOP}(A)) \\
= & \quad \{ \text{definition } f^{k+1} \} \\
 & f(f^k(\mathrm{STOP}(A))) \\
= & \quad \{ \text{induction hypothesis, definition } f \} \\
 & (p{\cdot}(T_k \restriction A)\, \mathbf{w}\, \mathrm{PR}(S)) \restriction A \\
= & \quad \{ \text{calculus} \} \\
 & ((p{\cdot}T_k) \restriction (p{\cdot}A)\, \mathbf{w}\, \mathrm{PR}(S)) \restriction A \\
= & \quad \{ \mathbf{a}(p{\cdot}T_k) \cap \mathbf{a}\mathrm{PR}(S) = p{\cdot}A \subseteq A \cup p{\cdot}A \} \\
 & (p{\cdot}T_k\, \mathbf{w}\, \mathrm{PR}(S)) \restriction (A \cup p{\cdot}A) \restriction A \\
= & \quad \{ \text{property projection, (1)} \} \\
 & T_{k+1} \restriction A
\end{array}
$$

Therefore, we have

$$
\begin{array}{ll}
 & \mathrm{PR}(c) \\
= & \quad \{ \text{definitions} \} \\
 & (\mathbf{W}\, i : i \geq 0 : (p{\cdot})^i \mathrm{PR}(S)) \restriction A \\
= & \quad \{ (3) \} \\
 & (\cup j : j \geq 0 : T_j) \restriction A \\
= & \quad \{ \text{property projection, (2)} \} \\
 & (\cup j : j \geq 0 : T_j \restriction A) \\
= & \quad \{ (4) \} \\
 & (\cup j : j \geq 0 : f^j(\mathrm{STOP}(A)))
\end{array}
$$

(End of Proof)

2 Properties of processes and systems

2.0 Introduction

In this chapter we discuss the phenomena *nondeterminism*, *divergence*, and *deadlock* in relation to processes and systems. Properties of processes and systems are defined expressing the absence of one or two of the above phenomena. Furthermore, we introduce two special classes of processes: *conservative* processes and *cubic* processes, the latter forming a subclass of the former. The relation between cubic processes and processes defined by *partial orders* on sets of *occurrences* is shown. Finally, for cubic processes *sequence functions* are introduced describing restricted (clocked) behaviours of the processes.

2.1 Nondeterminism and divergence

In this section we study conditions under which the (external) process of a system forms an adequate description of the external behaviour of the mechanism corresponding to the system.

Let S be the system $sys(c;\ a \mid d;\ b)\restriction\{a,b\}$. We have $\mathrm{PR}(S) = \mathrm{PR}(a \mid b)$. However, process $\mathrm{PR}(S)$ does not adequately describe the external behaviour of system S: after occurrence of internal event c external event b is not possible any more. We say that b is disabled by an internal event. The same holds for internal event d and external event a. On the other hand, though, one may infer from $\mathrm{PR}(S)$ that both a and b are possible. We say that system S has (internal) nondeterminism.

Let S be the system $sys((b \mid a)^*)\restriction\{a\}$. We have $\mathrm{PR}(S) = \mathrm{PR}(a^*)$. Again process $\mathrm{PR}(S)$ does not adequately describe the external behaviour of system S. Before the first external event a and between any two consecutive external events a an unbounded number of internal events, b's, may occur. This phenomenon is called *divergence*.

We first investigate the relation between the mechanism corresponding to a process T and the mechanism corresponding to $T\restriction A$, where A is a subset of $\mathbf{a}T$. In the sequel T is a process and A an alphabet such that $A \subseteq \mathbf{a}T$.

Alphabet A is called *non-disabling* with respect to T if

$$(\mathbf{A}\, t : t \in \mathbf{t}T : \mathit{after}(t, T)\restriction A = \mathit{after}(t\restriction A, T\restriction A))$$

This notion is called I_1 in [Ka]. It may be interpreted as follows: after the occurrence of trace t of T one may expect every external (i.e. in A) continuation as given by process $T\restriction A$ after trace $t\restriction A$. We say that A is *disabling* with respect to T if A is not non-disabling with respect to T. Notice that both $\varnothing$ and $\mathbf{a}T$ are non-disabling with respect to T. Notice that $\{a, b\}$ is disabling with respect to $\mathrm{PR}(c;\ a \mid d;\ b)$.

A system S is called *non-disabling* if $\mathbf{e}S$ is non-disabling with respect to $\mathbf{W}(\mathrm{p}S)$.

Theorem 2.1.0

The following assertions are equivalent:

(0) A is non-disabling with respect to T

(1) $(\mathbf{A}\, s, t : s \in \mathbf{t}T \wedge t \in \mathbf{t}T \wedge s\restriction A = t\restriction A : \mathit{after}(s, T)\restriction A = \mathit{after}(t, T)\restriction A)$

(2) $(\mathbf{A}\, s, t : s \in \mathbf{t}T \wedge t \in \mathbf{t}T \wedge s\restriction A \leq t\restriction A : (\mathbf{E}\, u : su \in \mathbf{t}T : su\restriction A = t\restriction A))$

Proof

(0) $\equiv$ (1)

We derive

$$\begin{array}{ll} & A \text{ is non-disabling with respect to } T \\ = & \quad \{\text{definition non-disabling}\} \\ & (\mathbf{A}\, t : t \in \mathbf{t}T : \mathit{after}(t, T)\restriction A = \mathit{after}(t\restriction A, T\restriction A)) \\ = & \quad \{\text{property 1.1.3}\} \\ & (\mathbf{A}\, t : t \in \mathbf{t}T : \mathit{after}(t, T)\restriction A = (\cup\, s : s \in \mathbf{t}T \wedge s\restriction A = t\restriction A : \mathit{after}(s, T)\restriction A)) \\ = & \quad \{\text{set calculus}\} \\ & (\mathbf{A}\, t : t \in \mathbf{t}T : (\mathbf{A}\, s : s \in \mathbf{t}T \wedge s\restriction A = t\restriction A : \mathit{after}(s, T)\restriction A \subseteq \mathit{after}(t, T)\restriction A)) \\ = & \quad \{\text{idempotence conjunction, renaming dummies}\} \\ & (\mathbf{A}\, s, t : s \in \mathbf{t}T \wedge t \in \mathbf{t}T \wedge s\restriction A = t\restriction A : \mathit{after}(s, T)\restriction A \subseteq \mathit{after}(t, T)\restriction A) \\ & \wedge\ (\mathbf{A}\, s, t : s \in \mathbf{t}T \wedge t \in \mathbf{t}T \wedge s\restriction A = t\restriction A : \mathit{after}(t, T)\restriction A \subseteq \mathit{after}(s, T)\restriction A) \\ = & \quad \{\text{calculus}\} \\ & (\mathbf{A}\, s, t : s \in \mathbf{t}T \wedge t \in \mathbf{t}T \wedge s\restriction A = t\restriction A : \mathit{after}(s, T)\restriction A = \mathit{after}(t, T)\restriction A) \end{array}$$

(1) $\Rightarrow$ (2)

Assume (1). Let $s \in \mathbf{t}T$ and $t \in \mathbf{t}T$ such that $s\restriction A \leq t\restriction A$. Choose t_0 and t_1 such that $t = t_0 t_1$ and $s\restriction A = t_0\restriction A$. We now have

$$
\begin{array}{ll}
& t_0 t_1 \in \mathbf{t}T \\
\Rightarrow & \quad \{ \text{definition } \textit{after}, \text{ property projection} \} \\
& t_1 \restriction A \in \mathbf{t}\textit{after}(t_0, T) \restriction A \\
= & \quad \{ s \restriction A = t_0 \restriction A, (1) \} \\
& t_1 \restriction A \in \mathbf{t}\textit{after}(s, T) \restriction A \\
= & \quad \{ \text{definition projection, definition } \textit{after} \} \\
& (\mathbf{E}\, u : su \in \mathbf{t}T : u \restriction A = t_1 \restriction A) \\
= & \quad \{ s \restriction A = t_0 \restriction A, t = t_0 t_1 \} \\
& (\mathbf{E}\, u : su \in \mathbf{t}T : (su) \restriction A = t \restriction A)
\end{array}
$$

(2) $\Rightarrow$ (1)

Assume (2). Let $s \in \mathbf{t}T$ and $t \in \mathbf{t}T$ be such that $s \restriction A = t \restriction A$. We derive

$$
\begin{array}{ll}
& r \in \mathbf{t}\textit{after}(s, T) \restriction A \\
= & \quad \{ \text{definition projection, definition } \textit{after} \} \\
& (\mathbf{E}\, u : su \in \mathbf{t}T : u \restriction A = r) \\
= & \quad \{ s \restriction A = t \restriction A, (2) \} \\
& (\mathbf{E}\, u : su \in \mathbf{t}T : u \restriction A = r \wedge (\mathbf{E}\, v : tv \in \mathbf{t}T : tv \restriction A = su \restriction A)) \\
= & \quad \{ s \restriction A = t \restriction A, \text{ calculus} \} \\
& (\mathbf{E}\, u, v : su \in \mathbf{t}T \wedge tv \in \mathbf{t}T : u \restriction A = r \wedge v \restriction A = r)
\end{array}
$$

Observe that the last predicate in the derivation is symmetric in s and t. Hence, it is equivalent to $r \in \mathbf{t}\textit{after}(t, T) \restriction A$ as well.

(End of Proof)

The next two theorems give conditions under which an alphabet is non-disabling with respect to the weave of two processes.

Theorem 2.1.1

Let T and U be processes. Let A and B be alphabets such that $A \subseteq \mathbf{a}T$, $B \subseteq \mathbf{a}U$, and $\mathbf{a}T \cap \mathbf{a}U = A \cap B$. If A is non-disabling with respect to T and B is non-disabling with respect to U then $A \cup B$ is non-disabling with respect to $T \,\mathbf{w}\, U$.

Proof

Assume A is non-disabling with respect to T and B is non-disabling with respect to U. Let $t \in \mathbf{t}(T \,\mathbf{w}\, U)$. We have

$$
\begin{array}{ll}
 & \mathit{after}(t\restriction(A \cup B), (T \,\mathbf{w}\, U)\restriction(A \cup B)) \\
= & \quad \{ \text{theorem 1.1.11} \} \\
 & \mathit{after}(t\restriction(A \cup B), T\restriction A \,\mathbf{w}\, U\restriction B) \\
= & \quad \{ \text{theorem 1.1.13} \} \\
 & \mathit{after}(t\restriction A, T\restriction A) \;\mathbf{w}\; \mathit{after}(t\restriction B, U\restriction B) \\
= & \quad \{ A \text{ is non-disabling w.r.t. } T,\ B \text{ is non-disabling w.r.t. } U \} \\
 & \mathit{after}(t\restriction \mathbf{a}T, T)\restriction A \;\mathbf{w}\; \mathit{after}(t\restriction \mathbf{a}U, U)\restriction B \\
= & \quad \{ \text{theorem 1.1.11} \} \\
 & (\mathit{after}(t\restriction \mathbf{a}T, T) \;\mathbf{w}\; \mathit{after}(t\restriction \mathbf{a}U, U))\restriction (A \cup B) \\
= & \quad \{ \text{theorem 1.1.13} \} \\
 & \mathit{after}(t, T \,\mathbf{w}\, U)\restriction (A \cup B)
\end{array}
$$

(End of Proof)

Corollary 2.1.2

If R and S are non-disabling systems and $\mathbf{aW}(\mathbf{p}R) \cap \mathbf{aW}(\mathbf{p}S) = \mathbf{e}R \cap \mathbf{e}S$, then $R \parallel S$ is a non-disabling system.

(End of Corollary)

Theorem 2.1.3

Let T and U be processes. Let A be an alphabet such that $A \subseteq \mathbf{a}U \setminus \mathbf{a}T$. If $U\restriction(\mathbf{a}T \cap \mathbf{a}U) \subseteq T\restriction(\mathbf{a}T \cap \mathbf{a}U)$, $\mathbf{a}T \cap \mathbf{a}U$ is non-disabling with respect to T and A is non-disabling with respect to U, then A is non-disabling with respect to $T \,\mathbf{w}\, U$.

Proof

Assume $U\restriction(\mathbf{a}T \cap \mathbf{a}U) \subseteq T\restriction(\mathbf{a}T \cap \mathbf{a}U)$. We derive

$$
\begin{array}{ll}
 & (T \,\mathbf{w}\, U)\restriction \mathbf{a}U \\
= & \quad \{ \mathbf{a}T \cap \mathbf{a}U \subseteq \mathbf{a}U, \text{ theorem 1.1.10} \} \\
 & T\restriction(\mathbf{a}T \cap \mathbf{a}U) \;\mathbf{w}\; U \\
= & \quad \{ \text{theorem 1.1.7} \} \\
 & T\restriction(\mathbf{a}T \cap \mathbf{a}U) \;\mathbf{w}\; U\restriction(\mathbf{a}T \cap \mathbf{a}U) \;\mathbf{w}\; U \\
= & \quad \{ U\restriction(\mathbf{a}T \cap \mathbf{a}U) \subseteq T\restriction(\mathbf{a}T \cap \mathbf{a}U), \text{ theorem 1.1.7} \} \\
 & U\restriction(\mathbf{a}T \cap \mathbf{a}U) \;\mathbf{w}\; U \\
= & \quad \{ \text{theorem 1.1.7} \} \\
 & U
\end{array}
$$

Assume $\mathbf{a}T \cap \mathbf{a}U$ is non-disabling with respect to T and A is non-disabling with respect to U. Let $t \in \mathbf{t}(T \mathbf{w} U)$. We derive

$$
\begin{array}{ll}
 & \mathit{after}(t, T \mathbf{w} U) \restriction A \\
= & \quad \{ \text{theorem 1.1.13} \} \\
 & (\mathit{after}(t \restriction \mathbf{a}T, T) \mathbf{w} \mathit{after}(t \restriction \mathbf{a}U, U)) \restriction A \\
= & \quad \{ A \subseteq \mathbf{a}U,\ \mathbf{a}T \cap \mathbf{a}U \subseteq \mathbf{a}U, \text{theorem 1.1.10} \} \\
 & (\mathit{after}(t \restriction \mathbf{a}T, T) \restriction (\mathbf{a}T \cap \mathbf{a}U) \mathbf{w} \mathit{after}(t \restriction \mathbf{a}U, U)) \restriction A \\
= & \quad \{ \mathbf{a}T \cap \mathbf{a}U \text{ is non-disabling with respect to } T, \text{theorem 1.1.13} \} \\
 & \mathit{after}(t \restriction \mathbf{a}U, T \restriction (\mathbf{a}T \cap \mathbf{a}U) \mathbf{w} U) \restriction A \\
= & \quad \{ T \restriction (\mathbf{a}T \cap \mathbf{a}U) \mathbf{w} U = U,\ A \text{ is non-disabling with respect to } U \} \\
 & \mathit{after}(t \restriction A, U \restriction A) \\
= & \quad \{ (T \mathbf{w} U) \restriction \mathbf{a}U = U,\ A \subseteq \mathbf{a}U \} \\
 & \mathit{after}(t \restriction A, (T \mathbf{w} U) \restriction A)
\end{array}
$$

(End of Proof)

Corollary 2.1.4

Let R and S be systems such that $\mathbf{aW}(\mathbf{p}R) \cap \mathbf{aW}(\mathbf{p}S) = \mathbf{e}R \cap \mathbf{e}S$ and $\mathbf{e}R \subseteq \mathbf{e}S$. If R and $S \restriction (\mathbf{e}S \setminus \mathbf{e}R)$ are non-disabling systems and $\mathrm{PR}(S) \restriction \mathbf{e}R \subseteq \mathrm{PR}(R)$ then $(R \parallel S) \restriction (\mathbf{e}S \setminus \mathbf{e}R)$ is a non-disabling system.

(End of Corollary)

The next theorem gives a condition under which an alphabet is non-disabling with respect to the projection of a process on some alphabet.

Theorem 2.1.5

Let T be a process. Let A and B be alphabets such that $B \subseteq A \subseteq \mathbf{a}T$. If A is non-disabling with respect to T then

$$B \text{ is non-disabling with respect to } T \equiv B \text{ is non-disabling with respect to } T \restriction A$$

Proof

Assume A is non-disabling with respect to T. We derive

$$
\begin{array}{ll}
 & B \text{ is non-disabling with respect to } T \\
= & \quad \{ \text{definition non-disabling} \}
\end{array}
$$

$$
\begin{array}{ll}
 & (\mathbf{A}\, t : t \in \mathbf{t}T : \mathit{after}(t,T)\restriction B = \mathit{after}(t\restriction B, T\restriction B)) \\
= & \quad \{\, B \subseteq A, \text{ property projection} \,\} \\
 & (\mathbf{A}\, t : t \in \mathbf{t}T : \mathit{after}(t,T)\restriction A\restriction B = \mathit{after}(t\restriction A\restriction B, T\restriction A\restriction B)) \\
= & \quad \{\, A \text{ is non-disabling with respect to } T \,\} \\
 & (\mathbf{A}\, t : t \in \mathbf{t}T : \mathit{after}(t\restriction A, T\restriction A)\restriction B = \mathit{after}(t\restriction A\restriction B, T\restriction A\restriction B)) \\
= & \quad \{\, \text{calculus} \,\} \\
 & (\mathbf{A}\, u : u \in \mathbf{t}T\restriction A : \mathit{after}(u, T\restriction A)\restriction B = \mathit{after}(u\restriction B, (T\restriction A)\restriction B)) \\
= & \quad \{\, \text{definition non-disabling} \,\} \\
 & B \text{ is non-disabling with respect to } T\restriction A
\end{array}
$$

(End of Proof)

Corollary 2.1.6

If S is a non-disabling system and $A \subseteq \mathbf{e}S$ then

$$S\restriction A \text{ is a non-disabling system} \quad \equiv \quad A \text{ is non-disabling with respect to } \mathrm{PR}(S)$$

(End of Corollary)

In the sequel let T be a process and A an alphabet such that $A \subseteq \mathbf{a}T$. Alphabet $\mathbf{a}T \setminus A$ will be denoted by $\overline{A}$. Alphabet A is called *divergent* with respect to T if

$$(\mathbf{E}\, t : t \in \mathbf{t}T : (\mathbf{A}\, n : n \geq 0 : (\mathbf{E}\, u : u \in (\overline{A})^* \wedge tu \in \mathbf{t}T : \ell(u) > n)))$$

For instance, $\{a\}$ is divergent with respect to both $\mathrm{PR}((b \mid a)^*)$ and $\mathrm{PR}(b^* \mid a)$. Alphabet A is called *non-divergent* with respect to T if A is not divergent with respect to T, i.e.

$$(\mathbf{A}\, t : t \in \mathbf{t}T : (\mathbf{E}\, n : n \geq 0 : (\mathbf{A}\, u : u \in (\overline{A})^* \wedge tu \in \mathbf{t}T : \ell(u) \leq n)))$$

Notice that $\mathbf{a}T$ is non-divergent with respect to T.

A system S is called *divergent* if $\mathbf{e}S$ is divergent with respect to $\mathbf{W}(\mathbf{p}S)$. A system is called *non-divergent* if it is not divergent.

The next theorem gives two alternative characterizations of non-divergence in case the alphabet of process T is finite.

Theorem 2.1.7

Let

$$P_0 \quad \equiv \quad A \text{ is non-divergent with respect to } T$$

$$P_1 \equiv (\mathbf{A}\, t : t \in \mathbf{t}T : \mathbf{t}after(t,T) \cap (\overline{A})^* \text{ is finite })$$
$$P_2 \equiv (\mathbf{A}\, s : s \in \mathbf{t}T\restriction A : \{\, t \mid t \in \mathbf{t}T \wedge t\restriction A = s \,\} \text{ is finite })$$

Then $P_2 \Rightarrow P_1$ and $P_1 \Rightarrow P_0$.
If $\mathbf{a}T$ is finite, then P_0, P_1, and P_2 are equivalent.

Proof

i) We prove that P_1 implies P_0. We derive

$$\begin{array}{ll} & (\mathbf{A}\, t : t \in \mathbf{t}T : \mathbf{t}after(t,T) \cap (\overline{A})^* \text{ is finite }) \\ \Rightarrow & \quad \{\text{calculus}\} \\ & (\mathbf{A}\, t : t \in \mathbf{t}T : (\mathbf{E}\, n : n \geq 0 : (\mathbf{A}\, u : u \in \mathbf{t}after(t,T) \cap (\overline{A})^* : \ell(u) \leq n))) \\ = & \quad \{\text{definition } after\} \\ & (\mathbf{A}\, t : t \in \mathbf{t}T : (\mathbf{E}\, n : n \geq 0 : (\mathbf{A}\, u : tu \in \mathbf{t}T \wedge u \in (\overline{A})^* : \ell(u) \leq n))) \end{array}$$

ii) We prove that P_2 implies P_1. Assume P_2. Let $t \in \mathbf{t}T$. We have

$$\begin{array}{ll} & \{\, tu \mid u \in \mathbf{t}after(t,T) \cap (\overline{A})^* \,\} \\ = & \quad \{\text{definition } after, \text{ property projection}\} \\ & \{\, tu \mid tu \in \mathbf{t}T \wedge u \in (\overline{A})^* \wedge (tu)\restriction A = t\restriction A \,\} \\ \subseteq & \quad \{\text{set calculus}\} \\ & \{\, r \mid r \in \mathbf{t}T \wedge r\restriction A = t\restriction A \,\} \end{array}$$

The last set being finite due to P_2, we have that $\mathbf{t}after(t,T) \cap (\overline{A})^*$ is finite.

iii) Assume $\mathbf{a}T$ is finite. We prove that $\neg P_2$ implies $\neg P_0$. Assume $\neg P_2$. Let $s \in \mathbf{t}T\restriction A$ be such that $\{\, t \mid t \in \mathbf{t}T \wedge t\restriction A = s \,\}$ is infinite. Define

$$U = \text{PREF}(\{\, t \mid t \in \mathbf{t}T \wedge t\restriction A = s \,\})$$

Obviously U is infinite and $U \subseteq \mathbf{a}T^*$. Since $\mathbf{a}T$ is finite König's lemma [Kö] is applicable. Let $a(i : i \geq 0)$ be a sequence of symbols in $\mathbf{a}T$ such that

$$(\mathbf{A}\, n : n \geq 0 : a(i : 0 \leq i < n) \in U)$$

From the definition of U we infer that

$$(\mathbf{A}\, u : u \in U : \ell(u\restriction A) \leq \ell(s))$$

Hence, let $n_0 \geq 0$ be such that

$$(\mathbf{A}\, n : n \geq n_0 : a(n) \in \overline{A})$$

We then have

$$(\mathbf{A}\, n : n \geq n_0 : a(i : 0 \leq i < n_0)a(i : n_0 \leq i < n) \in \mathbf{t}T \\ \wedge\ a(i : n_0 \leq i < n) \in (\overline{A})^* \quad)$$

From this we infer that A is divergent with respect to T.

(End of Proof)

Theorems 2.1.8 and 2.1.11 give conditions under which an alphabet is non-divergent with respect to the weave of two processes.

Theorem 2.1.8

Let T and U be processes. Let A and B be alphabets such that $A \subseteq \mathbf{a}T$, $B \subseteq \mathbf{a}U$, and $A \cap \mathbf{a}U = B \cap \mathbf{a}T$. If A is non-divergent with respect to T and B be non-divergent with respect to U then $A \cup B$ is non-divergent with respect to $T\,\mathbf{w}\,U$.

Proof

Assume A is non-divergent with respect to T and B is non-divergent with respect to U. Let $\overline{A} = \mathbf{a}T \setminus A$ and $\overline{B} = \mathbf{a}U \setminus B$. Notice that $(\mathbf{a}T \cup \mathbf{a}U) \setminus (A \cup B) = \overline{A} \cup \overline{B}$ and $\overline{A} \cap \mathbf{a}U = \overline{B} \cap \mathbf{a}T$. Let $t \in \mathbf{t}(T\,\mathbf{w}\,U)$.
Let $m \geq 0$ be such that $(\mathbf{A}\, u : u \in (\overline{A})^* \wedge (t\restriction \mathbf{a}T)u \in \mathbf{t}T : \ell(u) \leq m\,)$, and let $n \geq 0$ be such that $(\mathbf{A}\, v : v \in (\overline{B})^* \wedge (t\restriction \mathbf{a}U)v \in \mathbf{t}U : \ell(v) \leq n\,)$. We now have

$$\begin{array}{ll} & w \in (\overline{A} \cup \overline{B})^* \wedge tw \in \mathbf{t}(T\,\mathbf{w}\,U) \\ = & \quad \{\text{definition weave, } w\restriction(\overline{A} \cup \overline{B}) = w\,\} \\ & w \in (\overline{A} \cup \overline{B})^* \wedge (t\restriction \mathbf{a}T)(\,w\restriction((\overline{A} \cup \overline{B}) \cap \mathbf{a}T)\,) \in \mathbf{t}T \\ & \wedge\ (t\restriction \mathbf{a}U)(\,w\restriction((\overline{A} \cup \overline{B}) \cap \mathbf{a}U)\,) \in \mathbf{t}U \\ = & \quad \{\,\overline{A} \subseteq \mathbf{a}T,\ \overline{B} \subseteq \mathbf{a}U,\ \overline{A} \cap \mathbf{a}U = \overline{B} \cap \mathbf{a}T\,\} \\ & w \in (\overline{A} \cup \overline{B})^* \wedge (t\restriction \mathbf{a}T)(w\restriction \overline{A}) \in \mathbf{t}T \wedge (t\restriction \mathbf{a}U)(w\restriction \overline{B}) \in \mathbf{t}U \\ \Rightarrow & \quad \{\text{definition of } m \text{ and } n\,\} \\ & w \in (\overline{A} \cup \overline{B})^* \wedge \ell(w\restriction \overline{A}) \leq m \wedge \ell(w\restriction \overline{B}) \leq n \\ \Rightarrow & \quad \{\text{calculus}\,\} \\ & \ell(w) \leq m + n \end{array}$$

(End of Proof)

Corollary 2.1.9

Let R and S be systems such that $\mathbf{aW}(\mathbf{p}R) \cap \mathbf{aW}(\mathbf{p}S) = \mathbf{e}R \cap \mathbf{e}S$. If R and S are non-divergent systems, then $R \parallel S$ is a non-divergent system.

(End of Corollary)

Lemma 2.1.10

Let T and U be processes. Let A be an alphabet such that $A \subseteq \mathbf{a}U \setminus \mathbf{a}T$. If

(0) $(\mathbf{A}\, s : s \in \mathbf{t}T\restriction(\mathbf{a}T \cap \mathbf{a}U) : \{\, t \mid t \in \mathbf{t}T \wedge t\restriction(\mathbf{a}T \cap \mathbf{a}U) = s \,\}$ is finite$)$

and

(1) $(\mathbf{A}\, v : v \in \mathbf{t}U\restriction A : \{\, u \mid u \in \mathbf{t}U \wedge u\restriction A = v \,\}$ is finite$)$

then

$(\mathbf{A}\, z : z \in \mathbf{t}(T\,\mathbf{w}\,U)\restriction A : \{\, w \mid w \in \mathbf{t}(T\,\mathbf{w}\,U) \wedge w\restriction A = z \,\}$ is finite$)$

Proof

Assume (0) and (1) hold.
Define $X(s) = \{\, t \mid t \in \mathbf{t}T \wedge t\restriction(\mathbf{a}T \cap \mathbf{a}U) = s \,\}$ for $s \in \mathbf{t}T\restriction(\mathbf{a}T \cap \mathbf{a}U)$, and $Y(v) = \{\, u \mid u \in \mathbf{t}U \wedge u\restriction A = v \,\}$ for $v \in \mathbf{t}U\restriction A$. Notice that $X(s)$ is finite for all $s \in \mathbf{t}T\restriction(\mathbf{a}T \cap \mathbf{a}U)$, and $Y(v)$ is finite for all $v \in \mathbf{t}U\restriction A$.
Let $z \in \mathbf{t}(T\,\mathbf{w}\,U)\restriction A$. Notice that $z \in \mathbf{t}U\restriction A$. We now have

$$
\begin{array}{ll}
 & \{\, w \mid w \in \mathbf{t}(T\,\mathbf{w}\,U) \wedge w\restriction A = z \,\} \\
= & \quad \{\, \text{definition weave} \,\} \\
 & \{\, w \mid w \in (\mathbf{a}T \cup \mathbf{a}U)^* \wedge w\restriction \mathbf{a}T \in \mathbf{t}T \wedge w\restriction \mathbf{a}U \in \mathbf{t}U \wedge w\restriction A = z \,\} \\
= & \quad \{\, w\restriction A = w\restriction \mathbf{a}U\restriction A, \text{ calculus} \,\} \\
 & (\cup u : u \in Y(z) \wedge u\restriction(\mathbf{a}T \cap \mathbf{a}U) \in \mathbf{t}T\restriction(\mathbf{a}T \cap \mathbf{a}U) \\
 & \qquad : \{\, w \mid w \in (\mathbf{a}T \cup \mathbf{a}U)^* \wedge w\restriction \mathbf{a}T \in \mathbf{t}T \wedge w\restriction \mathbf{a}U = u \,\}) \\
= & \quad \{\, \text{calculus} \,\} \\
 & (\cup u : u \in Y(z) \wedge u\restriction(\mathbf{a}T \cap \mathbf{a}U) \in \mathbf{t}T\restriction(\mathbf{a}T \cap \mathbf{a}U) \\
 & \qquad : (\cup t : t \in X(u\restriction(\mathbf{a}T \cap \mathbf{a}U)) \\
 & \qquad\quad : \{\, w \mid w \in (\mathbf{a}T \cup \mathbf{a}U)^* \wedge w\restriction \mathbf{a}T = t \wedge w\restriction \mathbf{a}U = u \,\}))
\end{array}
$$

The set $\{\, w \mid w \in (\mathbf{a}T \cup \mathbf{a}U)^* \wedge w\restriction \mathbf{a}T = t \wedge w\restriction \mathbf{a}U = u \,\}$ being finite for all $t \in \mathbf{t}T$ and $u \in \mathbf{t}U$ we have that $\{\, w \mid w \in \mathbf{t}(T\,\mathbf{w}\,U) \wedge w\restriction A = z \,\}$ is finite.

(End of Proof)

Combining theorem 2.1.7 and lemma 2.1.10 yields

Theorem 2.1.11

Let T and U be processes such that $\mathbf{a}T$ and $\mathbf{a}U$ are finite. Let A be an alphabet such that $A \subseteq \mathbf{a}U \setminus \mathbf{a}T$. If $\mathbf{a}T \cap \mathbf{a}U$ is non-divergent with respect to T, and A is non-divergent with respect to U then A is non-divergent with respect to $T \,\mathbf{w}\, U$.

(End of Theorem)

Corollary 2.1.12

Let R and S be systems such that $\mathbf{aW}(\mathbf{p}R)$ and $\mathbf{aW}(\mathbf{p}S)$ are finite, $\mathbf{aW}(\mathbf{p}R) \cap \mathbf{aW}(\mathbf{p}S) = \mathbf{e}R \cap \mathbf{e}S$, and $\mathbf{e}R \subseteq \mathbf{e}S$. If R and $S\restriction(\mathbf{e}S \setminus \mathbf{e}R)$ are non-divergent systems then $(R \parallel S)\restriction(\mathbf{e}S \setminus \mathbf{e}R)$ is a non-divergent system.

(End of Corollary)

The following example shows that the above theorem need not hold in case the alphabets are infinite.

Example 2.1.13

$T = \mathrm{PREF}(\langle \{ x_i \mid i \geq 0 \} \cup \{y\}, \{ x_i y^i x_i \mid i \geq 0 \} \rangle)$
$U = \mathrm{PREF}(\langle \{ x_i \mid i \geq 0 \} \cup \{a\}, \{ a x_i x_i a \mid i \geq 0 \} \rangle)$
$\{ x_i \mid i \geq 0 \}$ is non-divergent with respect to T
$\{a\}$ is non-divergent with respect to U
$\{a\}$ is divergent with respect to $T \,\mathbf{w}\, U$

(End of Example)

Next we investigate non-divergence with respect to the projection of a process on some alphabet.

Lemma 2.1.14

Let T be a process. Let A and B be alphabets such that $B \subseteq A \subseteq \mathbf{a}T$.

0 $(\mathbf{A}\, s : s \in \mathbf{t}T\restriction B : \{ t \mid t \in \mathbf{t}T \wedge t\restriction B = s \}$ is finite$)$
$\Rightarrow (\mathbf{A}\, s : s \in \mathbf{t}T\restriction B : \{ r \mid r \in \mathbf{t}T\restriction A \wedge r\restriction B = s \}$ is finite$)$

1 If $(\mathbf{A}\, r : r \in \mathbf{t}T\restriction A : \{ t \mid t \in \mathbf{t}T \wedge t\restriction A = r \}$ is finite$)$ then

$$(\mathbf{A}\, s : s \in \mathbf{t}T\restriction B : \{\, t \mid t \in \mathbf{t}T \wedge t\restriction B = s \,\} \text{ is finite})$$
$$\equiv\ (\mathbf{A}\, s : s \in \mathbf{t}T\restriction B : \{\, r \mid r \in \mathbf{t}T\restriction A \wedge r\restriction B = s \,\} \text{ is finite})$$

Proof

0 Assume $(\mathbf{A}\, s : s \in \mathbf{t}T\restriction B : \{\, t \mid t \in \mathbf{t}T \wedge t\restriction B = s \,\} \text{ is finite})$. Let $s \in \mathbf{t}T\restriction B$. We have that $\{\, t \mid t \in \mathbf{t}T \wedge t\restriction B = s \,\}$ is finite. Hence, the set

$$\{\, r \mid r \in \mathbf{t}T\restriction A \wedge r\restriction B = s \,\} = \{\, t \mid t \in \mathbf{t}T \wedge t\restriction B = s \,\}\restriction A$$

is finite.

1 Let $(\mathbf{A}\, r : r \in \mathbf{t}T\restriction A : \{\, t \mid t \in \mathbf{t}T \wedge t\restriction A = r \,\} \text{ is finite})$. Assume

$$(\mathbf{A}\, s : s \in \mathbf{t}T\restriction B : \{\, r \mid r \in \mathbf{t}T\restriction A \wedge r\restriction B = s \,\} \text{ is finite})$$

Let $s \in \mathbf{t}T\restriction B$. We now have that $\{\, u \mid u \in \mathbf{t}T\restriction A \wedge u\restriction B = s \,\}$ is finite, and that for all $r \in \mathbf{t}T\restriction A$ $\{\, t \mid t \in \mathbf{t}T \wedge t\restriction A = r \,\}$ is finite. Therefore, the set

$$\{\, t \mid t \in \mathbf{t}T \wedge t\restriction B = s \,\}$$
$$= (\cup r : r \in \{\, u \mid u \in \mathbf{t}T\restriction A \wedge u\restriction B = s \,\} : \{\, t \mid t \in \mathbf{t}T \wedge t\restriction A = r \,\})$$

is finite.

(End of Proof)

Combining theorem 2.1.7 and lemma 2.1.14 yields

Theorem 2.1.15

Let T be a process such that $\mathbf{a}T$ is finite. Let A and B be alphabets such that $B \subseteq A \subseteq \mathbf{a}T$.

0 B is non-divergent with respect to T
$\Rightarrow$ B is non-divergent with respect to $T\restriction A$

1 If A is non-divergent with respect to T then
B is non-divergent with respect to T
$\equiv$ B is non-divergent with respect to $T\restriction A$.

(End of Theorem)

The following example shows the above theorem not to hold in case $\mathbf{a}T$ is infinite.

Example 2.1.16

$T = \mathrm{PREF}(\langle \{a,b\} \cup \{ x_i \mid i \geq 0 \}, \{ x_i b a^i b \mid i \geq 0 \} \rangle)$
$\{a,b\}$ is non-divergent with respect to T
$\{b\}$ is non-divergent with respect to T
$\{b\}$ is divergent with respect to $T{\restriction}\{a,b\}$

(End of Example)

Corollary 2.1.17

Let S be a system such that $\mathbf{aW}(\mathbf{p}S)$ is finite. Let $B \subseteq \mathbf{e}S$.
If S is non-divergent then

$S{\restriction}B$ is a non-divergent system $\equiv$ B is non-divergent with respect to $\mathrm{PR}(S)$

(End of Corollary)

Theorem 2.1.18

Let command S in

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

satisfy

$$(\mathbf{A}\, t : t \in \mathbf{t}\mathrm{PR}(S) \wedge t \neq \varepsilon : \ell(t{\restriction}p{\cdot}A) < \ell(t{\restriction}A))$$

Then $sys(c)$ is non-divergent.

Proof

We have

$$\mathrm{PR}(c) = (\mathbf{W}\, i : i \geq 0 : (p{\cdot})^i \mathrm{PR}(S)){\restriction}A$$

For $t \in \mathbf{t}\mathrm{PR}(c)$ we define

$$V(t) = \{ s \mid s \in \mathbf{t}(\mathbf{W}\, i : i \geq 0 : (p{\cdot})^i \mathrm{PR}(S)) \wedge s{\restriction}A = t \}$$

Let $t \in \mathbf{t}\mathrm{PR}(c)$ and $s \in V(t)$. Then $s{\restriction}A = t$ and

$$(\mathbf{A}\, i : i \geq 0 : s\restriction(p\cdot)^i(A \cup p\cdot A) \in \mathbf{t}(p\cdot)^i\mathrm{PR}(S))$$

It follows that

$$(\mathbf{A}\, i : i \geq 0 \wedge s\restriction(p\cdot)^i(A \cup p\cdot A) \neq \varepsilon : \ell(s\restriction(p\cdot)^{i+1}A) < \ell(s\restriction(p\cdot)^iA))$$

and, hence,

(0) $(\mathbf{A}\, i : 0 \leq i < \ell(t) : \ell(s\restriction(p\cdot)^iA) \leq \ell(t) - i)$
(1) $(\mathbf{A}\, i : i \geq \ell(t) : \ell(s\restriction(p\cdot)^iA) = 0)$

We derive

$$\begin{array}{ll} & \ell(s) \\ = & \quad \{\text{calculus}\} \\ & (\mathbf{S}\, i : i \geq 0 : \ell(s\restriction(p\cdot)^iA)) \\ = & \quad \{(1)\} \\ & (\mathbf{S}\, i : 0 \leq i < \ell(t) : \ell(s\restriction(p\cdot)^iA)) \\ \leq & \quad \{(0)\} \\ & (\mathbf{S}\, i : 0 \leq i < \ell(t) : \ell(t) - i) \\ = & \quad \{\text{calculus}\} \\ & \frac{1}{2} \cdot \ell(t) \cdot (\ell(t) + 1) \end{array}$$

Observe that

$$V(t) = V(t) \cap (\cup\, i : 0 \leq i < \ell(t) : (p\cdot)^iA)^*$$

We conclude that $V(t)$ is finite. Therefore, by theorem 2.1.7, A is non-divergent with respect to $(\mathbf{W}\, i : i \geq 0 : (p\cdot)^i\mathrm{PR}(S))$.

(End of Proof)

The condition on the command in the above theorem is also found in [Ud] and [Ka] where it is shown to imply the existence of a unique fixpoint for the recursive equation defined by a recursive component.

As we already illustrated in the introduction to this section, nondeterminism and divergence are properties to be avoided. Therefore, we now introduce a third notion that is a combination of non-disabling and non-divergent. This notion was first introduced by Anne Kaldewaij in [Ka].

Let T be a process and let A be an alphabet such that $A \subseteq \mathbf{a}T$. Alphabet A is called

transparent with respect to T if A is both non-disabling and non-divergent with respect to T. If A is transparent with respect to T, then after the occurrence of trace t in process T performing internal events (i.e. events in $\overline{A}$) is guaranteed to terminate in a state after which no events in $\overline{A}$ are possible. The events that are possible are exactly the successors of $t{\restriction}A$ in $T{\restriction}A$. Notice that $\mathbf{a}T$ is transparent with respect to T. System S is called *transparent* if $\mathbf{e}S$ is transparent with respect to $\mathbf{W}(\mathbf{p}S)$.

Property 2.1.19

0 A is transparent with respect to T
$\Rightarrow\ (\mathbf{A}\, t : t \in \mathbf{t}T : (\mathbf{E}\, u : u \in (\overline{A})^* \wedge tu \in \mathbf{t}T : suc(tu, T) = suc(t{\restriction}A, T{\restriction}A)))$

1 A is transparent with respect to T
$\Rightarrow\ (\mathbf{A}\, s : s \in \mathbf{t}T{\restriction}A : (\mathbf{E}\, t : t \in \mathbf{t}T \wedge t{\restriction}A = s : suc(t, T) = suc(s, T{\restriction}A)))$

(End of Property)

Transparence may be characterized differently ([Ka]):

Theorem 2.1.20

A is transparent with respect to T
$\equiv\ (\mathbf{A}\, t : t \in \mathbf{t}T \wedge suc(t, T) \subseteq A : suc(t, T) = suc(t{\restriction}A, T{\restriction}A))$
$\wedge\ (\ A$ is non-divergent with respect to $T\ )$

(End of Theorem)

Combining 2.1.1 and 2.1.8 yields

Theorem 2.1.21

Let T and U be processes. Let A and B be alphabets such that $A \subseteq \mathbf{a}T$, $B \subseteq \mathbf{a}U$, and $\mathbf{a}T \cap \mathbf{a}U = A \cap B$. If A is transparent with respect to T, and B is transparent with respect to U then $A \cup B$ is transparent with respect to $T\,\mathbf{w}\,U$.

(End of Theorem)

Corollary 2.1.22

If R and S are transparent systems and $\mathbf{aW}(\mathbf{p}R) \cap \mathbf{aW}(\mathbf{p}S) = \mathbf{e}R \cap \mathbf{e}S$, then $R \parallel S$ is transparent.

(End of Corollary)

Combining 2.1.3 and 2.1.11 yields

Theorem 2.1.23

Let T and U be processes such that $\mathbf{a}T$ and $\mathbf{a}U$ are finite. Let A be an alphabet such that $A \subseteq \mathbf{a}U \setminus \mathbf{a}T$. If $\mathbf{a}T \cap \mathbf{a}U$ is transparent with respect to T, A is transparent with respect to U, and $U\restriction(\mathbf{a}T \cap \mathbf{a}U) \subseteq T\restriction(\mathbf{a}T \cap \mathbf{a}U)$, then A is transparent with respect to $T \,\mathbf{w}\, U$.

(End of Theorem)

Corollary 2.1.24

Let R and S be systems such that $\mathbf{aW}(\mathbf{p}R)$ and $\mathbf{aW}(\mathbf{p}S)$ are finite, $\mathbf{aW}(\mathbf{p}R) \cap \mathbf{aW}(\mathbf{p}S) = \mathbf{e}R \cap \mathbf{e}S$, and $\mathbf{e}R \subseteq \mathbf{e}S$. If R and $S\restriction(\mathbf{e}S \setminus \mathbf{e}R)$ are transparent systems, and $\mathrm{PR}(S)\restriction\mathbf{e}R \subseteq \mathrm{PR}(R)$ then $(R \parallel S)\restriction(\mathbf{e}S \setminus \mathbf{e}R)$ is a transparent system.

(End of Corollary)

Example 2.1.13 shows that the above theorem does not hold in case the alphabets are infinite. Combining 2.1.5 and 2.1.15 yields

Theorem 2.1.25

Let T be a process such $\mathbf{a}T$ is finite. Let A and B be alphabets such that $B \subseteq A \subseteq \mathbf{a}T$. If A is transparent with respect to T then

$$B \text{ is transparent with respect to } T \;\equiv\; B \text{ transparent with respect to } T\restriction A$$

(End of Theorem)

The following theorem shows how the results in theorem 2.1.25 change if the condition $\mathbf{a}T$ is finite is dropped.

Theorem 2.1.26

Let T be a process. Let A and B be alphabets such that $B \subseteq A \subseteq \mathbf{a}T$. If A is transparent with respect to T then

$$B \text{ is transparent with respect to } T \;\Rightarrow\; B \text{ is transparent with respect to } T\restriction A$$

Proof

Due to theorem 2.1.5 it remains to show that non-divergence of B with respect to T implies non-divergence of B with respect to $T\restriction A$.
Assume B is divergent with respect to $T\restriction A$. Choose $r \in \mathbf{t}T\restriction A$ such that

$$(\mathbf{A}\, n : n \geq 0 : (\mathbf{E}\, v : v \in (A \setminus B)^* \wedge rv \in \mathbf{t}T\restriction A : \ell(v) \geq n))$$

Choose $t \in \mathbf{t}T$ such that $t\restriction A = r$. Let $n \geq 0$. Let $v \in (A \setminus B)^*$ be such that $rv \in \mathbf{t}T\restriction A$ and $\ell(v) \geq n$. Let $u \in (\overline{B})^*$ be such that $tu \in \mathbf{t}T$ and $(tu)\restriction A = rv$ (such traces exist since A is transparent with respect to T and $B \subseteq A$). Since $\ell(u) \geq \ell(u\restriction A)$ and $u\restriction A = v$, we have $\ell(u) \geq n$. Hence B is divergent with respect to T.

(End of Proof)

The reverse implication in the above theorem does not hold as the following example shows.

Example 2.1.27

$T = \mathrm{PREF}(\langle \{a, y\} \cup \{ x_i \mid i \geq 0 \}, \{ ax_i y^i x_i a \mid i \geq 0 \} \rangle)$
$\{a\} \cup \{ x_i \mid i \geq 0 \}$ is transparent with respect to T
$\{a\}$ is not transparent with respect to T (divergence)
$\{a\}$ is transparent with respect to $T\restriction(\{a\} \cup \{ x_i \mid i \geq 0 \})$

(End of Example)

Corollary 2.1.28

Let S be a system. Let $A \subseteq \mathbf{e}S$. If S is transparent then

0 $S\restriction A$ is a transparent system $\Rightarrow$ A is transparent with respect to $\mathrm{PR}(S)$
1 if $\mathbf{aW}(\mathbf{p}S)$ is finite then
$S\restriction A$ is a transparent system $\equiv$ A is transparent with respect to $\mathrm{PR}(S)$

(End of Corollary)

2.2 Deadlock

Consider the system

$$S = \langle\, \{a_0, a_1, b_0, b_1, c_0, c_1, d_0, d_1\}, \{U_0, U_1, R_0, R_1\}\,\rangle$$

where

$$U_0 = \mathrm{PR}((a_0\,,\, b_0\,;\, c_0\,,\, d_0)^*)$$
$$U_1 = \mathrm{PR}((a_1\,,\, b_1\,;\, c_1\,,\, d_1)^*)$$
$$R_0 = \mathrm{PR}((a_0\,;\, c_0 \mid a_1\,;\, c_1)^*)$$
$$R_1 = \mathrm{PR}((b_0\,;\, d_0 \mid b_1\,;\, d_1)^*)$$

Processes R_0 and R_1 describe resources that may be accessed by processes U_0 and U_1 under mutual exclusion. Both process U_0 and process U_1 first need to gain access to both resources, before they can continue. We observe that trace $a_0\, b_1$ of $\mathbf{W}(\mathbf{p}S)$ has no successors in $\mathbf{W}(\mathbf{p}S)$. However, the projection of $a_0\, b_1$ on the alphabet of each of the processes in the process set of S does have a successor in that process. The system has terminated whereas none of the composing processes has terminated. This situation is referred to as deadlock.

Let T be a process. Let $t \in \mathbf{t}T$. We say that T has *terminated* after t if $\mathit{after}(t, T) = \mathrm{STOP}(\mathbf{a}T)$ or, equivalently, $\mathit{suc}(t, T) = \varnothing$. Process T is called *non-terminating* if after each trace t of $\mathbf{t}T$ process T has not terminated, i.e.

$$(\mathbf{A}\, t : t \in \mathbf{t}T : \mathit{suc}(t, T) \neq \varnothing\,)$$

Notice that a process that is not non-terminating *may* terminate.

Theorem 2.2.0

Let T be a non-terminating process. Then there exists a non-terminating process S such that $S \subseteq T$ and $(\mathbf{A}\, s, t : s \in \mathbf{t}S \wedge t \in \mathbf{t}S : s \leq t \vee t \leq s\,)$, i.e. $\mathbf{t}S$ is totally ordered.

(End of Theorem)

In the sequel X is a set of processes.

Property 2.2.1

$$(\mathbf{A}\, t : t \in \mathbf{tW}(X) : (\mathbf{A}\, T : T \in X : \mathit{suc}(t \restriction \mathbf{a}T, T) = \varnothing\,) \;\Rightarrow\; \mathit{suc}(t, \mathbf{W}(X)) = \varnothing\,)$$

(End of Property)

Set X is called *lockfree* if the reverse implication holds, i.e.

$$\begin{aligned}&\mathit{lockfree}(X) \equiv \\ &(\mathbf{A}\, t : t \in \mathbf{tW}(X) : \mathit{suc}(t, \mathbf{W}(X)) = \varnothing \Rightarrow (\mathbf{A}\, T : T \in X : \mathit{suc}(t \restriction \mathbf{a}T, T) = \varnothing))\end{aligned}$$

Notice that due to property 2.2.1 the implication sign in the definition of lockfree may be replaced by an equivalence sign. If X is not lockfree, we say that X has danger of lock.

System S is called lockfree if $\mathit{lockfree}(\mathbf{p}S)$ holds.

Property 2.2.2

0 $\mathit{lockfree}(\varnothing)$

1 $\mathit{lockfree}(\{T\})$ for any process T

(End of Property)

Theorem 2.2.3

0 $\mathit{lockfree}(X) \equiv (\mathbf{A}\, T : T \in X : \mathit{lockfree}(\{T, \mathbf{W}(X \setminus \{T\})\}))$

1 $\mathit{lockfree}(X) \equiv (\mathbf{A}\, Y : Y \subseteq X : \mathit{lockfree}(\{\mathbf{W}(Y), \mathbf{W}(X \setminus Y)\}))$

(End of Theorem)

Theorem 2.2.4

Let X and Y be sets of processes. If $\mathit{lockfree}(X)$ and $\mathit{lockfree}(Y)$ then

$$\mathit{lockfree}(X \cup Y) \equiv \mathit{lockfree}(\{\mathbf{W}(X), \mathbf{W}(Y)\})$$

Proof

Assume $\mathit{lockfree}(X)$ and $\mathit{lockfree}(Y)$.

$$\begin{aligned}&\mathit{lockfree}(X \cup Y) \\ =\ &\quad \{\text{ definition } \mathit{lockfree} \} \\ &(\mathbf{A}\, t : t \in \mathbf{tW}(X \cup Y) \\ &\qquad : \mathit{suc}(t, \mathbf{W}(X \cup Y)) = \varnothing \equiv (\mathbf{A}\, T : T \in X \cup Y : \mathit{suc}(t \restriction \mathbf{a}T, T) = \varnothing)) \\ =\ &\quad \{\, \mathit{lockfree}(X),\ \mathit{lockfree}(Y) \,\} \\ &(\mathbf{A}\, t : t \in \mathbf{t}(\mathbf{W}(X)\, \mathbf{w}\, \mathbf{W}(Y)) \\ &\qquad : \mathit{suc}(t, \mathbf{W}(X)\, \mathbf{w}\, \mathbf{W}(Y)) = \varnothing \\ &\qquad\quad \equiv \mathit{suc}(t \restriction \mathbf{aW}(X), \mathbf{W}(X)) = \varnothing \wedge \mathit{suc}(t \restriction \mathbf{aW}(Y), \mathbf{W}(Y)) = \varnothing)\end{aligned}$$

$$
\begin{array}{ll}
= & \{ \text{definition } \textit{lockfree} \} \\
& \textit{lockfree}(\{\mathbf{W}(X), \mathbf{W}(Y)\})
\end{array}
$$

(End of Proof)

The next theorem shows that processes may be projected on transparent alphabets when one investigates the absence of lock.

Theorem 2.2.5

Let T and U be processes. Let A and B be alphabets such that $A \subseteq \mathbf{a}T$, $B \subseteq \mathbf{a}U$, and $A \cap B = \mathbf{a}T \cap \mathbf{a}U$. If A is transparent with respect to T, and B is transparent with respect to U then

$$\textit{lockfree}(\{T, U\}) \equiv \textit{lockfree}(\{T\restriction A, U\restriction B\})$$

Proof

Assume A is transparent with respect to T and B is transparent with respect to U. From theorem 2.1.21 we infer that $A \cup B$ is transparent with respect to $T \,\mathbf{w}\, U$. Due to theorem 1.1.11 we have $(T \,\mathbf{w}\, U)\restriction(A \cup B) = T\restriction A \,\mathbf{w}\, U\restriction B$.

i) Assume $\textit{lockfree}(\{T, U\})$. Let $u \in \mathbf{t}(T\restriction A \,\mathbf{w}\, U\restriction B)$ be such that $\textit{suc}(u, T\restriction A \,\mathbf{w}\, U\restriction B) = \varnothing$. Choose $t \in \mathbf{t}(T \,\mathbf{w}\, U)$ such that

$$t\restriction(A \cup B) = u \wedge \textit{suc}(t, T \,\mathbf{w}\, U) = \textit{suc}(u, T\restriction A \,\mathbf{w}\, U\restriction B)$$

(theorem 2.1.19.1).
Since $\textit{suc}(t, T \,\mathbf{w}\, U) = \varnothing$ and $\textit{lockfree}(\{T, U\})$ we now have $\textit{suc}(t\restriction \mathbf{a}T, T) = \varnothing \subseteq A$ and $\textit{suc}(t\restriction \mathbf{a}U, U) = \varnothing \subseteq B$.
By theorem 2.1.20 it follows that $\textit{suc}(t\restriction A, T\restriction A) = \varnothing$ and $\textit{suc}(t\restriction B, U\restriction B) = \varnothing$.
Since $t\restriction(A \cup B) = u$ this implies that $\textit{suc}(u\restriction A, T\restriction A) = \varnothing$ and $\textit{suc}(u\restriction B, U\restriction B) = \varnothing$.

ii) Assume $\textit{lockfree}(\{T\restriction A, U\restriction B\})$. Let $t \in \mathbf{t}(T \,\mathbf{w}\, U)$ be such that $\textit{suc}(t, T \,\mathbf{w}\, U) = \varnothing$. By theorem 1.1.17 we have $\textit{suc}(t\restriction \mathbf{a}T, T) \subseteq \mathbf{a}T \cap \mathbf{a}U \subseteq A$ and $\textit{suc}(t\restriction \mathbf{a}U, U) \subseteq \mathbf{a}T \cap \mathbf{a}U \subseteq B$. We derive

$$
\begin{array}{ll}
& \textit{suc}(t, T \,\mathbf{w}\, U) = \varnothing \\
= & \{ \textit{suc}(t, T \,\mathbf{w}\, U) \subseteq A \cup B, \text{ theorem } 2.1.20 \} \\
& \textit{suc}(t\restriction(A \cup B), (T \,\mathbf{w}\, U)\restriction(A \cup B)) = \varnothing \\
= & \{ \text{calculus} \} \\
& \textit{suc}(t\restriction(A \cup B), T\restriction A \,\mathbf{w}\, U\restriction B) = \varnothing
\end{array}
$$

$$
\begin{aligned}
&= \quad \{\, \mathit{lockfree}(\{T\restriction A, U\restriction B\})\,\} \\
&\quad \mathit{suc}(t\restriction A, T\restriction A) = \varnothing \wedge \mathit{suc}(t\restriction B, U\restriction B) = \varnothing \\
&= \quad \{\, \mathit{suc}(t\restriction \mathbf{a}T, T) \subseteq A,\ \mathit{suc}(t\restriction \mathbf{a}U, U) \subseteq B, \text{ theorem 2.1.20}\,\} \\
&\quad \mathit{suc}(t\restriction \mathbf{a}T, T) = \varnothing \wedge \mathit{suc}(t\restriction \mathbf{a}U, U) = \varnothing
\end{aligned}
$$

(End of Proof)

Corollary 2.2.6

Let T and U be processes such that $\mathbf{a}T \cap \mathbf{a}U$ is transparent with respect to both T and U. Then

$$\mathit{lockfree}(\{T, U\}) \ \equiv \ \mathit{lockfree}(\,\{T\restriction(\mathbf{a}T \cap \mathbf{a}U), U\restriction(\mathbf{a}T \cap \mathbf{a}U)\}\,)$$

If, moreover, $T\restriction(\mathbf{a}T \cap \mathbf{a}U) = U\restriction(\mathbf{a}T \cap \mathbf{a}U)$, then $\mathit{lockfree}(\{T, U\})$.

(End of Corollary)

Combining theorems 2.2.4 and 2.2.5 yields the following two corollaries.

Corollary 2.2.7

Let X and Y be sets of processes. Let A and B be alphabets such that $A \subseteq \mathbf{aW}(X)$, $B \subseteq \mathbf{aW}(Y)$, and $\mathbf{aW}(X) \cap \mathbf{aW}(Y) = A \cap B$. If A is transparent with respect to $\mathbf{W}(X)$, B is transparent with respect to $\mathbf{W}(Y)$, and both $\mathit{lockfree}(X)$ and $\mathit{lockfree}(Y)$ hold then

$$\mathit{lockfree}(X \cup Y) \ \equiv \ \mathit{lockfree}(\,\{\mathbf{W}(X)\restriction A, \mathbf{W}(Y)\restriction B\}\,)$$

(End of Corollary)

Corollary 2.2.8

Let R and S be systems such that $\mathbf{aW}(\mathbf{p}R) \cap \mathbf{aW}(\mathbf{p}S) = \mathbf{e}R \cap \mathbf{e}S$. If R and S are lockfree and transparent, then

$$R \parallel S \text{ is a lockfree system} \ \equiv \ \mathit{lockfree}(\,\{\mathrm{PR}(R), \mathrm{PR}(S)\}\,)$$

(End of Corollary)

2.3 Conservative processes

Consider process $T = \mathrm{PR}((a\,,\, b;\, c)^*)$. The states of T are

$$[\varepsilon]_T = \{\, t \mid t \in \mathbf{t}T \;\wedge\; \ell(t\restriction a) = \ell(t\restriction b) = \ell(t\restriction c)\,\}$$
$$[a]_T = \{\, t \mid t \in \mathbf{t}T \;\wedge\; \ell(t\restriction a) - 1 = \ell(t\restriction b) = \ell(t\restriction c)\,\}$$
$$[b]_T = \{\, t \mid t \in \mathbf{t}T \;\wedge\; \ell(t\restriction b) - 1 = \ell(t\restriction a) = \ell(t\restriction c)\,\}$$
$$[ab]_T = \{\, t \mid t \in \mathbf{t}T \;\wedge\; \ell(t\restriction a) - 1 = \ell(t\restriction b) - 1 = \ell(t\restriction c)\,\}$$

The states of T only depend on the number of occurrences of events. Furthermore, it is easily shown that all subsets of $\mathbf{a}T$ are non-disabling with respect to T. In this section we will introduce a class of processes having these properties. Process T will be an element of that class. Due to the first property they will be called *conservative* processes ([Ve86]).

Process T is called *persistent* if

$$(\mathbf{A}\, t, a, b : ta \in \mathbf{t}T \;\wedge\; tb \in \mathbf{t}T \;\wedge\; a \neq b : tab \in \mathbf{t}T \;\wedge\; tba \in \mathbf{t}T\,)$$

Process T is called *commutative* if

$$(\mathbf{A}\, t, a, b : tab \in \mathbf{t}T \;\wedge\; tba \in \mathbf{t}T : [tab] = [tba]\,)$$

Process T is called *conservative* if T is both persistent and commutative, i.e.

$$(\mathbf{A}\, t, a, b : ta \in \mathbf{t}T \;\wedge\; tb \in \mathbf{t}T \;\wedge\; a \neq b : tab \in \mathbf{t}T \;\wedge\; tba \in \mathbf{t}T \;\wedge\; [tab] = [tba]\,)$$

In order to give a different characterization of conservativity we first define a new operation on traces. Let t be a trace and B be a bag (multiset) of symbols. Trace t *minus* bag B, denoted by $t \setminus B$, (see [Ve85]) is obtained by removing from t from left to right $\ell(t\restriction a)\,\mathbf{min}\,(\mathbf{N}\, b : b \in B : b = a)$ occurrences of a for each symbol a in B. It is defined by

$$\begin{array}{lcll} \varepsilon \setminus B & = & \varepsilon & \\ as \setminus B & = & a(s \setminus B) & a \notin B \\ as \setminus B & = & s \setminus (B - \{a\}) & a \in B \end{array}$$

Projection on an alphabet can be expressed in terms of the minus operator. Let A be an alphabet. If B is the bag consisting of the elements of $\Omega \setminus A$, each infinitely often, then

$$t\restriction A = t \setminus B$$

For trace t the *bag of symbols* of t, denoted by $\#t$, is defined by

$$\begin{aligned} \#\varepsilon &= \varnothing \\ \#as &= \{a\} + \#s \end{aligned}$$

Notice that symbol a occurs $\ell(t\restriction a)$ times in $\#t$.
In the following s, t, and u are traces, A is an alphabet, and B and C are bags.

Property 2.3.0 (bag of symbols of a trace)

0 $\#st = \#s + \#t$
1 $\#(s \setminus B) = \#s - B$
2 $s \leq t \Rightarrow \#s \subseteq \#t$
$\#s \subseteq \#t \Rightarrow \ell(s) \leq \ell(t)$

(End of Property)

Property 2.3.1 (minus)

0 $s \setminus B = s \setminus (B \cap \#s)$
1 $s \setminus B = s \equiv B \cap \#s = \varnothing$ $\qquad (s \setminus \varnothing = s\,,\ s \setminus \#\varepsilon = s)$
2 $s \setminus B = \varepsilon \equiv \#s \subseteq B$ $\qquad (s \setminus \#s = \varepsilon)$
3 $\ell(s \setminus B) = \ell(s) - |B \cap \#s|$
$\ell(s) - |B| \leq \ell(s \setminus B) \leq \ell(s)$
4 $st \setminus B = (s \setminus B)(t \setminus (B - \#s))$
5 $s \leq t \Rightarrow s \setminus B \leq t \setminus B$
6 $(s \setminus B) \setminus C = s \setminus (B + C) = (s \setminus C) \setminus B$
$(s \setminus \#t) \setminus \#u = s \setminus \#(tu) = s \setminus \#(ut) = (s \setminus \#u) \setminus \#t$
7 $(s \setminus \#t)\restriction A = (s\restriction A) \setminus \#(t\restriction A)$
8 $(st) \setminus \#(su) = t \setminus \#u$

(End of Property)

Example 2.3.2

Let $s = a\,b\,a\,c\,d\,b\,a$. We have

$s\restriction\{a, c\} = a\,a\,c\,a$
$\#s = \{a, a, a, b, b, c, d\}$
$s \setminus \{a, c\} = b\,a\,d\,b\,a$ $\qquad s \setminus \{a, a, b, b\} = c\,d\,a$

(End of Example)

Lemma 2.3.3

If T is a conservative process then

$$(\mathbf{A}\, t, u, a : ta \in \mathbf{t}T \wedge tu \in \mathbf{t}T : ta(u \setminus \#a) \in \mathbf{t}T)$$

Proof

Let T be a conservative process. The proof is given by induction on the length of trace u. Let $tu \in \mathbf{t}T$.

base $\ell(u) = 0$

We have $u = \varepsilon$. Let $ta \in \mathbf{t}T$. Then $ta(u \setminus \#a) = ta \in \mathbf{t}T$.

step $\ell(u) > 0$

Assume

$$(\mathbf{A}\, t, w, a : ta \in \mathbf{t}T \wedge tw \in \mathbf{t}T \wedge \ell(w) < \ell(u) : ta(w \setminus \#a) \in \mathbf{t}T)$$

Let $ta \in \mathbf{t}T$ and $u = bv$. We distinguish two cases.

i) $a = b$

$$\begin{array}{ll} & ta((bv) \setminus \#a) \\ = & \quad \{\, a = b, \text{ property 2.3.1} \,\} \\ & tav \\ \in & \quad \{\, a = b,\ tu \in \mathbf{t}T \,\} \\ & \mathbf{t}T \end{array}$$

ii) $a \neq b$ We derive

$$\begin{array}{ll} & ta \in \mathbf{t}T \wedge tbv \in \mathbf{t}T \wedge a \neq b \\ \Rightarrow & \quad \{\, T \text{ is a conservative process} \,\} \\ & tab \in \mathbf{t}T \wedge tba \in \mathbf{t}T \wedge [tab] = [tba] \wedge tbv \in \mathbf{t}T \wedge a \neq b \\ \Rightarrow & \quad \{\, \text{induction hypothesis} \,\} \\ & tab \in \mathbf{t}T \wedge tba(v \setminus \#a) \in \mathbf{t}T \wedge [tab] = [tba] \wedge a \neq b \\ \Rightarrow & \quad \{\, \text{calculus} \,\} \\ & tab(v \setminus \#a) \in \mathbf{t}T \wedge a \neq b \\ \Rightarrow & \quad \{\, \text{property 2.3.1} \,\} \\ & ta(bv \setminus \#a) \in \mathbf{t}T \end{array}$$

(End of Proof)

Theorem 2.3.4

T is conservative $\equiv$ $(\mathbf{A}\, s,t : s \in \mathbf{t}T \wedge t \in \mathbf{t}T : s(t \setminus \#s) \in \mathbf{t}T)$

Proof

i) Let T be conservative. We prove the right hand side by induction on the length of trace s. Let $s \in \mathbf{t}T$.

base $\ell(s) = 0$

Let $t \in \mathbf{t}T$. We derive

$$\begin{array}{ll} & s(t \setminus \#s) \\ = & \quad \{\, s = \varepsilon, \text{ property 2.3.1} \,\} \\ & t \\ \in & \quad \{\, \text{assumption} \,\} \\ & \mathbf{t}T \end{array}$$

step $\ell(s) > 0$

Assume $(\mathbf{A}\, r,t : r \in \mathbf{t}T \wedge t \in \mathbf{t}T \wedge \ell(r) < \ell(s) : r(t \setminus \#r) \in \mathbf{t}T)$.
Let $s = ua$ and $t \in \mathbf{t}T$. We derive

$$\begin{array}{ll} & ua \in \mathbf{t}T \wedge t \in \mathbf{t}T \\ \Rightarrow & \quad \{\, T \text{ is a process, induction hypothesis} \,\} \\ & u(t \setminus \#u) \in \mathbf{t}T \wedge ua \in \mathbf{t}T \\ \Rightarrow & \quad \{\, \text{lemma 2.3.3} \,\} \\ & ua((t \setminus \#u) \setminus \#a) \in \mathbf{t}T \\ = & \quad \{\, \text{property 2.3.1} \,\} \\ & ua(t \setminus \#(ua)) \in \mathbf{t}T \end{array}$$

ii) Let $(\mathbf{A}\, s,t : s \in \mathbf{t}T \wedge t \in \mathbf{t}T : s(t \setminus \#s) \in \mathbf{t}T)$. Let $ta \in \mathbf{t}T$, $tb \in \mathbf{t}T$, and $a \neq b$. We derive

$$\begin{array}{ll} & ta \in \mathbf{t}T \wedge tb \in \mathbf{t}T \\ \Rightarrow & \quad \{\, \text{assumption} \,\} \\ & ta((tb) \setminus \#(ta)) \in \mathbf{t}T \\ = & \quad \{\, \text{property 2.3.1, } a \neq b \,\} \\ & tab \in \mathbf{t}T \end{array}$$

For reasons of symmetry we infer that $tab \in \mathbf{t}T$ and $tba \in \mathbf{t}T$. Furthermore, we derive

$$
\begin{array}{ll}
 & tabu \in \mathbf{t}T \\
\Rightarrow & \quad \{\, tba \in \mathbf{t}T, \text{ assumption} \,\} \\
 & tba((tabu) \setminus \#(tba)) \in \mathbf{t}T \\
= & \quad \{\, \text{property 2.3.1} \,\} \\
 & tbau \in \mathbf{t}T
\end{array}
$$

Likewise, we have $tbau \in \mathbf{t}T \Rightarrow tabu \in \mathbf{t}T$.
Hence, $[tab] = [tba]$.

(End of Proof)

From theorem 2.3.4 and property 2.3.1 the next theorems follow.

Theorem 2.3.5

Let T be a process. Let A be an alphabet. If T is conservative then

0 $(\mathbf{A}\, s,t : s \in \mathbf{t}T \wedge t \in \mathbf{t}T \wedge \#s = \#t : [s] = [t])$

1 $T\restriction A$ is conservative

(End of Theorem)

Theorem 2.3.6

If X is a set of conservative processes then $\mathbf{W}(X)$ is a conservative process.

(End of Theorem)

If S is a system such that $(\mathbf{A}\, T : T \in \mathbf{p}S : T \text{ is conservative})$, then due to theorems 2.3.5 and 2.3.6 process $\mathbf{W}(\mathbf{p}S)$ and, hence, process $\mathrm{PR}(S)$ are conservative.

Theorem 2.3.7

If $(T_n)_{n\geq 0}$ is a sequence of conservative processes such that $(\mathbf{A}\, n : n \geq 0 : T_n \subseteq T_{n+1})$ then $(\cup n : n \geq 0 : T_n)$ is conservative.

(End of Theorem)

The next theorem shows that all subsets of the alphabet of a conservative process are non-disabling with respect to that process.

Theorem 2.3.8

Let T be a process. Let A be an alphabet such that $A \subseteq \mathbf{a}T$. If T is conservative then A is non-disabling with respect to T.

Proof

Assume T is conservative.
Let $t \in \mathbf{t}T$. Property 1.1.3 yields $after(t,T)\restriction A \subseteq after(t\restriction A, T\restriction A)$.
We will show that $after(t\restriction A, T\restriction A) \subseteq after(t,T)\restriction A$. Let $u \in \mathbf{t}after(t\restriction A, T\restriction A)$. Choose r and s such that $rs \in \mathbf{t}T$, $r\restriction A = t\restriction A$, and $s\restriction A = u$. By theorem 2.3.4 we have that $t((rs) \setminus \#t) \in \mathbf{t}T$. Furthermore,

$$
\begin{array}{ll}
 & (rs \setminus \#t)\restriction A \\
= & \quad \{\text{property 2.3.1}\} \\
 & (r\restriction A)(s\restriction A) \setminus \#(t\restriction A) \\
= & \quad \{r\restriction A = t\restriction A, \text{property 2.3.1}\} \\
 & s\restriction A
\end{array}
$$

Since $s\restriction A = u$, we conclude $u \in \mathbf{t}after(t,T)\restriction A$.

(End of Proof)

Corollary 2.3.9

If S is a system such that $(\mathbf{A}\, T : T \in \mathbf{p}S : T \text{ is conservative})$
then system S is non-disabling.

(End of Corollary)

The next theorems show some results concerning (non-)termination and absence of deadlock. First, we show that for conservative T the negation of "T is non-terminating" is "T terminates".

Theorem 2.3.10

Let T be a conservative process. We have

0 $(\mathbf{A}\, t : t \in \mathbf{t}T : suc(t,T) = \varnothing \;\equiv\; (\mathbf{A}\, s : s \in \mathbf{t}T : \#s \subseteq \#t))$
$(\mathbf{A}\, s,t : s \in \mathbf{t}T \wedge t \in \mathbf{t}T \wedge suc(s,T) = \varnothing \wedge suc(t,T) = \varnothing : \#s = \#t)$

1 $\neg(T \text{ is non-terminating}) \;\equiv\; (\mathbf{E}\, t : t \in \mathbf{t}T : (\mathbf{A}\, s : s \in \mathbf{t}T : \ell(s) \leq \ell(t)))$

2 The following three assertions are equivalent

a. T is non-terminating
b. $(\mathbf{E}\, S : S \subseteq T : S \text{ is non-terminating})$
c. $(\mathbf{E}\, S : S \subseteq T \wedge (\mathbf{A}\, s,t : s \in \mathbf{t}S \wedge t \in \mathbf{t}S : s \leq t \vee t \leq s)$
$: S \text{ is non-terminating})$

Proof

0 Let $t \in \mathbf{t}T$.

i) Assume $suc(t,T) = \emptyset$. We derive

$s \in \mathbf{t}T$
$\Rightarrow$ { T is conservative }
$t(s \setminus \#t) \in \mathbf{t}T$
$=$ { $suc(t,T) = \emptyset$ }
$s \setminus \#t = \varepsilon$
$=$ { property 2.3.1 }
$\#s \subseteq \#t$

ii) Assume $(\mathbf{A}\, s : s \in \mathbf{t}T : \#s \subseteq \#t)$. We derive

$a \in suc(t,T)$
$=$ { definition successor set }
$ta \in \mathbf{t}T$
$\Rightarrow$ { assumption }
$\#(ta) \subseteq \#t$
$=$ { definition bag }
false

The second assertion is a direct consequence of the first one.

1 We derive

$\neg(T \text{ is non-terminating})$
$=$ { definition }
$(\mathbf{E}\, t : t \in \mathbf{t}T : suc(t,T) = \emptyset)$
$=$ { 0 }
$(\mathbf{E}\, t : t \in \mathbf{t}T : (\mathbf{A}\, s : s \in \mathbf{t}T : \#s \subseteq \#t))$
$\Rightarrow$ { property 2.3.0 }
$(\mathbf{E}\, t : t \in \mathbf{t}T : (\mathbf{A}\, s : s \in \mathbf{t}T : \ell(s) \leq \ell(t)))$

$$\begin{array}{ll} \Rightarrow & \{\,(\mathbf{A}\,a : ta \in \mathbf{t}T : \ell(ta) > \ell(t))\,\} \\ & (\mathbf{E}\,t : t \in \mathbf{t}T : suc(t,T) = \varnothing\,) \end{array}$$

2

$a \Rightarrow b$ Take $S = T$.

$b \Rightarrow c$ Theorem 2.2.0

$c \Rightarrow b$ Trivial

$b \Rightarrow a$ Let $S \subseteq T$ be such that S is non-terminating. Let $t \in \mathbf{t}T$. Choose $s \in \mathbf{t}S$ such that $\ell(s) > \ell(t)$. Since $S \subseteq T$ and T is conservative, we have $t(s \setminus \#t) \in \mathbf{t}T$. Furthermore, we have $\ell(s \setminus \#t) \geq \ell(s) - \ell(t) > 0$ which implies $s \setminus \#t \neq \varepsilon$. Hence, $suc(t,T) \neq \varnothing$.

(End of Proof)

The next theorem provides a method to prove the absence of deadlock.

Theorem 2.3.11

Let X be a set of conservative processes. If

$$(\mathbf{A}\,t,T : T \in X \wedge t \in \mathbf{t}T : (\mathbf{E}\,s : s \in \mathbf{tW}(X) : \ell(s \restriction \mathbf{a}T) = \ell(t))\,)$$

then $lockfree(X)$ holds.

Proof

Assume

$$(\mathbf{A}\,t,T : T \in X \wedge t \in \mathbf{t}T : (\mathbf{E}\,s : s \in \mathbf{tW}(X) : \ell(s \restriction \mathbf{a}T) = \ell(t))\,)$$

Let $t \in \mathbf{tW}(X)$. Let $T \in X$ and $a \in \mathbf{a}T$ be such that $a \in suc(t \restriction \mathbf{a}T, T)$. Choose $s \in \mathbf{tW}(X)$ such that $\ell(s) = \ell((t \restriction \mathbf{a}T)a) = \ell(t \restriction \mathbf{a}T) + 1$. By theorem 2.3.6 we have that $\mathbf{W}(X)$ is conservative. Hence, $t(s \setminus \#t) \in \mathbf{tW}(X)$. Furthermore,

$$\begin{array}{ll} & \ell(s \setminus \#t) \\ \geq & \{\,\text{calculus}\,\} \\ & \ell((s \setminus \#t) \restriction \mathbf{a}T) \\ = & \{\,\text{property 2.3.1}\,\} \\ & \ell((s \restriction \mathbf{a}T) \setminus \#(t \restriction \mathbf{a}T)) \end{array}$$

$$
\begin{array}{ll}
\geq & \{\text{ property 2.3.1 }\} \\
& \ell(s\restriction \mathbf{a}T) - \ell(t\restriction \mathbf{a}T) \\
\geq & \{\text{ choice of } s\} \\
& 1
\end{array}
$$

Therefore, $s \setminus \#t \neq \varepsilon$ and, hence, $suc(t, \mathbf{W}(X)) \neq \varnothing$.

(End of Proof)

2.4 Cubic processes

In this section we introduce the cubic processes forming a subclass of the conservative processes. A process T is said to be *cubic* if T is conservative and satisfies

$$(\mathbf{A}\, t, a, b, c : tac \in \mathbf{t}T \wedge tbc \in \mathbf{t}T \wedge a \neq b : tc \in \mathbf{t}T)$$

Example 2.4.0

Process PR(a ; b; c | a ; c; b | b; a; c | b; c; a) is conservative but not cubic.
Process PR(a , b, c) is cubic.

(End of Example)

Theorem 2.4.1

$$
\begin{array}{l}
T \text{ is cubic } \equiv \text{ (} T \text{ is conservative)} \\
\qquad\qquad \wedge\ (\mathbf{A}\, t, u, v, c : tuc \in \mathbf{t}T \wedge tvc \in \mathbf{t}T \wedge \#u \cap \#v = \varnothing : tc \in \mathbf{t}T)
\end{array}
$$

Proof

Obviously, the right hand side implies the left hand side.

Assume T is cubic. Let $t \in \mathbf{t}T$. Let u, v, and c be such that $tuc \in \mathbf{t}T$, $tvc \in \mathbf{t}T$, and $\#u \cap \#v = \varnothing$. The proof is done by induction on $\ell(u) \cdot \ell(v)$.

base $\ell(u) \cdot \ell(v) = 0$

Then $u = \varepsilon \vee v = \varepsilon$. From this and $tuc \in \mathbf{t}T \wedge tvc \in \mathbf{t}T$ we infer $tc \in \mathbf{t}T$.

step $\ell(u) \cdot \ell(v) > 0$

Assume

$$
\begin{array}{l}
(\mathbf{A}\, r, s, d : trd \in \mathbf{t}T \wedge tsd \in \mathbf{t}T \\
\qquad\qquad \wedge\ \ell(r) \cdot \ell(s) < \ell(u) \cdot \ell(v) \wedge \#r \cap \#s = \varnothing : td \in \mathbf{t}T).
\end{array}
$$

We have $\ell(u) > 0$ and $\ell(v) > 0$. Let $u = ax$ and $v = by$. We distinguish two cases.

i. $a = c \vee b = c$

Since $ta \in \mathbf{t}T \wedge tb \in \mathbf{t}T$ we now have $tc \in \mathbf{t}T$.

ii. $a \neq c \wedge b \neq c$

We derive

$$
\begin{array}{ll}
& taxc \in \mathbf{t}T \wedge tbyc \in \mathbf{t}T \\
= & \quad \{ T \text{ is conservative, theorem 2.3.4} \} \\
& taxc \in \mathbf{t}T \wedge ta(tbyc \setminus \#(ta)) \in \mathbf{t}T \wedge tbyc \in \mathbf{t}T \wedge tb(taxc \setminus \#(tb)) \in \mathbf{t}T \\
= & \quad \{ \text{property 2.3.1, } \#(ax) \cap \#(by) = \varnothing,\ a \neq c,\ b \neq c \} \\
& taxc \in \mathbf{t}T \wedge tabyc \in \mathbf{t}T \wedge tbyc \in \mathbf{t}T \wedge tbaxc \in \mathbf{t}T \\
\Rightarrow & \quad \{ \ell(x){\cdot}\ell(by) < \ell(u){\cdot}\ell(v),\ \ell(ax){\cdot}\ell(y) < \ell(u){\cdot}\ell(v),\ \#(u) \cap \#(v) = \varnothing \} \\
& tac \in \mathbf{t}T \wedge tbc \in \mathbf{t}T \\
\Rightarrow & \quad \{ T \text{ is cubic, } a \neq b \} \\
& tc \in \mathbf{t}T
\end{array}
$$

(End of Proof)

Theorem 2.4.2

Let X be a set of processes. If

$$(*) \quad (\mathbf{A}\, T : T \in X : (\mathbf{A}\, t, a, b, c : tac \in \mathbf{t}T \wedge tbc \in \mathbf{t}T \wedge a \neq b : tc \in \mathbf{t}T))$$

then process $\mathbf{W}(X)$ satisfies

$$(\mathbf{A}\, t, a, b, c : tac \in \mathbf{tW}(X) \wedge tbc \in \mathbf{tW}(X) \wedge a \neq b : tc \in \mathbf{tW}(X))$$

Proof

Assume $(*)$ holds. Let $tac \in \mathbf{tW}(X)$, $tbc \in \mathbf{tW}(X)$, and $a \neq b$. Let $T \in X$. We derive

$$
\begin{array}{ll}
& tac \in \mathbf{tW}(X) \wedge tbc \in \mathbf{tW}(X) \wedge a \neq b \\
\Rightarrow & \quad \{ \text{definition } \mathbf{W} \} \\
& (t\lceil \mathbf{a}T)(a\lceil \mathbf{a}T)(c\lceil \mathbf{a}T) \in \mathbf{t}T \wedge (t\lceil \mathbf{a}T)(b\lceil \mathbf{a}T)(c\lceil \mathbf{a}T) \in \mathbf{t}T \wedge a \neq b \\
\Rightarrow & \quad \{ (a \notin \mathbf{a}T \vee b \notin \mathbf{a}T : a\lceil \mathbf{a}T = \varepsilon \vee b\lceil \mathbf{a}T = \varepsilon),\ (a \in \mathbf{a}T \wedge b \in \mathbf{a}T : (*)) \} \\
& (t\lceil \mathbf{a}T)(c\lceil \mathbf{a}T) \in \mathbf{t}T
\end{array}
$$

Therefore, $(\mathbf{A}\, T : T \in X : (tc)\lceil \mathbf{a}T \in \mathbf{t}T)$. Hence, we have $tc \in \mathbf{tW}(X)$.

(End of Proof)

Combining theorems 2.3.6 and 2.4.2 yields

Corollary 2.4.3

If X is a set of cubic processes, then $\mathbf{W}(X)$ is a cubic process.

(End of Corollary)

In order to prove that projection of a cubic process on an alphabet yields a cubic process we need another characterization of cubic processes. First, we introduce projection on bags of symbols. The *projection* of trace t on *bag* B, denoted by $t \uparrow B$, is defined by

$$\begin{array}{rcll} \varepsilon \uparrow B & = & \varepsilon & \\ as \uparrow B & = & s \uparrow B & a \notin B \\ as \uparrow B & = & a(s \uparrow (B - \{a\})) & a \in B \end{array}$$

Notice that $aa\lceil a = aa$ whereas $aa \uparrow \{a\} = a$.
In the following s, t, and u are traces, A is an alphabet, and B and C are bags.

Property 2.4.4

$$\#(s \uparrow B) = \#s \cap B$$

(End of Property)

Property 2.4.5

0 $s \uparrow B = s \uparrow (B \cap \#s)$

1 $s \uparrow B = s \equiv \#s \subseteq B \qquad (s \uparrow \#s = s)$

2 $s \uparrow B = \varepsilon \equiv \#s \cap B = \emptyset \qquad (s \uparrow \emptyset = \varepsilon)$

3 $\ell(s \uparrow B) \leq \ell(s)$

4 $st \uparrow B = (s \uparrow B)(t \uparrow (B - \#s))$

5 $s \leq t \Rightarrow s \uparrow B \leq t \uparrow B$

6 $(s \uparrow B) \uparrow C = s \uparrow (B \cap C) = (s \uparrow C) \uparrow B$
$(s \uparrow \#t) \uparrow \#u = s \uparrow \#(t \uparrow \#u) = s \uparrow \#(u \uparrow \#t) = (s \uparrow \#u) \uparrow \#t$

7 $(s \uparrow B)\lceil A = (s\lceil A) \uparrow B$
$(s \uparrow \#t)\lceil A = (s\lceil A) \uparrow \#(t\lceil A)$

8 $(st) \uparrow \#(su) = s(t \uparrow \#u)$

9 $(s \uparrow B) \setminus C = (s \setminus C) \uparrow (B - C)$
$(s \setminus B) \uparrow C = (s \uparrow (B + C)) \setminus B$

(End of Property)

Theorem 2.4.6

$$T \text{ is cubic } \equiv (\ T \text{ is conservative }) \wedge (\mathbf{A}\, s,t : s \in \mathbf{t}T \wedge t \in \mathbf{t}T : s \uparrow \#t \in \mathbf{t}T\ )$$

Proof

0 Assume T is conservative and

$$(\mathbf{A}\, s,t : s \in \mathbf{t}T \wedge t \in \mathbf{t}T : s \uparrow \#t \in \mathbf{t}T\ )$$

Let $tac \in \mathbf{t}T$, $tbc \in \mathbf{t}T$, and $a \neq b$. We derive

$$\begin{array}{ll} & (tac) \uparrow \#(tbc) \\ = & \quad \{ \text{property 2.4.5} \} \\ & t(\, ac \uparrow \#(bc)\,) \\ = & \quad \{ a \neq b, \text{ definition } \uparrow \} \\ & tc \end{array}$$

Since $(tac) \uparrow \#(tbc) \in \mathbf{t}T$, we have $tc \in \mathbf{t}T$.

1 Assume T is cubic. We prove that

$$(\mathbf{A}\, s : s \in \mathbf{t}T : (\mathbf{A}\, t : t \in \mathbf{t}T : s \uparrow \#t \in \mathbf{t}T\ ))$$

by induction on the length of trace s. Let $s \in \mathbf{t}T$.

base $\ell(s) = 0$
We have $s = \varepsilon$ and for all $t \in \mathbf{t}T$ $\varepsilon \uparrow \#t = \varepsilon \in \mathbf{t}T$.

step $\ell(s) > 0$
Assume

$$(\mathbf{A}\, r : r \in \mathbf{t}T \wedge \ell(r) < \ell(s) : (\mathbf{A}\, t : t \in \mathbf{t}T : r \uparrow \#t \in \mathbf{t}T\ ))$$

Let $s = ua$. Let $t \in \mathbf{t}T$. We distinguish two cases.
i. $\ell(t\restriction a) \leq \ell(u\restriction a)$.
We derive

$$
\begin{array}{ll}
 & (ua) \uparrow \#t \\
= & \quad \{ \text{property 2.4.5} \} \\
 & (u \uparrow \#t)(a \uparrow (\#t - \#u)) \\
= & \quad \{ \ell(t\restriction a) \leq \ell(u\restriction a) \} \\
 & u \uparrow \#t \\
\in & \quad \{ \text{induction hypothesis} \} \\
 & \mathbf{t}T
\end{array}
$$

ii. $\ell(t\restriction a) > \ell(u\restriction a)$
Let $t \setminus \#u = vaw$ such that $\ell(v\restriction a) = \varepsilon$.

$$
\begin{array}{ll}
 & ua \in \mathbf{t}T \wedge t \in \mathbf{t}T \\
= & \quad \{ \text{induction hypothesis} \} \\
 & u \uparrow \#t \in \mathbf{t}T \wedge ua \in \mathbf{t}T \wedge t \in \mathbf{t}T \\
\Rightarrow & \quad \{ T \text{ is conservative, theorem 2.3.4} \} \\
 & (u \uparrow \#t)(ua \setminus \#(u \uparrow \#t)) \in \mathbf{t}T \wedge (u \uparrow \#t)(t \setminus \#(u \uparrow \#t)) \in \mathbf{t}T \\
= & \quad \{ \text{property 2.4.4, property 2.4.5, property 2.3.1, } \ell(t\restriction a) > \ell(u\restriction a) \} \\
 & (u \uparrow \#t)(u \setminus \#t)a \in \mathbf{t}T \wedge (u \uparrow \#t)(t \setminus \#u) \in \mathbf{t}T \\
\Rightarrow & \quad \{ t \setminus \#u = vaw \} \\
 & (u \uparrow \#t)(u \setminus \#t)a \in \mathbf{t}T \wedge (u \uparrow \#t)va \in \mathbf{t}T \\
\Rightarrow & \quad \{ \text{theorem 2.4.1, } \#(u \setminus \#t) \cap \#v = \varnothing \} \\
 & (u \uparrow \#t)a \in \mathbf{t}T \\
= & \quad \{ \text{definition } \uparrow, \text{ property 2.4.5, } \ell(t\restriction a) > \ell(u\restriction a) \} \\
 & ua \uparrow \#t \in \mathbf{t}T
\end{array}
$$

(End of Proof)

Theorem 2.4.7

If T is cubic then $T\restriction A$ is cubic.

Proof

Let T be a cubic process. By theorem 2.3.5 we have that $T\restriction A$ is conservative. Let $u \in \mathbf{t}T\restriction A$ and $v \in \mathbf{t}T\restriction A$. Let $s \in \mathbf{t}T$ and $t \in \mathbf{t}T$ such that $s\restriction A = u$ and $t\restriction A = v$. We derive

$$
\begin{array}{ll}
 & s \in \mathbf{t}T \wedge t \in \mathbf{t}T \\
\Rightarrow & \quad \{ T \text{ is cubic, theorem 2.4.6} \}
\end{array}
$$

$$
\begin{array}{ll}
& s \uparrow \#t \in \mathbf{t}T \\
\Rightarrow & \quad \{ \text{projection} \} \\
& (s \uparrow \#t)\restriction A \in \mathbf{t}T\restriction A \\
= & \quad \{ \text{property 2.4.5} \} \\
& (s\restriction A) \uparrow \#(t\restriction A) \in \mathbf{t}T\restriction A \\
= & \quad \{ s\restriction A = u,\ t\restriction A = v \} \\
& u \uparrow \#v \in \mathbf{t}T\restriction A
\end{array}
$$

(End of Proof)

We introduce *restricted commands*, forming an important subclass of the commands, and show that their processes are cubic. Restricted commands are defined inductively by the following rules.

- ε is a restricted command
- a is restricted command for all symbols a
- if S is a restricted command not containing any star then S^* and S^0 are restricted commands
- if S and T are restricted commands and S contains no star then $S\,;\,T$ is a restricted command
- if S and T are restricted commands such that $\mathbf{a}\mathrm{TR}(S) \cap \mathbf{a}\mathrm{TR}(T) = \varnothing$ then $S\,,\,T$ is a restricted command

Property 2.4.8

Let S be a restricted command. If S contains no stars then for every alphabet A

$$(\mathbf{A}\, s,t : s \in \mathbf{t}\mathrm{TR}(S) \wedge t \in \mathbf{t}\mathrm{TR}(S) : \ell(s\restriction A) = \ell(t\restriction A))$$

(End of Property)

The projection of restricted command S on alphabet A, denoted by $S\restriction A$, is defined inductively as follows.

$$
\begin{array}{lll}
\varepsilon\restriction A = \varepsilon & & \\
a\restriction A = a & & a \in A \\
a\restriction A = \varepsilon & & a \notin A
\end{array}
$$

$$S^*\restriction A = (S\restriction A)^*$$
$$(S\,;\,T)\restriction A = (S\restriction A)\,;\,(T\restriction A)$$
$$(S\,,\,T)\restriction A = (S\restriction A)\,,\,(T\restriction A) \qquad \mathbf{aTR}(S)\cap\mathbf{aTR}(T) = \varnothing$$
$$S^0\restriction A = (S\restriction A)^0$$

Property 2.4.9

Let S be a restricted command and A an alphabet. Then $S\restriction A$ is a restricted command and $\mathrm{TR}(S\restriction A) = \mathrm{TR}(S)\restriction A$.

(End of Property)

By induction on the structure of restricted commands one can prove

Theorem 2.4.10

If S is a restricted command, then $\mathrm{PR}(S)$ is cubic.

(End of Theorem)

Corollary 2.4.11

0 Let component c be defined by

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

where S is a restricted command. Then $\mathrm{PR}(c)$ is cubic.

1 Let component c be defined by

com $c(A)$:
sub $p_0 : c_0, p_1 : c_1, \ldots, p_{n-1} : c_{n-1}$ **bus**
$[x_0 = y_0, x_1 = y_1, \ldots, x_{m-1} = y_{m-1}]$
S
moc

where S is a restricted command and for i, $0 \leq i < n$, $\mathrm{PR}(c_i)$ is a cubic process. Then $\mathrm{PR}(c)$ is cubic.

2 Let component c be defined by

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

where S is a restricted command. Then PR(c) is cubic.

(End of Corollary)

2.5 Partial orders and sequence functions

In this section we address the subject of ordering (sequencing) the events of a process. For process T the *set of occurrences* of T, denoted by $occ(T)$, is defined by

$$occ(T) = \{ (a, \ell(t\restriction a)) \mid ta \in \mathbf{t}T \}$$

Notice that

$$(a,i) \in occ(T) \;\equiv\; (\mathbf{E}\, t : ta \in \mathbf{t}T : \ell(t\restriction a) = i\,)$$

In the sequel we show that the ordering of events of process T can be expressed as a partial order on $occ(T)$.

For process T and occurrence $(a,i) \in occ(T)$ the set $pre(a,i,T)$ of all occurrences that have to precede (a,i) in T is defined by

$$pre(a,i,T) = \{ (b,j) \mid b \in \mathbf{a}T \wedge j \geq 0 \wedge (\mathbf{A}\, t : ta \in \mathbf{t}T \wedge \ell(t\restriction a) = i : \ell(t\restriction b) > j\,) \}$$

Observe that $pre(a,i,T) \subseteq occ(T)$.

Property 2.5.0

Let $(a,i) \in occ(T)$ and $(b,j) \in occ(T)$.

0 $pre(a,i,T)$ is finite

1 $(a,i) \notin pre(a,i,T)$
  $(\mathbf{A}\, k : 0 \leq k < i : (a,k) \in pre(a,i,T)\,)$
  $|pre(a,i,T)| \geq i$

2 $(b,j) \in pre(a,i,T) \;\Rightarrow\; pre(b,j,T) \subseteq pre(a,i,T)$

3 $\neg((a,i) \in pre(b,j,T) \wedge (b,j) \in pre(a,i,T))$

(End of Property)

A process defines a partial order on its set of occurrences. The binary relation $<_T$ on $occ(T)$ is defined by

$$(a,i) <_T (b,j) \equiv (a,i) \in pre(b,j,T) \qquad (a,i),(b,j) \in occ(T)$$

From property 2.5.0 it follows that relation $<_T$ is both anti-reflexive and transitive, and, hence,

Theorem 2.5.1

$\langle occ(T), <_T \rangle$ is a partially ordered set.

(End of Theorem)

Theorem 2.5.2

Let T and U be processes with equal alphabets. Let X be a set of processes. Let A be an alphabet. Then

0 $occ(T\restriction A) = \{\, (a,i) \mid (a,i) \in occ(T) \wedge a \in A \,\}$

1 $occ(\mathbf{W}(X)) \subseteq (\cup S : S \in X : occ(S))$

2 $T \subseteq U \Rightarrow occ(T) \subseteq occ(U)$

3 $(\mathbf{A}\, a,b,i,j : (a,i) \in occ(T\restriction A) \wedge (b,j) \in occ(T\restriction A)$
$\quad : (a,i) <_{T\restriction A} (b,j) \equiv (a,i) <_T (b,j))$

4 $(\mathbf{A}\, S : S \in X : (\mathbf{A}\, a,b,i,j : (a,i) \in occ(\mathbf{W}(X)) \cap occ(S)$
$\quad \wedge (b,j) \in occ(\mathbf{W}(X)) \cap occ(S)$
$\quad : (a,i) <_S (b,j) \Rightarrow (a,i) <_{\mathbf{W}(X)} (b,j)))$

5 $T \subseteq U \Rightarrow (\mathbf{A}\, a,b,i,j : (a,i) \in occ(T) \wedge (b,j) \in occ(T)$
$\quad : (a,i) <_U (b,j) \Rightarrow (a,i) <_T (b,j))$

(End of Theorem)

Partial order $<_0$ on $occ(T)$ is said to *respect* partial order $<_1$ on $occ(T)$ if

$$(\mathbf{A}\, a,b,i,j : (a,i) \in occ(T) \wedge (b,j) \in occ(T) \wedge (a,i) <_1 (b,j) : (a,i) <_0 (b,j))$$

We are interested in partial orders on $occ(T)$ that respect partial order $<_T$.
For partial order $<$ on $occ(T)$ process $\mathrm{PR}(T,<)$ is defined to be the least process U satisfying

0 $\quad \mathbf{a}U = \mathbf{a}T$

1 $\quad \varepsilon \in \mathbf{t}U$

2 $\quad (\mathbf{A}\, t,a : t \in \mathbf{t}U \wedge (a,\ell(t\restriction a)) \in occ(T)$
$\qquad \wedge\ (\mathbf{A}\, b,j : (b,j) \in occ(T) \wedge (b,j) < (a,\ell(t\restriction a)) : \ell(t\restriction b) > j\,) : ta \in \mathbf{t}U\,)$

We have

Theorem 2.5.3

0 $\quad T \subseteq \mathrm{PR}(T,<_T)$

1 If partial order $<_0$ on $occ(T)$ respects partial order $<_1$ on $occ(T)$
then $\mathrm{PR}(T,<_0) \subseteq \mathrm{PR}(T,<_1)$

Proof

0 We prove that $\mathbf{t}T \subseteq \mathbf{t}\mathrm{PR}(T,<_T)$ by induction on the length of traces from $\mathbf{t}T$.

base $\quad \ell(t) = 0 \qquad t = \varepsilon \in \mathbf{t}\mathrm{PR}(T,<_T)$

step $\quad \ell(t) > 0$

Assume $(\mathbf{A}\, s : s \in \mathbf{t}T \wedge \ell(s) < \ell(t) : s \in \mathbf{t}\mathrm{PR}(T,<_T)\,)$. Let $t = ua$. We derive

$$
\begin{array}{ll}
 & ua \in \mathbf{t}T \\
= & \quad \{\, \mathrm{T} \text{ is a process, } \ell(u) < \ell(t), \text{ definition } occ \,\} \\
 & u \in \mathbf{t}\mathrm{PR}(T,<_T) \wedge (a,\ell(u\restriction a)) \in occ(T) \wedge ua \in \mathbf{t}T \\
\Rightarrow & \quad \{\, \text{definition } <_T \text{ and } pre \,\} \\
 & u \in \mathbf{t}\mathrm{PR}(T,<_T) \wedge (a,\ell(u\restriction a)) \in occ(T) \\
 & \ \wedge\ (\mathbf{A}\, b,j : (b,j) \in occ(T) \wedge (b,j) <_T (a,\ell(u\restriction a)) : \ell(u\restriction b) > j\,) \\
= & \quad \{\, \text{definition } \mathrm{PR}(T,<_T) \,\} \\
 & ua \in \mathbf{t}\mathrm{PR}(T,<_T)
\end{array}
$$

1 Let partial order $<_0$ on $occ(T)$ respect partial order $<_1$ on $occ(T)$. We prove that $\mathbf{t}\mathrm{PR}(T,<_0) \subseteq \mathbf{t}\mathrm{PR}(T,<_1)$ by induction on the length of traces from $\mathbf{t}\mathrm{PR}(T,<_0)$.

base $\quad \ell(t) = 0 \qquad t = \varepsilon \in \mathbf{t}\mathrm{PR}(T,<_1)$

step $\quad \ell(t) > 0$

Assume $(\mathbf{A}\, s : s \in \mathbf{t}\mathrm{PR}(T,<_0) \wedge \ell(s) < \ell(t) : s \in \mathbf{t}\mathrm{PR}(T,<_1)\,)$. Let $t = ua$. We derive

$$
\begin{array}{ll}
& ua \in \mathbf{t}\mathrm{PR}(T, <_0) \\
= & \quad \{ \text{definition } \mathrm{PR}(T, <_0) \} \\
& u \in \mathbf{t}\mathrm{PR}(T, <_0) \wedge (a, \ell(u\restriction a)) \in occ(T) \\
& \quad \wedge (\mathbf{A}\, b, j : (b, j) \in occ(T) \wedge (b, j) <_0 (a, \ell(u\restriction a)) : \ell(u\restriction b) > j\,) \\
\Rightarrow & \quad \{ \ell(u) < \ell(t),\ <_0 \text{ respects } <_1 \} \\
& u \in \mathbf{t}\mathrm{PR}(T, <_1) \wedge (a, \ell(u\restriction a)) \in occ(T) \\
& \quad \wedge (\mathbf{A}\, b, j : (b, j) \in occ(T) \wedge (b, j) <_1 (a, \ell(u\restriction a)) : \ell(u\restriction b) > j\,) \\
= & \quad \{ \text{definition } \mathrm{PR}(T, <_1) \} \\
& ua \in \mathbf{t}\mathrm{PR}(T, <_1)
\end{array}
$$

(End of Proof)

Theorem 2.5.4

Let T be a process. If $<$ is a partial order on $occ(T)$ then $\mathrm{PR}(T, <)$ is cubic.

Proof

Let $<$ be a partial order on $occ(T)$.

i. Let $ta \in \mathbf{t}\mathrm{PR}(T, <)$, $tb \in \mathbf{t}\mathrm{PR}(T, <)$, and $a \neq b$. We derive

$$
\begin{array}{ll}
& ta \in \mathbf{t}\mathrm{PR}(T, <) \\
\Rightarrow & \quad \{ \text{definition } \mathrm{PR}(T, <) \} \\
& (a, \ell(t\restriction a)) \in occ(T) \\
& \quad \wedge (\mathbf{A}\, c, k : (c, k) \in occ(T) \wedge (c, k) < (a, \ell(t\restriction a)) : \ell(t\restriction c) > k\,) \\
\Rightarrow & \quad \{ a \neq b \} \\
& (a, \ell(tb\restriction a)) \in occ(T) \\
& \quad \wedge (\mathbf{A}\, c, k : (c, k) \in occ(T) \wedge (c, k) < (a, \ell(tb\restriction a)) : \ell(tb\restriction c) > k\,) \\
\Rightarrow & \quad \{ tb \in \mathbf{t}\mathrm{PR}(T, <), \text{ definition } \mathrm{PR}(T, <) \} \\
& tba \in \mathbf{t}\mathrm{PR}(T, <)
\end{array}
$$

Analogously, one can derive $tab \in \mathbf{t}\mathrm{PR}(T, <)$.

ii. Let $tab \in \mathbf{t}\mathrm{PR}(T, <)$ and $tba \in \mathbf{t}\mathrm{PR}(T, <)$. We prove by induction on the length of trace u

$$(\mathbf{A}\, u : u \in \mathbf{a}T^* : tabu \in \mathbf{t}\mathrm{PR}(T, <) \;\equiv\; tbau \in \mathbf{t}\mathrm{PR}(T, <)\,)$$

base $\ell(u) = 0$
We have $u = \varepsilon$ and $tab\varepsilon \in \mathbf{t}\mathrm{PR}(T, <) \;\equiv\; tba\varepsilon \in \mathbf{t}\mathrm{PR}(T, <)$.

step $\ell(u) > 0$
Let $u = vc$. We derive

$$
\begin{array}{ll}
& tabvc \in \mathbf{t}\mathrm{PR}(T,<) \\
= & \quad \{ \text{definition } \mathrm{PR}(T,<) \} \\
& tabv \in \mathbf{t}\mathrm{PR}(T,<) \wedge (c, \ell(tabv\restriction c)) \in occ(T) \\
& \quad \wedge (\mathbf{A}\, d,k : (d,k) \in occ(T) \wedge (d,k) < (c,\ell(tabv\restriction c)) : \ell(tabv\restriction d) > k\,) \\
= & \quad \{ \ell(v) < \ell(u), \text{ property } \ell \} \\
& tbav \in \mathbf{t}\mathrm{PR}(T,<) \wedge (c, \ell(tbav\restriction c)) \in occ(T) \\
& \quad \wedge (\mathbf{A}\, d,k : (d,k) \in occ(T) \wedge (d,k) < (c,\ell(tbav\restriction c)) : \ell(tbav\restriction d) > k\,) \\
= & \quad \{ \text{definition } \mathrm{PR}(T,<) \} \\
& tbavc \in \mathbf{t}\mathrm{PR}(T,<)
\end{array}
$$

iii. Let $tac \in \mathbf{t}\mathrm{PR}(T,<)$, $tbc \in \mathbf{t}\mathrm{PR}(T,<)$, and $a \neq b$. We derive

$$
\begin{array}{ll}
& tac \in \mathbf{t}\mathrm{PR}(T,<) \wedge tbc \in \mathbf{t}\mathrm{PR}(T,<) \\
\Rightarrow & \quad \{ \text{definition } \mathrm{PR}(T,<) \} \\
& (c, \ell(ta\restriction c)) \in occ(T) \wedge (c, \ell(tb\restriction c)) \in occ(T) \\
& \quad \wedge (\mathbf{A}\, d,k : (d,k) \in occ(T) \wedge (d,k) < (c,\ell(ta\restriction c)) : \ell(ta\restriction d) > k\,) \\
& \quad \wedge (\mathbf{A}\, d,k : (d,k) \in occ(T) \wedge (d,k) < (c,\ell(tb\restriction c)) : \ell(tb\restriction d) > k\,) \\
\Rightarrow & \quad \{ a \neq b, \text{ property 2.5.0, } < \text{ is a partial order} \} \\
& (c, \ell(t\restriction c)) \in occ(T) \\
& \quad \wedge (\mathbf{A}\, d,k : (d,k) \in occ(T) \wedge (d,k) < (c,\ell(t\restriction c)) : \ell(ta\restriction d) > k \wedge \ell(tb\restriction d) > k\,) \\
\Rightarrow & \quad \{ a \neq b, t \in \mathbf{t}\mathrm{PR}(T,<), \text{ definition } \mathrm{PR}(T,<) \} \\
& tc \in \mathbf{t}\mathrm{PR}(T,<)
\end{array}
$$

(End of Proof)

Example 2.5.5

Let $T = \mathrm{PR}(\, a\,;\, b\,;\, c \mid b\,;\, c\,;\, a\,)$. Then partial order $<_T$ is characterized by

$$(b,0) <_T (c,0)$$

and $\mathrm{PR}(T,<_T) = \mathrm{PR}(\, a\,;\, b\,;\, c \mid b\,;\, a\,;\, c \mid b\,;\, c\,;\, a\,)$. Define partial order $<$ on $occ(T)$ by

$$
\begin{array}{rcl}
(b,0) & < & (a,0) \\
(a,0) & < & (c,0)
\end{array}
$$

Then $<$ respects $<_T$, and we have $\mathrm{PR}(T,<) = \mathrm{PR}(\, b\,;\, a\,;\, c\,)$. Notice that neither $\mathrm{PR}(T,<) \subseteq T$ nor $T \subseteq \mathrm{PR}(T,<)$ holds.

(End of Example)

The above example shows that if partial order $<$ respects $<_T$ we need not have that $\mathrm{PR}(T,<) \subseteq T$. From theorem 2.5.3 it follows that in case $T = \mathrm{PR}(T,<_T)$ we have that $\mathrm{PR}(T,<) \subseteq T$ for all partial orders $<$ respecting $<_T$. Therefore, we concentrate on the class of processes T satisfying $T = \mathrm{PR}(T,<_T)$. Tom Verhoeff ([Ve86]) showed

Theorem 2.5.6

Let T be a process. We have

$$T = \mathrm{PR}(T,<_T) \;\equiv\; T \text{ is cubic}$$

Proof

If $T = \mathrm{PR}(T,<_T)$ then by theorem 2.5.4 T is cubic.

Assume T is cubic. By theorem 2.5.3 we have that $T \subseteq \mathrm{PR}(T,<_T)$. By theorem 2.5.4 we have that $\mathrm{PR}(T,<_T)$ is cubic. We prove by induction on the length of trace t that

$$(\mathbf{A}\, t : t \in \mathbf{t}\mathrm{PR}(T,<_T) : t \in \mathbf{t}T\,)$$

base $\ell(t) = 0 \quad t = \varepsilon \in \mathbf{t}T$

step $\ell(t) > 0$

Assume

$$(\mathbf{A}\, u : u \in \mathbf{t}\mathrm{PR}(T,<_T) \wedge \ell(u) < \ell(t) : u \in \mathbf{t}T\,)$$

Let $t = ua$. Since $t \in \mathbf{t}\mathrm{PR}(T,<_T)$ we have

$$\begin{array}{l} u \in \mathbf{t}T \wedge (a, \ell(u\restriction a)) \in occ(T) \\ \quad \wedge\; (\mathbf{A}\, b,j : (b,j) \in occ(T) \wedge (b,j) <_T (a,\ell(t\restriction a)) : \ell(u\restriction b) > j\,) \end{array}$$

Since $(a,\ell(u\restriction a)) \in occ(T)$ choose s such that

$$\begin{array}{l} sa \in \mathbf{t}T \wedge \ell(s\restriction a) = \ell(u\restriction a) \\ \quad \wedge\; (\mathbf{A}\, r : ra \in \mathbf{t}T \wedge \ell(r\restriction a) = \ell(u\restriction a) : \ell(r) \geq \ell(s)\,) \end{array}$$

We prove that

$$(\mathbf{A}\, v,b : vb \leq s : (b,\ell(v\restriction b)) <_T (a,\ell(u\restriction a))\,)$$

Let $vb \leq s$. Assume $\neg(\,(b,\ell(v\restriction b)) <_T (a,\ell(u\restriction a))\,)$ or, equivalently, $(b,\ell(v\restriction b)) \notin pre(a,\ell(u\restriction a),T)$. Choose w such that

$$wa \in \mathbf{t}T \wedge \ell(w\restriction a) = \ell(u\restriction a) \wedge \ell(w\restriction b) \leq \ell(v\restriction b)$$

We derive

$$
\begin{array}{ll}
 & sa \in \mathbf{t}T \wedge wa \in \mathbf{t}T \\
\Rightarrow & \quad \{ T \text{ is cubic, theorem 2.4.6} \} \\
 & sa \uparrow \#(wa) \in \mathbf{t}T \\
= & \quad \{ \text{property 2.4.5, } \ell(w\restriction a) = \ell(u\restriction a) = \ell(s\restriction a) \} \\
 & (s \uparrow \#w)a \in \mathbf{t}T
\end{array}
$$

Furthermore, we have

$$\ell((s \uparrow \#w)\restriction a) = \ell(s\restriction a) = \ell(u\restriction a)$$

and

$$\ell(w\restriction b) \leq \ell(v\restriction b) < \ell(vb\restriction b) \leq \ell(s\restriction b)$$

The above implies that $\ell(s \uparrow \#w) < \ell(s)$ which contradicts the assumption made about s. Therefore, we have $(b, \ell(v\restriction b)) <_T (a, \ell(u\restriction a))$ and

$$(\mathbf{A}\, v, b : vb \leq s : \ell(u\restriction b) > \ell(v\restriction b))$$

This implies that $\#s \subseteq \#u$ and, since $\ell(s\restriction a) = \ell(u\restriction a)$,

$$ua = u(sa \setminus \#u) \in \mathbf{t}T$$

(End of Proof)

We now focus on the orderings of events of cubic processes that can be expressed by so-called sequence functions. Let T be a cubic process. Let $\sigma \in occ(T) \rightarrow \mathcal{N}$. Define binary relation $<_\sigma$ on $occ(T)$ by

$$(a,i) <_\sigma (b,j) \;\equiv\; \sigma(a,i) < \sigma(b,j) \qquad\qquad (a,i) \in occ(T), (b,j) \in occ(T)$$

Then $\langle occ(T), <_\sigma \rangle$ is a partially ordered set.
Function σ is called a *sequence function* for process T if

$$(\mathbf{A}\, a, b, i, j : (a,i) \in occ(T) \wedge (b,j) \in occ(T) \wedge (a,i) <_T (b,j) : (a,i) <_\sigma (b,j))$$

i.e. partial order $<_\sigma$ respects partial order $<_T$ ([Ee]). For occurrence $(a,i) \in occ(T)$ $\sigma(a,i)$ may be interpreted as the moment in time at which (a,i) takes place. Then function σ describes a possible synchronous (clocked) behaviour of the mechanism corresponding to process T.

Theorem 2.5.7

Let T be a cubic process. Let $\sigma \in occ(T) \to \mathcal{N}$ be defined by

$$\sigma(a,i) = |pre(a,i,T)| \qquad (a,i) \in occ(T)$$

Then σ is a sequence function for T.

(End of Theorem)

Example 2.5.8

Let $T = \mathrm{PR}(\, a\,,\, b\,;\, c\,)$. Then σ_0 defined by

$$\begin{aligned} \sigma_0(a,0) &= 0 \\ \sigma_0(b,0) &= 0 \\ \sigma_0(c,0) &= 2 \end{aligned}$$

and σ_1 defined by

$$\begin{aligned} \sigma_1(a,0) &= 0 \\ \sigma_1(b,0) &= 1 \\ \sigma_1(c,0) &= 2 \end{aligned}$$

are sequence functions for T. Notice that

$$\sigma_0(d,i) = |pre(d,i,T)| \qquad (d,i) \in occ(T)$$

(End of Example)

From the following theorem it follows that if σ is a sequence function for T and $f \in \mathcal{N} \to \mathcal{N}$ is an increasing function then $f \circ \sigma$ is a sequence function for T as well.

Theorem 2.5.9

Let T be a cubic process. Let σ be a sequence function for T. If $\rho \in occ(T) \to \mathcal{N}$ satisfies

$$\begin{aligned} (\mathbf{A}\, a,b,i,j : (a,i) \in occ(T) \wedge (b,j) \in occ(T) \wedge (a,i) <_T (b,j) \\ : \sigma(b,j) - \sigma(a,i) \leq \rho(b,j) - \rho(a,i)\,) \end{aligned}$$

then ρ is a sequence function for T.

(End of Theorem)

In order to give a different characterization of sequence functions we need two lemmas.

Lemma 2.5.10

Let T be a cubic process. Let σ be a sequence function for T. Then

$$(\mathbf{A}\, t, a, b : tab \in \mathbf{t}T \wedge \sigma(a, \ell(t\restriction a)) \geq \sigma(b, \ell(t\restriction b)) : tba \in \mathbf{t}T\,)$$

Proof

Let $tab \in \mathbf{t}T$ and $\sigma(a, \ell(t\restriction a)) \geq \sigma(b, \ell(t\restriction b))$. The case $a = b$ being trivial we assume $a \neq b$. We derive

$$\begin{array}{ll} & \sigma(a, \ell(t\restriction a)) \geq \sigma(b, \ell(t\restriction b)) \\ = & \quad \{\text{calculus}\} \\ & \neg(\,\sigma(a, \ell(t\restriction a)) < \sigma(b, \ell(t\restriction b))\,) \\ \Rightarrow & \quad \{\sigma \text{ is a sequence function for } T\} \\ & \neg(\,(a, \ell(t\restriction a)) <_T (b, \ell(t\restriction b))\,) \\ = & \quad \{\text{definition } <_T\} \\ & (a, \ell(t\restriction a)) \notin pre(b, \ell(t\restriction b), T) \\ = & \quad \{\text{definition } pre\} \\ & (\mathbf{E}\, s : sb \in \mathbf{t}T \wedge \ell(s\restriction b) = \ell(t\restriction b) : \ell(s\restriction a) \leq \ell(t\restriction a)\,) \end{array}$$

Choose trace s such that $sb \in \mathbf{t}T$, $\ell(s\restriction b) = \ell(t\restriction b)$, and $\ell(s\restriction a) \leq \ell(t\restriction a)$. Then $\#(ab) \cap (\#s - \#t) = \varnothing$. We derive

$$\begin{array}{ll} & sb(\,tab \setminus \#(sb)\,) \\ = & \quad \{\text{property 2.3.1}\} \\ & sb(\,(\,tab \setminus \#s\,) \setminus \#b\,) \\ = & \quad \{\text{property 2.3.1}\} \\ & sb(\,(t \setminus \#s)(ab \setminus (\#s - \#t)) \setminus \#b\,) \\ = & \quad \{\#(ab) \cap (\#s - \#t) = \varnothing, \text{ property 2.3.1}\} \\ & sb(\,(t \setminus \#s)ab \setminus \#b\,) \\ = & \quad \{\#b \cap (\#t - \#s) = \varnothing, \text{ property 2.3.1}, a \neq b\} \\ & sb(t \setminus \#s)a \end{array}$$

Since T is conservative we conclude that $sb(t \setminus \#s)a \in \mathbf{t}T$. Furthermore,

$$
\begin{array}{ll}
 & t(\,sb(t \setminus \#s)a \setminus \#t\,) \\
= & \quad \{\,\text{property 2.3.1}\,\} \\
 & t(\,sb \setminus \#t\,)(\,(t \setminus \#s)a \setminus (\#t - \#(sb))\,) \\
= & \quad \{\,\text{property 2.3.1, } \ell(s\restriction b) = \ell(t\restriction b)\,\} \\
 & t(s \setminus \#t)b(\,(t \setminus \#s)a \setminus \#(t \setminus \#s)\,) \\
= & \quad \{\,\text{property 2.3.1}\,\} \\
 & t(s \setminus \#t)ba
\end{array}
$$

Since T is conservative we conclude that $t(s \setminus \#t)ba \in \mathbf{t}T$. Since $\ell(s\restriction a) \leq \ell(t\restriction a)$ we have $\#(s \setminus \#t) \cap \#a = \varnothing$. From this, $t(s \setminus \#t)b \in \mathbf{t}T$, and $tab \in \mathbf{t}T$ we infer, since T is cubic, that $tb \in \mathbf{t}T$. From $ta \in \mathbf{t}T$, $tb \in \mathbf{t}T$, and $a \neq b$ it now follows that $tba \in \mathbf{t}T$.

(End of Proof)

Lemma 2.5.11

Let T be a cubic process. Let $\sigma \in occ(T) \rightarrow \mathcal{N}$ be such that

(i) $(\mathbf{A}\, a, i : (a,i) \in occ(T) : (\mathbf{A}\, j : 0 \leq j < i : \sigma(a,j) < \sigma(a,i))\,)$

and

(ii) $(\mathbf{A}\, t, a, b : tab \in \mathbf{t}T \wedge tba \notin \mathbf{t}T : \sigma(a, \ell(t\restriction a)) < \sigma(b, \ell(t\restriction b))\,)$

Then for all t, $t \in \mathbf{t}T$, we have

$$
\begin{array}{l}
(\mathbf{E}\, s : s \in \mathbf{t}T : \#t = \#s \wedge \\
\qquad\qquad (\mathbf{A}\, u, v, c, d : ucvd \leq s : \sigma(c, \ell(u\restriction c)) \leq \sigma(d, \ell(ucv\restriction d))\,)\,)
\end{array}
$$

Proof

Let $t \in \mathbf{t}T$. We construct a trace $s \in \mathbf{t}T$ such that $\#t = \#s$ and

$$(\mathbf{A}\, u, v, c, d : ucvd \leq s : \sigma(c, \ell(u\restriction c)) \leq \sigma(d, \ell(ucv\restriction d))\,)$$

or, equivalently,

$$(\mathbf{A}\, u, c, d : ucd \leq s : \sigma(c, \ell(u\restriction c)) \leq \sigma(d, \ell(uc\restriction d))\,)$$

Consider the following algorithm.

$$
\begin{array}{l}
s, k := t, (\mathbf{N}\, u, v, c, d : ucvd \leq t : \sigma(c, \ell(t\restriction c)) > \sigma(d, \ell(ucv\restriction d)))\\
\{\ \text{invariant: } s \in \mathbf{t}T \wedge \#s = \#t\\
\qquad\qquad \wedge\ k = (\mathbf{N}\, u, v, c, d : ucvd \leq s : \sigma(c, \ell(u\restriction c)) > \sigma(d, \ell(ucv\restriction d)))\\
\quad \text{, variant function: } k\ \}\\
; \mathbf{do}\ k \neq 0 \rightarrow \{k > 0\}\\
\qquad \text{let } s = ucdv \text{ such that } \sigma(c, \ell(u\restriction c)) > \sigma(d, \ell(uc\restriction d))\\
\qquad \{\ \text{(i) implies } c \neq d, \text{ (ii) implies } udc \in \mathbf{t}T,\\
\qquad\ \ T \text{ is conservative and therefore } udcv \in \mathbf{t}T\}\\
\qquad ; s, k := udcv, k - 1\\
\quad \mathbf{od}\\
\{\ k = 0, \text{ hence } (\mathbf{A}\, u, c, d : ucd \leq s : \sigma(c, \ell(u\restriction c)) \leq \sigma(d, \ell(uc\restriction d)))\}
\end{array}
$$

(End of Proof)

Theorem 2.5.12 ([Ee])

Let T be a cubic process. Let $\sigma \in occ(T) \rightarrow \mathcal{N}$. Function σ is a sequence function for T if and only if

(i) $(\mathbf{A}\, a, i : (a, i) \in occ(T) : (\mathbf{A}\, j : 0 \leq j < i : \sigma(a, j) < \sigma(a, i)))$

and

(ii) $(\mathbf{A}\, t, a, b : tab \in \mathbf{t}T \wedge tba \notin \mathbf{t}T : \sigma(a, \ell(t\restriction a)) < \sigma(b, \ell(t\restriction b)))$

Proof

0 Assume σ is a sequence function for T. Let $(a, i) \in occ(T)$ and $0 \leq j < i$. Then $(a, j) \in pre(a, i, T)$ and

$$
\begin{array}{ll}
& (a, j) \in pre(a, i, T)\\
= & \qquad \{\text{definition } <_T\}\\
& (a, j) <_T (a, i)\\
\Rightarrow & \qquad \{\sigma \text{ is a sequence function for } T\}\\
& \sigma(a, j) < \sigma(a, i)
\end{array}
$$

From lemma 2.5.10 it follows that (ii) holds.

1 Assume σ satisfies (i) and (ii). Let $(b, j) \in occ(T)$ and $(a, i) \in pre(b, j, T)$. If

$a = b$, then $i < j$ and, by (i), $\sigma(a,i) < \sigma(b,j)$. Hence, we assume $a \neq b$. Choose trace t such that $tb \in \mathbf{t}T$ and $\ell(t\restriction b) = j$. We now have $\ell(t\restriction a) > i$. Using lemma 2.5.11 we choose $s \in \mathbf{t}T$ such that $\#s = \#tb$ and

$$(\mathbf{A}\, u,v,c,d : ucvd \leq s : \sigma(c,\ell(u\restriction c)) \leq \sigma(d,\ell(ucv\restriction d)))$$

Let $ub \leq s$ and $\ell(u\restriction b) = j$. We now have $\ell(u\restriction a) > i$. Let $u = vaw$ be such that $\ell(v\restriction a) = i$. Then $\sigma(a,\ell(v\restriction a)) \leq \sigma(b,\ell(vaw\restriction b))$.
Assume that equality holds. Consider the following algorithm.

$y{:=}\, w$
{ invariant : $vayb \in \mathbf{t}T \wedge vay \leq s \wedge \sigma(a,\ell(v\restriction a)) = \sigma(b,\ell(vay\restriction b))$,
variant function : $\ell(y)$ }
; **do** $y \neq \varepsilon \rightarrow$ let $y = zc$
{ $\sigma(c,\ell(vaz\restriction c)) = \sigma(b,\ell(vay\restriction b))$,
(i) implies $c \neq b$, (ii) implies $vazb \in \mathbf{t}T$ }
; $y{:=}\, z$
od
{ $vab \in \mathbf{t}T \wedge \sigma(a,\ell(v\restriction a)) = \sigma(b,\ell(va\restriction b))$, (ii) implies $vba \in \mathbf{t}T$}

Thus $vba \in \mathbf{t}T$. This implies that $\ell(v\restriction a) > i$ which contradicts $\ell(v\restriction a) = i$.
Hence $\sigma(a,i) = \sigma(a,\ell(v\restriction a)) < \sigma(b,\ell(vaw\restriction b)) = \sigma(b,j)$.

(End of Proof)

Theorem 2.5.13

Let X be a set of cubic processes. Let for all $T \in X$ σ_T be a sequence function for T. If

(0) $(\mathbf{A}\, T,U : T \in X \wedge U \in X$
$: (\mathbf{A}\, a,i : (a,i) \in occ(T) \cap occ(U) \cap occ(\mathbf{W}(X)) : \sigma_T(a,i) = \sigma_U(a,i)))$

then $\sigma \in occ(\mathbf{W}(X)) \rightarrow \mathcal{N}$ defined by

(1) $\sigma(a,i) = \sigma_T(a,i)$ $\qquad T \in X, (a,i) \in occ(\mathbf{W}(X)) \cap occ(T)$

is a sequence function for $\mathbf{W}(X)$.

Proof

Assume (0) holds. Let σ be defined by (1). Function σ is well defined due to condition (0). Clearly, function σ satisfies condition (i) of theorem 2.5.12. We will show that

condition (ii) is also satisfied.
Let $tab \in \mathbf{tW}(X)$ and $tba \notin \mathbf{tW}(X)$. Observe that $a \neq b$. We have

$$
\begin{array}{l}
tab \in \mathbf{tW}(X) \wedge tba \notin \mathbf{tW}(X) \\
\equiv\ (\mathbf{A}\, T : T \in X \wedge a \in \mathbf{a}T \wedge b \notin \mathbf{a}T : (t\restriction \mathbf{a}T)a \in \mathbf{t}T) \\
\quad \wedge\ (\mathbf{A}\, T : T \in X \wedge a \in \mathbf{a}T \wedge b \in \mathbf{a}T : (t\restriction \mathbf{a}T)ab \in \mathbf{t}T) \\
\quad \wedge\ (\mathbf{A}\, T : T \in X \wedge a \notin \mathbf{a}T \wedge b \in \mathbf{a}T : (t\restriction \mathbf{a}T)b \in \mathbf{t}T) \\
\quad \wedge\ (\mathbf{E}\, T : T \in X \wedge a \in \mathbf{a}T \wedge b \in \mathbf{a}T : (t\restriction \mathbf{a}T)ba \notin \mathbf{t}T)
\end{array}
$$

Choose $T \in X$ such that $a \in \mathbf{a}T$, $b \in \mathbf{a}T$, $(t\restriction \mathbf{a}T)ab \in \mathbf{t}T$, and $(t\restriction \mathbf{a}T)ba \notin \mathbf{t}T$. We derive

$$
\begin{array}{ll}
 & (t\restriction \mathbf{a}T)ab \in \mathbf{t}T \wedge (t\restriction \mathbf{a}T)ba \notin \mathbf{t}T \\
\Rightarrow & \quad \{\, \sigma_T \text{ is a sequence function for } T, \text{ theorem 2.5.12}, a \neq b \,\} \\
 & \sigma_T(a, \ell(t\restriction \mathbf{a}T\restriction a)) < \sigma_T(b, \ell(t\restriction \mathbf{a}T\restriction b)) \\
= & \quad \{\, a \in \mathbf{a}T,\ b \in \mathbf{a}T, \text{ property projection} \,\} \\
 & \sigma_T(a, \ell(t\restriction a)) < \sigma_T(b, \ell(t\restriction b)) \\
= & \quad \{\, \text{definition } \sigma,\ (a, \ell(t\restriction a)),\ (b, \ell(t\restriction b)) \in occ(\mathbf{W}(X)) \cap occ(T) \,\} \\
 & \sigma(a, \ell(t\restriction a)) < \sigma(b, \ell(t\restriction b))
\end{array}
$$

(End of Proof)

Theorem 2.5.14

Let X be a set of cubic processes. Let for all T, $T \in X$, σ_T be a sequence function for T. If

(0) $(\mathbf{A}\, T, U : T \in X \wedge U \in X : occ(T\restriction \mathbf{a}U) = occ(U\restriction \mathbf{a}T))$

and

(1) $(\mathbf{A}\, T, U : T \in X \wedge U \in X$
$\quad : (\mathbf{A}\, a, i : (a, i) \in occ(T) \cap occ(U) : \sigma_T(a, i) = \sigma_U(a, i)))$

then $lockfree(X)$.

Proof

Assume (0) and (1). Let $t \in \mathbf{tW}(X)$ be such that $(\mathbf{E}\, U : U \in X : suc(t\restriction \mathbf{a}U, U) \neq \varnothing)$. Choose $T \in X$ and $a \in suc(t\restriction \mathbf{a}T, T)$ such that

$$\sigma_T(a, \ell(t\restriction a)) = (\mathbf{MIN}\, U, b : U \in X \wedge b \in suc(t\restriction \mathbf{a}U, U) : \sigma_U(b, \ell(t\restriction b)))$$

We show that $ta \in \mathbf{tW}(X)$. Define $Y = \{U \mid U \in X \wedge a \in \mathbf{a}U\}$. Let $S \in Y$. Since $(a, \ell(t\restriction a)) \in occ(S)$ and S is cubic, there exists a trace x such that

$$(t\restriction \mathbf{a}S)xa \in \mathbf{t}S \wedge x\restriction a = \varepsilon$$
$$\ell(x) = (\mathbf{MIN}\, y : (t\restriction \mathbf{a}S)ya \in \mathbf{t}S \wedge y\restriction a = \varepsilon : \ell(y))$$

On basis of the algorithm given in the proof of lemma 2.5.11 we assume

$$(\mathbf{A}\, u, v, b, c : ubvc \leq xa : \sigma_S(b, \ell((t\restriction \mathbf{a}S)u\restriction b)) \leq \sigma_S(c, \ell((t\restriction \mathbf{a}S)ubv\restriction c)))$$

Assume $x = dz$. From the algorithm given in the proof of theorem 2.5.12 and the definition of x it follows that

$$\sigma_S(d, \ell(t\restriction d)) = \sigma_S(d, \ell((t\restriction \mathbf{a}S)\restriction d)) < \sigma_S(a, \ell((t\restriction \mathbf{a}S)dz\restriction a)) = \sigma_S(a, \ell(t\restriction a))$$

This, however, contradicts

$$\sigma_S(a, \ell(t\restriction a)) = \sigma_T(a, \ell(t\restriction a)) \leq \sigma_S(d, \ell(t\restriction d))$$

We conclude that $x = \varepsilon$ and, hence, $a \in suc(t\restriction \mathbf{a}S, S)$. We now have

$$(\mathbf{A}\, U : U \in X \wedge a \in \mathbf{a}U : a \in suc(t\restriction \mathbf{a}U, U))$$

and, therefore, $ta \in \mathbf{tW}(X)$.

(End of Proof)

Theorem 2.5.15

Let T and U be cubic processes such that $U \subseteq T$. Let $\sigma \in occ(U) \rightarrow \mathcal{N}$ be a sequence function for U. Define $\tau \in occ(T) \rightarrow \mathcal{N}$ as follows

$$\tau(a,i) = \sigma(a,i) \qquad \text{for } (a,i) \in occ(U)$$
$$\tau(a,i) = (\mathbf{MAX}\, c,k : (c,k) \in occ(U) \wedge (c,k) <_T (a,i) : \sigma(c,k) + 1)\, \mathbf{max}\, 0 + |pre(a,i,T) \cap occ(T) \setminus occ(U)| \qquad \text{for } (a,i) \in occ(T) \setminus occ(U)$$

Then τ is a sequence function for T.

Proof

Let $(a,i) \in occ(T)$, $(b,j) \in occ(T)$, and $(a,i) <_T (b,j)$. Assume $(a,i) \in occ(T) \setminus occ(U)$ and $(b,j) \in occ(U)$. Let $ub \in \mathbf{t}U$ be such that $\ell(u\restriction b) = j$. Since $U \subseteq T$ and $(a,i) <_T (b,j)$, we now have $\ell(u\restriction a) > i$ and, hence, $(a,i) \in occ(U)$ which contradicts $(a,i) \in occ(T) \setminus occ(U)$. Therefore, we only have to distinguish the following cases

i. $(a,i) \in occ(U) \wedge (b,j) \in occ(U)$

We derive

$$
\begin{array}{ll}
& (a,i) <_T (b,j) \\
\Rightarrow & \quad \{\, U \subseteq T, \text{ theorem } 2.5.2 \,\} \\
& (a,i) <_U (b,j) \\
\Rightarrow & \quad \{\, \sigma \text{ is a sequence function for } U \,\} \\
& \sigma(a,i) < \sigma(b,j) \\
= & \quad \{\, \text{definition } \tau \,\} \\
& \tau(a,i) < \tau(b,j)
\end{array}
$$

ii. $(a,i) \in occ(T) \setminus occ(U) \wedge (b,j) \in occ(T) \setminus occ(U)$
We derive

$$
\begin{array}{ll}
& (a,i) <_T (b,j) \\
\Rightarrow & \quad \{\, \text{definition } <_T, \text{ property } 2.5.0 \,\} \\
& pre(a,i,T) \cup \{(a,i)\} \subseteq pre(b,j,T) \\
\Rightarrow & \quad \{\, \text{set calculus, } (a,i) \in occ(T) \setminus occ(U) \,\} \\
& pre(a,i,T) \cap occ(U) \subseteq pre(b,j,T) \cap occ(U) \\
& \quad \wedge\ (\, pre(a,i,T) \cap occ(T) \setminus occ(U) \,) \cup \{(a,i)\} \subseteq pre(b,j,T) \cap occ(T) \setminus occ(U) \\
\Rightarrow & \quad \{\, \text{definition } <_T, \text{ definition } \tau \,\} \\
& \tau(a,i) < \tau(b,j)
\end{array}
$$

iii. $(a,i) \in occ(U) \wedge (b,j) \in occ(T) \setminus occ(U)$
We derive

$$
\begin{array}{ll}
& \tau(a,i) \\
= & \quad \{\, \text{definition } \tau \,\} \\
& \sigma(a,i) \\
< & \quad \{\, \text{calculus} \,\} \\
& \sigma(a,i) + 1 \\
\leq & \quad \{\, \text{definition } \tau,\ (a,i) <_T (b,j),\ (a,i) \in occ(U) \,\} \\
& \tau(b,j)
\end{array}
$$

(End of Proof)

Theorem 2.5.16

Let T be a cubic process and let A be an alphabet. Let $\sigma \in occ(T) \rightarrow \mathcal{N}$ be sequence function for T. Define $\tau \in occ(T \restriction A) \rightarrow \mathcal{N}$ by

$$\tau(a,i) = \sigma(a,i) \qquad\qquad (a,i) \in occ(T \restriction A)$$

Then τ is a sequence function for $T\restriction A$.

Proof

Let $(a,i) \in occ(T\restriction A)$ and $(b,j) \in occ(T\restriction A)$. We derive

$$
\begin{array}{ll}
 & (a,i) <_{T\restriction A} (b,j) \\
= & \quad \{ \text{theorem 2.5.2} \} \\
 & (a,i) <_T (b,j) \\
\Rightarrow & \quad \{ \sigma \text{ is a sequence function for } T \} \\
 & \sigma(a,i) < \sigma(b,j) \\
= & \quad \{ \text{definition } \tau \} \\
 & \tau(a,i) < \tau(b,j)
\end{array}
$$

(End of Proof)

Theorem 2.5.17

Let T be a cubic process. Let A be an alphabet. Let $\sigma \in occ(T) \rightarrow \mathcal{N}$ be a sequence function for T. Let $\rho \in occ(T\restriction A) \rightarrow \mathcal{N}$. If

$$
\begin{array}{ll}
(0) & (\mathbf{A}\, a,b,i,j : (a,i) \in occ(T\restriction A) \wedge (b,j) \in occ(T\restriction A) \\
 & \qquad \wedge\ (a,i) <_{T\restriction A} (b,j) : \sigma(b,j) - \sigma(a,i) \leq \rho(b,j) - \rho(a,i)\,)
\end{array}
$$

and

$$
(1) \quad (\mathbf{A}\, a : (a,0) \in occ(T\restriction A) : \sigma(a,0) \leq \rho(a,0)\,)
$$

then there exists a function $\tau \in occ(T) \rightarrow \mathcal{N}$ that is a sequence function for T and satisfies

$$
(\mathbf{A}\, a,i : (a,i) \in occ(T\restriction A) : \tau(a,i) = \rho(a,i)\,)
$$

Proof

Let ρ satisfy (0) and (1). From theorems 2.5.9 and 2.5.16 it follows that ρ is a sequence function for $T\restriction A$. Furthermore, we have that

$$
(\mathbf{A}\, a,i : (a,i) \in occ(T\restriction A) : \sigma(a,i) \leq \rho(a,i)\,)
$$

Define for $(a,i) \in occ(T)$

$$
\begin{array}{l}
M(a,i) \\
= (\mathbf{MAX}\, c,k : (c,k) \in occ(T\restriction A) \wedge (c,k) <_T (a,i) : \rho(c,k) - \sigma(c,k)\,) \,\mathbf{max}\, 0
\end{array}
$$

Define $\tau \in occ(T) \to \mathcal{N}$ by

$$\tau(a,i) = \rho(a,i) \qquad (a,i) \in occ(T \restriction A)$$
$$\tau(a,i) = \sigma(a,i) + M(a,i) \qquad (a,i) \in occ(T) \setminus occ(T \restriction A)$$

Let $(a,i) \in occ(T)$, $(b,j) \in occ(T)$, and $(a,i) <_T (b,j)$. We distinguish four cases.

i. $(a,i) \in occ(T \restriction A) \wedge (b,j) \in occ(T \restriction A)$

We now have

$$\tau(a,i) = \rho(a,i) < \rho(b,j) = \tau(b,j)$$

ii. $(a,i) \in occ(T) \setminus occ(T \restriction A) \wedge (b,j) \in occ(T) \setminus occ(T \restriction A)$

We derive

$$\begin{array}{ll} & \tau(a,i) \\ = & \quad \{ \text{definition } \tau \} \\ & \sigma(a,i) + M(a,i) \\ < & \quad \{ (a,i) <_T (b,j), \sigma \text{ is a sequence function for } T, \text{property 2.5.0} \} \\ & \sigma(b,j) + M(b,j) \\ = & \quad \{ \text{definition } \tau \} \\ & \tau(b,j) \end{array}$$

iii. $(a,i) \in occ(T \restriction A) \wedge (b,j) \in occ(T) \setminus occ(T \restriction A)$

We derive

$$\begin{array}{ll} & \tau(b,j) \\ = & \quad \{ \text{definition } \tau \} \\ & \sigma(b,j) + M(b,j) \\ \geq & \quad \{ (a,i) <_T (b,j), (a,i) \in occ(T \restriction A), \sigma(a,i) \leq \rho(a,i) \} \\ & \sigma(b,j) + \rho(a,i) - \sigma(a,i) \\ > & \quad \{ (a,i) <_T (b,j), \sigma \text{ is a sequence function for } T \} \\ & \rho(a,i) \\ = & \quad \{ \text{definition } \tau \} \\ & \tau(a,i) \end{array}$$

iv. $(a,i) \in occ(T) \setminus occ(T \restriction A) \wedge (b,j) \in occ(T \restriction A)$

We derive

$$
\begin{array}{ll}
 & \tau(a,i) \\
= & \quad \{ \text{ definition } \tau \} \\
 & \sigma(a,i) + M(a,i) \\
\leq & \quad \{ (a,i) <_T (b,j),\ (b,j) \in occ(T\restriction A),\ (0) \} \\
 & \sigma(a,i) + \\
 & (\mathbf{MAX}\, c,k : (c,k) \in occ(T\restriction A) \wedge (c,k) <_T (a,i) : \rho(b,j) - \sigma(b,j)\,)\,\mathbf{max}\,0 \\
\leq & \quad \{ \rho(b,j) - \sigma(b,j) \geq 0 \} \\
 & \sigma(a,i) + \rho(b,j) - \sigma(b,j) \\
< & \quad \{ (a,i) <_T (b,j),\ \sigma \text{ is a sequence function for } T \} \\
 & \rho(b,j) \\
= & \quad \{ \text{ definition } \tau \} \\
 & \tau(b,j)
\end{array}
$$

(End of Proof)

Let X be a set of cubic processes. Let σ be a sequence function for $\mathbf{W}(X)$. Let $T \in X$. Define $\rho_T \in occ(\mathbf{W}(X)\restriction \mathbf{a}T) \rightarrow \mathcal{N}$ by

$$\rho_T(a,i) = \sigma(a,i) \qquad\qquad (a,i) \in occ(\mathbf{W}(X)\restriction \mathbf{a}T)$$

On account of theorem 2.5.16 ρ_T is a sequence function for $\mathbf{W}(X)\restriction \mathbf{a}T$.
Since $\mathbf{W}(X)\restriction \mathbf{a}T \subseteq T$, there exists, on account of theorem 2.5.15, a sequence function σ_T for T such that

$$\sigma_T(a,i) = \rho_T(a,i) = \sigma(a,i) \qquad\qquad (a,i) \in occ(\mathbf{W}(X)\restriction \mathbf{a}T)$$

On the other hand, let for all T, $T \in X$, σ_T be a sequence function for T and let

$$
\begin{array}{l}
(\mathbf{A}\, T,U : T \in X \wedge U \in X \\
\qquad : (\mathbf{A}\, a,i : (a,i) \in occ(T) \cap occ(U) \cap occ(\mathbf{W}(X)) : \sigma_T(a,i) = \sigma_U(a,i)\,)\,)
\end{array}
$$

Then, by theorem 2.5.13 the function $\sigma \in occ(\mathbf{W}(X)) \rightarrow \mathcal{N}$ defined by

$$\sigma(a,i) = \sigma_T(a,i) \qquad\qquad T \in X, (a,i) \in occ(\mathbf{W}(X)) \cap occ(T)$$

is a sequence function for $\mathbf{W}(X)$.

Let S be a system such that $(\mathbf{A}\, T : T \in \mathbf{p}S : T$ is cubic).
A function $\sigma \in occ(\mathrm{PR}(S)) \rightarrow \mathcal{N}$ is called a *sequence function* for system S if there exists a sequence function τ for $\mathbf{W}(\mathbf{p}(S))$ such that

$$(\mathbf{A}\, a, i : (a,i) \in occ(\mathrm{PR}(S)) : \tau(a,i) = \sigma(a,i))$$

Observe that σ is a sequence function for $\mathrm{PR}(S)$.

Corollary 2.5.18

Let S be a system such that $(\mathbf{A}\, T : T \in \mathbf{p}S : T$ is cubic). Let $\sigma \in occ(\mathrm{PR}(S)) \rightarrow \mathcal{N}$ be a sequence function for S. Let $\rho \in occ(\mathrm{PR}(S)) \rightarrow \mathcal{N}$. If

$$(\mathbf{A}\, a, b, i, j : (a,i) \in occ(\mathrm{PR}(S)) \wedge (b,j) \in occ(\mathrm{PR}(S)) \\ \wedge\ (a,i) <_{\mathrm{PR}(S)} (b,j) : \sigma(b,j) - \sigma(a,i) \leq \rho(b,j) - \rho(a,i))$$

and

$$(\mathbf{A}\, a : (a,0) \in occ(\mathrm{PR}(S)) : \sigma(a,0) \leq \rho(a,0))$$

then ρ is a sequence function for S.

(End of Corollary)

In the following we consider components whose corresponding systems contain cubic processes only. This is, for instance, the case when all occurring commands are restricted commands. Let c be a component satisfying the above condition. A function $\sigma \in occ(\mathrm{PR}(c)) \rightarrow \mathcal{N}$ is called a *sequence function* for c if σ is a sequence function for $sys(c)$.
The following theorem will be applied in the proof of a theorem concerning sequence functions for simple recursive components.

Theorem 2.5.19

Let T be a cubic process and A an alphabet such that $\mathbf{a}T = A \cup p{\cdot}A$,

(0) $\quad (\mathbf{A}\, a, i : a \in A : (a,i) \in occ(T) \ \equiv\ (p{\cdot}a, i) \in occ(T))$

and

(1) $\quad (\mathbf{A}\, a, i : a \in A \wedge (a,i) \in occ(T) : (a,i) <_T (p{\cdot}a, i))$

Let $\sigma \in occ(T) \rightarrow \mathcal{N}$ be a sequence function for T satisfying

$$(*)\quad (\mathbf{A}\, a,b,i,j : a \in A \wedge b \in A \wedge (a,i) \in occ(T) \\ \wedge (b,j) \in occ(T) \wedge (a,i) <_T (b,j) \\ : \sigma(b,j) - \sigma(a,i) \leq \sigma(p{\cdot}b,j) - \sigma(p{\cdot}a,i))$$

Define for $(p{\cdot}a,i) \in occ(T)$

$$M(p{\cdot}a,i) = (\mathbf{MAX}\, c,k : c \in A \wedge (c,k) \in occ(T) \\ \wedge (c,k) <_T (p{\cdot}a,i) : \sigma(p{\cdot}c,k) - \sigma(c,k))$$

Then $\tau \in occ(T) \to \mathcal{N}$ defined by

$$\begin{aligned} \tau(a,i) &= \sigma(p{\cdot}a,i) && a \in A, (a,i) \in occ(T) \\ \tau(p{\cdot}a,i) &= \sigma(p{\cdot}a,i) + M(p{\cdot}a,i) && a \in A, (p{\cdot}a,i) \in occ(T) \end{aligned}$$

is a sequence function for T. If, moreover, T satisfies

$$(\mathbf{A}\, a,b,i,j : a,b \in A \wedge (a,i),(b,j) \in occ(T) \\ : (a,i) <_T (b,j) \Rightarrow (p{\cdot}a,i) <_T (p{\cdot}b,j))$$

then τ satisfies $(*)$.

Proof

Observe that τ is indeed a function mapping $occ(T)$ into $\mathcal{N}$. We will show that τ satisfies the condition in theorem 2.5.9. Let $a \in A$ and $b \in A$. We distinguish four cases.

i. $(a,i) \in occ(T) \wedge (b,j) \in occ(T) \wedge (a,i) <_T (b,j)$

We derive

$$\begin{aligned} & \tau(b,j) - \tau(a,i) \\ = & \quad \{ \text{definition } \tau \} \\ & \sigma(p{\cdot}b,j) - \sigma(p{\cdot}a,i) \\ \geq & \quad \{ (a,i) <_T (b,j), (*) \} \\ & \sigma(b,j) - \sigma(a,i) \end{aligned}$$

ii. $(p{\cdot}a,i) \in occ(T) \wedge (p{\cdot}b,j) \in occ(T) \wedge (p{\cdot}a,i) <_T (p{\cdot}b,j)$

We derive

$$\begin{aligned} & \tau(p{\cdot}b,j) - \tau(p{\cdot}a,i) \\ = & \quad \{ \text{definition } \tau \} \\ & \sigma(p{\cdot}b,j) - \sigma(p{\cdot}a,i) + M(p{\cdot}b,j) - M(p{\cdot}a,i) \\ \geq & \quad \{ (p{\cdot}a,i) <_T (p{\cdot}b,j), \text{definition } M, (1) \} \\ & \sigma(p{\cdot}b,j) - \sigma(p{\cdot}a,i) \end{aligned}$$

iii. $(p{\cdot}a, i) \in occ(T) \wedge (b, j) \in occ(T) \wedge (p{\cdot}a, i) <_T (b, j)$
We derive

$$\begin{array}{ll} & \tau(b,j) - \tau(p{\cdot}a,i) \\ = & \quad \{ \text{definition } \tau \} \\ & \sigma(p{\cdot}b,j) - \sigma(p{\cdot}a,i) - M(p{\cdot}a,i) \\ \geq & \quad \{ (p{\cdot}a,i) <_T (b,j), (*), \text{definition } M \} \\ & \sigma(p{\cdot}b,j) - \sigma(p{\cdot}a,i) - \sigma(p{\cdot}b,j) + \sigma(b,j) \\ = & \quad \{ \text{calculus} \} \\ & \sigma(b,j) - \sigma(p{\cdot}a,i) \end{array}$$

iv. $(a, i) \in occ(T) \wedge (p{\cdot}b, j) \in occ(T) \wedge (a, i) <_T (p{\cdot}b, j)$
We derive

$$\begin{array}{ll} & \tau(p{\cdot}b,j) - \tau(a,i) \\ = & \quad \{ \text{definition } \tau \} \\ & \sigma(p{\cdot}b,j) - \sigma(p{\cdot}a,i) + M(p{\cdot}b,j) \\ \geq & \quad \{ (a,i) <_T (p{\cdot}b,j) \} \\ & \sigma(p{\cdot}b,j) - \sigma(p{\cdot}a,i) + \sigma(p{\cdot}a,i) - \sigma(a,i) \\ = & \quad \{ \text{calculus} \} \\ & \sigma(p{\cdot}b,j) - \sigma(a,i) \end{array}$$

From the above we infer that τ is a sequence function for T.
Assume T satisfies

$$\begin{array}{l} (\mathbf{A}\, a,b,i,j : a,b \in A \wedge (a,i),(b,j) \in occ(T) \\ \qquad : (a,i) <_T (b,j) \;\Rightarrow\; (p{\cdot}a,i) <_T (p{\cdot}b,j)\,) \end{array}$$

Let $a \in A$, $b \in A$, $(a, i) \in occ(T)$, $(b, j) \in occ(T)$, and $(a, i) <_T (b, j)$. We derive

$$\begin{array}{ll} & \tau(p{\cdot}b,j) - \tau(p{\cdot}a,i) \\ \geq & \quad \{ \text{ii.}, (p{\cdot}a,i) <_T (p{\cdot}b,j) \} \\ & \sigma(p{\cdot}b,j) - \sigma(p{\cdot}a,i) \\ = & \quad \{ \text{definition } \tau \} \\ & \tau(b,j) - \tau(a,i) \end{array}$$

(End of Proof)

Theorem 2.5.20

Let component c be recursive component

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

where $\mathbf{a}\mathrm{PR}(S) = A \cup p{\cdot}A$ and $\mathrm{PR}(S)$ is a cubic process satisfying the conditions in theorem 2.5.19. If $\sigma \in occ(\mathrm{PR}(S)) \rightarrow \mathcal{N}$ is a sequence function for $\mathrm{PR}(S)$ satisfying condition $(*)$ in theorem 2.5.19. then $\tau \in occ(\mathrm{PR}(c)) \rightarrow \mathcal{N}$ defined by

$$\tau(a,i) = \sigma(a,i) \qquad (a,i) \in occ(\mathrm{PR}(c))$$

is a sequence function for c and $sys(c)$ is a lockfree system.

Proof

Let $\sigma \in occ(\mathrm{PR}(S)) \rightarrow \mathcal{N}$ be a sequence function for $\mathrm{PR}(S)$ satisfying condition $(*)$ in theorem 2.5.19. We have

$$sys(c) = \langle\, A, \{\, (p{\cdot})^l \mathrm{PR}(S) \mid l \geq 0 \,\} \,\rangle$$

We define functions σ_l, $l \geq 0 \wedge \sigma_l \in occ((p{\cdot})^l \mathrm{PR}(S)) \rightarrow \mathcal{N}$, inductively by

$$\begin{aligned}
&\sigma_0 = \sigma \\
&\sigma_{l+1}((p{\cdot})^{l+1}a, i) = \sigma_l((p{\cdot})^{l+1}a, i) \qquad \text{for } l \geq 0,\ a \in A,\ (a,i) \in occ(\mathrm{PR}(S)) \\
&\sigma_{l+1}((p{\cdot})^{l+2}a, i) = \sigma_l((p{\cdot})^{l+1}a, i) \\
&\qquad + (\mathbf{MAX}\, c,k : c \in A \wedge (c,k) \in occ(\mathrm{PR}(S)) \\
&\qquad\quad \wedge (c,k) <_{\mathrm{PR}(S)} (p{\cdot}a, i) : \sigma_l((p{\cdot})^{l+1}c, k) - \sigma_l((p{\cdot})^l c, k)\,) \\
&\qquad\qquad \text{for } l \geq 0,\ a \in A,\ (p{\cdot}a, i) \in occ(\mathrm{PR}(S))
\end{aligned}$$

Then σ_l is a sequence function for $(p{\cdot})^l \mathrm{PR}(S)$ for all $l \geq 0$ and

$$\begin{aligned}
(\mathbf{A}\, k,l,c,m : 0 \leq k < l \wedge (c,m) \in occ((p{\cdot})^k \mathrm{PR}(S)) \cap occ((p{\cdot})^l \mathrm{PR}(S)) \\
: \sigma_k(c,m) = \sigma_l(c,m)\,)\,)
\end{aligned}$$

Then $\rho \in occ(\mathbf{W}(\mathbf{p}sys(c))) \rightarrow \mathcal{N}$ defined by

$$\rho(a,i) = \sigma_k(a,i) \qquad k \geq 0, (a,i) \in occ(\mathbf{W}(\mathbf{p}sys(c))) \cap occ((p{\cdot})^k \mathrm{PR}(S))$$

is a sequence function for $\mathbf{W}(\mathbf{p}sys(c))$ such that

$$\tau(a,i) = \rho(a,i) \qquad (a,i) \in occ(\mathrm{PR}(c))$$

(End of Proof)

Application of the above theorem is illustrated in examples 2.5.21 and 2.5.22.

Let T be a cubic process and σ be a sequence function for T. We already mentioned that for $(a,i) \in occ(T)$ $\sigma(a,i)$ may be interpreted as the moment at which (a,i) is to take place. Likewise, one may interpret for $(a,i) \in occ(T)$, $(b,j) \in occ(T)$, and $(a,i) <_T (b,j)$ $\sigma(b,j) - \sigma(a,i)$ as the time elapsed between the i-th occurrence of a and the j-th occurrence of b. Based on this interpretation we give the following definition of constant response time.

Let S be a system such that $(\mathbf{A}\, T : T \in \mathbf{p}S : T \text{ is cubic})$. System S is said to have *constant response time* if there exists a sequence function σ for S such that

$$(\mathbf{E}\, n : n \geq 1 : (\mathbf{A}\, t, a, b : tab \in \mathbf{t}\mathrm{PR}(S) \wedge (a, \ell(t\restriction a)) <_{\mathrm{PR}(S)} (b, \ell(ta\restriction b)) \\ : \sigma(b, \ell(ta\restriction b)) - \sigma(a, \ell(t\restriction a)) \leq n\,))$$

i.e. there exists a restricted (clocked) behaviour of the system such that there exists a global upperbound for the time elapsed between any two consecutive external events. A component c is said to have *constant response time* if $sys(c)$ has constant response time. Observe that the constant in the above condition may be interpreted as a measure for the response time of the system.

Example 2.5.21

Consider the following recursive component

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

We have $\mathrm{PR}(c) = \mathrm{SEM}_1(a,b)$. Let $U = \mathrm{PR}((a\,;\, p{\cdot}a\,;\, b\,;\, p{\cdot}b)^*)$. Define for i, $i \geq 0$,

$$\begin{aligned}
\sigma(a,i) &= 4i \\
\sigma(p{\cdot}a,i) &= 4i+1 \\
\sigma(b,i) &= 4i+2 \\
\sigma(p{\cdot}b,i) &= 4i+3
\end{aligned}$$

We now have that σ is a sequence function for U satisfying condition $(*)$ from theorem 2.5.19. By theorem 2.5.20 we have that σ restricted to $occ(\mathrm{PR}(c))$ is a sequence function for component c. Furthermore, we have

$$(\mathbf{A}\, t, c, d : tcd \in \mathbf{t}\mathrm{PR}(c) \wedge (c, \ell(t\restriction c)) <_{\mathrm{PR}(c)} (d, \ell(tc\restriction d)) \\ : \sigma(d, \ell(tc\restriction d)) - \sigma(c, \ell(t\restriction c)) \leq 2\,)$$

Therefore, component c has constant response time.

(End of Example)

Example 2.5.22

Consider the following recursive component

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

We have $\mathrm{PR}(c) = \mathrm{SEM}_1(a, b)$. Let $U = \mathrm{PR}(a\,;\ b\,;\ (p{\cdot}a\,;\ a\,;\ b\,;\ p{\cdot}b)^*)$. Define for i, $i \geq 0$,

$$\begin{aligned} \sigma(a, i) &= (i+1)^2 - 1 \\ \sigma(b, i) &= (i+1)^2 \\ \sigma(p{\cdot}a, i) &= (i+2)^2 - 2 \\ \sigma(p{\cdot}b, i) &= (i+2)^2 + 1 \end{aligned}$$

Then σ is a sequence function for U satisfying condition $(*)$ from theorem 2.5.19 and, therefore, theorem 2.5.20 is applicable. Component c does not have constant response time.

(End of Example)

3 Communication of values and data independence

3.0 Introduction

In this and the following chapters we restrict ourselves to processes describing mechanisms that can interact with their environment by sending and receiving values or messages via channels. The transmission of a value or message via a channel is represented by a pair consisting of the channel name followed by the message. Such pairs are considered to be symbols.

In section 1.0 we presented an example of such a mechanism, viz. the variable capable of storing integer values. The process describing such a variable is given by

$$\mathrm{VAR} = \mathrm{PREF}(\langle \{a,b\} \times \mathcal{Z}\,,\, (\cup n : n \in \mathcal{Z} : \{\, \langle a,n\rangle t \mid t \in \{\langle b,n\rangle\}^* \,\})^* \rangle)$$

Abstraction from the actual messages that are sent, i.e. abstraction from the values that the variable can store, results in process $\mathrm{VAR}_{\mathrm{com}} = \mathrm{PR}((a\,;\, b^*)^*)$. Process $\mathrm{VAR}_{\mathrm{com}}$ describes the so-called *communication behaviour* of the variable. As process VAR is identified with the variable, process $\mathrm{VAR}_{\mathrm{com}}$ will be identified with the communication behaviour of the variable. Let t be the current trace of the variable ($t \in \mathrm{tVAR}$). Let trace u describe the communication pattern corresponding to t ($u \in \mathrm{tVAR}_{\mathrm{com}}$). Observe that $ua \in \mathrm{tVAR}_{\mathrm{com}}$ implies the existence of $m \in \mathcal{Z}$ such that $t\langle a,m\rangle \in \mathrm{tVAR}$ (according to process VAR communication of an actual message via channel a may occur). The same observation can be made about channel b. In other words, the communication pattern does not depend on the messages (data) that are sent and received. Process VAR is called *data independent.*

An example of a process that is not data independent is FILTER, defined by

$$\begin{aligned}\mathrm{FILTER} = \mathrm{PREF}(&\langle \{a,b\} \times \mathcal{Z} \\ &,(\,(\cup n : n \geq 0 : \{\langle a,n\rangle\langle b,n\rangle\}) \cup (\cup n : n < 0 : \{\langle a,n\rangle\})\,)^* \rangle)\end{aligned}$$

Process FILTER describes a mechanism that filters the negative numbers received via channel a: it only transmits via channel b the nonnegative numbers received.

In this chapter we give a formal definition of data independence. It is shown that data independence can be expressed in terms of transparence. Furthermore, a number of properties of data independent processes are given.

3.1 Communication of values

Transmission of value or message m via channel c is modelled by the pair $\langle c, m\rangle$. We assume the existence of a set Γ of names and a set M (capital μ) of *values* or *messages*. Elements of Γ are called *channels*. Subsets of Γ are called *channel sets*.

Some channels are used to transmit only one kind of message, a so-called *signal*. These channels are used for synchronization only. In order to model signal transmission we introduce the so-called empty message $\surd$ (read "tick"). We assume that $\surd \notin \mathrm{M}$. We define $\mathrm{M}_{\surd} = \mathrm{M} \cup \{\surd\}$. From here on we take $\Omega = \Gamma \times \mathrm{M}_{\surd}$.

We observe that every process T from the previous chapters can be identified with a process that is obtained from T by substituting every symbol in T by a pair consisting of that symbol followed by $\surd$. Due to this identification we often denote pairs containing $\surd$ by their channel names only.

Abstraction from the messages being sent is represented by the function $\gamma \in \Omega \to \Gamma$ defined by

$$\gamma(\langle c, m\rangle) = c$$

Function γ is extended to traces by defining

$$\begin{aligned} \gamma(\varepsilon) &= \varepsilon \\ \gamma(t\langle c, m\rangle) &= \gamma(t)c \end{aligned}$$

Likewise, function γ is extended to trace sets, trace structures, and processes. Function γ may be interpreted as the projection on the channel names.

Let s and t be traces, X and Y trace sets, and T and U trace structures such that $\mathrm{a}T = \mathrm{a}U$.

Property 3.1.0

0 $\gamma(s) \in \Gamma^*$

1 $\gamma(st) = \gamma(s)\gamma(t)$

2 $\ell(\gamma(s)) = \ell(s)$

3 $s \leq t \Rightarrow \gamma(s) \leq \gamma(t)$
$X \subseteq Y \Rightarrow \gamma(X) \subseteq \gamma(Y)$
$T \subseteq U \Rightarrow \gamma(T) \subseteq \gamma(U)$

4 $\mathrm{PREF}(\gamma(X)) = \gamma(\mathrm{PREF}(X))$

5 Let T be a process. Let $t \in \mathrm{t}T$. Then
$\gamma(T)$ is a process
$\gamma(suc(t, T)) \subseteq suc(\gamma(t), \gamma(T))$

$$suc(\gamma(t),\gamma(T)) = (\cup s : s \in \mathbf{t}T \wedge \gamma(s) = \gamma(t) : \gamma(suc(s,T)))$$
$$\gamma(after(t,T)) \subseteq after(\gamma(t),\gamma(T))$$
$$after(\gamma(t),\gamma(T)) = (\cup s : s \in \mathbf{t}T \wedge \gamma(s) = \gamma(t) : \gamma(after(s,T)))$$

(End of Property)

Notice that in the above property process $\gamma(T)$ is identified with a process whose alphabet is $\gamma(\mathbf{a}T) \times \{\surd\}$, since formally speaking $\gamma(\mathbf{a}T)$ is not a subset of $\Gamma \times \mathrm{M}_{\surd} = \Omega$. Process $\gamma(T)$ describes the communication behaviour of process T.

Due to our interpretation of the pairs in Ω projection is almost always done on alphabets of the form $C \times \mathrm{M}_{\surd}$ where C is a channel set ($C \subseteq \Gamma$). Therefore, we introduce the following abbreviation. Projection of trace t on channel set C, denoted by $t\restriction C$, is defined by

$$t\restriction C = t\restriction(C \times \mathrm{M}_{\surd})$$

Accordingly, projection of trace sets, trace structures, and processes on channel sets is defined. Projection on channel sets has properties similar to properties of ordinary projection. We only mention one extra property.

Property 3.1.1

$$\gamma(s\restriction C) = \gamma(s)\restriction C$$
$$\gamma(T\restriction C) = \gamma(T)\restriction C$$

(End of Property)

Let T be a process. The *channel set* of T, denoted by $\mathbf{c}T$, is defined by

$$\mathbf{c}T = \gamma(\mathbf{a}T)$$

The elements of $\mathbf{c}T$ are called the *channels* of process T. Let c be a channel of T. The *type* of channel c in T, denoted by $type(c,T)$, is defined by

$$type(c,T) = \{\, m \mid m \in \mathrm{M}_{\surd} \wedge \langle c, m\rangle \in \mathbf{a}T \,\}$$

Notice that $type(c,T)$ equals the set of all messages that T may transmit via channel c according to its alphabet. The empty message $\surd$ was introduced explicitly to model signal transmission. Obviously, we must require that

$$(\mathbf{A}\, c : c \in \mathbf{c}T : \surd \in type(c,T) \;\equiv\; \{\surd\} = type(c,T))$$

In the following we tacitly assume all processes to satisfy this condition. Observe that for process T satisfying this condition and channel set C process $T\restriction C$ satisfies this condition as well.

Property 3.1.2

Let T be a process and C be a channel set.

0 $\mathbf{a}T = (\cup c : c \in \mathbf{c}T : \{c\} \times type(c,T))$

1 $\mathbf{c}(T\restriction C) = \mathbf{c}T \cap C$

2 $type(c, T\restriction C) = type(c,T) \qquad c \in \mathbf{c}T \cap C$

(End of Property)

Next, we investigate the composition of processes. The following two examples show that weaving of two processes may yield a result that is not in accordance with our appreciation of processes and their composition.

Example 3.1.3

Process $T = \langle \{\langle a,0\rangle\}, \{\langle a,0\rangle\}^* \rangle$ describes a mechanism that can repeatedly send zeroes via channel a. Process $U = \langle \{a\} \times \mathcal{Z}, (\{a\} \times \mathcal{Z})^* \rangle$ describes a mechanism that can repeatedly receive arbitrary integer values via channel a. On basis of our appreciation we would expect the composite of T and U to be described by $\langle \{a\} \times \mathcal{Z}, \{\langle a,0\rangle\}^* \rangle$. However, by just applying the definition of weaving we obtain

$$T \mathbf{w} U = \langle \{a\} \times \mathcal{Z}, (\{a\} \times \mathcal{Z})^* \rangle \qquad (= U)$$

(End of Example)

Example 3.1.4

Let $T = \mathrm{RUN}(\{\langle a, \mathrm{false}\rangle\})$ and $U = \mathrm{RUN}(\{\langle a, \mathrm{true}\rangle\})$.
We now have $T \mathbf{w} U = \mathrm{RUN}(\{\langle a, \mathrm{false}\rangle, \langle a, \mathrm{true}\rangle\})$. On basis of our appreciation we would expect the composite of T and U to be $\langle \{\langle a, \mathrm{false}\rangle, \langle a, \mathrm{true}\rangle\}, \{\varepsilon\} \rangle$.

(End of Example)

Observe that in both examples the type of channel a in process T differs from the type of a in process U. If in example 3.1.3 $type(a,T)$ and $type(a,U)$ were both $\mathcal{Z}$, and in example 3.1.4 $type(a,T)$ and $type(a,U)$ were both $\{\mathrm{false}, \mathrm{true}\}$, the weave of T and U

would yield the expected result.
Hence, we define the following notion. Processes T and U are said to be *compatible* if

$$(\mathbf{A}\, c : c \in \mathbf{c}T \cap \mathbf{c}U : type(c,T) = type(c,U))$$

It is easily seen that if two processes are compatible their weave will be the process we expect on basis of our appreciation. Moreover, if processes T and U are compatible, process $T\,\mathbf{w}\,U$ satisfies the tacit assumption

$$(\mathbf{A}\, c : c \in \mathbf{c}(T\,\mathbf{w}\,U) : \surd \in type(c, T\,\mathbf{w}\,U) \;\equiv\; \{\surd\} = type(c, T\,\mathbf{w}\,U))$$

provided that T and U satisfy it. In the following we consider composition of compatible processes only. Therefore, we tacitly assume processes that are to be composed to be compatible.

Property 3.1.5

Let T and U be processes. Then

0 $\mathbf{t}(T\,\mathbf{w}\,U) = \{\, t \mid t \in (\mathbf{a}T \cup \mathbf{a}U)^* \wedge t\restriction \mathbf{c}T \in \mathbf{t}T \wedge t\restriction \mathbf{c}U \in \mathbf{t}U \,\}$

1 $\mathbf{c}(T\,\mathbf{w}\,U) = \mathbf{c}T \cup \mathbf{c}U$

2 $\gamma(T\,\mathbf{w}\,U) \subseteq \gamma(T)\,\mathbf{w}\,\gamma(U)$

Proof

We prove only 2. We derive

$$\begin{array}{ll}
 & \gamma(T\,\mathbf{w}\,U) \\
= & \quad \{\text{definition } \gamma\} \\
 & \langle\, \gamma(\mathbf{a}(T\,\mathbf{w}\,U)), \{\, \gamma(s) \mid s \in \mathbf{t}(T\,\mathbf{w}\,U) \,\} \,\rangle \\
= & \quad \{\text{definition channel set, 0}\} \\
 & \langle\, \mathbf{c}(T\,\mathbf{w}\,U), \{\, \gamma(s) \mid s \in (\mathbf{a}T \cup \mathbf{a}U)^* \wedge s\restriction \mathbf{c}T \in \mathbf{t}T \wedge s\restriction \mathbf{c}U \in \mathbf{t}U \,\} \,\rangle \\
\subseteq & \quad \{1,\ \text{property 3.1.1, definition channel set}\} \\
 & \langle\, \mathbf{c}T \cup \mathbf{c}U, \{\, \gamma(s) \mid \gamma(s) \in (\mathbf{c}T \cup \mathbf{c}U)^* \wedge \gamma(s)\restriction \mathbf{c}T \in \mathbf{t}\gamma(T) \wedge \gamma(s)\restriction \mathbf{c}U \in \mathbf{t}\gamma(U) \,\} \,\rangle \\
= & \quad \{\text{definition weave, definition channel set}\} \\
 & \gamma(T)\,\mathbf{w}\,\gamma(U)
\end{array}$$

(End of Proof)

The inclusion in property 3.1.5.2 may be a true inclusion as the following example shows.

Example 3.1.6

$$T = \langle\, \{a\} \times \mathcal{Z}, \{\varepsilon, \langle a, 0\rangle\}\,\rangle$$
$$U = \langle\, \{a\} \times \mathcal{Z}, \{\varepsilon, \langle a, 1\rangle\}\,\rangle$$
$$T \,\mathbf{w}\, U = \mathrm{STOP}(\{a\} \times \mathcal{Z})$$
$$\gamma(T) = \gamma(U) = \mathrm{PR}(a)$$
$$\gamma(T \,\mathbf{w}\, U) = \mathrm{STOP}(a) \subset \mathrm{PR}(a) = \gamma(T) \,\mathbf{w}\, \gamma(U)$$

(End of Example)

We modify the definition of systems. We require that the processes of a system are compatible with one another, and replace the external alphabet of a system by a channel set consisting of the external channels of the system. More formally, a *system* is a pair $\langle C, X\rangle$ where C is a channel set and X is a set of processes such that every process $T \in X$ and every process $U \in X$ are compatible, and $C \subseteq \mathbf{cW}(X)$. Let S be a system. Instead of denoting the external alphabet, $\mathbf{e}S$ now denotes the external channel set of system S. The (external) process of system S, denoted by $\mathrm{PR}(S)$, is defined by $\mathrm{PR}(S) = \mathbf{W}(\mathbf{p}S)\restriction \mathbf{e}S$. The type of external channel c of system S, denoted by $type(c, S)$, is defined by

$$type(c, S) = type(c, \mathbf{W}(\mathbf{p}S))$$

The projection of system S on channel set C, denoted by $S\restriction C$, is defined by

$$S\restriction C = \langle\, \mathbf{e}S \cap C, \mathbf{p}S\,\rangle$$

For every system S the system $\gamma(S)$ is defined by

$$\gamma(S) = \langle\, \mathbf{e}S, \gamma(\mathbf{p}S)\,\rangle$$

Let S and T be systems. In order to compose S and T we require that $\mathbf{cW}(\mathbf{p}S) \cap \mathbf{cW}(\mathbf{p}T) = \mathbf{e}S \cap \mathbf{e}T$ and

$$(\mathbf{A}\, c : c \in \mathbf{e}S \cap \mathbf{e}T : type(c, S) = type(c, T)\,)$$

If S and T satisfy this last condition they are said to be *compatible*. For compatible systems S and T satisfying $\mathbf{cW}(\mathbf{p}S) \cap \mathbf{cW}(\mathbf{p}T) = \mathbf{e}S \cap \mathbf{e}T$ the composite of S and T, denoted by $S \parallel T$, is defined by

$$S \parallel T = \langle\, \mathbf{e}S \cup \mathbf{e}T, \mathbf{p}S \cup \mathbf{p}T\,\rangle$$

Notice that due to the imposed restrictions $S \parallel T$ is indeed a system. In the following we tacitly assume that systems that are to be composed are compatible.

Theorem 3.1.7

Let S and T be systems. Let C be a channel set. Then

$$\begin{array}{lcl} \gamma(S \restriction C) & = & \gamma(S) \restriction C \\ \gamma(S \parallel T) & = & \gamma(S) \parallel \gamma(T) \end{array}$$

(End of Theorem)

3.2 Data independence

A process is said to be data independent if the communication behaviour after the current trace depends on the communication pattern associated with the current trace, and not on the messages that have been sent and received. Formally, process T is called *data independent* if

$$(\mathbf{A}\, t : t \in \mathbf{t}T : \gamma(\mathit{after}(t, T)) = \mathit{after}(\gamma(t), \gamma(T)))$$

Example 3.2.0

Process

$$\begin{array}{l} \mathrm{ADDER} = \mathrm{PREF}(\langle \{a, b, c\} \times \mathcal{Z} \\ \qquad\qquad , (\{ \langle a, m\rangle\langle b, n\rangle\langle c, m+n\rangle \mid m, n \in \mathcal{Z} \} \\ \qquad\qquad \cup \{ \langle b, n\rangle\langle a, m\rangle\langle c, m+n\rangle \mid m, n \in \mathcal{Z} \})^*)) \end{array}$$

is data independent.

(End of Example)

The next theorem gives three alternative characterizations of data independence.

Theorem 3.2.1

Let T be a process. The following four assertions are equivalent.

0 $(\mathbf{A}\, t : t \in \mathbf{t}T : \gamma(after(t,T)) = after(\gamma(t),\gamma(T)))$

1 $(\mathbf{A}\, s,t : s \in \mathbf{t}T \wedge t \in \mathbf{t}T \wedge \gamma(s) = \gamma(t) : \gamma(after(s,T)) = \gamma(after(t,T)))$

2 $(\mathbf{A}\, t : t \in \mathbf{t}T : \gamma(suc(t,T)) = suc(\gamma(t),\gamma(T)))$

3 $(\mathbf{A}\, s,t : s \in \mathbf{t}T \wedge t \in \mathbf{t}T \wedge \gamma(s) = \gamma(t) : \gamma(suc(s,T)) = \gamma(suc(t,T)))$

Proof

Using property 3.1.0 one easily establishes the equivalence of 0 and 1, and of 2 and 3. Furthermore, it is easily seen that 1 implies 3. It remains to prove that 3 implies 1.

Assume that 3 holds. Let $s,t \in \mathbf{t}T$ be such that $\gamma(s) = \gamma(t)$. By induction on the length of trace u we prove

$$(\mathbf{A}\, u : u \in \Omega^* : u \in \mathbf{t}\gamma(after(s,T)) \equiv u \in \mathbf{t}\gamma(after(t,T)))$$

base $\ell(u) = 0$

$$\begin{array}{ll} & \varepsilon \in \mathbf{t}\gamma(after(s,T)) \\ = & \quad \{\, s\varepsilon \in \mathbf{t}T \,\} \\ & \text{true} \\ = & \quad \{\, t\varepsilon \in \mathbf{t}T \,\} \\ & \varepsilon \in \mathbf{t}\gamma(after(t,T)) \end{array}$$

step $\ell(u) > 0$

Assume

$$(\mathbf{A}\, w : w \in \Omega^* \wedge \ell(w) < \ell(u) : w \in \mathbf{t}\gamma(after(s,T)) \equiv w \in \mathbf{t}\gamma(after(t,T)))$$

Let $u = va$. We derive

$$\begin{array}{ll} & u \in \mathbf{t}\gamma(after(s,T)) \\ = & \quad \{\, u = va \,\} \\ & v \in \mathbf{t}\gamma(after(s,T)) \wedge va \in \mathbf{t}\gamma(after(s,T)) \\ = & \quad \{\, \text{induction hypothesis, } \ell(v) < \ell(u) \,\} \\ & v \in \mathbf{t}\gamma(after(t,T)) \wedge va \in \mathbf{t}\gamma(after(s,T)) \\ = & \quad \{\, \text{definition } after\text{, successor set, and } \gamma \,\} \\ & (\mathbf{E}\, y,z : y,z \in \mathbf{a}T^* : v = \gamma(y) \wedge ty \in \mathbf{t}T \\ & \qquad\qquad \wedge v = \gamma(z) \wedge sz \in \mathbf{t}T \wedge a \in \gamma(suc(sz,T))) \end{array}$$

Analogously, one can derive

$$u \in \mathbf{t}\gamma(after(t,T)) \equiv$$
$$(\mathbf{E}\, y,z : y,z \in \mathbf{a}T^* : v = \gamma(y) \wedge ty \in \mathbf{t}T$$
$$\wedge\ v = \gamma(z) \wedge sz \in \mathbf{t}T \wedge a \in \gamma(suc(ty,T)))$$

From $\gamma(s) = \gamma(t)$ and 3 it now follows that

$$u \in \mathbf{t}\gamma(after(s,T)) \equiv u \in \mathbf{t}\gamma(after(t,T))$$

(End of Proof)

Comparing the definition of data independence with the definition of non-disabling we see a close resemblance. Data independence can indeed be expressed in terms of transparence. To do this we first introduce the function $\theta \in (\Gamma \times \mathrm{M}_{\surd})^* \to (\Gamma \cup \mathrm{M}_{\surd})^*$ defined by

$$\begin{array}{lcl} \theta(\varepsilon) & = & \varepsilon \\ \theta(t\langle c,m\rangle) & = & \theta(t)\,c\,m \end{array}$$

Function θ is defined for trace sets in the obvious manner. For every process T the process $\theta(T)$ is defined by

$$\theta(T) = \mathrm{PREF}(\,\langle\, \mathbf{c}T \cup (\cup c : c \in \mathbf{c}T : type(c,T)\,),\, \theta(\mathbf{t}T)\,\rangle\,)$$

Furthermore, we assume that $\Gamma \cap \mathrm{M}_{\surd} = \varnothing$.

Lemma 3.2.2

Let T be a process and let $t \in \mathbf{t}T$.

0 $\gamma(t) = \theta(t)\restriction \mathbf{c}T$

1 $\gamma(T) = \theta(T)\restriction \mathbf{c}T$

2 $\gamma(suc(t,T)) = suc(\theta(t),\theta(T)) \subseteq \mathbf{c}T$
$\gamma(after(t,T)) = after(\theta(t),\theta(T))\restriction \mathbf{c}T$

3 $(\mathbf{A}\, u : u \in \mathbf{t}\theta(T) : suc(u,\theta(T)) \subseteq \mathbf{c}T \equiv (\mathbf{E}\, s : s \in \mathbf{t}T : \theta(s) = u))$

4 $\mathbf{c}T$ is non-divergent with respect to $\theta(T)$

(End of Lemma)

Theorem 3.2.3

Let T be a process. Then

$$T \text{ is data independent} \quad \equiv \quad \mathbf{c}T \text{ is transparent with respect to } \theta(T)$$

Proof

$$\begin{array}{ll} & T \text{ is data independent} \\ = & \quad \{ \text{theorem 3.2.1} \} \\ & (\mathbf{A}\, t : t \in \mathbf{t}T : \gamma(suc(t,T)) = suc(\gamma(t), \gamma(T))\,) \\ = & \quad \{ \text{lemma 3.2.2} \} \\ & (\mathbf{A}\, u : u \in \mathbf{t}\theta(T) \wedge suc(u, \theta(T)) \subseteq \mathbf{c}T : suc(u, \theta(T)) = suc(u \restriction \mathbf{c}T, \theta(T) \restriction \mathbf{c}T)\,) \\ = & \quad \{ \mathbf{c}T \text{ is non-divergent with respect to } \theta(T), \text{ theorem 2.1.20} \} \\ & \mathbf{c}T \text{ is transparent with respect to } \theta(T) \end{array}$$

(End of Proof)

Next, we investigate when the projection of a data independent process on a channel set is data independent. We have the following result.

Theorem 3.2.4

Let T be a data independent process. Let C be a channel set such that $C \subseteq \mathbf{c}T$. If C is non-disabling with respect to $\gamma(T)$ then $T \restriction C$ is data independent.

Proof

Assume C is non-disabling with respect to $\gamma(T)$. Let $t \in \mathbf{t}T$. We derive

$$\begin{array}{ll} & \gamma(after(t \restriction C, T \restriction C)) \\ \subseteq & \quad \{ \text{property 3.1.0} \} \\ & after(\gamma(t \restriction C), \gamma(T \restriction C)) \\ = & \quad \{ \text{property 3.1.1} \} \\ & after(\gamma(t) \restriction C, \gamma(T) \restriction C) \\ = & \quad \{ C \text{ is non-disabling with respect to } \gamma(T) \} \\ & after(\gamma(t), \gamma(T)) \restriction C \\ = & \quad \{ T \text{ is data independent} \} \\ & \gamma(after(t, T)) \restriction C \\ = & \quad \{ \text{property 3.1.1} \} \end{array}$$

$$
\begin{array}{ll}
& \gamma(after(t,T)\restriction C) \\
\subseteq & \quad \{\text{property projection}\} \\
& \gamma(after(t\restriction C, T\restriction C))
\end{array}
$$

(End of Proof)

The following example shows that if $T\restriction C$ is data independent C does not necessarily have to be non-disabling with respect to $\gamma(T)$.

Example 3.2.5

$T = \text{PREF}(\langle\, \{a,b,c,d,e\} \times \mathcal{Z}\,,\, \{\langle a,0\rangle\langle c,0\rangle\langle d,0\rangle\,,\, \langle b,0\rangle\langle c,0\rangle\langle e,0\rangle\}\,\rangle)$
$C = \{c,d,e\}$
T is data independent
$T\restriction C = \text{PREF}(\langle\, \{c,d,e\} \times \mathcal{Z}\,,\, \{\langle c,0\rangle\langle d,0\rangle\,,\, \langle c,0\rangle\langle e,0\rangle\}\,\rangle)$
$T\restriction C$ is data independent
$\gamma(T) = \text{PR}(\, a\,;\, c\,;\, d \mid b\,;\, c\,;\, e\,)$
C is not non-disabling with respect to $\gamma(T)$
$U = \text{PREF}(\langle\, \{a,b,c,d,e\} \times \mathcal{Z}\,,\, \{\langle a,0\rangle\langle c,0\rangle\langle d,0\rangle\,,\, \langle b,0\rangle\langle c,1\rangle\langle e,1\rangle\}\,\rangle)$
U is data independent
$U\restriction C = \text{PREF}(\langle\, \{c,d,e\} \times \mathcal{Z}\,,\, \{\langle c,0\rangle\langle d,0\rangle\,,\, \langle c,1\rangle\langle e,1\rangle\}\,\rangle)$
$U\restriction C$ is not data independent
$\gamma(U) = \gamma(T)$

(End of Example)

We now investigate the data independence of the weave of two data independent processes. First, we have the following lemma.

Lemma 3.2.6

Let T and U be data independent processes. Then

$$
\begin{array}{l}
(\mathbf{A}\, t : t \in \mathbf{t}(T\,\mathbf{w}\,U) \\
\quad : \gamma(\, suc(t\restriction \mathbf{c}T, T) \cap suc(t\restriction \mathbf{c}U, U)) = \gamma(suc(t\restriction \mathbf{c}T, T)) \;\cap\; \gamma(suc(t\restriction \mathbf{c}U, U)) \\
\quad\quad \equiv\; \gamma(suc(t, T\,\mathbf{w}\,U)) = suc(\gamma(t), \gamma(T)\,\mathbf{w}\,\gamma(U))\,)
\end{array}
$$

Proof

Let $t \in \mathbf{t}(T\,\mathbf{w}\,U)$. We derive

$$
\begin{array}{ll}
& \gamma(\, suc(t, T \,\mathbf{w}\, U)\,) \\
= & \quad \{\text{ theorem 1.1.17, calculus }\} \\
& \gamma(\, suc(t\restriction \mathbf{c}T, T) \cap suc(t\restriction \mathbf{c}U, U)) \\
& \cup\ \gamma(\, suc(t\restriction \mathbf{c}T, T) \setminus \mathbf{a}U\,) \cup \gamma(\, suc(t\restriction \mathbf{c}U, U) \setminus \mathbf{a}T\,) \\
= & \quad \{ T \text{ and } U \text{ are compatible, } T \text{ and } U \text{ are data independent }\} \\
& \gamma(\, suc(t\restriction \mathbf{c}T, T) \cap suc(t\restriction \mathbf{c}U, U)) \\
& \cup\ suc(\gamma(t)\restriction \mathbf{c}T, \gamma(T)) \setminus \mathbf{c}U \cup suc(\gamma(t)\restriction \mathbf{c}U, \gamma(U)) \setminus \mathbf{c}T
\end{array}
$$

and

$$
\begin{array}{ll}
& suc(\, \gamma(t), \gamma(T) \,\mathbf{w}\, \gamma(U)\,) \\
= & \quad \{\text{ theorem 1.1.17 }\} \\
& (suc(\, \gamma(t)\restriction \mathbf{c}T, \gamma(T)) \cap suc(\gamma(t)\restriction \mathbf{c}U, \gamma(U))\,) \\
& \cup\ suc(\gamma(t)\restriction \mathbf{c}T, \gamma(T)) \setminus \mathbf{c}U \cup suc(\gamma(t)\restriction \mathbf{c}U, \gamma(U)) \setminus \mathbf{c}T \\
= & \quad \{ T \text{ and } U \text{ are data independent }\} \\
& (\, \gamma(suc(t\restriction \mathbf{c}T, T)) \cap \gamma(suc(t\restriction \mathbf{c}U, U))\,) \\
& \cup\ suc(\gamma(t)\restriction \mathbf{c}T, \gamma(T)) \setminus \mathbf{c}U \cup suc(\gamma(t)\restriction \mathbf{c}U, \gamma(U)) \setminus \mathbf{c}T
\end{array}
$$

Observe that in the above derivations only unions of disjoint sets occur. Hence, we infer

$$
\begin{array}{l}
\gamma(suc(t, T \,\mathbf{w}\, U)) = suc(\gamma(t), \gamma(T) \,\mathbf{w}\, \gamma(U)) \\
\quad \equiv\ \gamma(\, suc(t\restriction \mathbf{c}T, T) \cap suc(t\restriction \mathbf{c}U, U)) = \gamma(suc(t\restriction \mathbf{c}T, T)) \,\cap\, \gamma(suc(t\restriction \mathbf{c}U, U))
\end{array}
$$

(End of Proof)

Theorem 3.2.7

Let T and U be data independent processes. Then

$$
\begin{array}{l}
(\mathbf{A}\, t : t \in \mathbf{t}(T \,\mathbf{w}\, U) : \gamma(\, suc(t\restriction \mathbf{c}T, T) \cap suc(t\restriction \mathbf{c}U, U)) \\
\qquad\qquad\qquad = \gamma(suc(t\restriction \mathbf{c}T, T)) \,\cap\, \gamma(suc(t\restriction \mathbf{c}U, U))\,) \\
\quad \equiv\ (T \,\mathbf{w}\, U \text{ is data independent }) \,\wedge\, \gamma(T \,\mathbf{w}\, U) = \gamma(T) \,\mathbf{w}\, \gamma(U)
\end{array}
$$

Proof

$$
\begin{array}{ll}
& (\mathbf{A}\, t : t \in \mathbf{t}(T \,\mathbf{w}\, U) : \gamma(\, suc(t\restriction \mathbf{c}T, T) \cap suc(t\restriction \mathbf{c}U, U)) \\
& \qquad\qquad\qquad = \gamma(suc(t\restriction \mathbf{c}T, T)) \cap \gamma(suc(t\restriction \mathbf{c}U, U))\,) \\
= & \quad \{\text{ lemma 3.2.6 }\} \\
& (\mathbf{A}\, t : t \in \mathbf{t}(T \,\mathbf{w}\, U) : \gamma(suc(t, T \,\mathbf{w}\, U)) = suc(\gamma(t), \gamma(T) \,\mathbf{w}\, \gamma(U))\,)
\end{array}
$$

$$
\begin{array}{ll}
= & \{\text{ see note }\} \\
& (\mathbf{A}\, t : t \in \mathbf{t}(T\,\mathbf{w}\,U) : \gamma(suc(t, T\,\mathbf{w}\,U)) = suc(\gamma(t), \gamma(T\,\mathbf{w}\,U))\,) \\
& \wedge\ (\mathbf{A}\, t : t \in \mathbf{t}(T\,\mathbf{w}\,U) : suc(\gamma(t), \gamma(T\,\mathbf{w}\,U)) = suc(\gamma(t), \gamma(T)\,\mathbf{w}\,\gamma(U))\,) \\
= & \{\text{ theorem 3.2.1, calculus }\} \\
& (T\,\mathbf{w}\,U \text{ is data independent}) \wedge \gamma(T\,\mathbf{w}\,U) = \gamma(T)\,\mathbf{w}\,\gamma(U)
\end{array}
$$

note

Assuming $t \in \mathbf{t}(T\,\mathbf{w}\,U) \wedge \gamma(suc(t, T\,\mathbf{w}\,U)) = suc(\gamma(t), \gamma(T)\,\mathbf{w}\,\gamma(U))$ we derive

$$
\begin{array}{ll}
& \gamma(suc(t, T\,\mathbf{w}\,U)) \\
\subseteq & \{\text{ property 3.1.0 }\} \\
& suc(\gamma(t), \gamma(T\,\mathbf{w}\,U)) \\
\subseteq & \{\text{ property 3.1.5 }\} \\
& suc(\gamma(t), \gamma(T)\,\mathbf{w}\,\gamma(U)) \\
= & \{\text{ assumption }\} \\
& \gamma(suc(t, T\,\mathbf{w}\,U))
\end{array}
$$

(End of Proof)

The following theorem provides conditions implying data independence of the weave that are stronger than the left hand side of the equivalence in 3.2.7.

Theorem 3.2.8

Let T and U be data independent processes. Let C and D be channel sets such that $C \subseteq \mathbf{c}T$, $D \subseteq \mathbf{c}U$, and $\mathbf{c}T \cap \mathbf{c}U \subseteq C \cup D$. If T and C satisfy

$$(\mathbf{A}\, t, c : t \in \mathbf{t}T \wedge c \in \gamma(suc(t, T)) \cap C : \{c\} \times type(c, T) \subseteq suc(t, T)\,)$$

and U and D satisfy

$$(\mathbf{A}\, u, d : u \in \mathbf{t}U \wedge d \in \gamma(suc(u, U)) \cap D : \{d\} \times type(d, U) \subseteq suc(u, U)\,)$$

then $T\,\mathbf{w}\,U$ is data independent, $\gamma(T\,\mathbf{w}\,U) = \gamma(T)\,\mathbf{w}\,\gamma(U)$, and $T\,\mathbf{w}\,U$ and $E = (C \cap D) \cup C \setminus \mathbf{c}U \cup D \setminus \mathbf{c}T$ satisfy

$$(\mathbf{A}\, t, c : t \in \mathbf{t}(T\,\mathbf{w}\,U) \wedge c \in \gamma(suc(t, T\,\mathbf{w}\,U)) \cap E : \{c\} \times type(c, T\,\mathbf{w}\,U) \subseteq suc(t, T\,\mathbf{w}\,U)\,)$$

(End of Theorem)

Being rather straightforward the proof of the above theorem is omitted.

Notice that the conditions in theorem 3.2.8 are both in terms of one process only. These conditions are met if we, for instance, distinguish between input and output channels in both processes thereby requiring that a process puts no restrictions on the values it is willing to receive via its input channels, and that each channel connecting the processes is an input channel in one process and an output channel in the other process.

Theorem 3.2.9

Let X be a set of data independent processes. Let for every process $T \in X$ C_T be a channel set such that $C_T \subseteq \mathbf{c}T$. If

$$0 \quad (\mathbf{A}\, t : t \in \mathbf{t}T \wedge c \in \gamma(suc(t,T)) \cap C_T : \{c\} \times type(c,T) \subseteq suc(t,T))$$
$$1 \quad (\mathbf{A}\, T,U : T \in X \wedge U \in X : \mathbf{c}T \cap \mathbf{c}U \subseteq C_T \cup C_U)$$

then $\mathbf{W}(X)$ is data independent, $\gamma(\mathbf{W}(X)) = \mathbf{W}(\gamma(X))$, and

$$(\mathbf{A}\, t : t \in \mathbf{t}\mathbf{W}(X) \wedge c \in \gamma(suc(t, \mathbf{W}(X))) \cap C : \{c\} \times type(c, \mathbf{W}(X)) \subseteq suc(t, \mathbf{W}(X)))$$

where

$$C = \{\, c \mid c \in \mathbf{c}\mathbf{W}(X) \wedge (\mathbf{A}\, T : T \in X \wedge c \in \mathbf{c}T : c \in C_T)\,\} \cup (\cup T : T \in X : C_T \setminus (\cup U : U \in X \setminus \{T\} : \mathbf{c}U))$$

If S is a system such that $\mathbf{p}S = X$ and $\gamma(S)$ is non-disabling, then $\mathrm{PR}(S)$ is data independent.

(End of Theorem)

Theorem 3.2.10

Let $(T_n)_{n \geq 0}$ be a sequence of data independent processes such that $(\mathbf{A}\, n : n \geq 0 : T_n \subseteq T_{n+1})$. Then $(\cup n : n \geq 0 : T_n)$ is data independent.

Proof

Let $t \in \mathbf{t}(\cup n : n \geq 0 : T_n)$. We derive

$$\begin{aligned} & \gamma(\, after(t, (\cup n : n \geq 0 : T_n))\,) \\ = \quad & \{\, \text{property 1.1.3} \,\} \\ & \gamma(\, (\cup n : n \geq 0 \wedge t \in \mathbf{t}T_n : after(t, T_n))\,) \end{aligned}$$

$$
\begin{array}{ll}
= & \{\text{ calculus }\} \\
& (\cup n : n \geq 0 \wedge t \in \mathbf{t}T_n : \gamma(\mathit{after}(t, T_n))) \\
= & \{(\mathbf{A}\, n : n \geq 0 : T_n \text{ is data independent })\} \\
& (\cup n : n \geq 0 \wedge t \in \mathbf{t}T_n : \mathit{after}(\gamma(t), \gamma(T_n))) \\
= & \{(\mathbf{A}\, n : n \geq 0 : T_n \subseteq T_{n+1}),\ \text{property } 1.1.3\} \\
& (\cup n : n \geq 0 \wedge \gamma(t) \in \mathbf{t}\gamma(T_n) : \mathit{after}(\gamma(t), \gamma(T_n))) \\
= & \{\text{ property } 1.1.3\} \\
& \mathit{after}(\gamma(t), (\cup n : n \geq 0 : \gamma(T_n))) \\
= & \{\text{ calculus }\} \\
& \mathit{after}(\gamma(t), \gamma((\cup n : n \geq 0 : T_n)))
\end{array}
$$

(End of Proof)

3.3 Split specifications

In section 1.2 we introduced specifications as a way to describe processes. A data independent process, however, may be described in a somewhat different manner. It is completely determined by its communication behaviour, the types of its channels, and the relation between the messages it sends and receives, i.e. its input/output relation. This is formalized as follows.

A *split specification* is a triple $\langle T, f, P \rangle$ where T is a process with $\mathbf{a}T \subseteq \Gamma$, $f \in \mathbf{a}T \to (\mathcal{P}(\mathrm{M}) \setminus \{\varnothing\}) \cup \{\{\surd\}\}$ is a function, and P is a predicate on $A = (\cup c : c \in \mathbf{a}T : \{c\} \times f(c))^*$ such that

$$
\begin{array}{l}
P(\varepsilon) \\
(\mathbf{A}\, c, t : c \in \mathbf{a}T \wedge t \in A^* : \gamma(t)c \in \mathbf{t}T \wedge (\mathbf{A}\, s : s \leq t : P(s)) \\
\qquad\qquad \Rightarrow\ (\mathbf{E}\, m : m \in f(c) : P(t\langle c, m\rangle)))
\end{array}
$$

Process T describes the communication behaviour and function f the types of the channels. The last condition mentioned above states that predicate P may put no further restrictions on the communication behaviour described by T. The process specified by split specification $\langle T, f, P \rangle$ is

$$
\begin{array}{l}
\langle (\cup c : c \in \mathbf{a}T : \{c\} \times f(c)) \\
\quad , \{\, t \mid t \in (\cup c : c \in \mathbf{a}T : \{c\} \times f(c))^* \wedge \gamma(t) \in \mathbf{t}T \wedge (\mathbf{A}\, s : s \leq t : P(s))\,\}\rangle
\end{array}
$$

The process specified by a split specification is data independent as the following theorem shows.

Theorem 3.3.0

Let $\langle T, f, P \rangle$ be a split specification specifying process U.
Then U is data independent and $\gamma(U) = T$.

Proof

Obviously $\gamma(U) \subseteq T$. By a simple inductive argument one can prove

$$(\mathbf{A}\, k : k \geq 0 : \{ t \mid t \in \mathbf{t}T \wedge \ell(t) \leq k \} \subseteq \mathbf{t}\gamma(U))$$

which implies $T \subseteq \gamma(U)$.
Let $t \in \mathbf{t}U$. We will show that $suc(\gamma(t), \gamma(U)) = \gamma(suc(t, U))$. Since $t \in \mathbf{t}U$ we have $\gamma(t) \in \mathbf{t}T$ and $(\mathbf{A}\, s : s \leq t : P(s))$. We derive

$$\begin{array}{ll} & c \in suc(\gamma(t), \gamma(U)) \\ = & \quad \{ \gamma(U) = T, (\mathbf{A}\, s : s \leq t : P(s)) \} \\ & \gamma(t)c \in \mathbf{t}T \wedge (\mathbf{A}\, s : s \leq t : P(s)) \\ = & \quad \{ \langle T, f, P \rangle \text{ is a split specification} \} \\ & (\mathbf{E}\, m : m \in f(c) : \gamma(t)c \in \mathbf{t}T \wedge (\mathbf{A}\, s : s \leq t\langle c, m \rangle : P(s))) \\ = & \quad \{ \gamma(t\langle c, m \rangle) = \gamma(t)c, \text{ definition } U \} \\ & (\mathbf{E}\, m : m \in f(c) : t\langle c, m \rangle \in \mathbf{t}U) \\ = & \quad \{ \text{definition } \gamma \text{ en } suc \} \\ & c \in \gamma(suc(t, U)) \end{array}$$

(End of Proof)

The following theorem shows that every data independent process can be specified by a split specification.

Theorem 3.3.1

Let U be a data independent process. Define $f \in \mathbf{c}U \rightarrow (\mathcal{P}(\mathrm{M}) \setminus \{\varnothing\}) \cup \{\{\surd\}\}$ by $f(c) = type(c, U)$ for all $c \in \mathbf{c}U$. Then $\langle \gamma(U), f, t : t \in \mathbf{t}U \rangle$ is a split specification specifying U.

Proof

Notice that $\varepsilon \in \mathbf{t}U$. Let $t \in \mathbf{a}U^*$ and $c \in \mathbf{c}U$. We derive

$$\begin{array}{ll} & \gamma(t)c \in \mathbf{t}\gamma(U) \wedge (\mathbf{A}\, s : s \leq t : s \in \mathbf{t}U) \\ = & \quad \{ \text{definition } suc, U \text{ is a process} \} \end{array}$$

$$
\begin{array}{ll}
& c \in suc(\gamma(t), \gamma(U)) \wedge t \in \mathbf{t}U \\
= & \quad \{\, U \text{ is data independent} \,\} \\
& c \in \gamma(suc(t, U)) \wedge t \in \mathbf{t}U \\
= & \quad \{\, \text{definition } \gamma \text{ and } suc \,\} \\
& (\mathbf{E}\, m : m \in type(c, U) : t\langle c, m\rangle \in \mathbf{t}U\,)
\end{array}
$$

Therefore, $\langle\, \gamma(U), f,\ t : t \in \mathbf{t}U \,\rangle$ is a split specification. Furthermore, we derive

$$
\begin{array}{ll}
& \langle\, (\cup c : c \in \mathbf{c}U : \{c\} \times f(c)\,) \\
& \quad , \{\, t \mid t \in (\cup c : c \in \mathbf{c}U : \{c\} \times f(c)\,)^* \wedge \gamma(t) \in \mathbf{t}\gamma(U) \\
& \qquad\qquad\qquad\qquad\qquad\qquad \wedge\ (\mathbf{A}\, s : s \leq t : s \in \mathbf{t}U\,)\,\} \,\rangle \\
= & \quad \{\, \text{definition } f, \text{ property 3.1.2, } U \text{ is a process} \,\} \\
& \langle\, \mathbf{a}U, \{\, t \mid t \in \mathbf{a}U^* \wedge t \in \mathbf{t}U \,\} \,\rangle \\
= & \quad \{\, \text{definition process} \,\} \\
& U
\end{array}
$$

(End of Proof)

The above two theorems show that the class of data independent processes and the class of processes generated by split specifications are identical.

Before presenting an example we introduce some notation. Function $\mu \in \Omega^* \rightarrow \mathrm{M}_{\surd}^*$ is defined by

$$
\begin{array}{rcl}
\mu(\varepsilon) & = & \varepsilon \\
\mu(t\langle c, m\rangle) & = & \mu(t)m
\end{array}
$$

Let t be a trace. Notice that $\mu(t)$ is a trace whose symbols are elements of $\mathrm{M}_{\surd}$. The i-th element of trace $\mu(t)$ is denoted by $\mu(t)[i]$ $(0 \leq i < \ell(t))$. Let c be a channel name. We define $c(i,t) = \mu(t\restriction c)[i]$ for all i, $0 \leq i < \ell(t\restriction c)$. Message $c(i,t)$ is the i-th message transmitted via channel c in trace t $(0 \leq i < \ell(t\restriction c))$. Whenever t is obvious from the context, we write $c(i)$ instead of $c(i,t)$.

Example 3.3.2

The process ADDER in example 3.2.0 may be specified by split specification

$$
\begin{array}{l}
\langle\, \mathrm{PR}((a\,,\, b;\, c)^*)\,,\ \{(a, \mathcal{Z}), (b, \mathcal{Z}), (c, \mathcal{Z})\} \\
\quad , t : (\mathbf{A}\, i : 0 \leq i < \ell(t\restriction a)\ \mathbf{min}\ \ell(t\restriction b)\ \mathbf{min}\ \ell(t\restriction c) : c(i) = a(i) + b(i)\,) \,\rangle
\end{array}
$$

(End of Example)

Theorem 3.3.3

Let $\langle T, f, P \rangle$ be a split specification specifying process V. Let S be a process such that $S \subseteq T$. Then $\langle S, f, P \rangle$ is a split specification. If U is the process specified by $\langle S, f, P \rangle$ we have $U \subseteq V$.

(End of Theorem)

Theorem 3.3.4

Let $\langle S, f, P \rangle$ and $\langle T, g, Q \rangle$ be split specifications specifying processes U and V, respectively. Then

$$\begin{aligned} & U \subseteq V \\ \equiv\ & S \subseteq T \wedge f = g \\ & \wedge (\mathbf{A}\, t : t \in \mathbf{a}U^* \wedge \gamma(t) \in \mathbf{t}S : (\mathbf{A}\, s : s \leq t : P(s)) \Rightarrow (\mathbf{A}\, s : s \leq t : Q(s))) \end{aligned}$$

and

$$\begin{aligned} & U = V \\ \equiv\ & S = T \wedge f = g \\ & \wedge (\mathbf{A}\, t : t \in \mathbf{a}U^* \wedge \gamma(t) \in \mathbf{t}S : (\mathbf{A}\, s : s \leq t : P(s)) \equiv (\mathbf{A}\, s : s \leq t : Q(s))) \end{aligned}$$

(End of Theorem)

The next two theorems give conditions under which the Conjunction-Weave Rule holds (cf. theorems 3.2.7 and 3.2.8).

Theorem 3.3.5

Let $\langle S, f, P \rangle$ and $\langle T, g, Q \rangle$ be split specifications specifying processes U and V, respectively. Let $f(c) = g(c)$ for all $c \in \mathbf{a}S \cap \mathbf{a}T$. Then

$$\begin{aligned} & (\mathbf{A}\, c, t : c \in \mathbf{a}S \cap \mathbf{a}T \wedge t \in (\mathbf{a}U \cup \mathbf{a}V)^* \wedge \gamma(t)c \in \mathbf{t}(S\, \mathbf{w}\, T) \\ & \qquad \wedge (\mathbf{A}\, s : s \leq t \restriction \mathbf{a}S : P(s)) \wedge (\mathbf{A}\, s : s \leq t \restriction \mathbf{a}T : Q(s)) \\ & \qquad : (\mathbf{E}\, m : m \in f(c) \cap g(c) : P(t\langle c, m \rangle) \wedge Q(t\langle c, m \rangle))) \\ \equiv\ & \langle S\, \mathbf{w}\, T, f \cup g, t : P(t \restriction \mathbf{a}S) \wedge Q(t \restriction \mathbf{a}T) \rangle \text{ is a split specification for } U\, \mathbf{w}\, V \end{aligned}$$

(End of Theorem)

Theorem 3.3.6

Let $\langle S, f, P \rangle$ and $\langle T, g, Q \rangle$ be split specifications specifying processes U and V, respectively. Let $f(c) = g(c)$ for all $c \in \mathbf{a}S \cap \mathbf{a}T$. Let $C \subseteq \mathbf{a}S$ and $D \subseteq \mathbf{a}T$ such that

$\mathbf{a}S \cap \mathbf{a}T \subseteq C \cup D$. If

$$(\mathbf{A}\, c,t : c \in C \wedge t \in \mathbf{a}U^* \wedge \gamma(t)c \in \mathbf{t}S \wedge (\mathbf{A}\, s : s \leq t : P(s)) \\ : (\mathbf{A}\, m : m \in f(c) : P(t\langle c,m\rangle)))$$

and

$$(\mathbf{A}\, d,t : d \in D \wedge t \in \mathbf{a}V^* \wedge \gamma(t)d \in \mathbf{t}T \wedge (\mathbf{A}\, s : s \leq t : Q(s)) \\ : (\mathbf{A}\, m : m \in g(d) : Q(t\langle d,m\rangle)))$$

then $\langle S\, \mathbf{w}\, T\, ,\, f \cup g\, ,\, t : P(t\restriction \mathbf{a}S) \wedge Q(t\restriction \mathbf{a}T)\rangle$ is a split specification specifying $U\, \mathbf{w}\, V$.

(End of Theorem)

Example 3.3.7

Let ADDER be the process in example 3.3.2. Let process COPY be specified by split specification

$$\langle \mathrm{PR}((d\,;\, a\,,\, b)^*)\, ,\, \{(d,\mathcal{Z}),(a,\mathcal{Z}),(b,\mathcal{Z})\}\, ,\, t : \mu(t\restriction a) \leq \mu(t\restriction d) \wedge \mu(t\restriction b) \leq \mu(t\restriction d)\rangle$$

The process DOUBLE = (COPY $\mathbf{w}$ ADDER)$\restriction\{d,c\}$ is then specified by

$$\langle \mathrm{PR}(d\,;\, c\,,\, d)^*)\, ,\, \{(c,\mathcal{Z}),(d,\mathcal{Z})\} \\ ,\, t : (\mathbf{A}\, i : 0 \leq i < \ell(t\restriction d)\, \mathbf{min}\, \ell(t\restriction c) : c(i) = 2 \cdot d(i))\rangle$$

(End of Example)

In sections 5.1 and 5.2 examples are given of how, under certain conditions, one easily derives a split specification for the projection. Such a derivation can also be given in the above example.

3.4 Properties of processes

In chapter 2 we introduced properties of processes expressing the absence of divergence or nondeterminism (or both), and the absence of deadlock. In this section we investigate the relationship between properties of a process T and properties of $\gamma(T)$, especially in the case that T is data independent.

In the sequel T is a process and C is a channel set such that $C \subseteq \mathbf{c}T$. We define AC to be

$$AC = (\cup\, c : c \in C : \{c\} \times type(c,T))$$

Observe that $AC \subseteq \mathbf{a}T$, $\gamma(AC) = C$, and $\gamma(\overline{AC}) = \overline{C}$.

Theorem 3.4.0

C is non-divergent with respect to $\gamma(T)$
$\Rightarrow$ AC is non-divergent with respect to T

Proof

Assume C is non-divergent with respect to $\gamma(T)$. Let $t \in \mathbf{t}T$. Let $n \geq 0$, be such that

$$(\mathbf{A}\, u : u \in (\overline{C})^* \wedge \gamma(t)u \in \mathbf{t}\gamma(T) : \ell(u) \leq n)$$

Let $s \in (\overline{AC})^*$. Then

$$\begin{array}{ll} & ts \in \mathbf{t}T \\ \Rightarrow & \{\text{calculus}\} \\ & \gamma(t)\gamma(s) \in \mathbf{t}\gamma(T) \\ \Rightarrow & \{\gamma(s) \in (\overline{C})^*\} \\ & \ell(\gamma(s)) \leq n \\ = & \{\text{property 3.1.0}\} \\ & \ell(s) \leq n \end{array}$$

(End of Proof)

The following example shows that the reverse implication does not hold in general.

Example 3.4.1

$T = \text{PREF}(\langle \{a, b, c\} \times \mathcal{Z}, \{\langle a, n\rangle\langle b, 0\rangle^n\langle c, 0\rangle \mid n \geq 0\}\rangle)$
$\{a, c\} \times \mathcal{Z}$ is non-divergent with respect to T
$\{a, c\}$ is divergent with respect to $\gamma(T)$
T is not data independent

(End of Example)

Provided T is data independent the reverse implication in theorem 3.4.0 does hold as the following theorem shows.

Theorem 3.4.2

Let T be data independent. Then

C is non-divergent with respect to $\gamma(T)$

$\equiv$ AC is non-divergent with respect to T

Proof

Assume that AC is non-divergent with respect to T. Let $u \in \mathbf{t}\gamma(T)$. Choose $t \in \mathbf{t}T$ such that $\gamma(t) = u$. Let $n \geq 0$ be such that

$$(\mathbf{A}\, s : s \in (\overline{AC})^* \wedge ts \in \mathbf{t}T : \ell(s) \leq n)$$

Let $v \in (\overline{C})^*$. We derive

$$\begin{array}{ll} & uv \in \mathbf{t}\gamma(T) \\ = & \quad \{ u = \gamma(t), \text{ definition } \mathit{after} \} \\ & v \in \mathbf{t}\mathit{after}(\gamma(t), \gamma(T)) \\ = & \quad \{ T \text{ is data independent} \} \\ & v \in \mathbf{t}\gamma(\mathit{after}(t, T)) \\ = & \quad \{ \text{calculus} \} \\ & (\mathbf{E}\, s : s \in \mathbf{t}\mathit{after}(t, T) : \gamma(s) = v) \\ = & \quad \{ v \in (\overline{C})^* \} \\ & (\mathbf{E}\, s : s \in \mathbf{t}\mathit{after}(t, T) : \gamma(s) = v \wedge \ell(s) \leq n) \\ \Rightarrow & \quad \{ \text{property 3.1.0} \} \\ & \ell(v) \leq n \end{array}$$

(End of Proof)

Corollary 3.4.3

Let S be a system such $(\mathbf{A}\, T : T \in \mathbf{p}S : T$ is data independent). If $\mathbf{W}(\mathbf{p}S)$ is data independent and $\gamma(\mathbf{W}(\mathbf{p}S)) = \mathbf{W}(\gamma(\mathbf{p}S))$ then

S is non-divergent $\equiv$ $\gamma(S)$ is non-divergent

(End of Corollary)

The following two examples show that we do not have theorems on non-disabling analogous to theorems 3.4.0 and 3.4.2.

Example 3.4.4

Let ADDER be the process defined in examples 3.2.0 and 3.3.2. We have that $\{a, c\}$ is non-disabling with respect to $\gamma(\text{ADDER}) = \text{PR}((a\,,\, b\,;\, c)^*)$. Since

$$\langle c,1\rangle \notin \mathbf{t}after(\langle a,0\rangle\langle b,0\rangle, \text{ADDER})$$

and

$$\langle c,1\rangle \in \mathbf{t}after(\langle a,0\rangle, \text{ADDER}\restriction\{a,c\})$$

we have that $\{a,c\} \times \mathcal{Z}$ is disabling with respect to ADDER.

(End of Example)

Example 3.4.5

Let T be the process specified by split specification

$$\langle\, \mathrm{PR}(\,(b\,;\,b)^*\,;\,(a\,;\,d \mid b\,;\,a\,;\,c)\,)\,,\ \{(a,\mathcal{Z})\} \cup (\{b,c,d\} \times \{\{\surd\}\})$$
$$,\ t : (\mathbf{A}\, s : sa \leq \gamma(t) : a(\ell(s\restriction a),t) = \ell(s\restriction b) \bmod 2\,)\,\rangle$$

Let $C = \{a,c,d\}$ and $AC = (\{a\} \times \mathcal{Z}) \cup (\{c,d\} \times \{\surd\})$. We have that AC is non-disabling with respect to T. Since $\mathbf{t}after(a,\gamma(T)) = \{d\}$ and $\mathbf{t}after(a,\gamma(T)\restriction C) = \{c,d\}$, we have that C is disabling with respect to $\gamma(T)$.

(End of Example)

Transparence of AC with respect to T implies transparence of C with respect to $\gamma(T)$ in case T is data independent as is shown by the following lemma and theorem.

Lemma 3.4.6

Let T be data independent. Let $\mathbf{c}T$ be finite. If AC is transparent with respect to T then

$$\begin{aligned}(\mathbf{A}\, s,t : s \in \mathbf{t}T \wedge t \in \mathbf{t}T \wedge \gamma(s)\restriction C = \gamma(t)\restriction C \\ : (\mathbf{E}\, u,v : u \in \mathbf{t}T \wedge v \in \mathbf{t}T \\ : \gamma(u) = \gamma(v) \wedge u\restriction C = s\restriction C \wedge v\restriction C = t\restriction C \\ \wedge\ suc(\gamma(u),\gamma(T)) \subseteq C \wedge suc(\gamma(v),\gamma(T)) \subseteq C\,)\,)\end{aligned}$$

Proof

Assume AC is transparent with respect to T. Since AC is non-divergent with respect to T we have that C is non-divergent with respect to $\gamma(T)$. Let $s \in \mathbf{t}T$ and $t \in \mathbf{t}T$ be such that $\gamma(s)\restriction C = \gamma(t)\restriction C$.
By theorem 2.1.7 we have that $\{w \mid w \in \mathbf{t}\gamma(T) \wedge w\restriction C \leq \gamma(t)\restriction C\}$ is finite. Observe that this set is nonempty. An algorithm to construct traces u and v is described by the following program.

$u, v, x, y, h := \varepsilon, \varepsilon, s\restriction C, t\restriction C, \varepsilon$
{ invariant : $u \in \mathbf{t}T \wedge v \in \mathbf{t}T \wedge \gamma(u) = \gamma(v) = h$
$\wedge\ (u\restriction C)x = s\restriction C \wedge (v\restriction C)y = t\restriction C,$
variant function :
$\ell(h)$ bounded by $(\mathbf{MAX}\, w : w \in \mathbf{t}\gamma(T) \wedge w\restriction C \leq \gamma(t)\restriction C : \ell(w))$ }
; **do** $(x \neq \varepsilon \wedge y \neq \varepsilon) \vee \neg(suc(h, \gamma(T)) \subseteq C)$
$\rightarrow$ **if** $\neg(suc(h, \gamma(T)) \subseteq C)$
$\rightarrow$ choose $d : d \in suc(h, \gamma(T)) \setminus C$
{ $d \notin C \wedge d \in \gamma(suc(u, T)) \wedge d \in \gamma(suc(v, T))$ }
; choose $m, n : m, n \in type(d, T) \wedge u\langle d, m\rangle \in \mathbf{t}T \wedge v\langle d, n\rangle \in \mathbf{t}T$
; $u, v, h := u\langle d, m\rangle, v\langle d, n\rangle, hd$
[] $suc(h, \gamma(T)) \subseteq C$
$\rightarrow$ { $suc(u, T) \subseteq AC \wedge suc(v, T) \subseteq AC \wedge x \neq \varepsilon \wedge y \neq \varepsilon$ }
{ $suc(u, T) = suc(u\restriction C, T\restriction C) \wedge suc(v, T) = suc(v\restriction C, T\restriction C)$
$\wedge\ x \neq \varepsilon \wedge y \neq \varepsilon$ }
choose $c, m, n, x_0, y_0 : c \in C \wedge m, n \in type(c, T)$
$\wedge\ x = \langle c, m\rangle x_0 \wedge y = \langle c, n\rangle y_0$
{ $\langle c, m\rangle \in suc(u, T) \wedge \langle c, n\rangle \in suc(v, T)$ }
; $u, v, x, y, h := u\langle c, m\rangle, v\langle c, n\rangle, x_0, y_0, hc$
fi
od
{ $u \in \mathbf{t}T \wedge v \in \mathbf{t}T \wedge \gamma(u) = \gamma(v) \wedge suc(\gamma(u), \gamma(T)) \subseteq C$
$\wedge\ suc(\gamma(v), \gamma(T)) \subseteq C \wedge u\restriction C = s\restriction C \wedge v\restriction C = t\restriction C$ }

(End of Proof)

Theorem 3.4.7

Let T be data independent. Let $\mathbf{c}T$ be finite. Then

AC is transparent with respect to T
$\Rightarrow$ C is transparent with respect to $\gamma(T)$

Proof

Assume that AC is transparent with respect to T. Then by theorem 2.1.20

$(\mathbf{A}\, t : t \in \mathbf{t}T \wedge suc(t, T) \subseteq AC : suc(t, T) = suc(t\restriction C, T\restriction C))$

and AC is non-divergent with respect to T. The latter is equivalent with channel set C is non-divergent with respect to $\gamma(T)$. We will show that

$$(\mathbf{A}\, w : w \in \mathbf{t}\gamma(T) \wedge suc(w, \gamma(T)) \subseteq C : suc(w, \gamma(T)) = suc(w\restriction C, \gamma(T)\restriction C))$$

Let $w \in \mathbf{t}\gamma(T)$ be such that $suc(w, \gamma(T)) \subseteq C$. Let $t \in \mathbf{t}T$ and $\gamma(t) = w$. By property 1.1.3 we have

$$suc(w\restriction C, \gamma(T)\restriction C) = (\cup\, s : s \in \mathbf{t}T \wedge \gamma(s)\restriction C = w\restriction C : suc(\gamma(s), \gamma(T)) \cap C)$$

Let $s \in \mathbf{t}T$ be such that $\gamma(s)\restriction C = w\restriction C$. Using lemma 3.4.6 choose $u \in \mathbf{t}T$ and $v \in \mathbf{t}T$ such that $\gamma(u) = \gamma(v)$, $suc(\gamma(u), \gamma(T)) \subseteq C$, $suc(\gamma(v), \gamma(T)) \subseteq C$, $u\restriction C = s\restriction C$, and $v\restriction C = t\restriction C$. We derive

$$\begin{array}{ll}
 & suc(\gamma(s), \gamma(T)) \cap C \\
= & \quad \{\, T \text{ is data independent, calculus} \,\} \\
 & \gamma(suc(s, T) \cap AC) \\
\subseteq & \quad \{\, \text{property 1.1.3} \,\} \\
 & \gamma(suc(s\restriction C, T\restriction C)) \\
= & \quad \{\, s\restriction C = u\restriction C,\ suc(u, T) \subseteq AC, \text{ theorem 2.1.20} \,\} \\
 & \gamma(suc(u, T)) \\
= & \quad \{\, T \text{ is data independent, } \gamma(u) = \gamma(v) \,\} \\
 & \gamma(suc(v, T)) \\
= & \quad \{\, t\restriction C = v\restriction C,\ suc(v, T) \subseteq AC, \text{ theorem 2.1.20} \,\} \\
 & \gamma(suc(t\restriction C, T\restriction C)) \\
= & \quad \{\, suc(t, T) \subseteq AC, \text{ theorem 2.1.20} \,\} \\
 & \gamma(suc(t, T)) \\
= & \quad \{\, T \text{ is data independent, } \gamma(t) = w \,\} \\
 & suc(w, \gamma(T))
\end{array}$$

(End of Proof)

Example 3.4.4 shows that the reverse implication does not hold in general.

We now focus on (non-)termination and absence of deadlock.

Lemma 3.4.8

Let T be a data independent process. Then

$$(\mathbf{A}\, t : t \in \mathbf{t}T : suc(t,T) = \varnothing \;\equiv\; suc(\gamma(t),\gamma(T)) = \varnothing\,)$$

Proof

Let $t \in \mathbf{t}T$. We derive

$$\begin{array}{ll} & suc(t,T) = \varnothing \\ = & \quad \{\text{ definition } \gamma\,\} \\ & \gamma(suc(t,T)) = \varnothing \\ = & \quad \{\, T \text{ is data independent}\,\} \\ & suc(\gamma(t),\gamma(T)) = \varnothing \end{array}$$

(End of Proof)

Theorem 3.4.9

Let T be a data independent process.

0 $(\mathbf{A}\, t : t \in \mathbf{t}T : T \text{ has terminated after } t \;\equiv\; \gamma(T) \text{ has terminated after } \gamma(t)\,)$

1 T is non-terminating $\equiv$ $\gamma(T)$ is non-terminating

(End of Theorem)

Theorem 3.4.10

Let X be a set of data independent processes. If $\gamma(\mathbf{W}(X)) = \mathbf{W}(\gamma(X))$ then

$$lockfree(X) \;\Rightarrow\; lockfree(\gamma(X))$$

Proof

Assume $\gamma(\mathbf{W}(X)) = \mathbf{W}(\gamma(X))$ and $lockfree(X)$. Let $t \in \mathbf{t}\mathbf{W}(\gamma(X))$.
Since $\gamma(\mathbf{W}(X)) = \mathbf{W}(\gamma(X))$ we choose $s \in \mathbf{t}\mathbf{W}(X)$ such that $\gamma(s) = t$. We derive

$$\begin{array}{ll} & suc(t, \mathbf{W}(\gamma(X))) = \varnothing \\ = & \quad \{\, \gamma(s) = t,\ \gamma(\mathbf{W}(X)) = \mathbf{W}(\gamma(X))\,\} \\ & suc(\gamma(s), \gamma(\mathbf{W}(X))) = \varnothing \\ \Rightarrow & \quad \{\, \text{property } 3.1.0\,\} \\ & suc(s, \mathbf{W}(X)) = \varnothing \end{array}$$

$$
\begin{array}{ll}
= & \{\, \mathit{lockfree}(X) \,\} \\
 & (\mathbf{A}\, T : T \in X : \mathit{suc}(s \restriction \mathbf{c}T, T) = \varnothing) \\
= & \{\, (\mathbf{A}\, T : T \in X : T \text{ is data independent}),\ \gamma(s) = t,\ \text{lemma } 3.4.8 \,\} \\
 & (\mathbf{A}\, T : T \in X : \mathit{suc}(t \restriction \mathbf{c}T, \gamma(T)) = \varnothing)
\end{array}
$$

(End of Proof)

The following example shows that the reverse implication in the above theorem does not hold in general.

Example 3.4.11

$T = \mathrm{PREF}(\langle \{a, b\} \times \mathcal{Z}\,,\ \{\langle a, 0\rangle\langle b, 0\rangle, \langle a, 1\rangle\langle b, 1\rangle\}\rangle)$
$U = \mathrm{PREF}(\langle \{b\} \times \mathcal{Z}\,,\ \{\langle b, 1\rangle\}\rangle)$
T and U are data independent
$\gamma(T \,\mathbf{w}\, U) = \mathrm{PR}(a\,;\, b) = \gamma(T) \,\mathbf{w}\, \gamma(U)$
$\neg \mathit{lockfree}(\{T, U\})$
$\mathit{lockfree}(\{\gamma(T), \gamma(U)\})$
$T \,\mathbf{w}\, U$ is not data independent

(End of Example)

Theorem 3.4.12

Let X be a set of data independent processes. If $\mathbf{W}(X)$ is data independent and $\gamma(\mathbf{W}(X)) = \mathbf{W}(\gamma(X))$ then

$$\mathit{lockfree}(X) \;\equiv\; \mathit{lockfree}(\gamma(X))$$

Proof

By theorem 3.4.10 we have $\mathit{lockfree}(X) \;\Rightarrow\; \mathit{lockfree}(\gamma(X))$. Assume $\mathit{lockfree}(\gamma(X))$. Let $t \in \mathbf{t}\mathbf{W}(X)$. We derive

$$
\begin{array}{ll}
 & \mathit{suc}(t, \mathbf{W}(X)) = \varnothing \\
= & \{\, \mathbf{W}(X) \text{ is data independent, lemma } 3.4.8 \,\} \\
 & \mathit{suc}(\gamma(t), \gamma(\mathbf{W}(X))) = \varnothing \\
= & \{\, \gamma(\mathbf{W}(X)) = \mathbf{W}(\gamma(X)) \,\} \\
 & \mathit{suc}(\gamma(t), \mathbf{W}(\gamma(X))) = \varnothing \\
= & \{\, \mathit{lockfree}(\gamma(X)) \,\}
\end{array}
$$

$$
\begin{array}{ll}
 & (\mathbf{A}\, T : T \in X : suc(\gamma(t)\restriction \mathbf{c}T, \gamma(T)) = \varnothing\,) \\
= & \quad \{\, (\mathbf{A}\, T : T \in X : T \text{ is data independent}\,),\ \text{lemma } 3.4.8\,\} \\
 & (\mathbf{A}\, T : T \in X : suc(t\restriction \mathbf{c}T, T) = \varnothing\,)
\end{array}
$$

(End of Proof)

Corollary 3.4.13

Let S be a system such that $(\mathbf{A}\, T : T \in \mathbf{p}S : T \text{ is data independent})$. If $\mathbf{W}(\mathbf{p}S)$ is data independent, and $\gamma(\mathbf{W}(\mathbf{p}S)) = \mathbf{W}(\gamma(\mathbf{p}S))$ then

$$S \text{ is lockfree} \equiv \gamma(S) \text{ is lockfree}$$

(End of Corollary)

3.5 Channel order independence

In section 3.2 we introduced the notion of data independence expressing that the communication behaviour of a process does not depend on the messages that are being transmitted. In this section we introduce the notion of channel order independence that expresses that at any moment the future behaviour of a process does not depend on the order in which the channels were used by the process. Formally, process T is said to be *channel order independent* if

$$
\begin{array}{l}
(\mathbf{A}\, t, a, b : ta \in \mathbf{t}T \wedge tb \in \mathbf{t}T \wedge \gamma(a) \neq \gamma(b) \\
\qquad\qquad : tab \in \mathbf{t}T \wedge tba \in \mathbf{t}T \wedge [tab] = [tba]\,)
\end{array}
$$

Observe the close resemblance to the definition of a conservative process. Conservativity expresses that at any moment the future behaviour of a process does not depend on the order in which the events have taken place. The following examples are to illustrate that channel order independence is a useful notion for the processes introduced in this chapter whereas conservativity is a notion more apt for the communication behaviours of these processes.

Example 3.5.0

Consider the process ADDER from example 3.2.0. This process is both data independent and channel order independent. Observe that the communications via channel a and b may take place simultaneously. If ADDER were to be conservative, it should contain traces like $\langle a,0\rangle\langle a,1\rangle \ldots \langle a,n\rangle$ $(n \geq 0)$. Clearly, this is not the case. Process $\gamma(T)$, on the other hand, is conservative.

(End of Example)

Example 3.5.1

Consider data independent process VAR as defined in section 3.0. Since $\langle a,0\rangle\langle a,1\rangle \in$ tVAR and $\langle a,0\rangle\langle b,0\rangle \in$ tVAR, but $\langle a,0\rangle\langle a,1\rangle\langle b,0\rangle \notin$ tVAR, process VAR is not channel order independent. Obviously, communications via channels a and b can not take place simultaneously. Observe that γ(VAR) is conservative.

(End of Example)

Process ADDER from example 3.5.0 can be specified by

$$\langle\, \mathrm{PR}((a\,,\, b;\, c)^*),$$
$$t : (\mathbf{A}\, i : 0 \le i < \ell(t\restriction a)\, \mathbf{min}\, \ell(t\restriction b)\, \mathbf{min}\, \ell(t\restriction c) : c(i) = a(i) + b(i))\,\rangle$$

Predicate $c(i) = a(i) + b(i)$ does not depend on the order of events in trace t. This is due to the fact that process ADDER is channel order independent. Process VAR from example 3.5.1 is not channel order independent and can be specified by

$$\langle\, \mathrm{PR}(a\,;\, (a \mid b)^*),$$
$$t : (\mathbf{A}\, i : 0 \le i < \ell(t\restriction b) \wedge f(i,t) \ge 0 : b(i) = a(f(i,t)))\,\rangle$$

where for $t \in \{a,b\}^*$ and $0 \le i < \ell(t\restriction b)$

$$f(i,t) = (\mathbf{MIN}\, s : s \le t \wedge \ell(s\restriction b) = i+1 : \ell(s\restriction a) - 1\,)$$

Notice that predicate $b(i) = a(f(i,t))$ depends on the order of a's and b's in trace t.

In general, specifications of data independent processes that are channel order independent as well resemble the above specification of process ADDER. In section 5.1 we introduce a new notation for specifications of data independent processes that are channel order independent. It is, to a large extent, based on the observations made above.

The following theorem shows that data independence and channel order independence of a process imply that the process describing its communication behaviour is conservative.

Theorem 3.5.2

If T is a data independent and channel order independent process, then $\gamma(T)$ is conservative.

Proof

Assume T is data independent and channel order independent. Let $t \in \mathbf{t}T$, $c \in \mathbf{c}T$, and $d \in \mathbf{c}T$ such that $\gamma(t)c \in \mathbf{t}\gamma(T)$, $\gamma(t)d \in \mathbf{t}\gamma(T)$, and $c \neq d$. We derive

$$
\begin{array}{ll}
& \gamma(t)c \in \mathbf{t}\gamma(T) \wedge \gamma(t)d \in \mathbf{t}\gamma(T) \\
= & \quad \{ \text{definition successor set, } T \text{ is data independent} \} \\
& c \in \gamma(suc(t,T)) \wedge d \in \gamma(suc(t,T)) \\
= & \quad \{ \text{definition } \gamma, \text{ definition successor set} \} \\
& (\mathbf{E}\, m,n : m \in type(c,T) \wedge n \in type(d,T) : t\langle c,m\rangle \in \mathbf{t}T \wedge t\langle d,n\rangle \in \mathbf{t}T) \\
\Rightarrow & \quad \{ c \neq d, \ T \text{ is channel order independent} \} \\
& (\mathbf{E}\, m,n : m \in type(c,T) \wedge n \in type(d,T) \\
& \qquad : t\langle c,m\rangle\langle d,n\rangle \in \mathbf{t}T \wedge t\langle d,n\rangle\langle c,m\rangle \in \mathbf{t}T \\
& \qquad \wedge\ after(t\langle c,m\rangle\langle d,n\rangle, T) = after(t\langle d,n\rangle\langle c,m\rangle, T)) \\
\Rightarrow & \quad \{ \text{definition } \gamma, \ T \text{ is data independent} \} \\
& \gamma(t)cd \in \mathbf{t}\gamma(T) \wedge \gamma(t)dc \in \mathbf{t}\gamma(T) \wedge after(\gamma(t)cd, T) = after(\gamma(t)dc, T))
\end{array}
$$

(End of Proof)

The reverse does not hold as is shown in the following example.

Example 3.5.3

$T = \text{PREF}(\langle \{a,b\} \times \mathcal{Z}, \{\langle a,0\rangle\langle b,1\rangle, \langle b,0\rangle\langle a,1\rangle\}\rangle)$
T is data independent
$\gamma(T)$ is conservative
T is not channel order independent

$U = \text{PREF}(\langle \{a,b,c\} \times \mathcal{Z}, \{\langle a,0\rangle\langle a,1\rangle\langle b,0\rangle\langle c,0\rangle, \langle a,1\rangle\langle a,0\rangle\langle c,1\rangle\langle b,1\rangle\}\rangle)$
U is not data independent
U is channel order independent
$\gamma(U)$ is conservative

(End of Example)

Theorem 3.5.4

Let T and U be channel order independent processes. Then $T \,\mathbf{w}\, U$ is channel order independent.

Proof

$$
\begin{array}{ll}
& ta \in \mathbf{t}(T \,\mathbf{w}\, U) \wedge tb \in \mathbf{t}(T \,\mathbf{w}\, U) \wedge \gamma(a) \neq \gamma(b) \\
= & \quad \{ \text{definition weave} \}
\end{array}
$$

$$
\begin{array}{ll}
& tab \in (\mathbf{a}T \cup \mathbf{a}U)^* \wedge ta\restriction\mathbf{a}T \in \mathbf{t}T \wedge tb\restriction\mathbf{a}T \in \mathbf{t}T \\
& \wedge\ ta\restriction\mathbf{a}U \in \mathbf{t}U \wedge tb\restriction\mathbf{a}U \in \mathbf{t}U \wedge \gamma(a) \neq \gamma(b) \\
\Rightarrow & \quad \{\, T \text{ and } U \text{ are channel order independent} \,\} \\
& tab \in (\mathbf{a}T \cup \mathbf{a}U)^* \\
& \wedge\ tab\restriction\mathbf{a}T \in \mathbf{t}T \wedge tba\restriction\mathbf{a}T \in \mathbf{t}T \wedge after(tab\restriction\mathbf{a}T, T) = after(tba\restriction\mathbf{a}T, T) \\
& \wedge\ tab\restriction\mathbf{a}U \in \mathbf{t}U \wedge tba\restriction\mathbf{a}U \in \mathbf{t}U \wedge after(tab\restriction\mathbf{a}U, U) = after(tba\restriction\mathbf{a}U, U) \\
\Rightarrow & \quad \{\, \text{definition weave, theorem 1.1.13} \,\} \\
& tab \in \mathbf{t}(T\,\mathbf{w}\,U) \wedge tba \in \mathbf{t}(T\,\mathbf{w}\,U) \wedge after(tab, T\,\mathbf{w}\,U) = after(tba, T\,\mathbf{w}\,U)
\end{array}
$$

(End of Proof)

Theorem 3.5.5

Let T be a channel order independent process.
Let $C \subseteq \mathbf{c}T$ and $AC = (\cup c : c \in C : \{c\} \times type(c, T))$.
If AC is transparent with respect to T then $T\restriction C$ is channel order independent.

Proof

Assume AC is transparent with respect to T. Let $t \in \mathbf{t}T\restriction C$. Choose $s \in \mathbf{t}T$ such that $s\restriction C = t$ and $suc(s, T) = suc(t, T\restriction C)$ (AC is transparent with respect to T, property 2.1.19). We derive

$$
\begin{array}{ll}
& ta \in \mathbf{t}T\restriction C \wedge tb \in \mathbf{t}T\restriction C \wedge \gamma(a) \neq \gamma(b) \\
= & \quad \{\, suc(s, T) = suc(t, T\restriction C) \,\} \\
& sa \in \mathbf{t}T \wedge sb \in \mathbf{t}T \wedge \gamma(a) \neq \gamma(b) \wedge \gamma(a) \in C \wedge \gamma(b) \in C \\
\Rightarrow & \quad \{\, T \text{ is channel order independent} \,\} \\
& sab \in \mathbf{t}T \wedge sba \in \mathbf{t}T \wedge after(sab, T) = after(sba, T) \wedge \gamma(a) \in C \wedge \gamma(b) \in C \\
\Rightarrow & \quad \{\, \text{calculus} \,\} \\
& sab\restriction C \in \mathbf{t}T\restriction C \wedge sba\restriction C \in \mathbf{t}T\restriction C \\
& \wedge\ after(sab, T)\restriction C = after(sba, T)\restriction C \wedge \gamma(a) \in C \wedge \gamma(b) \in C \\
= & \quad \{\, AC \text{ is non-disabling with respect to } T,\ s\restriction C = t \,\} \\
& tab \in \mathbf{t}T\restriction C \wedge tba \in \mathbf{t}T\restriction C \wedge after(tab, T\restriction C) = after(tba, T\restriction C)
\end{array}
$$

(End of Proof)

The following example shows that the condition "AC is transparent with respect to T" in the above theorem may not be replaced by "C is transparent with respect to $\gamma(T)$" even if T is data independent.

Example 3.5.6

$T = \mathrm{PREF}(\langle \{a,b,c\} \times \mathcal{Z}$
$, \{\langle a,1\rangle\langle b,1\rangle\langle c,0\rangle , \langle a,1\rangle\langle c,0\rangle\langle b,1\rangle$
$, \langle a,0\rangle\langle b,0\rangle\langle c,1\rangle , \langle a,0\rangle\langle c,1\rangle\langle b,0\rangle\} \rangle)$

T is channel order independent

T is data independent

$\{b,c\}$ is transparent with respect to $\gamma(T)$

$\{b,c\} \times \mathcal{Z}$ is not transparent with respect to T

$T\restriction\{b,c\}$ is not channel order independent

(End of Example)

Theorem 3.5.7

Let $(T_n)_{n\geq 0}$ be a sequence of channel order independent processes such that $(\mathbf{A}\, n : n \geq 0 : T_n \subseteq T_{n+1})$. Then $(\cup n : n \geq 0 : T_n)$ is channel order independent.

(End of Theorem)

4 Programs

4.0 Introduction

Consider process ADDER in example 3.2.0. This process describes a mechanism that repeatedly receives two integer values via its input channels and sends the sum of these values via its output channel. In the program notation to be presented in this chapter this process can be specified by the following component or program

com *adder*(**in** $a, b : int$, **out** $c : int$) :
    var $x, y, z : int$ **rav**
    $(a?x\ ,\ b?y\ ;\ z{:=}\ x + y\ ;\ c!z)^*$
moc

Here, x,y, and z are internal variables of the component, $a?x$ denotes the receiving of a value via channel a and the assignment of that value to variable x, and $c!z$ denotes the sending of the value of variable z via channel c. These notations have been adopted from CSP (see [Ho]). Variable z can be eliminated from the above program: one may replace "$z{:=}\ x + y\ ;\ c!z$" by "$c!(x + y)$". Notice that all channels and variables have a type (in this case they are all of type integer).

In this chapter we introduce a program notation generalizing the one presented in section 1.4. The above is an example of a program text. We will, however, not explicitly mention types in program texts when defining the program notation, but we will assume all channels and variables to be of the same type, namely M, the set of messages. This is done for simplicity's sake. Generalization to the case where the types of channels and variables may differ is rather straightforward.

4.1 Commands

In this section we extend the definition of a command from section 1.2. First we introduce some notions that shall be used in the definition of commands.

We assume the existence of a set of (names of) variables, denoted by VAR. The set of expressions EXP is defined to be the smallest set satisfying

- $m \in \mathrm{EXP}$ $\quad m \in \mathrm{M}$
- $x \in \mathrm{EXP}$ $\quad x \in \mathrm{VAR}$
- $\psi(e_0, e_1, \ldots, e_{l-1}) \in \mathrm{EXP}$ $\quad l \geq 0,\ \psi \in \mathrm{M}^l \rightarrow \mathrm{M},\ e_0, e_1, \ldots, e_{l-1} \in \mathrm{EXP}$

The function $var : \mathrm{EXP} \rightarrow \mathcal{P}(\mathrm{VAR})$ is defined recursively by

$$\begin{aligned} var(m) &= \varnothing && m \in \mathrm{M} \\ var(x) &= \{x\} && x \in \mathrm{VAR} \\ var(\psi(e_0, e_1, \ldots, e_{l-1})) &= (\cup i : 0 \leq i < l : var(e_i)) \end{aligned}$$

Notice that for all expressions e such that $var(e) = \varnothing$ we have that for all expressions e_s appearing in e $var(e_s) = \varnothing$ holds.
The function $val : \{ e \mid e \in \mathrm{EXP} \wedge var(e) = \varnothing \} \rightarrow \mathrm{M}$ is defined recursively by

$$\begin{aligned} val(m) &= m \\ val(\psi(e_0, e_1, \ldots, e_{l-1})) &= \psi(val(e_0), val(e_1), \ldots, val(e_{l-1})) \\ & \qquad var(e_1) = var(e_1) = \cdots = var(e_{l-1}) = \varnothing \end{aligned}$$

Let $\phi \in \mathrm{VAR} \rightarrow \mathrm{EXP}$. We will represent function ϕ with the set $\{ (x, \phi(x)) \mid x \in \mathrm{VAR} \wedge x \neq \phi(x) \}$. With function ϕ we associate substitution function $S_\phi \in \mathrm{EXP} \rightarrow \mathrm{EXP}$ defined recursively by

$$\begin{aligned} S_\phi(m) &= m \\ S_\phi(x) &= \phi(x) \\ S_\phi(\psi(e_0, e_1, \ldots, e_{l-1})) &= \psi(S_\phi(e_0), S_\phi(e_1), \ldots, S_\phi(e_{l-1})) \end{aligned}$$

Let $\phi, \chi \in \mathrm{VAR} \rightarrow \mathrm{EXP}$. Function $\phi \odot \chi \in \mathrm{VAR} \rightarrow \mathrm{EXP}$, the composition of ϕ and χ, is defined by

$$(\phi \odot \chi)(x) = S_\phi(\chi(x))$$

Since $\surd \notin \mathrm{M}$ we also have $\surd \notin \mathrm{EXP}$.
The functions var, val, and S_ϕ are generalized to functions having their domain in $(\Gamma \times (\mathrm{EXP} \cup \{\surd\}))^*$. The function $var : (\Gamma \times (\mathrm{EXP} \cup \{\surd\}))^* \rightarrow \mathcal{P}(\mathrm{VAR})$ is defined by

$$\begin{aligned} var(\varepsilon) &= \varnothing \\ var(t\langle c, e\rangle) &= var(t) \cup var(e) \\ var(t\langle c, \surd\rangle) &= var(t) \end{aligned}$$

The function $val : \{ t \mid (\Gamma \times (\mathrm{EXP} \cup \{\surd\}))^* \wedge var(t) = \varnothing \} \rightarrow \Omega^*$ $(= (\Gamma \times \mathrm{M}_\surd)^*)$ is defined by

$$val(\varepsilon) = \varepsilon$$
$$val(t\langle c, e\rangle) = val(t)\langle c, val(e)\rangle$$
$$val(t\langle c, \surd\rangle) = val(t)\langle c, \surd\rangle$$

Let $\phi \in \mathrm{VAR} \to \mathrm{EXP}$. The function $S_\phi \in (\Gamma \times (\mathrm{EXP} \cup \{\surd\}))^* \to (\Gamma \times (\mathrm{EXP} \cup \{\surd\}))^*$ is defined by

$$S_\phi(\varepsilon) = \varepsilon$$
$$S_\phi(t\langle c, e\rangle) = S_\phi(t)\langle c, S_\phi(e)\rangle$$
$$S_\phi(t\langle c, \surd\rangle) = S_\phi(t)\langle c, \surd\rangle$$

A *command structure* is a quadruple $\langle C, D, E, X\rangle$ where C, D, and E are channel sets and $X \subseteq ((C \times \{\surd\}) \cup ((D \cup E) \times \mathrm{EXP}))^* \times (\mathrm{VAR} \to \mathrm{EXP})$. We call C the set of signals of the command structure, D its set of input channels, E its set of output channels, and X its extended trace set. Elements of X are called extended traces. Notice that extended traces are pairs consisting of a trace and a substitution function.

Let T be a command structure. The set of signals of T is denoted by $\mathbf{s}T$, the set of input channels of T by $\mathbf{i}T$, the set of output channels of T by $\mathbf{o}T$, and the extended trace set of T by $\mathbf{et}T$. Furthermore we define

$$var(T) = (\cup t, \phi : (t, \phi) \in \mathbf{et}T : var(t))$$

We now introduce *commands*. With each command S we associate a command structure $\mathrm{CO}(S)$ and a set of variables $bind(S)$. Commands, associated command structures, and associated sets of variables are defined inductively by the following rules (remember our convention in representing functions in $\mathrm{VAR} \to \mathrm{EXP}$).

- ε is a command
  $\mathrm{CO}(\varepsilon) = \langle \varnothing, \varnothing, \varnothing, \{(\varepsilon, \varnothing)\}\rangle$
  $bind(\varepsilon) = \varnothing$
- c is a command for all $c \in \Gamma$
  $\mathrm{CO}(c) = \langle \{c\}, \varnothing, \varnothing, \{(\langle c, \surd\rangle, \varnothing)\}\rangle$
  $bind(c) = \varnothing$
- $c?x$ is a command for all $c \in \Gamma$ and $x \in \mathrm{VAR}$
  $\mathrm{CO}(c?x) = \langle \varnothing, \{c\}, \varnothing, \{(\langle c, m\rangle, \{(x, m)\}) \mid m \in \mathrm{M}\}\rangle$
  $bind(c?x) = \{x\}$
- $c!e$ is a command for all $c \in \Gamma$ and $e \in \mathrm{EXP}$
  $\mathrm{CO}(c!e) = \langle \varnothing, \varnothing, \{c\}, \{(\langle c, e\rangle, \varnothing)\}\rangle$
  $bind(c!e) = \varnothing$

- $x_0, x_1, \ldots, x_{m-1} := e_0, e_1, \ldots, e_{m-1}$ is a command
  for all $m > 0$, $x_0, x_1, \ldots, x_{m-1} \in \mathrm{VAR}$, $e_0, e_1, \ldots, e_{m-1} \in \mathrm{EXP}$
  $\mathrm{CO}(x_0, x_1, \ldots, x_{m-1} := e_0, e_1, \ldots, e_{m-1})$
  $= \langle \varnothing, \varnothing, \varnothing, \{(\varepsilon, \{ (x_i, e_i) \mid 0 \le i < m \wedge x_i \neq e_i \})\} \rangle$
  $\mathit{bind}(x_0, x_1, \ldots, x_{m-1} := e_0, e_1, \ldots, e_{m-1})$
  $= \{ x_i \mid 0 \le i < m \} \setminus (\cup i : 0 \le i < m : \mathit{var}(e_i))$
- $S \mid T$ is a command for all commands S and T such that
  $\mathbf{sCO}(S) \cup \mathbf{sCO}(T)$, $\mathbf{iCO}(S) \cup \mathbf{iCO}(T)$, and $\mathbf{oCO}(S) \cup \mathbf{oCO}(T)$ are disjoint sets
  $\mathrm{CO}(S \mid T) = \langle \mathbf{sCO}(S) \cup \mathbf{sCO}(T), \mathbf{iCO}(S) \cup \mathbf{iCO}(T)$
  $, \mathbf{oCO}(S) \cup \mathbf{oCO}(T), \mathbf{etCO}(S) \cup \mathbf{etCO}(T) \rangle$
  $\mathit{bind}(S \mid T) = \mathit{bind}(S) \cup \mathit{bind}(T)$
- $S ; T$ is a command for all commands S and T such that
  $\mathbf{sCO}(S) \cup \mathbf{sCO}(T)$, $\mathbf{iCO}(S) \cup \mathbf{iCO}(T)$, and $\mathbf{oCO}(S) \cup \mathbf{oCO}(T)$ are disjoint sets
  $\mathrm{CO}(S ; T) = \langle \mathbf{sCO}(S) \cup \mathbf{sCO}(T), \mathbf{iCO}(S) \cup \mathbf{iCO}(T), \mathbf{oCO}(S) \cup \mathbf{oCO}(T),$
  $\{ (s\, S_\phi(t), \phi \odot \chi) \mid (s, \phi) \in \mathbf{etCO}(S) \wedge (t, \chi) \in \mathbf{etCO}(T) \} \rangle$
  $\mathit{bind}(S ; T) = \mathit{bind}(S) \cup \mathit{bind}(T)$
- S , T is a command for all commands S and T such that
  $\mathbf{sCO}(S)$, $\mathbf{sCO}(T)$, $\mathbf{iCO}(S)$, $\mathbf{iCO}(T)$, $\mathbf{oCO}(S)$, and $\mathbf{oCO}(T)$ are disjoint sets,
  $\mathit{bind}(S) \cap (\mathit{bind}(T) \cup \mathit{var}(\mathrm{CO}(T))) = \varnothing$,
  and $\mathit{bind}(T) \cap (\mathit{bind}(S) \cup \mathit{var}(\mathrm{CO}(S))) = \varnothing$
  $\mathrm{CO}(S , T) = \langle \mathbf{sCO}(S) \cup \mathbf{sCO}(T), \mathbf{iCO}(S) \cup \mathbf{iCO}(T), \mathbf{oCO}(S) \cup \mathbf{oCO}(T),$
  $\{ (t, \phi \cup \chi) \mid t \in (A_S \cup A_T)^* \wedge (t \restriction A_S, \phi) \in \mathbf{etCO}(S)$
  $\wedge (t \restriction A_T, \chi) \in \mathbf{etCO}(T) \} \rangle$
  where $A_X = (\mathbf{sCO}(X) \times \{\surd\}) \cup ((\mathbf{iCO}(X) \cup \mathbf{oCO}(X)) \times \mathrm{EXP})$
  for $X = S$ and $X = T$
  $\mathit{bind}(S , T) = \mathit{bind}(S) \cup \mathit{bind}(T)$
- S^0 is a command for all commands S
  $\mathrm{CO}(S^0) = \langle \mathbf{sCO}(S), \mathbf{iCO}(S), \mathbf{oCO}(S), \{(\varepsilon, \varnothing)\} \rangle$
  $\mathit{bind}(S^0) = \varnothing$
- S^* is a command for all commands S
  $\mathrm{CO}(S^* = \langle \mathbf{sCO}(S), \mathbf{iCO}(S), \mathbf{oCO}(S), (\cup n : n \ge 0 : \mathbf{etCO}(S^n)) \rangle$
  where $S^{n+1} = S^n ; S \qquad (n \ge 0)$
  $\mathit{bind}(S^*) = \mathit{bind}(S)$

Command $c?x$ may be interpreted as the reception of a value via channel c and the assignment of that value to variable x. Command $c!e$ may be interpreted as the sending

of the value of expression e via channel c. Observe that the commands as defined in section 1.2 form a subclass of the commands defined above.

With each command S we associate a trace structure whose alphabet is a subset of $\Gamma \times (\mathrm{EXP} \cup \{\surd\})$. For command S trace structure $\mathrm{TR_E}(S)$ is defined by

$$\begin{aligned}\mathrm{TR_E}(S) = \langle\, &\mathbf{sCO}(S) \times \{\surd\} \cup (\mathbf{iCO}(S) \cup \mathbf{oCO}(S)) \times \mathrm{EXP} \\ &, \{\, t \mid (\mathbf{E}\,\phi : \phi \in \mathrm{VAR} \to \mathrm{EXP} : (t,\phi) \in \mathbf{etCO}(S)) \,\}\,\rangle\end{aligned}$$

Observe that for all commands S extended trace set $\mathbf{etCO}(S)$ is nonempty. Therefore, $\mathrm{PR_E}(S)$ defined by

$$\mathrm{PR_E}(S) = \mathrm{PREF}(\mathrm{TR_E}(S))$$

is a process.

A command S is called *closed* if $var(\mathrm{CO}(S)) = \varnothing$. Notice that the commands defined in section 1.2 are closed commands. With each closed command S we associate a trace structure $\mathrm{TR}(S)$ defined by

$$\begin{aligned}\mathrm{TR}(S) = \langle\, &\mathbf{sCO}(S) \times \{\surd\} \cup (\mathbf{iCO}(S) \cup \mathbf{oCO}(S)) \times \mathrm{M} \\ &, \{\, val(t) \mid (\mathbf{E}\,\phi : \phi \in \mathrm{VAR} \to \mathrm{EXP} : (t,\phi) \in \mathbf{etCO}(S)) \,\}\,\rangle\end{aligned}$$

Notice that $\mathbf{t}\mathrm{TR}(S) = \{\, val(t) \mid t \in \mathbf{t}\mathrm{TR_E}(S) \,\}$. For closed command S process $\mathrm{PR}(S)$ is defined by $\mathrm{PR}(S) = \mathrm{PREF}(\mathrm{TR}(S))$ and system $sys(S)$ is defined by $sys(S) = \langle \mathbf{c}\mathrm{PR}(S), \{\mathrm{PR}(S)\}\rangle$.

Example 4.1.0

Let $S = (a?x\,,\ b?y\,;\ z{:=}\ x + y\,;\ c!z)^*$. Then $var(S) = bind(S) = \{x, y, z\}$ and

$$\begin{aligned}&\mathrm{CO}(S) = \\ &\quad \langle\, \varnothing, \{a,b\}, \{c\} \\ &\qquad , (\{\, (\langle a,m\rangle\langle b,n\rangle\langle c, m+n\rangle, \{(x,m),(y,n),(z,m+n)\}) \mid m,n \in \mathcal{Z}\,\} \\ &\qquad\quad \cup \{\, (\langle b,n\rangle\langle a,m\rangle\langle c, m+n\rangle, \{(x,m),(y,n),(z,m+n)\}) \mid m,n \in \mathcal{Z}\,\})^* \,\rangle\end{aligned}$$

This leads to

$$\begin{aligned}\mathrm{TR}_E(S) = \langle\, &\{a,b,c\} \times \mathrm{EXP} \\ &, (\{\, \langle a,m\rangle\langle b,n\rangle\langle c, m+n\rangle \mid m,n \in \mathcal{Z}\,\} \\ &\quad \cup \{\, \langle b,n\rangle\langle a,m\rangle\langle c, m+n\rangle \mid m,n \in \mathcal{Z}\,\})^* \,\rangle\end{aligned}$$

and $\mathrm{PR}(S) = \mathrm{ADDER}$ where ADDER is the process defined in example 3.2.0.

(End of Example)

Observe that for a command S as defined in section 1.2 the definitions of $\mathrm{TR}(S)$, $\mathrm{PR}(S)$, and $sys(S)$ given here are equivalent to the definitions in section 1.2.
For command S command $\gamma(S)$ is defined inductively by

$$\begin{aligned}
&\gamma(\varepsilon) = \varepsilon \\
&\gamma(c) = c \\
&\gamma(c?x) = c \\
&\gamma(c!e) = c \\
&\gamma(x_0, x_1, \ldots, x_{m-1} := e_0, e_1, \ldots, e_{m-1}) = \varepsilon \\
&\gamma(S \mid T) = \gamma(S) \mid \gamma(T) \\
&\gamma(S\,;\, T) = \gamma(S)\,;\, \gamma(T) \\
&\gamma(S\,,\, T) = \gamma(S)\,,\, \gamma(T) \\
&\gamma(S^{\,0}) = \gamma(S)^0 \\
&\gamma(S^*) = \gamma(S)^*
\end{aligned}$$

Command $\gamma(S)$ reflects the communication pattern corresponding to command S.
Let T be a command structure. Command structure $\gamma(T)$ is defined by

$$\begin{aligned}
\gamma(T) = \langle\, &\mathbf{s}T \cup \mathbf{i}T \cup \mathbf{o}T, \varnothing, \varnothing, \\
&\{\, (\gamma(t), \varnothing) \mid (\mathbf{E}\,\phi : \phi \in \mathrm{VAR} \to \mathrm{EXP} : (t, \phi) \in \mathbf{et}T\,)\,\}\,\rangle
\end{aligned}$$

Property 4.1.1

Let S be a command.

0 $\gamma(S)$ is a closed command.

1 $\gamma(\mathrm{CO}(S)) = \mathrm{CO}(\gamma(S))$
$\gamma(\mathrm{TR}_{\mathrm{E}}(S)) = \mathrm{TR}(\gamma(S))$
$\gamma(\mathrm{PR}_{\mathrm{E}}(S)) = \mathrm{PR}(\gamma(S))$

2 If S is a closed command, then
$\gamma(\mathrm{TR}(S)) = \mathrm{TR}(\gamma(S))$
$\gamma(\mathrm{PR}(S)) = \mathrm{PR}(\gamma(S))$

(End of Property)

Analogously to the extension of the notion of commands we extend the notion of *restricted commands*. Restricted commands form a subset of the set of the commands introduced above and are defined inductively by the following rules (the conditions imposed by the operators given in the definition of commands remain valid but are omitted for clarity).

- ε is a restricted command
- c is a restricted command for all $c \in \Gamma$
- $c?x$ is a restricted command for all $c \in \Gamma$ and $x \in \mathrm{VAR}$
- $c!e$ is a restricted command for all $c \in \Gamma$ and $e \in \mathrm{EXP}$
- $x_0, x_1, \ldots, x_{m-1} := e_0, e_1, \ldots, e_{m-1}$ is a restricted command for all $m > 0, x_0, x_1, \ldots, x_{m-1} \in \mathrm{VAR}, e_0, e_1, \ldots, e_{m-1} \in \mathrm{EXP}$
- if S and T are restricted commands and S contains no stars then $S\,;\,T$ is a restricted command
- if S and T are restricted commands then $S\,,\,T$ is a restricted command
- if S is restricted command not containing any stars then S^0 and S^* are restricted commands

Property 4.1.2

If S is a restricted command then $\gamma(S)$ is a restricted command.

(End of Property)

Notice that the restricted commands as defined in section 1.2 form a subclass of the restricted commands defined above.

Property 4.1.3

Let S be a restricted command. If S contains no stars then for every channel set C

$$(\mathbf{A}\, s, t, \phi, \chi : (s, \phi) \in \mathbf{et}\mathrm{CO}(S) \wedge (t, \chi) \in \mathbf{et}\mathrm{CO}(S) : \ell(s \restriction C) = \ell(t \restriction C))$$

(End of Property)

We observe that theorem 2.4.10 does not hold for (closed) restricted commands as defined above (see example 3.5.0). In the sequel we will show that if S is a closed restricted command, then $\mathrm{PR}(S)$ is data independent and channel order independent. First, we have the following results.

Theorem 4.1.4

If S is a closed restricted command and $\mathrm{PR}_{\mathrm{E}}(S)$ is data independent, then $\mathrm{PR}(S)$ is data independent.

(End of Theorem)

The reverse does not hold as the following example shows.

Example 4.1.5

In this example we assume M = $\mathcal{Z}$. Let S be the command

$$b?x\,;\ (y{:=}\ x+x\,;\ a!y \mid y{:=}\ 2*x\,;\ a!y\,;\ c!x)$$

Then S is closed,

$$\begin{aligned}\mathrm{PR_E}(S) = \mathrm{PREF}(\langle\, &\{a,b,c\} \times \mathrm{EXP}\\ &,\{\,\langle b,m\rangle\langle a,m+m\rangle \mid m \in \mathcal{Z}\,\}\\ &\cup\{\,\langle b,m\rangle\langle a,2*m\rangle\langle c,m\rangle \mid m \in \mathcal{Z}\,\}\,\rangle)\end{aligned}$$

is not data independent ($m+m$ and $2*m$ are different expressions for all $m \in \mathcal{Z}$), and

$$\mathrm{PR}(S) = \mathrm{PREF}(\langle\, \{a,b,c\} \times \mathcal{Z}, \{\,\langle b,m\rangle\langle a,2\cdot m\rangle\langle c,m\rangle \mid m \in \mathcal{Z}\,\}\,\rangle)$$

is data independent.

(End of Example)

In theorem 4.1.4 we may not replace "data independent" by "channel order independent" as the following example shows.

Example 4.1.6

Let M = $\mathcal{Z}$. Let S be the command

$$x,y{:=}\ 0,1\,;\ (a!(x+y)\,;\ b!x\,,\ c!y \mid a!(y+x)\,;\ b!y\,,\ c!x)$$

Then S is closed,

$$\begin{aligned}\mathrm{PR_E}(S) = \mathrm{PREF}(\langle\, &\{a,b,c\} \times \mathrm{EXP}\\ &,\{\,\langle a,0+1\rangle\langle b,0\rangle\langle c,1\rangle\,,\ \langle a,0+1\rangle\langle c,1\rangle\langle b,0\rangle\\ &\quad,\ \langle a,1+0\rangle\langle b,1\rangle\langle c,0\rangle\,,\ \langle a,1+0\rangle\langle c,0\rangle\langle b,1\rangle\}\,\rangle)\end{aligned}$$

is both data independent and channel order independent, but

$$\begin{aligned}\mathrm{PR}(S) = \mathrm{PREF}(\langle\, &\{a,b,c\} \times \mathcal{Z}\\ &,\{\,\langle a,1\rangle\langle b,0\rangle\langle c,1\rangle\,,\ \langle a,1\rangle\langle c,1\rangle\langle b,0\rangle\\ &\quad,\ \langle a,1\rangle\langle b,1\rangle\langle c,0\rangle\,,\ \langle a,1\rangle\langle c,0\rangle\langle b,1\rangle\}\,\rangle)\end{aligned}$$

is not channel order independent.

(End of Example)

Theorem 4.1.7

0 If command S is equal to ε, c, $c?x$, $c?e$, or $x_0, x_1, \ldots, x_{m-1} := e_0, e_1, \ldots, e_{m-1}$ then $\mathrm{PR_E}(S)$ is both data independent and channel order independent.

1 If S and T are commands such that $S\,,\,T$ is a command, and $\mathrm{PR_E}(S)$ and $\mathrm{PR_E}(T)$ are data independent (channel order independent) then $\mathrm{PR_E}(S)$ is data independent (channel order independent).

2 If S and T are commands such that $S\,;\,T$ is a command, $\mathrm{PR_E}(S)$ and $\mathrm{PR_E}(T)$ are data independent (channel order independent), and $(\mathbf{E}\,n : n \geq 0 : (\mathbf{A}\,t : t \in \mathrm{tTR_E}(S) : \ell(t) = n\,))$ then $S\,;\,T$ is data independent (channel order independent).

3 If S is a command such that for n, $n \geq 0$, $\mathrm{PR_E}(S^n)$ is data independent (channel order independent), then $\mathrm{PR_E}(S^*)$ is data independent (channel order independent).

4 If S is a command, then $\mathrm{PR_E}(S^0)$ is data independent and channel order independent.

(End of Theorem)

Corollary 4.1.8

If S is a restricted command, then $\mathrm{PR_E}(S)$ is data independent and channel order independent.

(End of Corollary)

Combining theorem 4.1.4 and corollary 4.1.8 we have

Theorem 4.1.9

If S is a closed restricted command, then $\mathrm{PR}(S)$ is data independent.

(End of Theorem)

Lemma 4.1.10

Let S be a closed restricted command. Let $tu \in \mathrm{tPR}(S)$ and $tv \in \mathrm{tPR}(S)$. Then

$$(\mathbf{E}\,w, x, y : wx \in \mathrm{tPR_E}(S) \wedge wy \in \mathrm{tPR_E}(S) : val(wx) = tu \wedge val(wy) = tv\,)$$

(End of Lemma)

Using lemma 4.1.10 and corollary 4.1.8 one can prove

Theorem 4.1.11

If S is a closed restricted command, then PR(S) is channel order independent.

(End of Theorem)

From theorem 2.4.10 or theorem 3.5.2 it follows that

Theorem 4.1.12

If S is a closed restricted command, then γ(PR(S)) is cubic.

(End of Theorem)

Combining theorems 4.1.9, 4.1.12, 3.2.4, and 2.3.8 results in

Corollary 4.1.13

If S is a closed restricted command and C is a channel set such that $C \subseteq$ cPR(S), then PR(S)$\restriction C$ is data independent.

(End of Corollary)

This does not hold for channel order independence as the following example shows (cf. example 3.5.6).

Example 4.1.14

Let M = $\mathcal{Z}$. Let S be the closed restricted command

$$a?x\,;\; y{:=}\,1-x\,;\; (b!x\,,\, c!y)$$

We have that

$$\begin{aligned}\mathrm{PR}(S) = \mathrm{PREF}(\,\langle\, &\{a,b,c\} \times \mathcal{Z} \\ &,\{\,\langle a,m\rangle\langle b,m\rangle\langle c,1-m\rangle \mid m \in \mathcal{Z}\,\} \\ &\cup \{\,\langle a,m\rangle\langle c,1-m\rangle\langle b,m\rangle \mid m \in \mathcal{Z}\,\}\,\rangle\,)\end{aligned}$$

is channel order independent, but

$$\begin{aligned}\mathrm{PR}(S)\restriction\{b,c\} = \mathrm{PREF}(\,&\langle\,\{b,c\}\times\mathcal{Z}\\ &,\{\,\langle b,m\rangle\langle c,1-m\rangle \mid m\in\mathcal{Z}\,\}\\ &\cup\{\,\langle c,1-m\rangle\langle b,m\rangle \mid m\in\mathcal{Z}\,\}\,\rangle\,)\end{aligned}$$

is not channel order independent ($\langle b,1\rangle\langle c,1\rangle \notin \mathrm{tPR}(S)\restriction\{b,c\}$).

(End of Example)

Finally, we make some observations concerning possible extensions of the set of commands. Describing the filter mechanism in section 0.0 we already gave an example of a command containing an alternative statement:

$$(a?x\,;\ \mathbf{if}\ x\geq 0\rightarrow b!x[]\,x<0\rightarrow\varepsilon\ \mathbf{fi}\,)^*$$

The general form of such a command is

$$\mathbf{if}\ B_0\rightarrow S_0\,[]\,B_1\rightarrow S_1\,[]\ldots[]\,B_{N-1}\rightarrow S_{N-1}\ \mathbf{fi}$$

where B_0, B_1, ..., and B_{N-1} are boolean expressions and S_0, S_1, ..., and S_{N-1} are commands. They can be incorporated in the theory presented above by making extended traces triples consisting of a trace, a substitution function, and a boolean expression that is a conjunction of all guards that have to be satisfied to obtain the given trace. Likewise, one can introduce commands that contain a repetitive statement:

$$\mathbf{do}\ B_0\rightarrow S_0\,[]\,B_1\rightarrow S_1\,[]\ldots[]\,B_{N-1}\rightarrow S_{N-1}\ \mathbf{od}$$

where B_0, B_1, ..., and B_{N-1} are boolean expressions and S_0, S_1, ..., and S_{N-1} are commands. Both suggested extensions are generalizations since we have

$$\mathbf{if}\ \mathrm{true}\rightarrow S\,[]\,\mathrm{true}\rightarrow T\ \mathbf{fi} = S\mid T$$

and

$$\mathbf{do}\ \mathrm{true}\rightarrow S\ \mathbf{od} = S^*$$

We will not elaborate on these extensions any further since, in general, commands containing alternative or repetitive statements do not define data independent processes.

4.2 A program notation

In this section we generalize the program notation introduced in section 1.4. As before, a program or component defines a system. The process of a program is defined to be the process of the corresponding system.

We assume that with each channel set occurring in a process or a system a tripartition of that set is associated. The tripartition represents the distinction that is made between signals, input channels, and output channels. If C is a channel set, then $\mathbf{s}C$ denotes its set of signals, $\mathbf{i}C$ its set of input channels, and $\mathbf{o}C$ its set of output channels. Notice that this partition depends on the process or system in which C occurs. For a process T we abbreviate $\mathbf{s}(\mathbf{c}T)$ to $\mathbf{s}T$, $\mathbf{i}(\mathbf{c}T)$ to $\mathbf{i}T$, and $\mathbf{o}(\mathbf{c}T)$ to $\mathbf{o}T$.

Let S be a system. We define $\mathbf{s}S = \mathbf{s}(\mathbf{e}S)$, $\mathbf{i}S = \mathbf{i}(\mathbf{e}S)$, and $\mathbf{o}S = \mathbf{o}(\mathbf{e}S)$. Apart from the restrictions imposed on systems in the previous chapters we, furthermore, require that

$$(\mathbf{A}\, c : c \in \mathbf{cW}(\mathbf{p}S) : (\mathbf{N}\, T : T \in \mathbf{p}S : c \in \mathbf{o}T) \leq 1)$$
$$(\mathbf{A}\, T, U, c : T \in \mathbf{p}S \wedge U \in \mathbf{p}S \wedge c \in \mathbf{c}T \cap \mathbf{c}U : c \in \mathbf{s}T \equiv c \in \mathbf{s}U)$$
$$\mathbf{s}S = (\cup T : T \in \mathbf{p}S : \mathbf{s}T) \cap \mathbf{e}S$$
$$\mathbf{i}S = (\cup T : T \in \mathbf{p}S : \mathbf{i}T) \cap \mathbf{e}S$$
$$\mathbf{o}S = (\cup T : T \in \mathbf{p}S : \mathbf{o}T) \cap \mathbf{e}S$$

Let S be a command. We define $\mathbf{s}\mathrm{PR}_{\mathrm{E}}(S) = \mathbf{s}\mathrm{CO}(S)$, $\mathbf{i}\mathrm{PR}_{\mathrm{E}}(S) = \mathbf{i}\mathrm{CO}(S)$, and $\mathbf{o}\mathrm{PR}_{\mathrm{E}}(S) = \mathbf{o}\mathrm{CO}(S)$. If S is closed, analogous definitions are given for $\mathrm{PR}(S)$.

We make the same assumptions about the nature of the set Γ as we did about the nature of the set of symbols in section 1.4. Let Γ_s be the set of *simple* channel names. We assume that $\Gamma = (\cup n : n > 0 : \Gamma_s^n)$. An element of $\Gamma \setminus \Gamma_s$ is called a *compound* channel name. For all channel names c and d $c{\cdot}d$ is a channel name as well. Furthermore, we define

$$p{\cdot}\langle c, m\rangle = \langle p{\cdot}c, m\rangle$$

for all $p, c \in \Gamma$ and $m \in \mathrm{M}_{\surd}$. Symbol $\langle c, m\rangle \in \Omega = \Gamma \times \mathrm{M}_{\surd}$ is called *simple* if channel name c is simple, otherwise it is called *compound.*

For every command S we define the set of variables $v(S)$ inductively by

$$v(\varepsilon) = \varnothing$$
$$v(c) = \varnothing$$
$$v(c?x) = \{x\}$$
$$v(c!e) = var(e)$$

$$v(x_0, x_1, \ldots, x_{m-1} := e_0, e_1, \ldots, e_{m-1}) =$$
$$\{ x_i \mid 0 \leq i < m \} \cup (\cup i : 0 \leq i < m : var(e_i))$$
$$v(S \mid T) = v(S) \cup v(T)$$
$$v(S\,;\, T) = v(S) \cup v(T)$$
$$v(S\,,\, T) = v(S) \cup v(T)$$
$$v(S^0) = v(S)$$
$$v(S^*) = v(S)$$

The set $v(S)$ consists of all variable names that occur in command S.

The program

com c(**sig** C, **in** D, **out** E) :
  var $v_0, v_1, \ldots, v_{l-1}$ **rav**
  S
moc

denotes a component without subcomponents where c is the name of the component, C, D, and E are finite channel sets consisting of simple channel names only (the external channels of the component), $v_0, v_1, \ldots, v_{l-1}$ are l distinct variable names (the internal variables of the component), and S is a closed command. We require that

- $v(S) = \{v_0, v_1, \ldots, v_{l-1}\}$
- $\mathbf{sPR}(S) = C \quad \mathbf{iPR}(S) = D \quad \mathbf{oPR}(S) = E$

The system of component c, denoted by $sys(c)$, is defined by

$$sys(c) = sys(S)$$

Notice that $sys(c) = \langle C \cup D \cup E, \{\mathrm{PR}(S)\}\rangle$. The process of component c, denoted by $\mathrm{PR}(c)$, is defined to be $\mathrm{PR}(c) = \mathrm{PR}(sys(c))$. Notice that $\mathrm{PR}(c) = \mathrm{PR}(S)$.

The program

com c(**sig** C, **in** D, **out** E) :
  sub $p_0 : c_0, p_1 : c_1, \ldots, p_{n-1} : c_{n-1}$ **bus**
  $[x_0 = y_0, x_1 = y_1, \ldots, x_{m-1} = y_{m-1}]$
  var $v_0, v_1, \ldots, v_{l-1}$ **rav**
  S
moc

denotes a component with subcomponents where c, C, D, E, v_0, v_1, ..., v_{l-1}, and S play the same role as above, c_0, c_1, ..., and c_{n-1} are previously defined components, called the subcomponents of c and having names p_0, p_1, ..., and p_{n-1}, respectively. We require that C, D, and E contain simple channel names only and that p_0, p_1, ..., and p_{n-1} are n distinct, simple names. With subcomponent p_i system $p_i{\cdot}sys(c_i)$ is associated. The set

$$B = (\cup i : 0 \leq i < n : \mathbf{e}p_i{\cdot}sys(c_i))$$

is called the set of internal channels. We define

$$\mathcal{C} = \{C \cup D \cup E\} \cup \{\, \mathbf{e}p_i{\cdot}sys(c_i) \mid 0 \leq i < n \,\}$$

Notice that $\mathcal{C}$ is a collection of $n+1$ mutually disjoint channel sets. The equalities represent (internal) connections. We impose the same restrictions as in section 1.4 and add two restrictions dealing with input and output.

- $(\mathbf{A}\, j : 0 \leq j < m : x_j \in B)$
- $(\mathbf{A}\, j : 0 \leq j < m : y_j \in B \cup C \cup D \cup E)$
- $|\{\, x_i \mid 0 \leq i < m \,\}| = m$
- $\{\, x_i \mid 0 \leq i < m \,\} \cap \{\, y_j \mid 0 \leq j < m \,\} = \varnothing$
- for all j, $0 \leq j < m$, channels x_j and y_j belong to two different channel sets in $\mathcal{C}$
- for all i and j, $0 \leq i < j < m$ such that $y_i = y_j$ channels x_i and x_j belong to two different channels sets in $\mathcal{C}$
- $(\mathbf{A}\, j : 0 \leq j < m$
  $: (\mathbf{A}\, C_0, C_1 : C_0 \in \mathcal{C} \wedge C_1 \in \mathcal{C} \wedge x_j \in C_0 \wedge y_j \in C_1$
  $: (x_j \in \mathbf{s}C_0 \equiv y_j \in \mathbf{s}C_1)$
  $\wedge\ (x_j \in \mathbf{i}C_0 \equiv y_j \in \mathbf{o}C_1)$
  $\wedge\ (x_j \in \mathbf{o}C_0 \equiv y_j \in \mathbf{i}C_1)))$
- $(\mathbf{A}\, j : 0 \leq j < m \wedge (\mathbf{E}\, C_0 : C_0 \in \mathcal{C} : y_j \in \mathbf{i}C_0) : (\mathbf{N}\, i : 0 \leq i < m : y_i = y_j) = 1)$

Furthermore, we require that every external channel appears in the command S or is connected to an internal channel

$$C \cup D \cup E \subseteq \mathbf{c}\mathrm{PR}(S) \cup \{\, y_j \mid 0 \leq j < m \,\}$$

The channel set of command S consists of external channels and internal channels not in $\{\, x_j \mid 0 \leq j < m \,\}$

$$\mathbf{c}\mathrm{PR}(S) \subseteq C \cup D \cup E \cup B \setminus \{\, x_j \mid 0 \leq j < m \,\}$$

The system of component c, denoted by $sys(c)$, is defined by

$$sys(c) = ((\parallel\ i : 0 \leq i < n : (p_i{\cdot}sys(c_i))^{x_0,x_1,\ldots,x_{m-1}}_{y_0,y_1,\ldots,y_{m-1}}) \parallel sys(S)) \restriction (C \cup D \cup E)$$

The process of component c, denoted by $\mathrm{PR}(c)$, is defined by $\mathrm{PR}(c) = \mathrm{PR}(sys(c))$. Notice that $\mathbf{e}sys(c) = C \cup D \cup E$ and $\mathbf{c}\mathrm{PR}(c) = C \cup D \cup E$. We have

$$\mathrm{PR}(c) = ((\mathbf{W}\, i : 0 \leq i < n : (p_i{\cdot}\mathrm{PR}(c_i))^{x_0,x_1,\ldots,x_{m-1}}_{y_0,y_1,\ldots,y_{m-1}}) \,\mathbf{w}\, \mathrm{PR}(S)) \restriction (C \cup D \cup E)$$

Finally, we introduce recursive components. As in section 1.4 we restrict ourselves to the most simple form of recursion. Let component c be given by the program

com c(**sig** C, **in** D, **out** E) :
  sub $p : c$ **bus**
  var $v_0, v_1, \ldots, v_{l-1}$ **rav**
  S
moc

where C, D, and E are finite channel sets consisting of simple channel names only, p is a simple name, and S is a closed command. We require that

- $v(S) = \{v_0, v_1, \ldots, v_{l-1}\}$
- $\mathbf{s}\mathrm{PR}(S) = C \cup p{\cdot}C \quad \mathbf{i}\mathrm{PR}(S) = D \cup p{\cdot}E \quad \mathbf{o}\mathrm{PR}(S) = E \cup p{\cdot}D$

As in section 1.4 we define the system of component c, denoted by $sys(c)$, to be the unique fixpoint of $sys(c) = (p{\cdot}sys(c) \parallel sys(S)) \restriction (C \cup D \cup E)$, i.e.

$$sys(c) = \langle\, C \cup D \cup E, \{\, (p{\cdot})^i \mathrm{PR}(S) \mid i \geq 0 \,\} \rangle$$

The process of component c, denoted by $\mathrm{PR}(c)$, is defined by $\mathrm{PR}(c) = \mathrm{PR}(sys(c))$. We now have

$$\mathrm{PR}(c) = (\mathbf{W}\, i : i \geq 0 : (p{\cdot})^i \mathrm{PR}(S)) \restriction (C \cup D \cup E)$$

Finally, we present some results concerning components that have a restricted command. By theorems 4.1.9, 4.1.11, and 4.1.12 we have

Theorem 4.2.0

Let component c be given by program

com c(**sig** C, **in** D, **out** E) :
  var $v_0, v_1, \ldots, v_{l-1}$ **rav**
  S
moc

If S is a restricted command then PR(c) is both data independent and channel order independent, and γ(PR(c)) is cubic.

(End of Theorem)

Theorem 4.2.1

Let component c be given by program

com c(**sig** C, **in** D, **out** E) :
  sub $p_0 : c_0, p_1 : c_1, \ldots, p_{n-1} : c_{n-1}$ **bus**
  $[x_0 = y_0, x_1 = y_1, \ldots, x_{m-1} = y_{m-1}]$
  var $v_0, v_1, \ldots, v_{l-1}$ **rav**
  S
moc

If S is a restricted command, PR(c_i) is data independent for all i, $0 \leq i < n$, and γ(PR(c_i)) is cubic for all i, $0 \leq i < n$ then PR(c) is data independent and γ(PR(c)) is cubic.

Proof

Due to the restrictions imposed we can apply theorem 3.2.9. We now have that

$$W = (\mathbf{W}\, i : 0 \leq i < n : (p_i{\cdot}\mathrm{PR}(c_i))^{x_0,x_1,\ldots,x_{m-1}}_{y_0,y_1,\ldots,y_{m-1}}) \ \mathbf{w}\ \mathrm{PR}(S)$$

is data independent and

$$\gamma(W) = (\mathbf{W}\, i : 0 \leq i < n : (p_i{\cdot}\gamma(\mathrm{PR}(c_i)))^{x_0,x_1,\ldots,x_{m-1}}_{y_0,y_1,\ldots,y_{m-1}}) \ \mathbf{w}\ \gamma(\mathrm{PR}(S))$$

By corollary 2.4.3 and theorem 4.1.12 we now have that $\gamma(W)$ is cubic and by theorem 2.4.7 that $\gamma(\mathrm{PR}(c)) = \gamma(W)\restriction(C \cup D \cup E)$ is cubic. Using theorem 2.3.8 we infer that $C \cup D \cup E$ is non-disabling with respect to $\gamma(W)$. Using theorem 3.2.4 it follows that $\mathrm{PR}(c) = W\restriction(C \cup D \cup E)$ is data independent.

(End of Proof)

By applying the same reasoning as in the proof of the above theorem one can prove the following theorem.

Theorem 4.2.2

Let component c be given by the program

com c(**sig** C, **in** D, **out** E) :
  sub p : c **bus**
  var $v_0, v_1, \ldots, v_{l-1}$ **rav**
  S
moc

If S is a restricted command then PR(c) is data independent and γ(PR(c)) is cubic.

(End of Theorem)

5 Derivation and correctness of programs

5.0 Introduction

In this chapter we show how one may derive programs from specifications and prove them to be correct. The way in which programs are derived is also presented in [Re87]. The specifications considered are split specifications describing the external behaviour of a program or component. Again, we point out that in that case the communication behaviour is specified independently of the input/output relation, i.e. the interdependence between values received and values sent. In our derivations we see to it that this independence is maintained.Furthermore, the systems of the derived programs describe networks of processes that are characterized by the conditions for networks given in section 0.0.

After deriving a program we formally prove that it satisfies the specification using results from the previous chapters. Moreover, it is shown that the derived program defines a non-divergent and lockfree system that has constant response time.

In this chapter split specifications for data independent processes that are channel order independent as well are given in a form that differs somewhat from the form introduced in section 3.3. Moreover, a split specification also specifies the input channels, the output channels, and the signals. This new form is introduced in section 5.1.

In each of the following four sections a programming problem is presented for which a solution is derived. In section 5.1 a program is derived that recognizes palindromes of length N in an incoming sequence of integers for some N, $N \geq 0$. The system defined by the derived program consists of $(N \textbf{ div } 2) + 1$ different processes. In section 5.2 a program is derived that determines whether or not the sequence of integers received thus far is a square. The derived program is recursive and, therefore, defines a system that consists of an infinite sequence of processes, all of the same type. In sections 5.1 and 5.2 we show how one formally proves the derived programs to be correct. In [Ro,Tch] systolic arrays are given as solutions to the programming problem from section 5.2 and a programming problem similar to the one from section 5.1.

In section 5.3 a program is derived that computes the coefficients of the product of an arbitrary polynomial with a given polynomial. The number of processes of the system that is finally derived equals the degree of the given polynomial plus two. Only two

processes are of a type that differs from the type of the rest and of these two only one depends on the degree of the polynomial that is to be multiplied by the given polynomial. In [Re87] it is shown that such a component may be used to construct a component for the encoding of messages using a cyclic code. There it is mentioned that the program obtained differs totally from solutions found elsewhere.

In section 5.4 a program is derived that is an acceptor for a given regular expression, i.e. it determines whether the sequence of symbols received thus far is an element of the language defined by the regular expression. The derived system is a network of processes whose structure corresponds to the parse tree of the regular expression that is used in the derivation. The approach taken in the derivation is also found in [Fo,Ku], [We] and [An,Cl,Fo,Mi]. Different solutions can be found in [Fl,Ul] and [Sa].

Another example that illustrates this way of deriving programs can be found in [Ve88].

Finally, in section 5.5 we summarize the programming method presented in this chapter.

5.1 Recognition of palindromes

A finite sequence is called a *palindrome* if it is equal to its reverse. For N, $N \geq 0$, consider the following specification

$$
\begin{aligned}
\mathrm{PAL}_N = \; & \langle\, \mathrm{PR}(a^N\,;\,(b\,;\,a)^*) \\
& \;,\, \{\,(a, \mathcal{Z})\,,\,(b, \{\mathrm{false}, \mathrm{true}\})\,\} \\
& \;,\, t : (\mathbf{A}\, k : 0 \leq k < \ell(t\restriction b)\, \mathbf{min}\, (\ell(t\restriction a) - N + 1) \\
& \qquad : b(k,t) \;\equiv\; (\mathbf{A}\, i,j : 0 \leq i,j < N \wedge i + j = N - 1 \\
& \qquad\qquad : a(k+i,t) = a(k+j,t))\,)\,\rangle
\end{aligned}
$$

describing a process that recognizes palindromes of length N in an incoming sequence of integers. This informal interpretation shows that a is considered to be an input channel and b an output channel. In the sequel a specification as above is written as follows

$$
\begin{aligned}
\mathrm{PAL}_N : \; & \text{signals:} \quad — \\
& \text{input channels:} \quad a : \mathrm{int} \\
& \text{output channels:} \quad b : \mathrm{bool} \\
& \text{communication behaviour:} \quad a^N\,;\,(b\,;\,a)^* \\
& \text{input/output relation:} \\
& \quad b(k) \;\equiv\; (\mathbf{A}\, i,j : 0 \leq i,j < N \wedge i + j = N - 1 \\
& \qquad\qquad : a(k+i) = a(k+j))\,) \qquad\qquad k \geq 0
\end{aligned}
$$

The above form of a specification is more apt to our derivations. Notice that apart from specifying the types of the channels we also specify whether they are input channels, output channels, or signals. Furthermore, we dropped trace t in the input/output relation since it does not depend on the order of events within trace t (the specified process is channel order independent). We also omitted the upperbounds on k, since these restrictions on k are implied by the communication behaviour. We shall, however, interpret this differently. From the above input/output relation it can be seen that in order to compute $b(k)$, $k \geq 0$, one needs $a(k)$, ..., $a(k+N-1)$. The input/output relation, therefore, requires that $a(k+N-1)$ is received before $b(k)$ is sent. It is easily seen that this requirement is met by the given communication behaviour; in fact, the communication behaviour is more restrictive, since it also requires that $b(k)$ is sent before $a(k+N)$ is received. This means $b(k)$ is sent as soon as it can be computed. Whenever we derive new input/output relations we will adapt or derive communication behaviours according to the above interpretation.

Assume $N \geq 2$. We derive for k, $k \geq 0$,

$$
\begin{array}{ll}
 & b(k) \\
= & \quad \{ \text{input/output relation } \mathrm{PAL}_N \} \\
 & (\mathbf{A}\, i,j : 0 \leq i,j < N \wedge i+j = N-1 : a(k+i) = a(k+j)) \\
= & \quad \{ N \geq 2, \text{ calculus} \} \\
 & (\mathbf{A}\, i,j : 1 \leq i,j < N-1 \wedge i+j = N-1 : a(k+i) = a(k+j)) \\
 & \wedge\ (a(k) = a(k+N-1)) \\
= & \quad \{ \text{calculus} \} \\
 & (\mathbf{A}\, i,j : 0 \leq i,j < N-2 \wedge i+j = (N-2)-1 : a(k+1+i) = a(k+1+j)) \\
 & \wedge\ (a(k) = a(k+N-1))
\end{array}
$$

The first conjunct in the last predicate in the above derivation closely resembles the right hand side of the input/output relation of PAL_{N-2}. Therefore, assume that there exists a component pal_{N-2} whose process satisfies PAL_{N-2}. We introduce a subcomponent p of type pal_{N-2}, and we require

$$(0) \quad p{\cdot}a(k) = a(k+1) \qquad\qquad k \geq 0$$

It now follows that for k, $k \geq 0$,

$$
\begin{array}{ll}
 & p{\cdot}b(k) \\
= & \quad \{ \text{input/output relation } \mathrm{PAL}_{N-2} \} \\
 & (\mathbf{A}\, i,j : 0 \leq i,j < N-2 \wedge i+j = (N-2)-1 : p{\cdot}a(k+i) = p{\cdot}a(k+j)) \\
= & \quad \{ (0) \} \\
 & (\mathbf{A}\, i,j : 0 \leq i,j < N-2 \wedge i+j = (N-2)-1 : a(k+i+1) = a(k+j+1))
\end{array}
$$

and, hence,

(1) $b(k) \equiv p{\cdot}b(k) \wedge (a(k) = a(k+N-1))$

In order to compute $b(k)$ the values $a(k+N-1)$ and $a(k)$ have to be available. The communication behaviour implies that $a(k+N-1)$ is received immediately before $b(k)$ has to be sent. To have $a(k)$ available one could decide to store the last N values received via channel a. This, however, does not comply with the requirement that the processes in the system described by the component are simple. We have to solve this problem in another way. Observe that the values $a(k+1)$, $k \geq 0$, are sent to the subcomponent. Therefore, we could require that the subcomponent returns these values at a suitable moment via an additional output channel. As a consequence only value $a(0)$ needs to be stored. Let subcomponent p have an additional output channel $p{\cdot}c$ of type integer. We require that

(2) $p{\cdot}c(k) = p{\cdot}a(k) \qquad k \geq 0$

which implies, using (0),

(3) $p{\cdot}c(k) = a(k+1) \qquad k \geq 0$

Combining (1) and (3) and distinguishing the cases $k = 0$ and $k > 0$ yields

(4) $b(0) \equiv p{\cdot}b(0) \wedge (a(0) = a(N-1))$
(5) $b(k+1) \equiv p{\cdot}b(k+1) \wedge (p{\cdot}c(k) = a(k+N)) \qquad k > 0$

Adding output channel $p{\cdot}c$ to subcomponent p implies introducing an additional output channel c of the component. The input/output relation in PAL_N is extended with

(6) $c(k) = a(k) \qquad k \geq 0$

Combining (3) and (6) yields

(7) $c(0) = a(0)$
(8) $c(k+1) = p{\cdot}c(k) \qquad k \geq 0$

Relation (5) shows that $p{\cdot}b(k+1)$ and $p{\cdot}c(k)$ are needed for the same computation. Therefore, we require that subcomponent p sends these values simultaneously, i.e. its communication behaviour is

$$(p{\cdot}a)^{N-2};\ p{\cdot}b;\ (p{\cdot}a\,;\ p{\cdot}b\,,\ p{\cdot}c)^*$$

Consequently, the communication behaviour in PAL_N is changed to

$$a^N ; b ; (a ; b , c)^*$$

The remaining problem is to give a command whose process satisfies input/output relations (0), (4), (5), (7), and (8). Its communication behaviour S_N should satisfy

$$S_N \restriction \{a, b, c\} = a^N ; b ; (a ; b , c)^*$$

and

$$S_N \restriction \{p{\cdot}a, p{\cdot}b, p{\cdot}c\} = (p{\cdot}a)^{N-2} ; p{\cdot}b ; (p{\cdot}a ; p{\cdot}b , p{\cdot}c)^*$$

The restrictions on S_N imposed by (0), (4), (5), (7), and (8) should be taken into account. Furthermore, we require that there is as little buffering of values as possible. This can be expressed as follows

$$\begin{aligned} S_N \restriction \{a, p{\cdot}a\} &= a ; (a ; p{\cdot}a)^* \\ S_N \restriction \{b, p{\cdot}b, p{\cdot}c\} &= p{\cdot}b ; (b ; p{\cdot}b , p{\cdot}c)^* \\ S_N \restriction \{c, p{\cdot}c\} &= (p{\cdot}c ; c)^* \end{aligned}$$

A communication behaviour that satisfies all of the above requirements is

$$S_N = a ; (a ; p{\cdot}a)^{N-2} ; a , p{\cdot}b ; b , p{\cdot}a ; (a , p{\cdot}b , p{\cdot}c ; b , c , p{\cdot}a)^*$$

Observe the alternation of input and output in the repetition in the above command. This leads to the following program

com pal_N(**in** a : int, **out** b : bool, c : int) :
  sub p : pal_{N-2} **bus**
  var x, y, z : int, w : bool **rav**
  $a?x$; $(a?y ; p{\cdot}a!y)^{N-2}$
  ; $a?y , p{\cdot}b?w ; b!(w \wedge (x = y)) , p{\cdot}a!y$
  ; $(a?y , p{\cdot}b?w , p{\cdot}c?z ; b!(w \wedge (z = y)) , c!x , p{\cdot}a!y ; x := z)^*$
moc

Observe that the values received via $p{\cdot}c$ have to be buffered thereby necessitating an additional local variable.

Assume $N = 0$ or $N = 1$. We now have

$$b(k) = \text{true} \qquad\qquad k \geq 0$$

This leads to

com pal_0(**in** a : int, **out** b : bool, c : int) :
  var x : int **rav**
  b!true ; (a?x ; b!true , c!x)*
moc

and

com pal_1(**in** a : int, **out** b : bool, c : int) :
  var x, y : int **rav**
  a?x ; b!true ; (a?y ; b!true , c!x ; x:= y)*
moc

We will now formally prove that PR(pal_N) is the process specified by PAL$_N$. First we derive some preliminary results.

Let T_N be the command of component pal_N. Then we have $S_N = \gamma(T_N)$. Observe that PR(pal_0) and PR(pal_1) are data independent, and that γ(PR(pal_0)) and γ(PR(pal_1)) are cubic.

Let $N \geq 2$. Assume that PR(pal_{N-2}) is data independent, and γ(PR(pal_{N-2})) is cubic. Since T_N is a closed restricted command and, therefore, S_N is a restricted command, we have by theorems 4.1.9 and 2.4.10 that PR(T_N) is data independent and PR(S_N) is cubic. The conditions of theorem 3.2.8 being met we have that $p{\cdot}\mathrm{PR}(pal_{N-2})$ w PR(T_N) is data independent and

$$\gamma(\, p{\cdot}\mathrm{PR}(pal_{N-2}) \,\mathbf{w}\, \mathrm{PR}(T_N)\,) = \gamma(p{\cdot}\mathrm{PR}(pal_{N-2})) \,\mathbf{w}\, \mathrm{PR}(S_N)$$

Theorem 2.4.3 implies that $\gamma(p{\cdot}\mathrm{PR}(pal_{N-2}))$ w PR(S_N) is cubic. By theorem 2.3.8 we have that $\{a, b, c\}$ is non-disabling with respect to $\gamma(p{\cdot}\mathrm{PR}(pal_{N-2}))$ w PR(S_N), by theorem 3.2.4 that

$$\mathrm{PR}(pal_N) = (\, p{\cdot}\mathrm{PR}(pal_{N-2}) \,\mathbf{w}\, \mathrm{PR}(T_N)\,) \restriction \{a, b, c\}$$

is data independent, and by corollary 2.4.7 that γ(PR(pal_N)) is cubic.

Since $\{a, b, c\}$ is non-divergent with respect to PR(S_N) and $\{p{\cdot}a, p{\cdot}b, p{\cdot}c\}$ is non-divergent with respect to $\gamma(p{\cdot}\mathrm{PR}(pal_{N-2}))$ we have by theorem 2.1.11 that $\{a, b, c\}$ is non-divergent and, therefore, transparent with respect to $\gamma(p{\cdot}\mathrm{PR}(pal_{N-2}))$ w PR(S_N).

We now start with the actual proof. It is easily seen that PR(pal_0) and PR(pal_1) are the processes specified by PAL$_0$ and PAL$_1$, respectively.

Let $N \geq 2$. Assume that PR(pal_{N-2}) is the process specified by PAL$_{N-2}$. Process PR(T_N) is the process specified by $\langle$ PR(S_N), Q_N $\rangle$ (notice that we omit the function that describes the types of the channels) where

$$
\begin{aligned}
&Q_N(t) \equiv \\
&\quad (\mathbf{A}\, k : 0 \le k \le 0 < \ell(t\restriction b) : b(0) \equiv p{\cdot}b(0) \wedge (a(0) = a(N-1))) \\
&\quad \wedge (\mathbf{A}\, k : 0 < k < \ell(t\restriction b) \\
&\qquad\qquad : b(k) \equiv p{\cdot}b(k) \wedge (p{\cdot}c(k-1) = a(k+N-1))) \\
&\quad \wedge (\mathbf{A}\, k : 0 \le k \le 0 < \ell(t\restriction c) : c(0) = a(0)) \\
&\quad \wedge (\mathbf{A}\, k : 0 < k < \ell(t\restriction c) : c(k) = p{\cdot}c(k-1)) \\
&\quad \wedge (\mathbf{A}\, k : 0 \le k < \ell(t\restriction p{\cdot}a) : p{\cdot}a(k) = a(k+1))
\end{aligned}
$$

Applying the Conjunction-Weave rule (theorem 3.3.5) yields that $p{\cdot}\mathrm{PR}(pal_{N-2})\,\mathbf{w}\,\mathrm{PR}(T_N)$ is the process specified by

$$
\begin{aligned}
&\langle\, \mathrm{PR}(S_N), \\
&\quad t : (\mathbf{A}\, k : 0 \le k < \ell(t\restriction p{\cdot}b) \\
&\qquad\qquad : p{\cdot}b(k) \equiv (\mathbf{A}\, i,j : 0 \le i,j < N-2 \\
&\qquad\qquad\qquad\qquad \wedge\, i+j = N-3 : p{\cdot}a(k+i) = p{\cdot}a(k+j))) \\
&\qquad \wedge (\mathbf{A}\, k : 0 \le k < \ell(t\restriction p{\cdot}c) : p{\cdot}c(k) = p{\cdot}a(k)) \\
&\qquad \wedge Q_N(t) \\
&\rangle
\end{aligned}
$$

The above specification can be transformed into

$$
\begin{aligned}
&\langle\, \mathrm{PR}(S_N), \\
&\quad t : (\mathbf{A}\, k : 0 \le k < \ell(t\restriction b) : b(k) \equiv (\mathbf{A}\, i,j : 0 \le i,j < N \\
&\qquad\qquad\qquad\qquad \wedge\, i+j = N : a(k+i) = a(k+j))) \\
&\qquad \wedge (\mathbf{A}\, k : 0 \le k < \ell(t\restriction c) : c(k) = a(k)) \\
&\qquad \wedge (\mathbf{A}\, k : 0 \le k < \ell(t\restriction p{\cdot}a) : p{\cdot}a(k) = a(k+1)) \\
&\qquad \wedge (\mathbf{A}\, k : 0 \le k < \ell(t\restriction p{\cdot}b) \\
&\qquad\qquad : p{\cdot}b(k) \equiv (\mathbf{A}\, i,j : 0 \le i,j < N-2 \\
&\qquad\qquad\qquad\qquad \wedge\, i+j = N-3 : a(k+i+1) = a(k+j+1))) \\
&\qquad \wedge (\mathbf{A}\, k : 0 \le k < \ell(t\restriction p{\cdot}c) : p{\cdot}c(k) = a(k+1)) \\
&\rangle
\end{aligned}
$$

Let U_N be the process specified by PAL_N. From the above specification we infer that

$$
\mathrm{PR}(pal_N) = (\, p{\cdot}\mathrm{PR}(pal_{N-2})\,\mathbf{w}\,\mathrm{PR}(T_N)\,)\restriction\{a,b,c\} \subseteq U_N
$$

We prove that the reverse inclusion holds as well by induction on the length of traces from U_N. Let $t \in \mathbf{t}U_N$.

base $\ell(t) = 0 \qquad t = \varepsilon \in \mathbf{t}\mathrm{PR}(pal_N)$

step $\ell(t) > 0$

Let $t = u\langle d, m\rangle$. Since $\ell(u) < \ell(t)$ choose $s \in \mathbf{t}(p{\cdot}\mathrm{PR}(pal_{N-2})\,\mathbf{w}\,\mathrm{PR}(T_N))$ such that $s\restriction\{a,b,c\} = u$. Since

$$\gamma(\,p{\cdot}\mathrm{PR}(pal_{N-2})\,\mathbf{w}\,\mathrm{PR}(T_N)\,)\restriction\{a,b,c\} = \mathrm{PR}(S_N)\restriction\{a,b,c\} = \gamma(U_N)$$

we have that $\gamma(u)d \in \mathbf{t}\gamma(p{\cdot}\mathrm{PR}(pal_{N-2})\,\mathbf{w}\,\mathrm{PR}(T_N))\restriction\{a,b,c\}$. Since $\{a,b,c\}$ is transparent with respect to $\gamma(p{\cdot}\mathrm{PR}(pal_{N-2})\,\mathbf{w}\,\mathrm{PR}(T_N))$ and $\gamma(s)\restriction\{a,b,c\} = \gamma(u)$, choose v such that

$$\gamma(s)vd \in \mathbf{t}\gamma(p{\cdot}\mathrm{PR}(pal_{N-2})\,\mathbf{w}\,\mathrm{PR}(T_N)) \wedge v\restriction\{a,b,c\} = \varepsilon$$

Process $p{\cdot}\mathrm{PR}(pal_{N-2})\,\mathbf{w}\,\mathrm{PR}(T_N)$ being data independent choose $r\langle d, n\rangle$ such that

$$sr\langle d,n\rangle \in \mathbf{t}(p{\cdot}\mathrm{PR}(pal_{N-2})\,\mathbf{w}\,\mathrm{PR}(T_N)) \wedge \gamma(sr\langle d,n\rangle) = \gamma(s)vd$$

Notice that $r\restriction\{a,b,c\} = \varepsilon$ and, hence, $(sr)\restriction\{a,b,c\} = u$. In case $d = a$ value n may be chosen arbitrarily, especially $n = m$. In case $d = b$ or $d = c$ the specifications imply that $n = m$. From this we infer that

$$u\langle d,m\rangle \in \mathbf{t}(p{\cdot}\mathrm{PR}(pal_{N-2})\,\mathbf{w}\,\mathrm{PR}(T_N))\restriction\{a,b,c\}$$

The above implies that $\mathrm{PR}(pal_N)$ indeed is the process specified by PAL_N.

We now show that $sys(pal_N)$ is both non-divergent and lockfree. Observe that

$$sys(pal_N) = \langle\, \{a,b,c\}, \{\, (p{\cdot})^i\mathrm{PR}(T_{N-2i}) \mid 0 \le 2i \le N \,\} \,\rangle$$

that $\mathbf{W}(\mathbf{p}sys(pal_N))$ is data independent, and that $\gamma(\mathbf{W}(\mathbf{p}sys(pal_N))) = \mathbf{W}(\gamma(\mathbf{p}sys(pal_N)))$. This can be shown by induction using theorem 3.2.8. By corollary 3.4.3 and theorem 3.4.12 we have that in order to prove that $sys(pal_N)$ is non-divergent and lockfree it suffices to prove that $\gamma(sys(pal_N))$ is non-divergent and lockfree.

We prove that $\gamma(sys(pal_N))$ is transparent (and, therefore, non-divergent) by induction on N. Observe that $\gamma(sys(pal_0))$ and $\gamma(sys(pal_1))$ are transparent. Let $N \ge 2$. Assume that $\gamma(sys(pal_{N-2}))$ is transparent. Since $sys(S_N)\restriction\{a,b,c\}$ is transparent and $\mathrm{PR}(S_N)\restriction\{p{\cdot}a, p{\cdot}b, p{\cdot}c\} = \mathrm{PR}\{\gamma(sys(pal_{N-2})))$ we have according to corollary 2.1.24 that

$$\gamma(sys(pal_N)) = (\,p{\cdot}\gamma(sys(pal_{N-2})) \parallel sys(S_N)\,)\restriction\{a,b,c\}$$

is transparent.

We can prove that $\gamma(sys(pal_N))$ is non-divergent in another way. Observe that

$$(\mathbf{A}\, k : k \geq 2 : (\mathbf{A}\, t : t \in \mathrm{PR}(S_k) \wedge t \neq \varepsilon : \ell(t\restriction\{p{\cdot}a, p{\cdot}b, p{\cdot}c\}) < \ell(t\restriction\{a,b,c\})))$$

This condition for commands is also found in [Ud] and [Ka] where it is shown to imply the existence of a unique solution for the recursive equation defined by a simple recursive component. Let $N \geq 0$. Let $t \in \mathbf{t}\gamma(\mathrm{PR}(pal_N))$ such that $t \neq \varepsilon$. Define

$$V(t) = \{\, s \mid s \in \mathbf{tW}(\gamma(\mathbf{p}sys(pal_N))) \wedge s\restriction\{a,b,c\} = t \,\}$$

Since $\mathbf{p}\gamma(sys(pal_N)) = \{\, (p{\cdot})^i \mathrm{PR}(S_{N-2i}) \mid 0 \leq 2i \leq N \,\}$ it follows that

$$(\mathbf{A}\, k : 0 < k \leq N \,\mathbf{div}\, 2 : (\mathbf{A}\, s : s \in V(t) : \ell(s\restriction(p{\cdot})^k\{a,b,c\}) \leq \ell(t)))$$

and, hence,

$$(\mathbf{A}\, s : s \in V(t) : \ell(s) \leq \ell(t){\cdot}(N \,\mathbf{div}\, 2 + 1))$$

We conclude that $V(t)$ is finite. Choosing $t = \varepsilon$ we obtain $V(\varepsilon) = \{\varepsilon\}$. By theorem 2.1.7 we have that $\gamma(sys(pal_N))$ is non-divergent.

We prove that $\gamma(sys(pal_N))$ is lockfree by induction on N. Observe that $\gamma(sys(pal_0))$ and $\gamma(sys(pal_1))$ are lockfree. Let $N \geq 2$. Assume $\gamma(sys(pal_{N-2}))$ is lockfree. We have that $sys(S_N)$ is lockfree and transparent. We derive

$$\begin{array}{ll} & \gamma(sys(pal_N)) \text{ is lockfree} \\ = & \quad \{\, \text{definition } pal_N \,\} \\ & (\gamma(p{\cdot}sys(pal_{N-2})) \parallel sys(S_N))\restriction\{a,b,c\} \text{ is lockfree} \\ = & \quad \{\, \text{definition lockfree system} \,\} \\ & \gamma(p{\cdot}sys(pal_{N-2})) \parallel sys(S_N) \text{ is lockfree} \\ = & \quad \{\, \text{corollary 2.2.8} \,\} \\ & lockfree(\, \{\mathrm{PR}(\gamma(p{\cdot}sys(pal_{N-2}))), \mathrm{PR}(S_N)\}\,) \\ = & \quad \{\, \mathrm{PR}(S_N)\restriction\{p{\cdot}a, p{\cdot}b, p{\cdot}c\} = \mathrm{PR}(\gamma(p{\cdot}sys(pal_{N-2}))) \,\} \\ & \text{true} \end{array}$$

Finally, we show that $\gamma(sys(pal_N))$ has constant response time. Notice that by giving a sequence function for $\gamma(sys(pal_N))$ we once more show that this system is lockfree.
Let $K \geq 1$. Define for k, $0 < k \leq K$, $\sigma_{2k} \in occ(\mathrm{PR}(S_{2k})) \to \mathcal{N}$
and $\sigma_{2k+1} \in occ(\mathrm{PR}(S_{2k+1})) \to \mathcal{N}$ by

$$\begin{aligned}
\sigma_{2k}(a,i) &= \sigma_{2k+1}(a,i) &&= 3(K-k)+2i\\
\sigma_{2k}(p{\cdot}a,i) &= \sigma_{2k+1}(p{\cdot}a,i) &&= 3(K-k)+2i+3\\
\sigma_{2k}(p{\cdot}b,i) &= \sigma_{2k+1}(p{\cdot}b,i)-2 &&= 3K+k+2i-2\\
\sigma_{2k}(b,i) &= \sigma_{2k+1}(b,i)-2 &&= 3K+k+2i-1\\
\sigma_{2k}(p{\cdot}c,i) &= \sigma_{2k+1}(p{\cdot}c,i)-2 &&= 3K+k+2i\\
\sigma_{2k}(c,i) &= \sigma_{2k+1}(c,i)-2 &&= 3K+k+2i+1
\end{aligned}$$

Define $\sigma_0 \in occ(\mathrm{PR}(S_0)) \to \mathcal{N}$ and $\sigma_1 \in occ(\mathrm{PR}(S_1)) \to \mathcal{N}$ by

$$\begin{aligned}
\sigma_0(a,i) &= \sigma_1(a,i) &&= 3K+2i\\
\sigma_0(b,i) &= \sigma_1(b,i)-2 &&= 3K+2i-1\\
\sigma_0(c,i) &= \sigma_1(c,i)-2 &&= 3K+2i+1
\end{aligned}$$

Define for k, $0 \le k \le K$, $\tau_{2k} \in occ((p{\cdot})^{K-k}\mathrm{PR}(S_{2k})) \to \mathcal{N}$ and $\tau_{2k+1} \in occ((p{\cdot})^{K-k}\mathrm{PR}(S_{2k+1})) \to \mathcal{N}$ by

$$\begin{aligned}
\tau_{2k}((p{\cdot})^{K-k}d,i) &= \sigma_{2k}(d,i) && (d,i) \in occ(\mathrm{PR}(S_{2k}))\\
\tau_{2k+1}((p{\cdot})^{K-k}d,i) &= \sigma_{2k+1}(d,i) && (d,i) \in occ(\mathrm{PR}(S_{2k+1}))
\end{aligned}$$

Observe that

$$\begin{aligned}
(\mathbf{A}\,k : 0 \le k < K :\ & (\mathbf{A}\,d,i\ : (d,i) \in occ(p{\cdot}\mathrm{PR}(S_{2k})) \cap occ(\mathrm{PR}(S_{2k+2}))\\
&\qquad : \tau_{2k}((p{\cdot})^{K-k-1}d,i) = \tau_{2k+2}((p{\cdot})^{K-k-1}d,i)\,)\\
&\wedge\ (\mathbf{A}\,d,i\ : (d,i) \in occ(p{\cdot}\mathrm{PR}(S_{2k+1})) \cap occ(\mathrm{PR}(S_{2k+3}))\\
&\qquad : \tau_{2k+1}((p{\cdot})^{K-k-1}d,i) = \tau_{2k+3}((p{\cdot})^{K-k-1}d,i)\,)\,)
\end{aligned}$$

Hence, τ_{2K} (τ_{2K+1}) restricted to $occ(\gamma(\mathrm{PR}(pal_{2K})))$ ($occ(\gamma(\mathrm{PR}(pal_{2K+1})))$) is a sequence function for $\gamma(sys(pal_{2K}))$ ($\gamma(sys(pal_{2K+1}))$). Furthermore, we have for $l = 2K$ or $2K+1$

$$\begin{aligned}
(\mathbf{A}\,t,d,e : tde \in \mathbf{t}\gamma(\mathrm{PR}(pal_l)) \wedge (d,\ell(t\restriction d)) < (e,\ell(td\restriction e))\\
: \tau_l(e,\ell(td\restriction e)) - \tau_l(d,\ell(t\restriction d)) \le 2)
\end{aligned}$$

Therefore, $\gamma(sys(pal_{2K}))$ and $\gamma(sys(pal_{2K+1}))$ have constant response time. Observe that the given upperbound does not depend on K.

5.2 Recognition of squares

A finite sequence is called a *square* if it is the concatenation of two identical sequences. Notice that the empty sequence is a square, and that the length of a square is even. We derive a component that determines whether the sequence of integers received thus far is a square. Obviously only even length sequences have to be considered. Therefore, we propose the following specification

SQUARE : signals : —
input channels : a : int
output channels : b : bool
communication behaviour : $(b; a; a)^*$
input/output relation :
$b(i) \equiv (\mathbf{A}\, j : 0 \leq j < i : a(j) = a(i+j)) \qquad i \geq 0$

An approach like the one taken in section 5.1 does not yield predicates that closely resemble the input/output relation in the above specification. Hence, we will first investigate the following generalization of SQUARE

GSQUARE : signals : —
input channels : a, c : int
output channels : b : bool
communication behaviour : $(b; a, a, c)^*$
input/output relation :
$b(i) \equiv (\mathbf{A}\, j : 0 \leq j < i : c(j) = a(i+j)) \qquad i \geq 0$

If the sequences of integers received via channels a and c are identical then the sequence of booleans sent via channel b is as specified by the input/output relation of SQUARE. Observe that

(0) $\quad b(0) \equiv \text{true}$

For i, $i \geq 0$, we derive

$$
\begin{array}{ll}
 & b(i+1) \\
= & \quad \{\text{input/output relation GSQUARE}\} \\
 & (\mathbf{A}\, j : 0 \leq j < i+1 : c(j) = a(i+1+j)) \\
= & \quad \{i \geq 0\} \\
 & (\mathbf{A}\, j : 0 \leq j < i : c(j) = a(i+1+j)) \wedge (c(i) = a(2i+1))
\end{array}
$$

We introduce a subcomponent p whose process is also specified by GSQUARE and require that for i, $i \geq 0$,

(1) $p{\cdot}a(i) = a(i+1)$
(2) $p{\cdot}c(i) = c(i)$

Notice that the component we are deriving is recursive due to the above decision that amounts to introducing an infinite sequence of subprocesses of the same type.
We continue our derivation.

$$
\begin{array}{ll}
 & b(i+1) \\
= & \quad \{ \text{ above derivation } \} \\
 & (\mathbf{A}\, j : 0 \leq j < i : c(j) = a(i+1+j)) \wedge (c(i) = a(2i+1)) \\
= & \quad \{ (1), (2) \} \\
 & (\mathbf{A}\, j : 0 \leq j < i : p{\cdot}c(j) = p{\cdot}a(i+j)) \wedge (c(i) = a(2i+1)) \\
= & \quad \{ \text{ process of } p \text{ satisfies GSQUARE } \} \\
 & p{\cdot}b(i) \wedge (c(i) = a(2i+1))
\end{array}
$$

The problem is reduced to deriving a command whose process satisfies input/output relations (0), (1), (2), and

(3) $b(i+1) \equiv p{\cdot}b(i) \wedge (c(i) = a(2i+1)) \qquad i \geq 0$

Its communication behaviour S should satisfy

$$
\begin{array}{rcl}
S{\uparrow}\{a, b, c\} & = & (b\,;\, a\,;\, a\,,\, c)^* \\
S{\uparrow}\{p{\cdot}a, p{\cdot}b, p{\cdot}c\} & = & (p{\cdot}b\,;\, p{\cdot}a\,;\, p{\cdot}a\,,\, p{\cdot}c)^*
\end{array}
$$

Taking into account (0), (1), (2), and (3) and striving for as little buffering as possible leads to

$$
\begin{array}{rcl}
S{\uparrow}\{b, p{\cdot}b\} & = & (b\,;\, p{\cdot}b)^* \\
S{\uparrow}\{a, p{\cdot}a\} & = & a\,;\, (a\,;\, p{\cdot}a)^* \\
S{\uparrow}\{c, p{\cdot}c\} & = & (c\,;\, p{\cdot}c)^*
\end{array}
$$

Communication behaviour

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

satisfies all of the above requirements. Again observe the alternation of input and output in the repetition.

The above leads to the following program

com $gsquare$(**in** a, c : int, **out** b : bool) :
sub p : $gsquare$ **bus**
var x, y : int, w : bool **rav**
$b!\text{true}$; $a?x$
; ($a?x$, $c?y$, $p{\cdot}b?w$; $b!(w \wedge (x = y))$, $p{\cdot}a!x$
; $a?x$; $p{\cdot}a!x$, $p{\cdot}c!y$)*
moc

Let T_g be the command of the above component. Let U be the process that is specified by GSQUARE. As in section 5.1 we can prove that $(\text{PR}(T_g)\,\mathbf{w}\,p{\cdot}U)\restriction\{a, b, c\}$ also satisfies GSQUARE and hence

$$(\text{PR}(T_g)\,\mathbf{w}\,p{\cdot}U)\restriction\{a, b, c\} = U$$

Since T_g satisfies

$$(\mathbf{A}\,t : t \in \text{tPR}(T_g) \wedge t \neq \varepsilon : \ell(t\restriction\{p{\cdot}a, p{\cdot}b, p{\cdot}c\}) < \ell(t\restriction\{a, b, c\}))$$

equation

$$Y : (\text{PR}(T_g)\,\mathbf{w}\,p{\cdot}Y)\restriction\{a, b, c\} = Y$$

has a unique solution ([Ud],[Ka]) being

$$(\mathbf{W}\,i : i \geq 0 : (p{\cdot})^i\text{PR}(T_g))\restriction\{a, b, c\} = \text{PR}(gsquare)$$

Consequently PR($gsquare$) is the process specified by GSQUARE.
In order to obtain the output sequence specified by SQUARE we have to send $a(i)$ via channel c at the moment we send $a(2i+1)$ via channel a. Since $a(i)$ has already been sent via channel a, we require that the component generates $a(i)$ just before it has to be sent via channel c. We introduce an additional output channel d and consider the following generalization of GSQUARE

GGSQUARE : signals : —
input channels : a, c : int
output channels : b : bool, d : int
communication behaviour : $(b\,;\, a\,;\, d\,;\, a\,,\, c)^*$
input/output relation :
$b(i) \equiv (\mathbf{A}\,j : 0 \leq j < i : c(j) = a(i+j))$ $\qquad i \geq 0$
$d(i) = a(i)$ $\qquad i \geq 0$

Assuming a subcomponent p whose process satisfies GGSQUARE, a derivation again leads to input/output relations (0), (1), (2), and (3). Furthermore, we have

(4) $\quad d(0) = a(0)$

For i, $i \geq 0$, we derive

$$
\begin{array}{ll}
 & d(i+1) \\
= & \quad \{ \text{input/output relation GGSQUARE} \} \\
 & a(i+1) \\
= & \quad \{ (1), \text{process of } p \text{ satisfies GGSQUARE} \} \\
 & p{\cdot}d(i)
\end{array}
$$

We have to find a command whose process satisfies input/output relations (0), (1), (2), (3), (4), and

(5) $\quad d(i+1) = p{\cdot}d(i) \qquad\qquad i \geq 0$

and whose communication behaviour S satisfies, apart from the conditions imposed before,

$$S \upharpoonright \{d, p{\cdot}d\} = (d\,;\, p{\cdot}d)^*$$

We choose communication behaviour

$$b\,;\, a\,;\, d\,;\, (a\,,\, c\,,\, p{\cdot}b\,;\, b\,,\, p{\cdot}a\,;\, a\,,\, p{\cdot}d\,;\, d\,,\, p{\cdot}a\,,\, p{\cdot}c)^*$$

where input and output alternate. This leads to the following program

com *ggsquare*(**in** a, c : int, **out** b : bool, d : int) :
  sub p : *ggsquare* **bus**
  var x, y, z : int, w : bool **rav**
  $b!\text{true}\,;\, a?x\,;\, d!x$
  $;\, (\, a?x\,,\, c?y\,,\, p{\cdot}b?w\,;\, b!(w \wedge (x = y))\,,\, p{\cdot}a!x$
    $;\, a?x\,,\, p{\cdot}d?z\,;\, d!z\,,\, p{\cdot}a!x\,,\, p{\cdot}c!y\,)^*$
moc

It can be shown that PR(*ggsquare*) is the process specified by GGSQUARE.
We now define component *square* to be

com *square*(**in** a : int, **out** b : bool) :
sub p : *ggsquare* **bus**
[$p{\cdot}a = a, p{\cdot}b = b$]
var x : int **rav**
$(p{\cdot}d?x\,;\, p{\cdot}c!x)^*$
moc

It is easily seen that PR(*square*) indeed is the process specified by SQUARE.
Let T_{gg} be the command of component *ggsquare*. Command $S_{gg} = \gamma(T_{gg})$ satisfies

$$(\mathbf{A}\, t : t \in \mathbf{t}\mathrm{PR}(S_{gg}) \wedge t \neq \varepsilon : \ell(t{\restriction}\{p{\cdot}a, p{\cdot}b, p{\cdot}c, p{\cdot}d\}) < \ell(t{\restriction}\{a,b,c,d\}))$$
$$\wedge\ (\mathbf{A}\, t : t \in \mathbf{t}\mathrm{PR}(S_{gg}) : \ell(t{\restriction}\{a,b,c,d\} \leq 2{\cdot}\ell(t{\restriction}\{a,b\}))$$

Analogously to the proof of theorem 2.1.18 one can show that $\gamma(sys(ggsquare)){\restriction}\{a,b\}$ is non-divergent. Since $sys((d\,;\,c)^*)$ is non-divergent, we have by corollary 2.1.12 that

$$\gamma(sys(square)) = (\,\gamma(p{\cdot}sys(ggsquare))_{a,b}^{p{\cdot}a,p{\cdot}b} \parallel sys((p{\cdot}d\,;\,p{\cdot}c)^*)\,){\restriction}\{a,b\}$$

is non-divergent.
Observe that S_{gg} is a restricted command and, therefore, PR(S_{gg}) is cubic. Define function σ, $\sigma \in occ(\mathrm{PR}(S_{gg})) \rightarrow \mathcal{N}$, by

$$\begin{aligned}
\sigma(b,i) &= 4i \\
\sigma(a,i) &= 2i+1 \\
\sigma(d,i) &= 4i+2 \\
\sigma(c,i) &= 4i+3 \\
\sigma(p{\cdot}b,i) &= 4i+3 \\
\sigma(p{\cdot}a,i) &= 2i+4 \\
\sigma(p{\cdot}d,i) &= 4i+5 \\
\sigma(p{\cdot}c,i) &= 4i+6
\end{aligned}$$

for i, $i \geq 0$. Function σ is a sequence function for PR(S_{gg}) that satisfies condition $(*)$ of theorem 2.5.19. By theorem 2.5.20 function τ, $\tau \in occ(\mathrm{PR}(\gamma(sys(ggsquare)))) \rightarrow \mathcal{N}$, defined by

$$\tau(e,i) = \sigma(e,i) \qquad\qquad (e,i) \in occ(\mathrm{PR}(\gamma(sys(ggsquare))))$$

is a sequence function for $\gamma(sys(ggsquare))$, and system $\gamma(sys(ggsquare))$ is lockfree.

Since

$$(\mathbf{A}\, t, e, g : teg \in \mathrm{tPR}(\gamma(sys(ggsquare))) \wedge (e, \ell(t\restriction e)) < (g, \ell(te\restriction g)) \\ : \tau(g, \ell(te\restriction g)) - \tau(e, \ell(t\restriction e)) \leq 1\,)$$

system $\gamma(sys(ggsquare))$ has constant response time.
It easily seen that τ restricted to $occ(\mathrm{PR}(\gamma(sys(square))))$ is a sequence function for $\gamma(sys(square))$. Therefore, $\gamma(sys(square))$ is lockfree. Moreover , $\gamma(sys(square))$ has constant response time.

5.3 Polynomial multiplication

Multiplication of polynomials with integer coefficients is defined as follows. The product of polynomial p,

$$p(X) = (\mathbf{S}\, k : 0 \leq k < M : p_k \cdot X^{M-1-k}),$$

and polynomial q,

$$q(X) = (\mathbf{S}\, k : 0 \leq k < N : q_k \cdot X^{N-1-k}),$$

is polynomial $p * q$,

$$(p * q)(X) = (\mathbf{S}\, k : 0 \leq k < M + N - 1 : (p * q)_k \cdot X^{M+N-2-k}),$$

where for k, $0 \leq k < M + N - 1$,

$$(p * q)_k = (\mathbf{S}\, i, j : 0 \leq i < M \wedge 0 \leq j < N \wedge i + j = k : p_i * q_j\,)$$

In the sequel we will represent polynomials by integer sequences containing their coefficients. Polynomial p is represented by sequence $(p_k)_{0 \leq k < M}$. We will identify polynomials with the sequences representing them. The length $\ell(p)$ of sequence p is defined in the obvious way. Only sequences p with $\ell(p) \geq 1$ will be considered. Furthermore, we define for all sequences p such that $\ell(p) \geq 2$

$$t{\cdot}p = (p_{k+1})_{0 \leq k < \ell(p)-1}$$

Polynomial multiplication corresponds to a mapping of pairs of integer sequences onto integer sequences. Sequences p and q are mapped onto sequence

$$p * q = (\,(p * q)_k\,)_{0 \leq k < \ell(p)+\ell(q)-1}$$

where for k, $0 \leq k < \ell(p) + \ell(q) - 1$,

$$(p * q)_k = (\mathbf{S}\, i, j : 0 \leq i < \ell(p) \wedge 0 \leq j < \ell(q) \wedge i + j = k : p_i * q_j)$$

If p is an infinite sequence we define $p * q$ to be the infinite sequence $((p * q)_k)_{k \geq 0}$ where for k, $k \geq 0$,

$$(p * q)_k = (\mathbf{S}\, i, j : i \geq 0 \wedge 0 \leq j < \ell(q) \wedge i + j = k : p_i * q_j)$$

Let q be a given finite sequence such that $\ell(q) \geq 1$. Consider the following specification

GMUL_q : signals : —
input channels : a : int
output channels : b : int
communication behaviour : $(a\,;\, b)^*$
input/output relation : $b = a * q$

Let p be a finite sequence. Define infinite sequence $\overline{p}$ by $\overline{p}_k = p_k$ for k, $0 \leq k < \ell(p)$, and $\overline{p}_k = 0$ for k, $k \geq \ell(p)$. Then for k, $0 \leq k < \ell(p) + \ell(q) - 1$,

$$(\overline{p} * q)_k = (p * q)_k$$

and for k, $k \geq \ell(p) + \ell(q) - 1$,

$$(\overline{p} * q)_k = 0$$

Therefore, the process specified by GMUL_q can be used to compute the coefficients of $p * q$.

First assume $\ell(q) \geq 2$. We derive for k, $k \geq 0$,

$$\begin{array}{ll} & b(k) \\ = & \quad \{\, \text{input/output relation } \mathrm{GMUL}_q \,\} \\ & (a * q)_k \\ = & \quad \{\, \text{definition } a * q \,\} \\ & (\mathbf{S}\, i, j : i \geq 0 \wedge 0 \leq j < \ell(q) \wedge i + j = k : a(i) * q_j) \\ = & \quad \{\, \ell(q) \geq 2 \,\} \\ & (\mathbf{S}\, i, j : i \geq 0 \wedge 0 < j < \ell(q) \wedge i + j = k : a(i) * q_j) + a(k) * q_0 \\ = & \quad \{\, \text{calculus} \,\} \\ & (\mathbf{S}\, i, j : i \geq 0 \wedge 0 \leq j < \ell(q) - 1 \wedge i + j = k - 1 : a(i) * q_{j+1}) + a(k) * q_0 \end{array}$$

Assume that there exists a component $gmul_{t\cdot q}$ whose process satisfies $\mathrm{GMUL}_{t\cdot q}$. Introduce a subcomponent p of type $gmul_{t\cdot q}$, and require

(0) $\quad p{\cdot}a(k) = a(k) \qquad\qquad k \geq 0$

Then for k, $k \geq 0$,

$$
\begin{aligned}
& b(k) \\
= \quad & \{\text{ above derivation, (0), definition } t{\cdot}q,\ \ell(q) \geq 2\ \} \\
& (\mathbf{S}\, i,j : i \geq 0 \wedge 0 \leq j < \ell(t{\cdot}q) \wedge i + j = k - 1 : p{\cdot}a(i) * (t{\cdot}q)_j) + a(k) * q_0
\end{aligned}
$$

Hence,

(1) $\quad b(0) = a(0) * q_0$

and for k, $k > 0$,

$$
\begin{aligned}
& b(k) \\
= \quad & \{\text{ definition } *,\ k > 0\ \} \\
& ((p{\cdot}a) * (t{\cdot}q))_{k-1} + a(k) * q_0 \\
= \quad & \{\text{ process of } p \text{ satisfies } \mathrm{GMUL}_{t\cdot q}\ \} \\
& p{\cdot}b(k-1) + a(k) * q_0
\end{aligned}
$$

Our task is reduced to giving a command whose process satisfies input/output relation (0), (1), and

(2) $\quad b(k) = p{\cdot}b(k-1) + a(k) * q_0 \qquad\qquad k > 0$

and whose communication behaviour S satisfies

$$
\begin{aligned}
S\!\uparrow\!\{a, b\} &= (a\,;\, b)^* \\
S\!\uparrow\!\{p{\cdot}a, p{\cdot}b\} &= (p{\cdot}a\,;\, p{\cdot}b)^* \\
S\!\uparrow\!\{a, p{\cdot}a\} &= (a\,;\, p{\cdot}a)^* \\
S\!\uparrow\!\{b, p{\cdot}b\} &= (b\,;\, p{\cdot}b)^*
\end{aligned}
$$

The last two conditions are imposed partly by (0), (1), and (2), and partly by the requirement that there is as little buffering as possible. It now follows that

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

This leads to component $gmul_q$ defined by

com $gmul_q$(**in** a : int, **out** b : int) :
  sub p : $gmul_{t\cdot q}$ **bus**
  var x, y : int **rav**
  $a?x$; $b!(x * q_0)$, $p{\cdot}a!x$
  ; ($a?x$, $p{\cdot}b?y$; $b!(y + x * q_0)$, $p{\cdot}a!x$)*
moc

In case $\ell(q) = 1$ we derive

$$b(k) = a(k) * q_0 \qquad\qquad k \geq 0$$

which leads to

com $gmul_q$(**in** a : int, **out** b : int) :
  var x : int **rav**
  $(a?x\,;\, b!(x * q_0))^*$
moc

One may prove that PR($gmul_q$) is the process specified by GMUL_q in a way similar to the approach taken in section 5.1. Likewise, it can be shown that $\gamma(sys(gmul_q))$ is transparent (and, therefore, non-divergent) and lockfree.

Define σ, $\sigma \in occ(\mathrm{PR}((a\,;\, b)^*)) \to \mathcal{N}$, by

$$\begin{aligned} \sigma(a,i) &= 2i \\ \sigma(b,i) &= 2i+1 \end{aligned}$$

for i, $i \geq 0$. We have that σ is a sequence function for $\gamma(sys(gmul_q))$ in case $\ell(q) = 1$.
Assume $\ell(q) \geq 2$ and σ is a sequence function for $\gamma(sys(gmul_{t\cdot q}))$.
Define ρ, $\rho \in occ(\mathrm{PR}(a\,;\, (b\,,\, p{\cdot}a\,;\, a\,,\, p{\cdot}b)^*)) \to \mathcal{N}$, by

$$\begin{aligned} \rho(a,i) &= 2i \\ \rho(b,i) &= 2i+1 \\ \rho(p{\cdot}a,i) &= 2i+1 \\ \rho(p{\cdot}b,i) &= 2i+2 \end{aligned}$$

for i, $i \geq 0$. Then ρ is a sequence function for PR(a ; (b , $p{\cdot}a$; a , $p{\cdot}b$)*). By corollary 2.5.18 we have that ρ restricted to $occ(\mathrm{PR}((p{\cdot}a\,;\, p{\cdot}b)^*))$ is a sequence function for $\gamma(p{\cdot}sys(gmul_{t\cdot q}))$. Since ρ restricted to $occ(\mathrm{PR}((a\,;\, b)^*))$ equals σ we have that σ is a sequence function for $\gamma(sys(gmul_q))$. Observe that

$$(\mathbf{A}\, t, c, d \;:\; tcd \in \mathrm{tPR}(\gamma(sys(gmul_q))) \wedge (c, \ell(t\restriction c)) < (d, \ell(tc\restriction d)) \\ :\; \sigma(d, \ell(tc\restriction d)) - \sigma(c, \ell(t\restriction c)) \leq 1\,)$$

Therefore, $\gamma(sys(gmul_q))$ has constant response time.
Let $N \geq 1$. Consider the following component.

com $mul_{q,N}$(**in** a : int, **out** b : int) :
  sub $p : gmul_q$ **bus**
  var x, y : int **rav**
  ($(a?x\,;\, p{\cdot}a!x\,;\, p{\cdot}b?y\,;\, b!y)^{N-1}$
    ; $a?x\,;\, p{\cdot}a!x\,;\, p{\cdot}b?y$
    ; $(b!y\,,\, p{\cdot}a!0\,;\, p{\cdot}b?y)^{\ell(q)-1}\,;\, b!y\,)^*$
moc

Let T_N be the command of component $mul_{q,N}$. Process $\mathrm{PR}(T_N)$ satisfies the following input/output relations

(3) $p{\cdot}a(k{\cdot}(N + \ell(q) - 1) + l) = a(kN + l)$ $\quad k \geq 0,\ 0 \leq l < N$
(4) $p{\cdot}a(k{\cdot}(N + \ell(q) - 1) + l) = 0$ $\quad k \geq 0,\ N \leq l < N + \ell(q) - 1$
(5) $b(k) = p{\cdot}b(k)$ $\quad k \geq 0$

Define sequence a_k, $k \geq 0 \wedge \ell(a_k) = N$, by

(6) $(a_k)_i = a(kN + i)$ $\quad 0 \leq i < N$

We derive for k and l, $k \geq 0 \wedge 0 \leq l < N + \ell(q) - 1$,

$$\begin{array}{ll} & b(k{\cdot}(N + \ell(q) - 1) + l) \\ = & \quad \{\,(5)\,\} \\ & p{\cdot}b(k{\cdot}(N + \ell(q) - 1) + l) \\ = & \quad \{\,\text{input/output relation } \mathrm{GMUL}_q\,\} \\ & (\mathbf{S}\, i, j : i \geq 0 \wedge 0 \leq j < \ell(q) \wedge i + j = k \cdot (N + \ell(q) - 1) + l : p{\cdot}a(i) * q_j\,) \\ = & \quad \{\,(3),\ (4)\,\} \\ & (\mathbf{S}\, m, n, j : m \geq 0 \wedge 0 \leq n < N \wedge 0 \leq j < \ell(q) \\ & \qquad \wedge\ m \cdot (N + \ell(q) - 1) + n + j = k \cdot (N + \ell(q) - 1) + l : a(mN + n) * q_j\,) \\ = & \quad \{\,\text{calculus}\,\} \\ & (\mathbf{S}\, n, j : 0 \leq n < N \wedge 0 \leq j < \ell(q) \wedge n + j = l : a(kN + n) * q_j\,) \\ = & \quad \{\,(6),\ \text{definition } *\,\} \\ & (a_k * q)_l \end{array}$$

Therefore, component $mul_{q,N}$ can be repeatedly used to calculate the coefficients of the product of polynomials of length N and polynomial q. Observe that parameter N only appears in the command of component $mul_{q,N}$ and not in its subcomponent p of type $gmul_q$.

The above components for polynomial multiplication can be used in the design of components for polynomial division and for encoding messages, using a cyclic code (see [Re87]).

5.4 Acceptors for regular expressions

Regular expressions are defined inductively as follows

- ε is a regular expression
- a is a regular expression for all symbols a
- if E_0 is regular expression then E_0^* is a regular expression
- if E_0 and E_1 are regular expressions then $E_0\,;\,E_1$ and $E_0 \mid E_1$ are regular expressions

Notice that as we already mentioned in section 1.2 the set of regular expressions forms a subset of the set of commands. With each regular expression E a *language* $\mathcal{L}(E)$ is associated that is defined by

$$\mathcal{L}(E) = \mathrm{tTR}(E)$$

An *acceptor for regular expression* E is a process that upon receiving an input symbol computes whether the sequence of symbols received thus far is an element of $\mathcal{L}(E)$. Formally,

signals : —
input channels : c : sym
output channels : r : bool
communication behaviour : $(c\,;\,r)^*$
input/output relation : $r(i) \equiv c(k : 0 \leq k \leq i) \in \mathcal{L}(E) \qquad i \geq 0$

However, since we strive for a hierarchically structured component whose structure reflects the structure of the regular expression, the above specification is not adequate. Problems arise with the star operator and the semicolon operator, in which cases not only prefixes of the input sequence have to be accepted.

Therefore, we generalize the above specification as follows

ACC_E : signals : u
input channels : e : bool, c : sym
output channels : v, r : bool
communication behaviour : $u\,;\, v\,;\, (e\,,\, c\,;\, r)^*$
input/output relation :
$v(0) \equiv \varepsilon \in \mathcal{L}(E)$
$r(i) \equiv (\mathbf{E}\, j : 0 \leq j \leq i : e(j) \wedge c(k : j \leq k \leq i) \in \mathcal{L}(E)) \qquad i \geq 0$

Notice that if

$$\begin{array}{lcll} e(0) & \equiv & \text{true} & \\ e(i) & \equiv & \text{false} & i > 0 \end{array}$$

we have $r(i) \equiv c(k : 0 \leq k \leq i) \in \mathcal{L}(E)$ for i, $i \geq 0$.
Since channel v is used only once one could eliminate channel v by using channel r also for the message that is to be sent via channel v. We have not done so for reasons of clarity. The same applies to signal u whose role could be played by an additional first message via channel e, the contents of which is irrelevant.

Let $E = \varepsilon$. We derive

$$\begin{array}{ll} & v(0) \\ = & \qquad \{\, \text{input/output relation } \mathrm{ACC}_\varepsilon \,\} \\ & \varepsilon \in \mathcal{L}(\varepsilon) \\ = & \qquad \{\, \mathcal{L}(\varepsilon) = \{\varepsilon\} \,\} \\ & \text{true} \end{array}$$

and for i, $i \geq 0$,

$$\begin{array}{ll} & r(i) \\ = & \qquad \{\, \text{input/output relation } \mathrm{ACC}_\varepsilon \,\} \\ & (\mathbf{E}\, j : 0 \leq j \leq i : e(j) \wedge c(k : j \leq k \leq i) \in \mathcal{L}(\varepsilon)\,) \\ = & \qquad \{\, i \geq 0,\ \mathcal{L}(\varepsilon) = \{\varepsilon\} \,\} \\ & \text{false} \end{array}$$

This yields

com acc_ε(**sig** u, **in** e : bool, c : sym, **out** v, r : bool) :
var x : bool, z : sym **rav**
$u\,;\, v!\text{true}\,;\, (e?x\,,\, c?z\,;\, r!\text{false})^*$
moc

Let $E = a$ for some symbol a. We derive

$$
\begin{array}{ll}
 & v(0) \\
= & \quad \{ \text{input/output relation } \mathrm{ACC}_a \} \\
 & \varepsilon \in \mathcal{L}(a) \\
= & \quad \{ \mathcal{L}(a) = \{a\} \} \\
 & \text{false}
\end{array}
$$

and for i, $i \geq 0$,

$$
\begin{array}{ll}
 & r(i) \\
= & \quad \{ \text{input/output relation } \mathrm{ACC}_a \} \\
 & (\mathbf{E}\, j : 0 \leq j \leq i : e(j) \wedge c(k : j \leq k \leq i) \in \mathcal{L}(a)) \\
= & \quad \{ \text{calculus, } i \geq 0 \} \\
 & (\mathbf{E}\, j : 0 \leq j < i : e(j) \wedge c(k : j \leq k \leq i) \in \mathcal{L}(a)) \\
 & \quad \vee\ (e(i) \wedge c(k : i \leq k \leq i) \in \mathcal{L}(a)) \\
= & \quad \{ \mathcal{L}(a) = \{a\} \} \\
 & e(i) \wedge (c(i) = a)
\end{array}
$$

This yields

com acc_a(**sig** u, **in** e : bool, c : sym, **out** v, r : bool) :
  var x : bool, z : sym **rav**
  u ; v!false ; ($e?x$, $c?z$; $r!(x \wedge (z = a))$)*
moc

Let $E = E_0 \mid E_1$. We assume the existence of components acc_{E_0} and acc_{E_1} whose processes are specified by ACC_{E_0} and ACC_{E_1}, respectively. Our goal is to derive a component that has a subcomponent p of type acc_{E_0} and a subcomponent q of type acc_{E_1}. We derive

$$
\begin{array}{ll}
 & v(0) \\
= & \quad \{ \text{input/output relation } \mathrm{ACC}_{E_0 \mid E_1} \} \\
 & \varepsilon \in \mathcal{L}(E_0 \mid E_1) \\
= & \quad \{ \mathcal{L}(E_0 \mid E_1) = \mathcal{L}(E_0) \cup \mathcal{L}(E_1) \} \\
 & \varepsilon \in \mathcal{L}(E_0) \vee \varepsilon \in \mathcal{L}(E_1) \\
= & \quad \{ \text{process of } p \text{ satisfies } \mathrm{ACC}_{E_0}, \text{ process of } q \text{ satisfies } \mathrm{ACC}_{E_1} \} \\
 & p{\cdot}v(0) \vee q{\cdot}v(0)
\end{array}
$$

and for i, $i \geq 0$,

$$
\begin{array}{ll}
& r(i) \\
= & \quad \{ \text{input/output relation } \mathrm{ACC}_{E_0|E_1} \} \\
& (\mathbf{E}\, j : 0 \leq j \leq i : e(j) \wedge c(k : j \leq k \leq i) \in \mathcal{L}(E_0 \mid E_1)) \\
= & \quad \{ \mathcal{L}(E_0 \mid E_1) = \mathcal{L}(E_0) \cup \mathcal{L}(E_1) \} \\
& (\mathbf{E}\, j : 0 \leq j \leq i : e(j) \wedge c(k : j \leq k \leq i) \in \mathcal{L}(E_0)) \\
& \vee\ (\mathbf{E}\, j : 0 \leq j \leq i : e(j) \wedge c(k : j \leq k \leq i) \in \mathcal{L}(E_1))
\end{array}
$$

Therefore, we require for i, $i \geq 0$,

$$
(0) \qquad \begin{array}{lcl} p{\cdot}e(i) & \equiv & e(i) \\ p{\cdot}c(i) & = & c(i) \\ q{\cdot}e(i) & \equiv & e(i) \\ q{\cdot}c(i) & = & c(i) \end{array}
$$

We continue for i, $i \geq 0$,

$$
\begin{array}{ll}
& r(i) \\
= & \quad \{ \text{above derivation, } (0) \} \\
& (\mathbf{E}\, j : 0 \leq j \leq i : p{\cdot}e(j) \wedge p{\cdot}c(k : j \leq k \leq i) \in \mathcal{L}(E_0)) \\
& \vee\ (\mathbf{E}\, j : 0 \leq j \leq i : q{\cdot}e(j) \wedge q{\cdot}c(k : j \leq k \leq i) \in \mathcal{L}(E_1)) \\
= & \quad \{ \text{input/output relations } \mathrm{ACC}_{E_0} \text{ and } \mathrm{ACC}_{E_1} \} \\
& p{\cdot}r(i) \vee q{\cdot}r(i)
\end{array}
$$

This yields

com $acc_{E_0|E_1}$(**sig** u, **in** e : bool, c : sym, **out** v, r : bool) :
  sub $p : acc_{E_0}, q : acc_{E_1}$ **bus**
  var x, px, qx : bool, z : sym **rav**
  u ; $p{\cdot}u$, $q{\cdot}u$; $p{\cdot}v?px$, $q{\cdot}v?qx$; $v!(px \vee qx)$
  ; ($e?x$, $c?z$; $p{\cdot}e!x$, $q{\cdot}e!x$, $p{\cdot}c!z$, $q{\cdot}c!z$
    ; $p{\cdot}r?px$, $q{\cdot}r?qx$; $r!(px \vee qx)$)*
moc

Notice that the communication behaviour corresponding to the command of component $acc_{E_0|E_1}$ satisfies the requirements that are imposed by the derived input/output relations. Furthermore, observe that all of the above derivations could also have been given starting from the original specification. The following derivations, however, make it clear that the specification had to be generalized.

Let $E = E_0\,;\, E_1$. Again we assume the existence of components acc_{E_0} and acc_{E_1} whose processes are specified by ACC_{E_0} and ACC_{E_1} respectively. We strive for a component that has a subcomponent p of type acc_{E_0} and a subcomponent of type acc_{E_1}. We derive

$$
\begin{array}{ll}
 & v(0) \\
= & \quad \{\, \text{input/output relation } \mathrm{ACC}_{E_0\,;E_1} \,\} \\
 & \varepsilon \in \mathcal{L}(E_0\,;\, E_1) \\
= & \quad \{\, \mathcal{L}(E_0\,;\, E_1) = \mathcal{L}(E_0)\mathcal{L}(E_1) \,\} \\
 & \varepsilon \in \mathcal{L}(E_0) \wedge \varepsilon \in \mathcal{L}(E_1) \\
= & \quad \{\, \text{process of } p \text{ satisfies } \mathrm{ACC}_{E_0},\ \text{process of } q \text{ satisfies } \mathrm{ACC}_{E_1} \,\} \\
 & p{\cdot}v(0) \wedge q{\cdot}v(0)
\end{array}
$$

and for i, $i \geq 0$,

$$
\begin{array}{ll}
 & r(i) \\
= & \quad \{\, \text{input/output relation } \mathrm{ACC}_{E_0\,;E_1} \,\} \\
 & (\mathbf{E}\, j : 0 \leq j \leq i : e(j) \wedge c(k : j \leq k \leq i) \in \mathcal{L}(E_0\,;\, E_1)\,) \\
= & \quad \{\, \mathcal{L}(E_0\,;\, E_1) = \mathcal{L}(E_0)\mathcal{L}(E_1) \,\} \\
 & (\mathbf{E}\, j : 0 \leq j \leq i : e(j) \wedge c(k : j \leq k \leq i) \in \mathcal{L}(E_0) \wedge \varepsilon \in \mathcal{L}(E_1)\,) \\
 & \quad \vee\ (\mathbf{E}\, j : 0 \leq j \leq i : e(j) \wedge \varepsilon \in \mathcal{L}(E_0) \wedge c(k : j \leq k \leq i) \in \mathcal{L}(E_1)\,) \\
 & \quad \vee\ (\mathbf{E}\, j : 0 \leq j \leq i : e(j) \wedge (\mathbf{E}\, l : j \leq l < i : c(k : j \leq k \leq l) \in \mathcal{L}(E_0) \\
 & \qquad\qquad \wedge\ c(k : l+1 \leq k \leq i) \in \mathcal{L}(E_1))\,)
\end{array}
$$

We require for i, $i \geq 0$,

$$
(1) \qquad \begin{array}{lcl} p{\cdot}e(i) & \equiv & e(i) \\ p{\cdot}c(i) & = & c(i) \end{array}
$$

We continue for i, $i \geq 0$,

$$
\begin{array}{ll}
 & r(i) \\
= & \quad \{\, \text{above derivation, (1), process of } p\ (q) \text{ satisfies } \mathrm{ACC}_{E_0}\ (\mathrm{ACC}_{E_1}) \,\} \\
 & (p{\cdot}r(i) \wedge q{\cdot}v(0)) \\
 & \quad \vee\ (\mathbf{E}\, j : 0 \leq j \leq i : e(j) \wedge p{\cdot}v(0) \wedge c(k : j \leq k \leq i) \in \mathcal{L}(E_1)\,) \\
 & \quad \vee\ (\mathbf{E}\, l : 0 \leq l < i : (\mathbf{E}\, j : 0 \leq j \leq l : e(j) \wedge c(k : j \leq k \leq l) \in \mathcal{L}(E_0)\,) \\
 & \qquad\qquad \wedge\ c(k : l+1 \leq k \leq i) \in \mathcal{L}(E_1)\,) \\
= & \quad \{\, (1),\ \text{process of } p \text{ satisfies } \mathrm{ACC}_{E_0},\ \text{renaming dummy} \,\}
\end{array}
$$

$$
\begin{array}{ll}
 & (p{\cdot}r(i) \wedge q{\cdot}v(0)) \\
 & \quad \vee (\mathbf{E}\, j : 0 \le j \le i : e(j) \wedge p{\cdot}v(0) \wedge c(k : j \le k \le i) \in \mathcal{L}(E_1)) \\
 & \quad \vee (\mathbf{E}\, j : 0 \le j < i : p{\cdot}r(j) \wedge c(k : j+1 \le k \le i) \in \mathcal{L}(E_1)) \\
= & \quad \{ \text{calculus} \} \\
 & (p{\cdot}r(i) \wedge q{\cdot}v(0)) \\
 & \quad \vee (\mathbf{E}\, j : 0 \le j \le i : e(j) \wedge p{\cdot}v(0) \wedge c(k : j \le k \le i) \in \mathcal{L}(E_1)) \\
 & \quad \vee (\mathbf{E}\, j : 0 < j \le i : p{\cdot}r(j-1) \wedge c(k : j \le k \le i) \in \mathcal{L}(E_1))
\end{array}
$$

Therefore, we require

$$(2) \quad q{\cdot}e(0) \equiv e(0) \wedge p{\cdot}v(0)$$

and for i, $i \ge 0$,

$$(3) \quad \begin{array}{rcl} q{\cdot}e(i+1) & \equiv & (e(i+1) \wedge p{\cdot}v(0)) \vee p{\cdot}r(i) \\ q{\cdot}c(i) & = & c(i) \end{array}$$

We continue for i, $i \ge 0$,

$$
\begin{array}{ll}
 & r(i) \\
= & \quad \{ \text{above derivation, (2), (3)} \} \\
 & (p{\cdot}r(i) \wedge q{\cdot}v(0)) \vee (\mathbf{E}\, j : 0 \le j \le i : q{\cdot}e(j) \wedge q{\cdot}c(k : j \le k \le i) \in \mathcal{L}(E_1)) \\
= & \quad \{ \text{process of } q \text{ satisfies } \mathrm{ACC}_{E_1} \} \\
 & (p{\cdot}r(i) \wedge q{\cdot}v(0)) \vee q{\cdot}r(i)
\end{array}
$$

This leads to

```
com acc_{E0;E1}(sig u, in e : bool, c : sym, out v, r : bool) :
      sub p : acc_{E0}, q : acc_{E1} bus
      var x, px, qx, pv, qv : bool, z : sym rav
      u ; p·u , q·u ; p·v?pv , q·v?qv ; v!(pv ∧ qv) , px:= false
      ; ( e?x , c?z ; p·e!x , q·e!((x ∧ pv) ∨ px) , p·c!z , q·c!z
          ; p·r?px , q·r?qx ; r!((px ∧ qv) ∨ qx) )*
moc
```

Let $E = E_0^*$. Assume the existence of a component acc_{E_0} whose process is specified by ACC_{E_0}. Again we strive for a component that has a subcomponent p of type acc_{E_0}. We derive

$$
\begin{array}{ll}
 & v(0) \\
= & \quad \{ \text{input/output relation } \mathrm{ACC}_{E_0^*} \} \\
 & \varepsilon \in \mathcal{L}(E_0^*) \\
= & \quad \{ \varepsilon \in \mathcal{L}(E_0^*) \} \\
 & \text{true}
\end{array}
$$

and

$$
\begin{array}{ll}
 & r(0) \\
= & \quad \{ \text{input/output relation } \mathrm{ACC}_{E_0^*} \} \\
 & e(0) \wedge c(k : 0 \leq k \leq 0) \in \mathcal{L}(E_0^*) \\
= & \quad \{ a \in \mathcal{L}(E_0^*) \equiv a \in \mathcal{L}(E_0) \text{ for all symbols } a \} \\
 & e(0) \wedge c(k : 0 \leq k \leq 0) \in \mathcal{L}(E_0)
\end{array}
$$

We require

$$
(4) \qquad \begin{array}{lcl} p{\cdot}e(0) & \equiv & e(0) \\ p{\cdot}c(0) & = & c(0) \end{array}
$$

Then

$$
\begin{array}{ll}
 & r(0) \\
= & \quad \{ \text{above derivation, (4)} \} \\
 & p{\cdot}e(0) \wedge p{\cdot}c(k : 0 \leq k \leq 0) \in \mathcal{L}(E_0) \\
= & \quad \{ \text{process of } p \text{ satisfies } \mathrm{ACC}_{E_0} \} \\
 & p{\cdot}r(0)
\end{array}
$$

Furthermore, we derive for i, $i \geq 0$,

$$
\begin{array}{ll}
 & r(i+1) \\
= & \quad \{ \text{input/output relation } \mathrm{ACC}_{E_0^*} \} \\
 & (\mathbf{E}\, j : 0 \leq j \leq i+1 : e(j) \wedge c(k : j \leq k \leq i+1) \in \mathcal{L}(E_0^*)) \\
= & \quad \{ \mathcal{L}(E_0^*) = \mathcal{L}(E_0^*)\mathcal{L}(E_0) \cup \{\varepsilon\},\ i \geq 0 \} \\
 & (\mathbf{E}\, j : 0 \leq j \leq i+1 : e(j) \wedge (\mathbf{E}\, l : j \leq l \leq i+1 : c(k : j \leq k < l) \in \mathcal{L}(E_0^*) \\
 & \qquad\qquad \wedge\ c(k : l \leq k \leq i+1) \in \mathcal{L}(E_0))) \\
= & \quad \{ \varepsilon \in \mathcal{L}(E_0^*) \}
\end{array}
$$

$$
\begin{array}{ll}
 & (\mathbf{E}\, j : 0 \le j \le i+1 : e(j) \wedge c(k : j \le k \le i+1) \in \mathcal{L}(E_0)\,) \\
 & \vee\ (\mathbf{E}\, j : 0 \le j \le i+1 : e(j) \wedge (\mathbf{E}\, l : j < l \le i+1 : c(k : j \le k < l) \in \mathcal{L}(E_0^*) \\
 & \qquad\qquad \wedge\ c(k : l \le k \le i+1) \in \mathcal{L}(E_0)\,)\,) \\
= & \quad \{\text{calculus}\} \\
 & (\mathbf{E}\, j : 0 \le j \le i+1 : e(j) \wedge c(k : j \le k \le i+1) \in \mathcal{L}(E_0)\,) \\
 & \vee\ (\mathbf{E}\, l : 0 < l \le i+1 : \ (\mathbf{E}\, j : 0 \le j < l : e(j) \wedge c(k : j \le k < l) \in \mathcal{L}(E_0^*)) \\
 & \qquad\qquad \wedge\ c(k : l \le k \le i+1) \in \mathcal{L}(E_0)\,) \\
= & \quad \{\text{input/output relation } \mathrm{ACC}_{E_0^*}\text{, renaming dummy}\} \\
 & (\mathbf{E}\, j : 0 \le j \le i+1 : e(j) \wedge c(k : j \le k \le i+1) \in \mathcal{L}(E_0)\,) \\
 & \vee\ (\mathbf{E}\, j : 0 < j \le i+1 : r(j-1) \wedge c(k : j \le k \le i+1) \in \mathcal{L}(E_0)\,)
\end{array}
$$

Therefore, we require for i, $i \ge 0$,

$$
(5) \qquad \begin{array}{lcl} p{\cdot}e(i+1) & \equiv & e(i+1) \vee r(i) \\ p{\cdot}c(i+1) & = & c(i+1) \end{array}
$$

We continue for i, $i \ge 0$,

$$
\begin{array}{ll}
 & r(i+1) \\
= & \quad \{\text{above derivation, (4), (5)}\} \\
 & (\mathbf{E}\, j : 0 \le j \le i+1 : p{\cdot}e(j) \wedge p{\cdot}c(k : j \le k \le i+1) \in \mathcal{L}(E_0)\,) \\
= & \quad \{\text{process of } p \text{ satisfies } \mathrm{ACC}_{E_0}\} \\
 & p{\cdot}r(i+1)
\end{array}
$$

This leads to

```
com acc_{E0*}(sig u, in e : bool, c : sym, out v, r : bool) :
      sub p : acc_{E0} bus
      var x, px : bool, z : sym rav
      u ; p·u ; p·v?px ; v!true , px:= false
      ; ( e?x , c?z ; p·e!(x ∨ px) , p·c!z ; p·r?px ; r!px )*
moc
```

By induction on the structure of regular expressions one can prove that $\mathrm{PR}(acc_E)$ is indeed the process specified by ACC_E. The proof is analogous to the proof in section 5.1.

We prove that $\gamma(sys(acc_E))$ is transparent by induction on the structure of regular expressions, i.e. the structure of components. Let T_E be the command of component

acc_E and let $S_E = \gamma(T_E)$. Obviously $\gamma(sys(acc_\varepsilon))$ and $\gamma(sys(acc_a))$, for all symbols a, are transparent. Assume $\gamma(sys(acc_{E_0}))$ and $\gamma(sys(acc_{E_1}))$ to be transparent. By corollary 2.1.22 we have that $\gamma(p{\cdot}sys(acc_{E_0}) \parallel q{\cdot}sys(acc_{E_1}))$ is transparent. Since $S_{E_0|E_1} = S_{E_0;E_1}$, system $sys(S_{E_0|E_1})\restriction\{u,v,e,c,r\}$ is transparent, and

$$
\begin{array}{ll}
 & \mathrm{PR}(S_{E_0|E_1})\restriction\{p{\cdot}u, p{\cdot}v, p{\cdot}e, p{\cdot}c, p{\cdot}r, q{\cdot}u, q{\cdot}v, q{\cdot}e, q{\cdot}c, q{\cdot}r\} \\
= & \quad \{\text{ definition } S_{E_0|E_1}\text{, property projection }\} \\
 & \mathrm{PR}(p{\cdot}u\,,\, q{\cdot}u\,;\, p{\cdot}v\,,\, q{\cdot}v\,;\, (p{\cdot}e\,,\, q{\cdot}e\,,\, p{\cdot}c\,,\, q{\cdot}c\,;\, p{\cdot}r\,,\, q{\cdot}r)^*) \\
\subseteq & \quad \{\text{ calculus }\} \\
 & \mathrm{PR}(p{\cdot}u\,;\, p{\cdot}v\,;\, (p{\cdot}e\,,\, p{\cdot}c\,;\, p{\cdot}r)^*)\,\mathbf{w}\,\mathrm{PR}(q{\cdot}u\,;\, q{\cdot}v\,;\, (q{\cdot}e\,,\, q{\cdot}c\,;\, q{\cdot}r)^*) \\
= & \quad \{\,\mathrm{PR}(acc_{E_0}) \text{ satisfies } \mathrm{ACC}_{E_0},\ \mathrm{PR}(acc_{E_1}) \text{ satisfies } \mathrm{ACC}_{E_1}\,\} \\
 & \mathrm{PR}(\gamma(p{\cdot}sys(acc_{E_0})))\,\mathbf{w}\,\mathrm{PR}(\gamma(q{\cdot}sys(acc_{E_1}))) \\
= & \quad \{\text{ calculus }\} \\
 & \mathrm{PR}(\,\gamma(p{\cdot}sys(acc_{E_0}) \parallel q{\cdot}sys(acc_{E_1}))\,)
\end{array}
$$

we have by corollary 2.1.24 that

$$
\begin{array}{l}
\gamma(sys(acc_{E_0|E_1})) = \\
\quad (\,\gamma(p{\cdot}sys(acc_{E_0}) \parallel q{\cdot}sys(acc_{E_1})) \parallel sys(S_{E_0|E_1})\,)\restriction\{u,v,e,c,r\}
\end{array}
$$

and

$$
\begin{array}{l}
\gamma(sys(acc_{E_0;E_1})) = \\
\quad (\,\gamma(p{\cdot}sys(acc_{E_0}) \parallel q{\cdot}sys(acc_{E_1})) \parallel sys(S_{E_0;E_1})\,)\restriction\{u,v,e,c,r\}
\end{array}
$$

are transparent.
Since $sys(S_{E_0^*})\restriction\{u,v,e,c,r\}$ is transparent and

$$\mathrm{PR}(S_{E_0^*})\restriction\{p{\cdot}u, p{\cdot}v, p{\cdot}e, p{\cdot}c, p{\cdot}r\} = \mathrm{PR}(\gamma(p{\cdot}sys(acc_{E_0})))$$

we have by corollary 2.1.24 that

$$\gamma(sys(acc_{E_0^*})) = (\,\gamma(p{\cdot}sys(acc_{E_0})) \parallel sys(S_{E_0^*})\,)\restriction\{u,v,e,c,r\}$$

is transparent. Consequently, $\gamma(sys(acc_E))$ is non-divergent for all regular expressions E.

Up to this point we did not mention that a regular expression might be parsed in more that one way. For instance, $a\,;\, b\,;\, c\,;\, d$ may be parsed in five different ways:

$(((a);(b));(c));(d)$
$((a);((b);(c)));(d)$
$((a);(b));((c);(d))$
$(a);((b);((c);(d)))$
$(a);(((b);(c));(d))$

Consequently, one may obtain five different components when following the above method to construct an acceptor for E. The way in which a regular expression is parsed can be represented by a so-called *parse tree*. Notice that the parse tree of a regular expression and the structure of the component that is constructed using that parse tree correspond. In the above example the third way of parsing yields a parse tree of depth 2; the other ways yield a parse tree of depth 3.

As it turns out sequence functions for $\gamma(sys(acc_E))$ depend upon the depth of the chosen parse tree for E. This dependence is such that components constructed by using parse trees of least depth are to be preferred. In the above example the third way of parsing is to be preferred above the other ways.

We introduce *parenthesized* regular expressions:

- ε is a parenthesized regular expression
- a is a parenthesized regular expression for all symbols a
- if E_0 and E_1 are parenthesized regular expressions then $(E_0) \mid (E_1)$ and $(E_0);(E_1)$ are parenthesized regular expressions
- if E_0 is a parenthesized regular expression then $(E_0)^*$ is a parenthesized regular expression

Parenthesized regular expressions have a unique parse tree. Hence, the component acc_E that corresponds to parenthesized regular expression E is uniquely defined. In the sequel we only consider parenthesized regular expressions which we call regular expressions.

For every regular expression E the depth of (the parse tree of) E, denoted by $d(E)$, is defined inductively as follows

$$\begin{aligned} d(\varepsilon) &= 0 \\ d(a) &= 0 \\ d((E_0) \mid (E_1)) &= d(E_0) \max d(E_1) + 1 \\ d((E_0);(E_1)) &= d(E_0) \max d(E_1) + 1 \\ d((E_0)^*) &= d(E_0) + 1 \end{aligned}$$

By induction on the structure of regular expressions we prove that
$\sigma_E \in occ(\mathrm{PR}(\gamma(sys(acc_E)))) \to \mathcal{N}$ defined by

$$\begin{aligned}
\sigma_E(u,0) &= 0\\
\sigma_E(v,0) &= 2\cdot d(E)+1\\
\sigma_E(e,i) &= 2\cdot d(E)+2+i\cdot(2\cdot d(E)+2)\\
\sigma_E(c,i) &= 2\cdot d(E)+2+i\cdot(2\cdot d(E)+2)\\
\sigma_E(r,i) &= 4\cdot d(E)+3+i\cdot(2\cdot d(E)+2)
\end{aligned}$$

for i, $i \geq 0$, is a sequence function for $\gamma(sys(acc_E))$.
It is obvious that σ_E is a sequence function for $\gamma(sys(acc_E))$ in case $E = \varepsilon$ or $E = a$ for some symbol a.
Let $E = (E_0) \mid (E_1)$ or $E = (E_0)\,;\,(E_1)$. We have

$$S_E = u\,;\, p{\cdot}u\,,\, q{\cdot}u\,;\, p{\cdot}v\,,\, q{\cdot}v\,;\, v\,;\, (e\,,\, c\,;\, p{\cdot}e\,,\, q{\cdot}e\,,\, p{\cdot}c\,,\, q{\cdot}c\,;\, p{\cdot}r\,,\, q{\cdot}r\,;\, r)^*$$

Define $\tau_E \in occ(\mathrm{PR}(S_E)) \to \mathcal{N}$ by

$$\begin{aligned}
&\tau_E(u,0) = 0\\
&\tau_E(p{\cdot}u,0) = \tau_E(q{\cdot}u,0) = 1\\
&\tau_E(p{\cdot}v,0) = \tau_E(q{\cdot}v,0) = 2\cdot d(E)\\
&\tau_E(v,0) = 2\cdot d(E)+1\\
&\tau_E(e,i) = \tau_E(c,i) = 2\cdot d(E)+2+i\cdot(2\cdot d(E)+2)\\
&\tau_E(p{\cdot}e,i) = \tau_E(q{\cdot}e,i) = \tau_E(p{\cdot}c,i) = \tau_E(q{\cdot}c,i) = 2\cdot d(E)+3+i\cdot(2\cdot d(E)+2)\\
&\tau_E(p{\cdot}r,i) = \tau_E(q{\cdot}r,i) = 4\cdot d(E)+2+i\cdot(2\cdot d(E)+2)\\
&\tau_E(r,i) = 4\cdot d(E)+3+i\cdot(2\cdot d(E)+2)
\end{aligned}$$

for i, $i \geq 0$. Then τ_E is a sequence function for $\mathrm{PR}(S_E)$.
Notice that τ_E restricted to $occ(\mathrm{PR}(\gamma(sys(acc_E))))$ equals σ_E.
Next, we define $\rho_{E_0} \in occ(\mathrm{PR}(\gamma(sys(acc_{E_0})))) \to \mathcal{N}$ by

$$\rho_{E_0}(a,i) = \tau_E(p.a,i) \qquad (a,i) \in occ(\mathrm{PR}(\gamma(sys(acc_{E_0}))))$$

and $\rho_{E_1} \in occ(\mathrm{PR}(\gamma(sys(acc_{E_1})))) \to \mathcal{N}$ by

$$\rho_{E_1}(a,i) = \tau_E(q.a,i) \qquad (a,i) \in occ(\mathrm{PR}(\gamma(sys(acc_{E_0}))))$$

We have

$$
\begin{array}{ll}
 & \rho_{E_0}(v,0) - \rho_{E_0}(u,0) \\
= & \quad \{ \text{definition } \rho_{E_0} \} \\
 & 2 \cdot d(E) - 1 \\
= & \quad \{ \text{definition } d(E) \} \\
 & 2 \cdot (d(E_0) \,\mathbf{max}\, d(E_1) + 1) - 1 \\
\geq & \quad \{ \text{calculus} \} \\
 & 2 \cdot d(E_0) + 1 \\
= & \quad \{ \text{definition } \sigma_{E_0} \} \\
 & \sigma_{E_0}(v,0) - \sigma_{E_0}(u,0)
\end{array}
$$

$$
\begin{array}{ll}
 & \rho_{E_0}(e,0) - \rho_{E_0}(v,0) \\
= & \quad \{ \text{definition } \rho_{E_0} \} \\
 & 2 \cdot d(E) + 3 - 2 \cdot d(E) \\
\geq & \quad \{ \text{calculus} \} \\
 & 2 \cdot d(E_0) + 2 - (2 \cdot d(E_0) + 1) \\
= & \quad \{ \text{definition } \sigma_{E_0} \} \\
 & \sigma_{E_0}(e,0) - \sigma_{E_0}(v,0)
\end{array}
$$

for i, $i \geq 0$,

$$
\begin{array}{ll}
 & \rho_{E_0}(r,i) - \rho_{E_0}(e,i) \\
= & \quad \{ \text{definition } \rho_{E_0} \} \\
 & 4 \cdot d(E) + 2 + i \cdot (2 \cdot d(E) + 2) - 2 \cdot d(E) - 3 - i \cdot (2 \cdot d(E) + 2) \\
= & \quad \{ \text{definition } d(E) \} \\
 & 2 \cdot (d(E_0) \,\mathbf{max}\, d(E_1) + 1) - 1 \\
\geq & \quad \{ \text{calculus} \} \\
 & 2 \cdot d(E_0) + 1 \\
= & \quad \{ \text{calculus} \} \\
 & 4 \cdot d(E_0) + 3 + i \cdot (2 \cdot d(E_0) + 2) - 2 \cdot d(E_0) - 2 - i \cdot (2 \cdot d(E_0) + 2) \\
= & \quad \{ \text{definition } \sigma_{E_0} \} \\
 & \sigma_{E_0}(r,i) - \sigma_{E_0}(e,i)
\end{array}
$$

and for i, $i \geq 0$,

$$
\begin{array}{ll}
 & \rho_{E_0}(e,i+1) - \rho_{E_0}(r,i) \\
= & \quad \{ \text{definition } \rho_{E_0} \}
\end{array}
$$

$$
\begin{array}{ll}
& 2 \cdot d(E) + 3 + (i+1) \cdot (2 \cdot d(E) + 2) - 4 \cdot d(E) - 2 - i \cdot (2 \cdot d(E) + 2) \\
\geq & \quad \{ \text{calculus} \} \\
& 2 \cdot d(E_0) + 2 + (i+1) \cdot (2 \cdot d(E_0) + 2) - 4 \cdot d(E_0) - 3 - i \cdot (2 \cdot d(E_0) + 2) \\
= & \quad \{ \text{definition } \sigma_{E_0} \} \\
& \sigma_{E_0}(e, i+1) - \sigma_{E_0}(r, i)
\end{array}
$$

From the above it follows that

$$
\begin{array}{l}
(\mathbf{A}\, a, b, i, j : (a,i), (b,j) \in occ(\mathrm{PR}(\gamma(sys(acc_{E_0})))) \wedge (a,i) < (b,j) \\
\qquad : \sigma_{E_0}(b,j) - \sigma_{E_0}(a,i) \leq \rho_{E_0}(b,j) - \rho_{E_0}(a,i))
\end{array}
$$

Furthermore, we have

$$
\begin{array}{l}
\sigma_{E_0}(u,0) = 0 < 1 = \rho_{E_0}(u,0) \\
\sigma_{E_0}(v,0) = 2 \cdot d(E_0) + 1 < 2 \cdot d(E) = \rho_{E_0}(v,0) \\
\sigma_{E_0}(e,0) = 2 \cdot d(E_0) + 2 < 2 \cdot d(E) + 3 = \rho_{E_0}(e,0) \\
\sigma_{E_0}(r,0) = 4 \cdot d(E_0) + 3 < 4 \cdot d(E) + 2 = \rho_{E_0}(r,0)
\end{array}
$$

By corollary 2.5.18 we now have that ρ_{E_0} is a sequence function for $\gamma(sys(acc_{E_0}))$. Analogously, we can prove that ρ_{E_1} is a sequence function for $\gamma(sys(acc_{E_1}))$.

From the above we infer that σ_E is a sequence function for $\gamma(sys(acc_E))$.

Analogously, we can prove that in case $E = (E_0)^*$ σ_E is a sequence function for $\gamma(sys(acc_E))$.

By theorem 2.5.14 we have that $\gamma(sys(acc_E))$ is lockfree. Since

$$
\begin{array}{l}
(\mathbf{A}\, t, a, b : tab \in \mathbf{t}\mathrm{PR}(\gamma(sys(acc_E))) \wedge (a, \ell(t \restriction a)) < (b, \ell(ta \restriction b)) \\
\qquad : \sigma_E(b, \ell(ta \restriction b)) - \sigma_E(a, \ell(t \restriction a)) \leq 2 \cdot d(E) + 1)
\end{array}
$$

system $\gamma(sys(acc_E))$ has constant response time.

One may interpret the constant $2 \cdot d(E) + 1$ in the above condition as a measure for the response time of component acc_E. Then it is obvious that when constructing an acceptor for regular expression E one should use a parse tree of E of minimal depth.

Defining the length of regular expression E, denoted by $\ell(E)$, by

$$
\begin{array}{rcl}
\ell(\varepsilon) & = & 1 \\
\ell(a) & = & 1 \\
\ell((E_0) \mid (E_1)) & = & \ell(E_0) + \ell(E_1) + 1 \\
\ell((E_0)\,;\,(E_1)) & = & \ell(E_0) + \ell(E_1) + 1 \\
\ell((E_0)^*) & = & \ell(E_0) + 1
\end{array}
$$

one can show that

$$\log(\ell(E)) \leq d(E) \leq \ell(E)$$

This implies that the least response time that one might achieve is $2 \cdot \lceil \log(\ell(E)) \rceil + 1$.

Consider regular expression $E_m = a_0 ; a_1 ; \ldots ; a_{m-1}$ for m, $m > 0$. Regular expression E_m can be parsed to yield parse trees of depth $\lceil \log(2m-1) \rceil$ and $2m-1$ $(= \ell(E_m))$. Therefore, acceptors for E_m can be constructed having response time $2 \cdot \lceil \log(2m-1) \rceil + 1$ and $4m-1$, respectively.

Every parse tree of regular expression $E = a ; (b ; c ; d)^*$ has depth 4 whereas $\ell(E) = 8$ and $\log(\ell(E)) = 3$. This shows that the given lower bound is not always reachable.

Finally, we observe that $sys(acc_E)$ describes a network of processes that has the form of a tree. The nodes in the network that correspond to operators in E pass on the symbols they receive. The nodes that correspond to ε in E discard the symbols they receive. Only the nodes that correspond to symbols in E compare the symbols they receive with their own symbol. Observe that the nodes of the last two kinds are the leaves of the tree. Therefore, the processes in the network could be simplified by sending the symbols directly to the nodes corresponding to symbols in E instead of letting them be passed by all other nodes. This reduces the processes in nodes corresponding to operators and ε in E to processes that have to deal with boolean values only.

5.5 Final remarks

In the previous sections we illustrated a way of deriving programs from (split) specifications. Essentially, we employed the fact that the communication behaviour is specified independently of the input/output relation, i.e. all processes specified are data independent, and the fact that the input/output relation depends only on the numbers of events in the trace, i.e. all processes specified are channel order independent. Derivations were done primarily in terms of input/output relations. Programming techniques that were used are

- introduction of one or more subprocesses having a similar specification. Similar here means equal (cf. recursive procedures and functions in sequential programming) or with one or more parameters changed (cf. e.g. invariants obtained by replacing one or more constants in the postcondition by variables in sequential programming). In the former case one introduces an infinite number of subprocesses, all of the same type. In the latter case one usually arrives at a finite number of subprocesses, all of different type.
- introduction of one or more additional channels (cf. the introduction of auxiliary variables in sequential programming)
- generalization of the original specification

Communication behaviours were modified or derived according to the requirements imposed by the derived input/output relations and by considerations concerning buffering of values. This was done in such a way that the resulting systems are non-divergent and lockfree. The derived programs describe systems that satisfy the conditions for networks of processes given in section 0.0. This is due to the fact that all specifications in our derivations are split specifications having communication behaviours that are cubic, and to the way in which we introduced (sub)processes.

The formal proofs in sections 5.1 and 5.2 give an impression as to what kind of theorem may be formulated about deriving a split specification for the projection of a process from a split specification for that process. Among the conditions to be satisfied are the requirements that the channel set C on which one projects is transparent with respect to the communication behaviour (cf. theorem 3.2.4) and that the predicate in the specification can be written as the conjunction of two predicates, one solely in terms of channels in C and the other describing the values sent via channels not in C as functions of the values sent via channels in C. We have, however, not formulated such a theorem since the examples in 5.1 and 5.2 do illustrate the principle more clearly.

6 Conclusions

In this thesis we show how communication of values and parallel computations, especially those that are characterized by the conditions given in section 0.0, can be described using trace theory as a formalism. Furthermore, we show how programs may be derived from specifications. These programs can be proved to be correct, i.e. they satisfy their specifications, and have neither deadlock nor divergence.

To the above aim we introduced the notion of system. A system is a description for a network of processes. Formally, it is a pair consisting of an alphabet (the external channels of the network) and a set of processes. The external process of a system is defined to be the composition of all processes in its process set projected on its external alphabet. Systems can be composed and projected on alphabets. The program notation of trace theory is viewed as a means to describe a certain class of systems. The process of a program or component is defined to be the external process of the system corresponding to the program or component. In case of a recursive component this definition yields, in a natural way, the least fixpoint of the recursive equation defined by the component, thus confirming the choice that is made elsewhere ([Sn],[Ka]).

The phenomena divergence and nondeterminism are captured by the introduction of the concepts of non-divergent, non-disabling, and transparent alphabets (non-disabling corresponds to I_1 in [Ka]). These concepts are introduced for systems as well. Absence of divergence is characterized in several ways. A number of useful theorems dealing with the above phenomena in case of composition and projection is given. Absence of deadlock is modelled by defining what lockfree systems are. If one wants to investigate the absence or presence of deadlock one may project on transparent alphabets that contain the common symbols.

The classes of conservative and cubic processes ([Ve85], [Ve86]) are introduced, the latter being a subclass of the former. Both classes are closed under composition and projection. The behaviour of a conservative process after trace t only depends on the numbers of events in t, not on their order in t. Furthermore, each subset of the alphabet of a conservative process is non-disabling with respect to that process. With each process we associate a set of occurrences of events. A process defines a partial order on its set of occurrences. Vice versa, a partial order on a set of occurrences defines a process. If process T equals the process defined by the partial order corresponding to T then T is cubic. A sequence function for a cubic process describes a partial order

that is in accordance with the partial order defined by the process. It defines a cubic subprocess that can be interpreted as a restricted (clocked) behaviour of the original process. Sequence functions are also defined for systems consisting of cubic processes. Existence of a sequence function for such a system implies, under certain conditions, the absence of deadlock. Conditions for the existence of a sequence function for the system corresponding to a recursive component are given expressed in terms of the command of the component only. A system of cubic processes is said to have constant response time if there exists a sequence function for the system satisfying certain conditions.

Communication of values is described in terms of trace theory. Essentially, this is done by introducing symbols that are pairs consisting of a channel name and a value. Occurrence of pair $\langle c, m \rangle$ is interpreted as the passing of value m via channel c. In general, the communication behaviour of a process depends on the values that the process sends and receives. Processes for which this not the case are called data independent. Data independence can be expressed in terms of transparence. Data independent processes form the class of processes that are specified by split specifications. The Conjunction-Weave rule is formulated for this class. The projection of a data independent process on a channel set that is non-disabling with respect to the communication behaviour is data independent. Conditions implying the data independence of the composition of processes are formulated. They are easily verified if one introduces channel types, and distinguishes between input and output. Data independence allows one to discuss phenomena like deadlock and divergence in terms of communication behaviours only. A process is called channel order independent if its future behaviour does not depend on the order in which the channels have been used in the past. For processes that do not describe communication of values, e.g. communication behaviours, this definition equals the definition of conservativity. A new notation for specifications of data independent processes that are channel order independent as well is introduced.

Making a distinction between input and output, the program notation of trace theory is extended to include communication of values. Conditions are given that imply the process of a component to be data independent and channel order independent and its communication behaviour to be cubic.

Finally, a programming method is presented informally by means of examples. Among the examples given is the derivation of programs for acceptors of regular expressions. The programming method is based on the data independence and the channel order independence of processes, and the application of the Conjunction-Weave rule. Programs derived using this method describe systems that satisfy the conditions for networks given in section 0.0. We show that the derived programs can be proved to be correct. They satisfy the given specification, and their systems are free of deadlock and divergence. Furthermore, it is shown that the systems of the derived programs have constant response time. A number of analogies with sequential programming is mentioned.

Summarizing, we conclude that the material presented in the previous chapters provides adequate means to describe parallel computations, in particular the subclass we are mainly interested in. It also supports the reasoning in the programming method that is introduced.

References

[An,Cl,Fo,Mi] Anantharaman,T.S., E.M. Clarke, M.J. Foster, B. Mishra
Compiling path expressions into VLSI circuits
Distributed Computing **1** (1986), pp. 150–160

[Bi] Birkhoff, G.
Lattice Theory
American Mathematical Society, Providence, 1967
(AMS Colloquium Publications : vol. 25)

[Dij] Dijkstra, Edsger W.
Lecture Notes "Predicate transformers" (draft)
Eindhoven University of Technology, 1982
(EWD 835)

[Ee] Eemers, Henk
A description of communicating mechanisms and their behaviour
Eindhoven University of Technology, 1988
(Master's Thesis)

[Fl,Ul] Floyd, Robert W., Jeffrey D. Ullman
The Compilation of Regular Expressions into Integrated Circuits
Journal of the ACM, **29**, 3 (1982), pp.603–622

[Fo,Ku] Foster,M.J., H.T. Kung
Recognize regular languages with programmable building-blocks
VLSI 81 Very Large Scale Integration
ed. J.P. Gray
Academic Press, 1981, pp.75–84

[Ho] Hoare, C.A.R.
Communicating Sequential Processes
Prentice Hall, New York, 1985

[Ka] Kaldewaij, Anne
A Formalism for Concurrent Processes
Ph.D.-thesis
Eindhoven University of Technology, 1986

[Kö] König, D.
Theorie der endlichen und unendlichen Graphen
Chelsea, New York, 1950

[Ku] Kung, H.T.
Let's design algorithms for VLSI systems
Proc. 1st Caltech Conference, ed. C.L. Seitz
California Institute of Technology, Pasadena, 1979, pp. 65–90

[Re85] 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

[Re87] Rem, Martin
Trace theory and systolic computations
PARLE Parallel Architectures and Languages Europe
Volume I: Parallel Architectures
ed. J.W. de Bakker, A.J. Nijman, P.C. Treleaven
Springer, Berlin, 1987
(Lecture Notes in Computer Science: 258) pp. 14–33

[Ro,Tch] Robert, Yves, Maurice Tchuente
Réseaux Systoliques pour des Problèmes de Mots
R.A.I.R.O. Informatique théorique/Theoretical Informatics
19,2,1985, pp.107–123

[Sa] Salomaa, Arto
Computation and Automata
Cambridge University Press, 1985
(Encyclopedia of mathematics and its applications: 25)

[Sn] Snepscheut, Jan L.A. van de
Trace Theory and VLSI Design
Springer, Berlin, 1985
(Lecture Notes in Computer Science: 200)

[Ud] Udding, Jan Tijmen
On recursively defined set of traces
Eindhoven University of Technology, 1983
(THE Memorandum JTU0a)

[Ve85] Verhoeff, Tom
Notes on Delay-Insensitivity
Eindhoven University of Technology, 1985
(Master's Thesis)

[Ve86] Verhoeff, Tom
Private communication, 1986

[Ve88] Verhoeff, Tom
A Parallel Program That Generates the Möbius Sequence
Eindhoven University of Technology, 1988
(Computing Science Notes: 88/01)

[We] Wetering, Huub van de
Acceptors of regular sets
Eindhoven University of Technology, 1985
(Master's Thesis)

Index

Samenvatting

Parallelle berekeningen vormen het onderwerp van dit proefschrift. We behandelen de beschrijving van parallelle berekeningen met behulp van tracetheorie (een formalisme voor de beschrijving van parallelle processen ontwikkeld door Martin Rem ([Re85]), Jan L.A. van de Snepscheut ([Sn]) en Anne Kaldewaij ([Ka])) en een methode om programma's voor parallelle berekeningen af te leiden vanuit specificaties.

Onder parallelle berekeningen verstaan we netwerken van processoren of cellen die onderling waarden kunnen communiceren en die beschreven kunnen worden met processen. We richten ons daarbij voornamelijk op netwerken die als volgt gekarakteriseerd kunnen worden.

- het netwerk bestaat uit cellen die volgens een regelmatig patroon gerangschikt zijn (bijvoorbeeld een rechthoekig rooster of een boom)
- communicatie tussen cellen in het netwerk en tussen cellen en de omgeving van het netwerk vindt plaats via éénrichtingskanalen
- cellen zijn eenvoudig en communiceren via een vast aantal kanalen met buurcellen en/of de omgeving van het netwerk (vast betekent hier onafhankelijk van het totale aantal cellen)
- het communicatiegedrag van de cellen is onafhankelijk van de waarden die ze ontvangen en versturen
- cellen synchroniseren slechts op onderlinge communicaties (er is geen globale klok)

Netwerken die voldoen aan de eerste vier voorwaarden worden vaak systolisch genoemd (systolic arrays). Systolische netwerken hebben in het algemeen echter een globale klok voor de synchronisatie van de cellen.

Om parallelle berekeningen te beschrijven dient communicatie van waarden geformaliseerd te worden binnen de tracetheorie. Daartoe voeren we symbolen in die paren zijn bestaande uit een (kanaal)naam en een waarde. Het voorkomen van een paar $\langle c, m \rangle$ wordt geïnterpreteerd als het verzenden of ontvangen van waarde m via kanaal c. Een belangrijk aspect van een proces is de mate waarin de waarden die gecommuniceerd worden het communicatiegedrag van het proces bepalen. Een proces heet data

onafhankelijk als het communicatiegedrag van het proces niet afhangt van de waarden die het verstuurt en ontvangt. Het communicatiegedrag van een data onafhankelijk proces kan afzonderlijk beschreven worden. Een aantal eigenschappen zoals bijvoorbeeld divergentie hangt bij data onafhankelijke processen alleen af van het communicatiegedrag. Het communicatiegedrag kan ook invloed hebben op de waarden die gecommuniceerd worden. Indien de volgorde waarin een proces van zijn kanalen gebruik maakt geen invloed heeft op de waarden die gecommuniceerd worden, spreken we van een kanaalvolgorde onafhankelijk proces.

Bij de behandelde programmeermethode gaan we uit van specificaties van data onafhankelijke en kanaalvolgorde onafhankelijke processen. Gebruikmakend van de vorm van de specificaties van dergelijke processen leiden we er programma's uit af. De afleiding kenmerkt zich door het feit dat het communicatiegedrag en het verband tussen ontvangen en verstuurde waarden (input/output-relatie) onafhankelijk van elkaar behandeld kunnen worden. De afgeleide programma's zijn correct, d.w.z. ze voldoen aan hun specificatie en vertonen geen deadlock of divergentie. De afgeleide programma's beschrijven netwerken die voldoen aan de eerder vermelde voorwaarden. Bovendien hebben ze een constante responstijd.

We besluiten met een korte beschrijving van de inhoud van de hoofdstukken 1 t/m 5 in dit proefschrift.

In hoofdstuk 1 geven we een overzicht van tracetheorie. We voeren het begrip systeem in. Een systeem beschrijft een netwerk van processen en bestaat uit een alfabet (de externe kanalen van het netwerk) en een verzameling processen. Het externe proces van een systeem wordt gedefiniëerd als de compositie van de processen in de procesverzameling van het systeem geprojecteerd op het externe alfabet. Systemen kunnen worden samengesteld en worden geprojecteerd op een alfabet. De programmanotatie uit de tracetheorie wordt beschouwd als een middel om een bepaalde klasse van systemen te beschrijven. Het proces van een programma of component wordt gedefiniëerd als het externe proces van het systeem behorend bij het programma of de component. In het geval van een recursieve component geeft deze definitie op natuurlijke wijze het kleinste dekpunt van de recursieve vergelijking die gedefiniëerd wordt door de component. Dit stemt overeen met de keuze in [Sn] en [Ka].

In hoofdstuk 2 komen eerst nondeterminisme en divergentie aan de orde. We concentreren ons daarbij op het begrip transparantie uit [Ka]. Een aantal stellingen laat zien welke uitspraken omtrent transparantie gedaan kunnen worden bij samenstelling van processen en bij projectie van processen op een alfabet. Vervolgens komen beëindiging en deadlock aan de orde ([Ka]) en ten slotte voeren we de klasse van de conservatieve processen en de klasse van de cubische processen in ([Ve86]). De cubische processen vormen een deelklasse van de conservatieve processen. Een proces is conservatief als zijn toekomstig gedrag alleen afhangt van de aantallen gebeurtenissen in het verleden

en niet van de volgorde van die gebeurtenissen. De cubische processen zijn de processen die beschreven kunnen worden met een partiële ordening op voorkomens van gebeurtenissen ([Ve86]). Voor cubische processen en voor systemen bestaande uit cubische processen introduceren we sequence functies. Sequence functies beschrijven een beperkt (geklokt) gedrag van het proces of systeem. Met behulp van sequence functies definiëren we het begrip constante responstijd.

In hoofdstuk 3 modelleren we communicatie van waarden binnen tracetheorie. We voeren het begrip data onafhankelijkheid in en laten zien dat data onafhankelijkheid uitgedrukt kan worden in termen van transparantie. Data onafhankelijkheid van een proces blijft behouden bij projectie op een alfabet dat transparant is ten opzichte van het communicatiegedrag van het proces. Data onafhankelijke processen kunnen worden beschreven met zogenaamde gesplitste specificaties. Dit zijn specificaties waarbij het communicatiegedrag apart wordt beschreven. Met behulp van gesplitste specificaties formuleren we een conjunctie-weefregel voor data onafhankelijke processen. We laten zien dat bij beschouwingen over bijvoorbeeld divergentie en deadlock men zich bij data onafhankelijke processen kan beperken tot het communicatiegedrag van de processen. Een proces heet kanaalvolgorde onafhankelijk als het toekomstige gedrag van het proces niet afhangt van de volgorde waarin de kanalen zijn gebruikt. Deze definitie lijkt veel op de definitie van conservatieve processe. Het communicatiegedrag van een data onafhankelijk en kanaalvolgorde onafhankelijk proces is conservatief.

In hoofdstuk 4 breiden we de programmanotatie van tracetheorie uit zodat we in programma's communicatie van waarden kunnen beschrijven. Een aantal elementen in deze programmanotatie is ontleend aan CSP ([Ho]). In dit hoofdstuk maken we een onderscheid tussen input en output. Er worden voorwaarden gegeven waaronder het proces van een component data onafhankelijk en kanaalvolgorde onafhankelijk is en het communicatiegedrag cubisch is.

In hoofdstuk 5 presenteren we aan de hand van voorbeelden een programmeermethode. De methode is gebaseerd op de data onafhankelijkheid en de kanaalvolgorde onafhankelijkheid van de onderhavige processen en op het toepassen van de conjunctie-weefregel. De afgeleide programma's definiëren systemen die voldoen aan de eerder genoemde voorwaarden voor netwerken van processen. De afgeleide programma's zijn correct in de zin dat ze voldoen aan hun specificatie en dat de bijbehorende systemen vrij zijn van divergentie en deadlock. De wijze van programmeren vertoont een aantal analogieën met sequentiëel programmaren.

Curriculum vitae

naam	: Gerard Zwaan
15 december 1955	: geboren te Breda
20 juni 1974	: diploma Gymnasium β, Stedelijk Gymnasium, Breda
september 1976	: aanvang studie wiskunde aan de Technische Hogeschool Eindhoven
juni 1983 – mei 1984	: afstuderen onder leiding van dr. F. Eising met als onderwerp algoritmen ter bepaling van grootste rechterdelers van polynoommatrices
24 mei 1984	: doctoraal examen wiskunde, met lof, Technische Hogeschool Eindhoven
juni 1984 – mei 1988	: wetenschappelijk assistent bij de vakgroep Informatica van de Technische Hogeschool/Universiteit Eindhoven; onderzoek op het gebied van parallellisme onder leiding van prof. dr. M. Rem resulterend in dit proefschrift
vanaf juni 1988	: universitair docent bij de vakgroep Informatica van de Technische Universiteit Eindhoven; onderzoek op het gebied van systeemprogrammatuur onder leiding van prof. dr. F.E.J. Kruseman Aretz
huidig adres	: Faculteit Wiskunde en Informatica Technische Universiteit Eindhoven Postbus 513 5600 MB Eindhoven Nederland
e-mail	: wsinzwaan@heitue5.bitnet mcvax!eutws1!wsinswan

STELLINGEN

behorend bij het proefschrift

PARALLEL COMPUTATIONS

van

GERARD ZWAAN

Eindhoven
20 januari 1989

0 Als het gedrag van filosofen als volgt kan worden gekenmerkt

```
do true → think
        ; P(x)
        ; n:= n + 1
        ;if n mod 2 = 0 → V(d)
         [] n mod 2 = 1 → V(x); P(d); V(x)
         fi
        ; eat
        ; P(x)
        ; n:= n − 1
        ;if n mod 2 = 0 → V(d)
         [] n mod 2 = 1 → V(x); P(d); V(x)
         fi
od
```

waarin x en d binaire semaforen zijn, en als initieel $n = 0 \wedge x = 1 \wedge d = 0$ geldt, dan is het aantal etende filosofen te allen tijde even.

lit. – E.W. Dijkstra, *Hierarchical Ordering of Sequential Processes*, Acta Informatica 1 (1971), pp. 115–138
– E.W. Dijkstra, *A tutorial on the split binary semaphore*, 1979 (EWD 703)

1 Rubik's clock is een uitstekend middel om het begrip invariant aanschouwelijk te maken.

2 Een groot aantal graafalgoritmen laat zich eenvoudig afleiden door het gestelde probleem te herleiden tot een stelsel ongelijkheden waarvan de kleinste oplossing berekend dient te worden.

lit. – Joop van den Eijnde, *A derivation for the reachable vertices algorithm*, Eindhoven University of Technology, 1986 (Internal Memorandum JvdE 86/3)
– Gerard Zwaan, *Even and odd reachability*, Eindhoven University of Technology, 1987, (Internal Memorandum GZ 87/2)

3 Het aantal traces van $\mathrm{SEM}_4(a, b)$ ter lengte k, $k > 0$, is $2^{(k-1) \bmod 2} \cdot 3^{(k-1) \operatorname{div} 2}$.

4 De in hoofdstuk 5 van dit proefschrift beschreven parallelle programma's zijn op eenvoudige wijze als circuits te implementeren.

lit. – Anne Kaldewaij, *The translation of processes into circuits*, PARLE Parallel Architectures and Languages Europe, Volume I: Parallel Architectures, ed. J.W. de Bakker, A.J. Nijman, P.C. Treleaven, Springer Berlin 1987 (LNCS 258), pp. 195–212
– Jo Ebergen, *Translating programs into delay-insensitive circuits*, Ph.D.-thesis, Eindhoven University of Technology, 1987

5 Voor regelmatige berekeningen zijn sequence functies een effectief middel om uitspraken te doen over de voortgang en het real-time gedrag.

6 Ieder alfabet is non-disabling ten opzichte van een conservatief proces. Derhalve vallen bij conservatieve processen de begrippen transparantie en non-divergentie samen.

7 Voor processen wordt de "scheiding van data en control" geformaliseerd door de begrippen data onafhankelijkheid en kanaalvolgorde onafhankelijkheid.

8 De klasse van processen beschreven door restricted commands vormt een echte deelklasse van de klasse der reguliere, cubische processen.

9 Bij het automatisch genereren van het trefwoordenregister van een boek denke men aan het spreekwoord "Overdaad schaadt".

lit. – Donald E. Knuth, *The TEXbook*, Addison-Wesley, 1984

10 Bij hoogspringen en polsstokhoogspringen wordt, in tegenstelling tot andere onderdelen van de atletiek, de werkelijk geleverde prestatie niet gemeten.