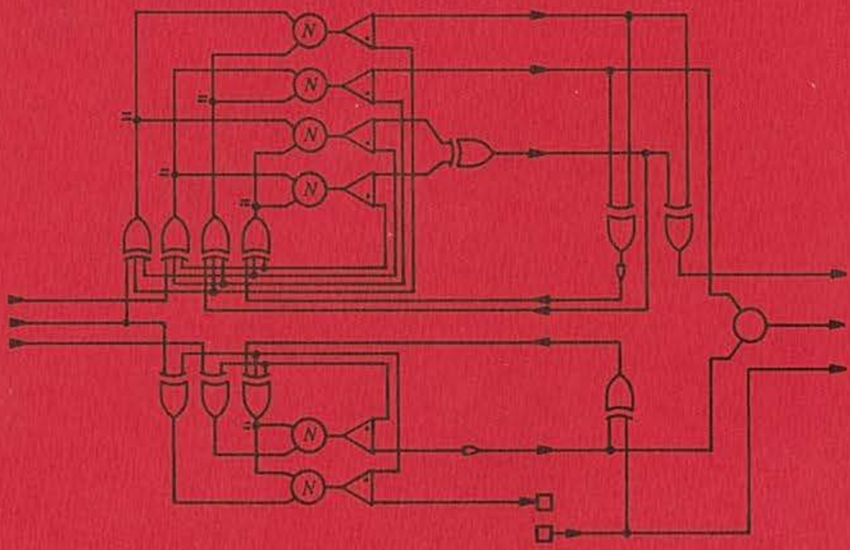
Translating Programs into Delay-Insensitive Circuits



Jo C. Ebergen

Stellingen

behorende bij het proefschrift

Translating Programs into Delay-Insensitive Circuits

Jo C. Ebergen

0. De formele en gestruktureerde wijze waarop vertragingsongevoelige circuits ontworpen kunnen worden zal hun testbaarheid verhogen.

1. De vraag of een willekeurig programma —waarin algemene recursie is toegestaan— vrij is van deadlock of livelock is onbeslisbaar.

   Literatuur: *Communicating Sequential Processes,* C.A.R. HOARE, Prentice-Hall, 1985.

2. Meervoudige staartrecursie kan het dupliceren van code voorkomen en leent zich voor een formele programmeermethode.

3. De in dit proefschrift voorgestelde programmanotatie biedt een goede basis voor het analyseren van de structurele complexiteit van problemen.

4. Er wordt nog te weinig aandacht besteed aan het ontwikkelen van ontwerpmethoden voor parallelle processen.

5. Schakeltheorie is minder geschikt voor het ontwerpen van vertragingsongevoelige circuits dan voor het ontwerpen van synchrone circuits.

6. Indien een component gespecificeerd door $E\upharpoonright$ wordt gedecomponeerd volgens de in dit proefschrift beschreven methode en E heeft de eigenschap

   $$(\mathbf{A}t : t \in \mathbf{t}E : Suc(t,E) \cap \mathbf{o}E = Suc(t\upharpoonright \mathbf{ext}E, E\upharpoonright) \cap \mathbf{o}E \;),$$

   dan is de decompositie vrij van deadlock.

   Literatuur: Dit proefschrift.

7. Laat de funktie $f : \mathfrak{T}_B \rightarrow \mathfrak{T}_B$ gedefinieerd zijn door

   $$f.R \;=\; (S \,\|\, s.R)\upharpoonright B,$$

   waarbij voor S en B geldt $S \in GC4$, $B = \mathbf{ext}S$ en waarin s is een herbenoemingsfunktie is zodanig dat $\mathbf{a}(s.R) = \mathbf{co}S$. Dan bestaat $\mu.f$ en er geldt $\mu.f \in C4$.

   Literatuur: Dit proefschrift en
   A Formalism for Concurrent Processes, A. KALDEWAIJ, Proefschrift, Technische Universiteit Eindhoven, 1986.

8. Een herkenner voor reguliere expressies zonder ϵ-cycli kan op een eenvoudige manier met vertragingsongevoelige componenten worden gerealiseerd.

Literatuur: Recognize Regular Languages with Programmable Building Blocks, M.J. FOSTER en H.T. KUNG, in: *VLSI 81* (JOHN P. GRAY, ed.), Academic Press, 1981, pp. 75-84.
The Compilation of Regular Expressions into Integrated Circuits, ROBERT W. FLOYD en JEFFREY D. ULLMAN, *Journal of the ACM,* **29** (1982), pp. 603-622.

9. Drachtigheidsdiagnostiek bij schapen door middel van real-time echografie is economisch onrendabel.

Literatuur: Accuracy of pregnancy diagnosis and prediction of foetal numbers in sheep with linear-array real-time ultrasound scanning, M.A.M. TAVERNE, M.C. LAVOIR, R. VAN OORD, en G.C. VAN DER WEYDEN, *The Veterinary Quaterly,* **7** , no. 4 (1985).

10. Een interdisciplinaire samenwerking tussen informatica en diergeneeskunde zal een vruchtbare toekomst tegemoet gaan.

Literatuur: Register van de burgelijke stand te Amsterdam.

Translating Programs
into
Delay-Insensitive Circuits

Translating Programs
into
Delay-Insensitive Circuits

PROEFSCHRIFT

TER VERKRIJGING VAN DE GRAAD VAN DOCTOR
AAN DE TECHNISCHE UNIVERSITEIT EINDHOVEN,
OP GEZAG VAN DE RECTOR MAGNIFICUS,
PROF. DR. F.N. HOOGE, VOOR EEN
COMMISSIE AANGEWEZEN DOOR HET COLLEGE VAN
DEKANEN IN HET OPENBAAR TE VERDEDIGEN OP
DINSDAG 13 OKTOBER 1987 TE 16:00 UUR

DOOR

JOSEPHUS CHRISTIANUS EBERGEN

GEBOREN TE LITH

Dit proefschrift is goedgekeurd
door de promotoren

Prof. dr. M. Rem

en

Prof. dr. C.E. Molnar.

To the Eindhoven VLSI Club

It is quite difficult to think about the code entirely in abstracto *without any kind of circuit.*
Alan M. Turing [44].

Contents

Chapter 0
Introduction

In 1938 Claude E. Shannon wrote his seminal article [41] entitled 'A Symbolic Analysis of Relay and Switching Circuits'. He demonstrated that Boolean algebra could be used elegantly in the design of switching circuits. The idea was to specify a circuit by a set of Boolean equations, to manipulate these equations by means of a calculus, and to realize this specification by a connection of basic elements. The result was that only a few basic elements, or even one element such as the 2-input NAND gate, suffice to synthesize any switching function specified by a set of Boolean equations. Shannon's idea proved to be very fertile and out of it grew a complete theory, called switching theory.

In this thesis we present a method for designing *delay-insensitive circuits.* The principal idea of this method is similar to that of Shannon's article: to design a circuit as a connection of basic elements and to construct this connection with the aid of a formalism. We construct such a circuit by translating programs satisfying a certain syntax. The result of such a translation is a connection of elements chosen from a finite set of basic elements. Moreover, this translation can be carried out in such a way that the number of basic elements in the connection is proportional to the length of the program. We formalize what it means that such a connection is a *delay-insensitive* connection.

Delay-insensitive circuits are a special type of circuits. We briefly describe their origins and how they are related to other types of circuits and design techniques. The most common distinction usually made between types of circuits is the distinction between *synchronous circuits* and *asynchronous circuits.* Synchronous circuits are circuits that perform their (sequential) computations based on the successive pulses of the clock. For the design of these circuits many techniques have been developed and are described by means of switching theory [29, 23]. The correctness of synchronous systems relies on the boundedness of delays in elements and wires. The satisfaction of these delay requirements cannot be guaranteed under all circumstances, and for this reason problems can crop up in the design of synchronous systems. In order to avoid these problems interest arose in the design of circuits without a clock. Such circuits have generally been called *asynchronous* circuits.

The design of asynchronous circuits has always been and still is a difficult subject. Several techniques for the design of such circuits have been developed and are discussed in, for example, [29, 23, 47]. For special types of such circuits formalizations and other design techniques have been proposed and discussed. David E. Muller has given a formalization of a type of circuits which he coined by the name of *speed-independent* circuits. An account of this formalization is given in [30].

From a design discipline that was applied in the Macromodules project [4, 5] at Washington University in St. Louis the concept of a special type of circuit evolved which was given the name *delay-insensitive* circuit. It was realized that a proper formalization of this concept was needed in order to specify and design such circuits in a well-defined manner. A formalization of the concept of a delay-insensitive circuit was later given by Jan Tijmen Udding in [45]. For the design and specification of delay-insensitive circuits several methods have been developed based on, for example, Petri Nets and techniques derived from switching theory [13, 33].

Recently, Alain Martin has proposed some interesting design techniques for circuits of which the functional operation is unaffected by delays in elements or wires [25, 26]. The techniques are based on the compilation of CSP-like programs into connections of basic elements. It is, however, not yet clear whether these techniques can be completely automated and to which type of programs they can be applied and which not. The techniques presented in this thesis exhibit a similarity with the techniques applied by Alain Martin.

Another name that is frequently used in the design of asynchronous circuits is *self-timed systems*. This name has been introduced by Charles L. Seitz in [40] in order to describe a method of system design without making any reference to timing except in the design of the self-timed elements. Other techniques and formalisms applied in the design and verification of (special types of) asynchronous circuits, but less related to the work presented in this thesis, are described in [10, 31, 22, 15].

The reasons to design delay-insensitive systems are manifold. Before we explain each of these reasons, we briefly sketch some of the motives of the first computer designers to incorporate a clock in their design. For them this was not an obvious decision, since most mechanical calculating machinery before the use of electronic devices was designed without a clock. The first widely disseminated reports on computer design that advocated the use of a clock are the reports on the EDVAC [34, 27, 1]. These reports have had a large influence on the design of computers. The basic logical organization of most computers nowadays has not changed much from the organization that was advocated then by Von Neumann and his associates.

The motives for incorporating a clock in their design were twofold. The first and most important reason was that all computations had to be done in purely sequential fashion: parallelism was explicitly forbidden (both to avoid the high cost of additional circuitry and to avoid complexity in the design). It turned out that for the realization of such computations the use of a clock had considerable advantages: the clock could, for example, be used to dictate the successive steps of the computations. The second reason was that various memory devices used at that time were dynamic devices, i.e. memory elements whose contents had to be refreshed regularly. Refreshing was usually done by means of clock pulses. Since, for this reason, a clock was already present for those devices, it could be used for other purposes as well.

In the report on the ACE [43], written shortly after the first report on the EDVAC, Alan Turing is more explicit about the use of a clock in the design and mentions it as one of twelve essential components. In [44] he motivates this choice as follows.

> We might say that the clock enables us to introduce a discreteness into time, so that time for some purposes can be regarded as a succession of instants instead of a continuous flow. A digital machine must essentially deal with discrete objects, and in the case of the ACE this is made possible by the use of a clock. All other digital computing machines except for human and other brains that I know of do the same. One can think up ways of avoiding it, but they are very awkward.

REMARK. Here, we also remark that at the time of the reports on the EDVAC and the ACE, i.e. in 1945-47, Boolean algebra was still considered of little use in the design of computer circuits [12]. It took more than ten years after Shannon's article before Boolean algebra was accepted and proved to be a useful formalism in the practical design of synchronous systems.
□

One reason why there has always been an interest in asynchronous systems is that synchronous systems tend to reflect a *worst-case* behavior, while asynchronous systems tend to reflect an *average-case* behavior. A synchronous system is divided into several parts, each of which performs a specific computation. At a certain clock pulse, input data are sent to each of these parts and at the next clock pulse the output data, i.e. the results of the computations, are sampled and sent to the next parts. The correct operation of such an organization is established by making the clock period larger than the worst-case delay for any subcomputation. Accordingly, this worst-case behavior may be disadvantageous in comparison with the average-case behavior of asynchronous systems.

Another more important reason for designing delay-insensitive systems is the so-called *glitch phenomenon.* A glitch is the occurrence of metastable behavior in circuits. Any computer circuit that has a number of stable states also has metastable states. When such a circuit gets into a metastable state, it can remain there for an indefinite period of time before it resolves into a stable state. For example, it may stay in the metastable state for a period larger than the clock period. Consequently, when a glitch occurs in a synchronous system, erroneous data may be sampled at the time of the clock pulses. In a delay-insensitive system it does not matter whether a glitch occurs: the computation is delayed until the metastable behavior has disappeared and the element has resolved into a stable state. Among the frequent causes for glitches are, for example, the asynchronous communications between independently clocked parts of a system.

The first mention of the glitch problem appears to date back to 1952 (cf. [2]). The first publication of experimental results of the glitch problem and a broad recognition of the fundamental nature of the problem came only after 1973 [3, 19] due to the pioneering work on this phenomenon at the Washington University in St. Louis.

A third reason is due to the effects of *scaling*. This phenomenon became prominent with the advent of integrated circuit technology. Because of the improvements of this technology, circuits could be made smaller and smaller. It turned out, however, that if all characteristic dimensions of a circuit are scaled down by a certain factor, including the clock period, delays in long wires do not scale down proportional to the clock period [28, 40]. As a consequence, some VLSI designs when scaled down may no longer work properly anymore, because delays for some computations have become larger than the clock period. Delay-insensitive systems do not have to suffer from this phenomenon if the basic elements are chosen small enough so that the effects of scaling are negligible with respect to the functional behavior of these elements [42].

The fourth reason is the clear separation between functional and physical correctness concerns that can be applied in the design of delay-insensitive systems. The correctness of the behavior of basic elements is proved by means of physical principles only. The correctness of the behavior of connections of basic elements is proved by mathematical principles only. Thus, it is in the design of the basic elements only that considerations with respect to delays in wires play a role. In the design of a connection of basic elements no reference to delays in wires or elements is made. This does not hold for synchronous systems where the functional correctness of a circuit also depends on timing considerations. For example, for a synchronous system one has to calculate the worst-case delay for each part of the system and for any computation in order to satisfy the requirement that this delay must be smaller than the clock period.

As a last reason, we believe that the translation of parallel programs into delay-insensitive circuits offers a number of advantages compared to the translation of parallel programs into synchronous systems. In this thesis a method is presented with which the synchronization and communication between parallel parts of a system can be programmed and realized in a natural way.

The method presented in this thesis for designing delay-insensitive circuits is briefly described as follows. We call an abstraction of a circuit a *component*;

components are specified by programs written in a notation based on *trace theory*. These programs are called *commands* and can be considered as an extension of the notation for regular expressions. Any component represented by a command can also be represented by a regular expression, i.e. it is also a *regular* component. The notation for commands, however, allows for a more concise representation of a component due to the additional programming primitives in this notation. These extra programming primitives include operations to express parallelism, tail recursion (for representing finite state machines), and projection (for introducing internal symbols).

Based on trace theory we formalize the concepts of *decomposition* of a component and of *delay-insensitivity*. The decomposition of a component is intended to represent the realization of that component by means of a connection of circuits. Delay-insensitivity is formalized in the definitions of *DI decomposition* and of *DI component*. A DI decomposition represents a realization of a component by means of a delay-insensitive connection of circuits. A DI component represents a circuit that communicates in a delay-insensitive way with its environment. It turns out that the definition of DI component is equivalent with Udding's formalization of a delay-insensitive circuit. One of the fundamental theorems in this thesis is that DI decomposition and decomposition are equivalent if all components involved are DI components. We also present some theorems that are helpful in finding decompositions of a component.

Based on the definition of DI component, we develop a number of so-called *DI grammars*, i.e. grammars for which any command generated by these grammars represents a (regular) DI component. With these grammars the language $\mathcal{L}_4$ of commands is defined. We show that any regular DI component represented by a command in the language $\mathcal{L}_4$ can be decomposed in a syntax-directed way into a finite set $\mathbf{B}$ of basic DI components and so-called *CAL components*. CAL components are also DI components. Consequently, the decomposition into these components is, by the above mentioned theorem, also a DI decomposition.

The set of all CAL components is, however, not finite. In order to show that a decomposition into a finite basis of components exists, we discuss two decompositions of CAL components: one decomposition into the finite basis $\mathbf{B}0$ and one decomposition into the finite basis $\mathbf{B}1$. The decomposition of CAL components into the finite basis $\mathbf{B}1$ is in general not a DI decomposition, since not every component in $\mathbf{B}1$ is a DI component. This decomposition can, however, be realized in a simple way if so-called *isochronic forks* are used in the realization. The decomposition of CAL components into the basis $\mathbf{B}0$ is an interesting but difficult subject. Since every component in $\mathbf{B}0$ is a DI component, every decomposition into $\mathbf{B}0$ is therefore also a DI decomposition. We briefly describe a general procedure, which we conjecture to be correct, for the decomposition of CAL components into the basis $\mathbf{B}0$.

The decomposition method can be described as a syntax-directed translation of commands in $\mathcal{L}_4$ into commands of the basic components in $\mathbf{B}0$ or $\mathbf{B}1$. Consequently, the decomposition method is a constructive method and can be completely automated. Moreover, we show that the result of the complete decomposition of any component expressed in $\mathcal{L}_4$ can be linear in the length of the command, i.e. the number of basic elements in the resulting connection is proportional to the length of the command.

Although many regular DI components can be expressed in the language $\mathcal{L}_4$, which is the starting point of the translation method, probably not every regular DI component can be expressed in this way. We indicate, however, that for any regular DI component there exists a decomposition into components expressed in $\mathcal{L}_4$, which can then each be translated by the method presented.

The formalism we use in this thesis is called trace theory. Trace theory was inspired by Hoare's CSP [17, 18] and developed by a number of people at the University of Technology in Eindhoven. It has proven to be a good tool in reasoning about parallel computations [36, 37, 42, 20] and, in particular, about delay-insensitive circuits [45, 46, 38, 39, 16, 21].

This thesis is organized as follows. In Chapter 1 the basic notions of trace theory are briefly presented. In Chapter 2 we present the program notation for commands and give a number of examples in which we illustrate the specification of a component by means of a command. In Chapter 3 the fundamental concepts of decomposition and delay-insensitivity are defined. The recognition of DI components is the subject of Chapter 4 in which several attribute grammars are presented, all of which generate commands representing DI components. The proofs of this chapter are given in the appendices. By means of these grammars, we subsequently describe a syntax-directed decomposition method in Chapters 5 and 6. Chapter 7 contains a number of examples and suggestions about optimizing the general decomposition method of Chapters 5 and 6. In Chapter 7 we also discuss the issues involved in the decomposition of any regular DI component into a finite basis of components. We conclude with some remarks. Each chapter has many examples to illustrate the subject matter in a simple way.

In this thesis we have tried to pursue the aim of delay-insensitive design as far as possible, i.e. to postpone correctness arguments based on delay-assumptions as long as possible, in order to see what sort of designs such a pursuit would lead to. In this approach our first concern has been the correctness of the designs and only in the second place have we addressed their efficiency.

0.1. Notational Conventions

The following notational conventions are used in the thesis.

Universal quantification is denoted by

$$(\mathbf{A}x: D(x): P(x)).$$

It is read as 'for all x satisfying $D(x)$, $P(x)$ holds'. The expression $D(x)$ denotes the domain over which the quantified identifier x ranges. Instead of one quantified identifier, we may also take two or more quantified identifiers. Existential quantification is denoted by

$$(\mathbf{E}x: D(x): P(x)).$$

It is read as 'there exists an x satisfying $D(x)$ for which $P(x)$ holds'.

The notations $R(i: 0 \leq i < n)$ and $E(i,j: 0 \leq i,j < n)$ denote arrays of elements $R.i$, $0 \leq i < n$, and $E.i.j$, $0 \leq i < n \wedge 0 \leq j < n$, respectively. Sometimes these arrays are referred to as vector $R(i: 0 \leq i < n)$ and matrix $(Ei,j: 0 \leq i,j < n)$ respectively.

In some cases functional application is denoted by the period, it is left-associative, and it has highest priority of all binary operations. For example, the function f applied to the argument a is denoted by $f.a$. The array $E(i,j: 0 \leq i,j < n)$ can be considered as a function E defined on the domain $0 \leq i < n \wedge 0 \leq j < n$. The function E applied to i, $0 \leq i < n$, yields the array $E.i(j: 0 \leq j < n)$; subsequent application to j, $0 \leq j < n$, yields the element $E.i.j$. Since function application is left-associative, we have $E.i.j = (E.i).j$. The notation for functional application is taken from [9].

Let *op* denote an associative binary operation with identity element *id*. Continued application of the operation *op* over all elements $a.i$ satisfying the domain restrictions $D(i)$ is denoted by

$$(op\ i: D(i): a.i).$$

For example, we have

$$(+i: 0 \leq i < 4: a.i.) = a.0 + a.1 + a.2 + a.3.$$

If domain $D(i)$ is empty, then

$$(op\ i: D(i): a.i) = id.$$

For example, we have $(+i: 0 \leq i < 0: a.i) = 0$.
(Notice that universal and existential quantification can also be expressed as $(\wedge x: D(x): P(x))$ and $(\vee x: D(x): P(x))$ respectively.) The notation $(\mathbf{N}i: D(i): P(i))$ denotes the number of i's satisfying $D(i)$ for which $P(i)$ holds.

Most proofs in the thesis have a special notational layout. For example, if we prove $P0 \Rightarrow P2$ by first showing $P0 \Rightarrow P1$ and then $P1 \equiv P2$, this is denoted by

$$
\begin{array}{l}
\quad P0 \\
\Rightarrow \{\text{hint why } P0 \Rightarrow P1\} \\
\quad P1 \\
= \{\text{hint why } P1 \equiv P2\} \\
\quad P2.
\end{array}
$$

This notation is taken from [7].

Chapter 1

Trace Theory

1.0. Introduction

In this chapter we present a brief introduction to trace theory. It contains the definitions and properties relevant to the rest of this thesis.

The first part summarizes previous results from trace theory. For a more thorough exposition on this part the reader is referred to [42, 36, 20]. In Sections 1.1.0 and 1.1.1 we define trace structures and the basic operations on them. Section 1.1.2 contains a number of properties of these operations. In Section 1.1.3 we define a program notation for expressing commands. Commands specify trace structures, and can be considered as generalizations of regular expressions.

The second part contains new material. In Section 1.2 we give a detailed presentation of *tail recursion.* Tail recursion can be used to express finite state machines in a concise and simple way. Moreover, tail recursion can be used conveniently to prove properties about programs. For these reasons the command language is extended with tail recursion.

We conclude with Section 1.3 in which we show a number of programs written in the command language.

1.1. TRACE STRUCTURES AND COMMANDS

1.1.0. Trace structures

A *trace structure* is a pair $<B,X>$, where B is a finite set of symbols and $X \subseteq B^*$. The set B^* is the set of all finite-length sequences of symbols from B. A finite sequence of symbols is called a *trace*. The empty trace is denoted by ϵ. Notice that $\varnothing^* = \{\epsilon\}$. For a trace structure $R = <B,X>$, the set B is called the *alphabet* of R and denoted by $\mathbf{a}R$; the set X is called the *trace set* of R and denoted by $\mathbf{t}R$.

NOTATIONAL CONVENTION. In the following, trace structures are denoted by the capitals R,S, and T; traces are denoted by the lower-case letters r, s, t, u, and v; alphabets are denoted by the capitals A and B; symbols are usually denoted by the lower-case letters with exception of r, s, t, u, and v.
□

1.1.1. Operations on trace structures

The definitions and notations for the operations *concatenation, union, repetition, (taking the) prefix-closure, projection,* and *weaving* of trace structures are as follows.

$$\begin{aligned}
R;S &= <\mathbf{a}R \cup \mathbf{a}S \;,\; \mathbf{t}R\,\mathbf{t}S> \\
R|S &= <\mathbf{a}R \cup \mathbf{a}S \;,\; \mathbf{t}R \cup \mathbf{t}S> \\
[R] &= <\mathbf{a}R \;,\; (\mathbf{t}R)^*> \\
\mathbf{pref}\,R &= <\mathbf{a}R \;,\; \{s \,|\, (\mathbf{E}t::st \in \mathbf{t}R)\}> \\
R{\upharpoonright}A &= <\mathbf{a}R \cap A \;,\; \{t{\upharpoonright}A \,|\, t \in \mathbf{t}R\}> \\
R\|S &= <\mathbf{a}R \cup \mathbf{a}S \;,\; \{t \in (\mathbf{a}R \cup \mathbf{a}S)^* \,|\, t{\upharpoonright}\mathbf{a}R \in \mathbf{t}R \wedge t{\upharpoonright}\mathbf{a}S \in \mathbf{t}S\}>,
\end{aligned}$$

where $t{\upharpoonright}A$ denotes the trace t projected on A, i.e. the trace t from which all symbols not in A have been deleted. Concatenation of sets is denoted by juxtaposition and $(\mathbf{t}R)^*$ denotes the set of all finite-length concatenations of traces in $\mathbf{t}R$.

The operations concatenation, union, and repetition are familiar operations from formal language theory. We have added three operations: (taking the) prefix-closure, projection, and weaving.

The **pref** operator constructs prefix-closed trace structures. A trace structure R is called *prefix-closed* if $\mathbf{pref}\,R = R$ holds. Later, we use prefix-closed and non-empty trace structures for the specification of components. We call a trace structure R *prefix-free* if

$$(\mathbf{A}s,t : s \in \mathbf{t}R \wedge st \in \mathbf{t}R : t = \epsilon)$$

holds, i.e. no trace in $\mathbf{t}R$ is a proper prefix of another trace in $\mathbf{t}R$.

The projection operator allows us to introduce internal symbols which are abstracted away by means of projection. These internal symbols can be used conveniently for a number of purposes, as we will see in the subsequent chapters.

The weave operation constructs trace structures whose traces are weaves of traces from the constituent trace structures. Notice that common symbols must match, and, accordingly, weaving expresses instantaneous synchronization. The set of symbols on which this synchronization takes place is the intersection of the alphabets.

The *successor set* of t with respect to trace structure R, denoted by $Suc(t,R)$, is defined by

$$Suc(t,R) = \{b \mid tb \in \mathbf{t}\,\mathbf{pref}\,R\}.$$

Finally we define a partial order $\leqslant$ on trace structures by

$$R \leqslant S \equiv \mathbf{a}R = \mathbf{a}S \wedge \mathbf{t}R \subseteq \mathbf{t}S.$$

1.1.2. Some properties

Below, a number of properties are given for the operations just defined. The proofs can be found in [42, 20].

PROPERTY 1.1.2.0. *For the operations on trace structures we have:*
Concatenation is associative and has $<\varnothing,\{\epsilon\}>$ *as identity.*
Union is commutative, associative, and has $<\varnothing,\varnothing>$ *as identity.*
Weaving is commutative, associative, and has $<\varnothing,\{\epsilon\}>$ *as identity.*
If we consider prefix-closed non-empty trace structures only, union has $<\varnothing,\{\epsilon\}>$ *as identity.*
□

PROPERTY 1.1.2.1. *Union and weaving are idempotent, i.e. for any* R *we have* $R|R=R$ *and* $R\|R=R$.
□

PROPERTY 1.1.2.2. (Distribution properties of ; and |.)
For any R,S *and* T *we have*

$$R;(S|T) = (R;S)|(R;T)$$

$$(S|T);R = (S;R)|(T;R)$$

□

PROPERTY 1.1.2.3. (Distribution properties of $\upharpoonright$.)
For any R,S,B, and A we have

$$\begin{aligned}(R;S)\upharpoonright B &= (R\upharpoonright B);(S\upharpoonright B)\\ (R|S)\upharpoonright B &= (R\upharpoonright B)|(S\upharpoonright B)\\ [R]\upharpoonright B &= [R\upharpoonright B]\\ (\mathbf{pref}\,R)\upharpoonright B &= \mathbf{pref}(R\upharpoonright B)\\ R\upharpoonright A\upharpoonright B &= R\upharpoonright(A\cap B)\\ (R\|S)\upharpoonright B &= (R\upharpoonright B)\|(S\upharpoonright B) \quad \text{if } \mathbf{a}R\cap\mathbf{a}S\subseteq B.\end{aligned}$$

□

PROPERTY 1.1.2.4. (Distribution properties of **pref**.)

$$\begin{aligned}\mathbf{pref}(R|S) &= (\mathbf{pref}\,R)|(\mathbf{pref}\,S)\\ \mathbf{pref}(R;S) &= \mathbf{pref}(R;(\mathbf{pref}\,S)).\end{aligned}$$

□

PROPERTY 1.1.2.5. *A weave of non-empty prefix-closed trace structures is non-empty and prefix-closed.*
□

PROPERTY 1.1.2.6. *For any R, S, A, and B with* $\mathbf{a}R\cap\mathbf{a}S\subseteq B$ *and* $A\subseteq\mathbf{a}R$ *we have*

$$(R\|S)\upharpoonright A = (R\|(S\upharpoonright B))\upharpoonright A.$$

PROOF. We observe

$$\begin{aligned}&(R\|S)\upharpoonright A\\ =\ &\{\text{Prop. 1.1.2.3, calc.}\}\\ &(R\|S)\upharpoonright(A\cup B)\upharpoonright A\\ =\ &\{\text{Prop. 1.1.2.3, } \mathbf{a}R\cap\mathbf{a}S\subseteq B\}\\ &((R\upharpoonright(A\cup B))\,\|\,(S\upharpoonright(A\cup B)))\upharpoonright A\\ =\ &\{\text{def. of projection}\}\\ &((R\upharpoonright(A\cup B))\,\|\,(S\upharpoonright\mathbf{a}S\upharpoonright(A\cup B)))\upharpoonright A\\ =\ &\{\mathbf{a}R\cap\mathbf{a}S\subseteq B \wedge A\subseteq\mathbf{a}R\text{ , Prop. 1.1.2.3., calc.}\}\\ &((R\upharpoonright(A\cup B))\,\|\,(S\upharpoonright B\upharpoonright(A\cup B)))\upharpoonright A\\ =\ &\{\text{Prop. 1.1.2.3, } \mathbf{a}R\cap\mathbf{a}S\subseteq B\text{, calc.}\}\end{aligned}$$

$$(R\|(S\uparrow B))\restriction(A\cup B)\restriction A$$

$$= \{\text{Prop. 1.1.2.3, calc.}\}$$

$$(R\|(S\uparrow B))\restriction A.$$

□

PROPERTY 1.1.2.7. *Let the trace structures $R.k$, $0\leqslant k<n$, satisfy* $\mathbf{a}(R.k)\cap\mathbf{a}(R.l)\subseteq B$ *for* $k\neq l \wedge 0\leqslant k,l<n$. *We have*

$$(\|k:0\leqslant k<n:R.k)\restriction B = (\|k:0\leqslant k<n:(R.k)\restriction B).$$

□

Property 1.1.2.7 is a generalization of the last law of property 1.1.2.3.

1.1.3. Commands and state graphs

A trace structure is called a *regular trace structure* if its trace set is a regular set, i.e. a set generated by some regular expression. A *command* is a notation similar to regular expressions for representing a regular trace structure.

Let U be a sufficiently large set of symbols. The characters b, with $b\in U$, ϵ, and $\varnothing$ are called *atomic commands.* They represent the atomic trace structures $<\{b\},\{b\}>$, $<\varnothing,\{\epsilon\}>$, and $<\varnothing,\varnothing>$ respectively. Every atomic command and every expression for a trace structure constructed from the atomic commands and operations defined in Section 1.1.1 is called a *command.* In this expression parentheses are allowed. For example, the expression $(a\|b);c$ is a command and represents the trace structure $<\{a,b,c\},\{abc,bac\}>$.

NOTATIONAL CONVENTION. In the following, commands are denoted by the capital E's. The alphabet and the trace set of the trace structure represented by command E are denoted by $\mathbf{a}E$ and $\mathbf{t}E$ respectively. In order to save on parentheses, we stipulate the following priority rules for the operations just defined. Unary operators have highest priority. Of the binary operators in Section 1.1.1, weaving has highest priority, then concatenation, then union, and finally projection.
□

PROPERTY 1.1.3.0. *Every command represents a regular trace structure.*
□

A command of the form $\mathbf{pref}(E)$, where E is an atomic command different from $\varnothing$, or E is constructed from atomic commands different from $\varnothing$ and the operations concatenation (;), union (|), or repetition ([ ]) is called a *sequential command.*

PROPERTY 1.1.3.1. *Every sequential command represents a prefix-closed non-empty regular trace structure.*
□

Syntactically different commands can express the same trace structure. We have, for example,

$$\mathbf{pref}[a;c] \parallel \mathbf{pref}[b;c] = \mathbf{pref}[a\|b;c]$$

$$\mathbf{pref}[a;c] \parallel \mathbf{pref}[c;b] = \mathbf{pref}(a;c;[a\|b;c]).$$

In this thesis, every directed graph of which the arcs are labelled with non-empty trace structures or commands and that has one node denoting the initial state is called a *state graph.* The nodes are called *the states of the state graph* and are usually labelled with lower-case q's. The initial state is denoted by an encircled node. An example of a state graph is given in Figure 1.1.0.

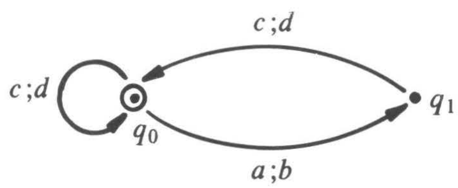


FIGURE 1.1.0. A state graph.

With each state graph we associate a trace structure in the following way. Let the state transition from state q_i to state q_j be labelled with non-empty trace structure $S.i.j$, $0 \leq i,j < n$. If there is no state transition between state q_i and state q_j then $S.i.j = <\varnothing, \varnothing>$. State q_0 is the initial state. The trace structure that corresponds to this state graph is given by $\mathbf{pref}<B,X>$, where

$$B = (\cup i,j : 0 \leq i,j < n : \mathbf{a}(S.i.j)) \text{ and}$$

$$X = \{t \mid t \text{ is a finite concatenation of traces of successive trace structures in the state graph starting in } q_0\}.$$

More precisely, let the trace structures $R.k.i$, $0 \leq k \wedge 0 \leq i < n$, be defined by

$$R.0.i = <B, \{\epsilon\}>, \text{ and}$$

$$R.(k+1).i = (|j : 0 \leq j < n : S.i.j; R.k.j), \text{ for all } i, \ 0 \leq i < n.$$

The trace structure corresponding to the state graph is defined by

$$\mathbf{pref}(|k : k \geq 0 : R.k.0).$$

Notice that $\mathbf{t}(R.k.i)$ contains all traces of concatenations of k successive trace structures in the state graph starting in state q_i. The trace structure corresponding to the state graph of Figure 1.1.0, for example, can be represented by $\mathbf{pref}[c;d \mid a;b;c;d]$.

Above we defined for each state graph the trace structure that corresponds to this state graph. For a given structure we can also construct a specific state graph in which the states of the state graph match the *states of the trace structure.* For this purpose, we first define the states of a trace structure.

For a trace structure R we define the relation $\sim_R$ on traces of $\mathbf{t}\,\mathbf{pref}\,R$ by

$$t \sim_R s \equiv (\mathbf{A}r :: tr \in \mathbf{t}R \equiv sr \in \mathbf{t}R).$$

The relation $\sim_R$ is an equivalence relation and the equivalence classes are called the *states of trace structure R.* The state containing t is denoted by $[\![t]\!]$. For example, for $R = \mathbf{pref}[a \| b;c]$ the states are given by $[\![\epsilon]\!]$, $[\![a]\!]$, $[\![b]\!]$, and $[\![ab]\!]$. In this thesis we keep to prefix-closed non-empty trace structures. Every state of these trace structures is also a so-called final state.

The relation $\sim_R$ is also a right congruence, i.e. for all r, s, and t with $tr \in \mathbf{t}\,\mathbf{pref}\,R$ and $sr \in \mathbf{t}\,\mathbf{pref}\,R$ we have

$$s \sim_R t \Rightarrow sr \sim_R tr.$$

Because $\sim_R$ is a congruence relation, we can represent a trace structure by a state graph in which the nodes are labelled with the states of R and the arcs are labelled with the atomic commands of the symbols of R. There is an arc labelled x, with $x \in \mathbf{a}R$, from state $[\![t]\!]$ to state $[\![r]\!]$ of R iff $[\![tx]\!] = [\![r]\!]$. The state graph obtained in this way for trace structure $R = \mathbf{pref}[a \| b;c]$ is given in Figure 1.1.1.

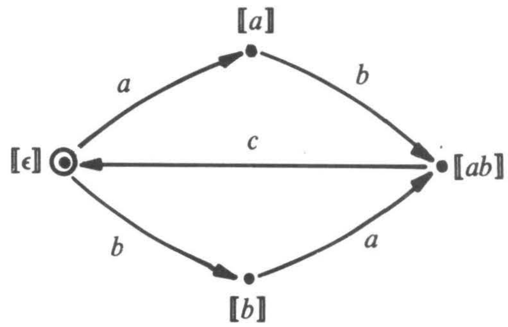


FIGURE 1.1.1. State graph for $\mathbf{pref}[a \| b;c]$.

1.2. Tail recursion

1.2.0. Introduction

From formal language theory we know that every finite state machine can be represented by a regular expression, and thus also by a command. In the language of commands that we have defined thus far, finite state machines cannot always be expressed as succinctly as we would like. This is one of the reasons to introduce tail recursion. We show that there is a simple correspondence between a finite state machine and a tail-recursive expression. Moreover, tail recursion can be used conveniently to prove properties about programs by means of *fixpoint induction.*

In the following sections, we first convey the idea of tail recursion by means of an introductory example. Then we briefly summarize some results of lattice theory. In the subsequent sections these results are used to define the semantics of tail recursion. We conclude by extending our command language with tail recursion.

1.2.1. An introductory example

Consider the finite state machine given by the state graph of Figure 1.2.0.

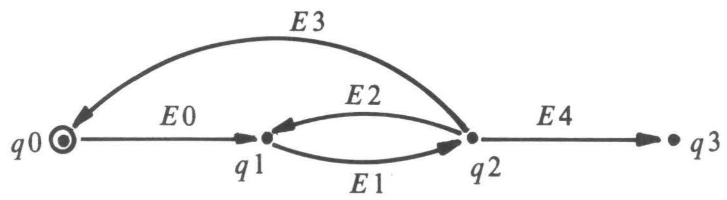


FIGURE 1.2.0. A state graph.

The states of this state graph are labeled with $q0$, $q1$, $q2$, and $q3$, where $q0$ is the initial state. The state transitions are labeled with the non-empty commands $E0$, $E1$, $E2$, $E3$, and $E4$. With this state graph the trace structure **pref**$<B,X>$ is associated, where

$$B = \mathbf{a}E0 \cup \mathbf{a}E1 \cup \mathbf{a}E2 \cup \mathbf{a}E3 \cup \mathbf{a}E4 \quad \text{and}$$

$$X = \{t \mid t \text{ is a finite concatenation of traces of successive commands in the state graph starting in } q0\}$$

Possible commands representing this trace structure are

$$\mathbf{pref}(E0;E1;[(E2\,|\,E3;E0);E1];E4) \quad \text{and}$$

$\mathbf{pref}(E0;[E1;(E2\,|\,E3;E0)];E1;E4)$.

The trace structure $\mathbf{pref}<B,X>$ can also be expressed as a least fixpoint of a so-called *tail function.* The tail function *tailf* corresponding to the state graph of Figure 1.2.0 is defined on a vector $R(i:0\leqslant i<n)$ of prefix-closed non-empty trace structures with alphabet B by

$$tailf.R.0 = \mathbf{pref}(E0;R.1)$$

$$tailf.R.1 = \mathbf{pref}(E1;R.2)$$

$$tailf.R.2 = \mathbf{pref}(E2;R.1\,|\,E3;R.0\,|\,E4;R.4)$$

$$tailf.R.3 = \mathbf{pref}(R.3).$$

(Recall that functional application is denoted by a period. The period has highest priority of all binary operations and is left-associative.) The least fixpoint of this tail function exists and is denoted by $\mu.tailf$. This fixpoint is a vector of trace structures for which component 0 satisfies

$$\mu.tailf.0 = \mathbf{pref}<B,X>.$$

We prove this in Section 1.2.4.

Since the tail function *tailf* is defined by commands, we call $\mu.tailf.0$ a command as well. The conditions under which $\mu.tailf.0$ is called a command, for an arbitrary tail function *tailf*, are given Section 1.2.5.

In the above we have given three commands for $\mathbf{pref}<B,X>$, i.e. two without tail recursion and one with tail recursion. Notice that in the two commands without tail recursion $E0$ and $E1$ occur twice, while in the tail function *tailf*, with which the third command $\mu.tailf.0$ is given, each command of the state graph occurs exactly once.

1.2.2. Lattice theory

The following definitions and theorems summarize some results from lattice theory. No proofs are given. For a more thorough introduction to lattice theory we refer to [0].

Let $(L,\leqslant)$ be a partially ordered set and V a subset of L. Element R of L is called the *greatest lower bound* of V, denoted by $(\sqcap S: S\in V: S)$, if

$$(\mathbf{A}S: S\in V: R\leqslant S) \wedge (\mathbf{A}T: T\in L \wedge (\mathbf{A}S: S\in V: T\leqslant S) : T\leqslant R).$$

Element R of L called the *least upper bound* of V, denoted by $(\sqcup S: S\in V: S)$, if

$$(\mathbf{A}S: S\in V: S\leqslant R) \wedge (\mathbf{A}T: T\in L \wedge (\mathbf{A}S: S\in V: S\leqslant T) : R\leqslant T).$$

We call $(L,\leqslant)$ a *complete lattice* if each subset of L has a greatest lower bound and a least upper bound. A complete lattice has a *least element,* denoted by $\bot$, for which we have $\bot=(\sqcup R: R\in\varnothing: R)$.

A sequence $R(k:k\geqslant 0)$ of elements of L is called an *ascending chain* if $(\mathbf{A}k:k\geqslant 0:\ R.k\leqslant R.(k+1))$.

Let f be a function from L to L. An element R of L is called a *fixpoint* of f if $f.R=R$. The function f is called *upward continuous* if for each ascending chain $R(k:k\geqslant 0)$ in L we have

$$f.(\sqcup k:k\geqslant 0:R.k) = (\sqcup k:k\geqslant 0:f.(R.k)).$$

The function f^k, $k\geqslant 0$, from L to L is defined by

$$f^0.R = R \text{ and } f^{k+1}.R=f.(f^k.R) \text{ for } k\geqslant 0 \text{ and } R\in L.$$

A predicate P defined on L is called *inductive,* if for each ascending chain $R(k:k\geqslant 0)$ in L we have

$$(\mathbf{A}k:k\geqslant 0:P(R.k)) \Rightarrow P(\sqcup k:k\geqslant 0:R.k).$$

THEOREM 1.2.2.0. (Knaster-Tarski)
An upward continuous function f defined on a complete lattice $(L,\leqslant)$ with least element $\perp$ has a least fixpoint, denoted by $\mu.f$, and $\mu.f=(\sqcup k:k\geqslant 0:f^k.\perp)$.
□

THEOREM 1.2.2.1. (Fixpoint induction)
Let f be an upward continuous function on the complete lattice $(L,\leqslant)$ with least element $\perp$. If P is an inductive predicate defined on L for which $P(\perp)$ holds and $P(R)\Rightarrow P(f.R)$ for any $R\in L$, i.e. f maintains P, then $P(\mu.f)$ holds.
□

1.2.3. Tail functions

We call a function, *tailf* say, a *tail function* if it is defined by

$$tailf.R.i = \mathbf{pref}(|j:0\leqslant j<n:\ S.i.j;R.j)$$

for vectors $R(i:0\leqslant i<n)$ of trace structures, where $S(i,j:0\leqslant i,j<n)$ is a matrix of trace structures. Consequently, a tail function is uniquely determined by the matrix $S(i,j:0\leqslant i,j<n)$ of trace structures. Let this matrix S be fixed for the next sections and let $A=(\cup i,j:0\leqslant i,j<n:\ \mathbf{a}(S.i.j))$.

We define $\mathcal{T}^n(A)$ as the set of all vectors $R(i:0\leqslant i<n)$ of prefix-closed non-empty trace structures with alphabet A. For elements R and T of $\mathcal{T}^n(A)$ we define the partial order $\leqslant$ by

$$R\leqslant T \equiv (\mathbf{A}i:0\leqslant i<n:\ \mathbf{t}(R.i)\subseteq\mathbf{t}(T.i)).$$

Furthermore we define the vector $\perp_n(A)$ by

$$\perp_n(A).i = <A,\{\epsilon\}>, \text{ for all } i,\ 0\leqslant i<n.$$

Theorem 1.2.3.0. *$(\mathcal{T}^n(A),\leqslant)$ is a complete lattice with least element $\perp_n(A)$.*

Proof. For each non-empty subset V of $\mathcal{T}^n(A)$ we have

$$(\sqcup R: R\in V: R).i \;=\; (|R: R\in V: R.i)$$
$$(\sqcap R: R\in V: R).i \;=\; <A, (\cap R: R\in V: \mathbf{t}(R.i))>,$$

for $0\leqslant i<n$. For $V=\varnothing$ we have

$$(\sqcup R: R\in\varnothing: R) \;=\; \perp_n(A) \text{ and}$$
$$(\sqcap R: R\in\varnothing: R).i \;=\; <A, A^*>, \text{ for all } i,\; 0\leqslant i<n.$$

□

By definition, the function *tailf* is defined on $\mathcal{T}^n(A)$. Furthermore, we define condition $P0$ by

$$P0:\quad (\mathbf{A}i: 0\leqslant i<n: (\mathbf{E}j: 0\leqslant j<n: \mathbf{t}(S.i.j)\neq\varnothing)).$$

We have

Theorem 1.2.3.1. *Let $P0$ hold. The function tailf is a function from $\mathcal{T}^n(A)$ to $\mathcal{T}^n(A)$ and is upward continuous.*

Proof. From the definition of *tailf* and $P0$ follows that $tailf.R\in\mathcal{T}^n(A)$, for any $R\in\mathcal{T}^n(A)$.

Let $R(k: k\geqslant 0)$ be an ascending chain of elements from $\mathcal{T}^n(A)$. We observe for all i, $0\leqslant i<n$,

$$tailf.(\sqcup k: k\geqslant 0: R.k).i$$
$$= \{\text{def. } tailf\}$$
$$\mathbf{pref}(|j: 0\leqslant j<n: S.i.j;(\sqcup k: k\geqslant 0: R.k).j)$$
$$= \{\text{def. } \sqcup\}$$
$$\mathbf{pref}(|j: 0\leqslant j<n: S.i.j;(|k: k\geqslant 0: R.k.j))$$
$$= \{\text{distribution Prop. 1.1.2.2}\}$$
$$\mathbf{pref}(|k,j: 0\leqslant j<n \wedge k\geqslant 0: S.i.j;R.k.j)$$
$$= \{\text{distribution Prop. 1.1.2.4}\}$$
$$(|k: k\geqslant 0: \mathbf{pref}(|j: 0\leqslant j<n: S.i.j;R.k.j))$$
$$= \{\text{def. } tailf\}$$
$$(|k: k\geqslant 0: tailf.(R.k).i)$$
$$= \{\text{def. } \sqcup\}$$
$$(\sqcup k: k\geqslant 0: tailf.(R.k)).i\;.$$

Consequently, $tailf.(\sqcup k: k \geq 0: R.k) = (\sqcup k: k \geq 0: tailf.(R.k))$.

(Notice that in the above proof we did not use the property that the chain $R(k: k \geq 0)$ was ascending.)
□

1.2.4. Least fixpoints of tail functions

From Theorems 1.2.2.0, 1.2.3.0, and 1.2.3.1 we derive

THEOREM 1.2.4.0. *If* $P0$ *holds, then tailf has a least fixpoint, denoted by* $\mu.tailf$, *and*

$$\mu.tailf = (\sqcup k: k \geq 0: tailf^k.\bot_n(A)).$$

□

The least fixpoint $\mu.tailf$ can be related to the trace structure corresponding to a state graph as follows. Consider a state graph with n states q_i, $0 \leq i < n$. If $\mathbf{t}(S.i.j) \neq \varnothing$, then there is a state transition from state q_i to state q_j labeled $S.i.j$. Let the trace structures $R.k.i$ for $0 \leq i < n \wedge k \geq 0$ be defined by

$$R.0.i = \langle A, \{\epsilon\} \rangle, \text{ and}$$

$$R.(k+1).i = (|j: 0 \leq j < n: S.i.j; R.k.j) \text{ for all } i, \ 0 \leq i < n.$$

In other words, $\mathbf{t}\,\mathbf{pref}(R.k.i)$ is the prefix-closure of all trace structures that can be formed by concatenating k successive trace structures starting from state q_i. The trace structure corresponding to the state graph is defined by $\mathbf{pref}(|k: k \geq 0: R.k.0)$. We prove that $\mu.tailf.i = \mathbf{pref}(|k: k \geq 0: R.k.i)$, i.e. $\mu.tailf.i$ is the prefix-closure of all finite concatenations of successive trace structures starting in state q_i.

THEOREM 1.2.4.1. *Let* $P0$ *hold. For all i,* $0 \leq i < n$,

$$\mu.tailf.i = \mathbf{pref}(|k: k \geq 0: R.k.i).$$

PROOF. By Theorem 1.2.4.0 we infer that $\mu.tailf$ exists and can be written as $(\sqcup k: k \geq 0: tailf^k.\bot_n(A))$.

We first prove that $tailf^k.\bot_n(A).i = \mathbf{pref}(R.k.i)$, $0 \leq i < n$, by induction to k.

Base. For $k = 0$ we have by definition

$$tailf^0.\bot_n(A).i = \langle A, \{\epsilon\} \rangle, \quad 0 \leq i < n.$$

Step. We observe for $0 \leqslant i < n$,

$$tailf^{k+1}.\perp_n(A).i$$
$$= \{\text{def. of } tailf^{k+1}\}$$
$$tailf.\ (tailf^k.\perp_n(A)).i$$
$$= \{\text{def. of } tailf\}$$
$$\mathbf{pref}(|j{:}\ 0 \leqslant j < n{:}\ S.i.j\,;tailf^k.\perp_n(A).j)$$
$$= \{\text{induction hypothesis for } k\}$$
$$\mathbf{pref}(|j{:}\ 0 \leqslant j < n{:}\ S.i.j;\ \mathbf{pref}(R.k.j))$$
$$= \{\text{distribution Prop. 1.1.2.4}\}$$
$$\mathbf{pref}(|j{:}\ 0 \leqslant j < n{:}\ S.i.j;R.k.j)$$
$$= \{\text{def. } R.(k+1).i\}$$
$$\mathbf{pref}(R.\ (k+1).i)\ .$$

Subsequently, we derive for all i, $0 \leqslant i < n$,

$$\mu.tailf.i$$
$$= \{\text{Theorem 1.2.4.0}\}$$
$$(\sqcup{:}\ k \geqslant 0{:}\ tailf^k.\perp_n(A)).i$$
$$= \{\text{def. } \sqcup\}$$
$$(|k{:}\ k \geqslant 0{:}\ tailf^k.\perp_n(A).i)$$
$$= \{\text{see above}\}$$
$$(|k{:}\ k \geqslant 0{:}\ \mathbf{pref}(R.k.i))$$
$$= \{\text{distribution Prop. 1.1.2.4}\}$$
$$\mathbf{pref}(|k{:}\ k \geqslant 0{:}\ R.k.i).$$

□

1.2.5. Commands extended

We extend the definition of commands with tail recursion. We stipulate that a tail function can also be specified by a matrix $E(i,j{:}\ 0 \leqslant i,j < n)$ of commands. When we write such a tail function, as we did in Section 1.2.1, we adopt the convention to omit alternatives $\varnothing;R.j$ and to abbreviate alternatives $\epsilon;R.j$ to $R.j$, for $0 \leqslant j < n$. The condition $P0$ is now formulated by

$$P1{:}\quad (\mathbf{A}i{:}\ 0 \leqslant i < n{:}\ (\mathbf{E}j{:}\ 0 \leqslant j < n{:}\ \mathbf{t}(E.i.j) \neq \varnothing)).$$

Every atomic command and every expression for a trace structure constructed with atomic commands and operations defined in Section 1.1.1 or tail recursion, i.e. with $\mu.tailf.0$ where $P1$ holds for *tailf*, is called an *extended command.*

If a tail function *tailf* is defined by a matrix $E(i,j: 0 \leqslant i,j < n)$ of commands for which $P1$ holds, and the commands of this matrix E are constructed with the operations concatenations (;), union (|), or repetition ([ ]) and the atomic commands, then we call $\mu.tailf.i$, $0 \leqslant i < n$, an *extended sequential command.* Every sequential command is also an *extended sequential command.* With these definitions of extended commands Property 1.1.3.0 and 1.1.3.1 also hold, i.e. we have

PROPERTY 1.2.5.0. *Every extended command represents a regular trace structure.*
□

PROPERTY 1.2.5.1. *Every extended sequential command represents a prefix-closed non-empty regular trace structure.*
□

Whenever in the remainder of this thesis we refer to commands or sequential commands we mean from now on extended commands or extended sequential commands respectively.

In the following, we also adopt the convention to define a tail function corresponding to a state graph in such a way that $\mu.tailf.0$ represents the trace structure associated with this state graph.

REMARK. For later purposes, we remark that every prefix-closed non-empty regular trace structure R can also be represented by a sequential command, even when the alphabet is larger than the set of symbols that occur in the trace set. To construct this command we first take a finite state machine that represents the regular trace set. Then we add state transitions and states that are unreachable from the initial state. We label these state transitions with symbols that occur in the alphabet but do not occur in the trace set. The tail function corresponding to this finite state machine satisfies $\mu.tailf.0=R$. For example, the trace structure $<\{a\},\{\epsilon\}>$ can be represented by $\mu.tailf.0$, where

$$tailf.R.0 = \mathbf{pref}(R.0)$$

$$tailf.R.1 = \mathbf{pref}(a;R.0).$$

□

1.3. Examples

The following examples illustrate that a trace structure can be expressed by many syntactically different commands. Sometimes a command can be rewritten, using rules from a calculus, into a different command that represents the same trace structure. Sometimes more complicated techniques are necessary to show that two commands express the same trace structure. For both cases we give examples. The freedom in manipulating the syntax of commands will become important later for two reasons. First, we will then be interested in trace structures that satisfy properties which can be verified syntactically and, second, in Chapters 5 and 6 we present a translation method for commands which is syntax-directed. Accordingly, by manipulating the syntax of a command we can influence the result of the syntactical check and the translation in a way that suits our purposes best.

Example 1.3.0. Every sequential command can be rewritten into the form $\mu.\mathit{tailf}.0$, where the tail function *tailf* is defined with atomic commands only. For example, the command $\mathbf{pref}(a;[b;(c\,|\,d;e)];f)$ can be rewritten into $\mu.\mathit{tailf}.0$, where

$$\begin{aligned}
\mathit{tailf}.R.0 &= \mathbf{pref}(a;R.1)\\
\mathit{tailf}.R.1 &= \mathbf{pref}(b;R.2\,|\,f;R.4)\\
\mathit{tailf}.R.2 &= \mathbf{pref}(c;R.1\,|\,d;R.3)\\
\mathit{tailf}.R.3 &= \mathbf{pref}(e;R.1)\\
\mathit{tailf}.R.4 &= \mathbf{pref}(R.4).
\end{aligned}$$

□

Example 1.3.1. The trace structure $\mathit{count}_n(a,b)$, $n>0$, is specified by

$$<\{a,b\},\ \{t\in\{a,b\}^*\,|\,(\mathbf{A}r,s:t=rs:\ 0\leqslant r\mathbf{N}a-r\mathbf{N}b\leqslant n)\}>,$$

where $s\mathbf{N}x$ denotes the number of x's in s. Symbol a can be interpreted as an increment and symbol b as a decrement for a counter. The value $t\mathbf{N}a-t\mathbf{N}b$ denotes the count of this counter after trace t. Any trace of a's and b's for which the count stays within the bounds 0 and n is a trace of $\mathit{count}_n(a,b)$.

There exist many commands for $\mathit{count}_n(a,b)$. For $n=1$, we have $\mathit{count}_n(a,b)=\mathbf{pref}[a;b]$. For $n\geqslant 1$, we give three equations from which a number of commands for $\mathit{count}_n(a,b)$ can be derived

(*i*) $\mathit{count}_n(a,b) = \mu.\mathit{tailf}_n.0$,

$$\begin{aligned}
\text{where } \mathit{tailf}_n.R.0 &= \mathbf{pref}(a;R.1)\\
\mathit{tailf}_n.R.i &= \mathbf{pref}(a;R.(i+1)\,|\,b;R.(i-1)), \text{ for } 0<i<n,\\
\mathit{tailf}_n.R.n &= \mathbf{pref}(b;R.(n-1)).
\end{aligned}$$

(*ii*) $count_{n+1}(a,b) = \mathbf{pref}[a;x] \parallel count_n(x,b) \upharpoonright \{a,b\}.$

(*iii*) $count_{2n+1}(a,b) = \mathbf{pref}[(a \mid y;b);(x;a \mid b)] \parallel count_n(x,y) \upharpoonright \{a,b\}.$

Techniques to prove these equations can be found in [36, 42, 20, 11]. As far as we know there are no simple transformations from one equation to the other.

With the first equation we can express $count_n(a,b)$ by a sequential command of length $\Theta(n)$. With (*ii*) we can express $count_n(a,b)$ by a weave of n sequential commands of constant length. With (*iii*), however, we can express $count_n(a,b)$ by a weave of $\Theta(\log n)$ sequential commands of constant length.

□

EXAMPLE 1.3.2. An n-place 1-bit buffer, denoted by $bbuf_n(a,b)$ is specified by

$$\begin{aligned}&<\{a0,a1,b0,b1\}\\&,\{t \mid (\mathbf{A}r,s : rs=t : 0 \leqslant r\mathbf{N}\{a0,a1\} - r\mathbf{N}\{b0,b1\} \leqslant n\\&\qquad\qquad \wedge\ r\upharpoonright\{b0,b1\} \preccurlyeq r\upharpoonright\{a0,a1\})\}\\&>,\end{aligned}$$

where $s \preccurlyeq t$ denotes that s is a prefix of t apart from a renaming of b into a. For $bbuf_3(a,b)$ we have

$$\begin{aligned}bbuf_3(a,b) = (\ &\mathbf{pref}[a0;x0 \mid a1;x1]\\ \parallel\ &\mathbf{pref}[x0,y0 \mid x1;y1]\\ \parallel\ &\mathbf{pref}[y0;b0 \mid y1;b1]\\ )&\upharpoonright\{a0,a1,b0,b1\}.\end{aligned}$$

A proof for this equation can be found in [11].

□

REMARK. It has been argued in [14] that regular expressions would be inconvenient for expressing counter-like components such as counters and buffers. As we have seen, the extension of regular expressions with a weave operator and projection effectively eliminates any such inconveniences.

□

Chapter 2

Specifying Components

2.0. Introduction

This chapter adresses the specification of components, which may be viewed as abstractions of circuits. Components are specified by prefix-closed, non-empty *directed trace structures*. In this thesis we shall keep to regular components, i.e. to regular directed trace structures. In a directed trace structure four types of symbols are distinghuished: *inputs, outputs, internal symbols of the component,* and *internal symbols of the environment*. In Section 2.1 we define directed trace structures and generalize the results of the previous chapter. Directed trace structures can be represented by *directed commands*. In Section 2.2 we explain how a directed trace structure prescribes all possible communication behaviors between a component and its *environment* at their mutual boundary. A number of basic components are then specified by means of directed commands. Section 2.3 contains a number of examples of specifications that will be used in later chapters.

2.1. Directed trace structures and commands

A *directed trace structure* is a quintuple $<B0,B1,B2,B3,X>$, where $B0$, $B1$, $B2$, and $B3$ are sets of symbols and $X \subseteq (B0 \cup B1 \cup B2 \cup B3)^*$. For a directed trace structure $R = <B0,B1,B2,B3,X>$ we give below the names and notations for the various alphabets and the trace set of R.

set	*name*	*notation*
$B0$	input alphabet of R	$\mathbf{i}R$
$B1$	output alphabet of R	$\mathbf{o}R$
$B2$	environment's internal alphabet of R	$\mathbf{en}R$
$B3$	component's internal alphabet of R	$\mathbf{co}R$
$B0 \cup B1$	external alphabet of R	$\mathbf{ext}R$
$B2 \cup B3$	internal alphabet of R	$\mathbf{int}R$
$B0 \cup B1 \cup B2 \cup B3$	alphabet of R	$\mathbf{a}R$
X	trace set of R	$\mathbf{t}R$

The operations defined on (undirected) trace structures are extended to directed trace structures as follows. For the input alphabet we have

$$\mathbf{i}(R;S) = \mathbf{i}R \cup \mathbf{i}S$$
$$\mathbf{i}(R|S) = \mathbf{i}R \cup \mathbf{i}S$$
$$\mathbf{i}[R] = \mathbf{i}R$$
$$\mathbf{i}\,\mathbf{pref}\,R = \mathbf{i}R$$
$$\mathbf{i}(R\!\upharpoonright\! A) = \mathbf{i}R \cap A$$
$$\mathbf{i}(R\|S) = \mathbf{i}R \cup \mathbf{i}S.$$

The other alphabets are defined similarly. The definitions for the trace sets remain the same as in Section 1.1.1. All properties of Section 1.1.2 are also valid for directed trace structures, where $<\varnothing,\varnothing>$ and $<\varnothing,\{\epsilon\}>$ are replaced by $<\varnothing,\varnothing,\varnothing,\varnothing,\varnothing>$ and $<\varnothing,\varnothing,\varnothing,\varnothing,\{\epsilon\}>$ respectively.

For a tail function *tailf* defined by matrix $S(i,j\!:\!0\leqslant i,j<n)$ of directed trace structures we define $A0, A1, A2$ and $A3$ by

$$A0 = (\cup i,j\!:\!0\leqslant i,j<n\!:\ \mathbf{i}(S.i.j))$$
$$A1 = (\cup i,j\!:\!0\leqslant i,j\leqslant n\!:\ \mathbf{o}(S.i.j))$$
$$A2 = (\cup i,j\!:\!0\leqslant i,j<n\!:\ \mathbf{en}(S.i.j))$$
$$A3 = (\cup i,j\!:\!0\leqslant i,j<n\!:\ \mathbf{co}(S.i.j)).$$

Let $\mathcal{T}^n(A0,A1,A2,A3)$ be the set of all prefix-closed non-empty directed trace structures R, with $\mathbf{i}R=A0$, $\mathbf{o}R=A1$, $\mathbf{en}R=A2$, and $\mathbf{co}R=A3$. By definition, the function *tailf* is defined on $\mathcal{T}^n(A0,A1,A2,A3)$. All results of Sections 1.2.3 and 1.2.4, with the appropriate replacements, hold for directed trace structures as well.

Directed commands are defined similar to (undirected) commands, with one exception for projection. There are six types of *directed atomic commands*; they are listed below together with the directed trace structure they represent.

directed atomic command	*directed trace structure*
$b?$	$<\{b\},\varnothing,\varnothing,\varnothing,\{b\}>$
$b!$	$<\varnothing,\{b\},\varnothing,\varnothing,\{b\}>$
$?b!$	$<\varnothing,\varnothing,\{b\},\varnothing,\{b\}>$
$!b?$	$<\varnothing,\varnothing,\varnothing,\{b\},\{b\}>$
ϵ	$<\varnothing,\varnothing,\varnothing,\varnothing,\{\epsilon\}>$
$\varnothing$	$<\varnothing,\varnothing,\varnothing,\varnothing,\varnothing>$,

Here $b \in U$, and U is a sufficiently large set of symbols. Every directed atomic command and every expression for a directed trace structure constructed from directed atomic commands and the operations concatenation (;), union (|), repetition ([ ]), prefix-closure (**pref**), weaving (||), or tail recursion ($\mu.tailf.0$, where $P1$ holds for *tailf*) is called a *directed command*. In a directed command parentheses are allowed. Any directed command of the form **pref**(E) where E is a directed atomic command different from $\varnothing$, or E is constructed with the operations concatenation (;), union (|), or repetition ([ ]) and directed atomic commands different from $\varnothing$ is called a *directed sequential command*. If a tail function *tailf* is defined by matrix $E(i,j: 0 \leq i,j < n)$ of directed commands, for which $P1$ holds, and if every directed command in this matrix E is a directed atomic command or is constructed with the operations concatenation (;), union (|), or repetition ([ ]) and directed atomic commands, then $\mu.tailf.i$, $0 \leq i < n$, is also called a *directed sequential command*.

Projection is used as follows in directed commands. If E is a directed command representing the directed trace structure R, then $E\upharpoonright$ is a directed command representing the directed trace structure $R\upharpoonright \mathbf{ext}R$. For example, we have

$$
\begin{aligned}
&(\,\mathbf{pref}[a?;!x?;b!] \\
&\parallel \mathbf{pref}[c?;!x?;d!] \\
&)\upharpoonright \\
= \; &\mathbf{pref}(a?\|c?;[(b!;a?)\|(d!;c?)]),
\end{aligned}
$$

where = denotes equality of directed trace structures.

2.2. Specifications

2.2.0. Introduction

A component is specified by a prefix-closed, non-empty, directed trace structure R with $\mathbf{int}R = \varnothing$ and $\mathbf{i}R \cap \mathbf{o}R = \varnothing$. The external alphabet of R contains all terminals of the component by which it can communicate with the environment. A communication action at a terminal is represented by the name of that terminal. The trace set R contains all communication behaviors that may take place between the component and its environment.

A communication behavior evolves by the production of communication actions. A communication action may be produced either by the component or by the environment. The sets $\mathbf{i}R, \mathbf{o}R$, and $\mathbf{t}R$ specify when which communication action may be produced and by whom. Let the current communication behavior be given by the trace $t \in \mathbf{t}R$, and let $tb \in \mathbf{t}R$, i.e. $b \in Suc(t,R)$. If $b \in \mathbf{i}R$, then the environment may produce a next communication action b; if $b \in \mathbf{o}R$, then the component may produce a next communication action b. These are also the only rules for the production of inputs and outputs for environment and component respectively.

Because the directed trace structure R specifies the behavior of both component and its environment, we speak of component R and environment R. The role of component and environment can be interchanged by reflecting R:

DEFINITION 2.2.0.1. The *reflection* of R, denoted by $\overline{R}$, is defined by

$$\overline{R} = <\mathbf{o}R,\ \mathbf{i}R,\ \mathbf{co}R,\ \mathbf{en}R,\ \mathbf{t}R>.$$

□

Operationally speaking, each external symbol b of R corresponds to a terminal of a circuit, and each occurrence of b in a trace of R corresponds to a voltage transition at that terminal. By convention we shall assume in this thesis that initially the voltage levels at the terminals are low, unless stated otherwise. The set of terminals constitutes the *boundary* between circuit and environment, which, for most components, is considered to be fixed. In the next chapter we discuss a special class of components, the so-called DI components, whose boundaries may be considered to be flexible.

In the following subsections, a number of components are specified by directed commands. For each of these components we also give a pictorial representation, called a *schematic*.

2.2.1. WIRE components

There are two WIRE components. The specifications and schematics of these components are given in Figure 2.0.

pref$[a?;b!]$

pref$[b!;a?]$

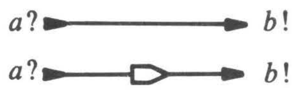


FIGURE 2.2.0. Two WIRE components.

A WIRE component describes the transmission of a signal from terminal to terminal, i.e. from boundary to boundary. We consider the boundaries of WIRE components to be flexible. All other components are considered to have a fixed boundary (for the time being).

Notice that both WIRE components have the same behavior except for a difference in initial states. For the WIRE component **pref**$[a?;b!]$ the environment initially produces a transition. For the WIRE component **pref**$[b!;a?]$ initially the component produces a transition. This difference in initial states (or the production of initial transitions) is depicted by an open arrow head in a schematic. We shall use this convention also in some of the following schematics. The components are, apart from a renaming, each other's reflection.

Operationally speaking, a WIRE component corresponds to a physical wire. Notice that there is always at most one transition propagating along this wire according to our interpretation of a specification.

2.2.2. CEL components

A k-CEL component, $k>0$, is specified by

$$(\| i: 0 \leq i < k: E.i), \text{ where}$$

$$\text{either } E.i = \mathbf{pref}[a.i?;b!] \text{ or } E.i = \mathbf{pref}[b!;a.i?], \quad 0 \leq i < n.$$

Notice that for $k=1$ a k-CEL component boils down to a WIRE component. A specification and schematic of a 4-CEL component are given in Figure 2.2.1.

$$\mathbf{pref}[b!;a.0?]$$
$$\| \mathbf{pref}[a.1?;b!]$$
$$\| \mathbf{pref}[b!;a.2?]$$
$$\| \mathbf{pref}[a.3?;b!]$$

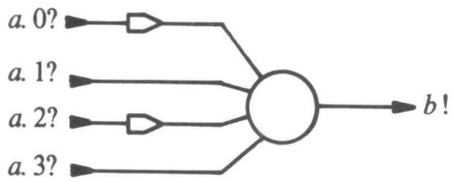


FIGURE 2.2.1. A CEL component.

Notice that here we have drawn open arrow heads on the inputs $a.0$ and $a.2$ of the CEL component denoting that initially a transition has already occurred on these inputs.

Schematics for other k-CEL components, $k>1$, are given similarly. A CEL component performs the primitive operation of synchronization. It can be represented by several directed commands: recall that

$$\mathbf{pref}[a?;c!] \| \mathbf{pref}[b?;c!] = \mathbf{pref}[a?\|b?;c!]$$

$$\mathbf{pref}[a?;c!] \| \mathbf{pref}[c!;b?] = \mathbf{pref}(a?;c!;[a?\|b?;c!]).$$

REMARK. The CEL components are generalizations of the Muller *C*-element named after D.E. Muller [32].
□

2.2.3. RCEL and NCEL components

The specification and schematic of the RCEL component with 2 replicated inputs are given in Figure 2.2.2.

$$\mathbf{pref}[(a?;d!)^2 \\ \mid (b?;e!)^2 \\ \mid (a?;d!\|c!)^2 \,\|\, (b?;e!\|c!)^2 \\ ]$$

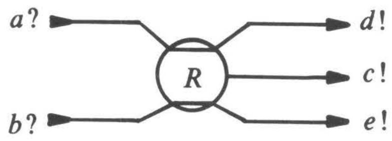


FIGURE 2.2.2. An RCEL component.

Here, E^2 denotes $E;E$. The specification of the RCEL component with one replicated input is given by $\mathbf{pref}[(a?;d!)^2 \mid (a?;d!\|c!)^2\|(b?;c!)^2]$ and depicted similarly.

The specification and schematic of the NCEL component is given in Figure 2.2.3.

$$\mathbf{pref}\,[(a?)^2 \mid (b?)^2 \mid (a?\|b?;c!)^2]$$

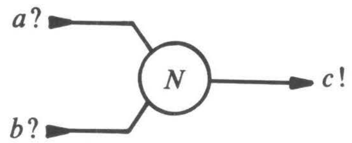


FIGURE 2.2.3. An NCEL component.

A component specified by $\mathbf{pref}[(b?)^2 \mid (a?\|b?;c!)^2]$ is also called an NCEL component and depicted analogously. (The letter N originates from the property that an NCEL component is *not* a DI component, as we will see later.)

2.2.4. FORK components

The k-FORK components, $k>0$, are specified by the reflections of the k-CEL components. A specification and schematic of a 4-FORK component are given in Figure 2.2.4.

pref[a?;b. 0!]
|| **pref**[b. 1!;a?]
|| **pref**[a?;b. 2!]
|| **pref**[b. 3!;a?]

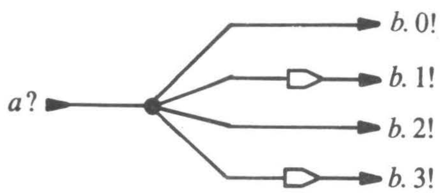


FIGURE 2.2.4. A FORK component.

Schematics for other k-FORK components, $k>1$, are given similarly. A FORK component performs the primitive operation of duplication.

2.2.5. XOR components

A k-XOR component, $k>0$, is specified by

(*i*) **pref**[E] or (*ii*) **pref**(b!;[E]),

where $E = (|i:0 \leq i<k: a.i?;b!)$.

Notice that 1-XOR components are WIRE components. In Figure 2.2.5 the two schematics for the two 4-XOR components are given.

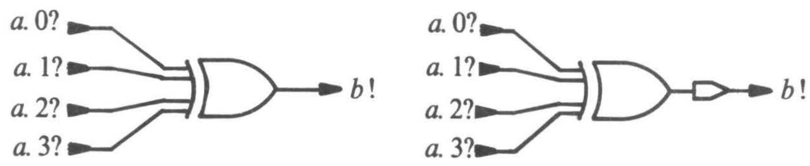


FIGURE 2.2.5. Two 4-XOR components.

Schematics for other k-XOR components, $k>1$, are depicted similarly.

2.2.6. TOGGLE component

The specification and schematic of the TOGGLE component are depicted in Figure 2.2.6.

pref$[a?;b!;a?;c!]$

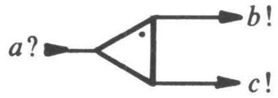


FIGURE 2.2.6. The TOGGLE component.

The TOGGLE component determines the parity of the input occurrences.

2.2.7. SEQ components

A k-SEQ component, $k>0$, is specified by

$$(\|i: 0 \leqslant i<k: \mathbf{pref}[a.i?;p.i!])$$
$$\| \mathbf{pref}[n?;(|i: 0 \leqslant i<k: p.i!)].$$

The specification and schematic of a 2-SEQ component are shown in Figure 2.2.7.

$$\mathbf{pref}[a?;p!]$$
$$\| \mathbf{pref}[b?;q!]$$
$$\| \mathbf{pref}[n?;(p!|q!)]$$

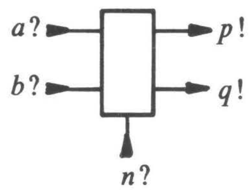


FIGURE 2.2.7. The 2-SEQ component.

Schematics for k-SEQ components, with $k>2$, are depicted similarly. Notice that a 1-SEQ component is a 2-CEL component.

For a k-SEQ component, $k>0$, we use the following terminology. Output $p.i$, $0 \leqslant i<k$, is called the *grant* of *request* $a.i$. We say that a request $a.i$, $0 \leqslant i<k$, is *pending* after trace t if $t\mathbf{N}a.i - t\mathbf{N}p.i = 1$. (Recall that $t\mathbf{N}x$ denotes the number of x's in trace t.) A SEQ component grants one request for each occurrence of input n. We also say that the SEQ component *sequences* the grants. In sequencing the grants it may have to arbitrate among several pending requests.

2.2.8. ARB components

The specification and schematic of a 2-ARB component is given in Figure 2.2.8.

$\mathbf{pref}[a1?;p1!;a0?;p0!]$

$\| \mathbf{pref}[b1?;q1!;b0?;q0!]$

$\| \mathbf{pref}[p1!;a0? \mid q1!;b0?]$

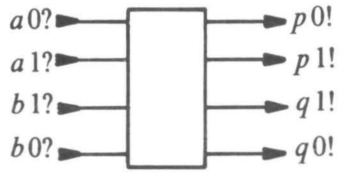


FIGURE 2.2.8. The 2-ARB component.

The 2-ARB component performs an operation similar to the 2-SEQ component, though it has a slightly more complicated communication protocol. The following names can be associated with the symbols

$a1?$ request $p1!$ grant

$a0?$ release $p0!$ confirm of release,

and similarly for the b and q symbols.

Generalizing the 2-ARB component to k-ARB components, $k>0$, is done similarly to the k-SEQ components.

2.2.9. SINK, SOURCE, and EMPTY components

Specifications for the SINK and SOURCE components are given in Figure 2.2.9.

$\mathbf{pref}(a?)$

$<\varnothing,\{a\},\varnothing,\varnothing,\{\epsilon\}>$

$\mathbf{pref}(a!)$

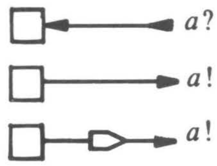


FIGURE 2.2.9. A SINK and two SOURCE components.

A SINK component has only one input terminal and can accept at most one transition at this terminal. A SOURCE component has only one output terminal and either does not produce any output transition at this terminal or it produces only one output transition. In the latter case, it is called an *active* SOURCE component. In the former case, it is called a *passive* SOURCE component. (Later, dangling inputs or outputs are connected to SOURCE or

SINK components, respectively.)

The component represented by the command ϵ is called the EMPTY component.

2.3. EXAMPLES

2.3.0. A conjunction component

Consider the component specified by the command

$$\mathbf{pref}[a0?\|b0?;c0! \mid a0?\|b1?;c0! \mid a1?\|b0?;c0! \mid a1?\|b1?;c1!].$$

We call this component a conjunction component for two binary variables, here a and b, encoded by a *two-rail scheme* in a *2-cycle signaling version* [40]. A two-rail scheme signifies that each binary variable is encoded by two symbols, one for each value. For the binary variable a we have the symbols $a0$ and $a1$, which correspond to two input terminals. A 2-cycle signaling protocol signifies that each communication cycle consists of the communication of an input value and an output value. A value is communicated by one transition at the terminal corresponding to that value. In 4-cycle signaling, each 2-cycle signaling is immediately followed by another 2-cycle signaling of the same values. Instead of the alternative $a0?\|b0?;c0!$, we have $a0?\|b0?;c0!;a0?\|b0?;c0!$, and similarly for the other alternatives. Since after each two voltage transitions the voltage has returned to its initial value, which is zero here, one also calls 4-cycle signaling return-to-zero signaling and 2-cycle signaling nonreturn-to-zero signaling [40].

Components specifying the disjunction, equivalence, negation, or combinations of these logical operators are similarly expressed by commands. Other ways of encoding data in delay-insensitive communications are given in [48].

2.3.1. A sequence detector

The specification of the following component demonstrates how a finite state machine with inputs and outputs can be specified by a directed command. The example is taken from [23].

A sequence detector has input alphabet $\{a0,a1\}$ and output alphabet $\{y,n\}$. The communication behavior of this component is described as follows. Inputs and outputs alternate, and if the last four inputs form the sequence $a0\,a1\,a1\,a0$, output y is produced; otherwise, output n is produced. Initially, the sequence detector receives an input.

The sequence detector can be specified by the state graph of Figure 2.3.0.

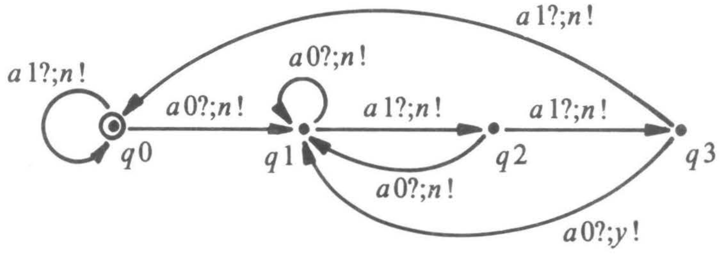


FIGURE 2.3.0. State graph for the sequence detector.

Consequently, the directed command for this component can be given by $\mu.tailf.0$, where *tailf* is defined by

$$tailf.R.0 = \textbf{pref}(a0?;n!;R.1 \mid a1?;n!;R.0)$$
$$tailf.R.1 = \textbf{pref}(a0?;n!;R.1 \mid a1?;n!;R.2)$$
$$tailf.R.2 = \textbf{pref}(a0?;n!;R.1 \mid a1?;n!;R.3)$$
$$tailf.R.3 = \textbf{pref}(a0?;y!;R.1 \mid a1?;n!;R.0).$$

2.3.2. A token-ring interface (0)

Consider a number of machines. For each machine we introduce a component, and all components are connected in a ring. Through this ring a so-called token is propagated from component to component. The ring-wise connection is called a *token ring*, and the components are called *token-ring interfaces*. Each machine communicates with the token ring through its token-ring interface.

Token rings can be used for many purposes. They are used, for example, to achieve mutual exclusion among machines entering a critical section [25] or to detect the termination of a distributed computation [8]. For each purpose a particular communication protocol is specified for the token-ring interfaces. In this and in the next section, we discuss two of these communication protocols, and we show how they can be specified by directed commands.

In order to achieve mutual exclusion among machines entering a critical section, the following protocol is described for a token-ring interface. The schematic of the token-ring interface is given in Figure 2.3.1.

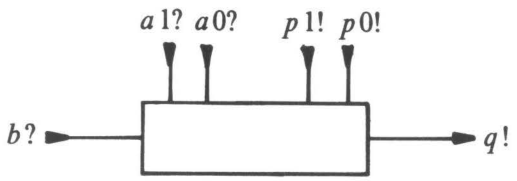


FIGURE 2.3.1. A token-ring interface.

The communication actions between token-ring interface and machine are interpreted as follows.

- $a\,1?$ request for the token
- $p\,1!$ grant of the token
- $a\,0?$ release of the token
- $p\,0!$ confirm of release.

With respect to these actions the protocol satisfies the specification **pref**$[a\,1?;p\,1!;a\,0?;p\,0!]$.

The communication actions between token-ring interface and the rest of the token ring are interpreted as follows.

- $b\,?$ receipt of the token
- $q\,!$ sending of the token.

With respect to these actions the protocol satisfies the specification **pref**$[b\,?;q\,!]$.

The synchronization between the two protocols must satisfy the following requirements. After each receipt of the token, the token can either be sent on to the next token-ring interface or, if there is also a request from the machine, the token can be granted to the machine. If the machine releases the token, it is sent on to the next token-ring interface. From the definition of weaving and the above we infer that the complete communication protocol can be specified by the directed command

$$\mathbf{pref}[a\,1?;p\,1!;a\,0?;p\,0!]$$
$$\|\,\mathbf{pref}[b\,?;(q\,!\mid p\,1!;a\,0?;q\,!)].$$

2.3.3. A token-ring interface (1)

The following specification for a communication protocol is inspired by [8].

We characterize the state of a machine by either black or white. A machine can change its color from black to white and vice versa. The token can also be black or white. The color of the token can be changed by the token-ring interface from white to black only. We are asked to design a communication protocol for the token-ring interface that satisfies the following requirements.

(*i*) Tokens are transmitted only if the machine is white.

(*ii*) A token is transmitted black only if after the previous transmission of a token the machine has become black at least once. Otherwise, the token is transmitted unchanged.

For the derivation of a communication protocol we introduce the symbols *b*, *w*, *tb*, and *tu* with the following interpretations.

b machine changes to black

w machine changes to white

tb transmit black token

tu transmit token unchanged.

These symbols represent the actual moments of change of color or of transmission. (Notice that we have not assigned a type to these symbols yet.) Designing a protocol with these symbols only, yields the command

$$\mathbf{pref}[[tu];b;w;[b;w];tb],$$

where we assume that the machine is white initially. Condition (*i*) is obviously satisfied: between equally numbered occurrences of *b* and *w*, i.e. when the machine is black, symbols *tu* and *tb* do not occur. Further, the command $b;w;[b;w]$ contains all traces in which the machine has become black (and changed to white) at least once. From this observation follows that (*ii*) is also satisfied.

We use the symbols *b*, *w*, *tu*, and *tb* to introduce the communication symbols. We introduce one set of symbols for the communication between machine and token-ring interface and one set of symbols for the communication between the rest of the token ring and the token-ring interface. We consider the symbols *b*, *w*, *tu*, and *tb* as internal symbols of the component. Consequently, the token-ring interface is considered an extension of the machine: the change of color of the machine takes place internally in the token-ring interface.

For the communication between the token-ring interface and (the rest of) the machine we introduce the symbols

rb? request to become black

gb! machine has become black

rw? request to become white

gw! machine has become white.

The protocol with respect to these symbols only and the internal symbols *b* and *w* is described by

$$\mathbf{pref}[rb?;!b?;gb!;rw?;!w?;gw!].$$

For the communication between token-ring interface and the rest of the token ring we introduce the symbols

btr? receipt of black token

wtr? receipt of white token

bts! sending of black token

wts! sending of white token.

The protocol with respect to these symbols only and the internal symbols *tu* an *tb* is specified by

$$\begin{array}{l} \mathbf{pref}[wtr?;(!tu?;wts!\mid !tb?;bts!) \\ \quad |btr?;(!tu?|!tb?);bts! \\ \quad]. \end{array}$$

The proper synchronization of these protocols is described by their weave. Projecting this weave on the external symbols gives the desired protocol, i.e.

$$\begin{array}{l} (\mathbf{pref}[rb?;!b?;gb!;rw?;!w?;gw!] \\ \| \ \mathbf{pref}[wtr?;(!tu?;wts!\mid !tb?;bts!) \\ \quad |btr?;(!tu?|!tb?);bts! \\ \quad] \\ \| \ \mathbf{pref}[[!tu?];!b?;!w?;[!b?;!w?];!tb?] \\ )\restriction. \end{array}$$

Finally, we remark that the last sequential command of the above weave can also be written as $\mu.tailf.0$, where *tailf* is defined by

$$tailf.R.0 = \mathbf{pref}(!tu?;R.0 \mid !b?;!w?;R.1)$$

$$tailf.R.1 = \mathbf{pref}(!tb?;R.0 \mid !b?;!w?;R.1).$$

It will turn out that this last sequential command is better suited for the syntactical check to be developed in Chapter 4 and the syntax-directed translation of Chapters 5 and 6.

2.3.4. The dining philosophers

A canonical example of a mutual exclusion problem is the paradigm of the dining philosophers [6]. In the following we derive a communication protocol for the dining philosophers expressed in a command.

Consider N dining philosophers, $N > 0$, whose lives consist of alternations of thinking and eating. The N philosophers are seated at a round table with N plates, one for each philosopher. Between any two successive plates lies one fork. A philosopher can start eating if he has got hold of both forks lying next to his plate. When a philosopher finishes eating, he releases both forks. A fork can be occupied by at most one philosopher. We are asked to design a communication protocol for the N dining philosophers such that no philosopher is kept from eating unnecessarily, i.e. no deadlock occurs (Notice that if all N philosophers pick up their right forks simultaneously, nobody can pick up his left fork as well, and thus they may keep each other from eating forever.)

Let the component with which the N philosophers communicate be called *TABLE*. We design a communication protocol for the component *TABLE*. The communication actions between philosopher i, $0 \leqslant i < N$, and *TABLE* are given by

$p.i!$ start thinking

$a.i?$ request to eat, i.e. finish thinking

$q.i!$ start eating

$b.i?$ request to think, i.e. finish eating

With respect to philosopher i, $0 \leqslant i < N$, the protocol satisfies

$$PHIL.i = \mathbf{pref}[p.i!;a.i?;q.i!;b.i?].$$

The synchronization among all N protocols $PHIL.i$, $0 \leqslant i < N$, must be such that each fork is occupied by at most one philosopher, i.e. no two neighbors are eating simultaneously. These restrictions are expressed by the commands

$$FORK.i = \mathbf{pref}[q.i!;b.i? \mid q.(i+1)!;b.(i+1)?], \quad \text{for } 0 \leqslant i < N,$$

where addition is modulo N.

The protocols $PHIL.i$ and $FORK.i$, $0 \leqslant i < N$, are the only restrictions that the communications must satisfy. Consequently, *TABLE* can be specified by

$$TABLE = (\|i : 0 \leqslant i < N : PHIL.i)$$
$$\| (\|i : 0 \leqslant i < N : FORK.i).$$

Notice that when philosopher i, $0 \leqslant i < N$, starts eating, he picks up both forks 'at the same time', since $q.i!$ occurs in the commands $FORK.(i-1)$, $FORK.i$, and $PHIL.i$. From this observation it follows that no philosopher is kept from eating unnecessarily, i.e. there is no deadlock.

Chapter 3

Decomposition and Delay-Insensitivity

3.0. Introduction

The idea of this thesis is to realize a component by means of a delay-insensitive connection of basic components. In this chapter we formalize this idea by means of three definitions and derive some theorems based on these definitions.

First, we define what we mean by 'a component can be realized by a connection of (other) components'. This is formulated in the definition of *decomposition*. Decomposition is defined as a relation holding between the component to be decomposed and the components in which it is decomposed. We stipulate that a component $S.0$ can be decomposed into the components $S.i$, $1 \leq i < n$, if the connection of components $S.i$, $1 \leq i < n$, realizes the prescribed behavior of component $S.0$, where it is assumed that the environment of this connection behaves as specified for environment $S.0$. (Recall from Section 2.2.0 that a directed trace structure prescribes both the behavior of a component and its environment.)

From the definition of decomposition we derive two theorems: the *Substitution Theorem*, which enables us to decompose a component in a hierarchical way, and the *Separation Theorem*, which enables us to decompose parts of a specification separately.

The realization of a component by means of a delay-insensitive connection of components is formalized by the definition of *DI decomposition*. We then consider connections of components in which corresponding input and output terminals are connected by WIRE components. WIRE components introduce, operationally speaking, a delay in the communications between the terminals. In the definition of DI decomposition it is required that these delays do not influence the functional behavior of the connection.

In order to link decomposition and DI decomposition we introduce *DI components.* A DI component may be interpreted as a component whose specification is valid at a flexible boundary, or, operationally speaking, a DI component communicates in a delay-insensitive way with its environment. By means of DI components we can formulate the fundamental theorem of this chapter: DI decomposition is equivalent to decomposition if all components involved are DI components. Because of the theorems that apply for decomposition, it is easier to work with decompositions than with DI decompositions. For this reason, we mostly discuss decompositions and DI components in the following chapters.

3.1. DECOMPOSITION

3.1.0. The definition

Below, we first present the definition of decomposition and then give a brief motivation for it.

DEFINITION 3.1.0.0. *We say that component* $S.0$ *can be decomposed into components* $S.i$, $1 \leq i < n$ *for a fixed* $n > 1$, *denoted by*

$$S.0 \rightarrow (i: 1 \leq i < n: S.i),$$

if the following conditions are satisfied.
Let $R.0 = \overline{S.0}$, $R.i = S.i$ *for* $1 \leq i < n$, *and* $W = (\|i: 0 \leq i < n: R.i)$.

(*i*) *(Closed connection)*
$(\cup i: 0 \leq i < n: \mathbf{o}(R.i)) = (\cup i: 0 \leq i < n: \mathbf{i}(R.i))$.

(*ii*) *(No output interference)*
$\mathbf{o}(R.i) \cap \mathbf{o}(R.j) = \varnothing$ *for* $0 \leq i, j < n \wedge i \neq j$.

(*iii*) *(Connection behaves as specified at boundary* $\mathbf{a}(S.0)$*)*
$\mathbf{t}W \upharpoonright \mathbf{a}(R.0) = \mathbf{t}(R.0)$.

(*iv*) *(Connection is free of computation interference)*
For all traces t, symbols x, and indexes i, $0 \leq i < n$, *we have*
$t \in \mathbf{t}W \wedge x \in \mathbf{o}(R.i) \wedge tx \upharpoonright \mathbf{a}(R.i) \in \mathbf{t}(R.i) \Rightarrow tx \in \mathbf{t}W$.

□

NOTATIONAL REMARK. The notation $(i: 0 \leq i < n: S.i)$ can be interpreted as an enumeration of the components $S.i$, $0 \leq i < n$. Notice, however, that the order of this enumeration is not important, as can be deduced from the specification.

Instead of, for example, $S.0 \to (i: 1 \leq i < 4: S.i)$ we sometimes write $S.0 \to S.1, S.2, S.3$ or $S.0 \to (i: 0 \leq i < 3: S.i), S.3$. Here, the comma separates the components or lists of components.

□

In Section 2.2.0, we stipulated that a directed trace structure $S.0$ prescribes the behavior of component and environment: it specifies when the component may produce outputs and when the environment may produce inputs. In a decomposition of component $S.0$ we require that the production of outputs of component $S.0$ are realized by a connection of components. We assume that the environment of this connection produces the inputs as specified for environment $S.0$. This environment can also be seen as component $\overline{S.0}$. Accordingly, in order to comprise all components that produce outputs relevant to the decomposition, we consider the connection of components $\overline{S.0}$ and $S.i$, $1 \leq i < n$.

Condition (*i*) says that there are no dangling inputs and outputs in the connection: every output is connected to an input, and every input is connected to an output. We call such a connection a *closed connection*.

Condition (*ii*) requires that outputs of distinct components are not connected with each other. If (*ii*) holds we say that the connection is *free of output interference.*

Condition (*iii*) requires that the behavior of the connection at the boundary $\mathbf{a}(S.0)$ behaves as specified by $\mathbf{t}(S.0)$. The behavior of the connection is given by $\mathbf{t}W = \mathbf{t}(\|i: 0 \leq i < n: R.i)$. Restriction to the boundary $\mathbf{a}(S.0)$ $(= \mathbf{a}(R.0))$ is expressed by $\mathbf{t}W \upharpoonright \mathbf{a}(R.0)$.

Condition (*iv*) requires that the connection is free of computation interference. We say that the connection has danger of *computation interference,* if there exists a trace t, symbol x, and index i, $0 \leq i < n$, such that

$$t \in \mathbf{t}W \ \wedge \ x \in \mathbf{o}(R.i) \ \wedge \ tx \upharpoonright \mathbf{a}(R.i) \in \mathbf{t}(R.i) \ \wedge \ tx \notin \mathbf{t}W.$$

In words, if after a mutually agreed behavior a component can produce an output that is not in accordance with the prescribed behavior of other components, then we say that the connection has danger of computation interference.

Since a specification may be interpreted as a boundary prescription for the behavior of component and environment, computation interference may also be interpreted as a boundary violation. For example, if WIRE component $\mathbf{pref}[a?;b!]$ receives two inputs a without producing an output b, we have a boundary violation for the WIRE component. Operationally speaking, in the case of this boundary violation more than one transition is propagating along a wire, which can cause hazardous behavior and must, therefore, be avoided. A boundary violation for a WIRE component is also called *transmission interference* [42]. (Consequently, transmission interference is a special case of computation interference.) In the following, a connection that satisfies conditions (*i*), (*ii*), and (*iv*) is briefly called a closed connection, free of interference.

REMARK. Some misbehaviors of circuits that are characterized in classical switching theory by hazards or critical races [23,29] can be seen as special cases of computation interference. Absence of interference in a decomposition guarantees that the thus synthesized circuit is free of hazards and critical races, if the components satisfy their specifications.
□

Notice that we have described decomposition as a goal-directed activity: we start with a component $S.0$ and try to find components $S.i$, $1 \leq i < n$, such that the relation $S.0 \rightarrow (i: 1 \leq i < n: S.i)$ holds. Thus, we explicitly use the assumption that the environment of the connection of components behaves as specified for environment $S.0$. We did not start with components $S.i$, $1 \leq i < n$, to find out what could be made of them without requiring anything from the environment. This is also the reason why this method is called decomposition instead of composition.

3.1.1. *Examples*

EXAMPLE 3.1.1.0. We demonstrate that WIRE component $\mathbf{pref}[a?;d!]$ can be decomposed into FORK component $\mathbf{pref}[a?;b!\|c!]$ and CEL component $\mathbf{pref}[b?\|c?;d!]$. A schematic of this decomposition is given in Figure 3.1.0.

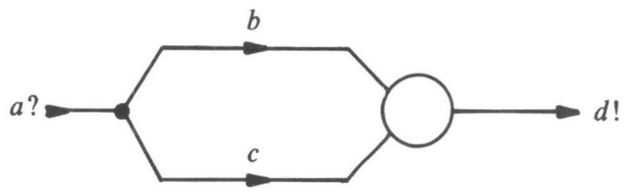


FIGURE 3.1.0. A decomposition of a WIRE component.

Let

$$R.0 = \mathbf{pref}[a!;d?],$$
$$R.1 = \mathbf{pref}[a?;b!\|c!], \text{ and}$$
$$R.2 = \mathbf{pref}[b?\|c?;d!].$$

By inspection, we infer that the connection of components $R.0$, $R.1$, and $R.2$ is closed and free of output interference. The behavior of this connection is represented by

$$\mathbf{t}W = \mathbf{t}(R.0 \| R.1 \| R.2)$$
$$= \mathbf{t}\,\mathbf{pref}[a;b\|c;d].$$

From this we derive $\mathbf{t}W\upharpoonright\mathbf{a}(R.0)=\mathbf{t}\,\mathbf{pref}[a\,;d\,]$. Accordingly, we conclude that the connection behaves as specified at the boundary $\mathbf{a}(R.0)$.

For absence of computation interference we have to prove for all t, x, i, $0\leq i<3$, that

$$t\in\mathbf{t}W \wedge x\in\mathbf{o}(R.i) \wedge tx\upharpoonright\mathbf{a}(R.i)\in\mathbf{t}(R.i) \Rightarrow tx\in\mathbf{t}W.$$

Instead of proving this for all triples (t,x,i), we take for all states of $\mathbf{t}W$ a representative t and consider all x and i, $0\leq i<3$, such that

$$t\in\mathbf{t}W \wedge x\in\mathbf{o}(R.i) \wedge tx\upharpoonright\mathbf{a}(R.i)\in\mathbf{t}(R.i).$$

It suffices to prove for these triples (t, x, i) that $tx\in\mathbf{t}W$. By inspection, we find that for the triples

$$(\epsilon, a, 0),\ (a, b, 1),\ (a, c, 1),\ (ab, c, 1),\ (ac, b, 1), \text{ and } (abc, d, 2)$$

indeed $tx\in\mathbf{t}W$. Consequently, we conclude that $\overline{R.0}$ can be decomposed into $R.1$ and $R.2$.

□

EXAMPLE 3.1.1.1. We examine whether WIRE component $\mathbf{pref}[a?;d!]$ can be decomposed into FORK component $\mathbf{pref}[a?;b!]\,\|\,\mathbf{pref}[a?;c!]$ and CEL component $\mathbf{pref}[b?;d!]\,\|\,\mathbf{pref}[d!;c?]$. Notice that this CEL component starts in a different initial state than the CEL component of the previous example. The tentative decomposition is given in Figure 3.1.1.

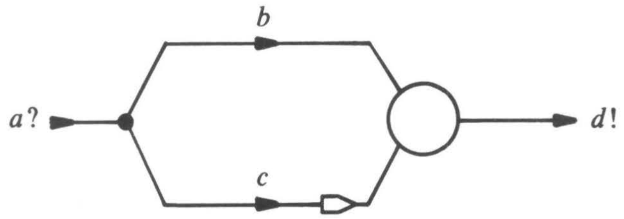


FIGURE 3.1.1. A tentative decomposition of a WIRE component.

Let

$$R.0 = \mathbf{pref}[a!;d?],$$
$$R.1 = \mathbf{pref}[a?;b!]\,\|\,\mathbf{pref}[a?;c!] \text{ ,and}$$
$$R.2 = \mathbf{pref}[b?;d!]\,\|\,\mathbf{pref}[d!;c?]\,.$$

Analogously to the previous example, we infer that the components $R.0$, $R.1$ and $R.3$ form a closed connection free of output interference. The behavior of this connection is given by

$$\mathbf{t}W = \mathbf{t}(R.0\,\|\,R.1\,\|\,R.2)$$
$$= \mathbf{t}\,\mathbf{pref}[a\,;b\,;d\,;c\,],$$

from which we readily derive $\mathbf{t}W\!\upharpoonright\!\mathbf{a}(R.0)=\mathbf{t}(R.0)$. We conclude that this connection behaves as specified at the boundary $\mathbf{a}(R.0)$.

There is, however, danger of computation interference in this connection: for $t,x,i:=a,c,1$ we have

$$a\in\mathbf{t}W \wedge c\in\mathbf{o}(R.1) \wedge ac\!\upharpoonright\!\mathbf{a}(R.1)\in\mathbf{t}(R.1) \wedge ac\notin\mathbf{t}W.$$

After the environment has produced an a, the FORK component can produce a c, which is not in accordance with the boundary prescription for the CEL component. Consequently, the tentative decomposition is not a decomposition.

□

EXAMPLE 3.1.1.2. We demonstrate that a 3-XOR component can be decomposed into two 2-XOR components, according to the schematic given in Figure 3.1.2.

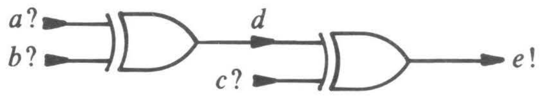


FIGURE 3.1.2. A decomposition for a 3-XOR component.

Let

$$R.0 = \mathbf{pref}[a!;e?\,|\,b!;e?\,|\,c!;e?],$$

$$R.1 = \mathbf{pref}[a?;d!\,|\,b?;d!] \text{ ,and}$$

$$R.2 = \mathbf{pref}[d?;e!\,|\,c?;e!].$$

By inspection, we find that the components $R.0$, $R.1$, and $R.3$ form a closed connection free of output interference. For the behavior of this connection we obtain

$$\mathbf{t}W = \mathbf{t}(R.0 \,\|\, R.1 \,\|\, R.2)$$
$$= \mathbf{t}\,\mathbf{pref}[a;d;e\,|\,b;d;e\,|\,c;e].$$

Accordingly, we derive $\mathbf{t}W\!\upharpoonright\!\mathbf{a}(R.0)=\mathbf{t}(R.0)$, i.e. the connection behaves as specified at the boundary $\mathbf{a}(R.0)$. Applying the same approach as in Example 3.1.1.0, we find for each of the triples (t,x,i) from

$(\epsilon, a, 0)$, $(\epsilon, b, 0)$, $(\epsilon, c, 0)$, $(a, d, 1)$, and $(c, e, 2)$, that

$$t\in\mathbf{t}W \wedge x\in\mathbf{o}(R.i) \wedge tx\!\upharpoonright\!\mathbf{a}(R.i) \wedge tx\in\mathbf{t}W.$$

Consequently, the connection is also free of computation interference, and we conclude that $R.0$ can be decomposed into $R.1$ and $R.2$.

□

EXAMPLE 3.1.1.3. Similarly to the above example, we can prove that 3-CEL component **pref**$[a?\|b?\|c?\,;e!]$ can be decomposed into 2-CEL components **pref**$[a?\|b?\,;d!]$ and **pref**$[d?\|c?\,;e!]$. This decomposition is depicted in Figure 3.1.3.

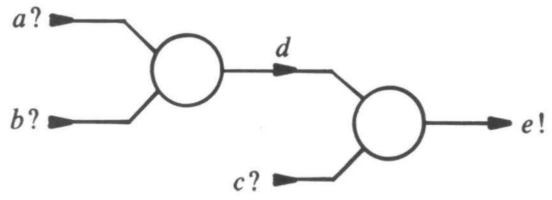


FIGURE 3.1.3. A decomposition of a 3-CEL component.

□

EXAMPLE 3.1.1.4. Also in the same fashion as the previous examples we can prove that the 2-CEL component **pref**$[c!;a?]\,\|\,$**pref**$[b?;c!]$ can be decomposed into the 2-CEL component **pref**$[d?;c!]\,\|\,$**pref**$[b?;c!]$ and the WIRE component **pref**$[d!;a?]$. This decomposition is depicted in Figure 3.1.4.

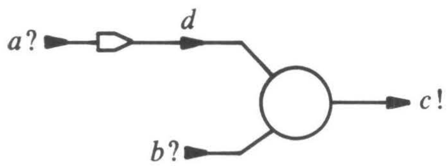


FIGURE 3.1.4. Decoupling an initial transition.

In general, any CEL component with initial transitions on some of its inputs can be decomposed into a CEL component without initial transitions on its inputs and WIRE components with initial transitions. A similar reasoning holds for XOR components.
□

EXAMPLE 3.1.1.5. We examine some decompositions of the form $S.0 \rightarrow S.1$, i.e. decompositions into one component only. First, we have $S \rightarrow S$ for any component S.

Second, for components $S.0$ and $S.1$ defined by

$S.0 =$ **pref**$[a?;b!;c?;d!]$ and

$S.1 =$ **pref**$[a?;b!\,|\,c?;d!]$,

for example, we have $S.0 \rightarrow S.1$.

Component $S.1$ can be decomposed further: let

$$S.2 = \mathbf{pref}[a?;b!] \parallel \mathbf{pref}[c?;d!],$$

then we infer $S.1 \rightarrow S.2$.

Given the decompositions $S.0 \rightarrow S.1$ and $S.1 \rightarrow S.2$, we may wonder whether $S.0 \rightarrow S.2$ holds as well. This is indeed so; in the next section we derive this decomposition by application of the Substitution Theorem.

We can still go one step further in the decomposition of $S.1$, since we have

$$S.2 \rightarrow \mathbf{pref}[a?;b!]\,,\ \mathbf{pref}[c?;d!].$$

This last decomposition is a special case of the Separation Theorem, which is also discussed in the next section.

□

3.1.2. The Substitution Theorem

A theorem that may be helpful in finding decompositions of a component is the Substitution Theorem. This theorem applies to problems of the following kind. Suppose that component $S.0$ can be decomposed into a number of components of which T is one such component. Suppose, moreover, that T can be decomposed further into a number of components. Under what conditions can the decomposition of T be substituted in the decomposition of $S.0$?

We have

THEOREM 3.1.2.0. (Substitution Theorem)
Let components $S.i$, $0 \leqslant i < m$, and T satisfy for $1 \leqslant n < m$

$$(\cup i: 0 \leqslant i < n: \mathbf{a}(S.i)) \cap (\cup i: n \leqslant i < m: \mathbf{a}(S.i)) = \mathbf{a}T. \tag{3.1}$$

We have

$$\begin{aligned} & S.0 \rightarrow (i: 1 \leqslant i < n: S.i),\ T \\ & \wedge\ T \rightarrow (i: n \leqslant i < m: S.i) \\ \Rightarrow\ & S.0 \rightarrow (i: 1 \leqslant i < m: S.i). \end{aligned}$$

□

Condition (3.1) of the above theorem is essentially a void condition, since, by an appropriate renaming of the internal symbols in the decomposition of T, this condition can always be satisfied. The internal symbols of the decomposition of T are given by $(\cup : n \leqslant i < m: \mathbf{a}(S.i)) \setminus \mathbf{a}T$.

Proof (of Theorem 3.1.2.0). Let

$$R.0 = \overline{S.0}\ ,\ R.i = S.i \text{ for } 1 \leqslant i < m,$$
$$W0 = (\|i: 0 \leqslant i < m: R.i),$$
$$W1 = (\|i: 0 \leqslant i < n: R.i) \,\|\, T, \quad \text{and}$$
$$W2 = (\|i: n \leqslant i < m: R.i) \,\|\, \overline{T}.$$

(*i*) We observe

$$S.0 \to (i: 1 \leqslant i < n: S.i), T$$
$$\wedge\ T \to (i: n \leqslant i < m: S.i)$$
$$\Rightarrow \{\text{condition } (i) \text{ of decomposition}\}$$
$$(\cup i: 0 \leqslant i < n: \mathbf{o}(R.i)) \cup \mathbf{o}T = (\cup i: 0 \leqslant i < n: \mathbf{i}(R.i)) \cup \mathbf{i}T$$
$$\wedge\ (\cup i: n \leqslant i < m: \mathbf{o}(R.i)) \cup \mathbf{i}T = (\cup i: n \leqslant i < m: \mathbf{i}(R.i)) \cup \mathbf{o}T$$
$$\Rightarrow \{\text{calc. }, \mathbf{o}T \cap \mathbf{i}T = \varnothing\}$$
$$(\cup i: 0 \leqslant i < m: \mathbf{o}(R.i)) = (\cup i: 0 \leqslant i < m: \mathbf{i}(R.i)).$$

(*ii*) Since

$$S.0 \to (i: 1 \leqslant i < n: S.i), T \text{ and}$$
$$T \to (i: n \leqslant i < m: S.i),$$

we have, by condition (*ii*) of decomposition, for $i \neq j$

$$\mathbf{o}(R.i) \cap \mathbf{o}(R.j) = \varnothing\ ,\ \text{for } 0 \leqslant i,j < n \vee n \leqslant i,j < m, \text{ and}$$
$$\mathbf{o}(R.i) \cap \mathbf{o}T = \varnothing \wedge \mathbf{o}(R.j) \cap \mathbf{i}T = \varnothing \quad \text{for} \quad 0 \leqslant i < n \wedge n \leqslant j < m.$$

From condition (3.1) in the theorem follows

$$\mathbf{o}(R.i) \cap \mathbf{o}(R.j) \subseteq \mathbf{a}T \quad \text{for } 0 \leqslant i < n \wedge n \leqslant j < m.$$

For component T, we have $\mathbf{i}T \cap \mathbf{o}T = \varnothing$. This combined with the above yields

$$\mathbf{o}(R.i) \cap \mathbf{o}(R.j) = \varnothing \quad \text{for } 0 \leqslant i,j < m \wedge i \neq j.$$

(*iv*) (We first prove (*iv*) and then (*iii*) of the definition of decomposition.) We show that for all t, b, i, $0 \leqslant i < m$,

$$t \in \mathbf{t}(W1 \| W2) \wedge b \in \mathbf{o}(R.i) \wedge tb{\restriction}\mathbf{a}(R.i) \in \mathbf{t}(R.i)$$
$$\Rightarrow tb \in \mathbf{t}(W1 \| W2). \tag{3.2}$$

and that $\mathbf{t}(W1 \| W2) = \mathbf{t}(W0)$. From these two properties condition (*iv*) of decomposition can then be concluded.

Let $0 \leq i < n$. We observe

$$t \in \mathbf{t}(W1 \| W2) \ \wedge \ b \in \mathbf{o}(R.i) \ \wedge \ tb \upharpoonright \mathbf{a}(R.i) \in \mathbf{t}(R.i)$$

$\Rightarrow$ {def. of weaving}

$$t \upharpoonright \mathbf{a}W1 \in \mathbf{t}W1 \ \wedge \ b \in \mathbf{o}(R.i) \ \wedge \ tb \upharpoonright \mathbf{a}(R.i) \in \mathbf{t}(R.i)$$

$\Rightarrow$ $\{S.0 \rightarrow (i: 1 \leq i < n: S.i), T$,condition (*iv*) of decomposition, calc.$\}$

$$tb \upharpoonright \mathbf{a}W1 \in \mathbf{t}W1. \tag{3.3}$$

To prove also that $tb \upharpoonright \mathbf{a}W2 \in \mathbf{t}W2$ for $0 \leq i < n$, we consider two cases: $b \notin \mathbf{a}W2$ and $b \in \mathbf{a}W2$. For $b \notin \mathbf{a}W2$ we have, by the definition of weaving,

$$t \in \mathbf{t}(W1 \| W2) \wedge b \notin \mathbf{a}W2 \Rightarrow tb \upharpoonright \mathbf{a}W2 \in \mathbf{t}W2.$$

For $b \in \mathbf{a}W2$, we derive

$$t \in \mathbf{t}(W1 \| W2) \ \wedge \ b \in \mathbf{o}(R.i) \ \wedge \ tb \upharpoonright \mathbf{a}(R.i) \in \mathbf{t}(R.i) \ \wedge \ b \in \mathbf{a}W2$$

$\Rightarrow$ $\{(3.3),\ 0 \leq i < n\}$

$$t \in \mathbf{t}(W1 \| W2) \ \wedge \ b \in \mathbf{o}(R.i) \ \wedge \ b \in (\mathbf{a}W2 \cap \mathbf{a}W1) \ \wedge \ tb \upharpoonright \mathbf{a}W1 \in \mathbf{t}W1$$

$\Rightarrow$ {condition (3.1), def. of weaving}

$$t \in \mathbf{t}(W1 \| W2) \ \wedge \ b \in \mathbf{o}(R.i) \ \wedge \ b \in \mathbf{a}T \ \wedge \ tb \upharpoonright \mathbf{a}T \in \mathbf{t}T$$

$\Rightarrow$ $\{S.0 \rightarrow (i: 1 \leq i < n: S.i), T$,condition (*ii*) of decomposition$\}$

$$t \in \mathbf{t}(W1 \| W2) \ \wedge \ b \in \mathbf{i}T \ \wedge \ tb \upharpoonright \mathbf{a}T \in \mathbf{t}T$$

$\Rightarrow$ {def. of reflection, def. of weaving}

$$t \upharpoonright \mathbf{a}W2 \in \mathbf{t}W2 \ \wedge \ b \in \mathbf{o}\overline{T} \ \wedge \ tb \upharpoonright \mathbf{a}\overline{T} \in \mathbf{t}\overline{T}$$

$\Rightarrow$ $\{T \rightarrow (i: n \leq i < m: S.i)$, condition (*iv*) of decomposition, calc.$\}$

$$tb \upharpoonright \mathbf{a}W2 \in \mathbf{t}W2.$$

Since $tb \in (\mathbf{a}W1 \cup \mathbf{a}W2)^*$, we derive with (3.3) and the definition of weaving that $tb \in \mathbf{t}(W1 \| W2)$.

For $n \leq i < m$, we derive similarly that (3.2) holds.

Subsequently, we show that $\mathbf{t}(W1 \| W2) = \mathbf{t}W0$. We observe $\mathbf{a}(W1 \| W2) = \mathbf{a}W0$ and $\mathbf{t}(W1 \| W2) = \mathbf{t}(W0 \| T)$. By definition of weaving, we derive $\mathbf{t}(W1 \| W2) \subseteq \mathbf{t}W0$. We prove $t \in \mathbf{t}W0 \Rightarrow t \in \mathbf{t}(W1 \| W2)$ by induction to the length of t.

Base: $W0$ and $W1 \| W2$ are prefix-closed and non-empty, hence $\epsilon \in \mathbf{t}W0 \wedge \epsilon \in \mathbf{t}(W1 \| W2)$.

Step: We observe

$tb \in \mathbf{t}W0$

$\Rightarrow$ $\{W0$ is prefix-closed$\}$

$t \in \mathbf{t}W0 \wedge tb \in \mathbf{t}W0$

$\Rightarrow$ {induction hypothesis for t}

$t \in \mathbf{t}(W1 \| W2) \wedge tb \in \mathbf{t}W0$

$\Rightarrow$ {by (*i*) in this proof and def. of weaving}

$(\mathbf{E}i : 0 \leq i < m : t \in \mathbf{t}(W1 \| W2) \wedge b \in \mathbf{o}(R.i) \wedge tb \upharpoonright \mathbf{a}(R.i) \in \mathbf{t}(R.i))$

$\Rightarrow$ {(3.2)}

$tb \in \mathbf{t}(W1 \| W2)$.

(*iii*) To prove $\mathbf{t}W0 \upharpoonright \mathbf{a}(R.0) = \mathbf{t}(R.0)$, we use a result of (*iv*), i.e. $\mathbf{t}W0 = \mathbf{t}(W1 \| W2)$. We observe

$\mathbf{t}W0 \upharpoonright \mathbf{a}(R.0)$

$=$ {see (*iv*)}

$\mathbf{t}(W1 \| W2) \upharpoonright \mathbf{a}(R.0)$

$=$ $\{\mathbf{a}(R.0) \subseteq \mathbf{a}W1$, by (3.1): $\mathbf{a}W1 \cap \mathbf{a}W2 = \mathbf{a}T$, Prop. 1.1.2.6$\}$

$\mathbf{t}(W1 \| (W2 \upharpoonright \mathbf{a}T)) \upharpoonright \mathbf{a}(R.0)$

$=$ $\{T \rightarrow (i : n \leq i < m : S.i)$, condition (*iii*) of decomposition, calc.$\}$

$\mathbf{t}(W1 \| T) \upharpoonright \mathbf{a}(R.0)$

$=$ {calc.}

$\mathbf{t}W1 \upharpoonright \mathbf{a}(R.0)$

$=$ $\{S.0 \rightarrow (i : 1 \leq i < n : S.i), T$, condition (*iii*) of decomposition$\}$

$\mathbf{t}(R.0)$.

□

In (*i*), (*ii*), and (*iv*) of the above proof we did not use condition (*iii*) of decomposition. Consequently, we conclude

Theorem 3.1.2.1.

$$(3.4) \wedge (3.5) \wedge (3.1)$$
$$\Rightarrow (3.6) \wedge (\|i{:}\,0 \leqslant i < m{:}\, R.i) = (\|i{:}\,0 \leqslant i < m{:}\, R.i) \,\|\, T,$$

where

(3.4) $\equiv$ *the components $R.i$, $0 \leqslant i < n$, and T form a closed connection, free of interference.*

(3.5) $\equiv$ *the components $R.i$, $n \leqslant i < m$, and $\overline{T}$ form a closed connection, free of interference.*

(3.6) $\equiv$ *the components $R.i$, $0 \leqslant i < m$, form a closed connection, free of interference.*

□

Example 3.1.2.2. Consider the components $S.0$, $S.1$, and $S.2$ of Example 3.1.1.5 again. We have

$$S.0 \to S.1 \;\wedge\; S.1 \to S.2 \;\wedge$$
$$(\mathbf{a}(S.0) \cup \mathbf{a}(S.1)) \cap (\mathbf{a}(S.1) \cup \mathbf{a}(S.2)) \;=\; \mathbf{a}(S.1).$$

By the Substitution Theorem we conclude $S.0 \to S.2$. Moreover, we also have

$$S.2 \to \mathbf{pref}[a?;b!]\ ,\ \mathbf{pref}[c?;d!].$$

Here as well the condition for the Substitution Theorem is satisfied, and we conclude

$$S.0 \to \mathbf{pref}[a?;b!]\ ,\ \mathbf{pref}[c?;d!].$$

Consequently, $S.0$ can be decomposed into two WIRE components.
□

Notational Remark. In the derivation for a decomposition of a component we sometimes use a notation similar to the proofs in this thesis. For example, for the derivation of a decomposition $S.0 \to S.1, S.2, S.3$ we may write

$$S.0$$
$$\to\{\text{hint why } S.0 \to S.1, S.2\}$$
$$S.1, S.2$$
$$\to\{\text{hint why } S.1 \to S.3, S.4\}$$
$$S.3, S.4, S.2.$$

Such a derivation is then based on the Substitution Theorem, and it is assumed that the condition for its application holds.

□

3.1.3. The Separation Theorem

Another theorem that may be convenient in finding decompositions of a component is the Separation Theorem. It pertains to problems of the following kind. Suppose that for the components $S0, S1, S2, T0, T1$, and $T2$ we have $S0 \rightarrow S1, S2$ and $T0 \rightarrow T1, T2$. Can we derive from these decompositions a decomposition for component $S0 \| T0$? For example, does $S0 \| T0 \rightarrow S1 \| T1, S2 \| T2$ hold?

We have

Theorem 3.1.3.0. (Separation Theorem) *Let components* $S.k.i$, $0 \leq k < n \wedge 0 \leq i < m$, *satisfy* $S.k.0 \rightarrow (i: 1 \leq i < m: S.k.i)$. *We have*

$$(\|k: 0 \leq k < n: S.k.0) \rightarrow (i: 1 \leq i < m: (\|k: 0 \leq k < n: S.k.i))$$

if the following conditions are satisfied.

$$A.k \cap A.l \subseteq \mathbf{a}(S.k.0) \quad \textit{for } 0 \leq k,l < n \wedge k \neq l, \tag{3.7}$$

$$Out.i \cap Out.j = \varnothing \quad \textit{for } 0 \leq i,j < n \wedge i \neq j, \tag{3.8}$$

where

$A.k = (\cup i: 0 \leq i < m: \mathbf{a}(S.k.i))$ *for* $0 \leq k < n$,
$Out.i = (\cup k: 0 \leq k < n: \mathbf{o}(S.k.i))$ *for* $1 \leq i < m$, *and*
$Out.0 = (\cup k: 0 \leq k < n: \mathbf{o}(\overline{S.k.0}))$.

□

Condition (3.7) can be interpreted as 'the internal symbols of the decompositions are row-wise disjoint', where the internal symbols of the decomposition of $S.k.0$, $0 \leq k < n$, (i.e. row k) are given by $A.k \setminus \mathbf{a}(S.k.0)$. Condition (3.8) can be interpreted as 'the outputs are column-wise disjoint', where the outputs of column k, $0 \leq i < m$, are given by $Out.i$. (Notice that $Out.0$ represents the outputs of the components $\overline{S.k.0}$, $0 \leq k < n$.)

Proof (of Theorem 3.1.3.0).
Let $R.k.0 = \overline{S.k.0}$ and $R.k.i = S.k.i$ for $1 \leq i < m$ and $0 \leq k < n$.
(*i*) We observe

$$\begin{aligned} & (\cup i: 0 \leq i < m: \mathbf{o}(\|k: 0 \leq k < n: R.k.i)) \\ = & \{\text{calc.}\} \\ & (\cup k: 0 \leq k < n: (\cup i: 0 \leq i < m: \mathbf{o}(R.k.i)) \\ = & \{S.k.0 \rightarrow (i: 0 \leq i < m: S.k.i), \text{ calc.}\} \\ & (\cup k: 0 \leq k < n: (\cup i: 0 \leq i < m: \mathbf{i}(R.k.i)) \end{aligned}$$

$= \{\text{calc.}\}$

$(\cup i: 0 \leqslant i < m: \mathbf{i}(\| k: 0 \leqslant k < n: R.k.i)).$

(*ii*) The property $\mathbf{o}(\| k: 0 \leqslant k < n: R.k.i) \cap \mathbf{o}(\| k: 0 \leqslant k < n: R.k.j) = \varnothing$, for $0 \leqslant i,j < m \wedge i \neq j$, follows directly from condition (3.8) in the theorem.

(*iii*) Let $B = \mathbf{a}(\| k: 0 \leqslant k < n: R.k.0)$. We observe

$$\begin{aligned}
&\mathbf{t}(\| i: 0 \leqslant i < m: (\| k: 0 \leqslant k < n: R.k.i)) \upharpoonright B \\
= &\{\text{calc.}\} \\
&\mathbf{t}(\| k: 0 \leqslant k < n: (\| i: 0 \leqslant i < m: R.k.i)) \upharpoonright B \\
= &\{\text{condition (3.7), Prop. 1.1.2.7, calc.}\} \\
&\mathbf{t}(\| k: 0 \leqslant k < n: (\| i: 0 \leqslant i < m: R.k.i) \upharpoonright B) \\
= &\{\text{calc., condition (3.7)}\} \\
&\mathbf{t}(\| k: 0 \leqslant k < n: (\| i: 0 \leqslant i < m: R.k.i) \upharpoonright \mathbf{a}(R.k.0)) \\
= &\{S.k.0 \rightarrow (i: 1 \leqslant i < m: S.k.i), \text{calc.}\} \\
&\mathbf{t}(\| k: 0 \leqslant k < n: R.k.0).
\end{aligned}$$

(*iv*) Let

$$\begin{aligned}
WC.i &= (\| k: 0 \leqslant k < n: R.k.i), \\
WR.k &= (\| i: 0 \leqslant i < m: R.k.i), \text{ and} \\
W &= (\| i: 0 \leqslant i < m: WC.i).
\end{aligned}$$

Notice that we also have $W = (\| k: 0 \leqslant k < n: WR.k)$. We first prove that under condition (3.8) we have

$$\begin{aligned}
&b \in \mathbf{o}(WC.i) \\
\Rightarrow &(\mathbf{A}k: 0 \leqslant k < n: b \notin \mathbf{a}(WR.k) \vee b \in \mathbf{o}(R.k.i)). \qquad (3.9)
\end{aligned}$$

Let $b \in \mathbf{o}(WC.i)$, i.e. $b \in Out.i$. Let k satisfy $0 \leqslant k < n$. If $b \in \mathbf{a}(WR.k)$, then $b \in \mathbf{o}(R.k.j)$ for some j, $0 \leqslant j < m$, since the components $(i: 0 \leqslant i < m: R.k.i)$ form a closed connection. By condition (3.8) then follows $i = j$.
Second, we derive for arbitrary k, $0 \leqslant k < n$,

$$\begin{aligned}
&t \in \mathbf{t}W \wedge b \in \mathbf{o}(WC.i) \wedge tb \upharpoonright \mathbf{a}(WC.i) \in \mathbf{t}(WC.i) \\
\Rightarrow &\{\text{definition of weaving, (3.9)}\} \\
&t \upharpoonright \mathbf{a}(WR.k) \in \mathbf{t}(WR.k) \wedge (b \notin \mathbf{a}(WR.k) \vee b \in \mathbf{o}(R.k.i)) \\
&\wedge tb \upharpoonright \mathbf{a}(R.k.i) \in \mathbf{t}(R.k.i) \\
\Rightarrow &\{S.k.0 \rightarrow (i: 1 \leqslant i < m: S.k.i), \text{calc.}\} \\
&tb \upharpoonright \mathbf{a}(WR.k) \in \mathbf{t}(WR.k).
\end{aligned}$$

By the definition of weaving, we consequently deduce $tb \in \mathbf{t}W$.

□

In the proof of the Separation Theorem condition (3.7) is only used in (*iv*). For this reason, we conclude

THEOREM 3.1.3.1. *For the components $S.k.i$, $0 \leq k < n \wedge 0 \leq i < m$, we have*

$$S.k.0 \to (i: 1 \leq i < m: S.k.i) \text{ for all } k,\ 0 \leq k < n,$$
$$\wedge\ (3.8)$$
$$\Rightarrow (\| k: 0 \leq k < n: \overline{S.k.0}),\ (i: 0 \leq i < m: (\| k: 0 \leq k < n: S.k.i))$$

forms a closed connection, free of interference.

□

From the Separation Theorem two corollaries can be derived.

COROLLARY 3.1.3.2. *If for components $S0, S1$, and $S0 \| T$ we have $S0 \to S1$, then $S0 \| T \to S1 \| T$.*

PROOF. Take

$$S.0.0 = S0\ ,\ S.0.1 = S1,$$
$$S.1.0 = T\ ,\ S.1.1 = T,$$

and let $S0 \to S1$. Then we have $S.0.0 \to S.0.1$ and $S.1.0 \to S.1.1$. Since there are no internal symbols for these decompositions, condition (3.7) of the Separation Theorem is satisfied. For component $S0 \| T$ we have

$$\mathbf{i}S0 \cap \mathbf{o}T = \varnothing \wedge \mathbf{o}S0 \cap \mathbf{i}T = \varnothing.$$

By $S0 \to S1$, we also have $\mathbf{i}S0 = \mathbf{i}S1 \wedge \mathbf{o}S0 = \mathbf{o}S1$. Since $S0, S1$ and T are components, we infer from the above

$$(\mathbf{i}S0 \cup \mathbf{i}T) \cap (\mathbf{o}S1 \cup \mathbf{o}T) = \varnothing.$$

Consequently, $Out.0 \cap Out.1 = \varnothing$ and condition (3.8) holds. Application of the Separation Theorem yields the desired result.

□

COROLLARY 3.1.3.3. *If for component $(\| k: 0 \leq k < n: T.k)$ we have $\mathbf{o}(T.k) \cap \mathbf{o}(T.l) = \varnothing$ for $0 \leq k,\ l < n \wedge k \neq l$, then*

$$(\| k: 0 \leq k < n: T.k) \to (k: 0 \leq k < n: T.k).$$

PROOF. Take $S.k.0 = T.k$ for $0 \leq k < n$, $S.k.(k+1) = T.k$, and $S.k.i = \epsilon$ for $1 \leq i < (n+1) \wedge (k+1) \neq i$. We have $S.k.0 \to (i: 1 \leq i < (n+1): S.k.i)$. Here as well there are no internal symbols for the decompositions, and condition (3.7) of the Separation Theorem is satisfied. Since $(\|k:0\leqslant k<n: T.k)$ is a component, we have

$$\mathbf{i}(T.k)\cap\mathbf{o}(T.l)=\varnothing \quad \text{for} \quad 0\leqslant k,l<n \wedge k\neq l,$$

i.e. *Out.* 0 and *Out.i* are disjoint for $0<i<(n+1)$. If

$$\mathbf{o}(T.k)\cap\mathbf{o}(T.l)=\varnothing \quad \text{for} \quad 0\leqslant k,l<n \wedge k\neq l,$$

then $Out.i\cap Out.j=\varnothing$ for all $0<i,j<(n+1) \wedge i\neq j$. Accordingly, the outputs are column-wise disjoint, and condition (3.8) of the Separation Theorem can be concluded. Application of this theorem gives the desired result.
□

EXAMPLE 3.1.3.4. We demonstrate how a decomposition for component $S0=\mathbf{pref}[a?;b!;c?d!]\,\|\,\mathbf{pref}[b!;e?]$ can be derived with the above theorems. We observe

$\mathbf{pref}[a?;b!;c?;d!]\,\|\,\mathbf{pref}[b!;e?]$

$\rightarrow$ {Ex. 3.1.1.5 , Cor. 3.1.3.2}

$\mathbf{pref}[a?;b!]\,\|\,\mathbf{pref}[c?;d!]\,\|\,\mathbf{pref}[b!;e?]$

$\rightarrow$ {Cor. 3.1.3.3 , calc.}

$\mathbf{pref}\,[a?;b!]\,\|\,\mathbf{pref}\,[b!;e?]$

, $\mathbf{pref}\,[c?;d!]$.

From these last lines (and the Substitution Theorem) we infer that component $S0$ can be decomposed into a 2-CEL component and a WIRE component. The decomposition is depicted in Figure 3.1.5.

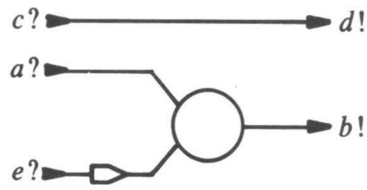


FIGURE 3.1.5. A decomposition of $S0$.

□

More applications of the above theorems and corollaries, and some suggestions for other theorems on decomposition, are given in Chapters 5, 6 and 7.

3.2. DELAY-INSENSITIVITY

3.2.0. *DI decomposition*

In Chapter 2 we stipulated that the behavior of a non-WIRE component (and its environment) is specified at a fixed boundary. For a connection of such components it seems highly unlikely that their fixed boundaries would fit exactly at the connection points. Therefore, in order to connect corresponding input and output terminals in this connection, we introduce WIRE components. The terminals are connected via an *intermediate boundary* as exemplified in Figure 3.2.0. Since WIRE components have flexible boundaries, this intermediate boundary can be placed anywhere between the fixed boundaries of the components.

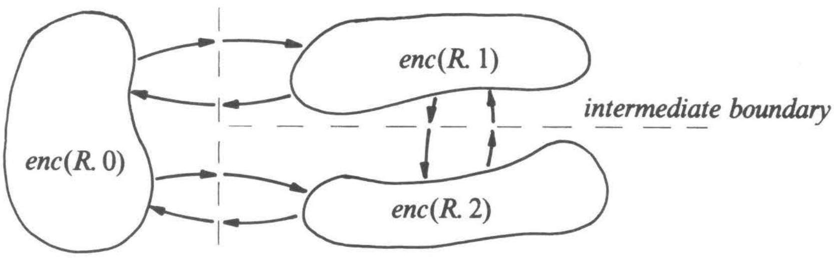


FIGURE 3.2.0. DI decomposition.

Operationally speaking, the WIRE components introduce delays in the communications between components and the intermediate boundary. Thus, they may affect the functional behavior of the connection of components at the intermediate boundaries. If this closed connection operates as specified, irrespective of delays, and the connection is free of interference, then we call such a connection a delay-insensitive connection.

The formalization of a delay-insensitive connection of components is done as follows. For the components $S.k$, $0 \leq k < n$, we define $R.0 = \overline{S.0}$ and $R.k = S.k$, $1 \leq k < n$. Let $\mathbf{a}(R.k)$, $0 \leq k < n$, stand for an intermediate boundary and define the *enclosure* $enc(R.k)$ of this boundary by

$enc(R.k)$ is the trace structure obtained by replacing
each output a in $R.k$ by oa_k and
each input a in $R.k$ by ia_k.

(We assume that the characters i and o do not occur in $R.k$). For each k, $0 \leq k < n$ and $a \in \mathbf{a}(R.k)$ we introduce the WIRE component $Wire(k,a)$ between the boundary of the enclosure and the intermediate boundary by

$$\begin{aligned} Wire(k,a) &= \mathbf{pref}[oa_k?;a!] \quad \text{if } a \in \mathbf{o}(R.k) \\ &= \mathbf{pref}[a?;ia_k!] \quad \text{if } a \in \mathbf{i}(R.k). \end{aligned}$$

The collection of WIRE components for $R.k$, $0 \leq k < n$, and its weave are defined by

$$Wires(R.k) = (a : a \in \mathbf{a}(R.k) : Wire(k,a))$$

$$WWires(R.k) = (\| a : a \in \mathbf{a}(R.k) : Wire(k,a)).$$

With these definitions we can formulate

DEFINITION 3.2.0.0. *We say that the components* $S.k$, $1 \leq k < n$ *form a DI decomposition of component* $S.0$, *denoted by*

$$S.0 \overset{DI}{\rightarrow} (k : 1 \leq k < n : S.k),$$

if all components $enc(R.k)$ *and* $Wires(R.k)$, $0 \leq k < n$, *form a closed connection, free of interference, and*

$$\mathbf{t}(\| k : 0 \leq k < n : enc(R.k) \parallel WWires(R.k)) \upharpoonright \mathbf{a}(R.0) = \mathbf{t}(R.0).$$

□

Notice that the last condition requires that the connection behaves as specified at the intermediate boundary $\mathbf{a}(R.0)$. Thus, we incorporate the delays in the communications not only with the components $S.k$, $1 \leq k < n$, but also with environment $S.0$.

EXAMPLE 3.2.0.1. We have the relations

$$\mathbf{pref}[a?; b! \| c!] \overset{DI}{\rightarrow} \mathbf{pref}[a?; b! \| c!], \text{ and}$$

$$\mathbf{pref}[a?; b! \| c!] \overset{DI}{\rightarrow} \mathbf{pref}[a?; b!; c!].$$

Notice that the ordering between outputs b and c for component $\mathbf{pref}[a?;b!;c!]$ is lost at the intermediate boundary due to the 'delays' introduce by the WIRE components. Consequently, there does not exist a DI decomposition of this component that can realize this ordering between outputs b and c, i.e. we do not have,

$$\mathbf{pref}[a?;b!;c!] \overset{DI}{\rightarrow} \mathbf{pref}[a?;b!;c!].$$

□

3.2.1. DI components

In this thesis we are interested in DI decompositions of a component. In general, DI decompositions are more difficult to verify or derive than decompositions. The two decompositions are equivalent, however, if all components involved are so-called DI components. DI components are defined by

DEFINITION 3.2.1.0. Component S is called a DI component, if

$$S \rightarrow Wires(S), enc(S).$$

□

Since WIRE components have flexible boundaries, it follows from Definition 3.2.1.0 that a DI component can be characterized as a component whose specification is valid at a flexible boundary.

We have

THEOREM 3.2.1.1. *If all components $S.i$, $0 \leqslant i < n$, are DI components, then*

$$S.0 \rightarrow (i: 1 \leqslant i < n: S.i) \equiv S.0 \overset{DI}{\rightarrow} (i: 1 \leqslant i < n: S.i).$$

PROOF. Let $R.0 = \overline{S.0}$ and $R.i = S.i$, $1 \leqslant i < n$. First, we make two observations. We infer

$$\begin{aligned}
&\text{components } R.i,\ 0 \leqslant i < n, \text{ form a} \\
&\text{closed connection, free of interference} \\
\Rightarrow\ &\{\text{Th. 3.1.2.1, } R.i \rightarrow Wires(R.i), enc(R.i) \quad \text{for } 0 \leqslant i < n\} \qquad (3.10) \\
&\text{components } enc(R.i) \text{ and } Wires(R.i),\ 0 \leqslant i < n, \text{ form a} \\
&\text{closed connection, free of interference} \\
&\wedge\ (3.11),
\end{aligned}$$

where (3.11) stands for the equality

$$\begin{aligned}
&(\|i: 0 \leqslant i < n: enc(R.i) \,\|\, WWires(R.i)) \\
= \ &(\|i: 0 \leqslant i < n: enc(R.i) \,\|\, WWires(R.i) \,\|\, R.i). \qquad (3.11)
\end{aligned}$$

Second, we derive

$$\begin{aligned}
&(\|i: 0 \leqslant i < n: enc(R.i) \,\|\, WWires(R.i)) \restriction \mathbf{a}(R.0) \\
= \ &\{(3.11)\} \\
&(\|i: 0 \leqslant i < n: enc(R.i) \,\|\, WWires(R.i) \,\|\, R.i) \restriction \mathbf{a}(R.0) \\
= \ &\{\text{Prop. 1.1.2.7 with } A,B := \mathbf{a}(R.0), \mathbf{a}(R.i) \text{ for } 0 \leqslant i < n\} \qquad (3.12) \\
&(\|i: 0 \leqslant i < n: (enc(R.i) \,\|\, WWires(R.i) \,\|\, R.i) \restriction \mathbf{a}(R.i)) \restriction \mathbf{a}(R.0) \\
= \ &\{R.i \rightarrow Wires(R.i), enc(R.i) \text{ ,calc.}\} \\
&(\|i: 0 \leqslant i < n: R.i) \restriction \mathbf{a}(R.0).
\end{aligned}$$

With these observations the proof goes as follows.

Let $S.0 \rightarrow (i: 1 \leqslant i < n: S.i)$ hold. By (3.10) we infer that the components $enc(R.i)$ and $Wires(R.i)$, $0 \leqslant i < n$, form a closed connection, free of interference and (3.11) holds. With (3.12) we infer

$$S.0 \rightarrow (i: 1 \leq i < n: S.i)$$

$$\Rightarrow \{\text{def. of decomposition}\}$$

$$\mathbf{t}(\|i: 0 \leq i < n: R.i) \upharpoonright \mathbf{a}(R.0) = \mathbf{t}(R.0)$$

$$\Rightarrow \{(3.12), (3.11)\}$$

$$\mathbf{t}(\|i: 0 \leq i < n: enc(R.i) \,\|\, WWires(R.i)) \upharpoonright \mathbf{a}(R.0) = \mathbf{t}(R.0).$$

Consequently, $S.0 \overset{DI}{\rightarrow} (i: 1 \leq i < n: S.i)$.

Let $S.0 \overset{DI}{\rightarrow} (i: 1 \leq i < n: S.i)$ hold. By definition of $enc(R.i)$ and $WWires(R.i)$ we derive

components $enc(R.i)$ and $Wires(R.i)$, $0 \leq i < n$,

form a closed connection, free of output interference

$\Rightarrow$ {calc.}

components $R.i$, $0 \leq i < n$,

form a closed connection, free of output interference

Consider the special behavior in the closed connection of components $enc(R.i)$ and $Wires(R.i)$, $0 \leq i < n$, where each output oa_i, $0 \leq i < n \wedge a \in \mathbf{o}(R.i)$, is immediately followed by a and all ia_j, $0 \leq j < n \wedge a \in \mathbf{i}(R.j)$. Operationally speaking, we assume that the communications by the WIRE components are instantaneous communications. Since in this special behavior computation interference does not occur, there is no computation interference in the connection of components $R.i$, $0 \leq i < n$, either. Accordingly, we have that the components $R.i$, $0 \leq i < n$, form a closed connection, free of interference. By (3.10) and (3.12) we then infer

$$S.0 \overset{DI}{\rightarrow} (i: 1 \leq i < n: S.i)$$

$$\Rightarrow \{\text{def. of DI decomposition}\}$$

$$\mathbf{t}(\|i: 0 \leq i < n: enc(R.i) \,\|\, WWires(R.i)) \upharpoonright \mathbf{a}(R.0) = \mathbf{t}(R.0)$$

$$\Rightarrow \{(3.10), (3.12)\}$$

$$\mathbf{t}(\|i: 0 \leq i < n: R.i) \upharpoonright \mathbf{a}(R.0) = \mathbf{t}(R.0).$$

Consequently, $S.0 \rightarrow (i: 1 \leq i < n: S.i)$.
□

From now on, we mostly restrict ourselves to DI components and decompositions. By Theorem 3.2.1.1, it then follows that such decompositions are DI decompositions.

We say that a component $S.0$ is *DI decomposable* if there exists a collection of components $S.i$, $0\leqslant i<n$, that form a DI decomposition of $S.0$.

Remark. It can happen that for a given component decompositions exist in which not every component is a DI component. If this component is realized by a circuit according to such a decomposition but with the use of connection wires, then this circuit may malfunction: some delays can cause incorrect behavior. In order for this circuit to operate correctly, delay requirements must be met. We try to avoid such requirements as long as possible.
□

The following two theorems can be used to infer whether a component is DI. From the definition of DI decomposition and DI component we derive

Theorem 3.2.1.2. *If a component is DI decomposable, then it is a DI component.*

Proof. Let $S.0 \xrightarrow{DI} (i:0\leqslant i<n: S.i)$. Take $R.0=\overline{S.0}$, $R.i=S.i$, $1\leqslant i<n$, and define T by $\mathbf{i}T=\mathbf{i}(S.0)$, $\mathbf{o}T=\mathbf{o}(S.0)$,

$$\mathbf{t}T=\mathbf{t}(i:0\leqslant i<n: enc(R.i) \,\|\, WWires(R.i))\upharpoonright\mathbf{a}(R.0).$$

Since the components $enc(R.i)$ and $Wires(R.i)$, $0\leqslant i<n$, form a closed connection, free of interference, we infer that the connection $enc(R.0)$, $Wires(R.0)$, and T is closed and free of interference as well. By definition of DI decomposition we have $T=S.0$. Accordingly, also $\overline{S.0}$, $Wires(S.0)$, $enc(S.0)$ is a closed connection, free of interference. Moreover, for any $S.0$ we have

$$(enc(S.0) \,\|\, WWires(S.0) \,\|\, \overline{S.0})\upharpoonright\mathbf{a}(\overline{S.0}) \;=\; \mathbf{t}(\overline{S.0}).$$

Accordingly, we conclude $S.0\rightarrow enc(S.0), Wires(S.0)$.
□

Consequently, if a component is not a DI component, then it is not DI decomposable.

Theorem 3.2.1.3. *For a component S we have*

$$S \textit{ is DI} \equiv S \xrightarrow{DI} S.$$

Proof. From Theorem 3.2.1.1 and the property $S\rightarrow S$, we infer S is DI $\Rightarrow S \xrightarrow{DI} S$. From Theorem 3.2.1.2, we derive $S \xrightarrow{DI} S \Rightarrow S$ is DI.
□

The characterization of a DI component S by the property $S\rightarrow Wires(S)$, $enc(S)$ can be considered as a formalization of the so-called *Foam Rubber Wrapper* (FRW) principle. Formally speaking, the FRW principle states that the specification of a component is invariant under the extension by WIRE components. Operationally speaking, the FRW metaphor expresses that the circuit specified by S is embedded in a 'Foam Rubber Wrapper' formed by the connection wires. The boundaries of the FRW are constituted by $\mathbf{a}\,enc(S)$ and $\mathbf{a}S$, as depicted in Figure 3.2.1.

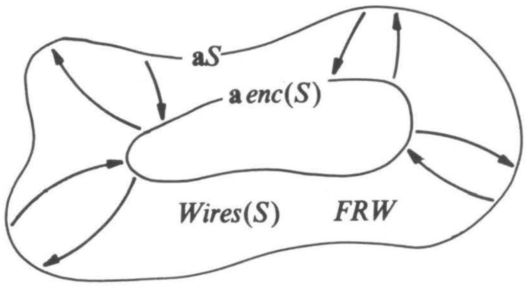


FIGURE 3.2.1. The Foam Rubber Wrapper principle $S \rightarrow Wires(S), enc(S)$.

The idea of formalizing delay-insensitivity by means of the FRW principle originates from Charles E. Molnar [33]. Jan Tijmen Udding was the first to give a rigorous formulation of this principle in terms of trace structures. In [45] he postulates a number of rules which a component must satisfy in order to meet the FRW principle. It turns out that Udding's definition of a DI component is equivalent to Definition 3.2.1.0 (cf. Theorem 4.1.0). T.P. Fang had earlier expressed the FRW principle −though less completely− by means of Petri Net rules. In [38] another formalization of the FRW principle is given by Huub Schols. For a proof of the equivalence of Udding's and Schols's formalization we refer to [38, 39].

Chapter 4

DI Grammars

4.0. Introduction

In order to apply Theorem 3.2.1.1 we have to know whether a component is a DI component or not. The recognition of DI components is the subject of this chapter. We present two methods for recognizing a DI component: DI grammars, which make up most of this chapter, and Udding's classification.

In [45] Jan Tijmen Udding postulates a number of rules with which the classes $C1$, $C2$, $C3$, and $C4$ of trace structures are defined. A class consists of all trace structures that satisfy a specific set of rules. It turns out that the largest class, i.e. class $C4$, is the class of all DI components. Udding's classification is briefly presented in Section 4.1.

The remaining sections of this chapter concern the definitions of so-called DI grammars. A grammar is called a *DI grammar* if it generates commands that represent DI components. Commands that represent DI components are called *DI commands.* DI grammars are attractive for two reasons. First, they enable a syntactical verification of the DI property, and, second, they can be used as a starting point for a syntax-directed decomposition method. At the end of this chapter, we show in a number of examples how a DI grammar can be used to verify whether a command is a DI command and to derive a DI command from a non-DI command. In the next chapters, a hierarchy of DI grammars is used to develop a syntax-directed decomposition method.

With the DI grammars of this chapter a large class of DI commands can be derived, although we conjecture that not every DI command can be derived with these grammars. Accordingly, the DI grammars may be used to prove that a command is a DI command, but in order to prove that a command is not DI we have to resort to other means such as Definition 3.2.1.0 or Udding's classification. The recognition of a DI command by means of a DI grammar is simple and straightforward, whereas the recognition of a DI component by means of Definition 3.2.1.0 or Udding's classification can be tedious.

The grammars defined in this chapter are *attribute grammars*. Attribute grammars are briefly explained in Section 4.2. The largest DI grammar, i.e. grammar $G4$, is then defined in Sections 4.3 to 4.7. In Sections 4.7 and 4.8 the grammars $G4'$, $G3'$, $G2'$, $G1'$, and GCL' are defined, which are all derived from grammar $G4$.

4.1 Udding's classification

We briefly summarize Udding's classification. For a more extensive discussion of this classification the reader is referred to [45].

In the following rules, the letter R denotes a directed trace structure with $\mathbf{int}\,R = \varnothing$, s and t denote arbitrary traces, and a,b, and c denote arbitrary symbols from $\mathbf{a}R$.

rule 1: (R is a component)
R is prefix-closed, non-empty, and $\mathbf{i}R \cap \mathbf{o}R = \varnothing$.

rule 2: (Absence of transmission interference)
$saa \notin \mathbf{t}R$.

rule 3: (Symbols of the same type commute)
If a and b are symbols of the same type, then $sabt \in \mathbf{t}R \equiv sbat \in \mathbf{t}R$.

rule 4′: (Symbols of distinct type commute (0))
If a and b are symbols of distinct type, then $sabt \in \mathbf{t}R \wedge sb \in \mathbf{t}R \Rightarrow sbat \in \mathbf{t}R$.

rule 4″: (Symbols of distinct type commute (1))
If a and b are symbols of distinct type and symbol c is of the same type as a, then $sabtc \in \mathbf{t}R \wedge sbat \in \mathbf{t}R \Rightarrow sbatc \in \mathbf{t}R$.

rule 5′: (No disabling)
If a and b are distinct symbols, then $sa \in \mathbf{t}R \wedge sb \in \mathbf{t}R \Rightarrow sab \in \mathbf{t}R$.

rule 5 ″: (Possible disabling of inputs)
If a and b are distinct symbols, not both inputs of R, then $sa \in \mathbf{t}R \wedge sb \in \mathbf{t}R \Rightarrow sab \in \mathbf{t}R$.

rule 5‴: (Possible disabling of inputs or outputs)
If a and b are distinct symbols of different type, then $sa \in \mathbf{t}R \wedge sb \in \mathbf{t}R \Rightarrow sab \in \mathbf{t}R$.

A class is defined by the collection of all trace structures R that satisfy a certain subset of the above rules. All trace structures R that satisfy

rule 1, 2, 3, 4′, and 5′ form class $C1$,

rule 1, 2, 3, 4′, and 5″ form class $C2$,

rule 1, 2, 3, 4′, and 5‴ form class $C3$,

rule 1, 2, 3, 4″, and 5‴ form class $C4$.

There exists a subset relation between these classes, viz. $C1 \subset C2 \subset C3 \subset C4$. We have

THEOREM 4.1.0. *R is DI* $\equiv R \in C4$.

PROOF. See Appendix A.
□

EXAMPLE 4.1.1. Consider the following components.

$$R.0 = \textbf{pref}\,(a?;b?;c!)\ ,$$
$$R.1 = \textbf{pref}\,[a?\|b?;c!]\ ,$$
$$R.2 = \textbf{pref}\,[a?;c!\,|\,b?;c!]\ ,$$
$$R.3 = \textbf{pref}\,[n?;(a!|b!)]\ ,$$
$$R.4 = \textbf{pref}\,(a!\|b?\ |\ b?;a!\|c!)\ ,$$
$$R.5 = \textbf{pref}\,((a?;d!)^2\ |\ (b?;e!)^2\ |\ (a?;d!\|c!)^2\ \|\ (b?;e!\|c!)^2)\ , \text{ and}$$
$$R.6 = \textbf{pref}\,[(a?)^2\,|\,(b?)^2\,|\,(a?\|b?;c!)^2]\ .$$

By inspection, we infer that $R.0 \notin C4$, since rule 3 is not satisfied. Similarly, $R.6 \notin C4$, since rule 2 is not satisfied. For the other trace structures we have

$$R.1 \in C1\,,\ R.2 \in C2\,,\ R.3 \in C3\,,\ R.4 \in C4, \text{ and } R.5 \in C2.$$

Notice that in $R.1$ there is no disabling of symbols; in $R.2$ there is a disabling between inputs; and in $R.3$ there is a disabling between outputs. For $R.4$ we observe that rule 4′ is not satisfied, though rule 4″ is satisfied, as well as rules 1, 2, 3, and 5′.
□

As the reader may have noticed in Example 4.1.1, verifying whether a component is DI by means of the rules for $C4$, $C3$, $C2$ or $C1$ can be tedious. For many components, represented by a command, a simple syntactical verification can also be applied, as is shown in the next sections.

4.2. Attribute grammars

The DI grammars defined in this chapter are attribute grammars. We briefly explain those properties of an attribute grammar that are needed to understand the next sections.

An attribute grammar consists of

- a *context-free grammar*
- a set of *attributes* for each grammar symbol
- a *condition* for each production rule, and
- a set of *evaluation rules* for each production rule.

In the attribute grammars of the next sections, the attributes, the conditions, and the evaluation rules are used to restrict the derivations of the context-free grammar. We explain how these restrictions are formulated.

Each derivation in the context-free grammar has a parse tree, and each node in that parse tree corresponds to a grammar symbol. The attributes of this grammar symbol are also associated with this node. For each attribute in the parse tree, its value is calculated according to the conditions and the evaluation rules of the grammar as follows.

The values of the attributes of each internal node are calculated from the values of its children. These calculations are specified in the evaluation rules which are associated with the production rule that is applied in that node. Attributes thus calculated are called *synthesized attributes* (as opposed to inherited attributes). The values of the attributes of the leaves are assumed to be given.

The values of the attributes in a node are calculated only if the condition for the production rule holds. The condition is formulated in terms of the attributes of the children of that node. If in all nodes the condition for the production rule holds, then the derivation is called a derivation of the attribute grammar. Thus, derivations of the context-free grammar are restricted to derivations of the attribute grammar.

In the following sections, the context-free grammar, the attributes, the conditions, and the evaluation rules for grammar $G4$ are defined. We then show that any derivable command of this grammar is a DI command.

4.3. The context-free grammar of $G4$

Below, the context-free grammar of the attribute grammar $G4$ is defined. In Table 4.3.0 the production rules are listed. The symbol [] is a meta symbol of the grammar; it separates the alternative productions. The prefixes *pc* and *pf* stand for prefix-closed and prefix-free respectively.

<table>
<tr><td><<i>dicom</i>> ::=</td><td></td><td><<i>pccom</i>></td><td>(<i>a</i>0)</td></tr>
<tr><td></td><td>▯</td><td>(<<i>pccom</i>>)↾</td><td>(<i>a</i>1)</td></tr>
<tr><td><<i>pccom</i>> ::=</td><td></td><td>ε</td><td></td></tr>
<tr><td></td><td>▯</td><td>μ.<<i>tailf</i>>.0</td><td>(<i>b</i>0)</td></tr>
<tr><td></td><td>▯</td><td><<i>pccom</i>>||<<i>pccom</i>></td><td>(<i>b</i>1)</td></tr>
<tr><td></td><td>▯</td><td>pref(<<i>pfcom</i>>)</td><td>(<i>b</i>2)</td></tr>
<tr><td></td><td>▯</td><td>pref[<<i>pfcom</i>>]</td><td>(<i>b</i>3)</td></tr>
<tr><td><<i>pfcom</i>> ::=</td><td></td><td><<i>marked syms</i>></td><td>(<i>c</i>0)</td></tr>
<tr><td></td><td>▯</td><td><<i>pfcom</i>> ; <<i>pfcom</i>></td><td>(<i>c</i>1)</td></tr>
<tr><td></td><td>▯</td><td><<i>pfcom</i>> | <<i>pfcom</i>></td><td>(<i>c</i>2)</td></tr>
<tr><td></td><td>▯</td><td>(<<i>pfcom</i>>)</td><td>(<i>c</i>3)</td></tr>
<tr><td><<i>marked syms</i>> ::=</td><td></td><td><<i>sym</i>>? ▯ <<i>sym</i>>? || <<i>sym</i>>?</td><td></td></tr>
<tr><td></td><td>▯</td><td><<i>sym</i>>! ▯ <<i>sym</i>>! || <<i>sym</i>>!</td><td></td></tr>
<tr><td></td><td>▯</td><td>!<<i>sym</i>>?</td><td></td></tr>
<tr><td></td><td>▯</td><td>?<<i>sym</i>>!</td><td></td></tr>
</table>

TABLE 4.3.0. The production rules of grammar *G*4.

The symbols <*sym*>, <*tailf*>, and all characters in the above table not enclosed by the < > brackets are terminal symbols of the grammar. All other symbols in Table 4.3.0 are non-terminals. The start symbol is <*dicom*>. The terminal <*sym*> represents a symbol from a sufficiently large alphabet. The terminal <*tailf*> represents a tail function defined by an array of commands $E(i,j: 0 \leq i,j < n)$, i.e. if $\mu.tailf.0$ is an instance of <*pccom*>, then the tail function *tailf* is defined by

$$tailf.R.i = \mathbf{pref}\,(|j: 0 \leq j < n: E.i.j;R.j), \quad 0 \leq i < n.$$

Later, when we define the conditions for production rule (*b*0), these conditions are formulated for array *E*. For example, we require $E.i.j \in$ <*pfcom*> for all i,j with $0 \leq i,j < n \wedge E.i.j \neq \varnothing \wedge E.i.j \neq \epsilon$. Thus, implicitly, commands of type <*pfcom*> are used in the application of rule (*b*0).

With the above context-free grammar, commands of the form *E* or *E*↾ can be derived, where *E* is expressed as a weave of (special) sequential commands.

4.4. THE ATTRIBUTES OF *G*4

At most eight attributes are associated with each grammar symbol. The attributes are represented by the names

O, *I*, *EN*, *CO*, *HD*, *TL*, *FIRST*, and *FIRSTEXT*.

All eight attributes are associated with the grammar symbols <*marked syms*>

and <*pfcom*>. With the grammar symbol <*pccom*> only the attributes *O*, *I*, *EN*, and *CO* are associated. The grammar symbol <*dicom*> has no attributes.

The evaluation rules and conditions are defined in such a way that the following semantics can be attached to the attributes. (This is proven in Appendix B.) For a command *E* derivable in (attribute) grammar *G*4 we have

$$I(E) = \mathbf{i}\,E,$$
$$O(E) = \mathbf{o}\,E,$$
$$EN(E) = \mathbf{en}\,E,$$
$$CO(E) = \mathbf{co}\,E.$$

The attributes *HD* and *TL* indicate with what kind of marks a command *E* starts and ends respectively. For a command *E* derivable in grammar *G*4 we have

$$
\begin{array}{ll}
HD(E) = empty & \text{if } E=\epsilon, \\
HD(E) = in & \text{if } E\neq\epsilon \wedge \mathbf{hd}E \subseteq \mathbf{i}E \cup \mathbf{en}E, \\
HD(E) = out & \text{if } E\neq\epsilon \wedge \mathbf{hd}E \subseteq \mathbf{o}E \cup \mathbf{co}E, \\
HD(E) = mixed & \text{otherwise,}
\end{array}
$$

$$
\begin{array}{ll}
TL(E) = empty & \text{if } E=\epsilon, \\
TL(E) = in & \text{if } E\neq\epsilon \wedge \mathbf{tl}E \subseteq \mathbf{i}E \cup \mathbf{co}E, \\
TL(E) = out & \text{if } E\neq\epsilon \wedge \mathbf{tl}E \subseteq \mathbf{o}E \cup \mathbf{en}E, \\
TL(E) = mixed & \text{otherwise ,}
\end{array}
$$

where $\mathbf{hd}E=\{b\,|\,\mathbf{E}t::bt\in\mathbf{t}E)\}$, and $\mathbf{tl}E=\{b\,|\,\mathbf{E}t::tb\in\mathbf{t}E)\}$. For example, for the command $E=a?\|b?;c!\mid ?d!;e?$, we have $HD(E)=in$ and $TL(E)=mixed$.

The attributes *FIRST* and *FIRSTEXT* represent a kind of 1-lookahead sets for a command. The type of these attributes is a set of sets of symbols (instead of a set of symbols for usual 1-lookahead sets). In the case of *FIRSTEXT* these sets of symbols consist of external symbols only. For a derivable command *E* in grammar *G*4 we have

$$FIRST(\epsilon)=\{\varnothing\} \wedge FIRSTEXT(\epsilon)=\{\varnothing\}.$$

If $HD(E)=out$, then

$$
\begin{aligned}
&FIRST(E) \\
&= \{set(t)\,|\,t\in(\mathbf{o}E)^* \wedge t\in\mathbf{t}\,\mathbf{pref}\,E \wedge t\neq\epsilon \wedge (Suc(t,E)\setminus \mathbf{o}E\neq\varnothing \vee Suc(t,E)=\varnothing)\} \\
&\quad \cup \{\{b\}\,|\,b\in\mathbf{co}E \wedge b\in\mathbf{t}\,\mathbf{pref}\,E\},
\end{aligned}
$$

and

$$
\begin{aligned}
&FIRSTEXT(E) \\
&= \{set(t\restriction\mathbf{ext}E)\,|\,t\in(\mathbf{o}E\cup\mathbf{co}E)^* \wedge t\in\mathbf{t}\,\mathbf{pref}\,E \\
&\qquad\qquad \wedge (Suc(t,E)\setminus(\mathbf{o}E\cup\mathbf{co}E)\neq\varnothing \vee Suc(t,E)=\varnothing)\}.
\end{aligned}
$$

Here, $set(t)$ denotes the set of symbols occurring in t. If $HD(E)=in$, then $FIRST(E)$ and $FIRSTEXT(E)$ are defined similarly, except that $\mathbf{o}E$ and $\mathbf{co}E$ are replaced by $\mathbf{i}E$ and $\mathbf{en}E$ respectively. Notice that for $\mathbf{int}\,E=\varnothing$ we have $FIRST(E)=FIRSTEXT(E)$. The elements of $FIRST(E)$ are sets of (concurrent) external symbols or singletons of internal symbols. The set $FIRSTEXT(E)$ contains sets of (concurrent) external symbols only. For example, for the command $E = a?\|b?;c!\ |\ ?d!;e?$, we obtain $FIRST(E)=\{\{a,b\},\{d\}\}$ and $FIRSTEXT(E)=\{\{a,b\},\{e\}\}$.

4.5 The conditions for $G4$

The conditions for the production rules are formulated with five predicates. These predicates are *ALFCOND, PROCOND, SEQCOND, ALTCOND,* and *TAILCOND*. They correspond to a condition for the alphabets, a condition expressing whether projection has to be applied, a condition for the sequential construct, a condition for the alternative construct, and a condition for the tail-recursive construct respectively.

$ALFCOND(E0,E1)$, $PROCOND(E)$, $SEQCOND(E0,E1)$, and $ALTCOND(E0,E1)$ are defined on commands derivable in $G4$ by

$$
\begin{array}{ll}
ALFCOND(E0,E1) & \equiv (\mathbf{A}\ ATT0,ATT1 \\
 & \quad : ATT0,ATT1 \in \{I,O,EN,CO\} \wedge ATT0 \neq ATT1 \\
 & \quad : ATT0(E0) \cap ATT1(E1)=\varnothing \\
 & \quad), \\
PROCOND(E) & \equiv EN(E)=\varnothing \wedge CO(E)=\varnothing, \\
SEQCOND(E0,E1) & \equiv \ (TL(E0)=in \wedge HD(E1)=out) \\
 & \quad \vee (TL(E0)=out \wedge HD(E1)=in) \\
 & \quad \vee (TL(E0)=empty \wedge HD(E1)\neq mixed) \\
 & \quad \vee (TL(E0)\neq mixed \wedge HD(E1)=empty), \\
ALTCOND(E0,E1) & \equiv \ HD(E0)\neq mixed \wedge HD(E0)=HD(E1) \\
 & \quad \wedge LLCOND(E0,E1) \\
 & \quad \wedge LLCONDEXT(E0,E1), \text{ where} \\
LLCOND(E0,E1) & \equiv \ (FIRST(E0)=\{\varnothing\} \wedge FIRST(E1)=\{\varnothing\}) \\
 & \quad \vee (\mathbf{A}\ A,B : A \in FIRST(E0) \wedge B \in FIRST(E1) \\
 & \qquad\qquad : \neg(A \subseteq B) \wedge \neg(B \subseteq A)\ )
\end{array}
$$

and $LLCONDEXT(E0,E1)$ is defined analogously with *FIRST* replaced by *FIRSTEXT*.

The condition $ALTCOND(E0,E1)$ requires that $E0$ and $E1$ start with marks of the same type and that the LL-1 conditions, both with respect to all types of symbols and with respect to external symbols only, are satisfied. These LL-1 conditions are a kind of generalized LL-1 conditions for LL-1 grammars. Notice that when the *FIRST* sets are non-empty and consist of singletons only we have

$$LLCOND(E0,E1) \equiv FIRST(E0) \cap FIRST(E1)=\varnothing .$$

The condition *TAILCOND*(*tailf*) consists of seven conditions defined on array $E(i,j:0\leq i,j<n)$ that determines the tail function *tailf*. Some of the conditions defined above appear in a more general form in these seven conditions. In the conditions defined below, the domain restrictions $D(i,j)$ stand for $0\leq i,j<n \wedge E.i.j\neq\varnothing \wedge E.i.j\neq\epsilon$; by $E\in <pfcom>$ we denote that E is a production of $<pfcom>$ in the attribute grammar. We have

$$TAILCOND(tailf) \equiv (0) \wedge (1) \wedge (2) \wedge (3) \wedge (4) \wedge (5) \wedge (6), \text{ where}$$

$$(0) \equiv (\mathbf{A}i: 0\leq i<n: (Ej: 0\leq j<n: E.i.j\neq\varnothing))$$

$$(1) \equiv \quad (\mathbf{A}i,j: 0\leq i,j<n \wedge i\neq j: E.i.j\neq\epsilon) \\ \wedge (\mathbf{A}i: 0\leq i<n: E.i.i=\epsilon \Rightarrow (\mathbf{A}j: 0\leq j<n \wedge i\neq j: E.i.j=\varnothing))$$

$$(2) \equiv ALFCOND(i,j: D(i,j): E.i.j)$$

$$(3) \equiv (\mathbf{A}i,j: D(i,j): E.i.j\in <pfcom>)$$

$$(4) \equiv (\mathbf{A}i,j,k: 0\leq i,j,k<n \wedge E.i.j\neq\varnothing \wedge E.j.k\neq\varnothing: SEQCOND(E.i.j,\ E.j.k))$$

$$(5) \equiv (\mathbf{A}i: 0\leq i<n: ALTCOND(j: D(i,j): E.i.j))$$

$$(6) \equiv \quad (\mathbf{A}i,j: D(i,j): FIRSTEXT(E.i.j)\neq\{\varnothing\}) \\ \vee (\mathbf{A}i,j: D(i,j): FIRSTEXT(E.i.j)=\{\varnothing\}),$$

where

$$ALFCOND\ (i,j: D(i,j): E.i.j) \\ \equiv (\mathbf{A}i,j,k,l: D(i,j) \wedge D(k,l): ALFCOND(E.i.j,\ E.k.l))$$

and, for $0\leq i<n$, if $(\mathbf{N}j::D(i,j))\leq 1$, then $ALTCOND(j: D(i,j): E.i.j)\equiv true$; otherwise,

$$ALTCOND(j: D(i,j): E.i.j) \\ \equiv ((\mathbf{A}j: D(i,j): HD(E.i.j)=in) \vee (\mathbf{A}j: D(i,j): HD(E.i.j)=out)) \\ \wedge\ LLCOND(j: D(i,j): E.i.j) \wedge LLCONDEXT(j: D(i,j): E.i.j)$$

$$LLCOND(j: D(i,j): E.i.j) \\ \equiv \quad (Aj: D(i,j): FIRST(E.i.j)=\{\varnothing\}) \\ \vee (\mathbf{A}j,k,A,B: D(i,j) \wedge D(i,k) \wedge j\neq k \wedge \\ A\in FIRST(E.i.j) \wedge B\in FIRST(E.i.k) \\ : \neg(A\subseteq B))$$

and analogously for $LLCONDEXT\ (j: D(i,j): E.i.j)$ with *FIRST* replaced by

FIRSTEXT.

Condition 3 requires that every command *E.i.j*, with *i,j* satisfying *D.i.j*, is of type <*pfcom*>. Condition 2, 4, and 5 are generalizations of the alphabet condition, the condition for the sequential construct, and the condition for the alternative construct respectively. The conditions 1 and 6 only are new conditions.

In Table 4.5.0 the conditions for the production rules of attribute grammar *G*4 are listed. Those production rules that are not listed do not have a condition.

Production rule	Production	Condition
$(a0)$	E	$PROCOND(E)$
$(b0)$	$\mu.tailf.0$	$TAILCOND(tailf)$
$(b1)$	$E0 \| E1$	$ALFOND(E0,E1)$
$(b3)$	**pref** $[E]$	$SEQCOND(E,E)$
$(c1)$	$E0;E1$	$SEQCOND(E0,E1) \wedge$ $ALFCOND(E0,E1)$
$(c2)$	$E0 \vert E1$	$ALTCOND(E0,E1) \wedge$ $ALFCOND(E0,E1)$

TABLE 4.5.0. The conditions for grammar *G*4.

Combined, the conditions may be summarized as follows.

(*i*) (The alphabet condition)
For any symbol used, all atomic commands in which it occurs are of the same type.

(*ii*) (The semicolon condition)
Input and output marks alternate. (This also holds for the repetitive construct and between state transitions in a tail function.)

(*iii*) (The bar condition)
In every alternative construct (also in a tail function) the alternatives start with marks of the same type and both LL-1 conditions are satisfied.

(*iv*) (The tail-function condition)
The array of each tail function satisfies three additional conditions:
- Each row contains a non-empty command
- Only a command at the diagonal can be ϵ, and if a diagonal element is ϵ, then all other commands in that row are $\emptyset$.
- Either all commands different from ϵ and $\emptyset$ contain external symbols, or all of them do not.

(*v*) (The non-projection condition)
If a command does not contain projection, then it does not contain internal symbols.

4.6. THE EVALUATION RULES FOR $G4$

If the condition for a production rule in a node of the parse tree holds, then the values of the attributes in that node can be calculated. The values of the attributes in the leaves, i.e. for commands of type <*marked syms*> and ϵ, are given in Table 4.6.0. These values are used to start the evaluation process.

Command E	Values for attributes of E	
$a?$	$I(E)=\{a\}, O(E)=\varnothing$ $HD(E)=in$ $FIRST(E)=\{\{a\}\}$	$, EN(E)=\varnothing, CO(E)=\varnothing,$ $, TL(E)=in,$ $, FIRSTEXT(E)=\{\{a\}\}.$
$a!$	$I(E)=\varnothing, O(E)=\{a\}$ $HD(E)=out$ $FIRST(E)=\{\{a\}\}$	$, EN(E)=\varnothing, CO(E)=\varnothing,$ $, TL(E)=out,$ $, FIRSTEXT(E)=\{\{a\}\}.$
$?a!$	$I(E)=\varnothing, O(E)=\varnothing$ $HD(E)=in$ $FIRST(E)=\{\{a\}\}$	$, EN(E)=\{a\}, CO(E)=\varnothing,$ $, TL(E)=out,$ $, FIRSTEXT(E)=\{\varnothing\}.$
$!a?$	$I(E)=\varnothing, O(E)=\varnothing$ $HD(E)=out$ $FIRST(E)=\{\{a\}\}$	$, EN(E)=\varnothing, CO(E)=\{a\},$ $, TL(E)=in,$ $, FIRSTEXT(E)=\{\varnothing\}.$
$a?\|b?$	$I(E)=\{a,b\}, O(E)=\varnothing$ $HD(E)=in$ $FIRST(E)=\{\{a,b\}\}$	$, EN(E)=\varnothing, CO(E)=\varnothing,$ $, TL(E)=in,$ $, FIRSTEXT(E)=\{\{a,b\}\}.$
$a!\|b!$	$I(E)=\varnothing, O(E)=\{a,b\}$ $HD(E)=out,$ $FIRST(E)=\{\{a,b\}\}$	$, EN(E)=\varnothing, CO(E)=\varnothing,$ $,TL(E)=out,$ $, FIRSTEXT(E)=\{\{a,b\}\}.$
ϵ	$I(E)=\varnothing, O(E)=\varnothing$ $HD(E)=empty$ $FIRST(E)=\{\varnothing\}$	$, EN(E)=\varnothing, CO(E)=\varnothing,$ $, TL(E)=empty,$ $, FIRSTEXT(E)=\{\varnothing\}.$

TABLE 4.6.0. Values of attributes for $E \in$ <*marked syms*>.

(Recall that $E\|E=E$ for $a?\|a?$, etc.)

The evaluation rules corresponding to production rules $(b0)$, $(b1)$, $(c1)$, and $(c2)$ are given in Table 4.6.1. The evaluation rules for $(b2)$ and $(b3)$ consist of copying the values of I, O, EN, and CO; the evaluation rules for $(c3)$ (and $(c0)$) consist of copying the values of all eight attributes. The domain restrictions $D(i,j)$ for the array of commands $E(i,j: 0 \leq i,j < n)$ stand for $D(i,j) \equiv 0 \leq i,j < n \wedge E.i.j \neq \varnothing \wedge E.i.j \neq \epsilon$.

Rule	Production	Evaluation of attributes
$(b0)$	$\mu.tailf.0$	$I(\mu.tailf.0) = (\cup i,j{:}D(i,j){:}\ I(E.i.j))$, $O(\mu.tailf.0) = (\cup i,j{:}D(i,j){:}\ O(E.i.j))$, $EN(\mu tailf.0) = (\cup i,j{:}D(i,j){:}\ EN(E.i.j))$, $CO(\mu.tailf.0) = (\cup i,j{:}D(i,j){:}\ CO(E.i.j))$.
$(b1)$	$E0\|\|E1$	$I(E0\|\|E1) = I(E0) \cup I(E1)$, $O(E0\|\|E1) = O(E0) \cup O(E1)$, $EN(E0\|\|E1) = EN(E0) \cup EN(E1)$, $CO(E0\|\|E1) = CO(E0) \cup CO(E1)$.
$(c1)$	$E0;E1$	$I(E0;E1) = I(E0) \cup I(E1)$, $O(E0;E1) = O(E0) \cup O(E1)$, $EN(E0;E1) = EN(E0) \cup EN(E1)$, $CO(E0;E1) = CO(E0) \cup CO(E1)$, $HD(E0;E1) = HD(E0)$, $TL(E0;E1) = TL(E1)$, $FIRST(E0;E1) = FIRST(E0)$, if $FIRSTEXT(E0) \neq \{\varnothing\}$, $FIRSTEXT(E0;E1) = FIRSTEXT(E0)$ otherwise $FIRSTEXT(E0;E1) = FIRSTEXT(E1)$.
$(c2)$	$E0\|E1$	$I(E0\|E1) = I(E0) \cup I(E1)$, $O(E0\|E1) = O(E0) \cup O(E1)$, $EN(E0\|E1) = EN(E0) \cup EN(E1)$, $CO(E0\|E1) = CO(E0) \cup CO(E1)$, $HD(E0\|E1) = HD(E0)$, $TL(E0\|E1) = TL(E0)$ if $TL(E0) = TL(E1)$ $= mixed$ otherwise, $FIRST(E0\|E1) = FIRST(E0) \cup FIRST(E1)$, $FIRSTEXT(E0\|E1) = FIRSTEXT(E0)$ $\cup FIRSTEXT(E1)$.

TABLE 4.6.1. The evaluation rules of grammar $G4$.

4.7. Some DI Grammars

Let the set of all commands derivable with attribute grammar $G4$ be denoted by $\mathfrak{L}(G4)$. Grammar $G4$ is a DI grammar, i.e.

THEOREM 4.7.0. *$E \in \mathfrak{L}(G4) \Rightarrow E$ is DI.*

PROOF. See Appendix B. □

We conjecture that there exist regular DI components that cannot be expressed as a command $E \in \mathcal{L}(G4)$. For example, we did not succeed in expressing the RCEL component as a command from $\mathcal{L}(G4)$. (This component is a DI component as is shown in Example 4.9.1.)

Remark. Grammar $G4$ may be extended in such a way that more concurrent inputs, outputs, and internal symbols are allowed. The production rules for *<marked syms>* then become

$$
\begin{aligned}
\langle \textit{marked syms} \rangle ::= \ & \langle \textit{sym} \rangle ! \{ \| \langle \textit{sym} \rangle ! \} \\
[\,] \ & \langle \textit{sym} \rangle ? \{ \| \langle \textit{sym} \rangle ? \} \\
[\,] \ & ? \langle \textit{sym} \rangle ! \{ \| ? \langle \textit{sym} \rangle ! \} \\
[\,] \ & ! \langle \textit{sym} \rangle ? \{ \| ! \langle \textit{sym} \rangle ? \},
\end{aligned}
$$

where { } are meta symbols denoting a finite replication of the enclosed. Since in the remainder of this thesis no use is made of this extension, we have not included it in the grammar $G4$.
□

The attribute grammars $G4'$, $G3'$, $G2'$, and $G1'$ are defined similarly to grammar $G4$. Each grammar has its specific restrictions with respect to $G4$.

The restriction for grammar $G4'$ is the reduction of the production rules for *<marked syms>* to

$$\langle \textit{marked sym} \rangle ::= \langle \textit{sym} \rangle ? \ [\,] \ \langle \textit{sym} \rangle ! \ [\,] \ ! \langle \textit{sym} \rangle ?,$$

i.e. no parallel inputs or outputs are allowed, and there are no internal symbols of the environment.

Grammar $G3'$ is obtained from grammar $G4'$ by removing the alternative $!\langle \textit{sym} \rangle ?$ from the production rules for *<marked syms>* as well, i.e. $G3'$ has no internal symbols.

Grammar $G2'$ is obtained from grammar $G3'$ by strengthening the condition $ALTCOND(E0,E1)$ to $ALTCOND2(E0,E1)$, where

$$
\begin{aligned}
& ALTCOND2(E0,E1) \\
\equiv \ & ALTCOND(E0,E1) \wedge HD(E0)=in \wedge HD(E1)=in.
\end{aligned}
$$

A similar strengthening is applied in the conditions of *TAILCOND*.

Grammar $G1'$ is obtained from grammar $G4'$ by removal of the production rules for tail recursion ($b0$) and for the alternative construct ($c2$).

Obviously, we have $\mathcal{L}(Gi') \subseteq \mathcal{L}(G4)$ for $1 \leq i < 5$. Accordingly, any command derivable with one of the grammars $G4'$, $G3'$ $G2'$, or $G1'$ represents a DI component.

It is furthermore conjectured that $\mathcal{L}(Gi') \subseteq Ci$, for $1 \leq i < 4$.

4.8. DI Grammar *GCL′*

The grammar *GCL′* produces so-called *combinational commands*. Combinational commands represent components for which the outputs uniquely depend on the current inputs.

Remark. Components represented by combinational commands bear a resemblance to combinational circuits, as used in switching theory. There, these circuits are also called combinational logic and denoted by the acronym *CL*.
□

The production rules for the attribute grammar *GCL′* are given in Table 4.8.0.

<dicom> ::=		*<pccom>*	(*a*2)
<pccom> ::=		ε	
	[]	**pref**(*<sym>*?)	(*b*4)
	[]	**pref**(*<sym>*!)	(*b*5)
	[]	**pref**[*<pfcom>*]	(*b*6)
	[]	**pref**(*<parout>*; [*<pfcom>*])	(*b*7)
	[]	*<pccom>*‖*<pccom>*	(*b*8)
<pfcom> ::=		*<parin>*; *<parout>*	(*c*4)
	[]	*<pfcom>*|*<pfcom>*	(*c*5)
<parin> ::=		*<sym>*? [] *<sym>*? ‖ *<sym>*?	
<parout> ::=		*<sym>*! [] *<sym>*! ‖ *<sym>*!	

Table 4.8.0. The production rules for grammar *GCL′*.

The conditions for these production rules are listed in Table 4.8.1.

Production rule	Production	Condition
(*b*7)	**pref**(*E*0;[*E*1])	*ALFCOND*(*E*0,*E*1)
(*b*8)	*E*0‖*E*1	*ALFCOND*(*E*0,*E*1)
(*c*4)	*E*0;*E*1	*ALFCOND*(*E*0,*E*1)
(*c*5)	*E*0|*E*1	*ALTCOND*(*E*0,*E*1) ∧ *ALFCOND*(*E*0,*E*1)

Table 4.8.1. The conditions for grammar *GCL′*.

The evaluation rules for (*b*4), (*b*5), (*b*6), (*b*8), (*c*4), and (*c*5) are analogous to those of (*b*2), (*b*2), (*b*3), (*b*1), (*c*1) and (*c*2) respectively. The evaluation rules for production rule (*b*7) are analogous to the evaluation rules for (*b*1) where *E*0‖*E*1 is replaced by **pref**(*E*0;[*E*1]).

Any combinational command of type ϵ, **pref**(<*sym*>?), **pref**(<*sym*>!), **pref**[<*pfcom*>], or **pref**(<*parout*>;[<*pfcom*>]) is called a *semi-sequential command.* From the above, we infer that any combinational command is expressed as a weave of semi-sequential commands.

We have

THEOREM 4.8.0. *$E \in \mathcal{L}(GCL') \Rightarrow E$ is DI.*

PROOF (Sketch). We indicate that any command $E0 \in \mathcal{L}(GCL')$ can be rewritten into a semantically equivalent command $E1 \in \mathcal{L}(G4')$.

We observe that each production rule in GCL' also occurs in $G4'$ except for production rule $(b7)$. With this production rule semi-sequential commands of the form $\mathbf{pref}(E0;[E1])$ are produced. These commands can be rewritten into commands $\mu.tailf.0$, where

$$tailf.R.0 = \mathbf{pref}(E0;R1)$$

$$tailf.R.1 = \mathbf{pref}(E1;R.1).$$

Let each command of the form $\mathbf{pref}(E0;[E1])$ occurring in $E \in \mathcal{L}(GCL')$ be rewritten as above. The result of this rewriting is derivable with the attribute grammar $G4'$ (even $G2'$). Notice that the SEQCOND conditions are always satisfied for commands in $\mathcal{L}(GCL')$.

□

4.9. EXAMPLES

EXAMPLE 4.9.0. We give a few examples of combinational commands. The only conditions that have to be checked for combinational commands are the alphabet condition and the bar condition, which are easily verified. For the following commands of a 2-XOR, WIRE, and 2-CEL component we have

$$\mathbf{pref}[a?;c!\,|\,b?;c!] \in \mathcal{L}(GCL'),$$

$$\mathbf{pref}(b!;[a?;b!]) \in \mathcal{L}(GCL'), \text{ and}$$

$$\mathbf{pref}[a?;c!] \,\|\, \mathbf{pref}(c!;[b?;c!]) \in \mathcal{L}(GCL')$$

respectively. For the conjunction component of Section 2.3.0 we have

$$\mathbf{pref}[a0?\|b0?;c0! \mid a0?\|b1?;c0! \mid a1?\|b0?;c0! \mid a1?\|b1?;c1!] \in \mathcal{L}(GCL').$$

The bar condition for this command amounts to $\neg(A \subseteq B)$, for $A,B \in \{\{a0,b0\},\{a0,b1\},\{a1,b0\},\{a1,b1\}\}$ and $A \neq B$.

□

EXAMPLE 4.9.1. For the commands of the basic components given in Section 2.2 we observe

$$\mathbf{pref}[a?;c!] \parallel \mathbf{pref}[b?;c!] \in \mathcal{L}(G1'),$$

$$\mathbf{pref}[a?;b!] \parallel \mathbf{pref}[a?;c!] \in \mathcal{L}(G1'),$$

$$\mathbf{pref}[a?;b!;a?;c!] \in \mathcal{L}(G1'),$$

$$\mathbf{pref}[(a?|b?);c!] \in \mathcal{L}(G2'),$$

$$\mathbf{pref}[a?;p!] \parallel \mathbf{pref}[b?;q!] \parallel \mathbf{pref}[n?;(p!|q!)] \in \mathcal{L}(G3'),$$

and

$$\mathbf{pref}[a1?;p1!;a0?;p0!]$$

$$\parallel \mathbf{pref}[b1?;q1!;b0?;q0!]$$

$$\parallel \mathbf{pref}[p1!;a0?\,|\,q1!;b0?] \in \mathcal{L}(G3').$$

From this we conclude that the 2-CEL, 2-FORK, TOGGLE, 2-XOR, 2-SEQ, and 2-ARB component(s) are DI components.

For the RCEL component $\mathbf{pref}[E]$, where

$$E = (a?;d!)^2 \,|\, (b?;e!)^2 \,|\, (a?;d!\|c!)^2 \|(b?;e!\|c!)^2,$$

we observe

$$\mathbf{pref}\,E \in C2 \wedge \mathbf{hd}E \subseteq \mathbf{i}E \wedge \mathbf{tl}E \subseteq \mathbf{o}E \wedge E \text{ is prefix-free.}$$

As a special case of Theorem B.4 on tail recursion in Appendix B, we infer $\mathbf{pref}[E] \in C4$, i.e. also the RCEL component is a DI component.

Obviously, the WIRE, SINK, SOURCE, and EMPTY components are also DI components.

□

EXAMPLE 4.9.2. In Section 2.3.1 the sequence detector is specified by $\mu.tailf.0$, where

$$tailf.R.0 = \mathbf{pref}\,(a0?;n!;R.1 \mid a1?;n!;R.0)$$

$$tailf.R.1 = \mathbf{pref}\,(a0?;n!;R.1 \mid a1?;n!;R.2)$$

$$tailf.R.2 = \mathbf{pref}\,(a0?;n!;R.1 \mid a1?;n!;R.3)$$

$$tailf.R.3 = \mathbf{pref}\,(a0?;y!;R.1 \mid a1?;n!;R.0).$$

Command $\mu.tailf.0$ can be derived with the context-free grammar of $G2'$. We verify for this command the conditions of grammar $G2'$. For the alphabet condition we observe that for any symbol used all atomic commands in which this symbol occurs are of the same type. For the semicolon condition we observe that input marks and output marks alternate. For the bar condition we observe that each alternative of an alternative construct starts with input marks, and that the LL-1 conditions are satisfied, since $\{\{a0\}\} \cap \{\{a1\}\} = \varnothing$.

For the tail-function condition we observe for the array of commands of *tailf*,
- each row contains a non-empty command,
- no command is equal to ϵ, and
- all non-empty commands consist of external symbols only.

Consequently, the tail-function condition is satisfied. The non-projection condition is also satisfied, since $\mu.tailf.0$ contains no internal symbols. Accordingly, we conclude $\mu.tailf.0 \in \mathcal{L}(G2')$.
□

EXAMPLE 4.9.3. In Section 2.3.2 the token-ring interface is specified by

$$E = \textbf{pref}\,[a\,1?;p\,1!;a\,0?;p\,0!]$$
$$\| \,\textbf{pref}\,[b\,?;(q\,!\,|\,p\,1!;a\,0?;q\,1!)].$$

This command can be derived with the context-free grammar of $G3'$. For the conditions of $G3'$ we observe that the alphabet condition is satisfied. Furthermore, input and output marks alternate; the semicolon is satisfied as well. For the only alternative construct in E, i.e. $q\,!\,|\,p\,1!;a\,0?;q\,1!$, we observe that the alternatives start with output marks and that $\{\{q\}\} \cap \{\{p\,1\}\} = \varnothing$. Consequently, the bar condition is satisfied. The non-projection condition is also satisfied, and we conclude that $E \in \mathcal{L}(G3')$.
□

EXAMPLE 4.9.4. In Section 2.3.3 another token-ring interface is specified by

$$E = (\textbf{pref}[rb\,?;!b\,?;gb\,!;rw\,?;!w\,?;gw\,!]$$
$$\| \,\textbf{pref}[wtr\,?;(!tu\,?;wts\,!\,|\,!tb\,?;bts\,!)$$
$$|btr\,?;(!tu\,?|!tb\,?);bts\,!$$
$$]$$
$$\| \,\mu.tailf.0$$
$$)\!\upharpoonright,$$
$$\text{where } tailf.R.\,0 = \textbf{pref}(!tu\,?;R.\,0\,|\,!b\,?;!w\,?;R.\,1)$$
$$tailf.R.\,1 = \textbf{pref}(!tb\,?;R.\,0\,|\,!b\,?;!w\,?;R.\,1).$$

This command can be derived with the context-free grammar of $G4'$. We observe that the alphabet condition is satisfied and that input marks and output marks alternate. There are four alternative constructs to be considered for the bar condition, viz.,

$!tu\,?;wts\,!\,|\,!tb\,?;bts\,!$,

$!tu\,?\,|\,!tb\,?$,

$!tu\,?\,|\,!b\,?;!w\,?$, and

$!tb\,?\,|\,!b\,?;!w\,?$.

Each of the above alternatives starts with output marks. For the first two constructs we observe

$$\{\{tu\}\}\cap\{\{tb\}\}=\varnothing \ \wedge\ \{\{wts\}\}\cap\{\{bts\}\}=\varnothing$$

and

$$\{\{tu\}\}\cap\{\{tb\}\}=\varnothing \ \wedge$$
$$FIRSTEXT(!tu?)=\varnothing \ \wedge\ FIRSTEXT(!tb?)=\varnothing$$

respectively, i.e. both LL-1 conditions are satisfied. Consequently, the bar condition is satisfied for the first two constructs. A similar reasoning applies to the other two alternative constructs. For the tail-function condition we observe that all commands in the matrix of *tailf* differ from $\varnothing$ and ϵ and consist of internal symbols only. Accordingly, the tail-function condition is satisfied. Since all conditions are satisfied, we conclude $E\in\mathfrak{L}(G4')$.

□

EXAMPLE 4.9.5. We derive for component $count_3(a,b)$ of Example 1.3.1 a command satisfying grammar $G4'$. The component $count_3(a,b)$ can be specified by the command

$$(\mathbf{pref}[a\,;x]\,\|\,\mathbf{pref}[x\,;y]\,\|\,\mathbf{pref}[y\,;b])\restriction\{a,b\}.$$

We assign to the external symbol a, i.e. the increment, and to the external symbol b, i.e. the decrement, the direction of input. Symbols x and y are given the type of internal symbols of the component. We then obtain

$$E0 \ =\ (\mathbf{pref}[a?;!x?]\,\|\,\mathbf{pref}[!x?;!y?]\,\|\,\mathbf{pref}[!y?;b?])\restriction.$$

This command cannot be derived with grammar $G4'$: input and output marks do not alternate in the first and last sequential command. But these conditions are easily met, if we introduce two fresh symbols $p!$ and $q!$ and write

$$E1 \ =\ (\mathbf{pref}[a?;!x?;p!]\,\|\,\mathbf{pref}[!x?;!y?]\,\|\,\mathbf{pref}[!y?;q!;b?])\restriction.$$

This command can be derived with grammar $G4'$ (even with grammar $G1'$). Moreover, we have $\mathbf{t}E1\restriction\{a,b\}=\mathbf{t}E0$.

We remark that the position at which to insert $p!$ is not unique. We could also have changed the first sequential command into $\mathbf{pref}[p!;a?;!x?]$.

By the introduction of symbols $p!$ and $q!$ we have introduced a communication protocol between component and environment in order to ensure proper delay-insensitive operation. Communication protocols like the one introduced here, i.e. with $a?$ and $p!$ alternating and $q!$ and $b?$ alternating, can be called *handshake protocols*. Various handshake protocols exist; in the next examples more of them are given. By using a DI grammar one can quickly and conveniently discover such handshake protocols.

The introduction of a handshake protocol imposes behavioral restrictions on the environment and on the component. For protocol $E1$, for example, the environment has to take care of the alternations of a's and p's and of b's and

q's only. The component, however, has to ensure proper internal synchronization as well. Therefore, designing a communication protocol always requires a balancing of restrictions put on the component and restrictions put on the environment.

In Example 1.3.1 several commands, which all have the same structure, were given for component $count_n(a,b)$. With some calculus these commands can be rewritten into

$$(\mathbf{pref}[a\,;x\,] \,\|\, E \,\|\, \mathbf{pref}[y\,;b\,])\restriction\{a,b\},$$

where command E is expressed as a weave of sequential commands. We can apply to this command the same procedure as above to obtain a DI command. Thus, we may get many commands from $\mathcal{L}(G4')$ that have all the same trace structure.

□

EXAMPLE 4.9.6. The 3-place binary buffer of Example 1.3.2 is specified by

$$\begin{aligned}&(\mathbf{pref}[a\,0;x\,0 \mid a\,1;x\,1]\\&\|\,\mathbf{pref}[x\,0;y\,0 \mid x\,1;y\,1]\\&\|\,\mathbf{pref}[y\,0;b\,0 \mid y\,1;b\,1]\\&)\restriction\{a\,0,a\,1,b\,0,b\,1\}.\end{aligned}$$

We derive a DI command for this component in the same fashion as we did in the previous example. This time, we assign to the external symbols $a\,0$ and $a\,1$ the direction of inputs and to the external symbols $b\,0$ and $b\,1$ the direction of outputs (as opposed to the previous example where b was assigned the direction of input). Symbols $x\,0$, $x\,1$, $y\,0$ and $y\,1$ are internal symbols of the component. We obtain

$$\begin{aligned}&(\mathbf{pref}[a\,0?;!x\,0? \mid a\,1?;!x\,1?]\\&\|\,\mathbf{pref}[!x\,0?;!y\,0? \mid !x\,1?;!y\,1?]\\&\|\,\mathbf{pref}[!y\,0?;b\,0! \mid !y\,1?;b\,1!]\\&)\restriction.\end{aligned}$$

Again, the semicolon condition is not satisfied. To repair this, we introduce symbols $p\,!$ and $q\,?$ and write

$$\begin{aligned}&(\mathbf{pref}\,[a\,0?;!x\,0?;p\,! \mid a\,1?;!x\,1?;p\,!]\\&\|\,\mathbf{pref}[!x\,0?;!y\,0? \mid !x\,1?;!y\,1?]\\&\|\,\mathbf{pref}[q\,?;(!y\,0?;b\,0! \mid !y\,1?;b\,1!)]\\&)\restriction.\end{aligned}$$

This command can be derived with grammar $G4'$.

□

EXAMPLE 4.9.7. In this example we demonstrate how a DI command may be obtained from an undirected command by the so-called *four-phase handshake expansion.* This expansion was introduced by Alain Martin [25, 26]. The formalization given below was inspired by a note of Rob Hoogerwoord [16].

The construction of the expansion is described as follows. Let E be an undirected command. Rewrite E, if possible, into a form $E0{\restriction}$, where $E0$ is expressed as a weave of sequential commands. Each symbol $b \in \mathbf{ext}\,E0$ can be either *passive* or *active.* For each passive symbol $b \in \mathbf{ext}\,E0$ we introduce the four-phase handshake protocol

$$\mathbf{pref}[b0?;b1!;b2?;b3!],$$

which indicates that the environment initiates this protocol. For each active symbol $b \in \mathbf{ext}\,E0$ we introduce the four-phase handshake protocol

$$\mathbf{pref}[b1!;b2?;b3!;b0?],$$

which indicates that the component initiates the protocol for this symbol. The command E is expanded as follows. Replace each atomic command b in $E0$, with $b \in \mathbf{ext}E0$, by $b1!;b2?$ and replace each atomic command b in $E0$, with $b \in \mathbf{int}E0$, by $!b?$. The projection (on $\mathbf{ext}\,E0$) of the weave of the four-phase handshake protocols and the expansion of $E0$ forms the four-phase handshake expansion of E.

For example, for the command

$$(\mathbf{pref}[a;x] \,\|\, \mathbf{pref}[x;y] \,\|\, \mathbf{pref}[y;b]) \restriction \{a,b\}.$$

of $count_3(a,b)$ we obtain for passive a and b

$$\begin{array}{l}
(\,\mathbf{pref}[a0?;a1!;a2?;a3!] \\
\|\,\mathbf{pref}[b0?;b1!;b2?;b3!] \\
\|\,\mathbf{pref}[a1!;a2?;!x?] \\
\|\,\mathbf{pref}[!x?;!y?] \\
\|\,\mathbf{pref}[!y?;b1!;b2?] \\
){\restriction}.
\end{array}$$

Notice that for an expansion thus obtained, the projection on all symbols $b1$, or all symbols $b2$, with $b \in \mathbf{ext}\,E0$ yields, after an appropriate renaming, the original command.

The four-phase handshake expansion gives rise to a command that satisfies the alphabet condition, the semicolon condition and the non-projection condition. The other conditions do not always have to be satisfied, however. We observe that the expansion for $count_3(a,b)$ is derivable with grammar $G1'$.

An advantage of this handshake expansion is that the only restrictions put on the environments are the four-phase handshake protocols for the external symbols. These protocols are independent of each other. A disadvantage is that this expansion can introduce many synchronizations between outputs

which may yield more complex decompositions, as we shall see in the next chapters.
□

Chapter 5

A Decomposition Method I

Syntax-Directed Translation of Combinational Commands

5.0. Introduction

In this and the next chapter we present a method to decompose components expressed in $\mathcal{L}(G4') \cup \mathcal{L}(GCL')$ into a finite set of basic components. The decomposition method can be described as a syntax-directed translation of commands from $\mathcal{L}(G4') \cup \mathcal{L}(GCL')$ into commands of basic components. Moreover, we show that the decomposition can be carried out such that the result is linear in the length of the command, i.e. the total number of basic components in the decomposition of command E is proportional to the length of E.

In order to make the presentation of the decomposition method more digestible, we have split it into two chapters. In this chapter we discuss the decomposition of components expressed in $\mathcal{L}(GCL')$ into basic components, i.e. the decomposition of components represented by combinational commands into basic components. In the next chapter we discuss the decomposition of components expressed in $\mathcal{L}(G4') \setminus \mathcal{L}(GCL')$ into components expressed in $\mathcal{L}(GCL')$, i.e. the decomposition of components represented by non-combinational commands in $\mathcal{L}(G4')$ into components represented by combinational commands. (This division in the decomposition method exhibits a similarity with the division in the synthesis method of synchronous circuits usually applied in switching theory, i.e. a division into the synthesis of combinational circuits and sequential circuits.) The techniques applied in Chapter 5 illustrate in a simple way the techniques that are also applied in Chapter 6. The remainder of this section is devoted to a general introduction to the complete decomposition method.

The method consists of a hierarchy of decomposition steps, each of which is described by means of DI grammars. In order to describe the decomposition steps on the highest level in the hierarchical decomposition we use grammars $G4'$, $G3'$, $G2'$, and GCL' of the previous chapter. By means of these grammars we define the hierarchy of languages

$$\mathfrak{L}_0 \subseteq \mathfrak{L}_1 \subseteq \mathfrak{L}_2 \subseteq \mathfrak{L}_3 \subseteq \mathfrak{L}_4, \text{ where}$$

$$\mathfrak{L}_4 = \mathfrak{L}(G4') \cup \mathfrak{L}_3,$$

$$\mathfrak{L}_3 = \mathfrak{L}(G3') \cup \mathfrak{L}_2,$$

$$\mathfrak{L}_2 = \mathfrak{L}(G2') \cup \mathfrak{L}_1,$$

$$\mathfrak{L}_1 = \mathfrak{L}(GCL') \cup \mathfrak{L}_0, \text{ and}$$

$$\mathfrak{L}_0 = \{\text{all commands of basic components}\}.$$

The method can be divided into four steps. In step k, $0 \leqslant k < 4$, for each command $E.0 \in \mathfrak{L}_{4-k}$ a collection of commands $E.i$, $1 \leqslant i < n$, is constructed in such a way that the following properties hold.

- $E.0 \rightarrow (i: 1 \leqslant i < n: E.i)$. (5.0)
- $E.i \in \mathfrak{L}_{4-k-1}$, for all i, $1 \leqslant i < n$, and (5.1)
- The decomposition can be described (5.2)

  as a syntax-directed translation.

From the properties (5.0), (5.1), and (5.2) and the Substitution Theorem, we conclude that any component represented by a command in $\mathfrak{L}_4$ can be decomposed in a syntax-directed way into basic components expressed in $\mathfrak{L}_0$. Similar to the division of the decomposition of $\mathfrak{L}_4$ into $\mathfrak{L}_0$ into four steps, each of these decomposition steps is, in its turn, divided into a number of substeps. Thus, by stepwise refinement, we obtain a hierarchical decomposition method based on the Substitution Theorem.

The language $\mathfrak{L}_0$ is defined as the set of all commands of the basic components. In this thesis, we show that for the finite set of basic components we may take the set $\mathbb{B}0 = \mathbb{B} \cup \{\text{RCEL}\}$ or the set $\mathbb{B}1 = \mathbb{B} \cup \{\text{NCEL}\}$, where

$$\mathbb{B} = \{\text{2-FORK, 2-CEL, 2-XOR, TOGGLE, 2-SEQ,}$$
$$\text{WIRE, SINK, SOURCE, EMPTY}\}.$$

Each basis has its particular advantages and disadvantages. For example, for the basis $\mathbb{B}0$ we observe that every component in $\mathbb{B}0$ is a DI component (cf. Example 4.9.1). Accordingly, by Theorem 3.2.1.1, any decomposition of a DI component into the basis $\mathbb{B}0$ is a DI decomposition. The basis $\mathbb{B}1$, however, contains one component that is not a DI component, viz. the NCEL component. For this reason, the decomposition of a DI component into the basis $\mathbb{B}1$ does not have to be a DI decomposition. Although the decomposition into the basis $\mathbb{B}1$ is not DI, it is simpler than the decomposition into $\mathbb{B}0$ and has some practical advantages. Realizations of this decomposition with connection wires still operate properly if certain (physical) forks behave as so-called *isochronic forks.* In this thesis an isochronic fork is a fork for which the differences between the delays in the branches are less than the delay in a basic NCEL component.

The choice of basis $\mathbb{B}0$ or $\mathbb{B}1$ has to be taken in one of the last decomposition steps only, viz. in the decomposition of so-called *CAL components.* CAL components are DI components. The decomposition of CAL components into the basis $\mathbb{B}1$ is presented in Section 5.6.2. The decomposition of CAL components into the basis $\mathbb{B}0$, which is more complicated, is only briefly discussed in Section 5.6.3. This section may be skipped at first reading.

The decomposition of a component $E \in \mathcal{L}_4$ according to the method described in this and the next chapter can be carried out such that the result is linear in the length of E. We prove this by showing that each decomposition

$$E1.0 \rightarrow (i: 1 \leq i < n: E1.i)$$

in the hierarchy of decomposition steps satisfies the property

$$(+i: 1 \leq i < n: |E1.i\,|) = \Theta(|E1.0|). \qquad (5.3)$$

Here, $|E|$ denotes the length of command E and is defined as the number of atomic commands occurring in E. (For a command $\mu.tailf.0$ it is defined as the number of atomic commands in the tail function $tailf$ different from $\varnothing$.) In this thesis, the expression $|f.E| = \Theta(|E|)$ for a function f defined on commands from a particular language $\mathcal{L}$ signifies

$$(\mathbf{E}K: K>0: (\mathbf{A}E: E \in \mathcal{L}: |f.E| < K|E|)).$$

The linear complexity of the complete decomposition method can be derived from property (5.3) as follows. Let

$$E.0 \rightarrow (i: 1 \leq i < m: E.i)$$

denote the complete decomposition of DI component $E.0 \in \mathcal{L}_4$ into $\mathcal{L}_0$. Because the number of decomposition steps is bounded and each step satisfies property (5.3), we infer

$$(+i: 1 \leq i < m: |E.i\,|) = \Theta(|E.0|).$$

Since there exists an upper bound for the lengths of the commands from $\mathcal{L}_0$, we deduce that m is proportional to $|E.0|$.

The above properties of the decomposition method emphasize the importance of the task of the programmer. First, the programmer must express a component in the language $\mathcal{L}_4$. Second, if there are several programs possible for a component, he has to choose that program that suits his purposes best with respect to the decomposition of that program. For example, he may choose a short program to obtain a decomposition with a few basic elements, or he may choose a program whose decomposition according to the syntax of the program exhibits more parallelism, but which may be a larger program.

A more detailed overview of the hierarchy of all decomposition steps and languages can be described as follows. The decomposition steps from $\mathcal{L}_3$ to $\mathcal{L}_2$

and from $\mathcal{L}_1$ to $\mathcal{L}_0$ are divided into several substeps. Most of these substeps are also described by means of DI grammars which will be defined as the need arises. For example, we will define the grammars *GSEL*, *GCL0*, *GCL1*, and *GCAL*. Grammars *GCL*0 and *GCL*1 will be derived from grammar *GCL'*, grammar *GCAL* will be derived from grammar *GCL*1, and grammar *GSEL* will be derived from grammar *G*3'. The hierarchy among all languages is displayed in Figure 5.0.0.

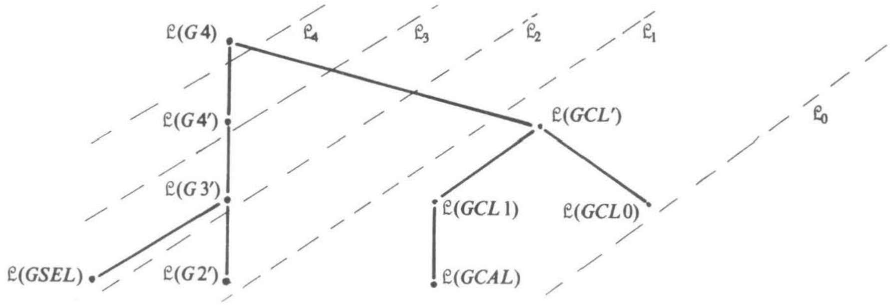


FIGURE 5.0.0. The hierarchy among the languages.

From Figure 5.0.0 we read, for example, that $\mathcal{L}(GCL1) \subseteq \mathcal{L}(GCL')$ and $\mathcal{L}(GCL') \subseteq \mathcal{L}(G4)$.

In order to give a concise overview of the hierarchy among all the decomposition steps we have displayed these steps symbolically in Table 5.0.0 together with the section in which these steps are presented.

Section	Decomposition step		
6.3	$\mathcal{L}(G4')$	$\rightarrow$	$\mathcal{L}(G3')$, $\mathcal{L}_0$
6.2.3	$\mathcal{L}(G3')$	$\rightarrow$	$\mathcal{L}(GSEL)$, $\mathcal{L}(GCL')$, $\mathcal{L}_0$
6.2.(4+5)	$\mathcal{L}(GSEL)$	$\rightarrow$	SEQ, $\mathcal{L}(G2')$, $\mathcal{L}(GCL')$
6.2.6	SEQ	$\rightarrow$	$\mathcal{L}_0$
6.1	$\mathcal{L}(G2')$	$\rightarrow$	$\mathcal{L}(GCL')$, $\mathcal{L}_0$
5.2	$\mathcal{L}(GCL')$	$\rightarrow$	$\mathcal{L}(GCL0)$, $\mathcal{L}(GCL1)$
5.3	$\mathcal{L}(GCL0)$	$\rightarrow$	XOR, CEL, FORK
5.4	XOR	$\rightarrow$	$\mathcal{L}_0$
5.4	CEL	$\rightarrow$	$\mathcal{L}_0$
5.4	FORK	$\rightarrow$	$\mathcal{L}_0$
5.5	$\mathcal{L}(GCL1)$	$\rightarrow$	$\mathcal{L}(GCAL)$, $\mathcal{L}_0$
5.6	$\mathcal{L}(GCAL)$	$\rightarrow$	$\mathcal{L}_0$

TABLE 5.0.0. The hierarchy of decomposition steps.

From this table we read, for example, that in Section 6.2 the decomposition step from $\mathcal{L}_3$ to $\mathcal{L}_2$, which is divided into three substeps, is discussed. First, components expressed in $\mathcal{L}(G3')$ are decomposed into components expressed in $\mathcal{L}(GSEL)$, $\mathcal{L}(GCL')$, and $\mathcal{L}_0$. Second, each component expressed in $\mathcal{L}(GSEL)$ is decomposed into SEQ components and components expressed in $\mathcal{L}(G2')$ and $\mathcal{L}(GCL')$. Finally, each SEQ component is decomposed into basic components.

Many of the above displayed decomposition steps follow a similar pattern. For example, if we have to decompose components E, where E is expressed as a weave of (semi-) sequential commands, then we first consider the decomposition of such components expressed by (semi-) sequential commands. Subsequently, we construct a decomposition for the weave of these commands by applying the Separation Theorem.

Since each decomposition step is precisely defined by means of the grammars, we can study the properties of each step in isolation. For each decomposition step of Table 5.0.0 we verify whether the decomposition can be carried out in a syntax-directed way and whether the decomposition is linear in the length of the command. For almost every step these properties are readily verified.

Most decomposition steps are introduced by means of an example from which the general decomposition procedure for that step easily follows. The discussions on the correctness of each decomposition are less formal than in Chapter 3. The simple decompositions are given by a schematic only. For the decomposition of CAL components into the basis $\mathbb{B}0$ we give a decomposition procedure which we conjecture to be correct.

Finally, we remark that the decomposition method presented in these two chapters is not the most efficient method. In these chapters, we are interested mainly in the existence of a syntax-directed (linear) decomposition method. Potential optimizations and decomposition techniques that can be applied to special commands are discussed in Chapter 7.

5.1. Decomposition of $\mathcal{L}_1$ into $\mathcal{L}_0$.

In the decomposition step from $\mathcal{L}_1$ to $\mathcal{L}_0$ each component $E \in \mathcal{L}(GCL')$ is decomposed into components expressed in $\mathcal{L}_0$. This decomposition step is divided into five substeps, and, in order to describe these steps, we first introduce the grammars $GCL0$, $GCL1$, and $GCAL$.

Grammar $GCL0$ is defined as grammar GCL' (see Section 4.8) except for one restriction: the production rule for *<parin>* is reduced to *<parin>*::=*<sym>*?, i.e. parallel inputs are not allowed. Grammar $GCL1$ is also defined as grammar GCL' except for two restrictions: the production rule for *<parout>* is reduced to *<parout>*::=*<sym>*!, i.e. parallel outputs are not allowed, and the other restriction is that all outputs differ. For example, we have

$$\begin{aligned}&\mathbf{pref}(e!\|g!;[a?;e!\|f! \mid b?;e!\|g! \mid c?;f!])\\ &\|\,\mathbf{pref}[c?;g! \mid d?;g!] \qquad\qquad \in \mathcal{L}(GCL0)\end{aligned}$$

and

$$\mathbf{pref}[a0?\|a1?;b0! \mid a0?\|a2?;b1! \mid a3?;b2! \mid a4?;b3!] \in \mathcal{L}(GCL1).$$

The grammar *GCAL* is defined analogously to grammar *GCL*1 except for two restrictions. The production rules for <*pccom*> and <*parin*> reduce to

<*pccom*> ::= **pref**[<*pfcom*>] and

<*parin*> ::= <*sym*>?||<*sym*>?,

where for the last production rule both inputs must differ. In words, any command for a CAL component is of the form **pref**[*E*], where each alternative in *E* is of the form <*sym*>? ||<*sym*>?;<*sym*>!. The command *E* satisfies the LL-1 conditions and all outputs in *E* differ. For example, we have

$$\mathbf{pref}[a0?\|b?;c0! \mid a1?\|b?;c1!] \in \mathcal{L}(GCAL) \text{ and}$$

$$\mathbf{pref}[a0?\|a1?;b0! \mid a0?\|a2?;b1! \mid a1?\|a2?;b2!] \in \mathcal{L}(GCAL).$$

A component expressed by a command in $\mathcal{L}(GCAL)$ is called a *CAL component,* which can be viewed as a 2-CEL component with alternatives.

The decomposition step is subdivided into five parts. First, we show how any component $E \in \mathcal{L}(GCL')$ is decomposed into a component $E0 \in \mathcal{L}(GCL0)$ and a component $E1 \in \mathcal{L}(GCL1)$. Second, we show how components $E \in \mathcal{L}(GCL0)$ can be decomposed into XOR, CEL, and FORK components. Third, we discuss the decomposition of XOR, CEL, and FORK components into the basis $\mathbb{B}$. Fourth, we present a method to decompose components $E \in \mathcal{L}(GCL1)$ into CAL components and components expressed in $\mathcal{L}_0$. Finally, we discuss the decomposition of CAL components into $\mathcal{L}_0$.

5.2. Decomposition of $\mathcal{L}(GCL')$

In this section, we show that for any command $E0 \in \mathcal{L}(GCL')$ there exist commands $E1 \in \mathcal{L}(GCL1)$ and $E2 \in \mathcal{L}(GCL0)$ such that $E0 \to E1, E2$. The commands $E1$ and $E2$ are constructed from the syntax of $E0$. Moreover, we have $|E1| + |E2| = \Theta(|E0|)$. Before we explain the decomposition we briefly recall that any command $E0 \in \mathcal{L}(GCL')$ is expressed as a weave of semi-sequential commands of the form ϵ, **pref**(*a*?), **pref**(*a*!), **pref**[*E*], or **pref**(<*parout*>;[*E*]). Command *E* is an alternative construct, where each alternative is of the form <*parin*>;<*parout*> (see Section 4.8).

First, we consider an example. Let *E*. 0.0 and *E*. 1.0 be defined by

$$E.0.0 = \mathbf{pref}[a0?\|a1?;b0!\|b1! \mid a0?\|a2?;b0!\|b2! \mid a3?;b1!]$$

$$E.1.0 = \mathbf{pref}(b3!;[a4?;b0!\|b3! \mid a0?;b4!]).$$

We observe that $E.0.0\|E.1.0 \in \mathcal{L}(GCL')$. Let *E*. 0.1, *E*. 0.2, *E*. 1.1, and *E*. 1.2 be defined by

$$E.0.1 = \mathbf{pref}[a0?\|a1?;q.0.0! \mid a0?\|a2?;q.0.1! \mid a3?;q.0.2!]\,,$$

$$E.0.2 = \mathbf{pref}[q.0.0?;b0!\|b1! \mid q.0.1?;b0!\|b2! \mid q.0.2?;b1!]\,,$$

$$E.1.1 = \mathbf{pref}[a4?;q.1.0! \mid a0?;q.1.1!]\,, \text{ and}$$

$$E.1.2 = \mathbf{pref}(b3!;[q.1.0?;b0!\|b3! \mid q.1.1?;b4!]).$$

By definition of decomposition, we derive

$$E.0.0 \to E.0.1\,, E.0.2 \text{ and}$$

$$E.1.0 \to E.1.1\,, E.1.2\,.$$

In order to apply the Separation Theorem we check conditions (3.7) and (3.8) for the above decompositions. We infer that the internal symbols of the decompositions are row-wise disjoint and that the outputs are column-wise disjoint. Consequently, application of the Separation Theorem yields

$$E.0.0\|E.1.0 \to E.0.1\|E.1.1\,,\ E.0.2\|E.1.2\,.$$

Moreover, we observe that in $E.0.1 \| E.1.1$ parallel outputs do not occur and all outputs differ, i.e. $E.0.1\|E.1.1 \in \mathcal{L}(GCL1)$. In $E.0.2 \| E.1.2$ parallel inputs do not occur, and consequently $E.0.2\|E.1.2 \in \mathcal{L}(GCL0)$.

The above decomposition procedure can be applied to any combinational command $E0 \in \mathcal{L}(GCL')$. By definition of grammar GCL', command $E0$ is expressed as a weave $(\|i: 0 \leq i < n: E.i.0)$ of semi-sequential commands $E.i.0 \in \mathcal{L}(GCL')$. Let command $E.i.0$ have $m(i)$ alternatives, $m(i) \geq 0$. We introduce the internal symbol $q.i.j$ for the semicolon in alternative j, $0 \leq j < m(i)$, of semi-sequential command $E.i.0$, $0 \leq i < n$. Subsequently, we split command $E.i.0$ into $E.i.1$ and $E.i.2$ such that $E.i.0 \to E.i.1, E.i.2$ holds, similarly to the example above. For the semi-sequential commands ϵ and $\mathbf{pref}(a!)$ and $\mathbf{pref}(a?)$ we take $\epsilon \to \epsilon, \epsilon$ and $\mathbf{pref}(a!) \to \epsilon, \mathbf{pref}(a!)$ and $\mathbf{pref}(a?) \to \mathbf{pref}(a?), \epsilon$ respectively. For the decompositions $E.i.0 \to E.i.1, E.i.2$, $0 \leq i < n$, it follows that the internal symbols are row-wise disjoint and that the outputs are column-wise disjoint. Consequently, by the Separation Theorem, we derive $E0 \to E1, E2$, where

$$E0 = (\|i: 0 \leq i < n: E.i.0),$$

$$E1 = (\|i: 0 \leq i < n: E.i.1), \text{ and}$$

$$E2 = (\|i: 0 \leq i < n: E.i.2).$$

Moreover, from the construction of $E1$ and $E2$ follows $E1 \in \mathcal{L}(GCL1)$, $E2 \in \mathcal{L}(GCL0)$, and $|E1| + |E2| = \mathcal{O}(|E0|)$.

5.3. DECOMPOSITION OF $\mathfrak{L}(GCL0)$

5.3.0. Decomposition of semi-sequential commands

Consider the component E, with

$$E = \mathbf{pref}(e!\|g!;[a?;e!\|f! \mid b?;e!\|g! \mid c?;f!]).$$

We observe that E is a semi-sequential command and $E \in \mathfrak{L}(GCL0)$. By definition of decomposition, component E can be decomposed into the XOR components

$$XOR0 = \mathbf{pref}(e!;[a?;e! \mid b?;e!]),$$

$$XOR1 = \mathbf{pref}[a?;f! \mid c?;f!], \text{ and}$$

$$XOR2 = \mathbf{pref}(g!;[b?;g!]).$$

Notice that $XOR0 = E \restriction \mathbf{a}XOR0$, and that similar properties hold for XOR1 and XOR2. The decomposition is depicted in Figure 5.3.0.

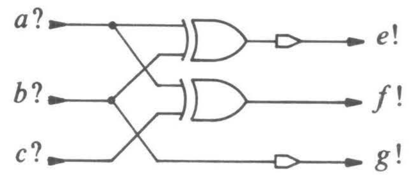


FIGURE 5.3.0. Decomposition of semi-sequential command $E \in \mathfrak{L}(GCL0)$.

In general, any semi-sequential command $E \in \mathfrak{L}(GCL0)$ can be decomposed in the same way. The procedure for this decomposition is described as follows. Each semi-sequential command $E \in \mathfrak{L}(GCL0)$ is of the form ϵ, $\mathbf{pref}(a?)$, $\mathbf{pref}(a!)$, $\mathbf{pref}[E1]$, or $\mathbf{pref}(E0;[E1])$. Component ϵ is the EMPTY component, and the components specified by $\mathbf{pref}(a?)$ or $\mathbf{pref}(a!)$ are a SINK or an active SOURCE component respectively. A component specified by $\mathbf{pref}(E0;[E1])$ or $\mathbf{pref}[E1]$ can be decomposed into XOR and active SOURCE components as follows. We take for each output in $E1$ a k-XOR component, where k equals the number of alternatives in which this output occurs. The input that occurs in each such alternative is connected to this XOR component. (By definition of $GCL0$ there is exactly one input in each alternative.) If an input is connected to more than one XOR component, then a FORK component is used to duplicate this input. If the output occurs in $E0$ as well, then the XOR component initially starts with producing an output. For each output that occurs in $E0$ but not in $E1$ we take an active SOURCE component.

The above described procedure yields a syntax-directed decomposition of semi-sequential commands $E \in \mathfrak{L}(GCL0)$ that is linear in the length of the command E.

5.3.1. The general decomposition

The general decomposition of a component $E \in \mathcal{L}(GCL0)$, where E is a weave of semi-sequential commands, is obtained by application of the Separation Theorem. We consider an example first.

Let $E.0.0$ and $E.1.0$ be defined by

$$E.0.0 = \mathbf{pref}(e! \| g!; [a?;e! \| f! \mid b?;e! \| g! \mid c?;f!]), \text{ and}$$

$$E.1.0 = \mathbf{pref}[c?;g! \mid d?;g!].$$

We observe that $E.0.0$ and $E.1.0$ are semi-sequential commands from $\mathcal{L}(GCL0)$ and $E.0.0 \| E.1.0 \in \mathcal{L}(GCL0)$. Let $E.i.j$, with $0 \leqslant i < 2$ and $1 \leqslant j < 5$, be defined by

$$E.0.1 = \mathbf{pref}(e0! \| g0!; [a?;e0! \| f0! \mid b?;e0! \| g0! \mid c?;f0!]),$$

$$E.1.1 = \mathbf{pref}[c?;g1! \mid d?;g1!],$$

$$E.0.2 = \mathbf{pref}[e0?;e!],$$

$$E.1.2 = \epsilon,$$

$$E.0.3 = \mathbf{pref}[f0?;f!],$$

$$E.1.3 = \epsilon,$$

$$E.0.4 = \mathbf{pref}[g0?;g!], \text{ and}$$

$$E.1.4 = \mathbf{pref}[g1?;g!].$$

Components $E.0.1$ and $E.1.1$ are similar to $E.0.0$ and $E.1.0$. Components $E.0.2$, $E.0.3$, $E.0.4$, and $E.1.4$ are WIRE components. Since $E.0.0$ and $E.1.0$ are DI commands, we have (see also Definition 3.2.1.0)

$$E.0.0 \rightarrow E.0.1,\ E.0.2,\ E.0.3,\ E.0.4$$

$$E.1.0 \rightarrow E.1.1,\ E.1.2,\ E.1.3,\ E.1.4.$$

In order to apply the Separation Theorem, we check conditions (3.7) and (3.8) for the above decompositions. We observe that the internal symbols of these decompositions are row-wise disjoint and that the outputs are column-wise disjoint. Consequently,

$$E.0.0 \| E.1.0 \rightarrow E.0.1 \| E.1.1,\ E.0.2 \| E.1.2,\ E.0.3 \| E.1.3,\ E.0.4 \| E.1.4.$$

Since we also have $\mathbf{o}(E.0.1) \cap \mathbf{o}(E.1.1) = \varnothing$, we can apply Corollary 3.1.3.3 yielding $E.0.1 \| E.1.1 \rightarrow E.0.1, E.1.1$. From the preceding subsection, we know how to decompose the semi-sequential commands $E.0.1$ and $E.1.1$. Components $E.0.2 \| E.1.2$, $E.0.3 \| E.1.3$, and $E.0.4 \| E.1.4$ are CEL components of which $E.0.3 \| E.1.3$ and $E.0.2 \| E.1.2$ reduce to WIRE components. The complete decomposition of $E.0.0 \| E.1.0$ is depicted in Figure 5.3.1.

The above procedure can be applied to any component $E0 \in \mathcal{L}(GCL0)$. By definition of grammar $GCL0$, the command $E0$ is expressed as a weave

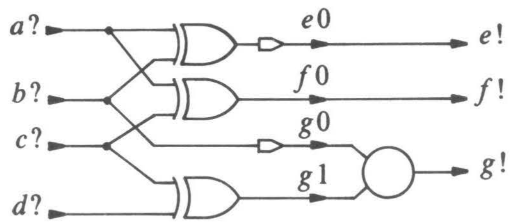


FIGURE 5.3.1. Decomposition of $E.0.0 \| E.1.0$.

$(\|i: 0 \leqslant i < n: E.i.0)$ of semi-sequential commands $E.i.0 \in \mathcal{L}(GCL0)$, $0 \leqslant i < n$. Similarly to the example above, component $E0$ can be decomposed into a collection $(i: 0 \leqslant i < n: E.i.1)$ of components expressed as semi-sequential commands $E.i.1 \in \mathcal{L}(GCL0)$ and a collection $(i: 0 \leqslant i < m: \mathrm{CEL}.i)$ of CEL components, where m equals the number of outputs in $E0$. For the commands $E.i.1$, $0 \leqslant i < n$, and $\mathrm{CEL}.i$, $0 \leqslant i < m$, we derive

$$\begin{aligned} & (+i: 0 \leqslant i < n: |E.i.1| = \Theta(|E0|) \\ & \wedge (+i: 0 \leqslant i < m: |\mathrm{CEL}.i|) = \Theta(|E0|) \\ \Rightarrow & \ \{\text{calc.}\} \\ & (+i: 0 \leqslant i < n: |E.i.1|) + (+i: 0 \leqslant i < m: |\mathrm{CEL}.i|) = \Theta(|E0|). \end{aligned}$$

Observe that this decomposition can also be described as a syntax-directed translation.

From Sections 5.3.0 and 5.3.1 we conclude that components $E \in \mathcal{L}(GCL0)$ can be decomposed into XOR, CEL, FORK, SINK, SOURCE and EMPTY components. The SINK, SOURCE, and EMPTY components are basic components. The decomposition of XOR, CEL, and FORK components into basic components is discussed in the next section.

5.4. Decomposition of XOR, CEL, and FORK components

There are several ways to decompose a k-XOR component, $k > 1$, into 2-XOR components. In Example 3.1.1.2 we decomposed a 3-XOR component into two 2-XOR components. The 4-XOR component E, with

$$E = \mathbf{pref}[a0?;b! \mid a1?;b! \mid a2?;b! \mid a3?;b!],$$

can be decomposed in two ways into 2-XOR components as depicted in Figure 5.4.0.

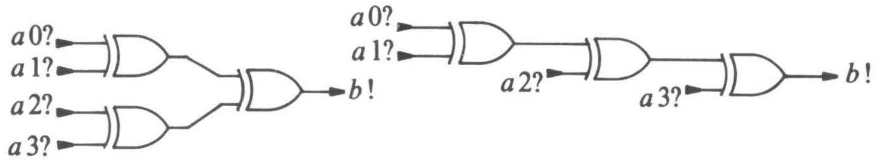


FIGURE 5.4.0. Two decompositions of 4-XOR component E.

In general, any k-XOR component, $k>1$, can be decomposed into $(k-1)$ 2-XOR components. These decompositions can be described as syntax-directed translations.

A k-CEL component, $k>1$, can be decomposed into 2-CEL components in several ways as well. In Example 3.1.1.3 a 3-CEL component is decomposed into two 2-CEL components. In Figure 5.4.1 two ways are shown to decompose the 4-CEL component E, with

$$E \;=\; \textbf{pref}[b\,!;a\,0?] \parallel \textbf{pref}[a\,1?;b\,!] \parallel \textbf{pref}[b\,!;a\,2?] \parallel \textbf{pref}[a\,3?;b\,!].$$

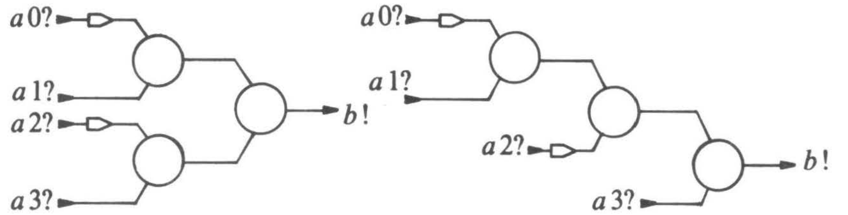


FIGURE 5.4.1. Two decompositions of 4-CEL component E.

In general, any k-CEL component, $k>1$, can be decomposed into $(k-1)$ 2-CEL components. These decompositions can be described as syntax-directed translations as well.

For the k-FORK components, $k>1$, a similar reasoning holds as for the k-XOR and k-CEL components.

5.5. DECOMPOSITION OF $\mathcal{L}(GCL\,1)$

Any component expressed in $\mathcal{L}(GCL\,1)$ can be decomposed into CAL, WIRE, SINK, SOURCE and EMPTY components. Before we explain this decomposition, we briefly recall the definition of grammar $GCL\,1$. Any command $E0 \in \mathcal{L}(GCL\,1)$ is expressed as a weave of semi-sequential commands of the form ϵ, **pref**$(a?)$, **pref**$(a!)$, **pref**$[E]$, or **pref**$(a!;[E])$. The command E is an alternative construct, where the alternatives are of the form $<sym>? \parallel <sym>?;<sym>!$ or $<sym>?;<sym>!$. All outputs in $E0$ differ.

First we consider the decomposition of components E, where E is a semi-sequential command from $\mathcal{L}(GCL1)$. Component ϵ is the EMPTY component, and components **pref**$(a?)$ and **pref**$(a!)$ are a SINK and an active SOURCE component respectively. For a component specified by **pref**$(a!;[E])$ we have, by definition of grammar $GCL1$, that all outputs differ and that E begins with inputs. Consequently, **pref**$(a!;[E]) \rightarrow$ **pref**$(a!)$, **pref**$[E]$. Finally, we show that any component **pref**$[E] \in \mathcal{L}(GCL1)$ can be decomposed into a CAL component and a collection of WIRE components.

An example of a command **pref**$[E] \in \mathcal{L}(CLC1)$ is given by

$$E0 = \textbf{pref}[a0?\|a1?;b0! \mid a0?\|a2?;b1! \mid a3?;b2! \mid a4?;b3!].$$

We observe

$$E0$$

$$\Rightarrow \{\text{def. of decomposition}\}$$

$$\textbf{pref}[a0?\|a1?;b0! \mid a0?\|a2?;b1!],\ \textbf{pref}[a3?;b2!],\ \textbf{pref}[a4?;b3!].$$

Consequently, component $E0$ can be decomposed into a CAL component and two WIRE components.

In general, any component **pref**$[E] \in \mathcal{L}(GCL1)$ can be decomposed similarly. The command **pref**$[E]$ can be rewritten as **pref**$[E1 \mid E2]$, where $E1$ contains all alternatives with two parallel inputs and $E2$ contains all alternatives with one input only. Since **pref**$[E1 \mid E2] \in \mathcal{L}(GCL1)$, we infer from the LL-1 conditions that $\mathbf{i}E1 \cap \mathbf{i}E2 = \varnothing$ and that all inputs in $E2$ differ. Moreover, by definition of grammar $GCL1$, all outputs in $E1|E2$ differ. From these observations it follows that **pref**$[E1 \mid E2]$ can be decomposed into **pref**$[E1]$, which is a CAL component, and a collection of WIRE components, one for each alternative in $E2$. If E does not contain alternatives with one input only, then **pref**$[E]$ is already a CAL component, and if E does not contain alternatives with parallel inputs, then **pref**$[E]$ can be decomposed into WIRE components.

The decomposition of any component $E0 \in \mathcal{L}(GCL1)$ is obtained by application of Corollary 3.1.3.3. Any command $E0 \in \mathcal{L}(GCL1)$ is expressed as a weave $(\|i: 0 \leq i <: E.i)$ of semi-sequential commands $E.i \in \mathcal{L}(GCL1)$. By definition of $GCL1$, we have $\mathbf{o}(E.i) \cap \mathbf{o}(E.j) = \varnothing$ for $i \neq j$. Accordingly, we observe

$$(\|i: 0 \leq i < n: E.i)$$

$$\rightarrow \{\text{Cor. 3.1.3.3},\ \mathbf{o}(E.i) \cap \mathbf{o}(E.j) = \varnothing \text{ for } i \neq j\}$$

$$(i: 0 \leq i < n: E.i).$$

From the above, we know how to decompose the semi-sequential commands $E.i \in \mathcal{L}(GCL1)$. Consequently, by the Substitution Theorem, we infer that any component $E0 \in \mathcal{L}(GCL1)$ can be decomposed into a collection $(i: 0 \leq i < m: E1.i)$ of CAL, WIRE, SOURCE, SINK, and EMPTY components. Notice that $(+i: 0 \leq i < m: |E1.i|) = \mathcal{O}(|E0|)$ and that the decomposition into these components can be described as a syntax-directed translation. The WIRE, SOURCE, SINK, and EMPTY component are basic components.

The decomposition of CAL components into basic components is discussed in the next section.

5.6. Decomposition of $\mathcal{L}(GCAL)$

5.6.0. Introduction

The decomposition of components expressed in $\mathcal{L}(GCAL)$, i.e. the so-called CAL components, into $\mathcal{L}_0$ is divided into two steps. First, we present a method for decomposing CAL components into their so-called 4-cycle version and their *2-to-4 cycle converter.* A 2-to-4 cycle converter is a connection of components from the basis **B**. Subsequently, we show how the 4-cycle version of a CAL component can be decomposed into the basis **B**1. Finally, we briefly discuss the existence of a method that decomposes the 4-cycle version of a CAL component into the basis **B**0.

5.6.1 Conversion to 4-cycle signaling

The decomposition of CAL component E, where

$$E = \mathbf{pref}[a0?\|b?;c0! \mid a1?\|b?;c1!],$$

into its 4-cycle version $E4$, where

$$E4 = \mathbf{pref}[a0'?\|b'?;c0'!;a0'?\|b'?;c0'! \\ |a1'?\|b'?;c1'!;a1'?\|b'?;c1'! \\ ],$$

and its 2-to-4 cycle converter is depicted in Figure 5.6.0.

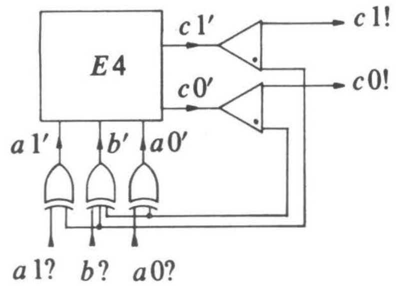


FIGURE 5.6.0. The 2-to-4 cycle conversion for E.

(Notice that $E4$ is also a DI command.) The connection of XOR and

TOGGLE components constitutes the 2-to-4 cycle converter for E.

In general any CAL component is converted into its 4-cycle version similarly. The 4-cycle CAL component is also a DI component. In each 4-cycle communication the 2-to-4 cycle converter feeds back the first output of the 4-cycle component to reset the inputs of the corresponding alternative to zero. In other words, the feedback initiates the return-to-zero phase. The second output of the 4-cycle component produces the output of the 2-cycle component.

A 2-to-4 cycle converter for a CAL component consists of k TOGGLE, k 2-FORK, and $2k$ 2-XOR components, where k is the number of alternatives in the command for the CAL component. The conversion to 4-cycle signaling can be described as a syntax-directed translation.

5.6.2. Decomposition of 4-cycle CAL components into $\mathbf{B}1$

We proceed with the decomposition of the 4-cycle CAL component $E4$ as specified in the previous subsection. Let the NCEL components $E1$ and $E2$ be defined by

$$E1 = \mathbf{pref}[a0'?\|b'?\,;c0'!\,;a0'?\|b'?\,;c0'! \mid b'?\,;b'?]$$

$$E2 = \mathbf{pref}[a1'?\|b'?\,;c1'!\,;a1'?\|b'?\,;c1'! \mid b'?\,;b'?].$$

Notice that $E4{\restriction}\mathbf{a}E1=E1$ and $E4{\restriction}\mathbf{a}E2=E2$. By definition of decomposition, we derive that $E4 \rightarrow E1,E2$. The decomposition is shown in Figure 5.6.1 (, where an isochronic fork is used for reasons explained below).

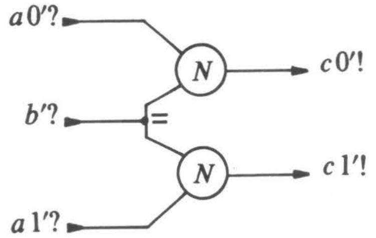


FIGURE 5.6.1. Decomposition of $E4$ into $\mathbf{B}1$.

Components $E1$ and $E2$ are not DI components, however. For this reason, the decomposition is not a DI decomposition. In order to ensure proper operation in a realization with connection wires, delay assumptions must be met. The delay assumptions that we make for this decomposition are the following: the differences between the delays in the branches of a (physical) fork are less than the delay in an NCEL component. In this thesis, we call a fork that meets this assumption an *isochronic fork.* A FORK component that must be realized by an isochronic fork is denoted in a schematic by an equality sign next to the fat dot denoting the FORK component. Notice that isochronic forks guarantee that all inputs of an NCEL component have returned to zero before a next 4-cycle communication begins.

In general, any 4-cycle CAL component **pref**$[E]$ can be decomposed into k NCEL components, where k is the number of alternatives in E. The realization of this decomposition with connection wires operates properly if isochronic forks are used to connect common inputs of NCEL components. Such a realization contains at most k isochronic forks. Notice that this general decomposition of 4-cycle CAL components into $\mathbb{B}1$ can be described as a syntax-directed translation.

5.6.3. Decomposition of 4-cycle CAL components into $\mathbb{B}0$

(This section may be skipped at first reading.) The decomposition of 4-cycle CAL components into a finite basis of DI components is one of the most difficult parts of the complete decomposition method. In this section we describe a method to decompose 4-cycle CAL components into the basis $\mathbb{B}0$. We conjecture that this decomposition is correct. The method is described merely to indicate the existence of a linear DI decomposition of CAL components. We first give a few examples of decompositions and then describe the general procedure.

Decompositions of components $E0$ and $E1$, where

$$E0 = \mathbf{pref}[(a0?\|b?;c0!)^2 \mid (a1?\|b?;c1!)^2],$$

$$E1 = \mathbf{pref}[(a0?\|a1?;b0!)^2 \mid (a0?\|a2?;b1!)^2 \mid (a1?\|a2?;b2!)^2],$$

are given in Figure 5.6.2 and 5.6.3 respectively.

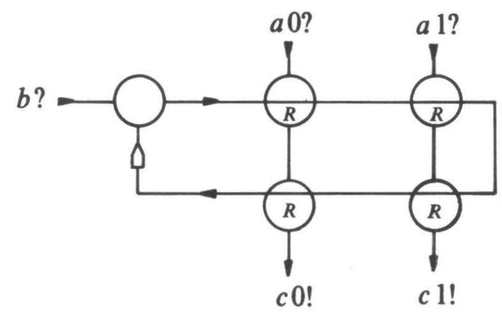


FIGURE 5.6.2. Decomposition of $E0$ into $\mathbb{B}0$.

The general decomposition procedure for a 4-cycle CAL component E is as follows. For each alternative in E we take a column of at most three (R)CEL components according to the following rules:

- if both inputs of the alternative do not occur in other alternatives, then we take one 2-CEL component;

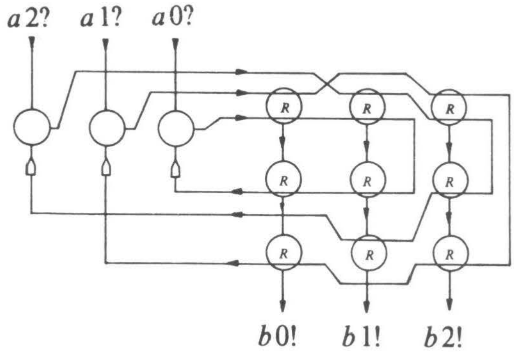


FIGURE 5.6.3. Decomposition of $E1$ into $\mathbb{B}0$.

- if one input only occurs in another alternative, then we take two RCEL components;
- if both inputs occur in other alternatives, then we take three RCEL components.

Per column, the output of the RCEL component in the i-th row is connected to an input of the RCEL component in the $i+1$st row, $1 \leqslant i < 3$, if present. The output of the last (R)CEL component in the column is the output corresponding to the output in the alternative. Each input of E is connected to the decomposition according to the following rules.

- if the input occurs in one alternative only it is connected to an input of the (R)CEL component in the first row and the column corresponding to that alternative.
- if the input occurs in more than one alternative it is connected to a so-called *interference-free loop,* as depicted in Figure 5.6.4.

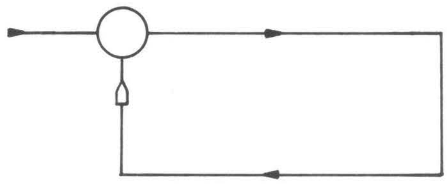


FIGURE 5.6.4. An interference-free loop.

This loop is first fed through the RCEL components in the first row and the columns that correspond to the alternatives in which this input occurs. Subsequently, the loop is fed back through a remaining RCEL component in each of the same columns.

This decomposition procedure yields the decompositions as given in Figures 5.6.2 and 5.6.3.

We make two remarks with respect to the behavior of the decompositions. First, in any interference-free loop transmission interference does not occur, i.e. for any behavior of the decomposition at most one transition is propagating along the loop. Second, when in any 4-cycle communication the second output is produced, all inputs of the RCEL components in the first row are still zero or have returned to zero. Consequently, neither of the RCEL components in the first row will produce a next output until both its inputs have changed again.

The above described procedure yields for any 4-cycle CAL component E a decomposition with $\Theta(|E|)$ components from $\mathbf{B}0$. Also this procedure can be described as a syntax-directed translation.

5.7. Schematics of decompositions

Decompositions obtained by the methods described in previous sections can be depicted in schematics that exhibit a regular structure. As an example we consider the decomposition of component E specified by

$$E = \mathbf{pref}[a0?\|a1?;b0!\|b1! \mid a0?\|a2?;b0!\|b2! \mid a3?;b1!]$$
$$\| \mathbf{pref}(b3!;[a4?;b0!\|b3! \mid a0?;b4!]).$$

From the preceding sections, it follows that the complete decomposition of this component into the basis $\mathbf{B}1$ can be depicted as in Figure 5.7.0.

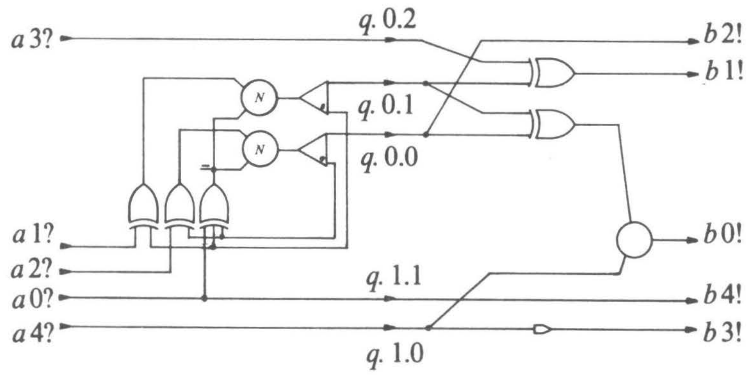


FIGURE 5.7.0. Decomposition of E.

The layout of this schematic can be rearranged in such a way that it exhibits a more regular pattern. This is done in Figure 5.7.1. The XOR components are shifted into one plane, the so-called *XOR-plane;* the NCEL (or CEL) and TOGGLE components are shifted into one plane, the so-called *CT plane*; and the remaining CEL components are shifted into one plane, the so-called *CEL plane.* FORK components are depicted in the CT plane and the XOR plane, where the FORK components in the CT plane must be realized by isochronic forks.

The decomposition of any component $E \in \mathfrak{L}(GCL')$ can be depicted similarly to Figure 5.7.1.

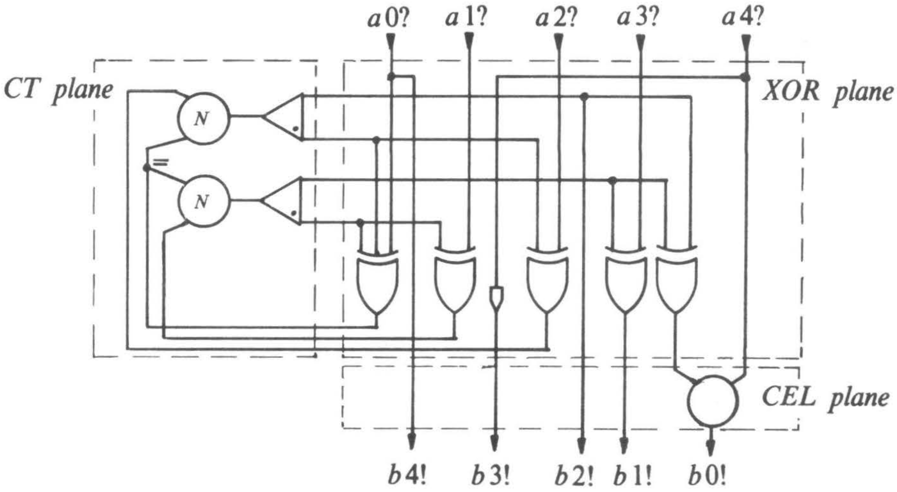


FIGURE 5.7.1. A regular schematic of decomposition of *E*.

Chapter 6

A Decomposition Method II

Syntax-Directed Translation of Non-Combinational Commands

6.0. Introduction

In this chapter we present the decomposition from $\mathcal{L}(G4') \setminus \mathcal{L}(GCL')$ into $\mathcal{L}(GCL')$, i.e. the decomposition of components represented by non-combinational commands in $\mathcal{L}(G4')$ into components represented by combinational commands. This decomposition is divided into three steps, viz. the decomposition from $\mathcal{L}_4$ into $\mathcal{L}_3$, the decomposition from $\mathcal{L}_3$ into $\mathcal{L}_2$, and the decomposition from $\mathcal{L}_2$ into $\mathcal{L}_1$. For the definition of the languages $\mathcal{L}_4$, $\mathcal{L}_3$, $\mathcal{L}_2$, and $\mathcal{L}_1$ and a general introduction to the decomposition presented in this chapter we refer to Section 5.0.

Each decomposition step is discussed similarly to the decompositions presented in the previous chapter. We describe each step by means of some grammars and study the properties of this step with respect to the syntax-directedness and linearity of the decomposition in the length of the command. Mostly these properties are readily verified. There is one step, however, that renders some difficulties in maintaining the linearity of the decomposition. This is the decomposition of components expressed in $\mathcal{L}(GSEL)$. In Section 6.2.4 a non-linear decomposition is discussed, and in Section 6.2.5 we show that a linear decomposition is also possible —though more difficult than the non-linear decomposition. The latter section may be skipped at first reading.

One could say that the decomposition steps presented in this chapter differ from the one presented in the previous chapter in the sense that here an encoding of state information is involved in the decomposition of a component. For the decomposition steps from $\mathcal{L}_2$ to $\mathcal{L}_1$ and from $\mathcal{L}_3$ to $\mathcal{L}_2$ we apply a so-called *state assignment* to each sequential command which is part of the complete command representing the component. For reasons of simplicity we use the *one-hot assignment* only, i.e. we introduce one symbol per state. For the decomposition step from $\mathcal{L}_4$ to $\mathcal{L}_3$ we change internal symbols into external symbols by applying a technique called *expansion* of internal symbols. For each internal symbol x of the component we introduce symbols ox and ix and expand each atomic command $!x?$ into $ox!;ix?$. The terminals ox and ix are then connected by a WIRE component.

6.1. DECOMPOSITION OF $\mathcal{L}_2$ INTO $\mathcal{L}_1$

6.1.0. Introduction

In the decomposition step from $\mathcal{L}_2$ to $\mathcal{L}_1$ each component $E0 \in \mathcal{L}(G2') \setminus \mathcal{L}(GCL')$ is decomposed into components $E1$, $E2$, and $E3$ such that $E1 \in \mathcal{L}(GCL')$, $E2 \in \mathcal{L}(GCL')$, and the command $E3$ is a weave of SOURCE and SINK components with disjoint alphabets. Consequently, by Corollary 3.1.3.3, component $E3$ can be decomposed further into a collection of SOURCE and SINK components. The commands $E1$, $E2$, and $E3$ are constructed form the syntax of $E0$, and we have $|E1|+|E2|+|E3|=\mathcal{O}(|E0|)$. The general connection pattern between the components $E1$, $E2$, and $E3$ is given in Figure 6.1.0.

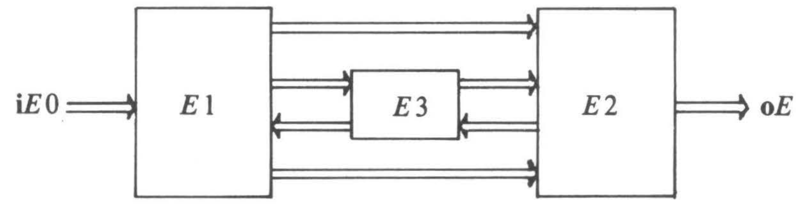


FIGURE 6.1.0. General connection pattern of decomposition of $E \in \mathcal{L}_2$.

Notice that for any command E it can be determined in a constructive way whether $E \in \mathcal{L}(G2') \setminus \mathcal{L}(GCL')$ by means of the grammars $G2'$ and GCL'. Before we describe the general decomposition procedure for this step, we give an example.

6.1.1. An example

Let the commands $E.0.0$ and $E.1.0$ be defined by

$$E.0.0 = \mathbf{pref}[(a?|b?);d0!;(a?;e! \mid b?;d0!)]$$

$$E.1.0 = \mu.tailf_1.0,$$

where $tailf_1$ is specified by

$$tailf_1.R.0 = \mathbf{pref}(e!;R.1)$$

$$tailf_1.R.1 = \mathbf{pref}(c?;R.0 \mid b?;R.2)$$

$$tailf_1.R.2 = \mathbf{pref}(R.2)$$

$$tailf_1.R.3 = \mathbf{pref}(d1!;R.1).$$

The state graph corresponding to $\mu.tailf_1.0$ is given in Figure 6.1.1.

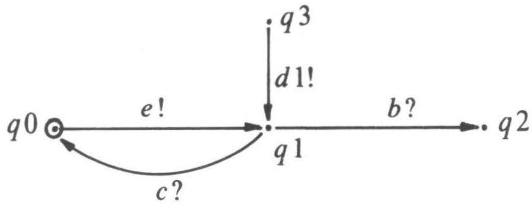


FIGURE 6.1.1. State graph corresponding to $tailf_1$.

We observe that $E.0.0 \| E.1.0 \in \mathcal{L}(G2') \setminus \mathcal{L}(GCL')$.

The decomposition of $E.0.0 \| E.1.0$ consists of two steps. In the first step we rewrite $E.0.0$ and $E.1.0$ into commands of the form $\mu.tailf.0 \in \mathcal{L}(G2')$, where *tailf* is defined by an array of atomic commands only. The sequential command $E.1.0$ is already written in this way. For $E.0.0$ we obtain the command $\mu.tailf_0.0$, where $tailf_0$ is specified by

$$tailf_0.R.0 = \mathbf{pref}(a?;R.1 \mid b?;R.1)$$

$$tailf_0.R.1 = \mathbf{pref}(d0!;R.2)$$

$$tailf_0.R.2 = \mathbf{pref}(a?;R.3 \mid b?;R.4)$$

$$tailf_0.R.3 = \mathbf{pref}(e!;R.0)$$

$$tailf_0.R.4 = \mathbf{pref}(d0!;R.0).$$

Rewriting a sequential command in such a form can be done in a syntax-directed way.

In the second step we apply a *state assignment* to each sequential command $E.k.0$, $0 \leqslant k < 2$. For reasons of simplicity, we take the so-called *one-hot assignment,* i.e. we introduce one internal symbol per state. For state i of sequential command $E.k.0$ we introduce the internal symbol $q.k.i$. Next, we split each sequential command into an *input part* and an *output part.* The input parts $E.0.1$ and $E.1.1$ and output parts $E.0.2$ and $E.1.2$ are defined by

$$\begin{aligned} E.0.1 = \mathbf{pref}[&a?\|q.0.0?;q.0.1! \mid b?\|q.0.0?;q.0.1! \\ &|a?\|q.0.2?;q.0.3! \mid b?\|q.0.2?;q.0.4! \\ &] \end{aligned}$$

$$E.0.2 = \mathbf{pref}(q.0.0!;[q.0.1?;d0!\|q.0.2! \mid q.0.3?;e!\|q.0.0!])$$

$$E.1.1 = \mathbf{pref}(q.1.0!;[q.1.1?\|c?;q.1.0! \mid q.1.1?\|b?;q.1.2!])$$

$$E.1.2 = \mathbf{pref}[q.1.0?;e!\|q.1.1! \mid q.1.3?;d1!\|q.1.1!].$$

Operationally speaking, an input part receives the current local state and an input and then produces the next local state. The output part receives the current local state upon which it produces the output and the next local state. Depending on whether an input or an output is produced initially, the input part or the output part starts with producing the initial state.

Not every internal symbol occurs both as an input and as an output in the above commands: there is a dangling input $q.1.3$ and a dangling output $q.1.2$. To connect this dangling input and output to an output and an input respectively, we introduce the passive SOURCE($q.1.3$) component and the SINK($q.1.2$) component. Let $E.0.3$ and $E.1.3$ be defined by

$$E.0.3 = \epsilon \text{ and}$$

$$E.1.3 = \text{SOURCE}(q.1.3) \,\|\, \text{SINK}(q.1.2).$$

By definition of decomposition, we derive

$$E.0.0 \rightarrow E.0.1, E.0.2, E.0.3 \text{ and}$$

$$E.1.0 \rightarrow E.1.1, E.1.2, E.1.3\,.$$

We check condition (3.7) and (3.8) for the application of the Separation Theorem and infer that the internal symbols of the decompositions are row-wise disjoint and that the outputs are column-wise disjoint. Consequently, by the Separation Theorem, we deduce

$$E0 \rightarrow E1,\ E2,\ E3, \text{ where}$$

$$E0 = E.0.0 \,\|\, E.1.0\,,$$

$$E1 = E.0.1 \,\|\, E.1.1\,,$$

$$E2 = E.0.2 \,\|\, E.1.2 \text{ ,and}$$

$$E3 = E.0.3 \,\|\, E.1.3\,.$$

Furthermore, we observe $E.1 \in \mathcal{L}(GCL')$, $E.2 \in \mathcal{L}(GCL')$, and

$$|E1| + |E2| + |E3| = \mathcal{O}(|E0|).$$

6.1.2. The general decomposition

The general decomposition method for any component $E0 \in \mathcal{L}(G2') \setminus \mathcal{L}(GCL')$ is carried out similarly to the previous example. By definition of grammar $G2'$, command $E0 \in \mathcal{L}(G2')$ is expressed as a weave $(\|k: 0 \leq k < N: E.k.0)$ of sequential commands $E.k.0 \in \mathcal{L}(G2')$. First, each sequential command $E.k.0$, $0 \leq k < N$, is rewritten into a command $\mu.tailf_k.0 \in \mathcal{L}(G2')$, where $tailf_k$ is a tail function defined by an array of atomic commands only . Let $e.k(i,j: 0 \leq i,j < n(k))$ denote an array of atomic commands for $tailf_k$, $0 \leq k < N$. Rewriting sequential command $E.k.0$ into $\mu.tailf_k.0$ can be done in a syntax-directed way such that

$$(\mathbf{N}k,i,j : 0 \leq k < N \wedge 0 \leq i,j < n(k) : e.k.i.j \neq \varnothing) = \mathcal{O}(|E0|). \qquad (6.0)$$

In the second step we introduce the internal symbols $q.k.i$ and split each sequential command in an input part and an output part. First, we define for each k, $0 \leq k < N$, the commands $PI.k$ and $PO.k$ as follows. If $e.k$ contains inputs,

$$PI.k = (|\ i,j : e.k.i.j \text{ is an input} : e.k.i.j \| q.k.i?; q.k.j!),$$

otherwise $PI.k = \epsilon$. If $e.k$ contains outputs,

$$PO.k = (|\ i,j : e.k.i.j \text{ is an output} : q.k.i?; q.k.j! \| e.k.i.j),$$

otherwise $PO.k = \epsilon$. Since $\mu.tailf_k.0 \in \mathcal{L}(G2')$, $0 \leq k < N$, it follows that $PI.k$ and $PO.k$ satisfy the LL-1 conditions. (Notice that for each i, $0 \leq i < n(k)$, there exists at most one j, $0 \leq j < n(k)$, such that $e.k.i.j$ is an output.) Subsequently, input part $E.k.1$ and output part $E.k.2$, $0 \leq k < N$, are defined by

$$
\begin{array}{lll}
E.k.1 & = \mathbf{pref}(q.0.0!;[PI.k]) & \text{if } PI.k \neq \epsilon \wedge Q.k \\
 & = \mathbf{pref}(q.0.0!) & \text{if } PI.k = \epsilon \wedge Q.k \\
 & = \mathbf{pref}[PI.k] & \text{if } PI.k \neq \epsilon \wedge \neg(Q.k) \\
 & = \epsilon & \text{if } PI.k = \epsilon \wedge \neg(Q.k) \\
E.k.2 & = \mathbf{pref}(q.0.0!;[PO.k]) & \text{if } PO.k \neq \epsilon \wedge \neg(Q.k) \\
 & = \mathbf{pref}(q.0.0!) & \text{if } PO.k = \epsilon \wedge \neg(Q.k) \\
 & = \mathbf{pref}[PO.k] & \text{if } PO.k \neq \epsilon \wedge Q.k \\
 & = \epsilon & \text{if } PO.k = \epsilon \wedge Q.k,
\end{array}
$$

where $Q.k \equiv$ '$E.k.0$ starts with an output', for all $0 \leq k < N$.

SOURCE and SINK components are introduced for dangling inputs or outputs as follows. For each k, $0 \leq k < N$, $Out.k$ and $In.k$ are defined by

$$Out.k = \mathbf{o}(E.k.1) \cup \mathbf{o}(E.k.2) \cup \{q.0.0\} \quad \text{and}$$

$$In.k = \mathbf{i}(E.k.1) \cup \mathbf{i}(E.k.2).$$

For each $q.k.i \in Out.k \setminus In.k$ we introduce a SINK($q.k.i$) component, and for each $q.k.i \in In.k \setminus Out.k$ we introduce a passive SOURCE($q.k.i$) component, where $0 \leq i < n(k) \wedge 0 \leq k < N$. Command $E.k.3$ is defined as the weave of these SINK and SOURCE components.

With the above definitions we derive for all k, $0 \leq k < N$,

$$E.k.0 \rightarrow E.k.1, E.k.2, E.k.3.$$

Since for these decompositions the internal symbols are row-wise disjoint and the outputs are column-wise disjoint, we deduce, by the Separation Theorem,

$$E0 \to E1, E2, E3, \text{ where}$$

$$E0 = (\|k: 0 \leq k < N: E.k.0),$$

$$E1 = (\|k: 0 \leq k < N: E.k.1),$$

$$E2 = (\|k: 0 \leq k < N: E.k.2), \text{ and}$$

$$E3 = (\|k: 0 \leq k < N: E.k.3).$$

Because $PI.k$ and $PO.k$, $0 \leq k < N$, satisfy the LL-1 conditions, we infer that $E1 \in \mathcal{L}(GCL')$, $E2 \in \mathcal{L}(GCL')$, and $E3$ is a weave of SOURCE and SINK components with disjoint alphabets. Finally, we observe that $E1$, $E2$, and $E3$ are constructed from the syntax of $E0$ and that, by (6.0),

$$|E1| + |E2| + |E3| = \Theta(|E0|).$$

6.1.3. Schematics of decompositions

Recall the specification of component $E0 \| E1$ of Section 6.1.1, where

$$E0 = \mathbf{pref}[(a? | b?); d0!; (a?; e! \mid b?; d0!)] ,$$

$$E1 = \mu.tailf_1.0,$$

and $tailf_1$ is specified by

$$tailf_1.R.0 = \mathbf{pref}(e!; R.1)$$

$$tailf_1.R.1 = \mathbf{pref}(c?; R.0 \mid b?; R.2)$$

$$tailf_1.R.2 = \mathbf{pref}(R.2)$$

$$tailf_1.R.3 = \mathbf{pref}(d1!; R.1).$$

A schematic of the complete decomposition of component $E0 \| E1$ according to the methods described in the previous sections is given in Figure 6.1.2. The schematic of this decomposition can also be rearranged into a connection of a CT, XOR, and a CEL plane.

The decomposition of the sequence detector of Section 2.3.1 according to the methods of the preceding sections yields the schematic of Figure 6.1.3.

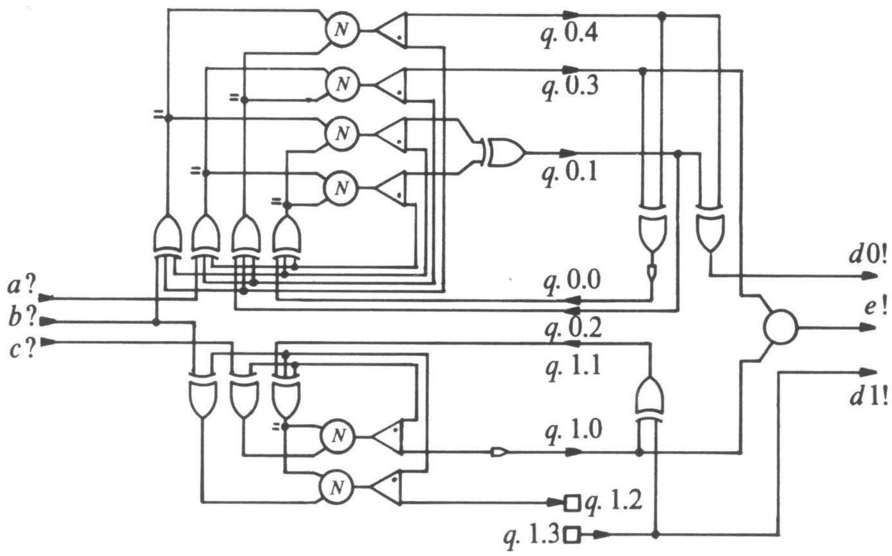


FIGURE 6.1.2. Decomposition of $E0\|E1$.

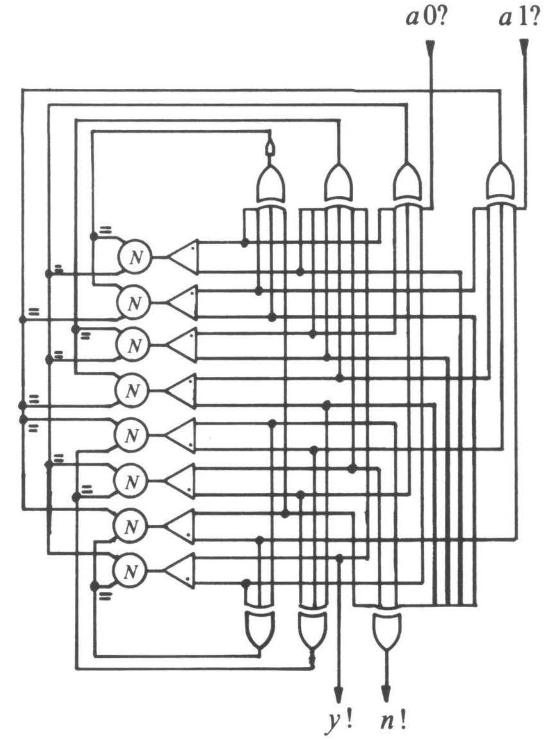


FIGURE 6.1.3. Decomposition of sequence detector.

6.2. Decomposition of $\mathcal{L}_3$ into $\mathcal{L}_2$

6.2.0. *Introduction*

In the decomposition step from $\mathcal{L}_3$ to $\mathcal{L}_2$ each component $E.0 \in \mathcal{L}_3 \setminus \mathcal{L}_2$ $(=\mathcal{L}(G3') \setminus \mathcal{L}(G2'))$ is decomposed in a syntax-directed way into a collection of components $E.i \in \mathcal{L}_2$, $1 \leq i < n$. (Notice that for each command E it can also be determined in a constructive way whether $E \in \mathcal{L}(G3') \setminus \mathcal{L}(G2')$ by means of the grammars $G3'$ and $G2'$.) The decomposition can be carried out in such a way that the result is linear in the length of the commands, i.e. $(+i : 1 \leq i < n : |E.i|) = \Theta(|E.0|)$. The step is divided into three substeps each of which is discussed briefly below before they are presented in the following sections.

In the first step each component $E0 \in \mathcal{L}_3 \setminus \mathcal{L}_2$ is decomposed into four components $E1$, $E2$, $E3$, and $E4$. Apart from component $E4$, this step is similar to the decomposition step from $\mathcal{L}_2$ to $\mathcal{L}_1$. The connection pattern between components $E1$, $E2$, $E3$, and $E4$ is given in Figure 6.2.0.

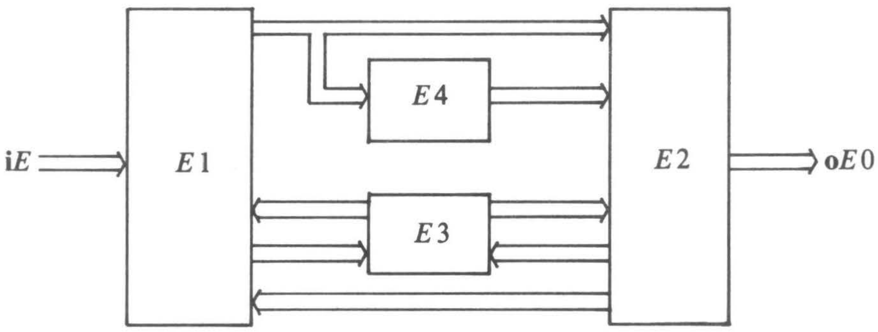


FIGURE 6.2.0. Decomposition of $E0 \in \mathcal{L}_3 \setminus \mathcal{L}_2$.

Components $E1$ and $E2$ are called the input part and output part respectively. We have $E1 \in \mathcal{L}(GCL')$, $E2 \in \mathcal{L}(GCL')$, and component $E3$ is a weave of SINK and (passive) SOURCE components with disjoint alphabets. Consequently, by Corollary 3.1.3.3, $E3$ can be decomposed further into SINK and SOURCE components. Component $E4$ is called the *selection part,* and command $E4$ satisfies a special syntax: we have $E4 \in \mathcal{L}(GSEL)$. Grammar $GSEL$ is presented in the next section. The commands $E1$, $E2$, $E3$, and $E4$ are constructed from the syntax of $E0$ in such a way that $|E1| + |E2| + |E3| + |E4| = \Theta(|E0|)$.

In the second step each selection part $E0 \in \mathcal{L}(GSEL)$ is decomposed into components $E1$, $E2$, and $E3$. The general connection pattern of this decomposition is depicted in Figure 6.2.1.

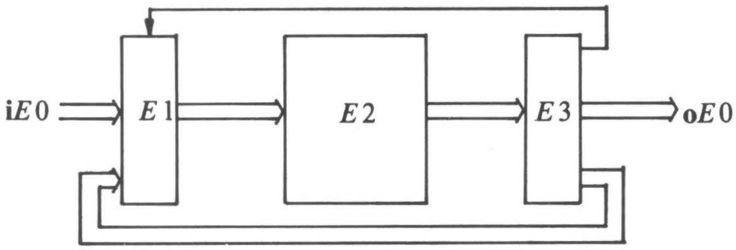


FIGURE 6.2.1. Decomposition of $E0 \in \mathfrak{L}(GSEL)$

Component $E1$ is a SEQ component, $E2 \in \mathfrak{L}(G2')$, and $E3 \in \mathfrak{L}(GCL')$. With respect to the length of these commands we have $|E1| = \Theta(|E0|)$ and $|E3| = \Theta(|E0|)$, but in general, however, we do not have $|E2| = \Theta(|E0|)$. Consequently, if $E2$ is decomposed according to methods discussed in previous sections, the decomposition of components $E0 \in \mathfrak{L}(GSEL)$ is in general not linear in the length of $E0$. Nevertheless, we show that it is possible —though more difficult— to obtain a linear decomposition of $E0$. For this purpose we decompose $E2$ further into a component $MASTER \in \mathfrak{L}(G2')$ and components $SLAVE.i \in \mathfrak{L}(G2')$, $0 \leq i < m$. The commands $MASTER$ and $SLAVE.i$, $0 \leq i < m$, are constructed from the syntax of $E0$ and satisfy

$$|MASTER| + (+i{:}\, 0 \leq i < m{:}\, |SLAVE.i|) = \Theta(|E0|).$$

Because the complete linear decomposition of a component $E0 \in \mathfrak{L}(GSEL)$ can become rather complicated, the non-linear decomposition is to be preferred in many cases to the linear decomposition. The decomposition into $MASTER$ and $SLAVE$ components is discussed in Section 6.2.5 and may be skipped at first reading.

In the third and final step each SEQ component is decomposed in a syntax-directed way into the basis $\mathbb{B}$. We demonstrate that a k-SEQ component, $k > 0$, can be decomposed into $\Theta(k)$ basic components from $\mathbb{B}$.

If the three steps are combined, we conclude, from the Substitution Theorem and from the properties that each step satisfies, that each component $E.0 \in \mathfrak{L}_3 \setminus \mathfrak{L}_2$ can be decomposed in a syntax-directed way into a collection of components $E.i \in \mathfrak{L}_2$, $1 \leq i < n$. Moreover, if in the second step the linear decomposition is applied, we have $(+i{:}\, 1 \leq i < n{:}\, |E.i|) = \Theta(|E.0|)$.

In each of the following sections a decomposition step is explained. We start with the definition of grammar $GSEL$, and subsequently, in Section 6.2.2, we discuss an example of the first decomposition step.

6.2.1. DI grammar GSEL

The grammar *GSEL* is an attribute grammar similar to *GCL'*. The production rules for its context-free grammar are defined as follows.

<dicom> ::=		*<pccom>*	
<pccom> ::=		ϵ	
	[]	**pref**[*<pfcom>*]	
	[]	*<com>*‖*<com>*	(b9)
<pfcom> ::=		*<sym>*?;(*<altout>*)	(c6)
	[]	*<pfcom>*|*<pfcom>*	(c7)
<altout> ::=		*<sym>*!	
	[]	*<altout>*|*<altout>*	(d0)

The conditions for the production rules (b9), (c6), (c7), and (d0) are as follows. For each of the rules we have the condition

$$ALFCOND(E0,E1) \wedge (\mathbf{i}E0 \cap \mathbf{i}E1) = \varnothing,$$

where $E0\|E1$, $E0;E1$, $E0|E1$, and $E0|E1$ are productions of the production rules (b9), (c6), (c7), and (d0) respectively. Consequently, all inputs in a command $E \in \mathcal{L}(GSEL)$ differ. For the production rules (c7) and (d0) we have the additional condition $ALTCOND(E0,E1)$. For example, we have

$\mathbf{pref}[a?;(b!|c!)] \in \mathcal{L}(GSEL)$, and

$\mathbf{pref}[d?;a! \mid e?;(a!|b!)] \;\|\; \mathbf{pref}[f?;(a!|c!) \mid g?;a!] \in \mathcal{L}(GSEL)$.

Notice that $\mathcal{L}(GSEL) \subseteq \mathcal{L}(G4)$. Consequently, the attribute grammar *GSEL* is also a DI grammar.

6.2.2. An example

Let the commands $E.0.0$ and $E.1.0$ be specified by

$E.0.0 = \mathbf{pref}[a?;c!;a?;(c!|d!)]$ and

$E.1.0 = \mathbf{pref}[b?;(c! \mid e!;b?;c!)]$.

We observe that $E.0.0\|E.1.0 \in \mathcal{L}(G3') \setminus \mathcal{L}(G2')$. In the following, we construct a decomposition for component $E.0.0\|E.1.0$ from the syntax of $E.0.0$ and $E.1.0$.

First, the commands $E.0.0$ and $E.1.0$ are rewritten in a syntax-directed way into the commands $\mu.tailf_0.0 \in \mathcal{L}(G3')$ and $\mu.tailf_1.0 \in \mathcal{L}(G3')$ respectively, where $tailf_0$ and $tailf_1$ are defined by arrays of atomic commands only. We obtain for $tailf_0$ and $tailf_1$,

$tailf_0.R.0 = \mathbf{pref}(a?;R.1)$

$tailf_0.R.1 = \mathbf{pref}(c!;R.2)$

$$tailf_0.R.2 = \mathbf{pref}(a?;R.3)$$
$$tailf_0.R.3 = \mathbf{pref}(c!;R.0 \mid d!;R.0)$$

and

$$tailf_1.R.0 = \mathbf{pref}(b?;R.1)$$
$$tailf_1.R.1 = \mathbf{pref}(c!;R.0 \mid e!;R.2)$$
$$tailf_1.R.2 = \mathbf{pref}(b?;R.3)$$
$$tailf_1.R.3 = \mathbf{pref}(c!;R.0).$$

Second, we apply a one-hot assignment to each sequential command. For state i of sequential command k we introduce the internal symbol $q.k.i$. Furthermore, for each sequential command $E.k.0$, $0 \leq k < 2$, we introduce the internal symbols x' for each $x \in \mathbf{o}(E.k.0)$. The commands $E.k.i$, $0 \leq k < 2 \wedge 1 \leq i < 5$, are defined as follows.

$$E.0.1 = \mathbf{pref}[q.0.0?\|a?;q.0.1! \mid q.0.2?\|a?;q.0.3!]\ ,$$
$$E.0.2 = \mathbf{pref}[q.0.1?\|c'?;q.0.2!\|c! \\ \mid q.0.3?\|c'?;q.0.0!\|c! \mid q.0.3?\|d'?;q.0.0!\|d! \\ ]\ ,$$
$$E.0.3 = \epsilon\ ,$$
$$E.0.4 = \mathbf{pref}[q.0.1?;c'! \mid q.0.3?;(c'!|d'!)]\ ,$$
$$E.1.1 = \mathbf{pref}[q.1.0?\|b?;q.1.1! \mid q.1.2?\|b?;q.1.3!]\ ,$$
$$E.1.2 = \mathbf{pref}[q.1.1?\|c'?;q.1.0!\|c! \mid q.1.1?\|e'?;q.1.2!\|e! \\ \mid q.1.3?\|c'?;q.1.0!\|c! \\ ]\ ,$$
$$E.1.3 = \epsilon\ , \text{ and}$$
$$E.1.4 = \mathbf{pref}[q.1.1?;(c'!|e'!) \mid q.1.3?;c'!].$$

Components $E.0.1$ and $E.1.1$ are the input parts of components $E.0.0$ and $E.1.0$ respectively. Components $E.0.2$ and $E.1.2$ are the output parts of components $E.0.0$ and $E.1.0$ respectively. $E.0.3$ and $E.1.3$ represent the weaves of the SOURCE and SINK components (of which there are none here). Components $E.0.4$ and $E.1.4$ are the selection parts of $E.0.0$ and $E.1.0$ respectively. The input parts determine from a current local state and an input the next local state. The output parts determine from the current local state and an internal symbol x' the next local state and the next output. (Notice that the output parts here differ from the output parts introduced in the decomposition from $\mathcal{L}_2$ to $\mathcal{L}_1$.) The selection part selects for a local state a next internal symbol x'.

By definition of decomposition, we have

$$E.0.0 \rightarrow E.0.1,\ E.0.2,\ E.0.3,\ E.0.4 \quad \text{and}$$

$$E.1.0 \rightarrow E.1.1,\ E.1.2,\ E.1.3,\ E.1.4\,.$$

In order to apply the Separation Theorem to the command $E.0.0 \| E.1.0$, we verify conditions (3.7) and (3.8). We observe that the outputs of the decompositions are column-wise disjoint, but the internal symbols of the decompositions, however, are not row-wise disjoint because of the symbols x'. Consequently, we can only conclude, by Theorem 3.1.3.1, that the connection of components $\overline{E0}$, $E1$, $E2$, $E3$ and $E4$, where

$$E0 = E.0.0 \| E.1.0\,,$$

$$E1 = E.0.1 \| E.1.1\,,$$

$$E2 = E.0.2 \| E.1.2\,,$$

$$E3 = E.0.3 \| E.1.3\,, \text{and}$$

$$E4 = E.0.4 \| E.1.4,$$

is closed and free of interference. We still have to show that $\mathbf{t}W \restriction \mathbf{a}E0 = \mathbf{t}E0$, where $W = \overline{E0} \| E1 \| E2 \| E3 \| E4$. By definition of weaving, we derive $\mathbf{t}W \restriction \mathbf{a}E0 \subseteq \mathbf{t}E0$. Furthermore, for the above kind of decomposition we can also show that any trace $t \in \mathbf{t}E0$ can be expanded with internal symbols into a trace in $\mathbf{t}W$. For example, the trace $a\,b\,c\,b\,a\,d$ can be expanded into

$$q.0.0\ a\ q.0.1\ q.1.0\ b\ q.1.1\ c'\ q.0.2\ q.1.0\ c\ b\ q.1.1\ a\ \ q.0.3\ d'\ q.0.0\ d \in \mathbf{t}W.$$

In general, the expansion consists of inserting the symbols for the local states and the internal symbols x' at the appropriate places. Consequently, we derive

$$E0 \rightarrow E1,\ E2,\ E3,\ E4.$$

Subsequently, from the definition of these commands, we observe $E1 \in \mathcal{L}(GCL')$, $E2 \in \mathcal{L}(GCL')$, and $E4 \in \mathcal{L}(GSEL)$. (Notice that in $E4$ all inputs differ.) The commands are constructed from the syntax of $E.0.0$ and $E.1.0$, and

$$|E1| + |E2| + |E3| + |E4| = \Theta(|E0|).$$

6.2.3. *The general decomposition*

The general decomposition of a component $E0 \in \mathcal{L}_3 \setminus \mathcal{L}_2$ into components $E1 \in \mathcal{L}(GCL')$, $E2 \in \mathcal{L}(GCL')$, a weave $E3$ of SINK and SOURCE components, and $E4 \in \mathcal{L}(GSEL)$ is performed in two steps as follows.

Let the command $E0 \in \mathcal{L}(G3') \setminus \mathcal{L}(G2')$ be expressed as a weave of sequential commands $E.k.0 \in \mathcal{L}(G3')$, $0 \leqslant k < N$. First, we rewrite each sequential command $E.k.0$ into a command $\mu.tailf_k.0 \in \mathcal{L}(G3')$, $0 \leqslant k < N$, where $tailf_k$ is defined by an array of atomic commands only. Let for each k, $0 \leqslant k < N$, array

$e.k(i,j:0 \leq i,j < n(k))$ denote the array of atomic commands for $tailf_k$. Each sequential command $E.k.0$ can be rewritten in a syntax-directed way into a command $\mu.tailf_k.0 \in \mathcal{L}(G3')$ such that

$$(\mathbf{N}k,i,j: 0 \leq k < N \wedge 0 \leq i,j < n(k): e.k.i.j \neq \varnothing) = \Theta(|E0|). \qquad (6.1)$$

Second, we define the input part $E.k.1$, the output part $E.k.2$, the weave $E.k.3$ of SOURCE and SINK components, and the selection part $E.k.4$ by means of array $e.k(i,j:0 \leq i,j < n(k))$ for each k, $0 \leq k < N$. The commands $E.k.1$, $0 \leq k < N$, are defined analogously to Section 6.1.2. The commands $E.k.2$, $0 \leq k < N$, are also defined analogously to Section 6.1.2, apart from the definition of $PO.k$, which is defined as follows. Let array $e'.k(i,j:0 \leq i,j < n)$ denote array $e.k(i,j:0 \leq i,j < n)$ in which each atomic command $x!$ and $x?$ is replaced by x', for $0 \leq k < N$. If array $e.k$ contains outputs for $0 \leq k < N$, then

$$PO.k = (|i,j: e.k.i.j \text{ is an output}: q.k.i?\|e'.k.i.j? \; ; \; q.k.j!\|e.k.i.j),$$

otherwise $PO.k = \epsilon$. Notice that, since $\mu.tailf_k.0 \in \mathcal{L}(G3')$, $PO.k$ satisfies the LL-1 conditions.

The selection part $E.k.4$ is defined as follows for $0 \leq k < N$. If array $e.k$ contains outputs,

$$E.k.4 = \mathbf{pref}[(|i: e.k.i \text{ contains an output} \\ : q.k.i?; (|j: e.k.i.j \text{ is an output}: e'.k.i.j!) \\ )],$$

otherwise $E.k.4 = \epsilon$. Since $\mu.tailf_k.0 \in \mathcal{L}(G3')$, it follows that each $E.k.4$, $0 \leq k < N$, satisfies the LL-1 conditions.

Finally, we determine, with these definitions of $E.k.1$, $E.k.2$, and $E.k.4$, which internal symbols are a dangling input or output. For each such symbol we introduce a passive SOURCE or a SINK component respectively, and the weave of these components for each k, $0 \leq k < N$, is denoted by $E.k.3$.

Subsequently, by these definitions and the definition of decomposition, we conclude for all k, $0 \leq k < N$,

$$E.k.0 \rightarrow E.k.1,\ E.k.2,\ E.k.3,\ E.k.4\,.$$

For these decompositions we check conditions (3.7) and (3.8) of the Separation Theorem. We observe that the outputs of these components are column-wise disjoint. In general, the internal symbols of these decompositions, however, do not have to be row-wise disjoint. By Theorem 3.1.3.1 we can, therefore, only conclude that the connection of components $\overline{E0}$, $E1$, $E2$, $E3$, and $E4$ is closed and free of interference, where

$$E0 = (\|k: 0 \leq k < N: E.k.0)\,,$$

$$E1 = (\|k: 0 \leq k < N: E.k.1)\,,$$

$$E2 = (\|k: 0 \leq k < N: E.k.2)\,,$$

$$E3 = (\|k: 0 \leq k < N: E.k.3)\,, \text{ and}$$

$$E4 = (\|k: 0 \leqslant k < N: E.k.4).$$

We prove that $\mathbf{t}W\!\upharpoonright\!\mathbf{a}E0 = \mathbf{t}E0$ holds as well, where $W = \overline{E0} \| E1 \| E2 \| E3 \| E4$. By definition of weaving, we have $\mathbf{t}W\!\upharpoonright\!\mathbf{a}E0 \subseteq \mathbf{t}E0$. Furthermore, from the definitions of $E.k.i$, $0 \leqslant k < N \wedge 1 \leqslant i < 5$, we derive that any trace in $\mathbf{t}E0$ can be expanded into a trace in $\mathbf{t}W$ by inserting the symbols for the local states and internal symbols x' at the appropriate places. Consequently, we have $\mathbf{t}W\!\upharpoonright\!\mathbf{a}E0 = \mathbf{t}E0$, and we infer by definition of decomposition

$$E0 \rightarrow E1,\ E2,\ E3,\ E4.$$

Moreover, we observe $E1 \in \mathcal{L}(GCL')$, $E2 \in \mathcal{L}(GCL')$, $E3$ is a weave of SOURCE and SINK components with disjoint alphabets, $E4 \in \mathcal{L}(GSEL)$, and by (6.1)

$$|E1| + |E2| + |E3| + |E4| = \mathcal{O}(|E0|).$$

6.2.4. *Decomposition of* $\mathcal{L}(GSEL)$

Components expressed by commands in $\mathcal{L}(GSEL)$ have to perform some kind of a selection. For example, the component $E = \mathbf{pref}[a?;(b!|c!)]$ has to select after receipt of input a an output from the set $\{b,c\}$, i.e. from the outputs in $Suc(a,E)$. The component E, where

$$E = \mathbf{pref}[d?;a! \mid e?;(a!|b!)] \parallel \mathbf{pref}[f?;(a!|c!) \mid g?;a!],$$

has to select after receipt of input f an output from the set $\{c\}$, i.e. from the outputs in $Suc(f,E)$. (Notice that $a \notin Suc(f,E)$.) After receipt of inputs f and e, however, this component has to select an output from the set $\{a,b,c\}$, i.e. from the outputs in $Suc(fe,E)$.

In the decomposition of components expressed in $\mathcal{L}(GSEL)$ the selections of outputs are realized by a connection of a SEQ component and components expressed in $\mathcal{L}_2$. Which output is selected is determined by the order in which requests are sequenced by the SEQ component. It is because of this sequencing of requests that the selection of a next output can be computed in a deterministic way, i.e. by components expressed in $\mathcal{L}_2$.

Let $E0 \in \mathcal{L}(GSEL)$. We show how to construct a decomposition for component $E0$. The construction of this decomposition can briefly be described as follows. First, we introduce so-called *auxiliary symbols* and construct the command E' from $E0$. Subsequently, we construct the commands $E1$, $E2$, and $E3$ from the command E'. Component $E1$ is a SEQ component with $|E1| = \mathcal{O}(|E0|)$, $E2$ is a sequential command from $\mathcal{L}(G2')$, and $E3 \in \mathcal{L}(GCL')$ with $|E3| = \mathcal{O}(|E0|)$. In the following, we first give the definition of the commands E', $E1$, $E2$, and $E3$ and then present an example. The connection pattern between these components is given in Figure 6.2.1. Finally, we prove $E0 \rightarrow E1, E2, E3$ and devote a few remarks to this decomposition.

The command E' is defined by

$$E' = E0 \parallel (\parallel x : x \in \mathbf{o}E0 : \mathbf{pref}[hx?;x!]).$$

For example, for $E0 = \mathbf{pref}[a?;(b!|c!)]$ we have

$$E' = E0 \parallel \mathbf{pref}[hb?;b!] \parallel \mathbf{pref}[hc?;c!].$$

The symbols hx, for $x \in \mathbf{o}E0$, are called *auxiliary symbols.* (We assume that $hx \notin \mathbf{a}E0$ for $x \in \mathbf{o}E0$.) Notice that $|E'| = \Theta(|E0|)$.

From command E' the commands $E1$, $E2$, and $E3$ are constructed. $E1$ is a k-SEQ component, where k equals the number of inputs of E'. $E1$ is defined by

$$\begin{aligned} E1 = {} & (\parallel x : x \in \mathbf{i}E' : \mathbf{pref}[x?;x''!]) \\ & \parallel \mathbf{pref}[n?;(|x : x \in \mathbf{i}E' : x''!)]. \end{aligned}$$

The command $E3$ is defined by

$$\begin{aligned} E3 = {} & (\parallel x : x \in \mathbf{o}E' : \mathbf{pref}(hx!;[x''?;hx! \parallel x!]) \\ & \parallel \mathbf{pref}(n!;[(|x : x \in \mathbf{o}E' : x''?;n!) \mid np?;n!]). \end{aligned}$$

Command $E2$ is defined by $E2 = \mu.tailf.0$, where $tailf$ is defined below. For the definition of $tailf$ we use the command E'' which is the command E' in which every symbol y is replaced by y''. The sequential behavior of component $E2$ is an alternation of inputs of E'' and outputs from $\{np\} \cup \mathbf{o}E''$, starting with an input. The output is determined by the inputs as follows. Let t be the current trace and let x'' be the next input. If $Suc(tx'' \upharpoonright \mathbf{a}E'', E'')$ contains an output, then the first one is produced. (For the time being we assume that $Suc(tx'' \upharpoonright \mathbf{a}E'', E'')$ is represented as a list of symbols.) If $Suc(tx'' \upharpoonright \mathbf{a}E'', E'')$ does not contain an output, then output np is produced. In order to formalize this specification we introduce some notation. Let $q.i$, $0 \leq i < n1$, denote the states of E''. By $t.i$ we denote a trace from state $q.i$, $0 \leq i < n1$. Let

$$V = \{i \mid 0 \leq i < n1 \wedge Suc(t.i, E'') \cap \mathbf{o}E'' = \varnothing\},$$

i.e. the set V is the set of all (indexes of the) states of E'' in which no output can be produced. The initial state is denoted by $q.0$, and, since E'' starts with inputs, we have $0 \in V$. For all $i \in V$, $tailf.R.i$ is defined by

$$\begin{aligned} tailf.R.i = {} & \mathbf{pref}((|x'' : D0(i,x'') : \ x''?;p(i,x'')!;R.\delta0(i,x'')) \\ & \qquad |(|x'' : D1(i,x'') : \ x''?;np! \quad ;R.\delta1(i,x'')) \\ & \qquad) \qquad\qquad \text{if } Suc(t.i, E'') \neq \varnothing \\ = {} & \mathbf{pref}(R.i) \qquad\qquad \text{otherwise,} \end{aligned}$$

where

$$D0(i,x'') \equiv Suc(t.i\,x'', E'') \cap \mathbf{o}E'' \neq \varnothing,$$

$$D1(i,x'') \equiv t.i\,x'' \in \mathbf{t}E'' \wedge \neg D0(i,x''),$$

$$\delta 1(i,x'') = j, \text{ where } t.i\,x'' \in q.j,$$

$$\delta 0(i,x'') = j, \text{ where } t.i\,x''p(i,x'') \in q.j, \text{ and}$$

$$p(i,x'') = \text{first output in } Suc\,(t.i\,x'',E'').$$

We assume that $Suc\,(r,E'')$ and $Suc\,(s,E'')$, for r and s traces of the same state of E'', represent the same list of symbols. Furthermore, we stipulate that if one of the domains $D0$ or $D1$ is empty, the corresponding quantified union is omitted. (Notice that only one domain can be empty.) We observe that *tailf* is well-defined, since for $E'' \in \mathfrak{L}(GSEL)$ we have

$$i \in V \wedge D0(i,x'') \Rightarrow \delta 0(i,x'') \in V \text{ and}$$

$$i \in V \wedge D1(i,x'') \Rightarrow \delta 1(i,x'') \in V.$$

EXAMPLE 6.2.4.0. Let $E0$ be defined by $E0 = \mathbf{pref}[a?;(b!|c!)]$. We construct the commands $E', E'', E1, E2$, and $E3$ according to the definitions given above. We obtain

$$E' = \mathbf{pref}[a?:(b!|c!)] \,\|\, \mathbf{pref}[hb?;b!] \,\|\, \mathbf{pref}[hc?;c!]$$

$$E'' = \mathbf{pref}[a''?;(b''!|c''!)] \,\|\, \mathbf{pref}[hb''?;b''!] \,\|\, \mathbf{pref}[hc''?;c''!]$$

$$E1 = \mathbf{pref}[a?;a''!] \,\|\, \mathbf{pref}[hb?;hb''!] \,\|\, \mathbf{pref}[hc?;hc''!]$$
$$\|\, \mathbf{pref}[n?;(a''!|hb''!|hc''!)],$$

$$E2 = \mu.tailf.0, \text{ where}$$

$$tailf.R.0 = \mathbf{pref}(a''?;np!;R.1 \mid hb''?;np!;R.2 \mid hc''?;np!;R.3)$$

$$tailf.R.1 = \mathbf{pref}(hb''?;b''!;R.0 \mid hc''?;c''!;R.0)$$

$$tailf.R.2 = \mathbf{pref}(a''?;b''!;R.0 \mid hc''?;np!;R.4)$$

$$tailf.R.3 = \mathbf{pref}(a''?;c''!;R.0 \mid hb''?;np!;R.4)$$

$$tailf.R.4 = \mathbf{pref}(a''?;b''!;R.3), \text{ and}$$

$$E3 = \mathbf{pref}(hb!;[b''?;b!\|hb!]) \,\|\, \mathbf{pref}(hc!;[c''?;c!\|hc!])$$
$$\|\, \mathbf{pref}(n!;[b''?;n! \mid c''?;n! \mid np?;n!]).$$

The connection pattern between the components $E1$, $E2$, and $E3$ is given in Figure 6.2.2.

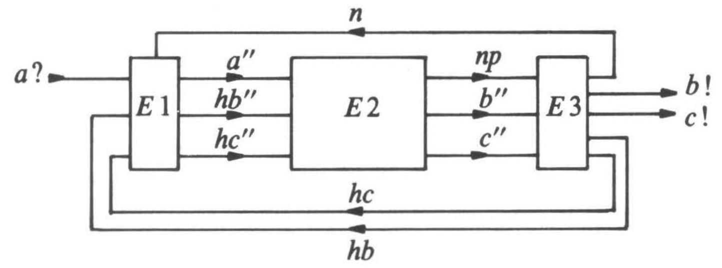


FIGURE 6.2.2. Decomposition of $E0$ into $E1$, $E2$, and $E3$.

□

From the definition of $E1$, $E2$, and $E3$ we derive $E2 \in \mathfrak{L}(G2')$, $E3 \in \mathfrak{L}(GCL')$, $|E1| = \Theta(|E0|)$, and $E3 = \Theta(|E0|)$. For $E2$ we have $E2 = \Theta(n2)$, where $n2$ equals the product of the numbers of states of the sequential commands in E''. Consequently, in general we do not have $E2 = \Theta(|E''|)$.

We prove $E0 \to E1, E2, E3$. First, we demonstrate that $\mathbf{t}(\overline{E0} \| E1 \| E2 \| E3) \upharpoonright \mathbf{a}E0 = \mathbf{t}E0$. We show that any trace $t \in \mathbf{t}E0$ can be expanded with internal symbols into a trace of $\mathbf{t}(\overline{E0} \| E1 \| E2 \| E3)$. Since, by definition of weaving, we also have $\mathbf{t}(\overline{E0} \| E1 \| E2 \| E3) \upharpoonright \mathbf{a}E0 \subseteq \mathbf{t}E0$, we then conclude $\mathbf{t}(\overline{E0} \| E1 \| E2 \| E3) \upharpoonright \mathbf{a}E0 = \mathbf{t}E0$. Define the expansion $f(t)$ for $t \in \mathbf{t}E0$ by

$$f(\epsilon) = \epsilon$$

$$f(tx) = f(t)\ x\ n\ x''\ np \qquad \text{if } x \in \mathbf{i}E0$$

$$f(tx) = f(t)\ hx\ n\ hx''\ x''\ x \qquad \text{if } x \in \mathbf{o}E0.$$

We observe that $f(t)$ is an expansion of t, for any $t \in \mathbf{t}E0$. By definition of $E1$ and $E3$ we observe $f(t) \upharpoonright \mathbf{a}E1 \in \mathbf{t}E1$ and $f(t) \upharpoonright \mathbf{a}E3 \in \mathbf{t}E3$ respectively. Furthermore, we have $f(t) \upharpoonright \mathbf{a}E'' \in \mathbf{t}E''$, and for any prefix r of $f(t)$ we infer by definition of E'' (and E'),

- if $r \upharpoonright \mathbf{a}E''$ ends with hx'', $x \in \mathbf{o}E0$, then $Suc\,(r \upharpoonright \mathbf{a}E'', E'') = \{x''\}$
- if $r \upharpoonright \mathbf{a}E''$ does not end with hx'', $x \in \mathbf{o}E0$, then $Suc\,(r \upharpoonright \mathbf{a}E'', E'') = \varnothing$.

From this we conclude, by definition of $E2$, that $f(t) \upharpoonright \mathbf{a}E2 \in \mathbf{t}E2$. Accordingly, by definition of weaving, $f(t) \in \mathbf{t}(\overline{E0} \| E1 \| E2 \| E3)$.

Second, we observe that the connection $\overline{E0}$, $E1$, $E2$, and $E3$ is closed and free of output interference. Since, by the introduction of the SEQ component, the internal computation performed by $E2$ is purely sequential, it follows that the connection is also free of computation interference, and we derive $E0 \to E1, E2, E3$.

We conclude with a few remarks on the decomposition described in this section. First, we observe that the selection of an output is based on the order in which the inputs and auxiliary symbols are sequenced by the SEQ component. Component $E2$ computes in a deterministic way the next output from the order in which it receives the inputs from the SEQ component. Second, we remark that the internal computation is performed in a purely sequential fashion. We have chosen this approach for reasons of simplicity. Under certain conditions techniques may be applied that yield decompositions with a higher degree of parallelism. For example, it may well be that $E0$ is expressed as a weave $E5 \| E6$, for which $\mathbf{o}E5 \cap \mathbf{o}E6 = \varnothing$. By Corollary 3.1.3.3, $E5 \| E6$ can be decomposed into $E5$ and $E6$. Both components $E5$ and $E6$ can then perform their computations in parallel. More optimization techniques are given in Chapter 7.

6.2.5. *A linear decomposition of* $\mathcal{L}(GSEL)$

(This section may be skipped at first reading.) In the previous section we gave for any component $E0 \in \mathcal{L}(GSEL)$ a decomposition $E0 \rightarrow E1, E2, E3$. The decomposition was not a linear decomposition, since in general $E2 = \Theta(|E0|)$ does not hold. In this section we define components *MASTER* and *SLAVE.i*, $0 \leq i < m$, such that

$$E2 \rightarrow MASTER, (i:0 \leq i < m: SLAVE.i), \text{ where}$$

$$MASTER \in \mathcal{L}(G2') \wedge SLAVE.i \in \mathcal{L}(G2'), \text{ for } 0 \leq i < m, \text{ and}$$

$$|MASTER| + (+i:0 \leq i < m: |SLAVE.i|) = \Theta(|E0|).$$

By the Substitution Theorem and the above decompositions, we can then conclude that there exists a linear decomposition of any component $E0 \in \mathcal{L}(GSEL)$ into components expressed in $\mathcal{L}_2$ and SEQ components. The commands *MASTER* and *SLAVE.i*, $0 \leq i < m$, are constructed from the command E'', which in its turn is constructed from $E0$ (see previous section).

Component $E2$ determines for a current trace $t \in \mathbf{t}E2$ and next input $x \in \mathbf{i}E2$ whether $Suc(tx \restriction \mathbf{a}E'', E'')$ contains an output or not. If it contains an output, the first one is produced, otherwise *np* is produced. We construct a decomposition of $E2$ in which the successor set of outputs with respect to E'' is recorded by a number of *SLAVE* components. First, we explain the idea behind the decomposition by means of an example. Consider the command

$$\begin{aligned} E1'' = \ & \mathbf{pref}[d''?;a''! \mid e''?;(a''!|b''!)] \\ & \| \, \mathbf{pref}[f''?;(a''!|c''!) \mid g''?;a''!] \\ & \| \, \mathbf{pref}[ha''?;a''!] \\ & \| \, \mathbf{pref}[hb''?;b''!] \\ & \| \, \mathbf{pref}[hc''?;c''!]. \end{aligned}$$

From this command we construct Table 6.2.0.

	a''	b''	c''
0	○	○	
1	○		○
2	○		
3		○	
4			○

TABLE 6.2.0.

In general, for a command E'' the corresponding table is constructed as follows. Let E'' be expressed as a weave $(\| k: 0 \leqslant k < N: E.k)$ of sequential commands $E.k \in \mathcal{L}(GSEL)$. For each $y \in \mathbf{o}E''$ and k, $0 \leqslant k < N$, we place a cell at entry (k,y) of the table iff $y \in \mathbf{o}(E.k)$. Each cell can be in one of two states: it is either black or white. Initially all cells are white. The state of the cells is in accordance with the following rules. Let $t \in \mathbf{t}E2$ be the current trace. For each $y \in \mathbf{o}(E.k)$ and k, $0 \leqslant k < N$, we have

P: $\quad y \in Suc(t \restriction \mathbf{a}(E.k), E.k) \equiv$ cell (k,y) is black.

For example, if $E'' = E1''$ and $t = hb''\ np\ ha''\ np\ f''\ np\ d''$, then the cells $(0,a'')$, $(1,a'')$, $(1,c'')$, $(2,a'')$, and $(3,b'')$ are black. If P holds, then the successor set of outputs of E'' is determined by

$$y \in Suc(t \restriction \mathbf{a}E'', E'') \equiv \text{all cells in column } y \text{ are black,}$$

for all $y \in \mathbf{o}E''$. For example, if $E'' = E1''$ and $t = hb''\ np\ ha''\ np\ f''\ np\ d''$, then all cells in column a'' are black. Consequently, $a'' \in Suc\,(t \restriction \mathbf{a}E1'', E1'')$.

The computation of component $E2$ can be expressed as a sequential algorithm that performs operations on a table of cells as defined above. The algorithm has P as an invariant. First, we present the algorithm and then we encode it in a communication protocol between a *MASTER* component and a number of *SLAVE* components. The algorithm is given below. 'Set cell (k,y)' means 'make cell (k,y) black'; resetting a cell means making the cell white.

```
t:=ε ; {P}
do true → x?; k:=r(x); y:=first_k(tx); suc :=false
        ;do ¬suc ∨ y≠nil
          → set cell (k,y)
            ; test if column y is black
            ; if column y is not black → y:=next_k(tx,y)
              [] column y is black → reset cells in column
                                     and adjacent cells
                                     ; suc :=true
              fi
          od
        ;if suc → y!; t:=t x y
         [] ¬suc → np!; t:=t x np
         fi{P}
od,
```

where

$r(x) = k$	if $x \in \mathbf{i}(E.k)$, $0 \leqslant k < N$.
$first_k(tx)$	is the first symbol in $Suc(tx \upharpoonright \mathbf{a}(E.k), E.k)$.
$next_k(tx,y)$	is the next symbol in $Suc(tx \upharpoonright \mathbf{a}(E.k), E.k)$ after y, if y is not the last symbol. Otherwise, it is *nil*.

Because $E'' \in \mathcal{L}(GSEL)$, all inputs are different in E''. Consequently, for each $x \in \mathbf{i}E''$ there is exactly one k such that $r(x)=k$, and $r(x)$ can be determined directly from the syntax of E''. Furthermore, we infer for $0 \leqslant k < N$,

$$tx \in \mathbf{t}E'' \wedge x \in \mathbf{i}(E.k)$$

$$\Rightarrow \{E'' \in \mathcal{L}(GSEL), \text{ calc.}\}$$

$$Suc(tx \upharpoonright \mathbf{a}(E.k), E.k) = Suc(x, E.k).$$

The set $Suc(x,E.k)$ can be determined directly from the syntax of E'' as well. For example, for $E''=E1''$ we have $r(d'')=0$, $r(e'')=0$, $Suc(d'',E.0)=\{a''\}$, and $Suc(e'',E.0)=\{a'',b''\}$.

We make one remark with respect to the resetting of cells. If all cells in column y, say, are black, then a number of cells must be reset such that P can be concluded after output y is produced. The cells that must be reset are not only the cells in column y but also those cells in each row that has a non-empty intersection with column y. For example, after trace $t=hb''\ np\ ha''\ np\ f''\ np\ d''$ all cells in column a'' are black. Before output a'' is produced the cells in column a'' are reset, but also cell $(1,c'')$ must be reset!

The algorithm is encoded in a communication protocol between a *MASTER* component and a number of *SLAVE* components. For each cell (k,y), $0 \leqslant k < N \wedge y \in \mathbf{o}(E.k)$, we have a component *SLAVE.k.y*, which records the state of cell (k,y). The set, test, and reset procedures of the algorithm are encoded in the protocols for communication between the *SLAVE* components. For this purpose, the *SLAVE* components are connected both column-wise and row-wise in a ring. A test or reset procedure is initiated by one *SLAVE* component which starts a signal either in the column-wise ring or in the row-wise ring. The other *SLAVE* components participate in the procedure by propagating the signal according to a specific protocol. Each *SLAVE* component is also connected to the *MASTER* component. The *MASTER* component determines for every receipt of input $x \in \mathbf{i}E2(=\mathbf{i}E'')$ which components *SLAVE.k.y*, with $k=r(x)$ and $y \in \mathbf{o}(E.k)$, must be set and in what order. The answer that component *SLAVE.k.y* returns to the *MASTER* , by means of *msuc.k.y* or *mfail.k.y*, determines whether output y or output *np*, respectively, is produced.

The component *MASTER* is defined by $MASTER = \mu.tailfM.0$, where

$$tailfM.R.0 = \mathbf{pref}(|x : x \in \mathbf{i}E'' : x?; R.first(x))$$

$$tailfM.R.1 = \mathbf{pref}(np!; R.0)$$

and for all pairs (x,y) with $x \in \mathbf{i}E'' \wedge y \in Suc(x, E.r(x))$

$$\begin{aligned} tailfM.R.(x,y) = \mathbf{pref}(&set.r(x).y! \\ &;(msuc.r(x).y?; y!; R.0 \\ &\mid mfail.r(x).y?; R.next(x,y) \\ &)). \end{aligned}$$

Here, for the definition of *tailfM* a collection of states have been labeled with pairs of symbols (x,y), $x \in \mathbf{i}E'' \wedge y \in Suc(x, E.r(x))$, and two states with 0 and 1. The functions $first(x)$ and $next(x,y)$ are defined by

- $first(x) \quad =(x,y)$, if y is the first symbol in $Suc(x, E.r(x))$
- $next(x,y) =(x,z)$, if z is the next symbol in $Suc(x, E.r(x))$ after y
  $=1$, if y is the last symbol in $Suc(x, E.r(x))$.

Below, in Figure 6.2.3 a schematic of a *SLAVE* component is depicted with the terminals with which it is connected to other *SLAVE* components only. (The terminals *msuc* and *mfail* with which it is connected to the *MASTER* component are missing.) The actual names of the terminals for component *SLAVE.k.y* can be derived from the connection pattern. We will not do so here.

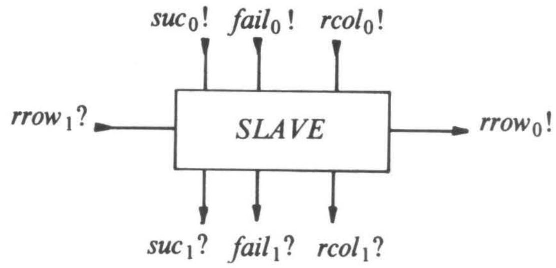


FIGURE 6.2.3. A *SLAVE* component with some of its terminals.

The *SLAVE* component is defined by $SLAVE = \mu.tailfS.0$, where

$$
\begin{aligned}
tailfS.R.0 = \mathbf{pref}(&set?;\{P0\}suc_0!;(fail_1?;\{P1\}mfail!;R.1\\
&\qquad\qquad\qquad\qquad |suc_1?;\{P2\}rrow_0!;rrow_1?\\
&\qquad\qquad\qquad\qquad ;rcol_0!;rcol_1?;\{P3\};msuc!;R.0\\
&\qquad\qquad\qquad\qquad)\\
&|suc_1?;fail_0!;R.0\\
&|fail_1?;fail_0!;R.0\\
&|rrow_1?;rrow_0!;R.0\\
&)\\
tailfS.R.1 = \mathbf{pref}(&suc_1?;suc_0!;R.1\\
&|fail_1?;fail_0!;R.1\\
&|rcol_1?;rrow_0!;rrow_1?;rcol_0!;R.0\\
&|rrow_1?;rrow_0!;R.0\\
&).
\end{aligned}
$$

The following interpretations can be attached to the symbols used above and to $P0$, $P1$, $P2$, and $P3$.

$P0$	initialize test procedure
$P1$	test failed
$P2$	test succeeded, initialize reset procedure
$P3$	completion of reset procedure
set	order of *MASTER* to set cell
mfail	answer to *MASTER* that test for this column failed
msuc	answer to *MASTER* that test and reset procedure were successful
suc	test procedure has been successful so far
fail	test procedure failed

rrow	reset all cells in this row
rcol	reset all rows that have a cell in this column.

Finally, we show

$$E2 \rightarrow MASTER,\ (k,y: 0 \leq k < N \wedge y \in \mathbf{o}(E.k){:}\ SLAVE.k.y). \quad (6.2)$$

First, we observe that the connection is closed and free of output interference. Because the computation is performed sequentially, it follows that the connection is free of computation interference as well. Moreover, since the connection realizes the algorithm described above, we derive that the connection behaves as specified by $\mathbf{t}E2$ at the boundary $\mathbf{a}E2$. Consequently, by definition of decomposition, we conclude (6.2). Furthermore, from the definitions of these components we observe that $MASTER \in \mathcal{L}(G2')$, $SLAVE.k.y \in \mathcal{L}(G2')$ for $0 \leq k < N \wedge y \in \mathbf{o}(E.k)$, and

$$|MASTER| = \mathcal{O}(|E''|) \wedge$$

$$(\mathbf{A}\, k,y: 0 \leq k < N \wedge y \in \mathbf{o}(E.k){:}\ |SLAVE.k.y| = \mathcal{O}(1))$$

$$\Rightarrow \{|E''| = \mathcal{O}(|E0|),\ \text{calc.}\}$$

$$|MASTER| + (+k,y: 0 \leq k < N \wedge y \in \mathbf{o}(E.k){:}\ |SLAVE.k.y|) = \mathcal{O}(|E0|).$$

Finally, we observe that the commands *MASTER* and *SLAVE.k.y*, $0 \leq k < N \wedge y \in \mathbf{o}(E.k)$, are constructed from the syntax of E'', i.e. from $E0$.

6.2.6. Decomposition of SEQ components

Any k-SEQ component, $k > 1$, can be decomposed into the basis $\mathbb{B}$. The decomposition is linear in k and can be described as a syntax-directed translation. The following is a discussion of a decomposition of the k-SEQ component, $k > 1$, specified by

$$(\|i: 0 \leq i < k{:}\ \mathbf{pref}[a.i?;b.i!])$$

$$\|\ \mathbf{pref}[n?;(|i: 0 \leq i < k{:}\ b.i!)].$$

As an example, we consider the decomposition of the 4-SEQ component depicted in Figure 6.2.4. The 4-SEQ component selects one out of at most four pending requests for each occurrence of input n. It then produces a grant for the selected request. In the decomposition, this function is realized in two steps by means of 2-SEQ components. In the first step two independent selections are made: one between the pending requests of inputs $a.0$ and $a.1$, and one between the pending requests of inputs $a.2$ and $a.3$. In the second step a selection is made between the grants of the first step. The selection in the second step determines the final grant and is made for each receipt of input n only. The selections in the first steps are made initially and each time when one of its pending requests has become the final grant.

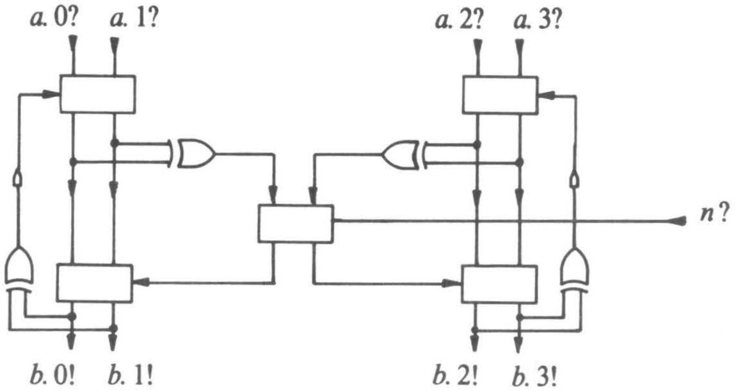


FIGURE 6.2.4. Decomposition of 4-SEQ component.

REMARK. The lower two SEQ components in Figure 6.2.4 may be replaced by CAL components of the form **pref**[a0?||b?;c0! | a1?||b?;c1!]. Notice that there is always at most one pending request for the lower two SEQ components.
□

In general, the selection process performed by a k-SEQ component can be distributed over a binary tree. Each node in this tree consists of 2-SEQ, 2-XOR, and 2-FORK components. For $k=7$, the decomposition of the k-SEQ component is depicted in Figure 6.2.5. The corresponding binary tree is given in Figure 6.2.6.

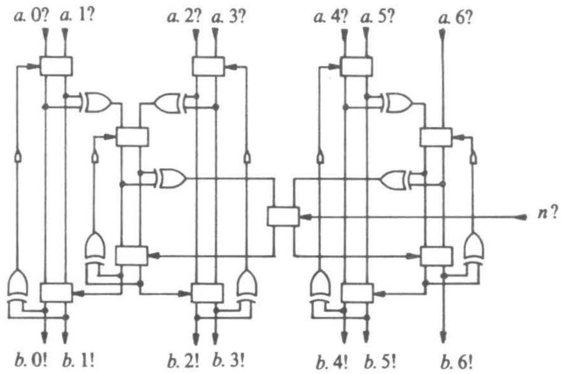


FIGURE 6.2.5. Decomposition of 7-SEQ component.

A pending request becomes a final grant if it is selected once at each node on the path from leaf to root. At the root a selection is made for each receipt of

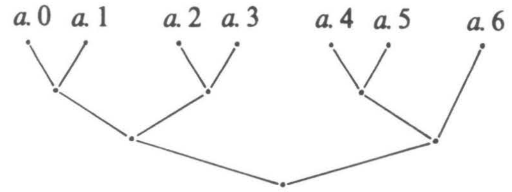


FIGURE 6.2.6. Binary tree corresponding to the distributed selection.

input n only. At any other node a selection is made initially and each time when one of its pending request has become a final grant. The decomposition of a k-SEQ component according to the above procedure consists of less than $2k$ 2-XOR, $2k$ 2-SEQ, and $4k$ 2-FORK components. Consequently, the decomposition is linear in k. Finally, we remark that the decomposition can be described as a syntax-directed translation.

6.3. DECOMPOSITION OF $\mathcal{L}_4$ INTO $\mathcal{L}_3$

In the decomposition step from $\mathcal{L}_4$ to $\mathcal{L}_3$ each component $E0 \in \mathcal{L}(G4') \setminus \mathcal{L}(G3')$ is decomposed into a component $E1 \in \mathcal{L}(G3')$ and a collection of WIRE components. This step is summarized in the following *Expansion Theorem.* Let $f_0.E$ and $Wires(E)$ for a command E be defined by

$f_0.E$ is the command E in which each atomic command $!x?$, with $x \in \mathbf{co}E$, is replaced by $ox!;ix?$. (We assume that $ix \notin \mathbf{a}E$ and $ox \notin \mathbf{a}E$.)

$Wires(E) = (x : x \in \mathbf{co}E : \mathbf{pref}[ox?;ix!])$.

We say that command $f_0.E$ is constructed from E by expansion of each atomic command $!x?$ into $ox!;ix?$. We have

THEOREM 6.3.0. (Expansion Theorem)
If $E \in \mathcal{L}(G4')$, *then* $E \rightarrow f_0.E$, $Wires(E)$ *and* $f_0.E \in \mathcal{L}(G3')$.
□

From the definition of $f_0.E$ and $Wires(E)$ it follows immediately that the decomposition described by the Expansion Theorem is syntax-directed and linear in the length of the command E. Notice also that, since $f_0.E \in \mathcal{L}(G3')$, any projection operator in the command $f_0.E$ may be removed.

EXAMPLE 6.3.1. Let E be defined by

$$E = (\mathbf{pref}[a?;!x?;p!] \,\|\, \mathbf{pref}[!x?;!y?] \,\|\, \mathbf{pref}[!y?;q!;b?]){\restriction}.$$

From Example 4.9.5 we know $E \in \mathcal{L}(G4')$. Consequently, by the Expansion Theorem, we infer

$$E$$

$$\rightarrow \{\text{Expansion Theorem}\}$$

$$\mathbf{pref}[a?;ox!;ix?;p!] \parallel \mathbf{pref}[ox!;ix?;oy!;iy?] \parallel \mathbf{pref}[oy!;iy?;q!;b?]$$

$$, \mathbf{pref}[ox?;ix!] , \mathbf{pref}[oy?;iy!].$$

Moreover, we have $f_0.E \in \mathcal{L}(G3')$.
□

PROOF OF THEOREM 6.3.0. Let $WWires(E)$ be defined by

$$WWires(E) = (\parallel x : x \in \mathbf{co}E : \mathbf{pref}[ox?;ix!]).$$

We prove $E \rightarrow f_0.E,\ WWires(E)$. Since $WWires(E)$ is a weave of WIRE components with disjoint alphabets, the theorem follows by application of Corollary 3.1.3.3 and the Substitution Theorem.

First, by definition of $f_0.E$ and $WWires(E)$ we observe that the connection of $\overline{E}$, $f_0.E$, and $WWires(E)$ is closed and free of output interference. Second, we derive that

$$\mathbf{t}(f_0.E)\restriction\mathbf{a}E = \mathbf{t}E \text{ and } \mathbf{t}(f_0.E)\restriction\mathbf{a}WWires(E) \subseteq \mathbf{t}WWires(E).$$

Consequently, by definition of weaving we deduce

$$\mathbf{t}(\overline{E} \parallel f_0.E \parallel WWires(E)) = \mathbf{t}(f_0.E).$$

Accordingly, we have $\mathbf{t}(\overline{E} \parallel f_0.E \parallel WWires(E))\restriction\mathbf{a}E = \mathbf{t}E$, i.e. the connection behaves as specified at the boundary $\mathbf{a}E$.

Third, we prove that the connection of $\overline{E}$, $f_0.E$, and $WWires(E)$ is free of computation interference. Let $W = \overline{E} \parallel f_0.E \parallel WWires(E)$. We have, by the above, $\mathbf{t}W = \mathbf{t}(f_0.E)$. We observe

(*i*)

$$t \in \mathbf{t}W \wedge x \in \mathbf{o}(f_0.E) \wedge tx\restriction\mathbf{a}(f_0.E) \in \mathbf{t}(f_0.E)$$

$$\Rightarrow \{\mathbf{t}W = \mathbf{t}(f_0.E)\}$$

$$tx \in \mathbf{t}W.$$

(*ii*)

$$t \in \mathbf{t}W \wedge x \in \mathbf{o}WWires(E) \wedge tx\restriction\mathbf{a}WWires(E) \in \mathbf{t}WWires(E)$$

$$\Rightarrow \{\mathbf{t}W = \mathbf{t}(f_0.E)\}$$

$$t \in \mathbf{t}(f_0.E) \wedge x \in \mathbf{o}WWires(E) \wedge tx\restriction\mathbf{a}WWires(E) \in \mathbf{t}WWires(E)$$

$$\Rightarrow \{\text{def. of } f_0.E\}$$

$$tx \in \mathbf{t}(f_0.E)$$
$$\Rightarrow\{\mathbf{t}W = \mathbf{t}(f_0.E)\}$$
$$tx \in \mathbf{t}W.$$

Let $f_0'.E$ be the command E in which each atomic command $!x?$ is replaced by $!ox?;!ix?$, and in which the projection operator has been deleted. We derive from the definition of $f_0'.E$ and the grammar $G4'$

$$\mathbf{t}(f_0'E) = \mathbf{t}(f_0.E) \wedge \mathbf{i}(f_0'.E)=\mathbf{i}E \wedge \mathbf{ext}(f_0'.E)=\mathbf{a}E \text{ and}$$
$$E \in \mathcal{L}(G4') \Rightarrow (f_0'.E){\restriction} \in \mathcal{L}(G4').$$

With these properties for $f_0'.E$ we deduce
(iii)

$$t \in \mathbf{t}W \wedge x \in \mathbf{o}\overline{E} \wedge tx{\restriction}\mathbf{a}\overline{E} \in \mathbf{t}\overline{E}$$
$$\Rightarrow\{\mathbf{t}W = \mathbf{t}(f_0.E), \text{ calc.}\}$$
$$t \in \mathbf{t}(f_0.E) \wedge x \in \mathbf{i}E \wedge tx{\restriction}\mathbf{a}E \in \mathbf{t}E$$
$$\Rightarrow\{f_0.E{\restriction}\mathbf{a}E = E, \text{ calc.}\}$$
$$(\mathbf{E}s :: t \in \mathbf{t}(f_0.E) \wedge x \in \mathbf{i}E \wedge sx \in \mathbf{t}(f_0.E) \wedge s{\restriction}\mathbf{a}E = t{\restriction}\mathbf{a}E)$$
$$\Rightarrow\{\text{def. of } f_0'.E\}$$
$$(\mathbf{E}s :: t \in \mathbf{t}(f_0'.E) \wedge x \in \mathbf{i}(f_0'.E) \wedge sx \in \mathbf{t}(f_0'.E) \wedge$$
$$s{\restriction}\mathbf{ext}(f_0'.E) = t{\restriction}\mathbf{ext}(f_0'.E)\ )$$
$$\Rightarrow\{E \in \mathcal{L}(G4') \Rightarrow (f_0'.E){\restriction} \in \mathcal{L}(G4'),$$
$$\text{i.e. } Disin(f_0'.E) \wedge \mathbf{en}(f_0'.E) = \varnothing, \text{ See Appendix B}\}$$
$$tx \in \mathbf{t}(f_0'.E)$$
$$\Rightarrow\{\mathbf{t}(f_0'.E) = \mathbf{t}W\}$$
$$tx \in \mathbf{t}W.$$

From *(i)*, *(ii)*, and *(iii)* follows that the connection is free of computation interference.

Finally, we remark that the property $E \in \mathcal{L}(G4') \Rightarrow f_0.E \in \mathcal{L}(G3')$ can be proved by means of recursion along the syntax of E using the definitions of $G4'$ and $G3'$.

□

Chapter 7

Special Decomposition Techniques

7.0. Introduction

In the previous chapter we presented a decomposition method which is applicable to components represented in $\mathfrak{L}_4$. For special commands other decomposition techniques, which may yield decompositions with fewer basic components, may be applied as well. The purpose of this chapter is to discuss some of these techniques and to demonstrate their application by means of examples. The style of presentation of these techniques, except for the one presented in the last section, is informal: no proofs are given, no theorems are formulated, and many topics are intended as suggestions for further research.

In the last section of this chapter we show that there exists a decomposition for any regular DI component into components expressed in $\mathfrak{L}_3$. This property is based on a special decomposition technique for decomposing regular DI components that are represented by deterministic commands, i.e. commands in which projection does not occur and that satisfy the LL-1 conditions irrespective of the type of the symbols. We believe, however, that this result is more of theoretical than of any practical interest.

7.1. Merging states and splitting off alternatives

The techniques discussed in this section are called *'merging states'* and *'splitting off alternatives'*. We explain the idea behind these techniques by means of some small examples. Both techniques yield decompositions of the form $E\,0 \rightarrow E\,1$. For this reason they can be used conveniently in combination with

Corollary 3.1.3.2. We demonstrate this in three examples, where decompositions for counter and buffer components are derived.

Consider the following decompositions.

$$\textbf{pref}\{Q0\}[a?;b!;\{Q1\}c?;d!] \rightarrow \textbf{pref}[a?;b! \mid c?;d!]\ ,$$

$$\textbf{pref}[b!;\{Q1\};c?;d!;\{Q2\}a?] \rightarrow \textbf{pref}(b!;[c?;d! \mid a?;b!])\ ,$$

and

$$\textbf{pref}\{Q0\}[a0?;b0!;\{Q1\}c0?;d! \mid a1?;b1!;\{Q2\}c1?;d!]$$
$$\rightarrow \textbf{pref}[a0?;b0! \mid c0?;d! \mid a1?;b1! \mid c1?;d!].$$

We say that the decompositions for these components are constructed by *merging the states* $Q0$ and $Q1$, $Q1$ and $Q2$, and $Q0,Q1$, and $Q2$ respectively. Notice that for each component the inputs that are received in the differently labeled states differ. Therefore, the different states can be distinguished in the decomposition by the difference in inputs.

By means of merging states, the number of states of a sequential command decreases. Thus, also the number of basic components in the final decomposition may decrease. The technique of merging states, however, can not be applied in general. For example, the inputs that are received in the states to be merged must differ. But also the resulting command must be a DI command again. Further study is required to formulate general conditions under which this technique may be applied.

The technique of *splitting off alternatives* is exemplified in the following decompositions.

$$\textbf{pref}[a?;b! \mid c?;d!] \rightarrow \textbf{pref}[a?;b!] \parallel \textbf{pref}[c?;d!],$$

$$\textbf{pref}(b!;[c?;d! \mid a?;b!] \rightarrow \textbf{pref}(b!;[a?;b!]) \parallel \textbf{pref}[c?;d!],$$

and

$$\textbf{pref}[a0?;b0! \mid c0?;d! \mid a1?;b1! \mid c1?;d!].$$
$$\rightarrow \textbf{pref}[a0?;b0!] \parallel \textbf{pref}[a1?;b1!] \parallel \textbf{pref}[c0?;d! \mid c1?;d!].$$

These decompositions suggest a technique for decomposing special commands with alternatives. We have called this technique splitting off alternatives. How this technique can be formulated is also left as a suggestion for future research.

Both techniques can be useful in deriving decompositions for components. This is illustrated in the following examples.

EXAMPLE 7.1.0. We give a derivation for a decomposition of the 3-counter which is specified in Example 4.9.5. First, by means of merging states and splitting off alternatives we derive (cf. above)

$$\mathbf{pref}[a?;ox!;ix?;p!] \;\rightarrow\; \mathbf{pref}[a?;ox!] \parallel \mathbf{pref}[ix?;p!], \qquad (0)$$

$$\mathbf{pref}[ox!;ix?;oy!;iy?] \;\rightarrow\; \mathbf{pref}(ox!;[iy?;ox!]) \parallel \mathbf{pref}[ix?;oy!] \qquad (1)$$

$$\mathbf{pref}[oy!;iy?;q!;b?] \;\rightarrow\; \mathbf{pref}(oy!;[b?;oy!]) \parallel \mathbf{pref}[iy?;q!]. \qquad (2)$$

With these decompositions we infer

$$(\mathbf{pref}[a?;!x?;p!] \parallel \mathbf{pref}[!x?;!y?] \parallel \mathbf{pref}[!y?;q!;b?])\upharpoonright$$

$\rightarrow$ {Expansion Theorem}

$$\mathbf{pref}[a?;ox!;ix?;p!] \parallel \mathbf{pref}[ox!;ix?;oy!;iy?] \parallel \mathbf{pref}[oy!;iy?;q!;b?]$$
$$,\mathbf{pref}[ox?;ix!],\, \mathbf{pref}[oy?;iy!]$$

$\rightarrow$ {(0), (1), and (2), Cor. 3.1.3.2 (3×)}

$$\mathbf{pref}[a?;ox!] \parallel \mathbf{pref}[ix?;p!]$$
$$\parallel \mathbf{pref}(ox!;[iy?;ox!]) \parallel \mathbf{pref}[ix?;oy!]$$
$$\parallel \mathbf{pref}(oy!;[b?;oy!]) \parallel \mathbf{pref}[iy?;q!]$$
$$,\mathbf{pref}[ox?;ix!],\, \mathbf{pref}[oy?;iy!]$$

$\rightarrow$ {rewriting}

$$\mathbf{pref}[ix?;p!]$$
$$\parallel \mathbf{pref}[ox!;iy?] \parallel \mathbf{pref}[a?;ox!]$$
$$\parallel \mathbf{pref}[oy!;b?] \parallel \mathbf{pref}[ix?;oy!]$$
$$\parallel \mathbf{pref}[iy?;q!]$$
$$,\mathbf{pref}[ox?;ix!],\, \mathbf{pref}[oy?;iy!]$$

$\rightarrow$ {Cor. 3.1.3.3}

$$\mathbf{pref}[ix?;p!]$$
$$,\mathbf{pref}[ox!;iy?] \parallel \mathbf{pref}[a?;ox!]$$
$$,\mathbf{pref}[oy!;b?] \parallel \mathbf{pref}[ix?;oy!]$$
$$,\mathbf{pref}[iy!;q!]$$
$$,\mathbf{pref}[ox?;ix!],\, \mathbf{pref}[oy?;iy!]$$

Consequently, from this derivation we conclude that the 3-counter can be decomposed into four WIRE components and two 2-CEL components. The decomposition is depicted in Figure 7.1.0.

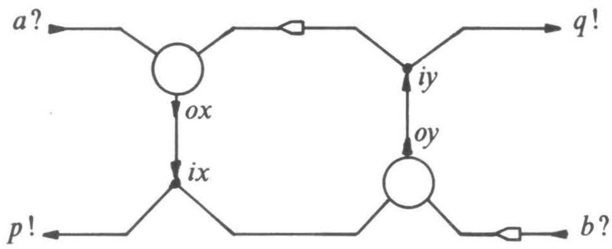


FIGURE 7.1.0. Decomposition of 3-counter.

Because some components have a common input, we may also say that the 3-counter can be decomposed into two 2-FORK and two 2-CEL components. In general, any k-counter, $k>1$, can be decomposed similarly into $k-1$ 2-FORK and $k-1$ 2-CEL components.
□

EXAMPLE 7.1.1. For the four-phase handshake expansion of $count_3(a,b)$ given in Example 4.9.7 we derive analogously to the previous example,

(**pref**[a0?;a1!;a2?;a3!] || **pref**[b0?;b1!;b2?;b3!]

|| **pref**[a1!;a2?;!x?] || **pref**[!x?;!y?] || **pref**[!y?;b1!;b2?])↾

→ {Expansion Theorem}

pref[a0?;a1!;a2?;a3!] || **pref**[b0?;b1!;b2?;b3!]

|| **pref**[a1!;a2?;ox!;ix?] || **pref**[ox!;ix?;oy!;iy?]

|| **pref**[oy!;iy?;b1!;b2?]

,**pref**[ox?;ix!], **pref**[oy?;iy!]

→ {Merging states, splitting off alternatives, Cor. 3.1.3.2}

pref[a0?;a1!] || **pref**[a2?;a3!] || **pref**[b0?;b1!] || **pref**[b2?;b3!]

|| **pref**[a1!;ix?] || **pref**[a2?;ox!] || **pref**[ox!;iy?] || **pref**[ix?;oy!]

|| **pref**[oy!;b2?] || **pref**[iy?;b1!]

,**pref**[ox?;ix!], **pref**[oy?;iy!]

→ {Cor. 3.1.3.3}

pref[a 0?;a 1!] || **pref**[a 1!;ix ?]

,**pref**[b 0?;b 1!] || **pref**[iy ?;b 1!]

,**pref**[a 2?;ox !] || **pref**[ox !;iy ?]

,**pref**[ix ?;oy !] || **pref**[oy !;b 2?]

,**pref**[a 2?;a 3!], **pref**[b 2?;b 3!], **pref**[ox ?;ix !], **pref**[oy ?;iy !].

Consequently, this component can be decomposed into four 2-CEL components and four WIRE components. The decomposition is depicted in Figure 7.1.1.

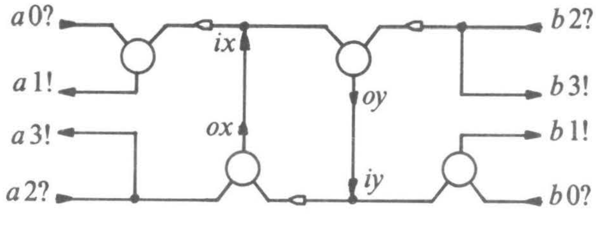


FIGURE 7.1.1. Decomposition of four-phase handshake expansion of $count_3(a,b)$.

□

EXAMPLE 7.1.2. Similar to the previous examples we can derive a decomposition of the 3-place 1-bit buffer which is specified in Example 4.9.6. After a number of steps in which we apply the Expansion Theorem, merging states, splitting off alternatives, Corollary 3.1.3.2, and Corollary 3.1.3.3 we obtain

(**pref**[a 0?;!x 0?;p ! | a 1?;!x 1?;p !]

|| **pref**[!x 0?;!y 0? | !x 1?;!y 1?]

|| **pref**[q ?;(!y 0?;b 0! | !y 1?;b 1!)])↾

→ {applying above mentioned techniques}

pref[a 0?;ox 0!] || **pref**[a 1?;ox 1!]

|| **pref**((ox 0!|ox 1!);[(iy 0?|iy 1?);(ox 0!|ox 1!)])

,**pref**[ix 0?;oy 0!] || **pref**[ix 1?;oy 1!] || **pref**[q ?;(oy 0!|oy 1!)]

,**pref**[ix 0?;p ! | ix 1?;p !]

,**pref**[iy 0?;b 0!], **pref**[iy 1?;b 1!]

,**pref**[ox 0?;ix 0!], **pref**[ox 1?;ix 1!], **pref**[oy 0?;iy 0?], **pref**[oy 1?;iy 1!]

The component in the first two lines of this list of components can be decomposed into a SEQ component and a XOR component. The other components in this list are all familiar components. The complete decomposition is depicted in Figure 7.1.2.

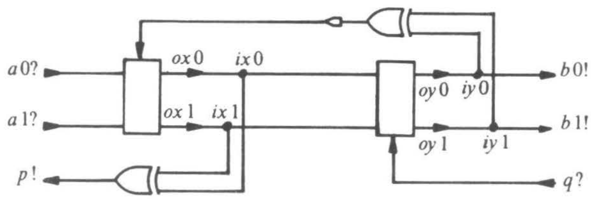


FIGURE 7.1.2. Decomposition of a 3-place 1-bit buffer.

In general, any n-place 1-bit buffer, $n>1$, can be decomposed similarly. Other decompositions for the above buffer can be derived using properties from trace theory. For example, the decomposition where instead of the 2-SEQ components CAL components of the form **pref**[*a* 0?||*b* ?;*c* 0! | *a* 1?||*b* ?;*c* 1!] are used can be derived as well. Finally, we mention that a 3-place n-bit buffer, $n>0$, specified by

$$
\begin{aligned}
&(\| \ i: 0 \leqslant i < n: \mathbf{pref}[a.i.\ 0?;!x.i.\ 0?;p\,! \mid a.i.\ 1?;!x.i.\ 1?;p\,!]) \\
\| &(\| \ i: 0 \leqslant i < n: \mathbf{pref}[!x.i.\ 0?;!y.i.\ 0? \mid !x.i.\ 1?;!y.i.\ 1?]) \\
\| &(\| \ i: 0 \leqslant i < n: \mathbf{pref}[q\,?;(!y.i.\ 0?;b.i.\ 0! \mid !y.i.\ 1?;b.i.\ 1!)]),
\end{aligned}
$$

can be decomposed into 3-place 1-bit buffers. The decomposition for $n=2$ is depicted in Figure 7.1.3, where *Bf* denotes the 3-place 1-bit buffer.

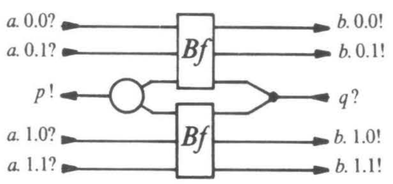


FIGURE 7.1.3. Decomposition of 3-place 2-bit buffer.

□

7.2. Realizing Logic Functions

In this section we show some techniques to realize logic functions of the form $c=f.a$, where c and a are vectors of binary variables and f is a function expressed with logic operations. The techniques are very similar to those applied in switching theory. We will even show that the techniques developed in switching theory can also be applied in the design of delay-insensitive systems. The difference with switching theory lies in the encoding of the data and in the signaling scheme that is applied. For the specification of logic functions by DI commands we apply a two-rail two-cycle signaling scheme in this section (cf. Section 2.3.0).

In Section 2.3.0 the conjunction is specified by a DI command applying a two-rail two-cycle signaling scheme. Negation and disjunction are specified similarly by the DI commands

$$\mathbf{pref}[a0?;c1! \mid a1?;c0!] \text{ and}$$

$$\mathbf{pref}[a0?\|b0?;c0! \mid a1?\|b1?;c1! \mid a0?\|b1?;c1! \mid a1?\|b0?;c1!].$$

respectively. Equivalence can also be specified in this way. In general, any logic function can be specified by a combinational command of the above form. Here, we assume that more than two parallel inputs are allowed in a combinational command. For a function $c=f.a$, where a is a vector of binary variables and c is one binary variable, we obtain a semi-sequential command in which for each set of input values there is one alternative. If f is a vector function $f(i:0\leq i<n)$, we take as the specification for f the weave of the semi-sequential commands for each $f.i$, $0\leq i<n$.

A component specified by a logic function can be decomposed in a natural way into components for the basic logic operations. For example, if the function f is specified by $f.(a,b)=\neg(\neg a\wedge\neg b)$, then the component specified by this function can be decomposed into negation and conjunction components as depicted in Figure 7.2.0.

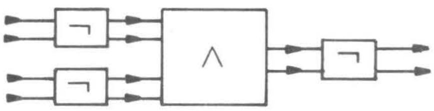


Figure 7.2.0. Decomposition corresponding to $\neg(\neg a\wedge\neg b)$.

We may consider the components for conjunction, disjunction, equivalence, and negation as basic components, but we may also decompose them by one of the techniques discussed in the previous chapter. For example, the negation component is easily decomposed into two WIRE components as shown in Figure 7.2.1.

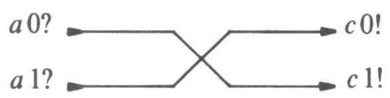


FIGURE 7.2.1. Decomposition of negation component.

Since the expression $\neg(\neg a \wedge \neg b)$ is equivalent to $a \vee b$, it follows that the disjunction component can be realized by the conjunction component when the terminals in each input and output pair are interchanged.

As another example of a decomposition, we consider the comparator and parity function defined by

$$comparator.(a,b) = (\wedge\, i: 0 \leq i < n:\ a.i \equiv b.i) \text{ and}$$

$$parity.a = (\equiv i: 0 \leq i < n:\ a.i),$$

where $a(i: 0 \leq i < n)$ and $b(i: 0 \leq i < n)$ are vectors of binary variables. Each of these functions can be specified by a DI command as sketched above. Decompositions of these components are shown in Figure 7.2.2 for $n = 4$.

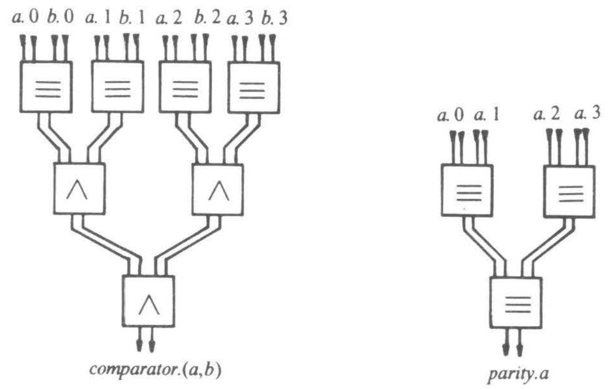


FIGURE 7.2.2. Decompositions for comparator and parity.

In switching theory a circuit that realizes a logic function is called a *combinational circuit.* Often such a circuit is also referred to as a combinational logic block and abbreviated by CL. Analogously, we call a connection of DI components that realizes the DI command corresponding to a logic function a *combinational logic block.*

Logic functions which have a feedback of output values to input values for the next application of f are used to describe a kind of finite state machine. The values that are fed back can be seen as the state information of the finite state machine. Logic functions with feedback of outputs values can be specified by DI commands using tail recursion. For example, the parity function $c = f.a$ for serial inputs a this time can be specified by the command

$\mu.tailf.0$, where

$$tailf.R.0 = \mathbf{pref}(a0?;c0!;R.0 \mid a1?;c1!;R.1)$$

$$tailf.R.1 = \mathbf{pref}(a0?;c1!;R.1 \mid a1?;c0!;R.0).$$

The comparator function $c = f.(a,b)$ with serial inputs a and b is specified by $\mu.tailf.0$, where *tailf* is now defined by

$$tailf.R.0 = \mathbf{pref}(a0?\|b0?;c1!;R.0 \mid a1?\|b1?;c1!;R.0$$
$$\mid a1?\|b0?;c0!;R.1 \mid a0?\|b1?;c0!;R.1)$$

$$tailf.R.1 = \mathbf{pref}(a0?\|b0?;c0!;R.1 \mid a1?\|b1?;c0!;R.1$$
$$\mid a1?\|b0?;c0!;R.1 \mid a0?\|b1?;c0!;R.1).$$

If f is a vector function $f(i:0 \leq i < n)$, we specify f by the weave of the DI commands for each $f.i$, $0 \leq i < n$.

Any logic function with a feedback of state information can be expressed by a logic function without feedback of state information. For example, if f is defined by $c = f.a$, then there exists a logic function g (without feedback) such that $(c,x_{n+1}) = g.(a,x_n)$ where x_n, $n \geq 0$, is a vector of binary variables containing the state information after the n-th application of f. The vector x_0 contains the initial state. Based on this expression for f, the component corresponding to f can be decomposed as depicted in Figure 7.2.3.

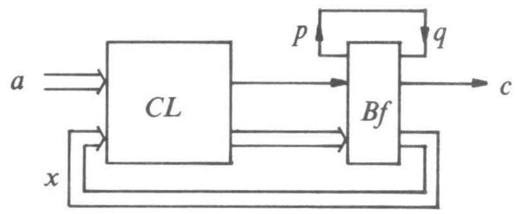


FIGURE 7.2.3. Decomposition for f.

The combinational logic block CL realizes the function g. Component Bf is a 3-place n-bit buffer as specified in Example 7.1.2, where n equals the number of outputs of the function g. The purpose of the buffer is to avoid computation interference in the feedback of state information to the combinational logic block. When all input data is stored in the buffer, output p is produced. This output is fed back to input q upon which the stored data is output. Input data arriving after the occurrence of output p do not interfere with the retrieval of the stored data.

In switching theory a logic function with feedback of state information is realized by a so-called *sequential circuit,* i.e. a combinational circuit and a clocked register. The configuration of Figure 7.2.3 is very similar to such a circuit. Here, the function of the clocked register is performed by the buffer. In the case of clocked systems the presence of new data, i.e. the beginning of a new cycle, is signaled by clock pulses sent to the clocked registers. In the case of delay-insensitive systems the beginning of each new cycle is encoded in the data itself, e.g. by applying a two-rail two-cycle signaling scheme. From the above observations we conclude that techniques used in switching theory for the design of clocked systems can also be applied in the design of delay-insensitive systems.

7.3. Efficient decompositions of $\mathcal{L}(G3')$

The decomposition of a component $E \in \mathcal{L}(G3')$ according to the general methods described in the previous chapter can become rather complicated. In many cases an ad hoc approach may yield a more efficient decomposition. We illustrate this by means of a decomposition for the token-ring interface specified in Section 2.3.2.

First we slightly simplify the command by applying techniques discussed in previous sections. We derive

$$\mathbf{pref}\{Q0\}[a1?;p1!;\{Q1\}a0?;p0!]$$

$$\| \, \mathbf{pref}\{Q2\}[b?;(q! \mid p1!;\{Q3\}a0?;q!)]$$

$$\rightarrow \{\text{Merging states } Q0 \text{ and } Q1, \text{ and } Q2 \text{ and } Q3, \text{ Cor. 3.1.3.2}\}$$

$$\mathbf{pref}[a1?;p1! \mid a0?;p0!]$$

$$\| \, \mathbf{pref}[b?;(q! | p1!) \mid a0?;q!]$$

$$\rightarrow \{\text{Splitting off alternatives, Cor. 3.1.3.2, Cor. 3.1.3.3}\}$$

$$\mathbf{pref}[a0?;p0!]$$

$$,\mathbf{pref}[a1?;p1!] \, \| \, \mathbf{pref}[b?;(q! | p1!) \mid a0?;q!].$$

The last component is specified by a command from $\mathcal{L}(GSEL)$. For the decomposition of this component we consider the decomposition of $E0 = \mathbf{pref}[a1?;p1!] \, \| \, \mathbf{pref}[b?;(q! | p1!)]$ in isolation first. Component $E0$ is decomposed in a similar fashion as discussed in Section 6.2.4. This time, however, we do not introduce auxiliary symbols. The decomposition is given in Figure 7.3.0.

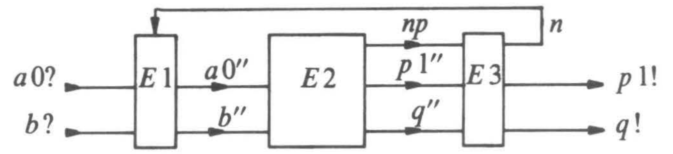


FIGURE 7.3.0. Decomposition of $E0$.

Components $E1$, $E2$, and $E3$ are defined by

$$E1 = \textbf{pref}[a0?;a0''!] \parallel \textbf{pref}[b?;b''!] \parallel \textbf{pref}[n?;(a0''!|b''!)]$$

$$E2 = \textbf{pref}[b''?;q''! \mid a0''?;np!;q''?;p1''!]$$

$$E3 = \textbf{pref}[q''?;q!] \parallel \textbf{pref}[p1''?;p1!]$$

$$\parallel \textbf{pref}(n!;[q''?;n! \mid p1''?;n! \mid np?;n!]).$$

The selection between the outputs $p1$ and q is determined by the order in which the inputs $a0$ and b are sequenced by the SEQ component $E1$. If component $E2$ first receives input b'', output q is produced. If component $E2$ first receives input $a0''$, then, after input b'' is received as well, output $p1$ is produced. We have $E0 \rightarrow E1, E2, E3$.

With the aid of the decomposition for $E0$ we can construct a decomposition for our original command $\textbf{pref}[a1?;p1!] \parallel \textbf{pref}[b?;(q!|p1!) \mid a0?;q!]$. To that end we have to take into account the alternative $a0?;q!$ only for the production of output q. We observe

$$\textbf{pref}[a0?;p0!]$$

$$,\textbf{pref}[a1?;p1!] \parallel \textbf{pref}[b?;(q!|p1!) \mid a0?;q!]$$

$$\rightarrow \{\text{decomposition above, calc., Cor. 3.1.3.3}\}$$

$$E1\ ,\ E2$$

$$,\textbf{pref}[q''?;q! \mid a0?;q!]$$

$$,\textbf{pref}[p1''?;p1!]$$

$$,\textbf{pref}(n!;[q''?;n! \mid p1''?;n! \mid np?;n!])$$

$$,\textbf{pref}[a0?;p0!].$$

Each of these components is either a basic component or can be decomposed by techniques explained in the previous chapters. A complete decomposition of the token-ring interface is shown in Figure 7.3.1.

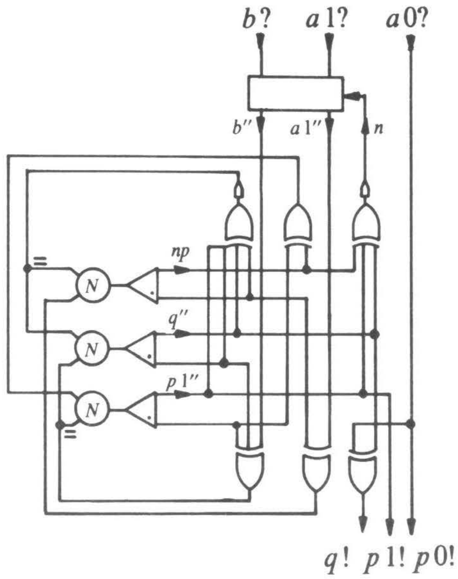


FIGURE 7.3.1. Decomposition of token-ring interface (0).

7.4. EFFICIENT DECOMPOSITIONS USING TOGGLE COMPONENTS

In the general decomposition method TOGGLE components were used in 2-to-4 cycle converters only. For a number special DI commands, TOGGLE components can also be used to obtain a more efficient decomposition. This holds in particular for components expressed in $\mathcal{L}(G1')$. We briefly illustrate how TOGGLE components may optimize decompositions.

Consider the component specified by the sequential command E, where

$$E = \mathbf{pref}[a?;c!;a?;d!;a?;c!].$$

Suppose we had a so-called 3-TOGGLE component specified by $\mathbf{pref}[a?;a1!;a?;a2!;a?;a3!]$. Then we derive

$$E$$

$\rightarrow$ {def. of decomposition}

$$\mathbf{pref}[a?;a1!;a?;a2!;a?;a3!]$$

$$,\mathbf{pref}[a1?;c!;a2?;d!;a3?;c!]$$

$\rightarrow$ {merging states}

$\textbf{pref}[a?;a1!;a?;a2!;a?;a3!]$

$,\textbf{pref}[a1?;c! \mid a2?;d! \mid a3?;c!]$

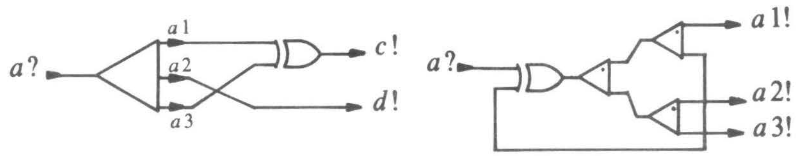
$\rightarrow$ {splitting off alternatives, Cor. 3.1.3.3}

$\textbf{pref}[a?;a1!;a?;a2!;a?;a3!]$

$,\textbf{pref}[a1?;c! \mid a3?;c!]$

$,\textbf{pref}[a2?;d!]$.

The decomposition of component E is depicted in Figure 7.4.0. A 3-TOGGLE component can be decomposed into (2-)TOGGLE components and a 2-XOR component. This decomposition is given in Figure 7.4.0 as well.



Decomposition of E. Decomposition of 3-TOGGLE.

FIGURE 7.4.0.

In command E there are three states in which an input a is received. Input a is also the only input that can be received in those states. By means of a 3-TOGGLE component we are able to make a distinction between those three states. In general, we can use n-TOGGLE components, $n>1$, to distinguish states in which the same input is received. The n-TOGGLE components, $n>1$, can be decomposed into 2-TOGGLE and XOR components similarly to the decomposition of the 3-TOGGLE component.

TOGGLE components can also be used to decompose modulo-N counters, $N>0$, in an efficient way. A modulo-N counter, $N>0$, is specified by

$$\textbf{pref}[(a?;q!)^{N-1};a?;p!],$$

where $E^1=E$ and $E^{n+1}=E^n;E$ for $n>0$. For $N=3$ a decomposition is given in Figure 7.4.1.

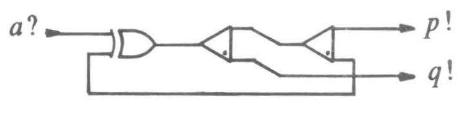


FIGURE 7.4.1. Decomposition of modulo 3-counter.

The modulo-3 counter will be drawn as a square box, as shown in Figure 7.4.2.

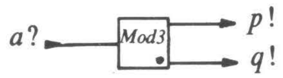


FIGURE 7.4.2. A schematic for the modulo 3-counter.

A decomposition of the modulo-17 counter into 2-XOR, TOGGLE, and modulo-3 counters is given in Figure 7.4.3.

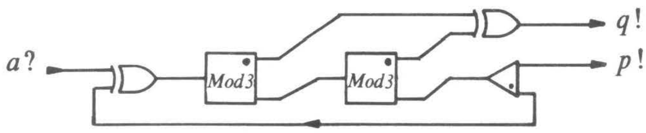


FIGURE 7.4.3. Decomposition of modulo-17 counter.

The decomposition of the modulo-17 counter is based on the calculations $17=18-1$ and $18=3\times3\times2$. In general, any modulo-N counter, $N>0$, can be decomposed into $\Theta(\log N)$ 2-XOR and TOGGLE components. Notice that in the above decompositions there is always at most one transition propagating through the connection. The operation of these counters is very similar to the operation of so-called ripple counters.

7.5. BASIS TRANSFORMATIONS

For the general decomposition method we chose as our basis the set $\mathbb{B}0$. We may wonder whether there exist other bases as well. For example, could the set $\mathbb{B}2$ serve as a basis, where $\mathbb{B}2$ is defined as $\mathbb{B}0$ in which the 2-SEQ component is replaced by the 2-ARB component? This could be an interesting basis, since we may know how to realize a 2-ARB component but we do not know yet how to realize a 2-SEQ component. We indicate that if one set can be used as a basis, so can the other. This is demonstrated by showing that
(*i*) the 2-SEQ component can be decomposed into $\mathbb{B}2$, and
(*ii*) the 2-ARB component can be decomposed into $\mathbb{B}0$.
Consequently, by the Substitution Theorem, we conclude that we can transform decompositions from one basis into the other and vice versa. Since we have shown that $\mathbb{B}0$ can serve as a basis, it follows that $\mathbb{B}2$ can serve as a basis as well.

We present decompositions for (*i*) and (*ii*) by means of schematics. No proofs are given. As specifications for the 2-SEQ and the 2-ARB component we take

$$\mathbf{pref}[a?;p!] \parallel \mathbf{pref}[b?;q!] \parallel \mathbf{pref}[n?;(p!|q!)]$$

and

$$\mathbf{pref}[a1?;p1!;a0?;p0!]$$
$$\parallel \mathbf{pref}[b1?;q1!;b0?;q0!]$$
$$\parallel \mathbf{pref}[p1!;a0? \mid q1!;b0?],$$

respectively. A decomposition of the 2-SEQ component into the 2-ARB component and component E is given in Figure 7.5.0.

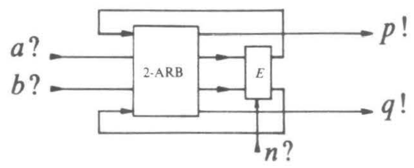


FIGURE 7.5.0. Decomposition of 2-SEQ component into **B**2.

Component E in Figure 7.5.0 is a CAL component specified by $\mathbf{pref}[p1?\|n?;a0! \mid q1?\|n?;b0!]$ and can be decomposed further into the basis $\mathbf{B}0 \setminus \{2\text{-SEQ}\}$ as shown in Section 5.6. A decomposition of the 2-ARB component into **B**0 is given in Figure 7.5.1.

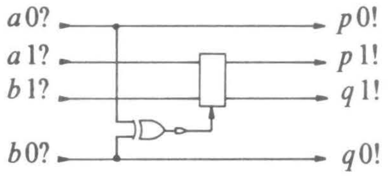


FIGURE 7.5.1. Decomposition of 2-ARB component into **B**0.

With the above transformations between the bases **B**0 and **B**2 it is not difficult to derive a decomposition of a k-ARB component, $k>1$, into the basis **B**2. First we decompose the k-ARB component into a k-SEQ component, a XOR component, and FORK components, similarly to the decomposition shown in Figure 7.5.1. Subsequently, the k-SEQ component is decomposed into the basis **B**0 as described in Section 6.2.6. Finally, each 2-SEQ component in this decomposition is decomposed into **B**2 as depicted in Figure 7.5.0. Thus, we obtain, by the Substitution Theorem, a decomposition of the k-ARB component, $k>1$, into the basis **B**2.

7.6. Decomposition of any regular DI component

With the methods described in Chapters 5 and 6 any regular DI component expressed in $\mathcal{L}_4$ can be decomposed into a finite basis of (DI) components. Since there may be regular DI components that cannot be expressed in $\mathcal{L}_4$, not every regular DI component may be decomposable into a finite basis of components. In this section we indicate that for every regular DI component there exists a decomposition into basic components. To this end, we present a decomposition into components expressed in $\mathcal{L}_3$ for any regular DI component represented by a deterministic command. Here, a deterministic command is defined as a command in which projection does not occur and that satisfies the LL-1 conditions irrespective of the types of the symbols. Because there exists for every regular component a representation in the form of a deterministic command, it follows, with the method discussed in the previous chapters, that for every regular DI component there exists a decomposition into $\mathcal{L}_0$.

We believe that the decomposition step discussed in this section is more of theoretical interest than of any practical interest. It turns out that decompositions started with this step can become rather complicated. Whenever a regular DI component can be expressed in the language $\mathcal{L}_4$, this is to be preferred to expressing the DI component as a command not in $\mathcal{L}_4$. Finding an appropriate command in which a DI component can be expressed is a task of the programmer.

We present for each regular DI component represented by a command $E0$ a decomposition into a component $E1 \in \mathcal{L}(G3')$ and a collection of WIRE components. The theorem on which the decomposition is based is formulated as follows. Let $f_1.E$, $f_2.E$, $WW(E)$, and $Wires(E)$ be defined by

$f_1.E$ — is a command that has the same trace structure as E but is expressed as a weave of deterministic sequential commands.

$f_2.E$ — is the command E in which each atomic command $x?$ and $x!$, for $x \in \mathbf{a}E$, is replaced by $ox!;ix?$.

$$WW(E) = (\|x : x \in \mathbf{i}E : \mathbf{pref}[x?;ox!]) \ \| \ (\|x : x \in \mathbf{o}E : \mathbf{pref}[ix?;x!]).$$

$$Wires(E) = (x : x \in \mathbf{a}E : \mathbf{pref}[ox?;ix!]).$$

We have

Theorem 7.6.0. *If E is a DI command, then*

$$E \rightarrow WW(E) \| f_2.(f_1.E),\ Wires(E)$$

and $WW(E) \| f_2.(f_1.E) \in \mathcal{L}(G3')$.

□

We shall not discuss the details of how to obtain command $f_1.E$. We know that for every command E, i.e. E represents a regular component, there exists a command $f_1.E$. For $f_1.E$ we can take a command corresponding to, for example, the minimal deterministic finite state machine for E. We conjecture that $f_1.E$ can be obtained in a constructive way from command E. Although the construction may be laborious, the essence is that it can be done. Furthermore, we remark that the linearity of the decomposition may be lost in constructing the command $f_1.E$. If $f_1.E$ and E are sequential commands, then $|f_1.E|$ can be exponential in $|E|$ in the worst case [49]. When weaving between sequential commands is allowed there is as yet little known about the relation between $|f_1.E|$ and $|E|$.

In general, we have $E \rightarrow WW(E) \| f_2.E,\ Wires(E)$ for any DI command E, but we do not have $f_2.E \in \mathcal{C}(G3')$. The reason for the construction of $f_1.E$ is to establish $f_2.(f_1.E) \in \mathcal{C}(G3')$. We demonstrate this in the following example.

EXAMPLE 7.6.1. Let command E be defined by

$$E = \mathbf{pref}(a?\|b! \mid a?;b!\|c!).$$

In Example 4.1.1 we inferred that $E \in C4$. Consequently, E is a DI command. Command E, however, is not a deterministic neither a sequential command. In order to obtain a deterministic sequential command for E, we define

$$f_1.E = \mathbf{pref}(b!;a? \mid a?;(b!;c! \mid c!;b!)).$$

Applying Theorem 7.6.0 with this definition for $f_1.E$ we derive

$$E \rightarrow WW(E) \| f_2.(f_1.E),\ Wires(E), \text{ where}$$

$$WW(E) = \mathbf{pref}[a?;oa!] \,\|\, \mathbf{pref}[ib?;b!] \,\|\, \mathbf{pref}[ic?;c!],$$

$$f_2.(f_1.E) = \mathbf{pref}(ob!;ib?;oa!;ia? \mid oa!;ia?;(ob!;ib?;oc!;ic? \mid oc!;ic?;ob!;ib?)), \text{ and}$$

$$Wires(E) = \mathbf{pref}[oa?;ia!], \mathbf{pref}[ob?;ib!], \mathbf{pref}[oc?;ic!].$$

Moreover, we have $WW(E) \| f_2.(f_1.E) \in \mathcal{C}(G3')$. Notice that $f_2.E \notin \mathcal{C}(G3')$.

As a comparison with the decomposition step from $\mathcal{C}_4$ to $\mathcal{C}_3$, we also decompose component E with the Expansion Theorem described in the previous chapter. To this end we rewrite E into a command $E1 \in \mathcal{C}(G4')$. Let $E1$ be defined by

$$E1 = (\mathbf{pref}(a?;!x?;c!) \,\|\, \mathbf{pref}((!y?|!x?);b!))\!\upharpoonright.$$

We observe that E and $E1$ have the same trace structure and $E1 \in \mathcal{C}(G4')$. By the Expansion Theorem we derive

$$E1$$

$$\rightarrow \{\text{Expansion Theorem}\}$$

$$\mathbf{pref}(a?;ox!;ix?;c!) \,\|\, \mathbf{pref}((oy!;iy? \mid ox!;ix?);b!)$$

$, \mathbf{pref}[ox?;ix!], \mathbf{pref}[oy?;iy!]$.

□

PROOF OF THEOREM 7.6.0. Let E be a DI command. Let, furthermore, command $f_1.E$ be denoted by $E1$ and

$$WWires(E1) = (\|x : x \in \mathbf{a}E1 : \mathbf{pref}[ox?;ix!]).$$

We prove $E1 \to WW(E1)\|f_2.E1, WWires(E1)$. Since $E1=E$, $WW(E1)=WW(E)$, and $WWires(E1)=WWires(E)$, we consequently conclude $E \to WW(E)\|f_2.(f_1.E), WWires(E)$. Because $WWires(E)$ is a weave of WIRE components with disjoint alphabets, the theorem follows by application of Corollary 3.1.3.3 and the Substitution Theorem.

First, we observe that the connection $\overline{E1}$, $WW(E1)\|f_2.E1$, $WWires(E1)$ is closed and free of output interference. Second, we prove

$$\mathbf{t}(\overline{E1} \,\|\, WW(E1) \,\|\, f_2.E1 \,\|\, WWires(E1))\lceil \mathbf{a}E1 = \mathbf{t}E1. \qquad (0)$$

To this end, we define

$f_3.E1$ as the command $E1$ in which, for $x \in \mathbf{a}E1$, each atomic command $x?$ is replaced by $x?;ox!;ix?$ and each atomic command $x!$ is replaced by $ox!;ix?;x!$, for $x \in \mathbf{a}E1$.

We derive

$$\mathbf{t}(f_3.E1)\lceil \mathbf{a}E1 = \mathbf{t}E1 \qquad (1)$$

$$\mathbf{t}(f_3.E1)\lceil \mathbf{a}(f_2.E1) = \mathbf{t}(f_2.E1) \qquad (2)$$

$$\mathbf{t}(f_3.E1)\lceil \mathbf{a}WW(E1) \subseteq \mathbf{t}WW(E1) \qquad (3)$$

$$\mathbf{t}(f_3.E1)\lceil \mathbf{a}WWires(E1) \subseteq \mathbf{t}WWires(E1). \qquad (4)$$

Consequently, we deduce

$\mathbf{t}E1$

$= \{(1)\}$

$\mathbf{t}(f_3.E1)\lceil \mathbf{a}E1$

$\subseteq$ {def. of weaving, (1), (2), (3), (4), calc.}

$\mathbf{t}(\overline{E1} \,\|\, WW(E) \,\|\, f_2.E1 \,\|\, WWires(E1))\lceil \mathbf{a}E1$

$\subseteq$ {def. of weaving}

$\mathbf{t}E1$.

From which we conclude that (0) holds.

Third, we prove that the connection $\overline{E1}$, $WW(E1)\|f_2.E1$, $WWires(E)$ is free of computation interference. For this purpose we define

$f_4.E1$ as the command $E1$ in which, for every $x \in \mathbf{a}E1$, each atomic command $x?$ is replaced by $ix?$ and each atomic command $x!$ is replaced by $ox!$.

and

$$W0 = \overline{E1} \parallel WW(E1) \parallel f_2.E1 \parallel WWires(E1)$$
$$W1 = \overline{E1} \parallel WW(E1) \parallel WWires(E1) \parallel f_4.E1$$
$$R.0 = \overline{E1}$$
$$R.1 = WW(E1) \parallel f_2.E1$$
$$R.2 = WWires(E1).$$

In the proof we use the following properties.

The connection $\overline{E1}, WW(E1), WWires(E1), f_4.E1$ (5)
is free of computation interference.

$$(f_2.E1)\upharpoonright \mathbf{a}(f_4.E1) \leqslant f_4.E1. \tag{6}$$

$$t \in \mathbf{t}(f_2.E1) \wedge b \in \mathbf{o}WWires(E1) \wedge \tag{7}$$
$$tb\upharpoonright \mathbf{a}WWires(E1) \in \mathbf{t}WWires(E1)$$
$$\Rightarrow tb \in \mathbf{t}(f_2.E1).$$

$$t \in \mathbf{t}W0 \Rightarrow t \in \mathbf{t}W1. \tag{8}$$

Property (5) follows from the property that E is a DI command and from Theorem 3.2.1.3 with an appropriate renaming. Properties (6) and (7) follow from the definitions of $f_2.E1$, $f_4.E1$, and $WWires(E1)$. Property (8) follows from (6) and the definition of weaving. We infer

$t \in \mathbf{t}W0 \wedge b \in \mathbf{o}(R.1) \wedge tb\upharpoonright \mathbf{a}(R.1) \in \mathbf{t}(R.1)$

$\Rightarrow$ {(8), def. of $WW(E1)$, $f_2.E1$, and $f_4.E1$}

$t \in \mathbf{t}W1 \wedge (b \in \mathbf{o}WW(E1) \vee b \in \mathbf{o}(f_4.E1)) \wedge tb\upharpoonright \mathbf{a}(R.1) \in \mathbf{t}(R.1)$

$\Rightarrow$ {def. of $R.1$, weaving, and (6), calc.}

$(t \in \mathbf{t}W1 \wedge b \in \mathbf{o}WW(E1) \wedge tb\upharpoonright WW(E1) \in \mathbf{t}WW(E1))$

$\vee (t \in \mathbf{t}W1 \wedge b \in \mathbf{o}(f_4.E1) \wedge tb\upharpoonright \mathbf{a}(f_4.E1) \in \mathbf{t}(f_4.E1))$

$\Rightarrow$ {(5)}

$tb \in \mathbf{t}W1.$

Similarly, we prove for $i=0$ and $i=2$ that

$$t\in \mathbf{t}W0 \wedge b\in \mathbf{o}(R.i) \wedge tb\upharpoonright \mathbf{a}(R.i)\in \mathbf{t}(R.i) \Rightarrow tb\in \mathbf{t}W1.$$

Furthermore, we infer for all i, $0\leq i<3$,

$$t\in \mathbf{t}W0 \wedge b\in \mathbf{o}(R.i) \wedge tb\upharpoonright (R.i)\in \mathbf{t}(R.i)$$

$\Rightarrow$ {def. of weaving, for $i=2$ use (7),

for $i=0$ use $b\in \mathbf{o}(R.0) \Rightarrow b\notin \mathbf{a}(f_2.E1)$}

$$tb\upharpoonright \mathbf{a}(f_2.E1)\in \mathbf{t}(f_2.E1).$$

With the last two derivations we deduce for all i, $0\leq i<3$,

$$t\in \mathbf{t}W0 \wedge b\in \mathbf{o}(R.i) \wedge tb\upharpoonright \mathbf{a}(R.i)\in \mathbf{t}(R.i)$$

$\Rightarrow$ {last two derivations}

$$tb\in \mathbf{t}W1 \wedge tb\upharpoonright \mathbf{a}(f_2.E1)\in \mathbf{t}(f_2.E1)$$

$\Rightarrow$ {def. of weaving}

$$tb\in \mathbf{t}W0,$$

i.e. the connection $R.0$, $R.1$, $R.2$ is free of computation interference.

Notice that until now we have not used the property that $E1$, i.e. $f_1.E$, is written as a weave of deterministic sequential commands. Consequently, we conclude that $E \rightarrow WW(E)\|f_2.E, WWires(E)$ holds for any DI command E.

Since $E1$ is a weave of deterministic sequential commands, we infer, by definition of grammar $G3'$, that $f_2.E1\in \mathcal{L}(G3')$. Hence, we also have $WW(E1)\|f_2.E1\in \mathcal{L}(G3')$.

□

Chapter 8

Concluding Remarks

In this thesis we have presented a method for designing delay-insensitive circuits. We have described a decomposition method into a finite basis of components for components that could be expressed in the language $\mathcal{L}_4$. The language $\mathcal{L}_4$ is defined by means of DI grammars, and each command in the language $\mathcal{L}_4$ represents a DI component, i.e. it is intended to specify a circuit that communicates in a delay-insensitive way with its environment. The decomposition can be described as a syntax-directed translation and is, therefore, a constructive method. Thus, we have shown that designing a circuit can be reduced to designing a program.

The program notation for commands has proved to be a convenient medium for expressing parallel computations in a succinct way. In particular, the operations weaving, projection, and tail recursion have been rewarding primitives. Weaving turned out to be a fruitful operation both for the design of parallel programs and for the decomposition of components: in [36, 20] it has been shown that the so-called conjunction-weave rule can be used conveniently for the design of parallel programs; in this thesis we have demonstrated that for the decomposition of components, specified by a weave of commands, the Separation Theorem can be used profitably. By means of projection we can introduce internal symbols, a programming primitive akin to declaring local variables. Tail recursion has been used both for the concise expression of finite state machines and for the description of the decomposition method.

The formalizations of decomposition and delay-insensitivity turned out to be useful as well. The definition of decomposition gave rise to the formulation of other definitions and theorems such as the definitions of DI decomposition and DI component, the Substitution Theorem, the Separation Theorem, the Expansion Theorem, and Theorem 3.2.1.1 on the equivalence of decomposition and DI decomposition. As outlined in Section 7.1, there are more theorems that may be formulated for decomposition, and in the examples of Chapter 7 we indicated that these theorems —together with the material discussed in Chapter 3, 5, and 6— might provide a calculus for finding decompositions.

The DI grammars of Chapter 4 have been valuable both for the recognition of DI commands and for the description of the decomposition method: the recognition of DI commands boils down to checking some simple syntactic rules, and the hierarchy in the decomposition method is defined by means of the hierarchy of the grammars. Moreover, the DI grammars can be used for the derivation of DI commands from non-DI commands as well; in some of the examples in Chapter 4 we derived a delay-insensitive communication protocol from a communication protocol that was not delay-insensitive by means of a DI grammar. Furthermore, since these grammars include the operations weaving, projection, and tail recursion, they offer great freedom in programming.

The proofs for Chapter 4, however, are rather long and tedious. Apparently, the difficulty of designing delay-insensitive circuits is concentrated in the recognition of DI components. Knowing whether a component is a DI component is, however, an important property, since we can resort to decomposition instead of DI decomposition if we know that all components involved are DI components. Because of all the theorems that apply to decomposition, this reduction is indeed a simplification.

The hierarchical decomposition method is straightforward, apart from the decomposition of CAL components into the basis $\mathbb{B}0$ and the decomposition of components expressed in $\mathcal{L}(GSEL)$ into SEQ components and components expressible in $\mathcal{L}_2$. Moreover, the results of many decompositions can be depicted in a regular schematic forming a connection of a CT, XOR, and CEL plane. Consequently, since these schematics also represent delay-insensitive connections of basic elements, it should not be too difficult to design layouts for these circuits.

We gave two decompositions for CAL components: one decomposition into the basis $\mathbb{B}0$ and one decomposition into the basis $\mathbb{B}1$. The decomposition we described into the basis $\mathbb{B}0$ is conjectured to be correct. The decomposition of CAL components into the basis $\mathbb{B}1$ does not have to be a DI decomposition. In order to ensure proper operation in a realization of this decomposition delay assumptions must be met. The delay assumptions are simple timing constraints and are met by isochronic forks. Notice that this is the only place in which we needed to introduce delay assumptions for proper operation. The decomposition of CAL components is carried out in one of the last steps only of the hierarchical decomposition method.

The complexity of the decomposition method has been kept under control as well: although the simple one-hot assignment has been applied, the decomposition can be linear in the length of a command, i.e. the total number of basic elements in the resulting decomposition is proportional to the length of the command. We believe that there exist many more techniques that may provide even more efficient decompositions; in Chapter 7 a few of them have been suggested. In particular, the decomposition of components expressed in

$\mathcal{L}(GSEL)$, which may yield a rather large number of components, can be optimized if special decomposition techniques are applied.

In Chapter 5 we have introduced, more or less arbitrarily, the basis **B**1. We did not motivate our choice there, but postponed the argument of that choice until now. The choice for the basis **B**1 has been based on the following criteria: first, the basic elements must be realizable in an area small enough such that the necessary internal timing constraints can easily be identified and met, and, second, the specification of each basic element must be in good harmony with the decomposition method. Most basic elements arose naturally from the decomposition strategy applied, except for the 2-SEQ component. The SEQ component is a kind of arbitration component, just as the ARB component is. Initially, we chose the 2-ARB component as the primitive arbitration component, since various realizations for this component exist. We have found, however, that the 2-SEQ component fitted better in the formal decomposition method. For example, the decomposition of the k-ARB component into 2-ARB components is more complicated than the decomposition of the k-SEQ component into 2-SEQ components. This was one of the reasons we chose the 2-SEQ component as a basic component. Once one finite basis is found, we can change from one basis to the other, by means of simple basis transformations as shown in Section 7.5.

A topic we have not discussed yet is the possibility of *deadlock* and *livelock* in decompositions. How deadlock and livelock can occur is illustrated by the following example. Consider the components $E0\upharpoonright$ and $E1\upharpoonright$, where

$$E0 = \mathbf{pref}(a?;!x?;b!) \parallel \mathbf{pref}(!x?\,|\,!y?) \quad \text{and}$$

$$E1 = \mathbf{pref}(a?;!x?;b!) \parallel \mathbf{pref}[!x?\,|\,!y?].$$

For $E0\upharpoonright$ we observe that

$$Suc(ay, E0) = \varnothing \quad \wedge \quad Suc(ay\upharpoonright \mathbf{ext}E0, E0\upharpoonright) = \{b\}.$$

In other words, after the receipt of input a, component $E0\upharpoonright$ can produce output b. But, if in the decomposition of component $E0\upharpoonright$ the internal action y is selected, output b will never be produced. We say that there is danger of *deadlock*. Component $E1\upharpoonright$ has a similar behavior. For this component we observe that for all $n>0$, $ay^n xb \in \mathbf{t}E1$. Accordingly, an unbounded number of internal y-actions can occur before an output b is produced. This phenomenon is called *livelock*.

Because the decomposition is syntax-directed, deadlock and livelock can also occur in the decompositions of $E0\upharpoonright$ and $E1\upharpoonright$. Absence of deadlock and livelock are required in a decomposition of a component. The translation method of Chapters 5 and 6 is defined such that absence of deadlock and livelock in a decomposition of component $E\upharpoonright$ only depends on the trace structures of E and $E\upharpoonright$ (and not on the syntax of E). In [20] the phenomena of deadlock and livelock are defined within the formalism of trace theory. There, the notion of *transparence* is introduced by means of which conditions can be formulated such that absence of deadlock and livelock is guaranteed. We believe that this property has also nice prospects for formulating conditions for the absence of livelock and deadlock in decompositions of components.

The general decomposition method we have described is restricted to components expressible in $\mathcal{L}_4$. This means that the programmer must try to represent a component in the language $\mathcal{L}_4$. In Sections 2.3 and 4.9 we have illustrated this programming issue by means of a number of characteristic examples. In Section 2.3.3 we showed how a command not in $\mathcal{L}_4$ may be rewritten into a command that does satisfy the restrictions imposed by the language $\mathcal{L}_4$. Although for many components a program can be found in the language $\mathcal{L}_4$, for some components this may well be impossible.

We discuss some of the restrictions of the language $\mathcal{L}_4$ and for what reasons they have been introduced. Consider the following commands which are not contained in $\mathcal{L}_4$:

$$\mathbf{pref}[a0?\|a1?;b!;a0?\|a1?;c!] \text{ and}$$

$$\mathbf{pref}(a?;[b!;c?;d!;a?;(b!;c?)\|(d!;c?)]).$$

The reasons for the absence of these commands in the language $\mathcal{L}_4$ is that in grammar $G4'$ a command is not allowed to have two or more parallel inputs or outputs, or that a command of type *<pfcom>* contains a weave. Although it is not too difficult to find decompositions for the components expressed by the above commands, we believe that in general it can become rather complicated to find decompositions for components expressed by commands that do not exhibit the above mentioned restrictions.

The restrictions imposed by the language $\mathcal{L}_4$ evolved from the following two requirements. First, we wanted to define a grammar for which any command generated by this grammar was a DI command. In the development of this grammar we have been led by the theorems that could be formulated on the DI property (cf. Appendix B). Allowing weaving in commands of type *<pfcom>* renders a condition, viz.

$$\mathbf{pref}(E0\|E1) = \mathbf{pref}E0 \,\|\, \mathbf{pref}E1$$

for commands $E0$ and $E1$ of type *<pfcom>*, which was too difficult to check mechanically. Second, every component represented by a command in $\mathcal{L}_4$ should be decomposable in a constructive and simple way. Allowing two or more parallel inputs or outputs, or weaving in commands of type *<pfcom>*, yielded too big problems for the description of a simple decomposition method that was generally applicable.

Because of the direct relation between the syntax of a command and its decomposition, complexity measures with respect to time and area of decompositions can be studied by examining just the syntactic structure of the command. For the same reason, we may choose a specific decomposition of a component by choosing a specific command for that component. Therefore, and because of the requirement to express a component in $\mathcal{L}_4$, it is important that programming techniques are developed with which commands can be derived conveniently from a specification.

In this thesis we first formalized, by means of a few basic definitions, the fundamental issues relevant to the design of delay-insensitive circuits. Subsequently, we built from these definitions a formal framework which provided the means to reason about such circuits, and even to derive such circuits, by only considering the corresponding programs. Thus, the abstraction offered by the formalism has enabled us to design circuits by thinking about the programs entirely *in abstracto*.

Appendix A

For the proof of Theorem 4.1.0 we recapitulate some definitions and introduce some new notations.

The rules for class C4 are defined as follows. In these rules, R denotes a directed trace structure with $\mathbf{int}R = \varnothing$; s and t denote arbitrary traces; a, b, and c denote arbitrary symbols; p.c.n.e. stands for prefix-closed and non-empty.

rule 1: R is p.c.n.e. and $\mathbf{i}R \cap \mathbf{o}R = \varnothing$.

rule 2: $s\,a\,a \notin \mathbf{t}R$.

rule 3: If a and b are of the same type, then $s\,a\,b\,t \in \mathbf{t}R \;\equiv\; s\,b\,a\,t \in \mathbf{t}R$.

rule 4'': If a and b are of different type and a and c are of the same type, then $s\,a\,b\,t\,c \in \mathbf{t}R \;\wedge\; s\,b\,a\,t \in \mathbf{t}R \;\Rightarrow\; s\,b\,a\,t\,c \in \mathbf{t}R$.

rule 5''': If a and b are of distinct type, then $s\,a \in \mathbf{t}R \;\wedge\; s\,b \in \mathbf{t}R \Rightarrow\; s\,a\,b \in \mathbf{t}R$.

By definition, we have $R \in C4$ iff R satisfies rule 1, 2, 3, 4'', and 5'''.

For the definition of DI component we introduce the following notations.

enc(R) is defined as the trace structure R in which each occurrence of a symbol $b \in \mathbf{o}R$ is replaced by *ob* and each occurrence of a symbol $b \in \mathbf{i}R$ is replaced by *ib*. (We assume that the characters o and i do not occur in $\mathbf{a}R$.)

$$Wire(b) = \mathbf{pref}[ob?;ib!] \qquad \text{for each } b \in \mathbf{a}R$$
$$Wires(R) = (b : b \in \mathbf{a}R : Wire(b))$$
$$WWires(R) = (\| b : b \in \mathbf{a}R : Wire(b))$$
$$Con(R) = enc(\overline{R}), Wires(R), enc(R)$$
$$W = enc(\overline{R}) \,\|\, WWires(R) \,\|\, enc(R).$$

($Con(R)$ stands for the connection of components $enc(\overline{R})$, $Wires(R)$, and $enc(R)$.) By definition, component R is DI iff $Con(R)$ is closed, free of interference, and $\mathbf{t}W \upharpoonright \mathbf{a}\,enc(\overline{R}) = \mathbf{t}\,enc(\overline{R})$.

We slightly simplify the definition of DI component first. Define the function $f(r)$ for traces $r \in \mathbf{t}R$ by

$$f(\epsilon) = \epsilon \text{ and } f(r\,b) = f(r)\,ob\,ib \text{ for } r\,b \in \mathbf{t}R.$$

For all $\overline{r \in \mathbf{t}R}$ we have $f(r) \in \mathbf{t}W$. Consequently, we infer $\mathbf{t}W \upharpoonright \mathbf{a}\,enc(\overline{R}) = \mathbf{t}\,enc(\overline{R})$. Moreover, if R is a component, i.e. $\mathbf{i}R \cap \mathbf{o}R = \varnothing$, then $Con(R)$ is closed and free of output interference. From these two observations follows

component R is DI

$\equiv$ R is a component and $Con(R)$ is free of computation interference.

We prove in the following

R is a component and $Con(R)$ is free of computation interference.

$\Rightarrow$ R satisfies rule 1, 2, 3, 4″, and 5‴. (1)

and

R satisfies rule 1, 2, 3, 4″, and 5‴

$\Rightarrow$ R is a component and $Con(R)$ is free of computation interference. (2)

From (1) and (2) we then conclude Theorem 4.1.0.

Proof of (1). Let R be a component and $Con(R)$ be free of computation interference. Since R is a component, rule 1 is obviously satisfied.

For rule 2 we observe

$s\,a\,a \in \mathbf{t}R \;\vee\; s\,a\,a \notin \mathbf{t}R$

$\Rightarrow$ {def. of f and W, R is p.c., calc.}

$(f(s) \in \mathbf{t}W \;\wedge\; s\,a\,a \in \mathbf{t}R) \;\vee\; s\,a\,a \notin \mathbf{t}R$

$\Rightarrow$\{$Con(R)$ is free of comp. interference, R is p.c., def. of $enc(R)$\}

$f(s)\,oa\;oa \in \mathbf{t}W \;\vee\; s\,a\,a \notin \mathbf{t}R$

$\Rightarrow$\{def. of $Wire(a)$, def. of weaving\}

$s\,a\,a \notin \mathbf{t}R.$

We prove that R satisfies rule 3 by induction to the length of t.
Base. For a and b of the same type we observe

$s\,a\,b \in \mathbf{t}R$

$\Rightarrow$\{def. of f and W, R is p.c.\}

$f(s) \in \mathbf{t}W \;\wedge\; s\,a\,b \in \mathbf{t}R$

$\Rightarrow$\{$Con(R)$ is free comp. of interference,

R is p.c, a and b of the same type\}

$f(s)\,oa\;ob \in \mathbf{t}W$

$\Rightarrow$\{def. of $Wire(a)$ and $Wire(b)$, $Con(R)$ is free of comp. interference\}

$f(s)\,oa\;ob\;ib\;ia \in \mathbf{t}W$

$\Rightarrow$\{def. of f and W, a and b of the same type\}

$s\,b\,a \in \mathbf{t}R.$

Step. For $\{a,b\} \subseteq \mathbf{i}R$ and $c \in \mathbf{o}R$ we observe

$s\,a\,b\,t\,c \in \mathbf{t}R$

$\Rightarrow$\{induction hypothesis for t, R is p.c\}

$s\,a\,b\,t\,c \in \mathbf{t}R \;\wedge s\,b\,a\,t \in \mathbf{t}R$

$\Rightarrow$\{def. of f and W, $\{a,b\} \subseteq \mathbf{i}R$, $c \in \mathbf{o}R$\}

$f(s)\,ob\;oa\;ia\;ib\;f(t)\,oc \in \mathbf{t}W$

$\Rightarrow$\{def. of $Wire(c)$, $Con(R)$ is free of comp. interference\}

$f(s)\,ob\;oa\;ia\;ib\;f(t)\,oc\;ic \in \mathbf{t}W$

$\Rightarrow$\{def. of f and W, $\{a,b\} \subseteq \mathbf{i}R$, $c \in \mathbf{o}R$\}

$s\,b\,a\,t\,c \in \mathbf{t}R.$

By a similar reasoning, or using symmetry, we prove the induction step also for $\{a,b\} \subseteq \mathbf{i}R \;\wedge\; c \in \mathbf{i}R$, $\{a,b\} \subseteq \mathbf{o}R \;\wedge\; c \in \mathbf{o}R$, and $\{a,b\} \subseteq \mathbf{o}R \;\wedge\; c \in \mathbf{i}R$.

R satisfies rule 4″ can also be proved with a similar reasoning as for the proof of the induction step above.

For rule 5‴ we observe for symbols a and b of different type

$s\,a \in \mathbf{t}R \;\wedge\; s\,b \in \mathbf{t}R$

$\Rightarrow$ {def. of f and W, R is p.c}

$f(s) \in \mathbf{t}W \ \wedge \ s\,a \in \mathbf{t}R \ \wedge \ s\,b \in \mathbf{t}R$

$\Rightarrow$ {$Con(R)$ is free of comp. interference, a and b are of different type}

$f(s)\, oa\ ob \in \mathbf{t}W$

$\Rightarrow$ {def. of $Wire(b)$, $Con(R)$ is free of comp. interference}

$f(s)\, oa\ ob\ ib \in \mathbf{t}W$

$\Rightarrow$ {def. of f and W, a and b are of different type}

$s\,a\,b \in \mathbf{t}R$.

□

For the proof of (2) we introduce a few notations. Let the relation $\mapsto$ on $(\mathbf{a}W)^* \times (\mathbf{a}W)^*$ be defined by

$$r\,x\,y\,t \mapsto r\,y\,x\,t$$

$$\equiv (r\,x\,y\,t \in \mathbf{t}W \Rightarrow r\,y\,x\,t \in \mathbf{t}\mathbf{W})$$

$$\wedge\ \neg(x \in \mathbf{i}\,enc(R)\ \wedge\ y \in \mathbf{o}\,enc(R))$$

$$\wedge\ \neg(x \in \mathbf{i}\,enc(\overline{R})\ \wedge\ y \in \mathbf{o}\,enc(\overline{R}))$$

$$\wedge\ \neg(\{x,y\} \subseteq \mathbf{a}\,Wire(b) \text{ for some } b \in \mathbf{a}R).$$

Let $\mapsto^*$ denote the transitive closure of $\mapsto$. In words, if $s \mapsto^* t$ holds, then s can be transformed into t by repeatedly interchanging two contiguous symbols in such a way that

- membership of $\mathbf{t}W$ is maintained,
- an output symbol is not shifted to the left over an input symbol of the same component, and
- symbols of a WIRE component are not interchanged.

The notations Out, $s\mathbf{L}a$, and $out(s)$, for $s \in \mathbf{t}W$ and $a \in \mathbf{a}R$, are defined by

$$Out\ =\ \{oa \mid a \in \mathbf{a}R\}$$

$$s\mathbf{L}a \equiv s\,ia \upharpoonright \mathbf{a}Wire(a) \in \mathbf{t}\,Wire(a)$$

$$out(s)\ =\ \{t \mid \text{trace } t \text{ corresponds to a permutation of } \{oa \mid s\mathbf{L}a\}\}$$

Notice that if R satisfies rule 1, then W is prefix-closed. In the following proofs the hint 'R satisfies rule 1' often refers to W is prefix-closed.

PROOF OF (2). Let R satisfy rule 1, 2, 3, 4″, and 5‴. From rule 1 follows that R is a component. We prove

(*i*) $s \in \mathbf{t}W \ \wedge\ s\mathbf{L}a \ \Rightarrow\ s\,ia \in \mathbf{t}W$ for $a \in \mathbf{a}R$, and

(*ii*) $s \in \mathbf{t}W \ \wedge\ oa \in \mathbf{o}\,enc(R) \ \wedge\ s\,oa \upharpoonright \mathbf{a}\,enc(R) \in \mathbf{t}\,enc(R) \ \Rightarrow\ s\,oa \in \mathbf{t}W$.

For reasons of symmetry, we conclude that (*ii*) also holds if R is replaced by $\overline{R}$. Consequently, $Con(R)$ is free of computation interference, if (*i*) and (*ii*) hold.

We observe for (*i*)

$$s \in \mathbf{t}W \ \wedge \ s\mathbf{L}a$$

$\Rightarrow$ {Lemma A.0, R satisfies rule 1, 3, 4″, and 5‴}

$$s \in \mathbf{t}W \ \wedge \ s\mathbf{L}a \ \wedge \ (\mathbf{E}r : r \in \mathbf{t}R : (\mathbf{A}t : t \in out(s) : s \mapsto f(r)t))$$

$\Rightarrow$ {def. of $out(s)$, calc.}

$$s \in \mathbf{t}W \ \wedge \ (\mathbf{E}r,t : r \in \mathbf{t}R \ \wedge \ oa\, t \in out(s) : s \mapsto^{*} f(r)\, oa\, t)$$

$\Rightarrow$ {Lemma A.1, R satisfies rule 1 and 5‴}

$$s \in \mathbf{t}W \ \wedge$$
$$(\mathbf{E}r,t : r \in \mathbf{t}R \ \wedge \ oa\, t \in out(s) : s \mapsto^{*} f(r)\, oa\, t \ \wedge \ f(r)\, oa\, t\, ia \in \mathbf{t}W)$$

$\Rightarrow$ {Lemma A.4, R satisfies rule 1 and 4″, calc.}

$$s\, ia \in \mathbf{t}W.$$

For (*ii*) we observe

$$s \in \mathbf{t}W \ \wedge \ oa \in \mathbf{o}\, enc(R) \ \wedge \ s\, oa \upharpoonright \mathbf{a}\, enc(R) \in \mathbf{t}\, enc(R) \ \wedge \ s\, oa \notin \mathbf{t}W$$

$\Rightarrow$ {def. of W, $Wire(a)$, and weaving}

$$s \in \mathbf{t}W \ \wedge \ oa \in \mathbf{o}\, enc(R) \ \wedge \ s\, oa \upharpoonright \mathbf{a}\, enc(R) \in \mathbf{t}\, enc(R) \ \wedge \ s\mathbf{L}a$$

$\Rightarrow$ {Lemma A.0, R satisfies rule 1, 3, 4″, and 5‴, $s\mathbf{L}a \Rightarrow oa \in Out$}

$$s \in \mathbf{t}W \ \wedge \ oa \in \mathbf{o}\, enc(R) \ \wedge \ s\, oa \upharpoonright \mathbf{a}\, enc(R) \in \mathbf{t}\, enc(R)$$
$$\wedge \ (\mathbf{E}r,t : r \in \mathbf{t}R \ \wedge \ t\, oa \in Out^{*} : s \mapsto^{*} f(r)\, t\, oa)$$

$\Rightarrow$ {Lemma A.3, R satisfies rule 1 and 4″}

$$(\mathbf{E}r,t :: f(r)\, t\, oa\, oa \upharpoonright \mathbf{a}\, enc(R) \in \mathbf{t}\, enc(R)) \wedge oa \in \mathbf{o}\, enc(R)$$

$\Rightarrow$ {R satisfies rule 2}

false.

From this derivation we conclude that (*ii*) holds.
□

LEMMA A.0. *Let R satisfy rule 1, 3, 4″, and 5‴. We have*

$$(\mathbf{A}s : s \in \mathbf{t}W : (\mathbf{E}r : r \in \mathbf{t}R : (\mathbf{A}t : t \in out(s) : s \mapsto^{*} f(r)\, t))).$$

PROOF. By induction to the length of s.

Base. If $s = \epsilon$, then we have $r = \epsilon$ and $t = \epsilon$.

Step. We observe for $a \in \mathbf{a}R$

$s\,oa \in \mathbf{t}W$

$\Rightarrow$ {R satisfies rule 1, 3, 4″, and 5‴, induction hypothesis for s}

$s\,oa \in \mathbf{t}W \ \wedge\ (\mathbf{E}r : r \in \mathbf{t}R : (\mathbf{A}t : t \in out(s) : s \mapsto^* f(r)\,t))$

$\Rightarrow$ {Lemma A.2, R satsfies rule 1 and 4″}

$s\,oa \in \mathbf{t}W \ \wedge\ (\mathbf{E}r : r \in \mathbf{t}R : (\mathbf{A}t : t \in out(s) : s\,oa \mapsto^* f(r)\,t\,oa))$

$\Rightarrow$ {Lemma A.6, R satisfies rule 3, def. of $out(s)$ and $\rightarrow^*$, calc.}

$(\mathbf{E}r : r \in \mathbf{t}R : (\mathbf{A}t : t \in out(s\,oa) : s\,oa \mapsto^* f(r)\,t))$

and

$s\,ia \in \mathbf{t}W$

$\Rightarrow$ {induction hypothesis for s, R satisfies rule 1, 3, 4″, and 5‴}

$s\,ia \in \mathbf{t}W \ \wedge\ (\mathbf{E}r : r \in \mathbf{t}R : (\mathbf{A}t : t \in out(s) : s \rightarrow^* f(r)\,t))$

$\Rightarrow$ {$s\,ia \in \mathbf{t}W \Rightarrow s\mathbf{L}a$, def. of $out(s)$}

$s\,ia \in \mathbf{t}W \ \wedge\ (\mathbf{E}r : r \in \mathbf{t}R : (\mathbf{A}t : oa\,t \in out(s) : s \mapsto^* f(r)\,oa\,t))$

$\Rightarrow$ {Lemma A.1, R satisfies rule 1 and 5‴}

$(\mathbf{E}r : r \in \mathbf{t}R : (\mathbf{A}t : oa\,t \in out(s) : s \mapsto^* f(r)\,oa\,t \ \wedge\ f(r)\,oa\,t\,ia \in \mathbf{t}W$
$\wedge\ f(r)\,oa\,t\,ia \mapsto^* f(r)\,oa\,ia\,t))$

$\Rightarrow$ {Lemma A.4, R satisfies rule 1 and 4″}

$(\mathbf{E}r : r \in \mathbf{t}R : (\mathbf{A}t : oa\,t \in out(s) : s\,ia \mapsto^* f(r)\,oa\,t\,ia$
$\wedge\ f(r)\,oa\,t\,ia \mapsto^* f(r)\,oa\,ia\,t))$

$\Rightarrow$ {def. of $\mapsto^*$, f, and $out(s)$}

$(\mathbf{E}r : r \in \mathbf{t}R : (\mathbf{A}t : t \in out(s\,ia) : s\,ia \mapsto^* f(r\,a)\,t))$.

□

LEMMA A.1. *Let R satisfy rule 1 and 5‴ and $r \in \mathbf{t}R$. We have*

$s \in \mathbf{t}W \ \wedge\ s \mapsto^* f(r)\,oa\,t \ \wedge\ oa\,t \in out(s)$

$\Rightarrow f(r)\,oa\,t\,ia \in \mathbf{t}W \ \wedge\ f(r)\,oa\,t\,ia \mapsto^* f(r)\,oa\,ia\,t.$

PROOF. We observe

$s \in \mathbf{t}W \ \wedge\ s \mapsto^* f(r)\,oa\,t \ \wedge\ oa\,t \in out(s)$

$\Rightarrow$ {def. of $\mapsto^*$ and $out(s)$}

$f(r)\,oa\,t \in \mathbf{t}W \ \wedge\ t \in (Out \setminus \{oa\})^*$

$\Rightarrow$ {R satisfies rule 1, def. of W and f}

$$f(r)\,oa\,t \in \mathbf{t}W \;\wedge\; r\,a \in \mathbf{t}R \;\wedge\; t \in (Out)\setminus\{oa\})^*$$

$$\Rightarrow \{\text{Lemma A.5, } R \text{ satisfies rule 1 and } 5'''\}$$

$$(\mathbf{A}u,v: t = u\,v: f(r)\,oa\,u\,ia\,v \in \mathbf{t}W).$$

Since $t \in (Out \setminus \{oa\})^*$ and by definition of $\mapsto^*$, we consequently derive

$$f(r)\,oa\,t\,ia \mapsto^* f(r)\,oa\,ia\,t \;\wedge\; f(r)\,oa\,t\,ia \in \mathbf{t}W.$$

□

LEMMA A.2. *Let R satisfy rule 1 and 4″. We have for* $a \in \mathbf{a}R$

$$(s \mapsto^* s') \Rightarrow (s\,oa \mapsto^* s'\,oa).$$

PROOF. We observe

$$s \mapsto s' \;\wedge\; s\,oa \in \mathbf{t}W$$

$$\Rightarrow \{\text{def. of weaving, } W, \mapsto, \text{ and rule } 4''\}$$

$$s'\,oa \in \mathbf{t}W.$$

Consequently, we infer $(s \mapsto s') \Rightarrow (s\,oa \mapsto s'\,oa)$. Taking the transitive closure of $\mapsto$ yields the lemma.
□

With a similar reasoning as in the last proof we obtain

LEMMA A.3. *Let R satisfy rule 1 and 4″ and* $a \in \mathbf{a}R$. *If* $s \mapsto^* s'$, *then* $s\,oa \upharpoonright \mathbf{a}\,enc(R) \in \mathbf{t}\,enc(R) \;\Rightarrow\; s'\,oa \upharpoonright \mathbf{a}\,enc(R) \in \mathbf{t}\,enc(R)$.
□

LEMMA A.4. *Let R satisfy rule 1 and 4″ and* $a \in \mathbf{a}R$. *If* $s \mapsto^* s' \;\wedge\; s'\,ia \in \mathbf{t}W$, *then* $s\,ia \mapsto^* s'\,ia \;\wedge\; (s \in \mathbf{t}W \Rightarrow s\,ia \in \mathbf{t}W)$.

PROOF. We observe

$$s \mapsto s' \;\wedge\; s \in \mathbf{t}W \;\wedge\; s'\,ia \in \mathbf{t}W$$

$$\Rightarrow \{\text{def. of weaving, } W, \mapsto, \text{ and rule } 4''\}$$

$$s\,ia \in \mathbf{t}W.$$

By definition of $\mapsto$, we also have

$$s \mapsto s' \;\wedge\; s'\,ia \in \mathbf{t}W \;\Rightarrow\; s\,ia \mapsto s'\,ia.$$

Taking the transitive closure of $\mapsto$ and using that R is prefix-closed (by rule 1), we obtain the lemma.
□

LEMMA A.5. *Let R satisfy rule 1 and 5‴ and* $a \in \mathbf{a}R$. *We have*

$$r\,a \in \mathbf{t}R \ \wedge\ f(r)\,oa\,t \in \mathbf{t}W \ \wedge\ t \in (Out \setminus \{oa\})^*$$
$$\Rightarrow (\mathbf{A}u,v : t = u\,v : f(r)\,oa\,u\,ia\,v \in \mathbf{t}W).$$

PROOF. By induction to the length of t.

Base. If $t = \epsilon$, then $u\,v = \epsilon$. The lemma follows from the definition of f and W.

Step. We observe for $b \in \mathbf{a}R$, $b \neq a$,

$$r\,a \in \mathbf{t}R \ \wedge\ f(r)\,oa\,s\,ob \in \mathbf{t}W \ \wedge\ s\,ob \in (Out \setminus \{oa\})^*$$
$$\Rightarrow \{\text{induction hypothesis for } s,\ R \text{ satisfies rule 1 and } 5'''\}$$
$$f(r)\,oa\,s\,ob \in \mathbf{t}W \ \wedge\ (\mathbf{A}u,v : s = u\,v : f(r)\,oa\,u\,ia\,v \in \mathbf{t}W)$$
$$\Rightarrow \{\text{def. of } W \text{ and weaving, } R \text{ satisfies rule } 5'''\}$$
$$f(r)\,oa\,s\,ob\,ia \in \mathbf{t}W \ \wedge\ f(r)\,oa\,s\,ia\,ob \in \mathbf{t}W$$
$$\wedge\ (\mathbf{A}u,v : s = u\,v : f(r)\,oa\,u\,ia\,v \in \mathbf{t}W)$$
$$\Rightarrow \{\text{calc.}\}$$
$$(\mathbf{A}u,v : s\,ob = u\,v : f(r)\,oa\,u\,ia\,v \in \mathbf{t}W).$$

□

LEMMA A.6. *Let R satisfy rule 3 and* $t \in Out^*$. *For all permutations* t' *of* t *we have* $s\,t \mapsto s\,t'$.

PROOF. Any permutation t' of t can be obtained by successively swapping two contiguous symbols. Using rule 3 and the definition of weaving yields the lemma.

□

Appendix B

B.0. Introduction

In this appendix we present the proof of Theorem 4.7.0, i.e.

$$E \in \mathcal{L}(G4) \Rightarrow E \text{ is DI.}$$

The proof is based on a long series of theorems, some of which have tedious proofs.

Before we present these theorems and their proofs, we introduce some new definitions. First, we generalize the classes $C3$ and $C4$, which are given in Section 4.1, to the classes $GC3$ and $GC4$ respectively. The classes $GC3$ and $GC4$ pertain to directed trace structures for which the set of internal symbols does not have to be empty (as opposed to the classes $C3$ and $C4$). In the following rules, R denotes a directed trace structure, s and t denote traces, and a, b, c, x and y denote symbols. Furthermore, the alphabets $\mathbf{in}R$ and $\mathbf{out}R$ are defined by

$$\mathbf{in}R = \mathbf{i}R \cup \mathbf{en}R \quad \text{and} \quad \mathbf{out}R = \mathbf{o}R \cup \mathbf{co}R.$$

(In words, the atomic commands of symbols from $\mathbf{in}R$ start with an input mark and the atomic commands of symbols from $\mathbf{out}R$ start with an output mark.). The abbreviation p.c.n.e. stands for prefix-closed and non-empty

rule $g1$: R is p.c.n.e. and the alphabets of R of distinct type are pairwise disjoint.

rule $g2$: For any $a \in \mathbf{ext}R$, $saa \notin \mathbf{t}R$.

rule $g3$: For all symbols x and y with

$$(x \in (\mathbf{i}R \cup \mathbf{co}R) \wedge y \in (\mathbf{i}R \cup \mathbf{en}R))$$
$$\vee\ (x \in (\mathbf{o}R \cup \mathbf{en}R) \wedge y \in (\mathbf{o}R \cup \mathbf{co}R))$$

we have $sxyt \in \mathbf{t}R \Rightarrow syxt \in \mathbf{t}R$.

rule $g4'$: For symbols a and b of different type, $\{a,b\} \subseteq \mathbf{ext}R$,

$sabt \in \mathbf{t}R \wedge sb \in \mathbf{t}R \Rightarrow sbat \in \mathbf{t}R.$

rule $g4''$: For symbols a and b of different type, $\{a,b\} \subseteq \mathbf{ext}R$ and $\{a,c\} \subseteq \mathbf{out}R \vee \{a,c\} \subseteq \mathbf{in}R$

$sabtc \in \mathbf{t}R \wedge sbat \in \mathbf{t}R \Rightarrow sbatc \in \mathbf{t}R.$

rule $g5'''$: For symbols a and b of different type, $\{a,b\} \subseteq \mathbf{ext}R$,

$sa \in \mathbf{t}R \wedge sb \in \mathbf{t}R \Rightarrow sba \in \mathbf{t}R.$

The classes $GC3$ and $GC4$ are defined analogously to the classes $C3$ and $C4$: $GC3$ is the class of all trace structures satisfying rules $g1$, $g2$, $g3$, $g4'$, and $g5'''$; class $GC4$ is the class of all trace structures satisfying rules $g1$, $g2$, $g3$, $g4''$, and $g5'''$.

Notice that from the definitions of these rules follows

$$R \in GC4 \wedge \mathbf{int}R = \varnothing \Rightarrow R \in C4,$$

and similarly for $GC3$ and $C3$. Consequently, $GC4$ and $GC3$ are indeed generalizations of $C3$ and $C4$.

At this point we would like to emphasize that rule $g3$ is used extensively in the remainder of this appendix; in many theorems and lemmas it occurs as a condition. This is not surprising if we realize that most theorems with respect to delay-insensitivity boil down to the shifting of symbols in a trace, and rule $g3$ is a convenient rule for this purpose.

The sets $\mathbf{first0}R$ and $\mathbf{first1}R$ for a directed trace structure R are defined as follows. Let $\mathbf{hd}R \subseteq \mathbf{out}R$, where $\mathbf{hd}R = \{b \mid (\mathbf{E}t :: bt \in \mathbf{t}R)\}$. If $\mathbf{t}R = \{\epsilon\}$, then $\mathbf{first0}R = \{\varnothing\}$. Otherwise

$$\mathbf{first0}R = \{set(t) \mid t \in (\mathbf{o}R)^* \wedge t \in \mathbf{t}\,\mathbf{pref}R \wedge t \neq \epsilon \wedge (Suc(t,R) \setminus \mathbf{o}R \neq \varnothing \vee Suc(t,R) = \varnothing)\} \cup \{\{b\} \mid b \in \mathbf{co}R \wedge b \in \mathbf{t}\,\mathbf{pref}R\}.$$

Here, $set(t)$ denotes the set of symbols occurring in trace t. For $\mathbf{hd}R \subseteq \mathbf{out}R$, the set $\mathbf{first1}R$ is defined by

$$\mathbf{first1}R = \{set(t\restriction \mathbf{ext}R) \mid t \in (\mathbf{out}R)^* \wedge t \in \mathbf{t\,pref}R \wedge (Suc(t,R) \setminus \mathbf{out}R \neq \varnothing \vee Suc(t,R) = \varnothing)\}.$$

If $\mathbf{hd}R \subseteq \mathbf{in}R$, then $\mathbf{first0}R$ and $\mathbf{first1}R$ are defined analogously with $\mathbf{o}R$, $\mathbf{co}R$, and $\mathbf{out}R$ replaced by $\mathbf{i}R$, $\mathbf{en}R$ and $\mathbf{in}R$ respectively. Otherwise, $\mathbf{first0}R$ and $\mathbf{first1}R$ are not defined. For example, we have for

$$E = a! \| b!;c? \mid !d?;e!$$

$$\mathbf{hd}E \subseteq \mathbf{out}E$$

$$\mathbf{first0}E = \{\{a,b\}, \{d\}\}$$

$$\mathbf{first1}E = \{\{a,b\}, \{e\}\}.$$

Finally, we define the predicates $Disin(R)$ and $Disout(R)$ for a directed trace structure R by

$$\begin{aligned} & Disin(R) \\ \equiv & (\mathbf{A}u,v,b : u \in \mathbf{t\,pref}R \wedge vb \in \mathbf{t\,pref}R \wedge b \in \mathbf{in}R \\ & \quad \wedge u\restriction(\mathbf{ext}R \cup \mathbf{en}R) = v\restriction(\mathbf{ext}R \cup \mathbf{en}R) \\ & \quad : ub \in \mathbf{t\,pref}R \\ &) \end{aligned}$$

and $Disout(R)$ is defined analogously with $\mathbf{in}R$ and $\mathbf{en}R$ replaced by $\mathbf{out}R$ and $\mathbf{co}R$ respectively. The predicates $Disin(R)$ and $Disout(R)$ concern the possible disabling of symbols of a certain type. For example, suppose that $Disin(R)$ holds and that two traces from R are equivalent with respect to the external symbols and internal symbols of the environment. If one of these traces can be extended with a symbol from $\mathbf{in}R$, then the other can be extended as well with this symbol, i.e. the symbol is not disabled for the other trace. The predicate $Disfree(R)$ is defined by

$$Disfree(R) \equiv Disin(R) \wedge Disout(R).$$

We prove the following theorems for commands E derivable in $G4$.

THEOREM B.0. $E \in \langle dicom \rangle \Rightarrow E \in C4.$
□

THEOREM B.1. $E \in \langle pccom \rangle \Rightarrow P1(E) \wedge P2(E).$
□

THEOREM B.2. $E \in \langle pfcom \rangle \Rightarrow P0(E) \wedge P2(E) \wedge P3(E) \wedge P4(E).$
□

The predicates $P0(E)$ through $P4(E)$ are defined by

$$\begin{array}{lll}
P0(E) & \equiv & \mathbf{pref}E \in GC3 \wedge \mathit{Disfree}(E) \\
& & \wedge\ E \text{ and } E\restriction\mathbf{ext}E \text{ are prefix-free} \\
& & \wedge\ \mathbf{t}E \neq \{\epsilon\} \wedge (\mathbf{hd}E \subseteq \mathbf{in}E \vee \mathbf{hd}E \subseteq \mathbf{out}E) \\
P1(E) & \equiv & E \in GC4 \wedge \mathit{Disfree}(E) \\
P2(E) & \equiv & I(E)=\mathbf{i}E \wedge O(E)=\mathbf{o}E \wedge EN(E)=\mathbf{en}E \wedge CO(E)=\mathbf{co}E \\
P3(E) & \equiv & FIRST(E)=\mathbf{first0}E \wedge FIRSTEXT(E)=\mathbf{first1}E \\
P4(E) & \equiv & HD(E)=in \quad \text{iff } \mathbf{t}R\neq\{\epsilon\} \wedge \mathbf{hd}R \subseteq \mathbf{in}R \\
& & \qquad\quad =out \quad \text{iff } \mathbf{t}R\neq\{\epsilon\} \wedge \mathbf{hd}R \subseteq \mathbf{out}R \\
& & \qquad\quad =empty \quad \text{iff } \mathbf{t}R=\{\epsilon\} \\
& & \qquad\quad =mixed \quad \text{otherwise,} \\
& & \wedge\ TL(E)=in \quad \text{iff } \mathbf{t}R\neq\{\epsilon\} \wedge \mathbf{tl}R \subseteq (\mathbf{i}R \cup \mathbf{co}R) \\
& & \qquad\quad =out \quad \text{iff } \mathbf{t}R\neq\{\epsilon\} \wedge \mathbf{tl}R \subseteq (\mathbf{o}R \cup \mathbf{en}R) \\
& & \qquad\quad =empty \quad \text{iff } \mathbf{t}R=\{\epsilon\} \\
& & \qquad\quad =mixed \quad \text{otherwise.}
\end{array}$$

The predicates $P0$, $P3$, and $P4$ are defined on commands of type <*pfcom*> in $G4$, i.e. $E\in$<*pfcom*>. The predicate $P1$ is defined on commands of type <*pccom*> and the predicate $P2$ is defined on commands of type <*pfcom*> and <*pccom*>.

The remainder of this appendix is organized as follows. First, in Section B.1 we list the theorems on which Theorems B.0, B.1, and B.2 are based. Subsequently, we present the proofs of Theorems B.0, B.1, and B.2 in Section B.2. In Section B.3 the proofs of Theorems B.3 through B.5 are presented, in Section B.4 the proofs of Theorems B.6 through B.9 are presented, and in Section B.5 the proofs for Theorems B.10 through B.16 are given. Lemmas used in a proof directly follow that proof.

B.1. The Theorems

The Theorems B.0, B.1 and B.3 are based on the theorems listed below. In order to formulate the conditions for these theorems, we introduce some notation first.

The predicate $\mathit{Alfcond}(R,S)$ is defined by

$$\mathit{Alfcond}(R,S) \equiv \text{aphabets of distinct type of } R \text{ and } S \text{ are pairwise disjoint.}$$

The generalization of this condition to trace structures $R.j$, $0 \leqslant j < n$, is

$$Alfcond(j: 0 \leqslant j < n: R.j) \equiv (\mathbf{A}i,j: 0 \leqslant i,j < n \wedge i \neq j: Alfcond(R.i,\ R.j)).$$

The predicate $Seqcond(R,S)$ is defined by

$$Seqcond(R,S) \equiv (\mathbf{tl}R \subseteq \mathbf{i}R \cup \mathbf{co}R \ \wedge\ \mathbf{hd}S \subseteq \mathbf{o}S \cup \mathbf{co}S)$$
$$\vee\ (\mathbf{tl}R \subseteq \mathbf{o}R \cup \mathbf{en}R \ \wedge\ \mathbf{hd}S \subseteq \mathbf{i}S \cup \mathbf{en}S).$$

In order to define $Altcond0(R,S)$ and $Altcond1(R,S)$ we first define the predicates $fprop0(R)$ and $fprop1(S)$. The predicate $fprop0(R)$ is defined by

$$fprop0(R) \equiv (\mathbf{A}t: t \in \mathbf{t\,pref}R \wedge t \neq \epsilon: (\mathbf{E}s: s \in \mathbf{t\,pref}R \wedge set(s) \in \mathbf{first0}R$$
$$: t \preccurlyeq s \vee s \preccurlyeq t \quad)).$$

The notation $t \preccurlyeq s$ denotes that t is a prefix of s. The property $fprop0(R)$ expresses that for any non-empty trace t in $\mathbf{pref}R$ there exists a trace s in $\mathbf{pref}R$ with $set(s) \in \mathbf{first0}R$ such that $t \preccurlyeq s$ or $s \preccurlyeq t$. For example, we have

$$fprop0([a?;b?]) \equiv false \quad \text{but} \quad fprop0([a?;b!]) \equiv true.$$

Notice that $\mathbf{first0}[a?;b?] = \varnothing$ and $\mathbf{first0}[a?;b!] = \{\{a\}\}$. For a non-empty trace structure R, with $fprop0(R)$, we have $\mathbf{first0}R \neq \varnothing$. The predicate $fprop1(R)$ is defined analogously to $fprop1(R)$ with $\mathbf{pref}R$ and $\mathbf{first0}R$ replaced by $\mathbf{pref}R \restriction \mathbf{ext}R$ and $\mathbf{first1}R$ respectively. In the remainder we are interested in trace structures R for which $fprop0(R)$ and $fprop1(R)$ hold.

The predicate $Altcond0(R,S)$ is, consequently, defined by

$$Altcond0(R,S)$$
$$\equiv ((\mathbf{hd}R \subseteq \mathbf{out}R \wedge \mathbf{hd}S \subseteq \mathbf{out}S) \vee (\mathbf{hd}R \subseteq \mathbf{in}S \wedge \mathbf{hd}S \subseteq \mathbf{in}S))$$
$$\wedge\ fprop0(R) \wedge fprop0(S) \wedge llcond0(R,S),$$

where

$$llcond0(R,S)$$
$$\equiv (\mathbf{first0}R \subseteq \{\varnothing\} \wedge \mathbf{first0}S \subseteq \{\varnothing\})$$
$$\vee\ (\mathbf{A}A,B: A \in \mathbf{first0}R \wedge B \in \mathbf{first0}S: \neg(A \subseteq B) \wedge \neg(B \subseteq A)).$$

$Altcond1(R,S)$ is defined analogously with $fprop0$, $llcond0$, and $\mathbf{first0}$ replaced by $fprop1$, $llcond1$, and $\mathbf{first1}$ respectively.

The generalizations of the predicates $Altcond0(R,S)$ and $Altcond1(R,S)$ to a collection of trace structures $R.j$, $0 \leqslant j < n$, is done as follows. For $n = 1$ we have $Altcond0(j: 0 \leqslant j < n: R.j) \equiv true$. Otherwise,

$$Altcond0(j: 0 \leqslant j < n: R.j)$$
$$\equiv ((\mathbf{A}j: 0 \leqslant j < n: \mathbf{hd}(R.j) \subseteq \mathbf{out}(R.j)) \vee (\mathbf{A}j: 0 \leqslant j < n: \mathbf{hd}(R.j) \subseteq \mathbf{in}(R.j)))$$
$$\wedge\ (\mathbf{A}j: 0 \leqslant j < n: fprop0(R.j)) \wedge llcond0(j: 0 \leqslant j < n: R.j)$$

where

$$\begin{aligned}&llcond0(j:0\leqslant j<n:R.j)\\ \equiv\ &(\mathbf{A}j:0\leqslant j<n:\ \mathbf{first0}(R.j)\subseteq\{\varnothing\})\\ &\vee\ (\mathbf{A}i,j,A,B:0\leqslant i,j<n\ \wedge\ i\neq j\ \wedge\ A\in\mathbf{first0}(R.i)\ \wedge\ B\in\mathbf{first0}(R.j)\\ &\qquad :\neg(A\subseteq B)).\end{aligned}$$

$Altcond1(j:0\leqslant j<n:R.j)$ is defined analogously to $Altcond0(j:0\leqslant j<n:R.j)$ with **first0**, *fprop* 0, and *llcond* 0 replaced by **first1**, *fprop* 1, and *llcond* 1 respectively.

Finally, we define the predicate *Tailcond*(*tailf*) for a tail function *tailf* defined by array $S(i,j:0\leqslant i,j<n)$ of trace structures. Let the tail function *tailf* be defined by

$$tailf.R.i\ =\ \mathbf{pref}(|j:0\leqslant j<n:S.i.j;R.j), \tag{B0}$$

for $0\leqslant i<n$. (As usual, *tailf* is defined on $\mathcal{T}^n(A0,A1,A2,A3)$, where $A0, A1, A2$, and $A3$ are defined in Section 2.1). The condition *Tailcond* (*tailf*) for the function *tailf* is defined for the array $S(i,j:0\leqslant i,j<n)$ of trace structures by

$$Tailcond(tailf)\ \equiv\ (0)\wedge(1)\wedge(2)\wedge(3)\wedge(4)\wedge(5)\wedge(6),\ \text{where}$$

$$\begin{aligned}
(0)\ &\equiv(\mathbf{A}i:0\leqslant i<n:(\mathbf{E}j:0\leqslant j<n:\mathbf{t}(S.i.j)\neq\varnothing))\\
(1)\ &\equiv(\mathbf{A}i,j:0\leqslant j<n\ \wedge\ i\neq j:\ \mathbf{t}(S.i.j)\neq\{\epsilon\})\\
&\quad\wedge\ (\mathbf{A}i:0\leqslant i<n:\mathbf{t}(S.i.i)=\{\epsilon\}\Rightarrow(\mathbf{A}j:0\leqslant j<n\ \wedge\ i\neq j:\mathbf{t}(Si.j)=\varnothing))\\
(2)\ &\equiv Alfcond(i,j:0\leqslant i,j<n:S.i.j)\\
(3)\ &\equiv(\mathbf{A}i,j:0\leqslant i,j<n\ \wedge\ \mathbf{t}(S.i.j)\neq\varnothing\\
&\qquad :\mathbf{pref}(S.i.j)\in GC3\ \wedge\ Disfree(S.i.j)\\
&\qquad \wedge\ S.i.j\text{ and }S.i.j\upharpoonright\mathbf{ext}(S.i.j)\text{ are prefix-free}\\
&\quad)\\
(4)\ &\equiv(\mathbf{A}i,j,k:0\leqslant i,j,k<n\ \wedge\ \mathbf{t}(S.i.j)\neq\varnothing\ \wedge\ \mathbf{t}(S.j.k)\neq\varnothing:Seqcond(S.i.j,S.j.k))\\
(5)\ &\equiv(\mathbf{A}i:0\leqslant i<n:\ Altcond0(j:0\leqslant j<n\ \wedge\ \mathbf{t}(S.i.j)\neq\varnothing:S.i.j)\\
&\qquad\qquad \wedge\ Altcond1(j:0\leqslant j<n\ \wedge\ \mathbf{t}(S.i.j)\neq\varnothing:S.i.j)\\
&\quad)\\
(6)\ &\equiv(\mathbf{A}i,j:0\leqslant i,j<n:\mathbf{ext}(S.i.j)=\varnothing)\ \vee\\
&\quad(\mathbf{A}i,j:0\leqslant i,j<n\ \wedge\ \mathbf{t}(S.i.j)\neq\varnothing\ \wedge\ \mathbf{t}(S.i.j)\neq\{\epsilon\}:\ \mathbf{t}(S.i.j)\upharpoonright\mathbf{ext}(S.i.j)\neq\{\epsilon\}).
\end{aligned}$$

With the above predicates, the theorems are formulated as follows.

THEOREM B.3. $R \in GC4 \wedge Disfree(R) \Rightarrow R\restriction \mathbf{ext}R \in C4$.
□

THEOREM B.4. *Let tailf be defined by (B0). If Tailcond(tailf) holds, then* $\mu.tailf.0$ *exists and for all i,* $0 \leqslant i < n$,

$$\mu.tailf.i \in GC3 \wedge Disfree(\mu.tailf.i).$$

□

THEOREM B.5.

0. $R \in GC4 \wedge S \in GC4 \wedge Alfcond(R,S) \Rightarrow R \| S \in GC4$.

1. *If R and S are prefix-closed, then*

$$Disfree(R) \wedge Disfree(S) \wedge Alfcond(R,S) \Rightarrow Disfree(R \| S).$$

□

THEOREM B.6. *Let* $\mathbf{pref}R \in GC3$, $\mathbf{pref}S \in GC3$, *and* $Alfcond(R,S)$ *hold.*

0. $Altcond0(R,S) \Rightarrow \mathbf{pref}(R|S) \in GC3$.

1. $Seqcond(R,S) \wedge R$ *is prefix-free* $\Rightarrow \mathbf{pref}(R;S) \in GC3$.
□

The generalization of Theorem B.6.0 is

THEOREM B.7. *For* $n > 0$ *we have*

$$\begin{aligned} &(\mathbf{A}j : 0 \leqslant j < n : \mathbf{pref}(R.j) \in GC3) \\ &\wedge Alfcond(j : 0 \leqslant j < n : R.j) \wedge Altcond0(j : 0 \leqslant j < n : R.j) \\ \Rightarrow\ &\mathbf{pref}(|j : 0 \leqslant j < n : R.j) \in GC3. \end{aligned}$$

□

THEOREM B.8. *Let R and S be non-empty trace structures for which* $Disfree(R)$, $Disfree(S)$, *and* $Alfcond(R,S)$ *hold.*

0. $Altcond1(R,S) \wedge \mathbf{pref}\, R$ *and* $\mathbf{pref}S$ *satisfy rule* $g3 \Rightarrow Disfree(R|S)$.

1. $Seqcond(R,S) \wedge R$ *and* $R\restriction\mathbf{ext}R$ *are prefix-free* $\Rightarrow Disfree(R;S)$.
□

The generalization of Theorem B.8.0 is

THEOREM B.9. *For $n \geq 0$ we have*

$$(\mathbf{A}j: 0 \leq j < n: \mathit{Disfree}(R.j) \wedge \mathbf{pref}(R.j) \text{ satisfies rule } g3)$$
$$\wedge\ \mathit{Alfcond}(j: 0 \leq j < n: R.j) \wedge \mathit{Altcond}1(j: 0 \leq j < n: R.j)$$
$$\Rightarrow\ \mathit{Disfree}(|j: 0 \leq j < n: R.j).$$

□

THEOREM B.10. *For non-empty prefix-free trace structures R and S we have*

0. $\mathit{Altcond}0(R,S) \Rightarrow R|S$ *is prefix-free.*

1. $\mathit{Altcond}1(R,S) \wedge \mathit{Alfcond}(R,S)$
   $\wedge\ R\!\upharpoonright\!\mathbf{ext}R$ *and* $S\!\upharpoonright\!\mathbf{ext}S$ *are prefix-free*
   $\wedge\ \mathbf{pref}R$ *and* $\mathbf{pref}S$ *satisfy rule* $g3$ $\Rightarrow (R|S)\!\upharpoonright\!\mathbf{ext}(R|S)$ *is prefix-free.*

□

THEOREM B.11. *For n.e. trace structures R and S we have*

0. *R and S are prefix-free* $\Rightarrow R;S$ *is prefix-free*

1. $R\!\upharpoonright\!\mathbf{ext}R$ *and* $S\!\upharpoonright\!\mathbf{ext}S$ *are prefix-free*
   $\wedge\ \mathit{Alfcond}(R,S)$ $\Rightarrow (R;S)\!\upharpoonright\!\mathbf{ext}(R;S)$ *is prefix-free.*

□

THEOREM B.12. (Without proof.) *For n.e. trace structures R and S we have*

0. $\mathbf{hd}(R|S) = \mathbf{hd}R \cup \mathbf{hd}S$.
1. $\mathbf{tl}(R|S) = \mathbf{tl}R \cup \mathbf{tl}S$.
2. $\mathbf{hd}(R;S) = \mathbf{hd}R$, if $\mathbf{t}R \neq \{\epsilon\}$ *and R is prefix-free.*
3. $\mathbf{tl}(R;S) = \mathbf{tl}S$, if $\mathbf{t}S \neq \{\epsilon\}$ *and S is prefix-free.*

□

THEOREM B.13. *For n.e. trace structures R and S we have*

$$\mathit{Alfcond}(R,S) \wedge \mathit{Altcond}0(R,S) \Rightarrow \mathbf{first0}(R|S) = \mathbf{first0}R \cup \mathbf{first0}S$$
$$\wedge\ \mathbf{first1}(R|S) = \mathbf{first1}R \cup \mathbf{first1}S.$$

□

THEOREM B.14. *For n.e. trace structures R and S with* $(\mathbf{hd}(R;S) \subseteq \mathbf{in}(R;S) \vee \mathbf{hd}(R;S) \subseteq \mathbf{out}(R;S)) \wedge \mathit{Alfcond}(R,S) \wedge \mathit{Seqcond}(R,S)$ *and R is prefix-free, we have*

0. $\mathbf{t}R=\{\epsilon\} \Rightarrow \mathbf{first0}(R;S)=\mathbf{first0}S$ *and*
   $\mathbf{t}R\neq\{\epsilon\} \Rightarrow \mathbf{first0}(R;S)=\mathbf{first0}R$.

1. *If, moreover,* $R\restriction\mathbf{ext}R$ *is prefix-free and* $\mathbf{pref}R$ *satisfies rule* $g3$*, then*
   $\mathbf{t}R\restriction\mathbf{ext}R=\{\epsilon\} \Rightarrow \mathbf{first1}(R;S)=\mathbf{first1}S$ and
   $\mathbf{t}R\restriction\mathbf{ext}R\neq\{\epsilon\} \Rightarrow \mathbf{first1}(R;S)=\mathbf{first1}R$.

□

THEOREM B.15. *If R is a prefix-free, non-empty trace structure with* $\mathbf{hd}R\subseteq\mathbf{out}R \vee \mathbf{hd}R\subseteq\mathbf{in}R$*, then* $fprop\,0(R) \wedge fprop\,1(R)$ *holds.*

□

THEOREM B.16. *For a non-empty prefix-free trace structure R we have*

0. $\mathbf{t}R=\{\epsilon\} \equiv \mathbf{first0}R\subseteq\{\varnothing\}$.

1. *If, moreover, R satisfies rule* $g3$*, then*
   $\mathbf{t}R\restriction\mathbf{ext}R=\{\epsilon\} \equiv \mathbf{first1}R\subseteq\{\varnothing\}$.

□

B.2. Proofs of Theorems B.0 through B.2

PROOF OF THEOREM B.0. Let $E\in$<*dicom*>. By production rules $(a0)$ and $(a1)$ of Table 4.3.0, we observe $E\in$<*pccom*> or $E=E0\restriction$, where $E0\in$<*pccom*>, respectively. From the condition for production rule $(a0)$ we have $EN(E)=\varnothing \wedge CO(E)=\varnothing$, and we derive

$E\in$<*pccom*> $\wedge\ EN(E)=\varnothing \wedge CO(E)=\varnothing$

⇒ {Theorem B.1}

$E\in GC4 \wedge \mathbf{int}E=\varnothing$

⇒ {calc.}

$E\in C4$.

For $E0\restriction$ we observe

$E0\in$<*pccom*>

⇒ {Theorem B.1}

$E0\in GC4 \wedge Disfree(E0)$

⇒ {Theorem B.3}

$E0\restriction\in C4$.

⇒ $\{E=E0\restriction\}$

$E \in C4.$

□

PROOF OF THEOREM B.1. We prove that for any command $E \in$ <*pccom*> obtained by applying production rule ($b0$), ($b2$), or ($b3$) of Table 4.3.0 satisfies $P1(E) \wedge P2(E)$. Obviously, we have $P1(\epsilon) \wedge P2(\epsilon)$. Application of production rule ($b1$) to commands $E0 \in$ <*pccom*> and $E1 \in$ <*pccom*> leaves $P1$ and $P2$ invariant, since we have

$$P1(E0) \wedge P2(E0) \wedge P1(E1) \wedge P2(E1) \wedge ALFCOND(E0,E1)$$

$$\Rightarrow \{\text{Th. B.5, eval. rules of Table 4.6.1, calc.}\}$$

$$P1(E0 \| E1) \wedge P2(E0 \| E1).$$

From these properties we then conclude the theorem.

For the command **pref**(E) obtained by application of rule ($b2$) we have $E \in$ <*pfcom*>. Hence, by Theorem B.2,

$$\mathbf{pref}(E) \in GC3 \wedge Disfree(E) \wedge P2(E).$$

Since $Disfree(E) \equiv Disfree(\mathbf{pref}(E))$, we derive $P1(\mathbf{pref}(E)) \wedge P2(\mathbf{pref}(E))$.

The command obtained by applying rule ($b3$) is a special case of tail recursion, since

$$\mathbf{pref}[E] = \mu.tailf_0.0, \quad \text{where} \quad tailf_0.R.0 = \mathbf{pref}(E;R.0).$$

Notice that

$$TAILCOND(tailf_0) \equiv E \in <pfcom> \wedge SEQCOND(E,E).$$

Consequently, if we prove that every command E obtained by application of rule ($b0$) satisfies $P1(E) \wedge P2(E)$, then also every command E obtained by application of rule ($b3$) satisfies $P1(E) \wedge P2(E)$.

For commands $\mu.tailf.0$ obtained by application of rule ($b0$) we show for the tail function *tailf* that

$$TAILCOND(tailf) \Rightarrow Tailcond(tailf). \tag{0}$$

From Theorem B.4 we then conclude $P1(\mu.tailf.0)$. (Notice that $GC3 \subseteq GC4$.) Furthermore, by definition of the alphabets of $\mu.tailf.0$ and the evaluation rules of Table 4.6.1, we infer $P2(\mu.tailf.0)$.

Let the tail function *tailf* be defined by array $E(i,j:0 \leq i,j < n)$ and let $TAILCOND(tailf)$ hold. For each command $E.i.j$, $0 \leq i,j < n$, we have, by (3) of $TAILCOND(tailf)$,

$$E.i.j \in <pfcom> \vee E.i.j = \epsilon \vee E.i.j = \varnothing.$$

Consequently, by Theorem B.2,

$$(\mathbf{t}(E.i.j) = \{\epsilon\} \equiv E.i.j = \epsilon) \wedge (\mathbf{t}(E.i.j) = \varnothing \equiv E.i.j = \varnothing). \tag{1}$$

From (1) we deduce

$$(0), (1), \text{ and } (3) \text{ of } TAILCOND(tailf)$$

$$\Rightarrow (0) \text{ and } (1) \text{ of } Tailcond(tailf).$$

Subsequently, we derive

$$ALFCOND(i,j: 0 \leq i,j < n \wedge \mathbf{t}(E.i.j) \neq \epsilon \wedge \mathbf{t}(E.i.j) \neq \varnothing : E.i.j)$$

$$\Rightarrow \{E.i.j \in <pfcom> \vee E.i.j = \epsilon \vee E.i.j = \varnothing, P2(E.i.j) \text{ by Th. B.2}\}$$

$$Alfcond(i,j: 0 \leq i,j < n: E.i.j).$$

Hence, (2) and (3) of $TAILCOND(tailf) \Rightarrow$ (2) of $Tailcond(tailf)$.
For condition (3), we observe

$$(\mathbf{A}i,j: 0 \leq i,j < n \wedge E.i.j \neq \varnothing \wedge E.i.j \neq \epsilon: E.i.j \in <pfcom>)$$

$$\Rightarrow \{\text{Th. B.2, calc.}\}$$

$$(\mathbf{A}i,j: 0 \leq i,j < n \wedge \mathbf{t}(E.i.j) \neq \varnothing$$

$$: \mathbf{pref}\,(E.i.j) \in GC3 \wedge Disfree(E.i.j)$$

$$\wedge\ E.i.j \text{ and } E.i.j \restriction \mathbf{ext}(E.i.j) \text{ are prefix-free}$$

$$).$$

Consequently, (3) of $TAILCOND(tailf) \Rightarrow$ (3) of $Tailcond(tailf)$.
Furthermore, we observe for commands $E.i.j \neq \varnothing$ and $E.j.k \neq \varnothing$, $0 \leq i,j,k < n$,

$$SECOND(E.i.j,\ E.j.k)$$

$$\Rightarrow \{E.i.j \in <pfcom> \vee E.j.k. = \epsilon, \text{ Th. B.2}\}$$

$$Seqcond(E.i.j,\ E.j.k).$$

Hence, (3) and (4) of $TAILCOND(tailf) \Rightarrow$ (4) of $Tailcond(tailf)$.
For condition (5), we first observe that for any command $E \in <pfcom>$ we have $\mathbf{first0}E \neq \varnothing \wedge \mathbf{first1}E \neq \varnothing$, because E is prefix-free and non-empty by Theorem B.2. Subsequently, we also conclude, by Theorem B.15, $fprop\,0(E) \wedge fprop\,1(E)$. Accordingly, for a command $E \in <pfcom>$ we derive

$$\mathbf{first0}E \subseteq \{\varnothing\} \equiv \mathbf{first0}E = \{\varnothing\} \wedge \mathbf{first1}E \subseteq \{\varnothing\} \equiv \mathbf{first1}E = \{\varnothing\}$$

$$\wedge\ fprop\,0(E) \wedge fprop\,1(E) \qquad (2)$$

We derive for all i, $0 \leq i < n$,

$$ALTCOND(j: 0 \leq j < n \wedge E.i.j \neq \varnothing \wedge E.i.j \neq \epsilon: E.i.j)$$

$$= \{E.i.j \in <pfcom>, \text{ Th. B.2, Th. B.15, (2) above,}$$

$$(1) \text{ of } TAILCOND(tailf) \text{ in case } E.i.i = \epsilon\}$$

$$Altcond\,0(j: 0 \leq j < n \wedge \mathbf{t}(E.i.j) \neq \varnothing : E.i.j)$$

$$\wedge\ Altcond\,1(j: 0 \leq j < n \wedge \mathbf{t}(E.i.j) \neq \varnothing : E.i.j).$$

Hence, (3) and (5) of $TAILCOND(tailf) \Rightarrow$ (5) of $Tailcond(tailf)$.

For condition (6) we first observe for any command $E \in \langle pfcom \rangle$, that,
by Theorem B.2, $FIRSTEXT(E) = \mathbf{first1}E$;
by Theorem B.2, B.16 and (2) above, $\mathbf{first1}E = \{\varnothing\} \equiv \mathbf{t}E \upharpoonright \mathbf{ext}E = \{\epsilon\}$;
and by the distribution Properties 1.1.2.3, $\mathbf{t}E \upharpoonright \mathbf{ext}E = \{\epsilon\} \equiv \mathbf{ext}E = \varnothing$.
Consequently,

$$E \in \langle pfcom \rangle \wedge FIRSTEXT(E) = \{\varnothing\} \Rightarrow \mathbf{ext}E = \varnothing.$$

We derive

$$(\mathbf{A}i,j\colon 0 \leq i,j < n \wedge E.i.j \neq \varnothing \wedge E.i.j \neq \epsilon\colon FIRSTEXT(E.i.j) \neq \{\varnothing\})$$

$$\vee\ (\mathbf{A}i,j\colon 0 \leq i,j < n \wedge E.i.j \neq \varnothing \wedge E.i.j \neq \epsilon\colon FIRSTEXT(E.i.j) = \{\varnothing\})$$

$$\Rightarrow \{E.i.j \in \langle pfcom \rangle \vee E.i.j = \epsilon \vee E.i.j = \varnothing, \text{ Th. B.2, Th. B.16, calc.}\}$$

$$(\mathbf{A}i,j\colon 0 \leq i,j < n \wedge E.i.j \neq \varnothing \wedge E.i.j \neq \epsilon\colon (E.i.j) \upharpoonright \mathbf{ext}(E.i.j) \neq \{\epsilon\})$$

$$\vee\ (\mathbf{A}i,j\colon 0 \leq i,j < n\colon \mathbf{ext}(E.i.j) = \varnothing\}.$$

Hence, (3) and (6) of $TAILCOND(tailf) \Rightarrow$ (6) of $Tailcond(tailf)$.
This concludes our proof of obligations for (0).
□

PROOF OF THEOREM B.2. First, we observe, by means of the definitions given in this appendix and Table 4.6.0, that $PF(E)$ holds for every command $E \in \langle marked\ syms \rangle$, where

$$PF(E) \equiv P0(E) \wedge P2(E) \wedge P3(E) \wedge P4(E).$$

Second, we prove that PF remains invariant under the application of production rule $(c1)$, $(c2)$, and $(c3)$ of Table 4.3.0. For rule $(c3)$ this is obvious. For rule $(c1)$ we first observe for any $E \in \langle pfcom \rangle$ that $\mathbf{first1}E \neq \varnothing$. By Theorem B.16.1 and $P0(E)$ we subsequently derive for any $E \in \langle pfcom \rangle$

$$E \upharpoonright \mathbf{ext}E = \{\epsilon\} \equiv \mathbf{first1}E = \{\varnothing\}. \tag{0}$$

We infer

$$PF(E0) \wedge PF(E1) \wedge ALFCOND(E0,E1) \wedge SEQCOND(E0,E1)$$

$$\Rightarrow \{\text{def.of } PF, \text{ calc.}\}$$

$$P0(E0) \wedge P2(E0) \wedge P3(E0) \wedge P4(E0)$$

$$\wedge\ P0(E1) \wedge P2(E1) \wedge P3(E1) \wedge P4(E1)$$

$$\wedge\ Alfcond(E0,E1) \wedge Seqcond(E0,E1)$$

$$\Rightarrow \{\text{Th. B.6.1, Th. B.8.1, Th. B.11, calc., Th. B.12}\}$$

$$P0(E0) \wedge P2(E0) \wedge P3(E0) \wedge P4(E0)$$

$$\wedge\ P0(E1) \wedge P2(E1) \wedge P3(E1) \wedge P4(E1)$$

$\wedge\ Alfcond(E0,E1) \wedge Seqcond(E0,E1) \wedge P0(E0;E1)$

$\Rightarrow$ {Th. B.12, eval. rules of Table 4.6.1, calc.}

$P0(E0) \wedge P3(E0) \wedge P0(E1) \wedge P3(E1)$

$\wedge\ P0(E0;E1) \wedge P2(E0;E1) \wedge P4(E0;E1)$

$\wedge\ Alfcond(E0,E1) \wedge Seqcond(E0,E1)$

$\Rightarrow$ {Th. B.14, eval. rules of Table 4.6.1, (0) above, calc.}

$P0(E0;E1) \wedge P2(E0;E1) \wedge P3(E0;E1) \wedge P4(E0;E1)$

$=$ {def. of PF}

$PF(E0;E1)$.

For rule ($c2$) we observe

$PF(E0) \wedge PF(E1) \wedge ALFCOND(E0,E1) \wedge ALTCOND(E0,E1)$

$\Rightarrow$ {def. of PF, calc.}

$P0(E0) \wedge P2(E0) \wedge P3(E0) \wedge P4(E0)$

$\wedge\ P0(E1) \wedge P2(E1) \wedge P3(E1) \wedge P4(E1)$

$\wedge\ Alfcond(E0,E1) \wedge ALTCOND(E0,E1)$

$\Rightarrow$ {def. $Altcond0$ and $Altcond1$, Th. B.15, (0) above, calc.}

$P0(E0) \wedge P2(E0) \wedge P3(E0) \wedge P4(E0)$

$\wedge\ P0(E1) \wedge P2(E1) \wedge P3(E1) \wedge P4(E1)$

$\wedge\ Alfcond(E0,E1) \wedge Altcond0(E0,E1) \wedge Altcond1(E0,E1)$

$\Rightarrow$ {Th. B.6.0, Th. B.8.0, Th. B.10, Th. B.12, calc.}

$P2(E0) \wedge P3(E0) \wedge P4(E0) \wedge P2(E1) \wedge P3(E1) \wedge P4(E1)$

$\wedge\ Alfcond(E0,E1) \wedge Altcond0(E0,E1) \wedge P0(E0|E1)$

$\Rightarrow$ {Th. B.13, Th. B.12, eval. rules of Table 4.6.1, calc.}

$P0(E0|E1) \wedge P2(E0|E1) \wedge P3(E0|E1) \wedge P4(E0|E1)$

$=$ {def. of PF}

$PF(E0|E1)$.

□

B.3. Proofs of Theorems B.3 through B.5

Proof of Theorem B.3. Let $R \in GC4 \wedge Disfree(R)$ hold. We prove $R\restriction \mathbf{ext}R \in C4$.

rule 1: Since R satisfies rule $g\,1$, it follows immediately that $R\restriction \mathbf{ext}R$ is p.c.n.e. and

$$\mathbf{i}(R\restriction \mathbf{ext}R)\cap \mathbf{o}(R\restriction \mathbf{ext}R) = \varnothing .$$

rule 2: We observe

$saa \in \mathbf{t}R\restriction \mathbf{ext}R$

$\Rightarrow$\{calc., R is p.c.\}

$(\mathbf{E}s_0,s_1 :: s_0 a s_1 a \in \mathbf{t}R \wedge s_0\restriction \mathbf{ext}R = s \wedge s_1 \in (\mathbf{int}R)^*)$

$\Rightarrow$\{R satisfies rule $g\,3$, $a \in \mathbf{ext}R$, Lemma B.17\}

$(\mathbf{E}s_2,s_3 :: s_2 a a s_3 \in \mathbf{t}R)$

$\Rightarrow$\{R satisfies rule $g\,2$, $a \in \mathbf{ext}R$, R is p.c.\}

false.

rule 3: For symbols a and b of the same type, $\{a,b\} \subseteq \mathbf{ext}R$, we observe

$sabt \in \mathbf{t}R\restriction \mathbf{ext}R$

$\Rightarrow$\{calc.\}

$(\mathbf{E}s_0,s_1,t_0 :: s_0 a s_1 b t_0 \in \mathbf{t}R \wedge s_0\restriction \mathbf{ext}R = s$

$\wedge\ s_1 \in (\mathbf{int}R)^* \wedge t_0\restriction \mathbf{ext}R = t)$

$\Rightarrow$\{R satisfies rule $g\,3$, $\{a,b\} \subseteq \mathbf{i}R \vee \{a,b\} \subseteq \mathbf{o}R$, Lemma B.17, calc.\}

$(\mathbf{E}s_2,t_1 :: s_2 a b t_1 \in \mathbf{t}R \wedge s_2\restriction \mathbf{ext}R = s \wedge t_1\restriction \mathbf{ext}R = t)$

$\Rightarrow$\{R satisfies rule $g\,3$, $\{a,b\} \subseteq \mathbf{i}R \vee \{a,b\} \subseteq \mathbf{o}R$\}

$(\mathbf{E}s_2,t_1 :: s_2 b a t_1 \in \mathbf{t}R \wedge s_2\restriction \mathbf{ext}R = s \wedge t_1\restriction \mathbf{ext}R = t)$

$\Rightarrow$\{calc.\}

$sbat \in \mathbf{t}R\restriction \mathbf{ext}R$.

rule 5‴: For symbols $a \in \mathbf{o}R \wedge b \in \mathbf{i}R$, we infer

$sa \in \mathbf{t}R\restriction \mathbf{ext}R \wedge sb \in \mathbf{t}R\restriction \mathbf{ext}R$

$\Rightarrow$\{calc., R is p.c.\}

$(\mathbf{E}s_0,s_1 :: s_0 a \in \mathbf{t}R \wedge s_1 b \in \mathbf{t}R \wedge s_0\restriction \mathbf{ext}R = s \wedge s_1\restriction \mathbf{ext}R = s)$ (1)

$\Rightarrow$\{Lemma B.18 (*i*), *Disfree*(R), R is p.c.,

$a \in \mathbf{o}R \wedge b \in \mathbf{i}R$, take $u_0,u_1,v_0,v_1,r_0,r_1 := \epsilon,\epsilon,\epsilon,\epsilon,s_0,s_1$\}

$(\mathbf{E}s_0,s_1,r :: s_0 a \in \mathbf{t}R \wedge s_1 b \in \mathbf{t}R \wedge s_0\restriction \mathbf{ext}R = s \wedge s_1\restriction \mathbf{ext}R = s$

$\wedge\ r \in \mathbf{t}R\ \wedge\ r{\restriction}(\mathbf{ext}R \cup \mathbf{co}R) = s_0{\restriction}(\mathbf{ext}R \cup \mathbf{co}R)$

$\wedge\ r{\restriction}(\mathbf{ext}R \cup \mathbf{en}R) = s_1{\restriction}(\mathbf{ext}R \cup \mathbf{en}R)\ )$

$\Rightarrow\{Disout(R)\ \wedge\ a \in \mathbf{o}R,\ Disin(R)\ \wedge\ b \in \mathbf{i}R,\ R$ is p.c.$\}$

$(\mathbf{E}s_0,s_1,r{::}\ s_0 a \in \mathbf{t}R\ \wedge\ s_1 b \in \mathbf{t}R\ \wedge\ s_0{\restriction}\mathbf{ext}R = s\ \wedge\ s_1{\restriction}\mathbf{ext}R = s$

$\wedge\ ra \in \mathbf{t}R\ \wedge\ r{\restriction}(\mathbf{ext}R \cup \mathbf{co}R) = s_0{\restriction}(\mathbf{ext}R \cup \mathbf{co}R)$

$\wedge\ rb \in \mathbf{t}R\ \wedge\ r{\restriction}(\mathbf{ext}R \cup \mathbf{en}R) = s_1{\restriction}(\mathbf{ext}R \cup \mathbf{en}R))$

$\Rightarrow\{R$ satisfies rule $g5'''$, $a \in \mathbf{o}R\ \wedge\ b \in \mathbf{i}R$, calc.$\}$

$(\mathbf{E}s_0,s_1,r{::}\ r{\restriction}\mathbf{ext}R = s$

$\wedge\ rab \in \mathbf{t}R\ \wedge\ r{\restriction}(\mathbf{ext}R \cup \mathbf{co}R) = s_0{\restriction}(\mathbf{ext}R \cup \mathbf{co}R)$

$\wedge\ rba \in \mathbf{t}R\ \wedge\ r{\restriction}(\mathbf{ext}R \cup \mathbf{en}R) = s_1{\restriction}(\mathbf{ext}R \cup \mathbf{en}R))$ (2)

$\Rightarrow\{$calc., $\{a,b\} \subseteq \mathbf{ext}R\}$

$sab \in \mathbf{t}R{\restriction}\mathbf{ext}R.$

For reasons of symmetry, we have for $a \in \mathbf{i}R\ \wedge\ b \in \mathbf{o}R$ an analogous proof.

rule 4″: For symbols a, b, and c with $\{a,c\} \subseteq \mathbf{o}R\ \wedge\ b \in \mathbf{i}R$, we observe

$sabtc \in \mathbf{t}R{\restriction}\mathbf{ext}R\ \wedge\ sbat \in \mathbf{t}R{\restriction}\mathbf{ext}R$

$\Rightarrow\{$Lemma B.19, R satisfies rule $g3$, R is p.c.$\}$

$(\mathbf{E}s_0,s_1,t_0,t_1,w_0,w_1{::}\ s_0 a w_0 b t_0 c \in \mathbf{t}R\ \wedge\ s_1 b w_1 a t_1 \in \mathbf{t}R$

$\wedge\ s_0{\restriction}\mathbf{ext}R = s\ \wedge\ w_0 \in (\mathbf{en}R)^*\ \wedge\ t_0{\restriction}\mathbf{ext}R = t$

$\wedge\ s_1{\restriction}\mathbf{ext}R = s\ \wedge\ w_1 \in (\mathbf{co}R)^*\ \wedge\ t_1{\restriction}\mathbf{ext}R = t)$

$\Rightarrow\{$Cf. (1) $\Rightarrow$ (2) in proof of rule $g5'''$, $Disfree(R)$, R is p.c.,

$a \in \mathbf{o}R\ \wedge\ b \in \mathbf{i}R$, take $r := v$, R sat. rule $5'''$, calc.$\}$

$(\mathbf{E}s_0,s_1,t_0,t_1,w_0,w_1,v{::}\ s_0 a w_0 b t_0 c \in \mathbf{t}R\ \wedge\ s_1 b w_1 a t_1 \in \mathbf{t}R$

$\wedge\ s_0{\restriction}\mathbf{ext}R = s\ \wedge\ w_0 \in (\mathbf{en}R)^*\ \wedge\ t_0{\restriction}\mathbf{ext}R = t$

$\wedge\ v{\restriction}\mathbf{ext}R = s\ \wedge\ w_1 \in (\mathbf{co}R)^*\ \wedge\ t_1{\restriction}\mathbf{ext}R = t$

$\wedge\ vab \in \mathbf{t}R\ \wedge\ v{\restriction}(\mathbf{ext}R \cup \mathbf{co}R) = s_0{\restriction}(\mathbf{ext}R \cup \mathbf{co}R)$

$\wedge\ vba \in \mathbf{t}R\ \wedge\ v{\restriction}(\mathbf{ext}R \cup \mathbf{en}R) = s_1{\restriction}(\mathbf{ext}R \cup \mathbf{en}R))$

$\Rightarrow\{$calc., $\{a,b\} \subseteq \mathbf{ext}R\}$

$(\mathbf{E}s_0,s_1,t_0,w_0,w_1,v{::}\ s_0 a w_0 b t_0 c \in \mathbf{t}R\ \wedge\ s_1 b w_1 a t_1 \in \mathbf{t}R$

$\wedge\ v{\restriction}\mathbf{ext}R = s\ \wedge\ w_0 \in (\mathbf{en}R)^*\ \wedge\ t_0{\restriction}\mathbf{ext}R = t\ \wedge\ s_0{\restriction}\mathbf{ext}R = s$

$\wedge\ t_0{\restriction}\mathbf{ext}R = t_1{\restriction}\mathbf{ext}R$

$\wedge\ vab \in \mathbf{t}R\ \wedge\ vab{\restriction}(\mathbf{ext}R \cup \mathbf{co}R) = s_0 a w_0 b{\restriction}(\mathbf{ext}R \cup \mathbf{co}R)$

$\wedge\ vba \in \mathbf{t}R\ \wedge\ vba{\restriction}(\mathbf{ext}R \cup \mathbf{en}R) = s_1 b w_1 a{\restriction}(\mathbf{ext}R \cup \mathbf{en}R))$

$\Rightarrow$ {Lemma B.18 (*ii*), *Disfree*(R), R is p.c., $a \in \mathbf{o}R\ \wedge\ b \in \mathbf{i}R$, R satisfies rule $g4''$, take $u_0, u_1, v_0, v_1, r_0, r_1 := s_0 a w_0 b,\ s_1 b w_1 a,\ vab,\ vba,\ t_0,\ t_1$}

$(\mathbf{E} s_0, t_0, w_0, v, r ::\ s_0 a w_0 b t_0 c \in \mathbf{t}R$

$\wedge\ v{\restriction}\mathbf{ext}R = s\ \wedge\ w_0 \in (\mathbf{en}R)^*\ \wedge\ t_0{\restriction}\mathbf{ext}R = t\ \wedge\ s_0{\restriction}\mathbf{ext}R = s$

$\wedge\ vabr \in \mathbf{t}R\ \wedge\ vabr{\restriction}(\mathbf{ext}R \cup \mathbf{co}R) = s_0 a w_0 b t_0{\restriction}(\mathbf{ext}R \cup \mathbf{co}R)$

$\wedge\ vbar \in \mathbf{t}R\ )$

$\Rightarrow$ {$c \in \mathbf{o}R$, *Disout*(R), R is p.c., calc.}

$(\mathbf{E} v, r ::\ v{\restriction}\mathbf{ext}R = s\ \wedge\ r{\restriction}\mathbf{ext}R = t\ \wedge\ vabrc \in \mathbf{t}R\ \wedge\ vbar \in \mathbf{t}R)$

$\Rightarrow$ {$\{a,c\} \subseteq \mathbf{o}R\ \wedge\ b \in \mathbf{i}R$, R satisfies rule $g4''$}

$(\mathbf{E} v, r ::\ v{\restriction}\mathbf{ext}R = s\ \wedge\ r{\restriction}\mathbf{ext}R = t\ \wedge\ vbarc \in \mathbf{t}R)$

$\Rightarrow$ {calc., $\{a,b,c\} \subseteq \mathbf{ext}R$}

$sbatc \in \mathbf{t}R{\restriction}\mathbf{ext}R$.

For reasons of symmetry, a similar reasoning applies if $\{a,c\} \subseteq \mathbf{i}R\ \wedge\ b \in \mathbf{o}R$.
□

LEMMA B.17. *If R satisfies rule g3 and $\{a,b\} \subseteq \mathbf{o}R\ \vee\ \{a,b\} \subseteq \mathbf{i}R$, then*

$$rasbt \in \mathbf{t}R\ \wedge\ s \in (\mathbf{int}R)^*$$
$$\Rightarrow (\mathbf{E} r', t' ::\ r'abt' \in \mathbf{t}R\ \wedge\ r'{\restriction}\mathbf{ext}R = r{\restriction}\mathbf{ext}R\ \wedge\ t'{\restriction}\mathbf{ext}R = t{\restriction}\mathbf{ext}R).$$

(Symbols a and b may be the same symbols.)

PROOF (Sketch). Assume $a \in \mathbf{o}R$ and $b \in \mathbf{o}R$. Let $rasbt \in \mathbf{t}R\ \wedge\ s \in (\mathbf{int}R)^*$. Since R satisfies rule $g3$, symbols from $\mathbf{co}R$ in s can be shifted to the left (over symbols in $\mathbf{en}R$ and $a \in \mathbf{o}R$) into r. Symbols from $\mathbf{en}R$ in s can be shifted to the right (over symbols in $\mathbf{co}R$ and $b \in \mathbf{o}R$) into t. For $a \in \mathbf{i}R$ and $b \in \mathbf{i}R$ the proof is similar (, only the shift-directions change).
□

LEMMA B.18. *If Disfree(R) and R is prefix-closed, then*

$$u_0 r_0 \in \mathbf{t}R\ \wedge\ v_0 \in \mathbf{t}R\ \wedge\ u_0{\restriction}(\mathbf{ext}R \cup \mathbf{co}R) = v_0{\restriction}(\mathbf{ext}R \cup \mathbf{co}R)$$
$$\wedge\ u_1 r_1 \in \mathbf{t}R\ \wedge\ v_1 \in \mathbf{t}R\ \wedge\ u_1{\restriction}(\mathbf{ext}R \cup \mathbf{en}R) = v_1{\restriction}(\mathbf{ext}R \cup \mathbf{en}R)$$
$$\wedge\ r_0{\restriction}\mathbf{ext}R = r_1{\restriction}\mathbf{ext}R$$
$$\Rightarrow\ (\mathbf{E} r ::\ v_0 r \in \mathbf{t}R\ \wedge\ u_0 r_0{\restriction}(\mathbf{ext}R \cup \mathbf{co}R) = v_0 r{\restriction}(\mathbf{ext}R \cup \mathbf{co}R)$$
$$v_1 r \in \mathbf{t}R\ \wedge\ u_1 r_1{\restriction}(\mathbf{ext}R \cup \mathbf{en}R) = v_1 r{\restriction}(\mathbf{ext}R \cup \mathbf{en}R)),$$

for traces v_0 *and* v_1 *such that*
(*i*) $v_0=\epsilon \wedge v_1=\epsilon$
or
(*ii*) $v_0=vab \wedge v_1=vba \wedge a\in \mathbf{o}R \wedge b\in \mathbf{i}R \wedge R$ *satisfies rule* $g4''$.

PROOF. By induction to the length of r_0 and r_1.
Base: For $r_0=\epsilon \wedge r_1=\epsilon$, take $r=\epsilon$.
Step: We consider two cases in order to comply with $r_0\lceil \mathbf{ext}R=r_1\lceil \mathbf{ext}R$ and the induction step with respect to the length of r_0 and r_1.
(A) $r_0'=r_0d_0 \wedge r_1'=r_1d_1$ with $l(d_0)=1 \wedge d_0\notin \mathbf{i}R \wedge d_1=d_0\lceil \mathbf{ext}R$.
(B) $r_0'=r_0d_0 \wedge r_1'=r_1d_1$ with $l(d_1)=1 \wedge d_1\notin \mathbf{o}R \wedge d_0=d_1\lceil \mathbf{ext}R$,
where $l(r)$ denotes the length of trace r. We have

$$(l(r_0')+l(r_1') = l(r_0)+l(r_1)+1)$$
$$\vee\ (l(r_0')+l(r_1') = l(r_0)+l(r_1)+2 \wedge d_0=d_1).$$

Notice that there is always one case that applies.

First, we consider case (A).

$$u_0r_0d_0\in \mathbf{t}R \wedge v_0\in \mathbf{t}R \wedge u_0\lceil(\mathbf{ext}R\cup \mathbf{co}R)=v_0\lceil(\mathbf{ext}R\cup \mathbf{co}R)$$
$$\wedge\ u_1r_1d_1\in \mathbf{t}R \wedge v_1\in \mathbf{t}R \wedge u_1\lceil(\mathbf{ext}R\cup \mathbf{en}R)=v_1\lceil(\mathbf{ext}R\cup \mathbf{en}R)$$
$$\wedge\ r_0d_0\lceil \mathbf{ext}R=r_1d_1\lceil \mathbf{ext}R.$$
$$\Rightarrow\{\text{ind. hyp. for } r_0 \text{ and } r_1,\ R \text{ is p.c., calc.,}$$
$$\textit{Disfree}(R),\ \text{case (A) or (B)}\}$$
$$(\mathbf{E}r{::}\ u_0r_0d_0\in \mathbf{t}R \wedge u_1r_1d_1\in \mathbf{t}R \wedge d_0\lceil \mathbf{ext}R=d_1\lceil \mathbf{ext}R$$
$$\wedge\ v_0r\in \mathbf{t}R \wedge u_0r_0\lceil(\mathbf{ext}R\cup \mathbf{co}R)=v_0r\lceil(\mathbf{ext}R\cup \mathbf{co}R)$$
$$\wedge\ v_1r\in \mathbf{t}R \wedge u_1r_1\lceil(\mathbf{ext}R\cup \mathbf{en}R)=v_1r\lceil(\mathbf{ext}R\cup \mathbf{en}R)\ )$$
$$\Rightarrow\{\text{ calc., case (A)}\}$$
$$(\mathbf{E}r,d'{::}\ u_0r_0d_0\in \mathbf{t}R \wedge d'=d_0\lceil(\mathbf{co}R\cup \mathbf{o}R)$$
$$\wedge\ v_0r\in \mathbf{t}R \wedge u_0r_0d_0\lceil(\mathbf{ext}R\cup \mathbf{co}R)=v_0rd'\lceil(\mathbf{ext}R\cup \mathbf{co}R)$$
$$\wedge\ v_1r\in \mathbf{t}R \wedge u_1r_1d_1\lceil(\mathbf{ext}R\cup \mathbf{en}R)=v_1rd'\lceil(\mathbf{ext}R\cup \mathbf{en}R)\ )$$
$$\Rightarrow\{\textit{Disout}(R),\ R \text{ is p.c., (A)} \Rightarrow d'=\epsilon \vee (d'\in \mathbf{co}R\cup \mathbf{o}R \wedge d'=d_0)\}$$
$$(\mathbf{E}r,d'{::}\ d'=d_0\lceil(\mathbf{co}R\cup \mathbf{o}R) \qquad (0)$$
$$\wedge\ v_0rd'\in \mathbf{t}R \wedge u_0r_0d_0\lceil(\mathbf{ext}R\cup \mathbf{co}R)=v_0rd'\lceil(\mathbf{ext}R\cup \mathbf{co}R)$$
$$\wedge\ v_1r\in \mathbf{t}R \wedge u_1r_1d_1\lceil(\mathbf{ext}R\cup \mathbf{en}R)=v_1rd'\lceil(\mathbf{ext}R\cup \mathbf{en}R)\ )$$

We distinguish between (*i*) and (*ii*) from here. For (*i*) we observe

$$(0)$$
$$\Rightarrow \{v_0=\epsilon \wedge v_1=\epsilon,\ \text{cf. } (i)\}$$

$(\mathbf{E}r,d' :: rd' \in \mathbf{t}R \wedge u_0 r_0 d_0 \restriction (\mathbf{ext}R \cup \mathbf{co}R) = rd' \restriction (\mathbf{ext}R \cup \mathbf{co}R)$

$\wedge\ rd' \in \mathbf{t}R \wedge u_1 r_1 d_1 \restriction (\mathbf{ext}R \cup \mathbf{en}R) = rd' \restriction (\mathbf{ext}R \cup \mathbf{en}R))$

$\Rightarrow \{r' = rd',\ r_0' = r_0 d_0,\ r_1' = r_1 d_1,\ v_0 = \epsilon,\ v_1 = \epsilon\}$

$(\mathbf{E}r' :: v_0 r' \in \mathbf{t}R \wedge u_0 r_0' \restriction (\mathbf{ext}R \cup \mathbf{co}R) = v_0 r' \restriction (\mathbf{ext} \cup \mathbf{co}R)$

$\wedge\ v_1 r' \in \mathbf{t}R \wedge u_1 r_1' \restriction (\mathbf{ext}R \cup \mathbf{en}R) = v_1 r' \restriction (\mathbf{ext}R \cup \mathbf{en}R))$.

For (*ii*) we infer

(0)

$\Rightarrow \{(ii),\}$

$(\mathbf{E}r,d' :: d' = d_0 \restriction (\mathbf{co}R \cup \mathbf{o}R)$

$\wedge\ vabrd' \in \mathbf{t}R \wedge u_0 r_0 d_0 \restriction (\mathbf{ext}R \cup \mathbf{co}R) = vabrd' \restriction (\mathbf{ext}R \cup \mathbf{co}R)$

$\wedge vbar \in \mathbf{t}R \wedge u_1 r_1 d_1 (\mathbf{ext}R \cup \mathbf{en}R) = vbard' \restriction (\mathbf{ext}R \cup \mathbf{en}R))$

$\Rightarrow \{R$ sat. rule $g4''$, $a \in \mathbf{o}R$, case (A) i.e..

$d' = \epsilon \vee d' \in (\mathbf{co}R \cup \mathbf{o}R)\}$

$(\mathbf{E}r,d' :: vabrd' \in \mathbf{t}R \wedge u_0 r_0 d_0 \restriction (\mathbf{ext}R \cup \mathbf{co}R) = vabrd' \restriction (\mathbf{ext}R \cup \mathbf{co}R)$

$\wedge\ vbard' \in \mathbf{t}R \wedge u_1 r_1 d_1 \restriction (\mathbf{ext}R \cup \mathbf{en}R) = vbard' \restriction (\mathbf{ext}R \cup \mathbf{en}R)\ )$

$\Rightarrow \{r' = rd',\ r_0' = r_0 d_0,\ r_1' = r_1 d_1,\ v_0 = vab,\ v_1 = vba\}$

$(\mathbf{E}r :: v_0 r' \in \mathbf{t}R \wedge u_0 r_0' \restriction (\mathbf{ext}R \cup \mathbf{co}R) = v_0 r' \restriction (\mathbf{ext}R \cup \mathbf{co}R)$

$\wedge\ v_1 r' \in \mathbf{t}R \wedge u_1 r_1' \restriction (\mathbf{ext}R \cup \mathbf{en}R) = v_1 r' \restriction (\mathbf{ext}R \cup \mathbf{en}R))$.

Case (B) is proved similarly, with use of $d' = d_1 \restriction (\mathbf{en}R \cup \mathbf{i}R)$ and $Disin(R)$.
□

LEMMA B.19. *If R is prefix-closed and R satisfies rule g3, then*

$sabtc \in \mathbf{t}R \restriction \mathbf{ext}R \wedge a \in \mathbf{o}R \wedge b \in \mathbf{i}R$

$\Rightarrow (\mathbf{E}s_0, w_0, t_0 :: s_0 a w_0 b t_0 c \in \mathbf{t}R$

$\wedge\ s_0 \restriction \mathbf{ext}R = s \wedge w_0 \in (\mathbf{en}R)^* \wedge t_0 \restriction \mathbf{ext}R = t)$

and

$sbatc \in \mathbf{t}R \restriction \mathbf{ext}R \wedge a \in \mathbf{o}R \wedge b \in \mathbf{i}R$

$\Rightarrow (\mathbf{E}s_1, w_1, t_1 :: s_1 b w_1 a t_1 c \in \mathbf{t}R$

$\wedge\ s_1 \restriction \mathbf{ext}R = s \wedge w_1 \in (\mathbf{co}R)^* \wedge t_1 \restriction \mathbf{ext}R = t)$.

The above properties also hold when symbol c is removed.

PROOF (Sketch). Let R be prefix-closed, $a \in \mathbf{o}R$ and $b \in \mathbf{i}R$, and $s_0 a w_0 b t_0 c$ be an expansion in R of $sabtc \in \mathbf{t}R \restriction \mathbf{ext}R$, i.e.

$$s_0 a w_0 b t_0 c \in \mathbf{t}R \ \wedge\ s_0 \restriction \mathbf{ext}R = s \ \wedge\ w_0 \in (\mathbf{int}R)^* \ \wedge\ t_0 \restriction \mathbf{ext}R = t.$$

Since R satisfies rule $g3$, symbols from $\mathbf{co}R$ in w_0 can be shifted to the right (over symbols from $\mathbf{en}R$ and $b \in \mathbf{i}R$) into t_0. Because $w_0 \in (\mathbf{int}R)^*$, this shifting yields a $w_0' \in (\mathbf{en}R)^*$.

A similar reasoning applies to the second part of the theorem.

□

Proof of Theorem B.4. Let *Tailcond*(*tailf*) hold, where *tailf* is defined by (B0). By condition (0) of *Tailcond*(*tailf*) and Theorem 1.2.4.0 we derive that $\mu.tailf$ exists. Let the predicate P on $V = \mathcal{T}^n(A0, A1, A2, A3)$, where $A0, A1, A2$, and $A3$ are defined as in Section 2.1, be defined by

$$P(R) \equiv (\mathbf{A}i: 0 \leq i < n:\ R.i \in GC3 \ \wedge\ Disfree(R.i) \ \wedge\ \mathbf{hd}(R.i) \subseteq \mathbf{hd}(|j: 0 \leq j < n:\ S.i.j) \ ). \quad \text{(B1)}$$

By means of fixpoint induction we prove that $P(\mu.tailf)$ holds. The theorem then follows from the definition of P.

First we observe, by Lemma B.20, that P is an inductive predicate on V. Second, we infer that $P(\perp_n(A0, A1, A2, A3))$ holds. Third, we prove that *tailf* maintains P, i.e. $P(R) \Rightarrow P(tailf.R)$ for any $R \in V$. By Theorem 1.2.2.1, 1.2.3.0, and 1.2.3.1 we then conclude $P(\mu.tailf)$.

We observe for all i, j, $0 \leq i, j < n$ and $\mathbf{t}(S.i.j) \neq \varnothing$.

$R \in V \wedge P(R)$

$\Rightarrow$ {$\mathbf{t}(S.i.j) \neq \varnothing$, (2) and (3) of *Tailcond*(*tailf*)}

$Alfcond(S.i.j, R.j) \wedge \mathbf{hd}(R.j) \subseteq \mathbf{hd}(|k: 0 \leq k < n: S.j.k)$

$\wedge\ \mathbf{pref}(S.i.j) \in GC3 \wedge R.j \in GC3 \wedge Disfree(S.i.j) \wedge Disfree(R.j)$

$\wedge$ $S.i.j$ and $S.i.j \restriction \mathbf{ext}(S.i.j)$ are prefix-free

$\Rightarrow$ {Lemma B.21, (1) and (4) of *Tailcond*(*tailf*), $\mathbf{t}(S.i.j) \neq \varnothing$}

$Alfcond(S.i.j,\ R.j) \wedge Seqcond(S.i.j,\ R.j)$

$\wedge\ \mathbf{pref}(S.i.j) \in GC3 \wedge R.j \in GC3 \wedge Disfree(S.i.j) \wedge Disfree(R.j)$

$\wedge$ $S.i.j$ and $S.i.j \restriction \mathbf{ext}(S.i.j)$ are prefix-free

$\Rightarrow$ {Theorem B.6.1, Theorem B.8.1, $\mathbf{pref}(R.j) = R.j$, calc.}

$Alfcond(S.i.j,\ R.j) \wedge Seqcond(S.i.j,\ R.j)$

$\wedge\ \mathbf{pref}(S.i.j; R.j) \in GC3 \wedge Disfree(S.i.j; R.j)$

$\wedge$ $S.i.j$ and $S.i.j \restriction \mathbf{ext}(S.i.j)$ are prefix-free.

Furthermore, we observe for all i, $0 \leq i < n$,

(6) of *Tailcond*(*tailf*)

$= \{$def. of *Tailcond*$\}$

$(\mathbf{A}j: 0 \leqslant j < n: \mathbf{ext}(S.i.j) \neq \varnothing)$

$\vee\ (\mathbf{A}j: 0 \leqslant j < n\ \wedge\ \mathbf{t}(S.i.j) \neq \varnothing\ \wedge\ \mathbf{t}(S.i.j) \neq \{\epsilon\}$

$: \mathbf{t}(S.i.j) \restriction \mathbf{ext}(S.i.j) \neq \{\epsilon\})$

$\Rightarrow \{R \in V$, def. of V, calc.$\}$

$(\mathbf{A}j: 0 \leqslant j < n\ \wedge\ \mathbf{t}(S.i.j) \neq \varnothing$

$: \mathbf{t}(S.i.j) \restriction \mathbf{ext}(S.i.j) = \{\epsilon\}\ \wedge\ \mathbf{t}(R.j) \restriction \mathbf{ext}(R.j) = \{\epsilon\})$

$\vee\ (\mathbf{A}j: 0 \leqslant j < n\ \wedge\ \mathbf{t}(S.i.j) \neq \varnothing\ \wedge\ \mathbf{t}(S.i.j) \neq \{\epsilon\}$

$: \mathbf{t}(S.i.j) \restriction \mathbf{ext}(S.i.j) \neq \{\epsilon\})$.

With these observations we derive for all i, $0 \leqslant i < n$,

$R \in V\ \wedge\ P(R)$

$\Rightarrow \{$*Tailcond*(*tailf*), see derivations above$\}$

$(\mathbf{A}j: 0 \leqslant j < n\ \wedge\ \mathbf{t}(S.i.j) \neq \varnothing$

$: Alfcond(S.i.j, R.j)\ \wedge\ Seqcond(S.i.j, R.j)$

$\wedge\ \mathbf{pref}(S.i.j;R.j) \in GC3\ \wedge\ Disfree(S.i.j;R.j)$

$\wedge$ $S.i.j$ and $S.i.j \restriction \mathbf{ext}(S.i.j)$ are prefix-free

$)$

$\wedge\ Altcond0(j: 0 \leqslant j < n\ \wedge\ \mathbf{t}(S.i.j) \neq \varnothing : S.i.j)$

$\wedge\ Altcond1(j: 0 \leqslant j < n\ \wedge\ \mathbf{t}(S.i.j) \neq \varnothing : S.i.j)$

$\wedge\ (\ (\mathbf{A}j: 0 \leqslant j < n\ \wedge\ \mathbf{t}(S.i.j) \neq \varnothing$

$: \mathbf{t}(S.i.j) \restriction \mathbf{ext}(S.i.j) = \{\epsilon\}\ \wedge\ \mathbf{t}(R.j) \restriction \mathbf{ext}(R.j) = \{\epsilon\})$

$\vee\ (\mathbf{A}j: 0 \leqslant j < n\ \wedge\ \mathbf{t}(S.i.j) \neq \varnothing\ \wedge\ \mathbf{t}(S.i.j) \neq \{\epsilon\}$

$: \mathbf{t}(S.i.j) \restriction \mathbf{ext}(S.i.j) \neq \{\epsilon\})$

$)$

$\Rightarrow \{$Lemma B.22, $\mathbf{pref}(S.i.j)$ sat. rule $g3$, (1) of *Tailcond*(*tailf*)$\}$

$(\mathbf{A}j: 0 \leqslant j < n\ \wedge\ \mathbf{t}(S.i.j) \neq \varnothing$

$: \mathbf{pref}(S.i.j;R.j) \in GC3\ \wedge\ Disfree(S.i.j;R.j)\ )$

$\wedge\ Altcond0(j: 0 \leqslant j < n\ \wedge\ \mathbf{t}(S.i.j) \neq \varnothing : S.i.j;R.j)$

$\wedge\ Altcond1(j: 0 \leqslant j < n\ \wedge\ \mathbf{t}(S.i.j) \neq \varnothing : S.i.j;R.j)$

$\Rightarrow \{$Th. B.7, Th. B.9, (0), (2) and (3) of *Tailcond*(*tailf*), i.e.

$\mathbf{pref}(S.i.j)$ satisfies rule $g3\}$

$$\mathbf{pref}(|j:0\leqslant j<n \wedge \mathbf{t}(S.i.j)\neq\varnothing : S.i.j;R.j)\in GC3$$
$$\wedge\ Disfree(|j:0\leqslant j<n \wedge \mathbf{t}(S.i.j)\neq\varnothing : S.i.j;R.j)$$
$\Rightarrow$ {(2) of *Tailcond*(*tailf*), $n>0$, calc.}
$$\mathbf{pref}(|j:0\leqslant j<n: S.i.j;R.j)\in GC3$$
$$\wedge\ Disfree(|j:0\leqslant j<n: S.i.j;R.j)$$
$\Rightarrow$ {def. of *tailf.R.i*, calc.}
$$tailf.R.i\in GC3 \wedge Disfree(tailf.R.i).$$

Finally, we infer for all i, $0\leqslant i<n$,

$$\mathbf{hd}(tailf.R.i)$$
$=$ {def. of *tailf.R.i*}
$$\mathbf{hd}\,\mathbf{pref}(|j:0\leqslant j<n: S.i.j;R.j)$$
$=$ {(3) of *Tailcond*(*tailf*), i.e. *S.i.j* is prefix-free, calc.}
$$\mathbf{hd}(|j:0\leqslant j<n: S.i.j)$$
$$\cup\,\mathbf{hd}(|j:0\leqslant j<n \wedge \mathbf{t}(S.i.j)=\{\epsilon\}: R.j)$$
$\subseteq$ {(1) of *Tailcond*(*tailf*), calc.}
$$\mathbf{hd}(|j:0\leqslant j<n: S.i.j)\cup\mathbf{hd}(R.i)$$
$=$ {$P(R)$}
$$\mathbf{hd}(|j:0\leqslant j<n: S.i.j).$$

Consequently, we conclude $P(R)\Rightarrow P(tailf.R)$.
□

LEMMA B.20. *The predicate P defined on V by (B 1) is inductive.*

PROOF. Let $R(k{:}k\geqslant 0)$ be an ascending chain in V where $P(R.k)$ holds for each k, $k\geqslant 0$. We show that rule $g4'$ of $GC3$ holds for the greatest lower bound $(\sqcup k: k\geqslant 0: R.k).i$ for all i, $0\leqslant i<n$. The other rules for $GC3$, $Disfree(R.i)$, and $\mathbf{hd}(R.i)\subseteq\mathbf{hd}(|j{:}0\leqslant j<n: S.i.j)$ are proved to be inductive similarly.

Let a and b be external symbols of different type and s and t denote traces. We observe

$$sa\in\mathbf{t}((\sqcup k: k\geqslant 0: R.k).i) \wedge sbat\in\mathbf{t}((\sqcup k: k\geqslant 0: R.k).i)$$
$=$ {def. of $\sqcup$, calc.}
$$sa\in\mathbf{t}(|k: k\geqslant 0: R.k.i) \wedge sbat\in\mathbf{t}(|k: k\geqslant 0: R.k.i)$$
$=$ {calc.}

$(\mathbf{E}k,l: k,l \geq 0: sa \in \mathbf{t}(R.k.i) \wedge sbat \in \mathbf{t}(R.l.i))$

$\Rightarrow (k := \max(k,l),\ R(k:k \geq 0)$ is an ascending chain$\}$

$(\mathbf{E}k: k \geq 0: sa \in \mathbf{t}(R.k.i) \wedge sbat \in \mathbf{t}(R.k.i))$

$\Rightarrow \{P(R.k),\ a$ and b are external symbols of different type$\}$

$(\mathbf{E}k: k \geq 0: sabt \in \mathbf{t}(R.k.i))$

$= \{$calc.$\}$

$sabt \in \mathbf{t}(|k: k \geq 0: R.k.i)$

$= \{$def. of $\sqcup\}$

$sabt \in \mathbf{t}(\sqcup k: k \geq 0: R.k).i$.

□

LEMMA B.21. *Let tailf be defined by (B0). Let* $R \in V$, *where* $V = \mathcal{T}^n(A0, A1, A2, A3)$, *and* $S.i.j$, $0 \leq i,j < n$, *be non-empty. We have for each* j, $0 \leq j < n$,

(1) *and* (4) *of* $Tailcond(tailf) \wedge \mathbf{hd}(R.j) \subseteq \mathbf{hd}(|k: 0 \leq k < n: S.j.k)$

$\Rightarrow Seqcond(S.i.j,\ R.j)$.

PROOF. We observe for all i,j with $0 \leq i,j < n$ and $\mathbf{t}(S.i.j) \neq \epsilon \wedge \mathbf{t}(S.i.j) \neq \varnothing$

(4) of $Tailcond(tailf)$

$\Rightarrow \{$def. of $Tailcond\}$

$(\mathbf{A}k: 0 \leq k < n \wedge \mathbf{t}(S.j.k) \neq \varnothing: Seqcond(S.i.j,\ S.j.k))$

$\Rightarrow \{\mathbf{t}(S.i.j) \neq \{\epsilon\}$, calc.$\}$

$Seqcond(S.i.j,\ (|k: 0 \leq k < n \wedge \mathbf{t}(S.j.k) \neq \varnothing: S.j.k))$

$\Rightarrow \{$calc.$\}$

$Seqcond(S.i.j,\ (|k: 0 \leq k < n: S.j.k))$

$\Rightarrow \{\mathbf{hd}(R.j) \subseteq \mathbf{hd}(|k: 0 \leq k < n: S.j.k)$, calc.$\}$

$Seqcond(S.i.j,\ R.j)$.

In case $\mathbf{t}(S.i.j) = \{\epsilon\}$, we derive by (1) of $Tailcond(tailf)$ that $i = j$ and $\mathbf{t}(|k: 0 \leq k < n: S.i.k) = \{\epsilon\}$. Consequently, we observe

$\mathbf{hd}(R.i) \subseteq \mathbf{hd}(|k: 0 \leq k < n: S.i.k)$

$\Rightarrow \{$calc., $\mathbf{t}(S.i.j) = \{\epsilon\}$,(1) of $Tailcond(tailf)\}$

$Seqcond(S.i.j,\ R.j)$.

□

LEMMA B.22. *Let for the arrays of non-empty trace structures* $R(j:0\leq j<n)$

$$\begin{aligned}
&(\mathbf{A}j:0\leq j<n: \mathit{Alfcond}(S.j,R.j) \wedge \mathit{Seqcond}(S.j,\, R.j) \\
&\qquad \wedge\ S.j \text{ and } S.j\restriction \mathbf{ext}(S.j) \text{ are prefix-free} \\
&\qquad \wedge\ \mathbf{pref}(S.j) \text{ satisfies rule } g3 \\
&) \\
&\wedge\ \mathit{Altcond}0(j:0\leq j<n: S.j) \wedge \mathit{Altcond}1(j:0\leq j<n: S.j) \\
&\wedge\ ((\mathbf{A}j:0\leq j<n \wedge \mathbf{t}(S.j)\neq\{\epsilon\}: \mathbf{t}(S.j)\restriction\mathbf{ext}(S.j)\neq\{\epsilon\}) \\
&\quad \vee\ (\mathbf{A}j:0\leq j<n: \mathbf{t}(S.j)\restriction\mathbf{ext}(S.j)=\{\epsilon\} \wedge \mathbf{t}(R.j)\restriction\mathbf{ext}(R.j)=\{\epsilon\}) \\
&).
\end{aligned}$$

If $n>1 \Rightarrow (\mathbf{A}j:0\leq j<n: \mathbf{t}(S.j)\neq\{\epsilon\})$, then for all $n\geq 0$ we have

$$\mathit{Altcond}0(j:0\leq j<n: S.j) \Rightarrow \mathit{Altcond}0(j:0\leq j<n: S.j;R.j)$$

and

$$\mathit{Altcond}1(j:0\leq j<n: S.j) \Rightarrow \mathit{Altcond}1(j:0\leq j<n: S.j;R.j).$$

PROOF. Let $R(j:0\leq j<n)$ and $S(j:0\leq j<n)$ be arrays of n.e. trace structures for which the above holds. Let furthermore

$$n>1 \Rightarrow (\mathbf{A}j:0\leq j<n: \mathbf{t}(S.j)\neq\{\epsilon\})$$

hold. We derive for $n>1$

(*i*) By Theorem B.12.2, $\mathbf{hd}(S.j)=\mathbf{hd}(S.j;R.j)$.

(*ii*) By Lemma B.23

$$\begin{aligned}
&\mathit{fprop}\,0(S.j) \Rightarrow \mathit{fprop}\,0(S.j;R.j) \\
\wedge\ &\mathit{fprop}\,1(S.j) \Rightarrow \mathit{fprop}\,1(S.j;R.j).
\end{aligned}$$

(*iii*) If $\mathbf{first0}(S.j)$ is defined for $0\leq j<n$, then it follows, with (*i*), that $\mathbf{hd}(S.j;R.j)\subseteq\mathbf{in}(S.j;R.j) \vee \mathbf{hd}(S.j;R.j)\subseteq\mathbf{out}(S.j;R.j)$. Hence, by Theorem B.14.0, $\mathbf{first0}(S.j)=\mathbf{first0}(S.j;R.j)$.

(*iv*) Furthermore, if for all j, $0\leq j<n$, $\mathbf{t}(S.j)\restriction\mathbf{ext}(S.j)\neq\{\epsilon\}$, then we derive by Theorem B.14.1,

$$\mathbf{first1}(S.j)=\mathbf{first1}(S.j;R.j).$$

If for all j, $0\leq j<n$, $\mathbf{t}(S.j)\restriction\mathbf{ext}(S.j)=\{\epsilon\} \wedge \mathbf{t}(R.j)\restriction\mathbf{ext}(R.j)=\{\epsilon\}$, then we derive by Theorem B.16.1 and by Theorem B.14.1,

$$\mathbf{first1}(S.j)\subseteq\{\varnothing\} \wedge \mathbf{first1}(R.j)\subseteq\{\varnothing\} \wedge \mathbf{first1}(S.j;R.j)=\mathbf{first1}(R.j).$$

Hence, $\mathbf{first1}(S.j)\subseteq\{\varnothing\} \wedge \mathbf{first1}(S.j;R.j)\subseteq\{\varnothing\}$.

Consequently, by definition of *Altcond*0 and *Altcond*1, we conclude for $n>1$ by (*i*), (*ii*), (*iii*), and (*iv*)

$$Altcond0(j:0 \leqslant j<n:S.j) \Rightarrow Altcond0(j:0 \leqslant j<n:S.j;R.j)$$

and

$$Altcond1(j:0 \leqslant j<n:S.j) \Rightarrow Altcond1(j:0 \leqslant j<n:S.j;R.j).$$

By definition of *Altcond*0 and *Altcond*1, these properties also hold for $n \leqslant 1$.
□

LEMMA B.23. *Let R and S be non-empty trace structures for which* $Alfcond(S, R) \wedge Seqcond(S, R)$ *holds and S is prefix-free. We have*

(*i*) $((\mathbf{t}S=\{\epsilon\} \wedge \mathbf{t}R=\{\epsilon\}) \vee \mathbf{t}S \neq \{\epsilon\})$
$\wedge\ fprop0(S) \qquad \Rightarrow fprop0(S;R).$

(*ii*) $((\mathbf{t}S\!\upharpoonright\!\mathbf{ext}S=\{\epsilon\} \wedge \mathbf{t}R\!\upharpoonright\!\mathbf{ext}R=\{\epsilon\}) \vee \mathbf{t}S\!\upharpoonright\!\mathbf{ext}S \neq \{\epsilon\})$
$\wedge$ $S\!\upharpoonright\!\mathbf{ext}S$ is prefix-free
$\wedge$ **pref**S satisfies rule $g3$
$\wedge\ fprop1(S) \qquad \Rightarrow fprop1(S;R).$

PROOF. Let S and R be n.e. trace structures for which $Alfcond(S,R) \wedge Seqcond(S,R)$ holds. We prove (*ii*). The proof for (*i*) is similar to the proof of (*ii*).

Let $fprop1(S)$ hold, S and $S\!\upharpoonright\!\mathbf{ext}S$ are prefix-free, and **pref**S satisfies rule $g3$. We observe

(*i*)

$$\mathbf{t}S\!\upharpoonright\!\mathbf{ext}S=\{\epsilon\} \wedge \mathbf{t}R\!\upharpoonright\!\mathbf{ext}R=\{\epsilon\}$$

$= \{Alfcond(S,R), \text{calc.}\}$

$$\mathbf{t}(S;R)\!\upharpoonright\!\mathbf{ext}(S;R)=\{\epsilon\}.$$

Consequently, in case $\mathbf{t}S\!\upharpoonright\!\mathbf{ext}S=\{\epsilon\} \wedge \mathbf{t}R\!\upharpoonright\!\mathbf{ext}R=\{\epsilon\}$ we conclude, by the definition of *fprop*1, that $fprop1(S;R)$ holds, because of the empty domain in the quantification.

(*ii*) If $\mathbf{t}S\!\upharpoonright\!\mathbf{ext}S \neq \{\epsilon\}$ we derive

$$\mathbf{t}S\!\upharpoonright\!\mathbf{ext}S \neq \{\epsilon\}$$

$= \{S\!\upharpoonright\!\mathbf{ext}S \text{ is prefix-free}\}$

$$\epsilon \notin \mathbf{t}S\!\upharpoonright\!\mathbf{ext}S.$$

Moreover, from $fprop1(R)$ follows that **first1**S is defined. Since $\epsilon \notin \mathbf{t}S\!\upharpoonright\!\mathbf{ext}S$, we have $\epsilon \notin \mathbf{t}S$, and by Theorem B.12.2 and S being prefix-free we conclude $\mathbf{hd}(S;R) \subseteq \mathbf{in}(S;R) \vee \mathbf{hd}(S;R) \subseteq \mathbf{out}(S;R)$. Furthermore, we infer

$$t \in \mathbf{t}\,\mathbf{pref}(S;R)\!\upharpoonright\!\mathbf{ext}(S;R) \wedge t \neq \epsilon$$

$\Rightarrow \{\text{calc., } \epsilon \notin \mathbf{t}S\!\upharpoonright\!\mathbf{ext}S \text{ see above, } Alfcond(S,R)\}$

$(\mathbf{E}s,r{::}\, t=sr \wedge s\in \mathbf{t}\,\mathbf{pref}\, S\!\upharpoonright\!\mathbf{ext}S \wedge r\in \mathbf{t}\,\mathbf{pref}R\!\upharpoonright\!\mathbf{ext}R$

$\quad \wedge\ (s\in \mathbf{t}S\!\upharpoonright\!\mathbf{ext}S \vee r=\epsilon) \wedge s\neq\epsilon$

$)$

$\Rightarrow \{fprop\,1(S)\}$

$(\mathbf{E}s,r,u{:}\, u\in \mathbf{t}\,\mathbf{pref}\, S\!\upharpoonright\!\mathbf{ext}S \wedge set(u)\in \mathbf{first1}S$

$\quad : (u\preccurlyeq s \vee s\preccurlyeq u) \wedge (s\in \mathbf{t}S\!\upharpoonright\!\mathbf{ext}S \vee r=\epsilon)$

$\quad \wedge\ t=sr \wedge s\in \mathbf{t}\,\mathbf{pref}S\!\upharpoonright\!\mathbf{ext}S \wedge r\in \mathbf{t}\,\mathbf{pref}\,R\!\upharpoonright\!\mathbf{ext}R$

$)$

$\Rightarrow$ {*Alfcond*(S,R), *Seqcond*(S,R), S and $S\!\upharpoonright\!\mathbf{ext}S$ are prefix-free, **pref**S satifies rule $g3$, $\mathbf{t}S\!\upharpoonright\!\mathbf{ext}S\neq\{\epsilon\}$, Theorem B.14.1, calc., $\mathbf{hd}(S;R)\subseteq \mathbf{out}(S;R) \vee \mathbf{hd}(S;R)\subseteq \mathbf{in}(S;R)$, see above}

$(\mathbf{E}s,r,u{:}\, u\in \mathbf{t}\,\mathbf{pref}\, S\!\upharpoonright\!\mathbf{ext}S \wedge set(u)\in \mathbf{first1}(S;R)$

$\quad :(u\preccurlyeq s \vee s\preccurlyeq u) \wedge (s\in \mathbf{t}S\!\upharpoonright\!\mathbf{ext}S \vee r=\epsilon)$

$\quad \wedge\ t=sr \wedge s\in \mathbf{t}\,\mathbf{pref}\, S\!\upharpoonright\!\mathbf{ext}S \wedge r\in \mathbf{t}\,\mathbf{pref}\, R\!\upharpoonright\!\mathbf{ext}R$

$)$

$\Rightarrow$ {calc., $S\!\upharpoonright\!\mathbf{ext}S$ is prefix-free}

$(\mathbf{E}u{:}\, u\in \mathbf{t}\,\mathbf{pref}(S;R) \wedge set(u)\in \mathbf{first1}(S;R)$

$\quad : u\preccurlyeq t \vee t\preccurlyeq u$

$)$.

By definition of *fprop* 1, we conclude that *fprop* $1(S;R)$ holds.

□

PROOF OF THEOREM B.5.0. Let $R\in GC4$, $S\in GC4$, and *Alfcond*(R,S). We prove $R\|S\in GC4$.

rule g1: Since R and S are p.c.n.e. we have that $R\|S$ is p.c.n.e. as well. Because of *Alfcond*(R,S), it follows that any two alphabets of distinct type of $R\|S$ are disjoint.

rule g2: Let $a\in \mathbf{ext}(R\|S)$, we observe

$saa\in \mathbf{t}(R\|S)$

$\Rightarrow$ {def. of weaving}

$saa\!\upharpoonright\!\mathbf{a}R\in \mathbf{t}R \wedge saa\!\upharpoonright\!\mathbf{a}S\in \mathbf{t}S$

$\Rightarrow$ {R and S satisfy rule $g2$, *Alfcond*(R,S), calc.}

$false.$

rule g3: Let the symbols x and y satisfy

$$(x \in \mathbf{i}(R\|S) \cup \mathbf{co}(R\|S) \ \wedge\ y \in \mathbf{i}(R\|S) \cup \mathbf{en}(R\|S))$$

$$\vee\ (x \in \mathbf{o}(R\|S) \cup \mathbf{en}(R\|S) \ \wedge\ y \in \mathbf{o}(R\|S) \cup \mathbf{co}(R\|S))$$

We observe

$$sxyt \in \mathbf{t}(R\|S)$$

$$\Rightarrow\{\text{def. of weaving}\}$$

$$sxyt\lceil \mathbf{a}R \in \mathbf{t}R \ \wedge\ sxyt\lceil \mathbf{a}S \in \mathbf{t}S$$

$$\Rightarrow\{\text{calc., } Alfcond(R,S),\ R \text{ and } S \text{ satisfy rule } g3\}$$

$$syxt\lceil \mathbf{a}R \in \mathbf{t}R \ \wedge\ syxt\lceil \mathbf{a}S \in \mathbf{t}S$$

$$\Rightarrow\{\text{def. of weaving, } syxt \in (\mathbf{a}R \cup \mathbf{a}S)^*\}$$

$$syxt \in \mathbf{t}(R\|S).$$

rule g4″: Let the symbols a and b be of different type, $\{a,b\} \subseteq \mathbf{ext}(R\|S)$. We observe

$$sabtc \in \mathbf{t}(R\|S) \ \wedge\ sbat \in \mathbf{t}(R\|S)$$

$$\Rightarrow\{\text{def. of weaving}\}$$

$$sabtc\lceil \mathbf{a}R \in \mathbf{t}R \ \wedge\ sbat\lceil \mathbf{a}R \in \mathbf{t}R$$

$$\wedge\ sabtc\lceil \mathbf{a}S \in \mathbf{t}S \ \wedge\ sbat\lceil \mathbf{a}S \in \mathbf{t}S$$

$$\Rightarrow\{\text{calc., } Alfcond(R,S),\ R \text{ and } S \text{ satisfy rule } g4''\}$$

$$sbatc\lceil \mathbf{a}R \in \mathbf{t}R \ \wedge\ sbatc\lceil \mathbf{a}S \in \mathbf{t}S$$

$$\Rightarrow\{\text{def. of weaving, } sbatc \in (\mathbf{a}R \cup \mathbf{a}S)^*\}$$

$$sbatc \in \mathbf{t}(R\|S).$$

rule g5‴: Similar to rule $g4''$.
□

PROOF OF THEOREM B.5.1. Let R and S be n.e.p.c. trace structures for which $Disfree(R)$, $Disfree(S)$, and $Alfcond(R,S)$ hold. We prove $Disfree(R\|S)$.
We observe for arbitrary traces u, v, and symbol b,

$$u \in \mathbf{t}\,\mathbf{pref}(R\|S) \ \wedge\ vb \in \mathbf{t}\,\mathbf{pref}(R\|S) \ \wedge\ b \in \mathbf{out}(R\|S)$$

$$\wedge\ u\lceil(\mathbf{ext}(R\|S) \cup \mathbf{co}(R\|S)) = v\lceil(\mathbf{ext}(R\|S) \cup \mathbf{co}(R\|S))$$

$$\Rightarrow\{\text{def. of weaving, } Alfcond(R,S), \text{ calc.}\}$$

$$u\!\restriction\!\mathbf{a}R \in \mathbf{t}\,\mathbf{pref}R \;\wedge\; vb\!\restriction\!\mathbf{a}R \in \mathbf{t}\,\mathbf{pref}R \;\wedge\; (b\!\restriction\!\mathbf{a}R \in \mathbf{out}R \;\vee\; b\!\restriction\!\mathbf{a}R = \epsilon)$$

$$\wedge\; u\!\restriction\!(\mathbf{ext}R \cup \mathbf{co}R) = v\!\restriction\!(\mathbf{ext}R \cup \mathbf{co}R)$$

$\Rightarrow$ {If $b\!\restriction\!\mathbf{a}R \neq \epsilon$ we use $Disout(R)$, calc.}

$$(u\!\restriction\!\mathbf{a}R)(b\!\restriction\!\mathbf{a}R) \in \mathbf{t}\,\mathbf{pref}R$$

$\Rightarrow$ {calc.}

$$ub\!\restriction\!\mathbf{a}R \in \mathbf{t}\,\mathbf{pref}R.$$

Similarly, with $Disout(S)$, we find $ub\!\restriction\!\mathbf{a}S \in \mathbf{t}\,\mathbf{pref}S$. Since $ub \in (\mathbf{a}R \cup \mathbf{a}S)^*$, we derive, by definition of weaving,

$$ub \in \mathbf{t}(\mathbf{pref}R \,\|\, \mathbf{pref}S)$$

$\Rightarrow$ {$\mathbf{pref}(R\|S) = \mathbf{pref}R \,\|\, \mathbf{pref}S$ for prefix-closed R and S}

$$ub \in \mathbf{t}\,\mathbf{pref}(R\|S).$$

Consequently, $Disout(R\|S)$ holds.

Similarly, we derive

$$Disin(R) \wedge Disin(S) \wedge Alfcond(R,S) \;\Rightarrow\; Disin(R\|S).$$

□

B.4. Proofs of Theorems B.6 through B.9

Proof of Theorem B.6.0. Let $\mathbf{pref}\,R \in GC3$, $\mathbf{pref}\,S \in GC3$, $Alfcond(R,S)$, and $Altcond0(R,S)$ hold. We prove $\mathbf{pref}(R|S) \in GC3$.

rule g1: Obviously, $\mathbf{pref}(R|S)$ is also prefix-closed and non-empty. Because of $Alfcond(R,S)$, it readily follows that any two alphabets of distinct type of $R|S$ are disjoint.

rule g2: Let $a \in \mathbf{ext}(R|S)$. We observe

$$saa \notin \mathbf{t}\,\mathbf{pref}(R|S)$$

= {calc.}

$$saa \notin \mathbf{t}\,\mathbf{pref}R \;\wedge\; saa \notin \mathbf{t}\,\mathbf{pref}S$$

= {$\mathbf{pref}R$ and $\mathbf{pref}S$ satisfy rule $g2$}

true.

rule g3: Let

$$(x \in \mathbf{i}(R|S) \cup \mathbf{co}(R|S) \;\wedge\; y \in \mathbf{i}(R|S) \cup \mathbf{en}(R|S))$$

$$\vee\; (x \in \mathbf{o}(R|S) \cup \mathbf{en}(R|S) \;\wedge\; y \in \mathbf{o}(R|S) \cup \mathbf{co}(R|S)).$$

We observe

$sxyt \in \mathbf{t}\,\mathbf{pref}(R|S)$

$= \{\text{calc.}\}$

$sxyt \in \mathbf{t}\,\mathbf{pref}R \;\vee\; sxyt \in \mathbf{t}\,\mathbf{pref}S$

$\Rightarrow \{Alfcond(R,S),\ \mathbf{pref}R$ and $\mathbf{pref}S$ satisfy rule $g3\}$

$syxt \in \mathbf{t}\,\mathbf{pref}R \;\vee\; syxt \in \mathbf{t}\,\mathbf{pref}S$

$= \{\text{calc.}\}$

$syxt \in \mathbf{t}\,\mathbf{pref}(R|S)$.

rule g4′: Let a and b be of different type, $\{a,b\} \subseteq \mathbf{ext}(R|S)$.

$sabt \in \mathbf{t}\,\mathbf{pref}(R|S) \;\wedge\; sb \in \mathbf{t}\,\mathbf{pref}(R|S)$

$= \{\text{calc.}\}$

$(sabt \in \mathbf{t}\,\mathbf{pref}R \;\vee\; sabt \in \mathbf{t}\,\mathbf{pref}S) \;\wedge\; (sb \in \mathbf{t}\,\mathbf{pref}R \;\vee\; sb \in \mathbf{t}\,\mathbf{pref}S)$

$= \{Altcond0(R,S),\ Alfcond(R,S)$, Lemma B. 24, calc.$\}$

$(sabt \in \mathbf{t}\,\mathbf{pref}R \;\wedge\; sb \in \mathbf{t}\,\mathbf{pref}R) \;\vee\; (sabt \in \mathbf{t}\,\mathbf{pref}S \;\wedge\; sb \in \mathbf{t}\,\mathbf{pref}S)$

$\Rightarrow \{\mathbf{pref}R$ and $\mathbf{pref}S$ satisfy rule $g4'$, $Alfcond(R,S\}$

$sbat \in \mathbf{t}\,\mathbf{pref}R \;\vee\; sbat \in \mathbf{t}\,\mathbf{pref}S$

$= \{\text{calc.}\}$

$sbat \in \mathbf{t}\,\mathbf{pref}(R|S)$.

rule g5‴: Let a and b be of different type $\{a,b\} \subseteq \mathbf{ext}(R|S)$.

$sa \in \mathbf{t}\,\mathbf{pref}(R|S) \;\wedge\; sb \in \mathbf{t}\,\mathbf{pref}(R|S)$

$\Rightarrow \{Alfcond(R,S),\ Altcond0(R,S)$, Lemma B.24, calc.$\}$

$(sa \in \mathbf{t}\,\mathbf{pref}R \;\wedge\; sb \in \mathbf{t}\,\mathbf{pref}R) \;\vee\; (sa \in \mathbf{t}\,\mathbf{pref}S \;\wedge\; sb \in \mathbf{t}\,\mathbf{pref}S)$

$\Rightarrow \{\mathbf{pref}R$ and $\mathbf{pref}S$ satisfy rule $g5'''$, $Alfcond(R,S)\}$

$sab \in \mathbf{t}\,\mathbf{pref}R \;\vee\; sba \in \mathbf{t}\,\mathbf{pref}S$

$\Rightarrow \{\text{calc.}\}$

$sab \in \mathbf{t}\,\mathbf{pref}(R|S)$.

□

LEMMA B.24. *For a and b of different type, $\{a,b\} \subseteq \mathbf{ext}(R|S)$, $Alfcond(R,S)$, and $Altcond0(R,S)$ we have*

$\neg(sa \in \mathbf{t}\,\mathbf{pref}R \;\wedge\; sb \in \mathbf{t}\,\mathbf{pref}S)$.

PROOF. Let $Alfcond(R,S)$ and $Altcond0(R,S)$ hold. Assume $a \in \mathbf{o}(R|S) \wedge b \in \mathbf{i}(R|S)$ and $\mathbf{hd}R \subseteq \mathbf{in}R \wedge \mathbf{hd}S \subseteq \mathbf{in}S$. We infer

$$sa \in \mathbf{t\,pref}R \wedge sb \in \mathbf{t\,pref}S$$

$$\Rightarrow \{\mathbf{hd}R \subseteq \mathbf{in}R,\ Alfcond(R,S) \Rightarrow a \in \mathbf{o}R, \text{ def. of } \mathbf{first0}R, \text{ calc.}\}$$

$$(\mathbf{E}r : r \in \mathbf{t\,pref}R \wedge set(r) \in \mathbf{first0}R$$

$$: r \preccurlyeq s \wedge sb \in \mathbf{t\,pref}S \wedge r \neq \epsilon)$$

$$\Rightarrow \{Altcond0(R,S) \Rightarrow fprop0(S), \text{ calc.}\}$$

$$(\mathbf{E}r,t : r \in \mathbf{t\,pref}R \wedge set(r) \in \mathbf{first0}R \wedge t \in \mathbf{t\,pref}S \wedge set(t) \in \mathbf{first0}S$$

$$: (r \preccurlyeq t \vee t \preccurlyeq r) \wedge r \neq \epsilon)$$

$$\Rightarrow \{Altcond0(R,S) \Rightarrow llcond0(R,S)\}$$

$$false.$$

For reasons of symmetry, a similar reasoning applies when $\mathbf{hd}R \subseteq \mathbf{out}R \wedge \mathbf{hd}S \subseteq \mathbf{out}S$.
□

PROOF OF THEOREM B.6.1.
Let $\mathbf{pref}R \in GC3$, $\mathbf{pref}S \in GC3$, $Alfcond(R,S)$, and $Seqcond(R,S)$ hold and R be prefix-free. We prove $\mathbf{pref}(R;S) \in GC3$.

rule g1: Since **pref**R and **pref**S are p.c.n.e. also **pref**$(R;S)$ is p.c.n.e.. Because of $Alfcond(R,S)$, it follows that any two alphabets of distinct type of **pref**$(R;S)$ are disjoint.

For each of the following rules three cases are distinguished corresponding to the ways in which a trace can be parsed as a member of **pref**$(R;S)$.

rule g2: Let $a \in \mathbf{ext}(R;S)$ and $saa \in \mathbf{t\,pref}(R;S)$. We distinguish three cases.
(*i*)

$$saa \in \mathbf{t\,pref}R$$

$$= \{\mathbf{pref}R \text{ satisfies rule } g2\}$$

$$false.$$

(*ii*)

$$sa \in \mathbf{t}R \wedge a \in \mathbf{t\,pref}S$$

$$= \{\text{def. of } \mathbf{tl} \text{ and } \mathbf{hd}\}$$

$$a \in \mathbf{tl}R \wedge a \in \mathbf{hd}S$$

$= \{ \mathit{Alfcond}(R,S),\ \mathit{Seqcond}(R,S) \}$

false.

(*iii*)

$(\mathbf{E}u,v :: saa = uvaa \ \wedge\ u \in \mathbf{t}R \ \wedge\ vaa \in \mathbf{t}\,\mathbf{pref}S)$

$= \{ \mathbf{pref}S$ satisfies rule $g2 \}$

false.

Hence, from (*i*), (*ii*) and (*iii*) we conclude $saa \notin \mathbf{t}\,\mathbf{pref}(R;S)$.

rule g3: Let the symbols x and y satisfy

$(x \in \mathbf{i}(R;S) \cup \mathbf{co}(R;S) \ \wedge\ y \in \mathbf{i}(R;S) \cup \mathbf{en}(R;S))$

$\vee\ (x \in \mathbf{o}(R;S) \cup \mathbf{en}(R;S) \ \wedge\ y \in \mathbf{o}(R;S) \cup \mathbf{co}(R;S))$

and $sxyt \in \mathbf{t}\,\mathbf{pref}(R;S)$. We consider three cases.

(*i*)

$(\mathbf{E}u,v :: sxyt = sxyuv \ \wedge\ sxyu \in \mathbf{t}R \ \wedge\ v \in \mathbf{t}\,\mathbf{pref}S)$

$\Rightarrow \{ \mathit{Alfcond}(R,S),\ \mathbf{pref}R$ satisfies rule $g3$, calc.$\}$

$(\mathbf{E}u,v :: t = uv \ \wedge\ syxu \in \mathbf{t}R \ \wedge\ v \in \mathbf{t}\,\mathbf{pref}S)$

$\Rightarrow \{$calc.$\}$

$syxt \in \mathbf{t}\,\mathbf{pref}(R;S)$.

(*ii*)

$sx \in \mathbf{t}R \ \wedge\ yt \in \mathbf{t}\,\mathbf{pref}S$

$\Rightarrow \{$def. of $\mathbf{hd}$ and $\mathbf{tl}\}$

$x \in \mathbf{tl}R \ \wedge\ y \in \mathbf{hd}S$

$\Rightarrow \{ \mathit{Alfcond}(R,S),\ \mathit{Seqcond}(R,S) \}$

false.

(*iii*)

$(\mathbf{E}u,v :: sxyt = uvxyt \ \wedge\ u \in \mathbf{t}R \ \wedge\ vxyt \in \mathbf{t}\,\mathbf{pref}S)$

$\Rightarrow \{ \mathit{Alfcond}(R,S),\ \mathbf{pref}S$ satisfies rule $g3 \}$

$(\mathbf{E}u,v :: s = uv \ \wedge\ u \in \mathbf{t}R \ \wedge\ vyxt \in \mathbf{t}\,\mathbf{pref}S)$

$\Rightarrow \{$calc.$\}$

$syxt \in \mathbf{t}\,\mathbf{pref}(R;S)$.

Consequently, (*i*) $\vee$ (*ii*) $\vee$ (*iii*) $\Rightarrow syxt \in \mathbf{t}\,\mathbf{pref}(R;S)$.

rule g4′: Let the symbols a and b be of different type and $\{a,b\} \subseteq \mathbf{ext}R \wedge sa \in \mathbf{t}\,\mathbf{pref}(R;S) \wedge sbat \in \mathbf{t}\,\mathbf{pref}(R;S)$. We distinguish three cases

(*i*)

$(\mathbf{E}u,v :: sa \in \mathbf{t}\,\mathbf{pref}(R;S) \wedge sbat = sbauv \wedge sbau \in \mathbf{t}R \wedge v \in \mathbf{t}\,\mathbf{pref}S)$

$\Rightarrow$ {R is prefix-free, Lemma B.25.0, calc.}

$(\mathbf{E}u,v :: sa \in \mathbf{t}\,\mathbf{pref}R \wedge t = uv \wedge sbau \in \mathbf{t}R \wedge v \in \mathbf{t}\,\mathbf{pref}S)$

$\Rightarrow${**pref**R satisfies rule $g4'$, *Alfcond*(R,S), Lemma B.26, R is prefix-free}

$(\mathbf{E}u,v :: t = uv \wedge sabu \in \mathbf{t}R \wedge v \in \mathbf{t}\,\mathbf{pref}S)$

$\Rightarrow${calc.}

$sabt \in \mathbf{t}\,\mathbf{pref}(R;S)$.

(*ii*)

$sa \in \mathbf{t}\,\mathbf{pref}(R;S) \wedge sb \in \mathbf{t}R \wedge at \in \mathbf{t}\,\mathbf{pref}S$

$\Rightarrow${R is prefix-free, Lemma B.25.0}

$sa \in \mathbf{t}\,\mathbf{pref}R \wedge sb \in \mathbf{t}R$

$\Rightarrow${*Alfcond*(R,S), **pref**R satisfies rule $g5'''$}

$sba \in \mathbf{t}\,\mathbf{pref}R \wedge sb \in \mathbf{t}R$

$=${R is prefix-free}

false.

(*iii*)

$(\mathbf{E}u,v :: sa \in \mathbf{t}\,\mathbf{pref}(R;S) \wedge sbat = uvbat \wedge u \in \mathbf{t}R \wedge vbat \in \mathbf{t}\,\mathbf{pref}S)$

$\Rightarrow${R is prefix-free, Lemma B.25.1}

$(\mathbf{E}u,v :: s = uv \wedge u \in \mathbf{t}R \wedge va \in \mathbf{t}\,\mathbf{pref}S \wedge vbat \in \mathbf{t}\,\mathbf{pref}S)$

$\Rightarrow${*Alfcond*(R,S), **pref**S satisfies rule $g4'$}

$(\mathbf{E}u,v :: s = uv \wedge u \in \mathbf{t}R \wedge vabt \in \mathbf{t}\,\mathbf{pref}S)$

$\Rightarrow${calc.}

$sabt \in \mathbf{t}\,\mathbf{pref}(R;S)$.

Accordingly, $(i) \vee (ii) \vee (iii) \Rightarrow sabt \in \mathbf{t}\,\mathbf{pref}(R;S)$.

rule $g5'''$: Similar to proof of rule $g4'$.
□

LEMMA B.25. (Without proof) *For prefix-free R and non-empty S, we have for traces r and s, and symbols a and b,*

0. $sb \in \mathbf{t}\,\mathbf{pref}R \wedge sa \in \mathbf{t}\,\mathbf{pref}(R\,;S) \Rightarrow sa \in \mathbf{t}\,\mathbf{pref}R.$
1. $r \in \mathbf{t}R \wedge rs \in \mathbf{t}\,\mathbf{pref}(R\,;S) \Rightarrow s \in \mathbf{t}\,\mathbf{pref}S.$

□

LEMMA B.26. *If* **pref***R satisfies rule g4′, R is prefix-free, and a and b are of different type with* $\{a,b\} \subseteq \mathbf{ext}R$, *then*

$$sa \in \mathbf{t}\,\mathbf{pref}R \wedge sbau \in \mathbf{t}R \Rightarrow sabu \in \mathbf{t}R.$$

PROOF. We observe

$$sa \in \mathbf{t}\,\mathbf{pref}R \wedge sbau \in \mathbf{t}R$$

$$\Rightarrow \{\mathbf{pref}R \text{ satisfies rule } g4',\ a \text{ and } b \text{ of dif. type, } \{a,b\} \subseteq \mathbf{ext}R\}$$

$$sabu \in \mathbf{t}\,\mathbf{pref}R.$$

Furthermore, for any symbol c we derive

$$sabuc \in \mathbf{t}\,\mathbf{pref}R \wedge sbau \in \mathbf{t}R$$

$$\Rightarrow \{\mathbf{pref}R \text{ satisfies rule } g4',\ a \text{ and } b \text{ of dif. type, } \{a,b\} \subseteq \mathbf{ext}R, \text{calc.}\}$$

$$sbauc \in \mathbf{t}\,\mathbf{pref}R \wedge sbau \in \mathbf{t}R$$

$$\Rightarrow \{R \text{ is prefix-free}\}$$

$$\mathit{false}.$$

From the above two observations we conclude the lemma.
□

PROOF OF THEOREM B.7. Generalization of proof of Theorem B.6.0 to n trace structures, $n > 0$.
□

PROOF OF THEOREM B.8.0. Let R and S be n.e. trace structures for which $\mathit{Disfree}(R) \wedge \mathit{Disfree}(S) \wedge \mathit{Altcond}\,1(R,S) \wedge \mathit{Alfcond}(R,S)$ holds. Furthermore, let **pref**R and **pref**S satisfy rule $g3$. We prove $\mathit{Disfree}(R|S)$.

We observe

$$u \in \mathbf{t}\,\mathbf{pref}(R|S) \wedge vb \in \mathbf{t}\,\mathbf{pref}(R|S) \wedge b \in \mathbf{out}(R|S)$$

$$\wedge\ u\restriction(\mathbf{ext}(R|S) \cup \mathbf{co}(R|S)) = v\restriction(\mathbf{ext}(R|S) \cup \mathbf{co}(R|S))$$

$$\Rightarrow \{\text{Assume } vb \in \mathbf{t}\,\mathbf{pref}R,\ \mathit{Alfcond}(R,S)\}$$

$$u \in \mathbf{t}\,\mathbf{pref}(R|S) \wedge vb \in \mathbf{t}\,\mathbf{pref}R \wedge b \in \mathbf{out}R$$

$$\wedge\ u\restriction(\mathbf{ext}(R|S) \cup \mathbf{co}(R|S)) = v\restriction(\mathbf{ext}(R|S) \cup \mathbf{co}(R|S))$$

$\Rightarrow\{$Lemma B.27, $Alfcond(R,S)$, $Altcond1(R,S)$,

pref$R$ and **pref**S satisfy rule $g3\}$

$u \in_{\mathbf{t}} \mathbf{pref}R \wedge vb \in_{\mathbf{t}} \mathbf{pref}R \wedge b \in \mathbf{out}R$

$\wedge\ u\restriction(\mathbf{ext}(R|S)\cup\mathbf{co}(R|S)) = v\restriction(\mathbf{ext}(R|S)\cup\mathbf{co}(R|S))$

$\Rightarrow\{$calc.$\}$

$u \in_{\mathbf{t}} \mathbf{pref}R \wedge vb \in_{\mathbf{t}} \mathbf{pref}R \wedge b \in \mathbf{out}R$

$\wedge\ u\restriction(\mathbf{ext}R\cup\mathbf{co}R) = v\restriction(\mathbf{ext}R\cup\mathbf{co}R)$

$\Rightarrow\{Disout(R)\}$

$ub \in_{\mathbf{t}} \mathbf{pref}R$

$\Rightarrow\{$calc.$\}$

$ub \in_{\mathbf{t}} \mathbf{pref}(R|S)$.

For $vb \in_{\mathbf{t}} \mathbf{pref}S$ a similar reasoning applies. Consequently, $Disout(R|S)$ holds.
Similarly, we derive

$Disin(R) \wedge Disin(S) \wedge Alfcond(R,S) \wedge Altcond1(R,S)$

$\wedge$ **pref**R and **pref**S satisfy rule $g3$

$\Rightarrow Disin(R|S)$.

□

LEMMA B.27. *For n.e. trace structures R and S with $Alfcond(R,S) \wedge Altcond1(R,S)$ and* **pref**R *and* **pref**S *satisfy rule g3, we have*

$u \in_{\mathbf{t}} \mathbf{pref}(R|S) \wedge vb \in_{\mathbf{t}} \mathbf{pref}R \wedge b \in \mathbf{out}R$

$\wedge\ u\restriction(\mathbf{ext}(R|S)\cup\mathbf{co}(R|S)) = v\restriction(\mathbf{ext}(R|S)\cup\mathbf{co}(R|S))$

$\Rightarrow u \in_{\mathbf{t}} \mathbf{pref}R$

and

$u \in_{\mathbf{t}} \mathbf{pref}(R|S) \wedge vb \in_{\mathbf{t}} \mathbf{pref}R \wedge b \in \mathbf{in}R$

$\wedge\ u\restriction(\mathbf{ext}(R|S) \wedge \mathbf{en}(R|S)) = v\restriction(\mathbf{ext}(R|S)\cup\mathbf{en}(R|S))$

$\Rightarrow u \in_{\mathbf{t}} \mathbf{pref}R$.

PROOF. Let R and S be n.e. trace structures with $Alfcond(R,S) \wedge Altcond1(R,S)$ and **pref**R and **pref**S satisfy rule $g3$. We observe

$u\restriction(\mathbf{ext}(R|S)\cup\mathbf{co}(R|S)) = v\restriction(\mathbf{ext}(R|S)\cup\mathbf{co}(R|S))$

$\Rightarrow\{$calc.$\}$

$$u{\restriction}\mathbf{ext}(R|S)=v{\restriction}\mathbf{ext}(R|S) \qquad (0)$$

$$\wedge\ u{\restriction}\mathbf{out}(R|S)=v{\restriction}\mathbf{out}(R|S). \qquad (1)$$

We first assume, because of *Altcond* 1(R,S), $\mathbf{hd}(R)\subseteq\mathbf{out}R \wedge \mathbf{hd}(S)\subseteq\mathbf{out}S$. We infer

$$u\in\mathbf{t\,pref}(R|S) \wedge vb\in\mathbf{t\,pref}R \wedge b\in\mathbf{out}R$$

$=\{$calc.$\}$

$$(u\in\mathbf{t\,pref}(R|S) \wedge vb\in\mathbf{t\,pref}R \wedge u\in(\mathbf{out}(R|S))^* \wedge vb\in(\mathbf{out}R)^*)$$

$$\vee\ (u\in\mathbf{t\,pref}(R|S) \wedge vb\in\mathbf{t\,pref}R \wedge (u{\restriction}\mathbf{in}(R|S)\neq\epsilon \vee vb{\restriction}\mathbf{in}R\neq\epsilon))$$

$\Rightarrow\{$Lemma B.28, *Alfcond*$(R,S) \wedge$ *Altcond* 1(R,S), $v, u := vb, u$, $\mathbf{pref}R$ and $\mathbf{pref}S$ satisfy rule $g3$, $\mathbf{hd}(R)\subseteq\mathbf{out}R \wedge \mathbf{hd}(S)\subseteq\mathbf{out}S$, (0)$\}$

$$(u\in\mathbf{t\,pref}(R|S) \wedge vb\in\mathbf{t\,pref}R \wedge u\in(\mathbf{out}(R|S))^* \wedge vb\in(\mathbf{out}R)^*)$$

$$\vee\ u\in\mathbf{t\,pref}R$$

$\Rightarrow\{$calc., (1)$\}$

$$u\in\mathbf{t\,pref}R.$$

For $\mathbf{hd}R\subseteq\mathbf{in}R \wedge \mathbf{hd}S\subseteq\mathbf{in}S$ we derive

$$u\in\mathbf{t\,pref}(R|S) \wedge vb\in\mathbf{t\,pref}R \wedge b\in\mathbf{out}R$$

$=\{$calc.$\}$

$$u\in\mathbf{t\,pref}(R|S) \wedge vb\in\mathbf{t\,pref}R \wedge (u{\restriction}\mathbf{out}(R|S)\neq\epsilon \vee vb{\restriction}\mathbf{out}R\neq\epsilon)$$

Similarly to the above derivation we then infer $u\in\mathbf{tpref}R$.
The second part of the theorem is proved similarly.
□

LEMMA B.28. *For n.e. trace structures R and S, with Alfcond(R,S), Altcond*1(R,S), $\mathbf{hd}(R)\subseteq\mathbf{out}R \wedge \mathbf{hd}(S)\subseteq\mathbf{out}S$, *and* $\mathbf{pref}R$ *and* $\mathbf{pref}S$ *satisfy rule g3, we have*

$$v\in\mathbf{t\,pref}R \wedge u\in\mathbf{t\,pref}(R|S)$$

$$\wedge\ (v{\restriction}\mathbf{in}R\neq\epsilon \vee u{\restriction}\mathbf{in}(R|S)\neq\epsilon)$$

$$\wedge\ u{\restriction}\mathbf{ext}(R|S)=v{\restriction}\mathbf{ext}(R|S)$$

$$\Rightarrow\ u\in\mathbf{t\,pref}R.$$

A similar lemma also holds with **out** *and* **in** *replaced by* **in** *and* **out** *respectively.*

PROOF. Let R and S be n.e. trace structures with *Alfcond*$(R,S) \wedge$ *Altcond*1(R,S), $\mathbf{hd}(R)\subseteq\mathbf{out}R \wedge \mathbf{hd}(S)\subseteq\mathbf{out}S$, $\mathbf{pref}R$ and $\mathbf{pref}S$ satisfy rule $g3$, and $u{\restriction}\mathbf{ext}(R|S)=v{\restriction}\mathbf{ext}(R|S)$. We prove (*i*) $v \in \mathbf{t\,pref}R \wedge u \in \mathbf{t\,pref}S \wedge v{\restriction}\mathbf{in}R \neq \epsilon \Rightarrow \mathit{false}.$

For reasons of symmetry, we can then also conclude

(*ii*) $v \in \mathbf{t\,pref}R \wedge u \in \mathbf{t\,pref}S \wedge u{\restriction}\mathbf{in}S \neq \epsilon \Rightarrow \mathit{false}.$

Derivation (*i*) and (*ii*) combined with

(*iii*) $u \in \mathbf{t\,pref}S \wedge \mathit{Alfcond}(R,S) \Rightarrow u{\restriction}\mathbf{in}(R|S) = u{\restriction}\mathbf{in}S.$

yields

$$v \in \mathbf{t\,pref}R \wedge u \in \mathbf{t\,pref}(R|S)$$
$$\wedge\ (v{\restriction}\mathbf{in}R \neq \epsilon \vee u{\restriction}\mathbf{in}(R|S) \neq \epsilon)$$
$$\Rightarrow\{(i), (ii), (iii), \text{calc.}\}$$
$$u \in \mathbf{t\,pref}R.$$

We proceed with the proof of (*i*).

$$v \in \mathbf{t\,pref}R \wedge v{\restriction}\mathbf{in}R \neq \epsilon \wedge \mathbf{hd}(R) \subseteq \mathbf{out}R \wedge u \in \mathbf{t\,pref}S$$
$$\Rightarrow\{\text{Lemma B.29 , } \mathbf{pref}R \text{ satisfies rule } g3\}$$
$$(\mathbf{E}r : \mathit{set}(r) \in \mathbf{first1}R$$
$$: r \preccurlyeq v{\restriction}\mathbf{ext}R \wedge r \in (\mathbf{o}R)^* \wedge r \neq \epsilon \wedge v \in \mathbf{t\,pref}R \wedge u \in \mathbf{t\,pref}S)$$
$$\Rightarrow\{u{\restriction}\mathbf{ext}(R|S) = v{\restriction}\mathbf{ext}(R|S),\ \mathit{Alfcond}(R,S), \text{calc.}\}$$
$$(\mathbf{E}r : \mathit{set}(r) \in \mathbf{first1}R$$
$$: r \preccurlyeq v{\restriction}\mathbf{ext}R \wedge r \in (\mathbf{o}R)^* \wedge r \neq \epsilon \wedge v{\restriction}\mathbf{ext}R = u{\restriction}\mathbf{ext}S \wedge u \in \mathbf{t\,pref}S)$$
$$\Rightarrow\{\mathit{Alfcond}(R,S), \text{calc.}\}$$
$$(\mathbf{E}r : \mathit{set}(r) \in \mathbf{first1}R$$
$$: r \in \mathbf{t\,pref}S{\restriction}\mathbf{ext}S \wedge r \in (\mathbf{o}S)^* \wedge r \neq \epsilon)$$
$$\Rightarrow\{\mathit{Altcond1}(R,S) \Rightarrow \mathit{fprop1}(S)\}$$
$$(\mathbf{E}r,s : \mathit{set}(r) \in \mathbf{first1}R \wedge \mathit{set}(s) \in \mathbf{first1}S$$
$$: (r \preccurlyeq s \vee s \preccurlyeq r) \wedge r \neq \epsilon)$$
$$\Rightarrow\{\mathit{Altcond1}(R,S) \Rightarrow \mathit{llcond1}(R,S), \text{calc.}\}$$
$$\mathit{false}.$$

□

LEMMA B.29. *If for a n.e. trace structure R,* **pref***R satisfies rule g3, then*

$$v \in \mathbf{t\,pref}R \wedge v{\restriction}\mathbf{in}R \neq \epsilon \wedge \mathbf{hd}(R) \subseteq \mathbf{out}R$$
$$\Rightarrow(\mathbf{E}r : \mathit{set}(r) \in \mathbf{first1}R : r \preccurlyeq v{\restriction}\mathbf{ext}R \wedge r \in (\mathbf{o}R)^* \wedge r \neq \epsilon).$$

PROOF. Let R be a n.e. trace structure and **pref**R satisfies rule $g3$. We observe

$$v \in \mathbf{t\,pref}R \wedge v \upharpoonright \mathbf{in}R \neq \epsilon \wedge \mathbf{hd}(R) \subseteq \mathbf{out}R$$

$\Rightarrow$ {calc.}

$$(\mathbf{E}r :: r \in (\mathbf{out}R)^* \wedge r \preccurlyeq v \wedge Suc(r,R) \setminus \mathbf{out}R \neq \varnothing \wedge v \in \mathbf{t\,pref}R)$$

$\Rightarrow$ {**pref**R satisfies rule $g3$, $\mathbf{hd}(R) \subseteq \mathbf{out}R$, see below}

$$(\mathbf{E}r :: r \in (\mathbf{out}R)^* \wedge r \preccurlyeq v \wedge Suc(r,R) \setminus \mathbf{out}R \neq \varnothing \wedge v \in \mathbf{t\,pref}R$$
$$\wedge\ r \upharpoonright \mathbf{o}R \neq \epsilon)$$

$\Rightarrow$ {$\mathbf{hd}(R) \subseteq \mathbf{out}R$, def. of **first1**$R$}

$$(\mathbf{E}r : set(r) \in \mathbf{first1}R : r \preccurlyeq v \upharpoonright \mathbf{ext}R \wedge r \in (\mathbf{o}R)^* \wedge r \neq \epsilon).$$

Let $rb \in \mathbf{t\,pref}R \wedge r \in (\mathbf{out}R)^* \wedge b \notin \mathbf{out}R$. We have $b \notin \mathbf{out}R \Rightarrow b \in \mathbf{in}R$. If $r \in (\mathbf{co}R)^*$, then it follows, with rule $g3$ for **pref**R, that $br \in \mathbf{t\,pref}R$ as well, contradicting $\mathbf{hd}R \subseteq \mathbf{out}R$. Consequently, $r \upharpoonright \mathbf{o}R \neq \epsilon$.
□

PROOF OF THEOREM B.8.1. Let R and S be n.e. trace structures such that $Disfree(R)$, $Disfree(S)$, and $Alfcond(R,S)$ hold. Let, furthermore, $Seqcond(R,S)$ hold and R and $R \upharpoonright \mathbf{ext}R$ be prefix-free. We prove $Disfree(R;S)$ by considering two cases corresponding to how a trace vb can be parsed as a member of $\mathbf{t\,pref}(R;S)$, viz.
(*i*) $vb \in \mathbf{t\,pref}R$ or
(*ii*) $vb \notin \mathbf{t\,pref}R \wedge vb \in \mathbf{t\,pref}(R;S)$.
We observe
(*i*)

$$u \in \mathbf{t\,pref}(R;S) \wedge vb \in \mathbf{t\,pref}R \wedge b \in \mathbf{out}(R;S)$$
$$\wedge\ u \upharpoonright (\mathbf{ext}(R;S) \cup \mathbf{co}(R;S)) = v \upharpoonright (\mathbf{ext}(R;S) \cup \mathbf{co}(R;S))$$

$\Rightarrow$ {calc.}

$$u \in \mathbf{t\,pref}(R;S) \wedge v \in \mathbf{t\,pref}R \wedge vb \in \mathbf{t\,pref}R \wedge b \in \mathbf{out}(R;S)$$
$$\wedge\ u \upharpoonright (\mathbf{ext}(R;S) \cup \mathbf{co}(R;S)) = v \upharpoonright (\mathbf{ext}(R;S) \cup \mathbf{co}(R;S))$$

$\Rightarrow$ {Lemma B.30 (*i*), $Disout(R)$, R and $R \upharpoonright \mathbf{ext}R$ are prefix-free,
$Alfcond(R,S)$, take $t_0,t_1 := u,v$}

$$(\mathbf{E}r_0,s_0 :: u = r_0 s_0 \wedge vb \in \mathbf{t\,pref}R \wedge b \in \mathbf{out}(R;S)$$
$$\wedge\ r_0 \in \mathbf{t\,pref}R \wedge s_0 \in \mathbf{t\,pref}S \wedge (s_0 = \epsilon \vee r_0 \in \mathbf{t}R)$$
$$\wedge\ r_0 \upharpoonright (\mathbf{ext}R \cup \mathbf{co}R) = v \upharpoonright (\mathbf{ext}R \cup \mathbf{co}R)\quad)$$

$\Rightarrow$ {$Disout(R)$, calc.}

$(\mathbf{E}r_0,s_0 :: u=r_0s_0 \wedge r_0b \in \mathbf{t}\,\mathbf{pref}R$

$\wedge\ s_0 \in \mathbf{t}\,\mathbf{pref}S \wedge (s_0=\epsilon \vee r_0 \in \mathbf{t}R)\ )$

$\Rightarrow\{R$ is prefix-free$\}$

$(\mathbf{E}r_0,s_0 :: u=r_0s_0 \wedge r_0b \in \mathbf{t}\,\mathbf{pref}R \wedge s_0 \in \mathbf{t}\,\mathbf{pref}S \wedge s_0=\epsilon)$

$\Rightarrow\{$calc.$\}$

$ub \in \mathbf{t}\,\mathbf{pref}R$

$\Rightarrow\{$calc.$\}$

$ub \in \mathbf{t}\,\mathbf{pref}(R;S)$.

(*ii*)

$u \in \mathbf{t}\,\mathbf{pref}(R;S) \wedge vb \notin \mathbf{t}\,\mathbf{pref}R \wedge vb \in \mathbf{t}\,\mathbf{pref}(R;S) \wedge b \in \mathbf{out}(R;S)$

$\wedge\ u\upharpoonright(\mathbf{ext}(R;S) \cup \mathbf{co}(R;S)) = v\upharpoonright(\mathbf{ext}(R;S) \cup \mathbf{co}(R;S))$

$\Rightarrow\{$calc.$\}$

$u \in \mathbf{t}\,\mathbf{pref}(R;S) \wedge v \in \mathbf{t}(R;\mathbf{pref}S) \wedge vb \in \mathbf{t}\,\mathbf{pref}(R;S) \wedge b \in \mathbf{out}(R;S)$

$\wedge\ u\upharpoonright(\mathbf{ext}(R;S) \cup \mathbf{co}(R;S)) = v\upharpoonright(\mathbf{ext}(R;S) \cup \mathbf{co}(R;S))$

$\Rightarrow\{$Lemma B.30 (*ii*), *Disout*(R), R and $R\upharpoonright\mathbf{ext}R$ are prefix-free, *Alfcond*(R,S), take $t_0,t_1 := u,v\}$

$(\mathbf{E}r_0,s_0,r_1,s_1 :: u=r_0s_0 \wedge v=r_1s_1$

$\wedge\ r_0 \in \mathbf{t}\,\mathbf{pref}R \wedge s_0 \in \mathbf{t}\,\mathbf{pref}S \wedge (r_0 \in \mathbf{t}R \vee s_0=\epsilon)$

$\wedge\ r_1 \in \mathbf{t}R \wedge s_1 \in \mathbf{t}\,\mathbf{pref}S \wedge vb \in \mathbf{t}\,\mathbf{pref}(R;S) \wedge b \in \mathbf{out}(R;S)$

$\wedge\ r_0\upharpoonright(\mathbf{ext}R \cup \mathbf{co}R) = r_1\upharpoonright(\mathbf{ext}R \cup \mathbf{co}R)$

$\wedge\ s_0\upharpoonright(\mathbf{ext}S \cup \mathbf{co}S) = s_1\upharpoonright(\mathbf{ext}S \cup \mathbf{co}S)\ \ )$

$\Rightarrow\{R$ is prefix-free, Lemma B.25.1, *Alfcond*$(R,S)\}$

$(\mathbf{E}r_0,s_0,r_1,s_1 :: u=r_0s_0 \wedge v=r_1s_1$

$\wedge\ r_0 \in \mathbf{t}\,\mathbf{pref}R \wedge s_0 \in \mathbf{t}\,\mathbf{pref}S \wedge (r_0 \in \mathbf{t}R \vee s_0=\epsilon)$

$\wedge\ r_1 \in \mathbf{t}R \wedge s_1 \in \mathbf{t}\,\mathbf{pref}S \wedge s_1b \in \mathbf{t}\,\mathbf{pref}S \wedge b \in \mathbf{out}S$

$\wedge\ r_0\upharpoonright(\mathbf{ext}R \cup \mathbf{co}R) = r_1\upharpoonright(\mathbf{ext}R \cup \mathbf{co}R)$

$\wedge\ s_0\upharpoonright(\mathbf{ext}S \cup \mathbf{co}S) = s_1\upharpoonright(\mathbf{ext}S \cup \mathbf{co}S)\ \ )$

$\Rightarrow\{$*Disout*$(S)\}$

$(\mathbf{E}r_0,s_0,r_1,s_1 :: u=r_0s_0 \wedge r_1 \in \mathbf{t}R \wedge b \in \mathbf{out}S$

$\wedge\ r_0 \in \mathbf{t}\,\mathbf{pref}R \wedge s_0b \in \mathbf{t}\,\mathbf{pref}S \wedge (r_0 \in \mathbf{t}R \vee s_0=\epsilon)$

$\wedge\ r_0\lceil(\mathbf{ext}R \cup \mathbf{co}R) = r_1\lceil(\mathbf{ext}R \cup \mathbf{co}R)\quad)$

$\Rightarrow$ {calc.}

$(\mathbf{E}r_0,s_0,r_1,s_1 :: u = r_0 s_0 \wedge r_1 \in \mathbf{t}R \wedge b \in \mathbf{out}S$

$\wedge\ r_0 \in \mathbf{t}\,\mathbf{pref}R \wedge s_0 b \in \mathbf{t}\,\mathbf{pref}S \wedge (r_0 \in \mathbf{t}R \vee (r_0 \notin \mathbf{t}R \wedge s_0 = \epsilon))$

$\wedge\ r_0\lceil(\mathbf{ext}R \cup \mathbf{co}R) = r_1\lceil(\mathbf{ext}R \cup \mathbf{co}R))$

$\Rightarrow$ {Lemma B.31, R and $R\lceil\mathbf{ext}R$ are prefix-free, *Disout*(R)}

$(\mathbf{E}r_0,s_0,r_1,s_1 :: u = r_0 s_0 \wedge r_0 \in \mathbf{t}\,\mathbf{pref}R \wedge s_0 b \in \mathbf{t}\,\mathbf{pref}S$

$\wedge\ (r_0 \in \mathbf{t}R \vee (\mathbf{tl}R \cap \mathbf{en}R \neq \varnothing \wedge \mathbf{hd}S \cap \mathbf{out}S \neq \varnothing)))$

$\Rightarrow$ {*Seqcond*(R,S)}

$(\mathbf{E}r_0,s_0 :: u = r_0 s_0 \wedge r_0 \in \mathbf{t}R \wedge s_0 b \in \mathbf{t}\,\mathbf{pref}S)$

$\Rightarrow$ {calc.}

$ub \in \mathbf{t}\,\mathbf{pref}(R;S)$.

For the proof of *Disin*$(R;S)$ a similar reasoning applies.

□

LEMMA B.30. *If R and $R\lceil\mathbf{ext}R$ are prefix-free and Disout(R) and Alfcond(R,S) hold, then for arbitrary traces t_0 and t_1 we have*

$t_0 \in \mathbf{t}\,\mathbf{pref}(R;S) \wedge t_1 \in \mathbf{t}\,\mathbf{pref}(R;S)$

$\wedge\ t_0\lceil(\mathbf{ext}(R;S) \cup \mathbf{co}(R;S)) = t_1\lceil(\mathbf{ext}(R;S) \cup \mathbf{co}(R;S))$

$\Rightarrow (\mathbf{E}r_0,s_0,r_1,s_1 :: t_0 = r_0 s_0 \wedge t_1 = r_1 s_1$

$\wedge\ r_0 \in \mathbf{t}\,\mathbf{pref}R \wedge s_0 \in \mathbf{t}\,\mathbf{pref}S \wedge (s_0 = \epsilon \vee r_0 \in \mathbf{t}R)$

$\wedge\ r_1 \in \mathbf{t}\,\mathbf{pref}R \wedge s_1 \in \mathbf{t}\,\mathbf{pref}S \wedge (s_1 = \epsilon \vee r_1 \in \mathbf{t}R)$

$\wedge\ r_0\lceil(\mathbf{ext}R \cup \mathbf{co}R) = r_1\lceil(\mathbf{ext}R \cup \mathbf{co}R)$

$\wedge\ s_0\lceil(\mathbf{ext}S \cup \mathbf{co}S) = s_1\lceil(\mathbf{ext}S \cup \mathbf{co}S)\quad)$.

Moreover, if
(i) $t_1 \in \mathbf{t}\,\mathbf{pref}R$, *then* $s_1 = \epsilon$
(ii) $t_1 \in \mathbf{t}(R;\mathbf{pref}S)$, *then* $r_1 \in \mathbf{t}R$.
A similar lemma holds with **co** *replaced by* **en** *and using Disin(R) instead of Disout(R) as a condition.*

PROOF. Let R and $R\lceil\mathbf{ext}R$ be prefix-free and *Disout*(R) and *Alfcond*(R,S) hold. Let furthermore

$t_0 \in \mathbf{t}\,\mathbf{pref}(R;S) \wedge t_1 \in \mathbf{t}\,\mathbf{pref}(R;S)$

$\wedge\ t_0\lceil(\mathbf{ext}(R;S) \cup \mathbf{co}(R;S)) = t_1\lceil(\mathbf{ext}(R;S) \cup \mathbf{co}(R;S))$.

By definition of concatenation we deduce

$$(\mathbf{E}r_0,s_0,r_1,s_1 :: t_0 = r_0 s_0 \ \wedge\ t_1 = r_1 s_1$$
$$\wedge\ r_0 \in \mathbf{t\,pref}R \ \wedge\ s_0 \in \mathbf{t\,pref}S \ \wedge\ (s_0 = \epsilon \ \vee\ r_0 \in \mathbf{t}R) \qquad (0)$$
$$\wedge\ r_1 \in \mathbf{t\,pref}R \ \wedge\ s_1 \in \mathbf{t\,pref}S \ \wedge\ (s_1 = \epsilon \ \vee\ r_1 \in \mathbf{t}R)).$$

We prove for r_0 and r_1 in (0) that $r_0 \upharpoonright (\mathbf{ext}R \cup \mathbf{co}R) = r_1 \upharpoonright (\mathbf{ext}R \cup \mathbf{co}R)$. Then we have $s_0 \upharpoonright (\mathbf{ext}S \cup \mathbf{co}S) = s_1 \upharpoonright (\mathbf{ext}S \cup \mathbf{co}S)$ as well, since

$$r_0 \upharpoonright (\mathbf{ext}R \cup \mathbf{co}R) = r_1 \upharpoonright (\mathbf{ext}R \cup \mathbf{co}R)$$
$$= \{r_0 \in \mathbf{t\,pref}R,\ r_1 \in \mathbf{t\,pref}R,\ \mathit{Alfcond}(R,S),\ \text{calc.}\}$$
$$r_0 \upharpoonright (\mathbf{ext}(R\,;S) \cup \mathbf{co}(R\,;S)) = r_1 \upharpoonright (\mathbf{ext}(R\,;S) \cup \mathbf{co}(R\,;S))$$
$$= \{r_0 s_0 \upharpoonright (\mathbf{ext}(R\,;S) \cup \mathbf{co}(R\,;S)) = r_1 s_1 \upharpoonright (\mathbf{ext}(R\,;S) \cup \mathbf{co}(R\,;S)),\ \text{calc.}\}$$
$$s_0 \upharpoonright (\mathbf{ext}(R\,;S) \cup \mathbf{co}(R\,;S)) = s_1 \upharpoonright (\mathbf{ext}(R\,;S) \cup \mathbf{co}(R\,;S))$$
$$= \{s_0 \in \mathbf{t\,pref}S,\ s_1 \in \mathbf{t\,pref}S,\ \mathit{Alfcond}(R,S),\ \text{calc.}\}$$
$$s_0 \upharpoonright (\mathbf{ext}S \cup \mathbf{co}S) = s_1 \upharpoonright (\mathbf{ext}S \cup \mathbf{co}S).$$

First, we infer

$$r_0 s_0 \upharpoonright (\mathbf{ext}(R\,;S) \cup \mathbf{co}(R\,;S)) = r_1 s_1 \upharpoonright (\mathbf{ext}(R\,;S) \cup \mathbf{co}(R\,;S))$$
$$\Rightarrow \{\text{calc.}\}$$
$$(r_0 \upharpoonright \mathbf{ext}R)(s_0 \upharpoonright \mathbf{ext}R) = (r_1 \upharpoonright \mathbf{ext}R)(s_1 \upharpoonright \mathbf{ext}R)$$
$$\Rightarrow \{R \upharpoonright \mathbf{ext}R \text{ is prefix-free, (0)}\}$$
$$r_0 \upharpoonright \mathbf{ext}R = r_1 \upharpoonright \mathbf{ext}R. \qquad (1)$$

We derive

$$r_0 \upharpoonright (\mathbf{ext}R \cup \mathbf{co}R) \prec r_1 \upharpoonright (\mathbf{ext}R \cup \mathbf{co}R) \qquad (2)$$
$$= \{r_0 \in \mathbf{t\,pref}R,\ r_1 \in \mathbf{t\,pref}R,\ \text{calc.}\}$$
$$(\mathbf{E}u,b :: r_0 \in \mathbf{t\,pref}R \ \wedge\ r_1 \in \mathbf{t\,pref}R \ \wedge\ ub \leqslant r_1 \ \wedge\ b \in (\mathbf{ext}R \cup \mathbf{co}R)$$
$$\wedge\ r_0 \upharpoonright (\mathbf{ext}R \cup \mathbf{co}R) = u \upharpoonright (\mathbf{ext}R \cup \mathbf{co}R))$$
$$= \{r_0 \upharpoonright \mathbf{ext}R = r_1 \upharpoonright \mathbf{ext}R,\ \text{cf. (1), calc.}\}$$
$$(\mathbf{E}u,b :: r_0 \in \mathbf{t\,pref}R \ \wedge\ r_1 \in \mathbf{t\,pref}R \ \wedge\ ub \leqslant r_1 \ \wedge\ b \in \mathbf{co}R$$
$$\wedge\ r_0 \upharpoonright (\mathbf{ext}R \cup \mathbf{co}R) = u \upharpoonright (\mathbf{ext}R \cup \mathbf{co}R)\ )$$
$$\Rightarrow \{\mathit{Disout}(R)\}$$
$$(\mathbf{E}b :: r_0 b \in \mathbf{t\,pref}R)$$
$$\Rightarrow \{R \text{ is prefix-free},\ t_0 = r_0 s_0 \ \wedge\ (s_0 = \epsilon \ \vee\ r_0 \in \mathbf{t}R),\ \text{cf. (0)}\}$$
$$t_0 = r_0$$

$\Rightarrow\{t_0\restriction(\mathbf{ext}(R;S)\cup\mathbf{co}(R;S))=t_1\restriction(\mathbf{ext}(R;S)\cup\mathbf{co}(R;S)),\ t_1=r_1s_1\}$

$r_0\restriction(\mathbf{ext}(R;S)\cup\mathbf{co}(R;S))=r_1s_1\restriction(\mathbf{ext}(R;S)\cup\mathbf{co}(R;S))$

$\Rightarrow\{\text{calc.}\}$

$r_0\restriction(\mathbf{ext}R\cup\mathbf{co}R)=r_1s_1\restriction(\mathbf{ext}R\cup\mathbf{co}R)$

$\Rightarrow\{\text{calc.}\}$

$r_0\restriction(\mathbf{ext}R\cup\mathbf{co}R)\succcurlyeq r_1\restriction(\mathbf{ext}R\cup\mathbf{co}R)$

$\Rightarrow\{(2)\}$

false.

For reasons of symmetry, we conclude

$$r_0\restriction(\mathbf{ext}R\cup\mathbf{co}R)=r_1\restriction(\mathbf{ext}R\cup\mathbf{co}R).$$

Finally, we observe that the properties (*i*) and (*ii*) follow from the property that R is prefix-free.
□

LEMMA B.31. *If R and $R\restriction\mathbf{ext}R$ are prefix-free and $Disout(R)$ holds, then*

$$r_1\in\mathbf{t}R\ \wedge\ r_0\in\mathbf{t}\,\mathbf{pref}R\ \wedge\ r_0\notin\mathbf{t}R$$
$$\wedge\ r_0\restriction(\mathbf{ext}R\cup\mathbf{co}R)=r_1\restriction(\mathbf{ext}R\cup\mathbf{co}R)$$
$$\Rightarrow\ \mathbf{tl}R\cap\mathbf{en}R\neq\varnothing.$$

Similarly, if $Disin(R)$ holds, then

$$r_1\in\mathbf{t}R\ \wedge\ r_0\in\mathbf{t}\,\mathbf{pref}R\ \wedge\ r_0\notin\mathbf{t}R$$
$$\wedge\ r_0\restriction(\mathbf{ext}R\cup\mathbf{en}R)=r_1\restriction(\mathbf{ext}R\cup\mathbf{en}R)$$
$$\Rightarrow\ \mathbf{tl}R\cap\mathbf{co}R\neq\varnothing.$$

PROOF. Let R and $R\restriction\mathbf{ext}R$ be prefix-free and $Disout(R)$ holds. Let furthermore $r_1\in\mathbf{t}R$ and $r_0\restriction(\mathbf{ext}R\cup\mathbf{co}R)=r_1\restriction(\mathbf{ext}R\cup\mathbf{co}R)$. We observe

$r_0\in\mathbf{t}\,\mathbf{pref}R\ \wedge\ r_0\notin\mathbf{t}R$

$\Rightarrow\{\text{calc.}\}$

$(\mathbf{E}u::r_0u\in\mathbf{t}R\ \wedge\ u\neq\epsilon)$

$\Rightarrow\{r_0\restriction(\mathbf{ext}R\cup\mathbf{co}R)=r_1\restriction(\mathbf{ext}R\cup\mathbf{co}R),\ \text{calc.}\}$

$(\mathbf{E}u::r_0u\in\mathbf{t}R\ \wedge\ u\neq\epsilon\ \wedge\ r_0\restriction\mathbf{ext}R=r_1\restriction\mathbf{ext}R)$

$\Rightarrow\{r_1\in\mathbf{t}R\Rightarrow r_1\restriction\mathbf{ext}R\in\mathbf{t}R\restriction\mathbf{ext}R,\ R\restriction\mathbf{ext}R\ \text{is prefix-free}\}$

$(\mathbf{E}u::r_0u\in\mathbf{t}R\ \wedge\ u\neq\epsilon\ \wedge\ u\restriction\mathbf{ext}R=\epsilon).$

If $u\restriction \mathbf{co}R \neq \epsilon$, let b be the first symbol in $u\restriction \mathbf{co}R$, i.e. for some v we have

$$r_0 u \in \mathbf{t}R \wedge r_0 vb \leqslant r_0 u \wedge v \in (\mathbf{en}R)^* \wedge b \in \mathbf{co}R$$
$$\Rightarrow \{r_0\restriction(\mathbf{ext}R \cup \mathbf{co}R) = r_1\restriction(\mathbf{ext}R \cup \mathbf{co}R),\ r_1 \in \mathbf{t}R,\ \text{calc.}\}$$
$$r_0 vb \in \mathbf{t}\,\mathbf{pref}R \wedge r_1 \in \mathbf{t}R \wedge b \in \mathbf{co}R$$
$$\wedge\ r_0 v\restriction(\mathbf{ext}R \cup \mathbf{co}R) = r_1\restriction(\mathbf{ext}R \cup \mathbf{co}R)$$
$$\Rightarrow \{Disout(R)\}$$
$$r_1 b \in \mathbf{t}\,\mathbf{pref}R \wedge r_1 \in \mathbf{t}R$$
$$\Rightarrow \{R \text{ is prefix-free}\}$$
$$\textit{false.}$$

Consequently, $u\restriction \mathbf{co}R = \epsilon$, and we find

$$(\mathbf{E}u :: r_0 u \in \mathbf{t}R \wedge u \neq \epsilon \wedge u \in (\mathbf{en}R)^*)$$
$$\Rightarrow \{\text{calc., def. of } \mathbf{tl}\}$$
$$\mathbf{tl}R \cap \mathbf{en}R \neq \varnothing.$$

The proof for the second part is done similarly.
□

PROOF OF THEOREM B.9. Generalization of the proof of Theorem B.8.0 to n trace structures, $n \geqslant 0$.
□

B.5. PROOFS OF THEOREMS B.10 THROUGH B.16

PROOF OF THEOREM B.10. We prove Theorem B.10.1. The proof of Theorem B.10.0 is similar. Let R and S be n.e. prefix-free trace structures, $\mathbf{pref}R$ and $\mathbf{pref}S$ satisfy rule $g3$, $R\restriction\mathbf{ext}R$ and $S\restriction\mathbf{ext}S$ are prefix-free, and $Altcond\,1(R,S) \wedge Alfcond(R,S)$ hold. We prove that $(R|S)\restriction\mathbf{ext}(R|S)$ is prefix-free, i.e.

$$rs \in \mathbf{t}(R|S)\restriction\mathbf{ext}(R|S) \wedge r \in \mathbf{t}(R|S)\restriction\mathbf{ext}(R|S) \Rightarrow s = \epsilon.$$

First we observe for $\epsilon \notin \mathbf{t}(R\restriction\mathbf{ext}R) \wedge \epsilon \notin \mathbf{t}(S\restriction\mathbf{ext}S)$

$$rs \in \mathbf{t}(R\restriction\mathbf{ext}R) \wedge r \in \mathbf{t}(S\restriction\mathbf{ext}S) \wedge r \neq \epsilon$$
$$\Rightarrow \{R \text{ and } S \text{ are prefix-free, calc.}\}$$
$$(\mathbf{E}r_0, s_0 :: r_0 \in \mathbf{t}R \wedge rs = r_0\restriction\mathbf{ext}R \wedge Suc(r_0, R) = \varnothing \wedge r \neq \epsilon$$
$$\wedge\ s_0 \in \mathbf{t}S \wedge r = s_0\restriction\mathbf{ext}S \wedge Suc(s_0, S) = \varnothing)$$

$\Rightarrow$ {$Altcond\,1(R,S) \Rightarrow$ **first1**R and **first1**S are defined, calc., **pref**R and **pref**S satisfy rule $g\,3$, Lemma B.32}

$(\mathbf{E}u,v: set(u) \in \mathbf{first1}R \wedge set(v) \in \mathbf{first1}S$

$: u \preccurlyeq rs \wedge u \neq \epsilon \wedge v \preccurlyeq r \wedge v \neq \epsilon\ )$

$\Rightarrow$ {calc.}

$(\mathbf{E}u,v: set(u) \in \mathbf{first1}R \wedge set(v) \in \mathbf{first1}S$

$: (u \preccurlyeq v \vee v \preccurlyeq u) \wedge u \neq \epsilon \wedge v \neq \epsilon\ )$

$\Rightarrow$ {$Altcond\,1(R,S) \Rightarrow llcond\,1(R,S)$, calc.}

false.

Similarly, $r \in \mathbf{t}(R \upharpoonright \mathbf{ext}R) \wedge rs \in \mathbf{t}(S \upharpoonright \mathbf{ext}S) \Rightarrow false$. Consequently, we infer

$rs \in \mathbf{t}(R|S) \upharpoonright \mathbf{ext}(R|S) \wedge r \in \mathbf{t}(R|S) \upharpoonright \mathbf{ext}(R|S) \wedge r \neq \epsilon$

$\Rightarrow$ {$Alfcond(R,S)$, calc.}

$(rs \in \mathbf{t}R \upharpoonright \mathbf{ext}R \vee rs \in \mathbf{t}S \upharpoonright \mathbf{ext}S) \wedge (r \in \mathbf{t}R \upharpoonright \mathbf{ext}R \vee r \in \mathbf{t}S \upharpoonright \mathbf{ext}S) \wedge r \neq \epsilon$

$\Rightarrow$ {see above, calc.}

$(rs \in \mathbf{t}R \upharpoonright \mathbf{ext}R \wedge s \in \mathbf{t}R \upharpoonright \mathbf{ext}R) \vee (rs \in \mathbf{t}S \upharpoonright \mathbf{ext}S \wedge r \in \mathbf{t}S \upharpoonright \mathbf{ext}S) \wedge r \neq \epsilon$

$\Rightarrow$ {$R \upharpoonright \mathbf{ext}R$ and $S \upharpoonright \mathbf{ext}S$ are prefix-free}

$s = \epsilon$.

For $\epsilon \in \mathbf{t}S \upharpoonright \mathbf{ext}S \vee \epsilon \in \mathbf{t}R \upharpoonright \mathbf{ext}R$, we observe

$\epsilon \in \mathbf{t}S \upharpoonright \mathbf{ext}S$

$=$ {$S \upharpoonright \mathbf{ext}S$ is prefix-free}

$\{\epsilon\} = \mathbf{t}S \upharpoonright \mathbf{ext}S$

$=$ {**pref**S satisfies rule $g\,3$, S is prefix-free and n.e., Th. B.16.1, calc.}

$\mathbf{first1}S = \{\varnothing\}$

$=$ {$Altcond\,1(R,S)$, R and S are n.e.}

$\mathbf{first1}R = \{\varnothing\}$

$=$ {**pref**R satisfies rule $g\,3$, R is prefix-free and n.e., Th. B.16.1, calc.}

$\{\epsilon\} = \mathbf{t}R \upharpoonright \mathbf{ext}R$

$=$ {$R \upharpoonright \mathbf{ext}R$ is prefix-free}

$\epsilon \in \mathbf{t}R \upharpoonright \mathbf{ext}R$.

Accordingly, $\epsilon \in \mathbf{t}S \upharpoonright \mathbf{ext}S \vee \epsilon \in \mathbf{t}R \upharpoonright \mathbf{ext}R \Rightarrow \mathbf{t}(R|S) \upharpoonright \mathbf{ext}(R|S) = \{\epsilon\}$, and we conclude that $(R|S) \upharpoonright \mathbf{ext}(R|S)$ is prefix-free.

LEMMA B.32. *If, for a n.e. trace structure R,* **first1**R *is defined and* **pref**R *satisfies rule g3, then*

$$r \in \mathbf{t}R \ \wedge \ Suc(r,R) = \varnothing \ \wedge \ r \restriction \mathbf{ext}R \neq \epsilon$$
$$\Rightarrow (\mathbf{E}u : set(u) \in \mathbf{first1}R : u \preccurlyeq r \restriction \mathbf{ext}R \ \wedge \ u \neq \epsilon).$$

PROOF. Since **first1**R is defined, we may assume **hd**$R \subseteq$ **out**R. We observe

$$r \in \mathbf{t}R \ \wedge \ Suc(r,R) = \varnothing \ \wedge \ r \restriction \mathbf{ext}R \neq \epsilon$$
$$\Rightarrow \{\text{calc., } \mathbf{hd}R \subseteq \mathbf{out}R\}$$
$$(\mathbf{E}u :: r \in \mathbf{t}\,\mathbf{pref}R \ \wedge \ u \preccurlyeq r \ \wedge \ u \in (\mathbf{out}R)^*$$
$$\wedge \ (Suc(u,R) \setminus \mathbf{out}R \neq \varnothing \ \vee \ Suc(u,R) = \varnothing)$$
$$) \wedge \ r \restriction \mathbf{ext}R \neq \epsilon$$
$$\Rightarrow \{\mathbf{hd}R \subseteq \mathbf{out}R, \ \mathbf{pref}R \text{ satisfies rule } g3, \text{ see below}\} \qquad (0)$$
$$(\mathbf{E}u :: u \in \mathbf{t}\,\mathbf{pref}\,R \ \wedge \ u \preccurlyeq r \ \wedge \ u \restriction \mathbf{ext}R \neq \epsilon \ \wedge \ u \in (\mathbf{out}R)^*$$
$$\wedge \ (Suc(u,R) \setminus \mathbf{out}R \neq \varnothing \ \vee \ Suc(u,R) = \varnothing)$$
$$)$$
$$\Rightarrow \{\text{def. of } \mathbf{first1}R, \text{ calc.}\}$$
$$(\mathbf{E}u : set(u) \in \mathbf{first1}R : u \preccurlyeq r \restriction \mathbf{ext}R \ \wedge \ u \neq \epsilon).$$

Step (0) in the above derivation follows from the derivation below.

$$u \preccurlyeq r \ \wedge \ u \in (\mathbf{out}R)^* \ \wedge \ u \restriction \mathbf{ext}R = \epsilon \ \wedge \ r \restriction \mathbf{ext}R \neq \epsilon \ \wedge \ r \in \mathbf{t}\,\mathbf{pref}R$$
$$\Rightarrow \{\text{calc.}\}$$
$$u \preccurlyeq r \ \wedge \ u \in (\mathbf{co}R)^* \ \wedge \ u \restriction \mathbf{ext}R = \epsilon \ \wedge \ r \restriction \mathbf{ext}R \neq \epsilon \ \wedge \ r \in \mathbf{t}\,\mathbf{pref}R$$
$$\Rightarrow \{Suc(u,R) \setminus \mathbf{out}R \neq \varnothing \ \vee \ Suc(u,R) = \varnothing, \text{ calc.}\}$$
$$Suc(u,R) \setminus \mathbf{out}R \neq \varnothing \ \wedge \ u \in (\mathbf{co}R)^*$$
$$\Rightarrow \{\text{calc.}\}$$
$$(\mathbf{E}b : b \in \mathbf{in}R : ub \in \mathbf{t}\,\mathbf{pref}\,R \ \wedge \ u \in (\mathbf{co}R)^*)$$
$$\Rightarrow \{\mathbf{pref}R \text{ satisfies rule } g3\}$$
$$(\mathbf{E}b : b \in \mathbf{in}R : bu \in \mathbf{t}\,\mathbf{pref}\,R)$$
$$\Rightarrow \{\mathbf{hd}R \subseteq \mathbf{out}R\}$$
$$\mathit{false}.$$

Hence, implication (0) holds.

For **hd**$R \subseteq$ **in**R a similar proof applies.

□

PROOF OF THEOREM B.11.0. Let R and S be n.e. prefix-free trace structures. We observe for arbitrary traces r and s

$$rs \in \mathbf{t}(R;S) \wedge r \in \mathbf{t}(R;S)$$

$$\Rightarrow \{\text{calc.}\}$$

$$(\mathbf{E}r_0,s_0 :: r = r_0 s_0 \wedge r_0 \in \mathbf{t}R \wedge s_0 \in \mathbf{t}S \wedge r_0 s_0 s \in \mathbf{t}(R;S))$$

$$\Rightarrow \{R \text{ is prefix-free }\}$$

$$(\mathbf{E}s_0 :: s_0 \in \mathbf{t}S \wedge s_0 s \in \mathbf{t}S)$$

$$\Rightarrow \{S \text{ is prefix-free}\}$$

$$s = \epsilon.$$

Consequently, $R;S$ is prefix-free.
□

PROOF OF THEOREM B.11.1. Let R and S be n.e. trace structures, $R \restriction \mathbf{ext}R$ and $S \restriction \mathbf{ext}S$ are prefix-free, and $Alfcond(R,S)$ holds. We observe

$$(R;S) \restriction \mathbf{ext}(R;S)$$

$$= \{Alfcond(R,S), \text{ calc.}\}$$

$$(R \restriction \mathbf{ext}R);(S \restriction \mathbf{ext}S).$$

Subsequently, by Theorem B.11.0 we immediately derive that $(R \restriction \mathbf{ext}R);(S \restriction \mathbf{ext}S)$ is prefix-free, and so $(R;S) \restriction \mathbf{ext}(R;S)$ is prefix-free as well.
□

PROOF OF THEOREM B.13. Let R and S be n.e. trace structures with $Alfcond(R,S) \wedge Altcond0(R,S)$. Since $Altcond0(R,S)$ holds, we may assume $\mathbf{hd}R \subseteq \mathbf{out}R \wedge \mathbf{hd}S \subseteq \mathbf{out}S$ and $\mathbf{first0}(R|S)$ is defined. For $\mathbf{hd}R \subseteq \mathbf{in}R \wedge \mathbf{hd}S \subseteq \mathbf{in}S$ the proof is similar.

(*i*) We observe

$$t \in (\mathbf{o}(R|S))^* \wedge t \in \mathbf{t}\,\mathbf{pref}(R|S) \wedge t \neq \epsilon$$
$$\wedge\ (Suc(t,R|S) \setminus \mathbf{o}(R|S) \neq \varnothing \vee Suc(t,R|S) = \varnothing)$$

$$= \{\text{calc.}, Alfcond(R,S)\}$$

$$t \in (\mathbf{o}R \cup \mathbf{o}S)^* \wedge t \in \mathbf{t}\,\mathbf{pref}(R|S) \wedge t \neq \epsilon$$
$$\wedge\ (Suc(t,R) \setminus \mathbf{o}R \neq \varnothing \vee Suc(t,S) \setminus \mathbf{o}S \neq \varnothing$$
$$\vee\ (Suc(t,R) = \varnothing \wedge Suc(t,S) = \varnothing)\ )$$

$$= \{\text{calc.}\}$$

$$(t \in (\mathbf{o}R \cup \mathbf{o}S)^* \wedge t \in \mathbf{t}\,\mathbf{pref}(R|S) \wedge Suc(t,R) \setminus \mathbf{o}R \neq \varnothing \wedge t \neq \epsilon)$$
$$\vee\ (t \in (\mathbf{o}R \cup \mathbf{o}S)^* \wedge t \in \mathbf{t}\,\mathbf{pref}(R|S) \wedge Suc(t,S) \setminus \mathbf{o}S \neq \varnothing \wedge t \neq \epsilon)$$

$$\vee\ (t\in(\mathbf{o}R\cup\mathbf{o}S)^{*}\ \wedge\ t\in\mathbf{t}\,\mathbf{pref}(R|S)$$
$$\wedge\ Suc(t,R)=\varnothing\ \wedge\ Suc(t,S)=\varnothing\ \wedge\ t\neq\epsilon)$$
$$=\ \{\text{calc., } Alfcond(R,S),\ Altcond0(R,S),\ \text{cf. equivalence (0) below}\}$$
$$(t\in(\mathbf{o}R)^{*}\ \wedge\ t\in\mathbf{t}\,\mathbf{pref}R\ \wedge\ Suc(t,R)\setminus\mathbf{o}R\neq\varnothing\ \wedge\ t\neq\epsilon)$$
$$\vee\ (t\in(\mathbf{o}S)^{*}\ \wedge\ t\in\mathbf{t}\,\mathbf{pref}S\ \wedge\ Suc(t,S)\setminus\mathbf{o}S\neq\varnothing\ \wedge\ t\neq\epsilon)$$
$$\vee\ (t\in(\mathbf{o}R)^{*}\ \wedge\ t\in\mathbf{t}\,\mathbf{pref}R\ \wedge\ Suc(t,R)=\varnothing\ \wedge\ t\neq\epsilon)$$
$$\vee\ (t\in(\mathbf{o}S)^{*}\ \wedge\ t\in\mathbf{t}\,\mathbf{pref}S\ \wedge\ Suc(t,S)=\varnothing\ \wedge\ t\neq\epsilon)$$
$$=\ \{\text{calc.}\}$$
$$(t\in(\mathbf{o}R)^{*}\ \wedge\ t\in\mathbf{t}\,\mathbf{pref}R\ \wedge\ t\neq\epsilon$$
$$\wedge\ (Suc(t,R)\setminus\mathbf{o}R\neq\varnothing\ \vee\ Suc(t,R)=\varnothing))$$
$$\vee\ (t\in(\mathbf{o}S)^{*}\ \wedge\ t\in\mathbf{t}\,\mathbf{pref}S\ \wedge\ t\neq\epsilon$$
$$\wedge\ (Suc(t,S)\setminus\mathbf{o}S\neq\varnothing\ \vee\ Suc(t,S)=\varnothing))$$

The equivalence

$$t\in(\mathbf{o}R\cup\mathbf{o}S)^{*}\ \wedge\ t\in\mathbf{t}\,\mathbf{pref}(R|S)$$
$$\wedge\ Suc(t,R)=\varnothing\ \wedge\ Suc(t,S)=\varnothing\ \wedge\ t\neq\epsilon$$
$$=\ \{Alfcond(R,S),\ Altcond0(R,S)\} \qquad (0)$$
$$(t\in(\mathbf{o}R)^{*}\ \wedge\ t\in\mathbf{t}\,\mathbf{pref}R\ \wedge\ Suc(t,R)=\varnothing\ \wedge\ t\neq\epsilon)$$
$$\vee\ (t\in(\mathbf{o}S)^{*}\ \wedge\ t\in\mathbf{t}\,\mathbf{pref}S\ \wedge\ Suc(t,S)=\varnothing\ \wedge\ t\neq\epsilon)$$

follows from

$$t\in(\mathbf{out}R)^{*}\ \wedge\ t\in\mathbf{t}\,\mathbf{pref}R\ \wedge\ Suc(t,R)=\varnothing\ \wedge\ Suc(t,S)\neq\varnothing\ \wedge\ t\neq\epsilon$$
$$\Rightarrow\ \{\mathbf{hd}R\subseteq\mathbf{out}R,\ \text{def. of } \mathbf{first0}R,\ \text{calc.}\}$$
$$(\mathbf{E}r:set(r)\in\mathbf{first0}R:r\leqslant t)\ \wedge\ t\in\mathbf{t}\,\mathbf{pref}S\ \wedge\ Suc(t,S)\neq\varnothing\ \wedge\ t\neq\epsilon$$
$$\Rightarrow\ \{\mathbf{hd}S\subseteq\mathbf{out}S,\ \text{def. of } \mathbf{first0}S,\ \text{calc.}\}$$
$$(\mathbf{E}r,s:set(r)\in\mathbf{first0}R\ \wedge\ set(s)\in\mathbf{first0}S$$
$$:r\leqslant t\ \wedge\ (t\leqslant s\ \vee\ s\leqslant t)\ \wedge\ s\neq\epsilon)$$
$$\Rightarrow\ \{Altcond0(R,S)\ \Rightarrow\ llcond0(R,S)\}$$
$$\mathit{false},$$

and, similarly, with R and S interchanged. This gives the $\Leftarrow$-part of (0). The $\Rightarrow$-part is obvious.

(*ii*) Furthermore, we derive

$$\{\{b\}|b\in\mathbf{co}(R|S)\ \wedge\ b\in\mathbf{t}\,\mathbf{pref}(R|S)\}$$

$= \{\text{calc., } Alfcond(R,S)\}$

$\{\{b\} \mid b \in \mathbf{co}R \wedge b \in \mathbf{t}\,\mathbf{pref}R\}$

$\cup \{\{b\} \mid b \in \mathbf{co}S \wedge b \in \mathbf{t}\,\mathbf{pref}S\}.$

(*iii*) Third, we have

$$\mathbf{t}(R|S) = \{\epsilon\} \equiv \mathbf{t}R = \{\epsilon\} \wedge \mathbf{t}S = \{\epsilon\}.$$

From (*i*), (*ii*), and (*iii*), and the definition of **first0** we conclude

$$\mathbf{first0}(R|S) = \mathbf{first0}R \cup \mathbf{first0}S.$$

Similarly to (*i*) we prove for $\mathbf{first1}(R|S)$

$t \in (\mathbf{out}(R|S))^* \wedge t \in \mathbf{t}\,\mathbf{pref}(R|S)$

$\wedge (Suc(t,R|S) \setminus \mathbf{out}(R|S) \neq \varnothing \vee Suc(t,R|S) = \varnothing)$

$= \{Alfcond(R,S),\ Altcond0(R,S)\}$

$(t \in (\mathbf{out}R)^* \wedge t \in \mathbf{t}\,\mathbf{pref}R \wedge (Suc(t,R) \setminus \mathbf{out}R \neq \varnothing \vee Suc(t,R) = \varnothing))$

$\vee (t \in (\mathbf{out}S)^* \wedge t \in \mathbf{t}\,\mathbf{pref}S \wedge (Suc(t,S) \setminus \mathbf{out}S \neq \varnothing \vee Suc(t,S) = \varnothing)).$

Consequently, we have by definition of **first1**

$$\mathbf{first1}(R|S) = \mathbf{first1}R \cup \mathbf{first1}S.$$

□

PROOF OF THEOREM B.14.0. Let R and S be n.e. trace structures, with $\mathbf{hd}(R;S) \subseteq \mathbf{in}(R;S) \vee \mathbf{hd}(R;S) \subseteq \mathbf{out}(R;S)$. Hence, $\mathbf{first0}(R;S)$ is defined. Let furthermore, R be prefix-free and $Alfcond(R,S) \wedge Seqcond(R,S)$ hold.

If $\mathbf{t}R = \{\epsilon\}$, then $\mathbf{t}(R;S) = \mathbf{t}S$ and $\mathbf{first0}(R;S) = \mathbf{first0}S$, by $Alfcond(R,S)$.

If $\mathbf{t}R \neq \{\epsilon\}$, then it follows, since R is prefix-free, that $\epsilon \notin \mathbf{t}R$. We observe for $\mathbf{hd}(R;S) \subseteq \mathbf{out}(R;S)$

(*i*)

$t \in (\mathbf{o}(R;S))^* \wedge t \in \mathbf{t}\,\mathbf{pref}(R;S) \wedge t \neq \epsilon$

$\wedge (Suc(t,R;S) \setminus \mathbf{o}(R;S) \neq \varnothing \vee Suc(t,R;S) = \varnothing)$

$= \{Seqcond(R,S),\ Alfcond(R,S),\ \text{calc., } \epsilon \notin \mathbf{t}R,\ R \text{ and } S \text{ are n.e.}\}$

$t \in (\mathbf{o}R)^* \wedge t \in \mathbf{t}\,\mathbf{pref}R \wedge t \neq \epsilon$

$\wedge (Suc(t,R;S) \setminus \mathbf{o}(R;S) \neq \varnothing \vee Suc(t,R;S) = \varnothing)$

$= \{R \text{ is prefix-free, } Alfcond(R,S) \text{ calc.}\}$

$t \in (\mathbf{o}R)^* \wedge t \in \mathbf{t}\,\mathbf{pref}R \wedge t \neq \epsilon$

$\wedge (Suc(t,R) \setminus \mathbf{o}R \neq \varnothing \vee Suc(t,R) = \varnothing).$

(*ii*) Moreover,

$$b \in \mathbf{co}(R;S) \wedge b \in \mathbf{t}\,\mathbf{pref}(R;S)$$
$$= \{Alfcond(R,S),\ \epsilon \notin \mathbf{t}R,\ \text{calc.},\ R \text{ and } S \text{ are n.e.}\}$$
$$b \in \mathbf{co}R \wedge b \in \mathbf{t}\,\mathbf{pref}R.$$

(*iii*) Third, we have $\mathbf{t}(R;S)=\{\epsilon\} \equiv \mathbf{t}R=\{\epsilon\} \wedge \mathbf{t}S=\{\epsilon\}$.

From (*i*), (*ii*), and (*iii*) and the definition of **first0** we conclude

$$\mathbf{t}R \neq \{\epsilon\} \Rightarrow \mathbf{first0}(R;S)=\mathbf{first0}R.$$

For reasons of symmetry the theorem also holds for $\mathbf{hd}(R;S) \subseteq \mathbf{in}(R;S)$.
□

PROOF OF THEOREM B.14.1. Let R and S be n.e. trace structures with $\mathbf{hd}(R;S) \subseteq \mathbf{in}(R;S) \vee \mathbf{hd}(R;S) \subseteq \mathbf{out}(R;S)$, $\mathbf{pref}R$ satisfies rule $g3$, R and $R{\restriction}\mathbf{ext}R$ are prefix-free, and $Alfcond(R,S) \wedge Seqcond(R,S)$ hold.

First we consider $\mathbf{t}(R{\restriction}\mathbf{ext}R) \neq \{\epsilon\}$. Because $R{\restriction}\mathbf{ext}R$ is prefix-free, it follows $\epsilon \notin \mathbf{t}R{\restriction}\mathbf{ext}R$. We observe, assuming $\mathbf{hd}(R;S) \subseteq \mathbf{out}(R;S)$,

$$t \in (\mathbf{out}(R;S))^* \wedge t \in \mathbf{t}\,\mathbf{pref}(R;S)$$
$$\wedge\ (Suc(t,R;S) \setminus \mathbf{out}(R;S) \neq \varnothing \vee Suc(t,R;S)=\varnothing)$$
$$= \{Alfcond(R,S),\ Seqcond(R,S),\ \epsilon \notin \mathbf{t}R{\restriction}\mathbf{ext}R,\ \mathbf{pref}R \text{ sat. rule } g3,$$
$$\text{Lemma B.33}\}$$
$$t \in (\mathbf{out}R)^* \wedge t \in \mathbf{t}\,\mathbf{pref}R$$
$$\wedge\ (Suc(t,R;S) \setminus \mathbf{out}(R;S) \neq \varnothing \vee Suc(t,R;S)=\varnothing)$$
$$= \{R \text{ is prefix-free},\ Alfcond(R,S)\}$$
$$t \in (\mathbf{out}R)^* \wedge t \in \mathbf{t}\,\mathbf{pref}R$$
$$\wedge\ (Suc(t,R) \setminus \mathbf{out}R \neq \varnothing \vee Suc(t,R)=\varnothing).$$

Since $t \in \mathbf{t}\,\mathbf{pref}R \wedge Alfcond(R,S)$, we have $t{\restriction}\mathbf{ext}(R;S)=t{\restriction}\mathbf{ext}R$. Consequently, we conclude from the definition of **first1** and $Alfcond(R,S)$

$$R{\restriction}\mathbf{ext}R \neq \{\epsilon\} \Rightarrow \mathbf{first1}(R;S)=\mathbf{first1}R.$$

For $\mathbf{t}(R{\restriction}\mathbf{ext}R)=\{\epsilon\}$ we derive the following. Assume $\mathbf{hd}(R;S) \subseteq \mathbf{out}(R;S)$ again. Consequently, by $Alfcond(R,S)$, $\mathbf{hd}R \subseteq \mathbf{out}R$. We derive

$$\mathbf{t}(R{\restriction}\mathbf{ext}R)=\{\epsilon\}$$
$$= \{\text{calc.}\}$$
$$\mathbf{t}R \subseteq (\mathbf{int}R)^*$$
$$= \{\mathbf{pref}R \text{ satisfies rule 3},\ \mathbf{hd}R \subseteq \mathbf{out}R,$$
$$\text{shift first } b \in \mathbf{en}R \text{ in any } t \in \mathbf{t}R, \text{ if present, to beginning of } t\}$$

$$\mathbf{t}R \subseteq (\mathbf{co}R)^*$$

$$\Rightarrow \{Seqcond(R,S),\ \mathbf{hd}(R;S) \subseteq \mathbf{out}(R;S),\ Alfcond(R,S),\ \text{calc.}\}$$

$$\mathbf{hd}S \subseteq \mathbf{out}S \wedge \mathbf{first1}S \text{ is defined.}$$

Subsequently,

$$t \in (\mathbf{out}(R;S))^* \wedge t \in \mathbf{t}\,\mathbf{pref}(R;S)$$

$$\wedge\ (Suc(t,R;S) \setminus \mathbf{out}(R;S) \neq \varnothing \vee Suc(t,R;S) = \varnothing)$$

$$= \{\mathbf{t}R \subseteq (\mathbf{co}R)^*,\ Alfcond(R,S),\ \text{calc.}\}$$

$$(\mathbf{E}r,s :: t = rs \wedge r \in \mathbf{t}R \wedge s \in \mathbf{t}\,\mathbf{pref}S \wedge r \in (\mathbf{co}R)^* \wedge s \in (\mathbf{out}S)^*$$

$$\wedge\ (Suc(s,S) \setminus \mathbf{out}S \neq \varnothing \vee Suc(s,S) = \varnothing)$$

$$\wedge\ t \restriction \mathbf{ext}(R;S) = s \restriction \mathbf{ext}S\ ).$$

Hence, from the definition of **first1**$(R;S)$ and **first1**S we infer

$$\mathbf{t}(R \restriction \mathbf{ext}R) = \{\epsilon\} \Rightarrow \mathbf{first1}(R;S) = \mathbf{first1}S.$$

For $\mathbf{hd}(R;S) \subseteq \mathbf{in}(R;S)$ a similar proof applies.
□

LEMMA B.33. *For n.e. trace structures R and S with* $Alfcond(R,S) \wedge Seqcond(R,S) \wedge \epsilon \notin \mathbf{t}R \restriction \mathbf{ext}R \wedge \mathbf{pref}R$ *satisfies rule g3, we have*

$$t \in (\mathbf{out}(R;S))^* \wedge t \in \mathbf{t}\,\mathbf{pref}(R;S)$$

$$\Rightarrow t \in (\mathbf{out}R)^* \wedge t \in \mathbf{t}\,\mathbf{pref}R.$$

A similar lemma holds for **out** *replaced by* **in**.

PROOF. Let $t \in \mathbf{t}\,\mathbf{pref}(R;S)$ and $t \in (\mathbf{out}(R;S))^*$. We have either $t \in \mathbf{t}\,\mathbf{pref}R$ or $(\mathbf{E}r,s :: t = rs \wedge r \in \mathbf{t}R \wedge s \in \mathbf{t}\,\mathbf{pref}S \wedge s \neq \epsilon)$. We observe

$$(\mathbf{E}r,s :: t = rs \wedge r \in \mathbf{t}R \wedge s \in \mathbf{t}\,\mathbf{pref}S \wedge s \neq \epsilon)$$

$$\Rightarrow \{t \in (\mathbf{out}(R;S))^*,\ Alfcond(R,S),\ \epsilon \notin \mathbf{t}R \restriction \mathbf{ext}R,\ \text{calc.}\}$$

$$(\mathbf{E}r,s :: r \in \mathbf{t}R \wedge r \in (\mathbf{out}R)^* \wedge r \restriction \mathbf{o}R \neq \epsilon$$

$$\wedge\ s \in \mathbf{t}\,\mathbf{pref}S \wedge s \in (\mathbf{out}S)^* \wedge s \neq \epsilon$$

$$)$$

$$\Rightarrow \{\mathbf{pref}R \text{ satisfies rule } g3, \text{ shift symbol from } \mathbf{o}R \text{ in } r \text{ to the end of } r\}$$

$$\mathbf{tl}R \cap \mathbf{o}R \neq \varnothing \wedge \mathbf{hd}S \cap \mathbf{out}S \neq \varnothing$$

$$\Rightarrow \{Seqcond(R,S)\}$$

$$\mathit{false}.$$

Consequently, we derive

$t \in (\mathbf{out}(R;S))^* \wedge t \in \mathbf{t}\,\mathbf{pref}(R;S)$

$\Rightarrow$ {*Alfcond*(R,S), *Seqcond*(R,S), $\epsilon \notin \mathbf{t}R \upharpoonright \mathbf{ext}R$,

$\mathbf{pref}\,R$ satisfies rule $g3$, see above}

$t \in (\mathbf{out}R)^* \wedge t \in \mathbf{t}\,\mathbf{pref}\,R$.

□

PROOF OF THEOREM B.15. Let R be a prefix-free, n.e. trace structure with $\mathbf{hd}R \subseteq \mathbf{out}R \vee \mathbf{hd}R \subseteq \mathbf{in}R$. We derive

$t \in \mathbf{t}\,\mathbf{pref}\,R \wedge t \neq \epsilon$

$\Rightarrow$ {R is prefix-free}

$(\mathbf{E}r : r \in \mathbf{t}R : t \preccurlyeq r \wedge Suc(r,R) = \varnothing \wedge t \neq \epsilon)$

$\Rightarrow$ {Assume $\mathbf{hd}R \subseteq \mathbf{out}R$, calc.}

$(\mathbf{E}r : r \in \mathbf{t}\,\mathbf{pref}\,R$

$\quad : (t \preccurlyeq r \vee r \preccurlyeq t)$

$\quad \wedge ((r \in (\mathbf{o}R)^* \wedge r \neq \epsilon \wedge (Suc(r,R) \setminus \mathbf{o}R \neq \varnothing \vee Suc(r,R) = \varnothing))$

$\quad\quad \vee r \in \mathbf{co}R$

$\quad\quad)$

$)$

$=$ {def. of $\mathbf{first0}R$, which is defined since $\mathbf{hd}R \subseteq \mathbf{out}R \vee \mathbf{hd}R \subseteq \mathbf{in}R$}

$(\mathbf{E}r : set(r) \in \mathbf{first0}R : r \preccurlyeq t \vee t \preccurlyeq r)$.

Consequently, *fprop*$\,0(R)$ holds. For reasons of symmetry *fprop*$\,0(R)$ also holds for $\mathbf{hd}R \subseteq \mathbf{in}R$.

The property *fprop*$\,1(R)$ is proved similarly.

□

PROOF OF THEOREM B.16. Let R be a n.e. trace structure with $\mathbf{hd}R \subseteq \mathbf{in}R \vee \mathbf{hd}R \subseteq \mathbf{out}R$, hence $\mathbf{first0}R$ and $\mathbf{first1}R$ are defined. For the proof of B.16.1, we infer $\mathbf{t}R \upharpoonright \mathbf{ext}R = \{\epsilon\} \Rightarrow \mathbf{first1}R \subseteq \{\varnothing\}$, by definition of $\mathbf{first1}R$. Furthermore, we derive

$\mathbf{t}R \upharpoonright \mathbf{ext}R \neq \{\epsilon\}$

$\Rightarrow$ {R is prefix-free and n.e.}

$(\mathbf{E}r :: r \in \mathbf{t}R \wedge r \upharpoonright \mathbf{ext}R \neq \epsilon \wedge Suc(r,R) = \varnothing)$

$\Rightarrow$ {Lemma B.32, $\mathbf{first1}R$ is defined, $\mathbf{pref}\,R$ satisifies rule $g3$}

$(\mathbf{E}u :: set(u) \in \mathbf{first1}R \wedge u \neq \epsilon)$

$\Rightarrow$ {calc.}

$\neg(\mathbf{first1}R \subseteq \{\varnothing\})$.

For B.16.0 a similar proof applies.

□

REFERENCES

[0] G. BIRKHOFF, *Lattice Theory,* American Mathematical Society, Providence, 1967, (AMS Colloquium Publications: Vol. 25).

[1] A.W. BURKS et. al., Preliminary Discussion of the Logical Design of an Electronic Computing Instrument, (1946), in: [35].

[2] T.J. CHANEY, *A Comprehensive Bibliography on Synchronizers and Arbiters,* Technical Memorandum No. 306C, Institute for Biomedical Computing, Washington University, St. Louis.

[3] T.J. CHANEY and C.E. MOLNAR, Anomalous Behavior of Synchronizer and Arbiter Circuits, *IEEE Transactions on Computers,* Vol. C-22 (1973), pp. 421-422.

[4] W.A. CLARK, Macromodular Computer Systems, *Proceedings of the Spring Joint Computer Conference,* AFIPS, April 1967.

[5] W.A. CLARK and C.E. MOLNAR, Macromodular Computer Systems, *Computers in Biomedical Research,* Vol. IV, (R. STACY, and B. WAXMAN eds.), Academic Press, New York, 1974.

[6] EDSGER W. DIJKSTRA, Hierarchical Ordering of Sequential Processes, *Acta Informatica,* **1** (1971), pp. 115-138.

[7] EDSGER W. DIJKSTRA, Lecture Notes '*Predicate Transformers*' (Draft). Eindhoven University of Technology, 1982, (EWD835).

[8] EDSGER W. DIJKSTRA, W.H.J. FEIJEN and A.J.M. VAN GASTEREN, Derivation of a Termination Detection Algorithm for Distributed Computations, *Information Processing Letters,* **16** (1983), pp. 127-219.

[9] EDSGER W. DIJKSTRA and A.J.M. VAN GASTEREN, *On Notation,* Private Communication (AvG65a/EWD950a).

[10] DAVID DILL and EDMUND CLARKE, Automatic Verification of Asynchronous Circuits using Temporal Logic, *Proceedings 1985 Chapel Hill Conference on VLSI,* (H. FUCHS ed.), Computer Science Press, 1985, pp. 127-245.

[11] JO C. EBERGEN, *Trace Theory and the Design of Parallel Programs,* Personal Memorandum, January 1984.

[12] J. PRESPER ECKERT, JR., Types of Circuits - General, The Moore School Lectures, (1947), in: *Charles Babbage Institute Reprint Series for the History of Computing,* Vol. 9, (MARTIN CAMPBELL-KELLY and MICHAEL R. WILLIAMS eds.), 1985, MIT Press, pp. 179-191.

[13] T.P. FANG and C.E. MOLNAR, *Synthesis of Reliable Speed-Independent Circuit Modules: II. Circuit and Delay Conditions to Ensure Operation Free of Problems From Races and Hazards,* Technical Memorandum 298, Computer Systems Laboratory, Washington University, St. Louis, 1983.

[14] ROBERT W. FLOYD and JEFFREY D. ULLMAN, The Compilation of Regular Expressions into Integrated Circuits, *Journal of the ACM,* **29** (1982), pp. 603-622.

[15] L.A. HOLLAAR, Direct Implementation of Asynchronous Control Units, *IEEE Transactions on Computers,* Vol. C-31 (1982), pp. 1133-1141.

[16] ROB HOOGERWOORD, *Some Reflections on the Implementation of Trace Structures,* Computing Science Notes 86/03, Department of Mathematics and Computing Science, Eindhoven University of Technology, 1986.
[17] C.A.R. HOARE, Communicating Sequential Processes, *Communications of the ACM,* **21** (1978), pp. 666-677.
[18] C.A.R. HOARE, *Communicating Sequential Processes,* Prentice-Hall, 1985.
[19] M. HURTADO, *Dynamic Structure and Performance of Asymptotically Bistable Systems,* D. Sc. Dissertation, Washington University, St. Louis, 1975.
[20] ANNE KALDEWAIJ, *A Formalism for Concurrent Processes,* Ph.D. Thesis, Eindhoven University of Technology, 1986.
[21] ANNE KALDEWAIJ, The Translation of Processes into Circuits, in: *Proceedings PARLE, Parallel Architectures and Languages Europe,* Vol. 1, (J.W. DE BAKKER, A.J. NIJMAN and P.C. TRELEAVEN eds.), Springer-Verlag, 1987, pp. 195-213.
[22] R.M. KELLER, Towards a Theory of Universal Speed-Independent Modules, *IEEE on Transactions on Computers,* Vol. C-23, **1** (1974), pp. 21-33.
[23] ZVI KOHAVI, *Switching and Finite Automata Theory,* McGraw-Hill, 1970.
[24] L.R. MARINO, General Theory of Metastable Operation, *IEEE Transactions on Computers,* Vol. C-30, **2** (1981), pp. 107-115.
[25] ALAIN J. MARTIN, The Design of a Self-Timed Circuit for Distributed Mutual Exclusion, *Proceedings 1985 Chapel Hill Conference on VLSI,* (H. FUCHS ed.), Computer Science Press, 1985, pp. 247-260.
[26] ALAIN J. MARTIN, Compiling Communicating Processes into Delay-Insensitive VLSI Circuits, *Distributed Computing,* **1** (1986), pp. 226-234.
[27] J.W. MAUCHLY, Preparation of Problems for EDVAC-type Machines, (1947), in: [35].
[28] CARVER MEAD and MARTIN REM, Minimum Propagation Delays in VLSI, *IEEE Journal of Solid-State Circuits,* Vol. SC-17 (1982), pp. 773-775.
[29] R.E. MILLER, *Switching Theory,* Wiley, 1965.
[30] R.E. MILLER, Chapter 10 in: [29].
[31] D.P. MISUNAS, Petri-Nets and Speed-Independent Design, *Communications of the ACM,* **8** (1973), pp. 474-481.
[32] DAVID E. MULLER, A Theory of Asynchronous Circuits, *Proceedings of an International Symposioum on the Theory of Switching,* 1-5 April, 1957, Harvard University Press, Cambridge, Mass., (1959), pp. 204-243.
[33] C.E. MOLNAR, T.P. FANG and F.U. ROSENBERGER, Synthesis of Delay-Insensitive Modules, *Proceedings 1985, Chapel Hill Conference on VLSI,* (H. FUCHS ed.), Computer Science Press, 1985, pp.67-86.
[34] J. VON NEUMANN, First Draft of a Report on the EDVAC, (1945), in: [35].
[35] BRIAN RANDELL, ed., *The Origins of Digital Computers - Selected Papers,* Springer-Verlag, 1973.
[36] MARTIN REM, Concurrent Computations and VLSI Circuits, in: *Control Flow and Data Flow: Concepts of Distributed Computing,* (M. BROY ed.),

Springer-Verlag, 1985, pp. 399-437.

[37] MARTIN REM, Trace Theory and Systolic Computations, *Proceedings PARLE, Parallel Architectures and Languages Europe,* Vol. 1, (J.W. DE BAKKER, A.J. NIJMAN and P.C. TRELEAVEN eds.), Springer-Verlag, 1987, pp. 14-34.

[38] HUUB M.J.L. SCHOLS, *A Formalisation of the Foam Rubber Wrapper Principle,* Master's Thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology, 1985.

[39] HUUB M.J.L. SCHOLS and TOM VERHOEFF, *Delay-Insensitive Directed Trace Structures Satisfy the Foam Rubber Wrapper Postulate,* Computing Science Notes 85/04, Department of Mathematics and Computing Science, Eindhoven University of Technology, 1985.

[40] C.L. SEITZ, System Timing, in: CARVER MEAD AND LYNN CONWAY, *Introduction to VLSI Systems,* Addison-Wesley, 1980, pp. 218-262.

[41] CLAUDE E. SHANNON, A Symbolic Analysis of Relay and Switching Circuits, *Trans. AIEE,* **57,** 1938, pp. 731-723.

[42] JAN L.A. VAN DE SNEPSCHEUT, *Trace Theory and VLSI Design,* LNCS 200, Springer-Verlag, 1985.

[43] ALAN M. TURING, Proposal for Development in the Mathematics Division of an Automatic Computing Engine (ACE), (1946), in: *Charles Babbage Institute Reprint Series for the History of Computing,* Vol. 10, (B.E. CARPENTAR and R.W. DORAN eds.), MIT Press, 1986.

[44] ALAN M. TURING, Lecture to the London Mathematical Society on 20 February 1947, in: *Charles Babbage Institute Reprint Series for the History of Computing,* Vol. 10, (B.E. CARPENTAR and R.W. DORAN eds.), MIT Press, 1986.

[45] JAN TIJMEN UDDING, *Classification and Composition of Delay-Insensitive Circuits,* Ph.D. Thesis, Eindhoven University of Technology, 1984.

[46] JAN TIJMEN UDDING, A Formal Model for Defining and Classifying Delay-Insensitive Circuits and Systems, *Distributed Computing,* **1** (1986), pp. 197-204.

[47] S.H. UNGER, *Asynchronous Sequential Switching Circuits,* Wiley Interscience, 1969.

[48] TOM VERHOEFF, *Delay-Insensitive Codes - An Overview,* Computing Science Notes 87/04, Department of Mathematics and Computing Science, Eindhoven University of Technology, 1987.

[49] TOM VERHOEFF, *Three Families of Maximally Non-Deterministic Automata,* Computing Science Notes 87/10, Department of Mathematics and Computing Science, Eindhoven University of Technology, 1987.

Index

Acknowledgements

Many people have contributed directly or indirectly to this thesis. I would like to thank all of them, in particular Martin Rem, for having made all this possible and for his unfailing support through all these years; Charles Molnar, for his many valuable criticisms and fruitful discussions on all aspects related to delay-insensitivity; Huub Schols, for his scrutinizing reading of many drafts; Carolien Swagerman, for her expert typing; Jennifer Steiner, Steven Pemberton, Lambert Meertens, Joke Sterringa, and Tom Verhoeff, for pointing out the obscurities, misspellings, and other abuses of language in earlier drafts; CWI for providing the opportunity to work for four years on this thesis; the Eindhoven VLSI Club, during whose weekly gatherings the ideas presented took shape; and, finally, many thanks go to Marie Cecile Lavoir.

Samenvatting

In dit proefschrift wordt een methode beschreven voor het ontwerpen van vertragingsongevoelige circuits. Vertragingsongevoelige circuits zijn circuits waarvan het funktionele gedrag onafhankelijk is van vertragingen in de elementen waaruit het circuit is opgebouwd of in de verbindingsdraden daartussen. Om een aantal uiteenlopende redenen, waarop we in deze samenvatting niet verder ingaan, is er thans een toenemende belangstelling voor deze circuits.

We proberen zo'n circuit samen te stellen uit een eindig aantal basiselementen. We doen dit door een constructiemethode voor zo'n samenstelling te geven die is gebaseerd op het vertalen van programma's. Een programma specificeert het gewenste gedrag van een circuit. Niet elk programma komt in aanmerking voor deze vertaling; we gaan uit van programma's die aan een zekere syntax voldoen. Het resultaat van zo'n vertaling is dan een vertragingsongevoelige samenstelling van basiscomponenten die het gedrag realiseert zoals is gespecificeerd in het programma. Bovendien heeft de vertaling de eigenschap dat het totale aantal aantal basiselementen in de samenstelling evenredig is met de lengte van het programma.

De methode kan als volgt worden samengevat. We noemen een abstractie van een circuit een *component*; componenten worden gespecificeerd door programma's geschreven in een notatie die is gebaseerd op tracetheorie. In hoofdstuk 1 geven we een korte inleiding tot de tracetheorie.

De programma's noemen we *commands*. Ze kunnen worden beschouwd als een uitbreiding van de notatie voor reguliere expressies. Elke component gerepresenteerd door een command kan ook worden gerepresenteerd door middel van een reguliere expressie. De notatie voor commands staat echter een compactere representatie toe dankzij enkele extra programmeerprimitiva zoals operaties om parallellisme uit te drukken, staartrecursie (voor het representeren van eindige automaten), en projectie (voor het introduceren van interne symbolen). In hoofdstuk 2 geven we een aantal voorbeelden hoe een component gespecificeerd kan worden door middel van een command.

Gebaseerd op tracetheorie formaliseren we de begrippen *decompositie* en *vertragingsongevoeligheid* in hoofdstuk 3. De decompositie van een component representeert een realisatie van een component door een samenstelling van circuits. Het begrip vertragingsongevoeligheid is geformaliseerd in de definities *DI decompositie* en *DI component*. Een DI decompositie representeert een realisatie van een component door middel van een vertragingsongevoelige samenstelling van circuits. Een DI component representeert een circuit dat op een vertragingsongevoelige wijze met zijn omgeving communiceert. Een van de hoofdstellingen in dit proefschrift is dat DI decompositie en decompositie equivalent zijn als alle betrokken componenten DI componenten zijn.

Met behulp van de definitie van DI component ontwikkelen we in hoofdstuk 4 een aantal *DI grammatika's*. Dit zijn grammatika's waarvoor geldt dat elke command die hiermee gegenereerd kan worden een DI component voorstelt. Met behulp van deze grammatika's definiëren we de taal $\mathcal{L}_4$ van commands.

In hoofdstuk 5 en 6 laten we zien dat elke component gerepresenteerd door een command uit de taal $\mathcal{L}_4$ gedecomponeerd kan worden in een eindige verzameling van basiscomponenten. Deze decompositie kan worden beschreven als een syntax-gerichte vertaling van een command uit $\mathcal{L}_4$ naar commands van (DI) basiscomponenten. Bovendien is het aantal basiscomponenten in deze vertaling evenredig met de lengte van de command uit $\mathcal{L}_4$. Op deze manier hebben we op een formele wijze een constructiemethode gegeven voor vertragingsongvoelige circuits gerepresenteerd door commands uit $\mathcal{L}_4$.

In hoofdstuk 7 worden een aantal suggesties voor optimalisering van de decompositie methode behandeld.

Curriculum Vitae

Jo Ebergen werd geboren op 27 maart 1956 te Lith, Noord-Brabant. Na het behalen van het Atheneum B diploma in 1974 aan het Maasland College te Oss studeerde hij wiskunde aan de Technische Universiteit Eindhoven. Van januari tot medio maart 1982 verrichtte hij een stage onder leiding van prof. dr. C.L. Seitz aan het California Institute of Technology met als onderwerp 'Self-Timed Systems'. Zijn afstudeerwerk vond plaats onder leiding van prof. dr. E.W. Dijkstra en had als onderwerp 'Trace Theory and Self-Timed Systems'. In april 1983 behaalde hij het diploma wiskundig ingenieur en sinds mei 1983 is hij werkzaam op het Centrum voor Wiskunde en Informatica te Amsterdam aan het project VLSI ontwerp.