Models of trace theory systems

Models of Trace Theory Systems

Asia van de Mortel-Fronczak

Models of Trace Theory Systems

To my parents
and Peter

Models of Trace Theory Systems

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de Rector Magnificus, prof. dr. J.H. van Lint, voor een commissie aangewezen door het College van Dekanen in het openbaar te verdedigen op dinsdag 4 mei 1993 om 16.00 uur

door

Joanna Maria van de Mortel-Fronczak

geboren te Pińczów (Polen)

druk: wibro dissertatiedrukkerij, helmond.

Dit proefschrift is goedgekeurd
door de promotor
prof. dr. M. Rem
en de copromotor
dr. P.H.M. America

Acknowledgements

I would like to thank Martin Rem and Pierre America for their helpful remarks and valuable suggestions concerning both the contents and the form of this thesis.

Wim Kloosterhuis and Johan Lukkien are gratefully acknowledged for commenting on parts of the draft versions.

Thanks are also due to Anne Kaldewaij and Roland Backhouse for improving my English and suggesting some useful changes.

Finally, I thank Gerard Zwaan for his help with LaTeX.

Contents

Introduction

Over the last two decades a variety of mathematical formalisms for reasoning about parallel programs and systems have been proposed. A few well-known representatives are: the theory of Petri Nets (see [Re0]), CCS (Calculus of Communicating Systems, [Mi1]), CSP (Communicating Sequential Processes, [Hoa]), ACP (Algebra of Communicating Processes, [Ber]), the theory of temporal logic (see [Man]), the theory of Mazurkiewicz traces ([Ma0]), and the trace theory as developed at Eindhoven University of Technology (see [Rem]). An important and controversial issue in reasoning about parallel programs and systems is the issue of the interleaving approach versus the, so-called, true concurrency (also known as partial-order approach). In some theories (e.g., CCS, CSP, and ACP), execution of parallel actions a and b is modelled by interleaving: first a and then b, or first b and then a. In theories based on true concurrency (e.g., Petri Nets and Mazurkiewicz traces), a and b are considered as independent actions. Both approaches have an impact on liveness. When considering parallel actions in an interleaving model, fairness assumptions are often necessary to prove certain liveness properties. In partial-order models, fairness assumptions are not needed in this case.

The purpose of this thesis is to present a formalism for reasoning about parallel systems using the partial-order approach. The systems that we consider are described in terms of trace theory ([Rem], [Sne], and [Kal]). A system is a mathematical model of several mechanisms co-operating with each other in a parallel manner. Each mechanism is modelled by a trace theory process.

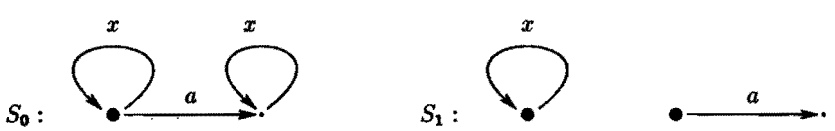
Systems are used for two purposes: to describe the desired behaviour of mechanisms we want to construct, and to describe the actual behaviour of existing mechanisms. The first descriptions are called specifications and the second ones are called implementations. To be able to formally verify whether a system is a correct implementation of (or simply — satisfies) a given specification, we introduce a pre-order relation on systems that is motivated by an operational view of functioning mechanisms. Establishing whether this relation holds between two systems is also referred to as comparing systems.

Internally a system may exhibit a complicated behaviour; for a user, however, only the visible (external) behaviour is of importance. In particular, the user is interested in a characterization that tells in which sequences of external actions the system may

engage (safety, also called possible external behaviour) and in which of them it must engage (liveness, also called guaranteed external behaviour). Within our formalism, these notions are defined in terms of transition systems that constitute the operational model of trace theory systems. System S_0 is then defined to be an implementation of specification S_1 if in every context, the possible external behaviour of S_0 is a subset of the possible external behaviour of S_1, and if the guaranteed external behaviour of S_1 is a subset of the guaranteed external behaviour of S_0. Parallel composition and projection are then shown to be monotonic with respect to the introduced relation on systems. This allows hierarchical designs in such a way that the verification of the composite can be done by verifying its components.

Subsequently, following the strategy of [Hen], an abstract model of systems is defined that provides, for a restricted class of systems, the same identifications and differentiations as in the operational model. An important feature of this model is compositionality with respect to the parallel composition operator and to the projection operator. However, the abstract model has a drawback of not being fully abstract with respect to the operational model.

A direct motivation for the research described in this thesis was the desire to adopt the failures model of [Hoa] to systems using the partial-order approach. In this way the undesired identifications that are made in the failures model would disappear. This is explained by means of a simple example. Consider two mechanisms: one that can choose between either executing an infinite sequence of internal (invisible for the environment) actions x, or executing a finite number of actions x and then external (visible for the environment) action a followed by infinitely many x's; and the other that consists of two independent parts, of which the first one executes an infinite sequence of x's and the second one executes a. The two mechanisms are specified by systems S_0 and S_1, respectively. Set $\{a\}$ is the external alphabet of both systems; S_0 consists of one process and S_1 consists of two processes running in parallel. Systems S_0 and S_1 are represented by the state graphs below



These two systems are identified in the failures model. This is not at all surprising, since in this mathematical model of systems parallelism is modelled by interleaving.

It turned out that it is difficult to adopt the failures model to systems in the intended way. The research resulted in a new model. In this model the two systems described above do not exhibit the same external behaviour. Namely, system S_1 is an implementation of system S_0, which in turn does not implement S_1. System S_0 does not satisfy

S_1 because action a will not happen if S_0 chooses to perform an infinite sequence of x's, whereas the occurrence of a is guaranteed by S_1.

As we assume that each process models a mechanism, we do not employ any fairness considerations in cases similar to system S_1. We do, however, introduce the possibility of treating S_0 as a correct implementation of S_1 under suitable fairness assumptions, viz., that S_0 is fair with respect to a. To achieve this, an extension of the abstract model with concepts of justice and fairness is proposed.

Several concepts and ideas known from the literature have influenced the research described in this thesis, specifically: safety and liveness ([Owi] and [Alp]), fairness ([Leh], [Fra], and [Paw]), the partial-order model of [Rel], the failures model of [Hoa], the testing relations of [Hen], and finally, the specification-oriented approach of [Hen] and [Old].

Both safety and liveness properties are associated with descriptions of the behaviour of systems. Safety properties define the maximal behaviour of a system in the sense that the system does nothing that is not allowed by its specification. Usually, the system that does nothing at all satisfies every specification in the above sense. As observed in [Owi], this concept corresponds to the notion of partial correctness of programs. Liveness properties define the minimal behaviour of a system in the sense that the system guarantees to perform the specified tasks. Program termination is an example of a liveness property ([Owi]). Both concepts are touched upon in this thesis.

The concept of fairness is connected to that of nondeterminism. Fairness properties can be seen as restrictions on infinite executions of systems: if there is a recurring choice out of a few alternatives, none of them should be constantly ignored. Fairness is a convenient abstraction from (irrelevant) implementation details. In [Lam] it has already been recognised that fairness affects liveness properties of systems (programs). Therefore, fairness assumptions should, in our opinion, be a part of a system description.

In [Rel] an operational model for CSP-like languages is proposed in which parallel commands are partially ordered, rather than interleaved (as, e.g., in [Plo]) or executed in a single step (as, e.g., in [Elr]). In, e.g., [Roz] and [Ber], in different settings, the two last approaches are combined in one model. [Rel] shows that a certain kind of unfair executions in interleaving models is due to expressing parallelism by means of interleaving.

The failures model of [Hoa] consists of three elements that are relevant to processes: the alphabet, the set of failures, and the set of divergences. A failure represents a sequence t of accepted communications together with a set of communications that can be refused after t. A divergence represents a sequence of communications after which the behaviour of the process is chaotic (e.g., when an infinite sequence of communications is concealed). In [Hoa], verifying whether a process meets a specification amounts

to proving that its traces, failures, and/or divergences satisfy specified conditions.

The testing relations of [Hen] serve to formally verify whether a system (process) satisfies a specification. They are based on the idea of an experimenter that interacts with the system. Since both the system and the specification are described in the same language, testing relations are simply (pre-order) relations on systems. Testing equivalence was proposed in [De1]. It can be regarded as a successor of observational equivalence of [Mil] and bisimulation equivalence of [Par] in the sense that it is another alternative behavioural equivalence for CCS.

Overview

In Chapter 1, we present a short recapitulation of that part of trace theory that forms the background for our study. This includes basic concepts and notions, followed by several useful results. Properties of processes and systems concerned with disabling and divergence are discussed. The failures model of systems is defined.

Chapter 2 focuses on an operational view of systems. In our approach, concurrency is not modelled by interleaving. Some new concepts are introduced, which are used later on for defining the testing (behavioural) relations. Moreover, some properties of the testing relations are proved. Specifically, parallel composition and projection are monotonic with respect to the implementation (**sat**) relation, and the equivalence relation induced by **sat** is a congruence with respect to parallel composition and to projection. Comparing systems in this setting involves calculations on a large domain of tests.

Chapter 3 provides an abstract model for comparing systems. This model is compositional with respect to the parallel composition operator and to the projection operator. For a restricted class of systems, this model is consistent with testing relations. In the abstract model, comparing systems is much simpler than by means of testing relations.

In Chapter 4, our abstract model is used to define a semantics of a subset of Tangram programs. (Tangram is a simple CSP-based programming language for describing VLSI designs.)

Finally, Chapter 5 considers incorporating fairness assumptions into the abstract model of systems. As an example, the Alternating Bit Protocol is discussed.

Each chapter contains several small examples illustrating notions and concepts introduced.

The Appendix contains some additional results in connection with the operational model of Chapter 2, including the comparison with an interleaving model, and an example considering three candidate implementations of a buffer.

Notation

In this thesis, the following notations are used.

For set A, we use A^* to denote the set of finite sequences of elements from A. If A is a set of symbols, A^* is the set of finite sequences of symbols from A. If A is a set of finite sequences of symbols, A^* is the set of finite concatenations of sequences from A.

For $a \in A$, the concatenation of i a's is denoted by a^i.

Numerical quantification $(\#x : P : Q)$ stands for "the number of x such that $P \wedge Q$".

For sets A and B, $A \div B = (A \cup B) \setminus (A \cap B)$.

To denote bags (multisets), symbols $\prec$ and $\succ$ are used. For instance, the bag with two a's and one b is denoted by $\prec a, a, b \succ$. For bag union and bag difference $+$ and $-$ operators are used, respectively.

Cardinality of set A is denoted by $|A|$.

The power set of A is denoted by $\mathcal{P}.A$.

Priorities of operators, from high to low:

- unary operators (like **a**, **t**, **p**, **ts**, **pr**, **c**),
- application dot,
- $p\cdot$, for symbol p,
- concatenation,
- projection ($\restriction$),
- binary operators ($\cup$, $\cap$, $\setminus$, $\times$, $+$, $-$, **w**, $\|$).

The proofs are presented in the style introduced in [Dij].

1 Basic concepts of trace theory

Trace theory provides our basic framework. An extended treatment of this theory can be found in [Sne] and [Kal]. In this chapter, we include only a few definitions and results that are used in this thesis.

In trace theory, processes are the main objects under consideration. A process is a mathematical model of a sequential (possibly nondeterministic) mechanism. A mechanism is represented by its repertoire of actions (also called events) and the set of finite sequences of actions that may be observed during its operation.

A derived notion is that of systems ([Klo], [Zwa]). In contrast to a process, a system is a mathematical model of several mechanisms co-operating with each other in a non-sequential (parallel) fashion. Therefore systems are our primary concern.

Composition of mechanisms is modelled by the composition of their corresponding systems, rather than their corresponding processes. Interaction (communication) between them is modelled by common actions.

Actions are assumed to be atomic, that is: they have no duration and they may happen either one after another or at the same time.

In Section 1.1, we present basic definitions and notions of trace theory, among them that of a process. For a class of regular processes an alternative way of specifying them is described. Fundamental results concerning processes and other related notions are mentioned. In Section 1.2, the notion of a system is introduced. Results concerning systems that are necessary for the rest of this thesis can be also found in this section. In Section 1.3, we discuss some properties of processes and systems that are associated with liveness. A special class of processes is also described here. Finally, Section 1.4 presents the failures model adjusted to systems in a non-compositional way. It still is an interleaving model.

1.1 Basic definitions and results

In trace theory, actions are modelled by symbols (names). We assume the existence of a universe Ω of symbols. Subsets of Ω are called *alphabets*.

A *trace* is an element of Ω^*. The empty trace is denoted by ε. Subsets of Ω^* are called

trace sets.

Throughout this thesis a, b, c, d, e, x, y, z (possibly primed) are used to denote elements of Ω, and s, t, u, v (possibly primed) are used to denote elements of Ω^* — unless explicitly stated otherwise.

The *length* of trace t is denoted by $\ell.t$, and is defined by

$$\begin{aligned} \ell.\varepsilon &= 0, \\ \ell.(ta) &= \ell.t + 1. \end{aligned}$$

The *concatenation* of traces t and u is denoted by tu. Concatenation has the highest priority of all operations on traces.

A frequently used operation is *projection*. For trace t and alphabet A, $t\restriction A$ denotes projection of t onto A (hiding of symbols that are not in A). Formally, for trace t and symbol a

$$\begin{aligned} \varepsilon\restriction A &= \varepsilon, \\ ta\restriction A &= t\restriction A, && \text{for } a \notin A, \\ ta\restriction A &= (t\restriction A)a, && \text{for } a \in A. \end{aligned}$$

In the sequel we abbreviate $t\restriction\{a\}$ to $t\restriction a$.

Another operation on traces that is used in this thesis is $\setminus$ (the same symbol is used to denote set difference). For trace t and bag (multiset) B of symbols, trace $t \setminus B$ is obtained by removing from t, from the left to the right, all occurrences of symbols that are in B, respecting their number in t. Formally, for trace t and symbol a

$$\begin{aligned} \varepsilon \setminus B &= \varepsilon, \\ at \setminus B &= a(t \setminus B), && \text{for } a \notin B, \\ at \setminus B &= t \setminus (B - \prec a \succ), && \text{for } a \in B. \end{aligned}$$

The *bag of symbols* of trace t is denoted by $\flat.t$. Formally, $\flat$ is defined by

$$\begin{aligned} \flat.\varepsilon &= \varnothing, \\ \flat.(at) &= \prec a \succ + \flat.t. \end{aligned}$$

In the bag of symbols of t, symbol a occurs $\ell.(t\restriction a)$ times.

For instance, for the trace $t = aabaac$, we have

$$\begin{aligned} &t \setminus \prec a, c \succ = abaa, \\ &t \setminus \prec a, a, c, c \succ = baa, \\ &\flat.t = \prec a, a, a, a, b, c \succ . \end{aligned}$$

In Sections 2.5 and 3.1 we refer to the following property ([Zwa]).

Property 1.1.1

For traces t, u, and v, and for alphabet A

- $(t \setminus\!\!\flat u)\restriction A = (t\restriction A) \setminus\!\!\flat (u\restriction A)$,
- $tu \setminus\!\!\flat (tv) = u \setminus\!\!\flat v$.

□

On traces relation $\leq$ is defined. Trace s is a *prefix* of trace t, denoted by $s \leq t$, if

$$(\exists u :: su = t).$$

We write $s < t$ if $s \leq t$ and $s \neq t$. Sometimes we use $t \geq s$ instead of $s \leq t$.
For set X of traces

$$pref.X = \{u \mid (\exists t : t \in X : u \leq t)\}.$$

Set $pref.X$ is also called the *prefix closure* of X. Trace set X is called *prefix-closed* if $X = pref.X$.
A *trace structure* is a pair $\langle A, X\rangle$, where A is an alphabet and $X \subseteq A^*$. Alphabet A is called the *alphabet* of the trace structure and X is called the *trace set* of the trace structure. For trace structure T, we denote its alphabet and its set of traces by $\mathbf{a}T$ and $\mathbf{t}T$, respectively.
The definition of projection is extended to sets of traces, sets of sets of traces, and trace structures. For set X of traces and alphabet A

$$X\restriction A = \{t\restriction A \mid t \in X\}.$$

For set $\mathcal{X}$ of sets of traces and alphabet A

$$\mathcal{X}\restriction A = \{X\restriction A \mid X \in \mathcal{X}\}.$$

For trace structure T and alphabet A, trace structure $T\restriction A$ is defined by

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

For trace structure T

$$pref.T = \langle \mathbf{a}T, pref.\mathbf{t}T\rangle.$$

Trace structure $pref.T$ is also called the *prefix closure* of T. Trace structure T is *prefix-closed* if $\mathbf{t}T$ is prefix-closed. Trace structure T is *non-empty* if $\mathbf{t}T \neq \emptyset$.

Trace structures are partially ordered by the inclusion relation $\subseteq$ defined as follows. For trace structures T and U

$$T \subseteq U \;\Leftrightarrow\; \mathbf{a}T = \mathbf{a}U \wedge \mathbf{t}T \subseteq \mathbf{t}U.$$

Intersection and union are defined for trace structures with equal alphabets. Let T and U be trace structures with the alphabet A. Then

$$T \cup U = \langle A, \mathbf{t}T \cup \mathbf{t}U \rangle \qquad \text{and} \qquad T \cap U = \langle A, \mathbf{t}T \cap \mathbf{t}U \rangle.$$

A *process* is defined as a non-empty and prefix-closed trace structure. The alphabet of process T represents the actions in which the (sequential) mechanism described by this process may participate. The trace set of process T denotes all its possible behaviours (sequences of actions).

In the following example, the process corresponding to a one-place buffer is defined.

Example 1.1.2

Consider a one-place buffer that is initially empty. The actions relevant for the buffer are

p: a value is put in the buffer,
r: a value is retrieved from the buffer.

Hence, the alphabet of process T that specifies this buffer is $\mathbf{a}T = \{p, r\}$.

In order to define the trace set, we first discuss when the actions may, and may not, take place. A value may be retrieved from the buffer only if the buffer is not empty, and no value can be put in the buffer if it already contains one. Additionally, as the buffer is initially empty, the trace set of T consists of the traces that do not start with an r and that are alternations of p and r. Formally,

$$t \in \mathbf{t}T \;\Leftrightarrow\; t \in \{p,r\}^* \wedge (\forall u : u \leq t : 0 \leq \ell.(u\restriction p) - \ell.(u\restriction r) \leq 1).$$

Examples of traces specified above are: ε, p, pr, prp, etc.

The complete formal definition of process T specifying the one-place buffer is then

$$T = \langle \{p,r\}, \{t \mid t \in \{p,r\}^* \wedge (\forall u : u \leq t : 0 \leq \ell.(u\restriction p) - \ell.(u\restriction r) \leq 1)\} \rangle.$$

□

Property 1.1.3 expresses the fact that projection of a process on an alphabet is again a process.

Property 1.1.3

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

□

For processes that are frequently used in trace theory names are introduced. These are: $\mathbf{stop}.A$, $\mathbf{run}.A$ and $\mathbf{sync}_{k,l}.A.B$, for alphabets A, B and natural numbers k, l.

For alphabet A, processes $\mathbf{stop}.A$ and $\mathbf{run}.A$ are defined by

$$\begin{aligned}\mathbf{stop}.A &= \langle A, \{\varepsilon\}\rangle,\\ \mathbf{run}.A &= \langle A, A^*\rangle.\end{aligned}$$

Process $\mathbf{stop}.\emptyset$ will also be denoted by $\mathbf{stop}$.

For alphabets A, B and natural numbers k, l (≥ 0), process $\mathbf{sync}_{k,l}.A.B$ is defined by

$$\begin{aligned}&\mathbf{sync}_{k,l}.A.B\\ &= \langle A \cup B, \{t \mid t \in (A \cup B)^* \wedge (\forall u : u \leq t : -l \leq \ell.(u\restriction A) - \ell.(u\restriction B) \leq k)\}\rangle.\end{aligned}$$

Process $\mathbf{sync}_{k,0}.A.B$ is also denoted by $\mathbf{buf}_k.A.B$.

Process $\mathbf{buf}.A.B$ is defined by

$$\mathbf{buf}.A.B = (\cup i : i \geq 0 : \mathbf{buf}_i.A.B).$$

When A or B are singletons we omit the braces.

The set of all processes with alphabet A is denoted by $\mathcal{T}.A$, and $(\mathcal{T}.A, \subseteq)$ is a complete lattice ([Bir]) with the least element $\mathbf{stop}.A$ and the greatest element $\mathbf{run}.A$ ([Kal]).

For process T and trace $t \in \mathbf{t}T$, the *successor set* of t in T contains the actions that may take place after t. The successor set of t in T is denoted by $suc.t.T$, and it is defined by

$$suc.t.T = \{a \mid a \in \mathbf{a}T \wedge ta \in \mathbf{t}T\}.$$

Property 1.1.4 ([Zwa]) shows how successor sets after projection can be calculated from the original successor sets.

Property 1.1.4

Let T be a process, $t \in \mathbf{t}T$, and $A \subseteq \mathbf{a}T$. Then

$$suc.(t\restriction A).(T\restriction A) = (\cup u : u \in \mathbf{t}T \wedge u\restriction A = t\restriction A : suc.u.T \cap A).$$

□

For process T and trace $t \in \mathbf{t}T$, process $after.t.T$ is defined as follows ([Zwa]):

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

A similar notion, applied to sets (e.g., [Rab]) and regular expressions ([Brz]), is well-known in automata theory, among others under the names derivative and quotient. The following notion is also similar to the notion of the state from automata theory (e.g., [Hop]).

The *states* of process T are the equivalence classes induced by *after*, i.e., corresponding to the equivalence relation $\overset{T}{\sim}$ defined for $s \in \mathbf{t}T$ and $t \in \mathbf{t}T$ by

$$s \overset{T}{\sim} t \Leftrightarrow after.s.T = after.t.T.$$

In the sequel we write $[t]_T$ to denote the equivalence class to which t belongs. We omit T in $[t]_T$ when it is obvious in a given context.

Processes that have a finite number of states are called *regular*.

Relation $\overset{T}{\sim}$ is an equivalence relation. It is right congruent with respect to concatenation, i.e.,

$$(\forall t, u, v : \{tv, uv\} \subseteq \mathbf{t}T \wedge t \overset{T}{\sim} u : tv \overset{T}{\sim} uv).$$

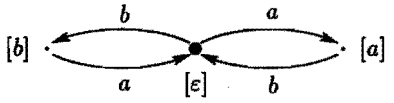
Hence, relation $\mathcal{R}$ on states can be defined by

$$[t]\mathcal{R}[u] \Leftrightarrow (\exists a : a \in \mathbf{a}T : [ta] = [u]).$$

This relation can be represented by a directed labelled graph, in the usual way ([Hop]). The directed labelled graph associated with T as described above is called *the state graph of* T. It is minimal and deterministic (each node has outgoing arcs with distinct labels and the number of nodes is equal to the number of states of the trace set). State graphs can be used to represent processes graphically.

Example 1.1.5

The state graph of process $\mathbf{sync}_{1,1}.a.b$ is



State $[\varepsilon]$ is called the initial state.

□

The state graph of $T\restriction A$ is obtained from the state graph of T by removing the labels that are not in A and by, subsequently, transforming the result into a minimal deterministic state graph. A well-known result from automata theory is that a finite nondeterministic state graph can be transformed into a finite minimal deterministic one. As a consequence, we have that regularity is closed under projection, which is expressed by Property 1.1.6.

Property 1.1.6

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

□

Processes can be composed by means of *weaving*. The *weave* of processes T and U, denoted by T **w** U, is defined by

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

The following property expresses that the set of processes is closed under weaving.

Property 1.1.7

Let T and U be processes. Then T **w** U is a process as well.

□

As the weave of two processes also is a process, it corresponds to a sequential mechanism.

The following property expresses that weaving is idempotent, commutative, and associative, and that **stop** is its unit ([Kal]).

Property 1.1.8

For processes T, U, and V

- $T \mathbf{w} T = T$,
- $T \mathbf{w} U = U \mathbf{w} T$,
- $(T \mathbf{w} U) \mathbf{w} V = T \mathbf{w} (U \mathbf{w} V)$,
- $T \mathbf{w} \mathbf{stop} = T$.

□

Property 1.1.9 states that weaving preserves regularity ([Kal]).

Property 1.1.9

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

□

Weaving can also be generalized to sets of processes. Let Y be a set of processes and let $A = (\cup T : T \in Y : \mathrm{a}T)$. The weave of the processes of Y is denoted by $\mathbf{W}.Y$ and is defined in the following way

$$\mathbf{W}.Y = \langle A, \{t \mid t \in A^* \wedge (\forall T : T \in Y : t \restriction \mathrm{a}T \in \mathrm{t}T)\}\rangle.$$

Note that $\mathbf{W}.\varnothing = \mathbf{stop}$, the unit element of weaving.
The next property shows that weaving is compositional.

Property 1.1.10

For sets of processes Y and Z

$$\mathbf{W}.(Y \cup Z) = \mathbf{W}.Y \mathbf{w} \mathbf{W}.Z.$$

□

We introduce the following convenient notation. Let Y be a set and let f be a function from Y to processes. The weave of processes $f.i$, for $i \in Y$, is denoted by

$$(\mathbf{W}i : i \in Y : f.i).$$

Using this notation we have $\mathbf{W}.Y = (\mathbf{W}T : T \in Y : T)$.
The following lemmata show the relationship between weaving and projection in three special cases ([Kal], [Zwa]).

Lemma 1.1.11

For processes T and U, and alphabet A

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

□

Lemma 1.1.12

For processes T and U, and alphabet A, such that $\mathbf{a}T \cap \mathbf{a}U \subseteq A$,

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

□

Lemma 1.1.13

For processes T and U, and alphabets A and B, such that $A \subseteq \mathbf{a}T$, $B \subseteq \mathbf{a}U$, and $\mathbf{a}T \cap \mathbf{a}U = A \cap B$,

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

□

Lemma 1.1.14 expresses that weaving and projection are monotonic with respect to process inclusion([Kal]).

Lemma 1.1.14

For processes T, U, V, and alphabet A

- $T \subseteq U \Rightarrow T \mathbf{w} V \subseteq U \mathbf{w} V$,
- $T \subseteq U \Rightarrow T\restriction A \subseteq U\restriction A$.

□

Lemma 1.1.15 shows that projection is monotonic with respect to set inclusion and with respect to trace prefixing ([Kal]).

Lemma 1.1.15

For traces s and t, sets of traces X and Z, and alphabet A

- $s \leq t \Rightarrow s\restriction A \leq t\restriction A$,
- $X \subseteq Z \Rightarrow X\restriction A \subseteq Z\restriction A$.

□

The following property shows that successor sets of the weave of two processes can be expressed in terms of successor sets of the processes ([Zwa]).

Property 1.1.16

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

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

□

Directly from the definition of projection the following property can be derived.

Property 1.1.17

For process T and alphabet A, such that $A \cap \mathbf{a}T = \varnothing$,

- $T\restriction \mathbf{a}T = T$,
- $T\restriction A = \mathbf{stop}$.

□

Properties 1.1.18 and 1.1.19 provide some more results ([Kal] and [Zwa]) concerning weaving and projection, which are used in this thesis.

Property 1.1.18

For processes T and U

$$T \mathbf{w} U\restriction \mathbf{a}T \subseteq T.$$

□

Property 1.1.19

Let f be a trace, or a set of traces, or a process. For alphabets A and B we have

$$(f\restriction A)\restriction B = (f\restriction B)\restriction A = f\restriction(A \cap B).$$

□

The following lemma is a generalization of the property $(T \mathbf{w} U)\restriction \mathbf{a}T \subseteq T$ ([Kal]).

Lemma 1.1.20

For set Y of processes and $T \in Y$

$$\mathbf{W}.Y\restriction \mathbf{a}T \subseteq T.$$

Proof

We derive

$$\begin{array}{ll} & \mathbf{W}.Y\restriction \mathbf{a}T \\ = & \quad \{\text{Property 1.1.10, } Y = \{T\} \cup Y\} \\ & (T \mathbf{w} \mathbf{W}.Y)\restriction \mathbf{a}T \\ = & \quad \{\mathbf{a}T \cap \mathbf{a}(\mathbf{W}.Y) \subseteq \mathbf{a}T, \text{ Lemma 1.1.12}\} \\ & T\restriction \mathbf{a}T \mathbf{w} \mathbf{W}.Y\restriction \mathbf{a}T \\ = & \quad \{\text{Property 1.1.18}\} \\ & T \mathbf{w} \mathbf{W}.Y\restriction \mathbf{a}T \\ \subseteq & \quad \{\text{Property 1.1.18}\} \\ & T. \end{array}$$

□

In general, for processes T and U, we have $(T \mathbf{w} U)\restriction \mathbf{a}T \subseteq T$ (see above). In a special case, when the alphabets of T and U are disjoint, the equality holds.

Lemma 1.1.21

For processes T and U, such that $\mathbf{a}T \cap \mathbf{a}U = \emptyset$,

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

Proof

We derive

$$
\begin{array}{ll}
& (T \mathbf{w} U)\restriction \mathbf{a}T \\
= & \quad \{\mathbf{a}T \cap \mathbf{a}U \subseteq \mathbf{a}T, \text{ Lemma } 1.1.12\} \\
& T\restriction \mathbf{a}T \mathbf{w} U\restriction \mathbf{a}T \\
= & \quad \{\mathbf{a}T \cap \mathbf{a}U = \emptyset, \text{ Property } 1.1.17\} \\
& T \mathbf{w} \mathbf{stop} \\
= & \quad \{\text{Property } 1.1.8\} \\
& T.
\end{array}
$$

□

Lemma 1.1.22 provides a simple way of checking whether

$$(\forall V : V \text{ is a process} : V \mathbf{w} T \subseteq V \mathbf{w} U)$$

holds for processes T and U.

Lemma 1.1.22

Let T and U be processes. Then

$$(\forall V : V \text{ is a process} : V \mathbf{w} T \subseteq V \mathbf{w} U) \Leftrightarrow T \subseteq U.$$

Proof

The implication from the right to the left holds on account of the monotonicity of weaving with respect to process inclusion (Lemma 1.1.14). For the implication the other way around we derive

$$
\begin{array}{ll}
& (\forall V : V \text{ is a process: } V \mathbf{w} T \subseteq V \mathbf{w} U) \\
\Rightarrow & \quad \{\text{predicate calculus}\} \\
& T \mathbf{w} T \subseteq T \mathbf{w} U \wedge U \mathbf{w} T \subseteq U \mathbf{w} U \\
\Rightarrow & \quad \{\text{Property } 1.1.8\} \\
& T \subseteq T \mathbf{w} U \wedge T \mathbf{w} U \subseteq U \\
\Rightarrow & \quad \{\subseteq \text{ is transitive}\} \\
& T \subseteq U.
\end{array}
$$

□

Lemmata 1.1.24 and 1.1.23, and Corollary 1.1.25 ([Kal]) facilitate calculations with **sync**'s and **buf**'s.

Lemma 1.1.23

Let k, l, m, and n be natural numbers, such that $k + l \geq 1$ and $m + n \geq 1$. For non-empty alphabets A, B, C, and D, such that $A \cap B = \emptyset$, $C \cap D = \emptyset$, $A \cap C = \emptyset$, $B \cap D = \emptyset$, $A \cap D \neq \emptyset$, and $B \cap C \neq \emptyset$,

$$(\mathbf{sync}_{k,l}.A.B \mathbf{\ w\ sync}_{m,n}.C.D)\restriction((A \cup C) \div (B \cup D))$$
$$=$$
$$\mathbf{sync}_{k+m,l+n}.((A \cup C) \setminus (B \cup D)).((B \cup D) \setminus (A \cup C)).$$

□

Lemma 1.1.24

Let k, l, m, and n be natural numbers, such that $k + l \geq 1$ and $m + n \geq 1$. For non-empty alphabets A, B, and C, such that $A \cap B = \emptyset$ and $B \cap C = \emptyset$,

$$(\mathbf{sync}_{k,l}.A.B \mathbf{\ w\ sync}_{m,n}.B.C)\restriction(A \div C) = \mathbf{sync}_{k+m,l+n}.(A \setminus C).(C \setminus A).$$

□

Corollary 1.1.25 is a special case of Lemma 1.1.24.

Corollary 1.1.25

Let k and l be natural numbers, such that $k + l \geq 1$. For non-empty alphabets A, B, and C, such that $A \cap B = \emptyset$, $B \cap C = \emptyset$, and $A \cap C = \emptyset$,

$$(\mathbf{buf}_k.A.B \mathbf{\ w\ buf}_l.B.C)\restriction(A \cup C) = \mathbf{buf}_{k+l}.A.C.$$

□

Regular processes can be described by a generalized form of regular expressions called *commands*. Every command $\mathcal{C}$ defines a trace structure $\mathbf{ts}\mathcal{C}$. Below follows an inductive definition of commands and their associated trace structures ([Kal], [Zwa]). In the sequel the following abbreviations are used: $\mathbf{a}\mathcal{C}$ for $\mathbf{a}(\mathbf{ts}\mathcal{C})$ and $\mathbf{t}\mathcal{C}$ for $\mathbf{t}(\mathbf{ts}\mathcal{C})$.

- ϵ is a command and $\mathbf{ts}\epsilon = \mathbf{stop}$,
- a is a command and $\mathbf{ts}a = \langle\{a\}, \{a\}\rangle$, for $a \in \Omega$,
- if $\mathcal{C}$ is a command then so is $\mathcal{C}^*$ and $\mathbf{ts}\mathcal{C}^* = \langle \mathbf{a}\mathcal{C}, (\mathbf{t}\mathcal{C})^* \rangle$,

- if C_0 and C_1 are commands then so is $C_0 \mid C_1$ and
  $\mathbf{ts}(C_0 \mid C_1) = \langle \mathbf{a}C_0 \cup \mathbf{a}C_1, \mathbf{t}C_0 \cup \mathbf{t}C_1 \rangle$,
- if C_0 and C_1 are commands then so is $C_0 ; C_1$ and
  $\mathbf{ts}(C_0 ; C_1) = \langle \mathbf{a}C_0 \cup \mathbf{a}C_1, \{tu \mid t \in \mathbf{t}C_0 \wedge u \in \mathbf{t}C_1\} \rangle$,
- if C_0 and C_1 are commands such that $\mathbf{ts}C_0 \mathbf{\ w\ } \mathbf{ts}C_1 \neq \mathbf{stop}.(\mathbf{a}C_0 \cup \mathbf{a}C_1)$
  then so is C_0 , C_1 and $\mathbf{ts}(C_0 , C_1) = \mathbf{ts}C_0 \mathbf{\ w\ } \mathbf{ts}C_1$,
- if C is a command then so is C^0 and $\mathbf{ts}C^0 = \mathbf{stop}.\mathbf{a}C$.

The highest priority belongs to the star operator ($*$), followed, in decreasing order, by: the zero (0), the comma (,), the semicolon (;) and the bar ($|$). Commands are defined to be equivalent if and only if their trace structures are equal.

Because trace structures of commands are non-empty, their prefix closures are processes. Hence, $\mathbf{pr}C$ defined by

$$\mathbf{pr}C = \mathit{pref}.\mathbf{ts}C$$

is a process.

For finite concatenations of the same command C an abbreviation is introduced. Formally, for command C and natural number $n \geq 0$

$$C^{n+1} = C ; C^n.$$

Example 1.1.26 illustrates the use of commands.

Example 1.1.26

For symbols a and b

$$\begin{aligned} \mathbf{sync}_{1,1}.a.b &= \mathbf{pr}((a , b)^*), \\ \mathbf{buf}_2.a.b &= \mathbf{pr}(a ; (a, b)^*), \\ \mathbf{stop}.a &= \mathbf{pr}(a^0). \end{aligned}$$

□

1.2 Systems

A process describes the behaviour of a sequential mechanism. In the weave of processes, which again is a process, parallelism is not represented because information about the individual mechanisms is no longer available. In contrast to this, a *system* does retain

information about the participating components. In Section A.3 of the Appendix, the difference between both approaches is explained in more detail.

A system ([Klo]) is a mathematical abstraction of co-operating mechanisms that are represented by processes in terms of trace theory. Formally, a system is a pair $\langle E, Y \rangle$, where Y is a set of processes and E is a subset of $\mathbf{a}(\mathbf{W}.Y)$. Process $\mathbf{W}.Y$ describes the entire sequentialized behaviour of the co-operating mechanisms, set E represents the set of external actions (only these may be used by the environment for interaction with the composite). The internal actions are those of $\mathbf{a}(\mathbf{W}.Y) \setminus E$.

For system S, we denote its external alphabet by $\mathbf{e}S$, and its set of processes by $\mathbf{p}S$. Process $\mathbf{W}.\mathbf{p}S \restriction \mathbf{e}S$ is denoted by $\mathbf{pr}S$. Notice that $\mathbf{a}(\mathbf{pr}S) = \mathbf{e}S$. For the sake of brevity, we abbreviate $\mathbf{a}(\mathbf{W}.\mathbf{p}S)$ to $\mathbf{a}S$, and $\mathbf{t}(\mathbf{W}.\mathbf{p}S)$ to $\mathbf{t}S$.

In the sequel, we restrict ourselves to systems S that satisfy the restriction that every action appears only in a finite number of processes, i.e.,

$$(\forall a : a \in \mathbf{a}S : \{T \mid T \in \mathbf{p}S \wedge a \in \mathbf{a}T\} \text{ is finite}).$$

This restriction flows from the demand that only finitely many mechanisms of a composite participate in a communication action.

In [Zwa], composition of systems is defined only for systems with common symbols belonging to the intersection of their external alphabets. That is, systems S_0 and S_1 may be composed only if $\mathbf{a}S_0 \cap \mathbf{a}S_1 = \mathbf{e}S_0 \cap \mathbf{e}S_1$. We would like to treat parallel composition of systems in a slightly different way. We prefer an operator realizing parallel composition that completely abstracts from internal symbols. The idea behind this is a very operational one. When connecting two components with each other by means of the accessible (external) connections in compliance with instructions received, one should not be concerned with the names of the connections inside the components. Systems that are identical except for the names of internal actions can therefore be treated as equivalent ones. This concept of equivalence is formalized below. For this purpose, we introduce some preliminary notions.

A bijection ρ from A onto B, $A \subseteq \Omega$ and $B \subseteq \Omega$, is called a *renaming* of A to B. Renaming is extended to traces, processes, and systems in the following way.

For trace t, $t \in A^*$, the renaming of t, $\rho.t$, is defined by

$$\begin{aligned} \rho.\varepsilon &= \varepsilon, \\ \rho.(ta) &= (\rho.t)(\rho.a). \end{aligned}$$

For process T, $\mathbf{a}T \subseteq A$, the renaming of T, $\rho.T$, is defined by

$$\rho.T = \langle \rho.\mathbf{a}T, \{\rho.t \mid t \in \mathbf{t}T\} \rangle.$$

For system S, $\mathbf{a}S \subseteq A$, the renaming of S, $\rho.S$, is defined by

$$\rho.S = \langle \rho.\mathbf{e}S, \{\rho.T \mid T \in \mathbf{p}S\}\rangle.$$

Systems S_0 and S_1 are defined to be equivalent if there exists a renaming ρ of $\mathbf{a}S_0$ to $\mathbf{a}S_1$ such that

$$(\forall a : a \in \mathbf{e}S_0 : \rho.a = a) \wedge S_1 = \rho.S_0.$$

The fact that S_0 is equivalent to S_1 in the above sense is denoted by $S_0 \approx S_1$. It can be proved that $\approx$ is an equivalence relation. By $[S]$, where S is a system, we denote the equivalence class to which S belongs.

Systems belonging to the same equivalence class have equal external processes. That is, for every $S' \in [S]$, $\mathbf{pr}S' = \mathbf{pr}S$.

Parallel composition operator, $\|$, is defined for the equivalence classes of systems under relation $\approx$.

Definition 1.2.1

The composition of systems S_0 and S_1 is denoted by $[S_0] \parallel [S_1]$, and it is defined by

$$[S_0] \parallel [S_1] = [\langle \mathbf{e}S_0' \cup \mathbf{e}S_1', \mathbf{p}S_0' \cup \mathbf{p}S_1'\rangle],$$

where S_0' and S_1' are systems such that $S_0' \approx S_0$, $S_1' \approx S_1$, and $\mathbf{a}S_0' \cap \mathbf{a}S_1' = \mathbf{e}S_0' \cap \mathbf{e}S_1'$.

□

For any S_0 and S_1 it is always possible to find suitable S_0' and S_1' provided the universe of symbols, Ω, is large enough to allow desirable renaming (such that the intersection of $\mathbf{a}S_0'$ and $\mathbf{a}S_1'$ equals the intersection of $\mathbf{e}S_0'$ and $\mathbf{e}S_1'$).

The above definition is a proper one, because for every representative S_0' of $[S_0]$ and S_1' of $[S_1]$, such that $\mathbf{a}S_0' \cap \mathbf{a}S_1' = \mathbf{e}S_0' \cap \mathbf{e}S_1'$, $\langle \mathbf{e}S_0' \cup \mathbf{e}S_1', \mathbf{p}S_0' \cup \mathbf{p}S_1'\rangle$ belongs to the same equivalence class.

The following property expresses that our parallel composition operator is commutative and associative, and that its unit is $[\langle \emptyset, \emptyset\rangle]$.

Property 1.2.2

For systems R, S, and T

- $[R] \parallel [S] = [S] \parallel [R]$,

- $([R] \parallel [S]) \parallel [T] = [R] \parallel ([S] \parallel [T])$,
- $[S] \parallel [\langle \emptyset, \emptyset \rangle] = [S]$.

□

Note that the parallel composition operator is not idempotent, that is $[S] \parallel [S] \neq [S]$. Consider, for instance, $S = \langle \emptyset, \{\mathbf{pr}x\} \rangle$. Then $[S] \parallel [S] = [\langle \emptyset, \{\mathbf{pr}(x, x')\} \rangle]$ and this is different from $[S]$.

If Y is a bag of systems, $(\parallel S : S \in Y : [S])$ denotes the parallel composition of systems of Y. For parallel composition of parametrized systems we introduce a convenient notation. For instance, let n be a natural number ($n \geq 0$). Parallel composition of systems S_i, for $0 \leq i < n$, is denoted by $(\parallel i : 0 \leq i < n : [S_i])$. Also, for set A we denote parallel composition of systems $S.a$, for $a \in A$, by $(\parallel a : a \in A : [S.a])$.

Since $\mathbf{pr}S' = \mathbf{pr}S$, for systems S and S' such that $S \approx S'$, the following definition is sound.

Definition 1.2.3

For system S

$$\mathbf{pr}[S] = \mathbf{pr}S.$$

□

An important result associated with $\parallel$ is the following lemma.

Lemma 1.2.4

For any two systems S_0 and S_1

$$\mathbf{pr}([S_0] \parallel [S_1]) = \mathbf{pr}S_0 \ \mathbf{w} \ \mathbf{pr}S_1.$$

Proof

Let $S_0' \approx S_0$, $S_1' \approx S_1$ and $\mathbf{a}S_0' \cap \mathbf{a}S_1' = \mathbf{e}S_0' \cap \mathbf{e}S_1'$. Then

$$[S_0] \parallel [S_1] = [\langle \mathbf{e}S_0' \cup \mathbf{e}S_1', \mathbf{p}S_0' \cup \mathbf{p}S_1' \rangle].$$

We derive

$$
\begin{aligned}
& \mathbf{pr}([S_0] \parallel [S_1]) \\
= & \quad \{ \text{Definition 1.2.1} \} \\
& \mathbf{pr}[(\mathbf{e}S_0' \cup \mathbf{e}S_1', \mathbf{p}S_0' \cup \mathbf{p}S_1')] \\
= & \quad \{ \text{Definition 1.2.3} \} \\
& \mathbf{pr}\langle \mathbf{e}S_0' \cup \mathbf{e}S_1', \mathbf{p}S_0' \cup \mathbf{p}S_1' \rangle \\
= & \quad \{ \mathbf{pr}S = \mathbf{W}.\mathbf{p}S \restriction \mathbf{e}S \} \\
& \mathbf{W}.(\mathbf{p}S_0' \cup \mathbf{p}S_1') \restriction (\mathbf{e}S_0' \cup \mathbf{e}S_1') \\
= & \quad \{ \text{Property 1.1.10} \} \\
& (\mathbf{W}.\mathbf{p}S_0' \mathbf{\ w\ } \mathbf{W}.\mathbf{p}S_1') \restriction (\mathbf{e}S_0' \cup \mathbf{e}S_1') \\
= & \quad \{ \mathbf{e}S_0' \subseteq \mathbf{a}S_0' \wedge \mathbf{e}S_1' \subseteq \mathbf{a}S_1' \wedge \mathbf{a}S_0' \cap \mathbf{a}S_1' = \mathbf{e}S_0' \cap \mathbf{e}S_1', \text{ Lemma 1.1.13} \} \\
& \mathbf{W}.\mathbf{p}S_0' \restriction \mathbf{e}S_0' \mathbf{\ w\ } \mathbf{W}.\mathbf{p}S_1' \restriction \mathbf{e}S_1' \\
= & \quad \{ \mathbf{pr}S = \mathbf{W}.\mathbf{p}S \restriction \mathbf{e}S \} \\
& \mathbf{pr}S_0' \mathbf{\ w\ } \mathbf{pr}S_1' \\
= & \quad \{ S_0' \approx S_0 \wedge S_1' \approx S_1, \text{ and } S' \approx S \Rightarrow \mathbf{pr}S' = \mathbf{pr}S \} \\
& \mathbf{pr}S_0 \mathbf{\ w\ } \mathbf{pr}S_1
\end{aligned}
$$

□

In the sequel, if it does not cause confusion, we write S, meaning $[S]$, in the context of parallel composition.

It is possible to restrict external communications of a given system by means of the projection on an alphabet. The system obtained by projection of system S on alphabet A, denoted by $S \restriction A$, is defined by

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

The following property ([Zwa]) shows the relationship between $\mathbf{pr}(S \restriction A)$ and $\mathbf{pr}S \restriction A$.

Property 1.2.5

For system S and alphabet A

$$\mathbf{pr}(S \restriction A) = \mathbf{pr}S \restriction A.$$

□

Property 1.2.6 ([Zwa]) is related to parallel composition and projection, and it is used in Chapter 4.

Property 1.2.6

For systems S and T, and alphabet A, such that $\mathbf{e}S \cap \mathbf{e}T \subseteq A$,

$$(S \parallel T)\restriction A = S\restriction A \parallel T\restriction A.$$

□

Composition of systems and projection may be used to describe hierarchical designs. For instance, $(R \parallel S)\restriction(\mathbf{e}R \div \mathbf{e}S)$ represents a mechanism built from two components specified by systems R and S, possible interactions with the environment may take place via actions from $\mathbf{e}R \div \mathbf{e}S$. (Common external symbols — $\mathbf{e}R \cap \mathbf{e}S$ — represent internal connections that are hidden from their environment.)

1.3 Properties of processes and systems

Two properties of processes that are important for this thesis concern the behaviour of a process with respect to the actions from a subset of its alphabet. The first one deals with the situation in which action a of A can be disabled by the process performing some action of its alphabet that is not in A, whereas the projection on A suggests that a is a possible continuation. The second one deals with the situation in which the process can engage in an infinite sequence of actions from outside of A.
These properties are defined for process T and $A \subseteq \mathbf{a}T$ by ([Kal], [Zwa])

$$\mathit{disabling}.A.T \;\Leftrightarrow\; (\exists t : t \in \mathbf{t}T : \mathit{after}.t.T\restriction A \neq \mathit{after}.(t\restriction A).(T\restriction A))$$

and

$$\mathit{divergent}.A.T \;\Leftrightarrow\; (\exists t : t \in \mathbf{t}T : (\forall n : n \geq 0 : (\exists u : u\restriction A = \varepsilon \wedge tu \in \mathbf{t}T : \ell.u > n))).$$

Example 1.3.1 illustrates both notions.

Example 1.3.1

Consider process T defined by

$$T = \mathbf{pr}(x\,;\, a \mid y\,;\, b).$$

Then $\mathit{disabling}.\{a,b\}.T$ holds because

$$\mathit{after}.y.T\restriction\{a,b\} = \langle\{a,b\},\{\varepsilon,b\}\rangle,$$

$$after.(y\restriction\{a,b\}).(T\restriction\{a,b\}) = \langle\{a,b\},\{\epsilon,a,b\}\rangle$$

and hence,

$$after.y.T\restriction\{a,b\} \neq after.(y\restriction\{a,b\}).(T\restriction\{a,b\}).$$

From $T\restriction\{a,b\}$ it can be concluded that initially both a and b are possible. However, from the analysis of T follows that if the process is in state $[y]$, a cannot occur because it is disabled. Hence, a is not necessarily possible initially.

Now consider process T' defined by

$$T' = \mathbf{pr}(x^*\,;\, a).$$

Then $divergent.\{a\}.T'$ holds because

$$(\forall n : n \geq 0 : x^n \in \mathbf{t}T').$$

From $T'\restriction a$ can be concluded that a may occur in the initial state. However, since there is no upper bound for the number of times x can be performed, it is not guaranteed that a will ever occur.

□

Both notions are used to express the possibility of lack of progress (e.g., [Kal] and [Hoa]). The second notion, however, unjustly classifies some processes as having the possibility of lacking progress. As an example consider process T defined by

$$T = \langle\{a,y\} \cup \{x_i \mid i \geq 0\}, \{t \mid (\exists i : i \geq 0 : t \leq x_i y^i a)\}\rangle.$$

According to the given definition, process T is divergent with respect to $\{a\}$, that is, $divergent.\{a\}.T$ holds. From $divergent.\{a\}.T$ one can conclude that T may engage in an infinite sequence of y actions. In this case a will not happen. On the other hand, every execution (maximal trace) of T ends in a, i.e., a is guaranteed to happen.

An even more convincing case is process $T = \mathbf{pr}(a\,;\, x^*)$. We have from the definition that T is divergent with respect to $\{a\}$. But after a has happened, no external activity is expected. Thus T cannot exhibit lack of external progress.

Both examples above suggest that the notion of divergence as defined in [Kal] (and in [Hoa]) is not quite suitable to characterize the behaviour of a process with respect to a subset of its actions, where progress is concerned.

The notion of divergence is also known under the names livelock ([Kal], the term livelock is due to [Ash]) and infinite chatter ([Mil]). It is used in the context of Hoare's processes as well.

The notion of disabling appears under different names too, for instance as internal nondeterminism ([Mil]) or as refusing ([Hoa]).

If $\neg$*disabling*.$A.T$ and $\neg$*divergent*.$A.T$ hold for process T and $A \subseteq \mathbf{a}T$, we say that $T\restriction A$ adequately describes the behaviour of T after projection on alphabet A. In other words: internal actions have no influence on the external behaviour of the mechanism specified by process T, with respect to possible interactions with its environment.

The notions of disabling and divergence are defined for systems as well ([Zwa]).

Definition 1.3.2

System S is called *disabling*, if *disabling*.$\mathbf{e}S.(\mathbf{W}.\mathbf{p}S)$ holds.

□

Definition 1.3.3

System S is called *divergent*, if *divergent*.$\mathbf{e}S.(\mathbf{W}.\mathbf{p}S)$ holds.

□

Because systems have more structure than processes do, the above definition of divergence qualifies more systems as being divergent than one would expect when taking this structure into account. This can easily be seen from the following example.

Example 1.3.4

Processes T and U are denoted by

$$T = \mathbf{pr}(a^*)$$

and

$$U = \mathbf{pr}(b^*).$$

Then, according to the definition, *divergent*.$\{a\}.(T \mathbf{w} U)$ holds. But the mechanisms represented by T and U are independent of each other (true concurrency is assumed) and so are a and b. Thus there is, in fact, no reason why a would not be executed while U can communicate an unbounded number of b's.

□

In this thesis we develop a more suitable characterization of the behaviour of systems with regard to divergence (the possibility of lack of external progress). This characterization makes use of parallelism in systems, if present.

Finally, we discuss a special class of processes introduced in [Zwa] which are called *conservative* processes. A few results concerning conservative processes are used in Chapters 2, 3, and 4.

Definition 1.3.5

Process T is called *conservative* if

$$(\forall t, a, b : \{ta, tb\} \subseteq \mathrm{t}T \wedge a \neq b : \{tab, tba\} \subseteq \mathrm{t}T \wedge [tab] = [tba]).$$

□

The states of a conservative process depend only on the number of occurrences of events (the name *conservative* is due to this property). Furthermore, such a process is non-disabling with respect to every subset of its alphabet.

Consider $T = \mathbf{pr}((a\,,\, b;\, c)^*)$ as an example. The states of T are

$$\begin{aligned}
[\varepsilon] &= \{t \mid t \in \mathrm{t}T \wedge \ell.(t\restriction a) = \ell.(t\restriction b) = \ell.(t\restriction c)\} \\
[a] &= \{t \mid t \in \mathrm{t}T \wedge \ell.(t\restriction a) - 1 = \ell.(t\restriction b) = \ell.(t\restriction c)\} \\
[b] &= \{t \mid t \in \mathrm{t}T \wedge \ell.(t\restriction a) = \ell.(t\restriction b) - 1 = \ell.(t\restriction c)\} \\
[ab] &= \{t \mid t \in \mathrm{t}T \wedge \ell.(t\restriction a) = \ell.(t\restriction b) = \ell.(t\restriction c) + 1\}.
\end{aligned}$$

It is easily shown that $\neg \mathit{disabling}.A.T$ holds for any $A \subseteq \mathrm{a}T$.

An important observation is that any process with at most one symbol in every successor set is conservative.

In the sequel, we refer to the following properties ([Zwa]).

Property 1.3.6

For conservative process T

- $(\forall s, t : \{s, t\} \subseteq \mathrm{t}T \wedge \mathit{suc}.s.T = \emptyset \wedge \mathit{suc}.t.T = \emptyset : \flat.s = \flat.t)$,
- $(\forall s, t : \{s, t\} \subseteq \mathrm{t}T \wedge \flat.s = \flat.t : [s] = [t])$.

□

Property 1.3.7

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

□

Property 1.3.8

Let A be an alphabet and T a conservative process. Then $T\restriction A$ also is a conservative process.

□

Property 1.3.9 provides an alternative characterization of conservative processes.

Property 1.3.9

Let T be a process. Then

$$T \text{ is conservative } \Leftrightarrow (\forall s,t : \{s,t\} \subseteq \mathbf{t}T : s(t \setminus \natural.s) \in \mathbf{t}T).$$

□

1.4 Failures model of systems

This section gives a short presentation of the failures model of trace theory systems; it is essentially based on [Hoa]. In the failures model, parallelism is explained in terms of interleaving. That is, a system in which two actions may occur concurrently is equivalent to a system in which these actions are interleaved.

The failures model provides a more detailed description of systems behaviour than the one introduced in the previous section, which is based on the predicates *disabling* and *divergent*. The model described in this section is not compositional.

First a few preliminary notions are introduced.

Definition 1.4.1

For system S, $a \in \mathbf{e}S$, and $u \in \mathbf{t}S$

$$dis.u.a.S \Leftrightarrow (\forall v : v\restriction \mathbf{e}S = \varepsilon \wedge uv \in \mathbf{t}S : uva \notin \mathbf{t}S).$$

□

If $dis.u.a.S$ holds, we say that a is disabled after u.

Definition 1.4.2

For system S, $a \in \mathbf{e}S$, and $u \in \mathbf{t}S$

$$div.u.a.S \;\Leftrightarrow\; (\forall n : 0 \leq n : (\exists v : v \restriction \mathbf{e}S = \varepsilon \wedge uva \in \mathbf{t}S : \ell.v > n)).$$

□

If $div.u.a.S$ holds, we say that after u system S is divergent with respect to a.

Definition 1.4.3

For system S, alphabet $R \subseteq \mathbf{e}S$, and $t \in \mathbf{t}(\mathbf{pr}S)$

$$ref.t.R.S \;\Leftrightarrow\; (\exists u : u \in \mathbf{t}S \wedge u \restriction \mathbf{e}S = t : (\forall a : a \in R : dis.u.a.S \vee div.u.a.S)).$$

□

If $ref.t.R.S$ holds, R is called a refusal set after t.

Definition 1.4.4

For system S

$$\mathbf{f}S = \{(t, R) \mid t \in \mathbf{t}(\mathbf{pr}S) \wedge R \subseteq \mathbf{e}S \wedge ref.t.R.S\}.$$

□

Elements of $\mathbf{f}S$ are called failures ([Hoa]). In the sequel, for the sake of brevity, only failures with maximal refusal sets are explicitly given in abstract models of concrete systems. This is possible on account of the following property.

Property 1.4.5

For system S

$$(t, R) \in \mathbf{f}S \wedge R' \subseteq R \;\Rightarrow\; (t, R') \in \mathbf{f}S.$$

□

Observe also that

$$\mathbf{t}(\mathbf{pr}S) = \{t \mid (t, \varnothing) \in \mathbf{f}S\}.$$

In [Hoa], a relation on processes is defined that expresses whether a process is a correct implementation of a specification (that is, some other process). We define a corresponding relation for systems.

Definition 1.4.6

Let S_0 and S_1 be systems. We say that S_0 implements S_1 if

$$\mathbf{e}S_0 = \mathbf{e}S_1 \wedge \mathbf{f}S_0 \subseteq \mathbf{f}S_1.$$

□

The following example illustrates the above notions.

Example 1.4.7

Let

$$S_0 = \langle \{a, b\}, \{\mathbf{pr}(x\,;\, a \mid y\,;\, z^*\,;\, a), \mathbf{pr}(y\,;\, b), \mathbf{pr}(a \mid b)\}\rangle$$

and

$$S_1 = \langle \{a, b\}, \{\mathbf{pr}(x'\,;\, a \mid y'\,;\, b)\}\rangle.$$

For S_0 we have $div.y.a.S_0$ and $div.y.b.S_0$, and hence,

$$(\varepsilon, \{a, b\}) \in \mathbf{f}S_0.$$

For S_1 we have $dis.y'.a.S_1$ and $dis.x'.b.S_1$, and hence,

$$(\varepsilon, \{a\}) \in \mathbf{f}S_1 \wedge (\varepsilon, \{b\}) \in \mathbf{f}S_1.$$

However, $(\varepsilon, \{a, b\}) \notin \mathbf{f}S_1$, because $\neg ref.\varepsilon.\{a, b\}.S_1$. The sets of failures with maximal refusal sets for both systems are then

$$\mathbf{f}S_0 = \{(\varepsilon, \{a, b\}), (a, \{a, b\}), (b, \{a, b\})\}$$

and

$$\mathbf{f}S_1 = \{(\varepsilon, \{a\}), (\varepsilon, \{b\}), (a, \{a, b\}), (b, \{a, b\})\}.$$

This yields that S_0 does not implement S_1 and S_1 does implement S_0. Intuitively, we would say that S_0 is equivalent to S_1 (thus also implements S_1), because $y\,;\, z^*$ cannot prevent the execution of b.

□

2 Operational model of systems

A process as defined in the trace theory of [Rem], [Sne], and [Kal] is a model describing possible behaviours of a sequential mechanism. As opposed to this, a system ([Klo], [Zwa]) is a model describing several mechanisms co-operating with each other in a non-sequential (parallel) fashion.

Operational models of both processes and systems can be defined by sets of sequences of global states and action occurrences of the objects in question, assuming that every action occurrence changes the global state in which it is executed. If mutually independent actions are possible in a state, they are usually interleaved. Such kinds of operational models are called interleaving models. These models explain parallelism in terms of nondeterminism. In yet another approach to operational models, mutually independent actions are treated as being partially ordered instead of totally ordered by interleaving (e.g., [Rel]). This idea, which originated in Petri Net theory ([Pet]), also gave rise to a variety of non-operational partial-order models (e.g., [Mal], [Pra], and [Win]). In partial-order models nondeterministic but sequential systems can be distinguished from concurrent ones.

In this chapter, we give a non-interleaving operational model of systems as defined in trace theory. The model is inspired by partial-order models, in particular by the one presented in [Rel]. It formalises our view of how systems actually work. The behaviour of a system in our model is represented by partially ordered subsets of local states (of processes that are the components of the system) and action occurrences. In this chapter, we provide a characterization of the external behaviour of systems based on this model.

In Section 2.1, a formal definition of the operational model of systems is given and it is illustrated by several examples. The operational model associates a labelled transition system with every trace theory system. The notion of computation is introduced to formally define the entire behaviour of a system (more precisely, the behaviour of the mechanisms described by the system). The internal activity is still explicit. In Section 2.2, the notion of acceptances is introduced to characterize the external behaviour of systems, thus abstracting from the internal activity. By means of a few examples acceptances are related to failures defined in Section 1.4. In Section 2.3, a more direct characterization of acceptances is presented. This characterization is given not in terms the operational model but in terms of the system itself, and therefore it

is simpler. Section 2.4 presents relations on systems based on the operational model. These relations formalize a verification method for systems that deals with safety and liveness properties. In Section 2.5, some properties of the relations introduced in the previous section are proved. Specifically, parallel composition and projection are monotonic with respect to the implementation relation (**sat**), and the equivalence relation induced by **sat** is a congruence with respect to parallel composition and to projection.

In the Appendix, some additional results associated with the operational model are described. They include some further properties of computations, the relationship between the transition systems of Section 2.1 and nets, the comparison of the model defined in this chapter with an interleaving model, and an example considering three candidate implementations of a buffer.

2.1 Operational model

The purpose of this section is to define a non-interleaving operational model of systems — a description of the behaviour of a system that maintains full information about concurrency. This description is then used to formally justify statements concerning guaranteed behaviour (progress) of a system, like "after trace t action a will certainly happen". More precisely, the operational model we aim at should allow us to conclude that for system $S = \langle\{a\},\{T,U\}\rangle$, where $T = \mathbf{pr}(b^*; e)$ and $U = \mathbf{pr}((c \mid e); a)$, the occurrence of a is guaranteed. This example is discussed in more detail at the end of this section (see Example 2.1.17).

Our model is inspired by [Rel]. The basic idea underlying this model is that of local states of constituent parts of the system and independence of actions. This is also the main distinction between our model and the labelled transition systems of [De0]. When a number of processes co-operate, the execution of an action does not change the state of a process that does not participate in it.

In order to give the definition of the operational model of a system, we first introduce some preliminary notions. In the sequel, S is always a system characterized by the requirement that every action of its alphabet appears only in a finite number of processes that constitute S (as defined in Section 1.2). Newly introduced notions are illustrated by referring to processes $\mathcal{T}$ and $\mathcal{U}$ defined by

$$\mathcal{T} = \mathbf{pr}(a\,,\, c; d),$$

$$\mathcal{U} = \mathbf{pr}((b \mid d); c),$$

and to system $\mathcal{S}$ defined by

$$\mathcal{S} = \langle\{a,b,c,d\},\{\mathcal{T},\mathcal{U}\}\rangle.$$

Note that the process and system names in this running example are *script letters* $\mathcal{T}$, $\mathcal{U}$, and $\mathcal{S}$, respectively.

The operational model of a system is defined as a transition system. In this thesis, by a transition system we mean a triple of which the first element defines the set of local (process) states, the second the set of action occurrences, and the third a transition relation between the local states and action occurrences (this is a slightly modified combination of the definitions from [Hen] and [De0]). In the sequel, these three elements associated with a system are introduced in the given order. The main goal is a suitable definition of a transition relation that is able to express parallelism explicitly. That is, in such a way that the two systems from the Introduction can be distinguished with the help of this relation.

The following definitions are inspired by [Re1].

A *configuration* is a pair (t,T), for process T and trace t of $\mathrm{t}T$. Configuration (ε,T) is called an *initial configuration.*

Examples of configurations are

$$(\varepsilon,\mathcal{T}),(a,\mathcal{T}),(c,\mathcal{T}),(acd,\mathcal{T}),\text{ and }(\varepsilon,\mathcal{U}),(d,\mathcal{U}),(b,\mathcal{U}),(dc,\mathcal{U}).$$

Definition 2.1.1

The set of configurations of system S, $Conf.S$, is defined by

$$Conf.S = \{(u\restriction \mathrm{a}T,T) \mid T \in \mathrm{p}S \wedge u \in \mathrm{t}S\}.$$

The set of initial configurations of S, $Init.S$, is defined by

$$Init.S = \{(\varepsilon,T) \mid T \in \mathrm{p}S\}.$$

□

The set $Conf.S$ is the first element of the transition system associated with S.

For $\mathcal{S}$, our running example, we have

$$Conf.\mathcal{S} = \{(\varepsilon,\mathcal{T}),(a,\mathcal{T}),(c,\mathcal{T}),(ac,\mathcal{T}),(ca,\mathcal{T}),(\varepsilon,\mathcal{U}),(b,\mathcal{U}),(bc,\mathcal{U})\}$$

and $Init.\mathcal{S} = \{(\varepsilon,\mathcal{T}),(\varepsilon,\mathcal{U})\}$ (see page 40 for a graphical representation of the transition system belonging to $\mathcal{S}$).

Note that configuration $(acd,\mathcal{T})$ is not a member of $Conf.\mathcal{S}$.

From Lemma 1.1.20 we infer $\mathbf{W}.\mathrm{p}S\restriction \mathrm{a}T \subseteq T$, for system S and process T of $\mathrm{p}S$. Hence, we have the following property.

Property 2.1.2

Let S be a system. Then

$$Conf.S \subseteq \{(t,T) \mid T \in \mathbf{p}S \wedge t \in \mathbf{t}T\}.$$

□

Intuitively, configurations represent local states (states of processes). Suitable sets of configurations form *global states* (states of the system), also called *distributed states*, or simply states.

Definition 2.1.3

A distributed state of system S is a subset D of $Conf.S$ such that

$$(\exists u : u \in \mathbf{t}S : D = \{(u\lceil \mathbf{a}T, T) \mid T \in \mathbf{p}S\}).$$

The set of distributed states of S is denoted $\mathcal{D}.S$.

□

Examples of distributed states of system $\mathcal{S}$ are $\{(\epsilon,\mathcal{T}),(\epsilon,\mathcal{U})\}$, $\{(a,\mathcal{T}),(\epsilon,\mathcal{U})\}$, and $\{(ac,\mathcal{T}),(bc,\mathcal{U})\}$.
We say that state D is *associated with* trace u of $\mathbf{t}S$ if

$$D = \{(u\lceil \mathbf{a}T, T) \mid T \in \mathbf{p}S\}.$$

Such a trace is not necessarily unique: in case of system $\mathcal{S}$, state $\{(a,\mathcal{T}),(b,\mathcal{U})\}$ is associated with both ab and ba.
Property 2.1.4 follows immediately from Definitions 2.1.1 and 2.1.3.

Property 2.1.4

Let S be a system. Then

$$Conf.S = (\cup D : D \in \mathcal{D}.S : D).$$

□

We say that action a is *enabled* in state D of system S if

$$(\exists u : ua \in \mathbf{t}S : D = \{(u\lceil \mathbf{a}T, T) \mid T \in \mathbf{p}S\}).$$

For instance, a is enabled in state $\{(\varepsilon,T),(\varepsilon,U)\}$ of S and c is enabled in state $\{(a,T),(b,U)\}$ of S.

Due to our restriction on systems, only finitely many configurations can *participate* in a single action that is enabled in state D. If action a is enabled in state D, only configurations $(t,T) \in D$ such that $a \in \mathbf{a}T$ participate in it.

In state $\{(\varepsilon,T),(\varepsilon,U)\}$ of system S, only configuration (ε,T) participates in a. In state $\{(\varepsilon,T),(b,U)\}$, both configurations participate in c.

For convenience, sets of configurations that participate in action a are collected in $C.S.a$. Formally, $C.S.a$ is defined for every $a \in \mathbf{a}S$ by

$$C.S.a = \{C \mid (\exists u : ua \in \mathbf{t}S : C = \{(u\lceil \mathbf{a}T, T) \mid T \in \mathbf{p}S \wedge a \in \mathbf{a}T\})\}.$$

Every set C belonging to $C.S.a$ is a subset of a distributed state.

Note that $C.S.a \subseteq \mathcal{P}.(Conf.S)$.

With every element C of $C.S.a$ an *occurrence* of action a, in which the configurations of C participate, is associated. For $a \in \mathbf{a}S$ and $C \in C.S.a$, the occurrence of action a associated with C is denoted by (a,C).

For system S we have

$$C.S.a = \{\{(\varepsilon,T)\},\{(c,T)\}\},$$
$$C.S.b = \{\{(\varepsilon,U)\}\},$$
$$C.S.c = \{\{(\varepsilon,T),(b,U)\},\{(a,T),(b,U)\}\},$$
$$C.S.d = \varnothing.$$

Thus the action occurrences of S are

$$(a,\{(\varepsilon,T)\}),(a,\{(c,T)\}),(b,\{(\varepsilon,U)\}),(c,\{(\varepsilon,T),(b,U)\}),(c,\{(a,T),(b,U)\}).$$

Definition 2.1.5

The set of action occurrences of system S, $Act.S$, is defined by

$$Act.S = \{(a,C) \mid a \in \mathbf{a}S \wedge C \in C.S.a\}.$$

□

The set $Act.S$ is the second element of the transition system associated with S.

We prefer to label action occurrences with natural numbers, instead of with sets of configurations. This is possible when every $C.S.a$ is countable, for $a \in \mathbf{a}S$. System S satisfies this requirement if

$$(\forall T, a : T \in \mathbf{p}S \wedge a \in \mathbf{a}T : \{u \mid ua \in \mathbf{t}S\}\lceil \mathbf{a}T \text{ is countable}),$$

which is proved by the following lemma.

Lemma 2.1.6

Let S be a system and let $a \in \mathbf{a}S$. Then

$$(\forall T : T \in \mathbf{p}S \wedge a \in \mathbf{a}T : \{u \mid ua \in \mathbf{t}S\}\lceil \mathbf{a}T \text{ is countable})$$
$$\Rightarrow$$
$$\mathcal{C}.S.a \text{ is countable.}$$

Proof

For $a \in \mathbf{a}S$ we have

$$\begin{aligned}
& \mathcal{C}.S.a \\
= \quad & \{\text{definition of } \mathcal{C}.S.a\} \\
& \{C \mid (\exists u : ua \in \mathbf{t}S : C = \{(u\lceil \mathbf{a}T, T) \mid T \in \mathbf{p}S \wedge a \in \mathbf{a}T\})\} \\
= \quad & \{\text{calculus}\} \\
& \{C \mid |C| = (\#T : T \in \mathbf{p}S : a \in \mathbf{a}T) \\
& \quad \wedge (\exists u : ua \in \mathbf{t}S : (\forall T : T \in \mathbf{p}S \wedge a \in \mathbf{a}T : (u\lceil \mathbf{a}T, T) \in C))\} \\
\subseteq \quad & \{\text{predicate calculus}\} \\
& \{C \mid |C| = (\#T : T \in \mathbf{p}S : a \in \mathbf{a}T) \\
& \quad \wedge (\forall T : T \in \mathbf{p}S \wedge a \in \mathbf{a}T : (\exists u : ua \in \mathbf{t}S : (u\lceil \mathbf{a}T, T) \in C))\}
\end{aligned}$$

Furthermore,

$$\begin{aligned}
& |\{C \mid |C| = (\#T : T \in \mathbf{p}S : a \in \mathbf{a}T) \\
& \quad \wedge (\forall T : T \in \mathbf{p}S \wedge a \in \mathbf{a}T : (\exists u : ua \in \mathbf{t}S : (u\lceil \mathbf{a}T, T) \in C))\}| \\
= \quad & \{\text{definition of cartesian product}\} \\
& |(\times T : T \in \mathbf{p}S \wedge a \in \mathbf{a}T : \{(u\lceil \mathbf{a}T, T) \mid ua \in \mathbf{t}S\})|. \\
= \quad & \{\text{calculus}\} \\
& |(\times T : T \in \mathbf{p}S \wedge a \in \mathbf{a}T : \{u\lceil \mathbf{a}T \mid ua \in \mathbf{t}S\})|. \\
= \quad & \{\text{definition of } \lceil \text{ for sets of traces}\} \\
& |(\times T : T \in \mathbf{p}S \wedge a \in \mathbf{a}T : \{u \mid ua \in \mathbf{t}S\}\lceil \mathbf{a}T)|.
\end{aligned}$$

From set theory we know that the cartesian product of a finite number of countable sets is countable. In Section 1.2 we assumed that $\{T \mid T \in \mathbf{p}S \wedge a \in \mathbf{a}T\}$ is finite and above we assumed that $\{u \mid ua \in \mathbf{t}S\}\lceil \mathbf{a}T$ is countable. Hence,

$$(\times T : T \in \mathbf{p}S \wedge a \in \mathbf{a}T : \{u \mid ua \in \mathbf{t}S\}\lceil \mathbf{a}T)$$

is countable. On account of the above derivations we can conclude that $\mathcal{C}.S.a$ is countable as well.

□

If every $\mathcal{C}.S.a$ is countable, for $a \in \mathbf{a}S$, there exists a function f from $Act.S$ to the natural numbers such that f restricted to $\{a\} \times \mathcal{C}.S.a$ is an injective function. Then the occurrence of action a associated with C is denoted by $(a, f.a.C)$, instead of by (a, C). We call f an *enumeration.*

We then permit ourselves to write: $Act.S = \{(a, f.a.C) \mid a \in \mathbf{a}S \wedge C \in \mathcal{C}.S.a\}$.

A possible enumeration for $\mathcal{S}$ is

$$f.a.\{(\varepsilon, T)\} = f.b.\{(\varepsilon, \mathcal{U})\} = f.c.\{(\varepsilon, T), (b, \mathcal{U})\} = 0,$$
$$f.a.\{(c, T)\} = f.c.\{(a, T), (b, \mathcal{U})\} = 1.$$

Thus we may write: $Act.\mathcal{S} = \{(a, 0), (b, 0), (c, 0), (a, 1), (c, 1)\}$.

As we already mentioned, the main goal of this section is the definition of a suitable relation between configurations and action occurrences. A very important demand for this relation is preserving concurrency, if it is present in a system.

A transition relation between configurations and action occurrences of system S is defined, formalizing how processes that form the system work. There are two kinds of transitions: from configurations to action occurrences and the other way around. Configuration (t, T) is in the relation with action occurrence (a, C) if $a \in \mathbf{a}T$ and (t, T) participates in this action (in another words: if $(t, T) \in C$). Action occurrence (a, C) is in the relation only with those configurations (ta, T) that satisfy $(t, T) \in C$ (trace of configuration (t, T) is extended with a). This is formally expressed by the following definition.

Definition 2.1.7

For system S, relation $\rightarrow_S \subseteq (Conf.S \times Act.S) \cup (Act.S \times Conf.S)$ is defined by

$$\begin{aligned}
&(\forall a, C, t, T : (a, C) \in Act.S \wedge (t, T) \in Conf.S \\
&\qquad : (t, T) \rightarrow_S (a, C) \;\Leftrightarrow\; (t, T) \in C) \\
&\wedge (\forall a, C, t, T : (a, C) \in Act.S \wedge (t, T) \in Conf.S \\
&\qquad : (a, C) \rightarrow_S (t, T) \;\Leftrightarrow\; (\exists u : t = ua : (u, T) \in C)).
\end{aligned}$$

□

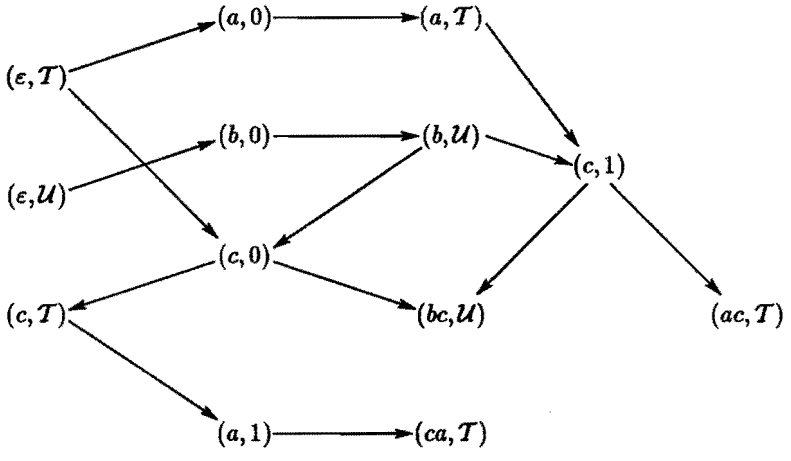
For $\mathcal{S}$ we have the following transition relation

$(\varepsilon,T) \to_S (a,0) \to_S (a,T) \to_S (c,1) \to_S (ac,T)$,
$(\varepsilon,U) \to_S (b,0) \to_S (b,U) \to_S (c,0) \to_S (bc,U)$,
$(\varepsilon,T) \to_S (c,0) \to_S (c,T) \to_S (a,1) \to_S (ca,T)$,
$(b,U) \to_S (c,1) \to_S (bc,U)$.

The transition system corresponding to system S describes operationally the behaviour of processes that constitute S. Therefore, we choose to define the operational model of system S, denoted by $\mathcal{O}[\![S]\!]$, as the triple $(Conf.S, Act.S, \to_S)$.

Note that $(Conf.S, Act.S, \to_S)$ is a bipartite graph.

The operational model of S is the triple $(Conf.S, Act.S, \to_S)$. It can also be presented in a graphical form, e.g.,



where arrows stand for $\to_S$ relation.

As another example consider the system with the empty set of processes, $\langle \emptyset, \emptyset \rangle$. The operational model of this system is then $(\emptyset, \emptyset, \emptyset)$.

In the sequel, we omit S in $\to_S$.

Note that in general $\to^*$ is not a partial order, as the following example shows.

Example 2.1.8

Consider processes $T = \mathbf{pr}(a, b)$, $U = \mathbf{pr}(b, c)$, and $V = \mathbf{pr}(c, a)$, and system S defined by

$S = \langle \{a, b, c\}, \{T, U, V\} \rangle$.

The set of configurations of S is

$$Conf.S = \{\ (\varepsilon,T),(\varepsilon,U),(\varepsilon,V),(a,T),(a,V),(b,T),(b,U),(c,U),(c,V),\\ (ab,T),(ba,T),(bc,U),(cb,U),(ac,V),(ca,V)\}.$$

We have

$$\mathcal{C}.S.a = \{\{(\varepsilon,T),(\varepsilon,V)\},\{(b,T),(\varepsilon,V)\},\{(\varepsilon,T),(c,V)\},\{(b,T),(c,V)\}\},$$
$$\mathcal{C}.S.b = \{\{(\varepsilon,T),(\varepsilon,U)\},\{(a,T),(\varepsilon,U)\},\{(\varepsilon,T),(c,U)\},\{(a,T),(c,U)\}\},$$
$$\mathcal{C}.S.c = \{\{(\varepsilon,U),(\varepsilon,V)\},\{(b,U),(\varepsilon,V)\},\{(\varepsilon,U),(a,V)\},\{(b,U),(a,V)\}\}.$$

A possible enumeration f is

$$f.a.\{(\varepsilon,T),(\varepsilon,V)\} = f.b.\{(\varepsilon,T),(\varepsilon,U)\} = f.c.\{(\varepsilon,U),(\varepsilon,V)\} = 0,$$
$$f.a.\{(b,T),(\varepsilon,V)\} = f.b.\{(a,T),(\varepsilon,U)\} = f.c.\{(b,U),(\varepsilon,V)\} = 1,$$
$$f.a.\{(\varepsilon,T),(c,V)\} = f.b.\{(\varepsilon,T),(c,U)\} = f.c.\{(\varepsilon,U),(a,V)\} = 2,$$
$$f.a.\{(b,T),(c,V)\} = f.b.\{(a,T),(c,U)\} = f.c.\{(b,U),(a,V)\} = 3.$$

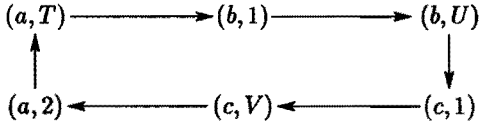
The set of action occurrences of S is

$$Act.S = \{(a,i) \mid 0 \le i \le 3\} \cup \{(b,i) \mid 0 \le i \le 3\} \cup \{(c,i) \mid 0 \le i \le 3\}.$$

Transition relation $\rightarrow$ is

$$\begin{aligned}
&(\varepsilon,T) \rightarrow (a,i) \rightarrow (a,T), && i = 0 \vee i = 2,\\
&(\varepsilon,V) \rightarrow (a,i) \rightarrow (a,V), && i = 0 \vee i = 1,\\
&(\varepsilon,T) \rightarrow (b,i) \rightarrow (b,T), && i = 0 \vee i = 2,\\
&(\varepsilon,U) \rightarrow (b,i) \rightarrow (b,U), && i = 0 \vee i = 1,\\
&(\varepsilon,U) \rightarrow (c,i) \rightarrow (c,U), && i = 0 \vee i = 2,\\
&(\varepsilon,V) \rightarrow (c,i) \rightarrow (c,V), && i = 0 \vee i = 1,\\
&(b,T) \rightarrow (a,i) \rightarrow (ba,T), && i = 1 \vee i = 3,\\
&(a,T) \rightarrow (b,i) \rightarrow (ab,T), && i = 1 \vee i = 3,\\
&(b,U) \rightarrow (c,i) \rightarrow (bc,U), && i = 1 \vee i = 3,\\
&(c,V) \rightarrow (a,i) \rightarrow (ca,V), && i = 2 \vee i = 3,\\
&(c,U) \rightarrow (b,i) \rightarrow (cb,U), && i = 2 \vee i = 3,\\
&(a,V) \rightarrow (c,i) \rightarrow (ac,V), && i = 2 \vee i = 3.
\end{aligned}$$

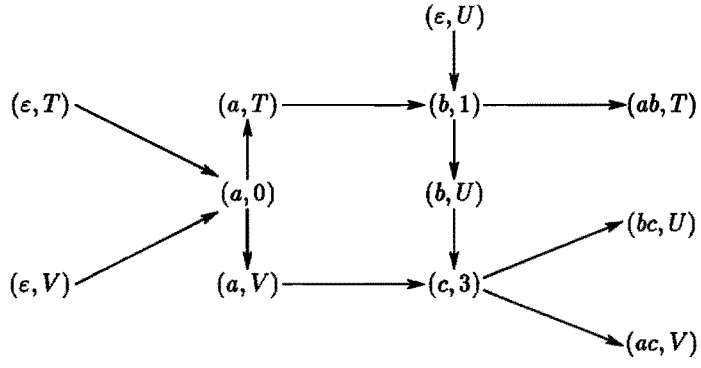
Then $\rightarrow$ contains a cycle, viz.,



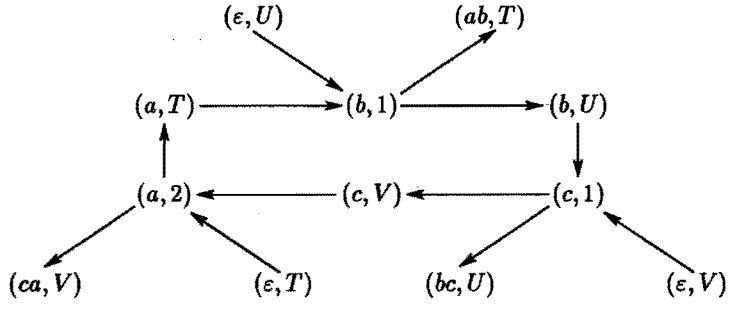
Hence, $\longrightarrow^*$ is not antisymmetric, and thus it is not a partial order.

□

The above example also serves the purpose of showing that we have to be careful when formally defining a behaviour of a system. We first explain what is meant by a full behaviour of a system ([Rel]). Intuitively, it is a maximal part of $\mathcal{O}[\![S]\!]$, such that *Init.S* is included in it, every configuration and every action occurrence is reachable from an initial configuration, and there are no choices (every configuration has at most one successor in it). An exemplary full behaviour from Example 2.1.8 is



But another part of $\mathcal{O}[\![S]\!]$ that also satisfies our informal definition is (see the cycle in Example 2.1.8 for comparison)



It clearly is not a behaviour of system S because traces of the final configurations (configurations that have no successors) cannot be combined into one global trace describing which actions have taken place. Such parts of $\mathcal{O}[\![S]\!]$ should be excluded as unrealistic ones — system S cannot behave accordingly to them. So we have to be careful about the formal definition of a behaviour of a system. The remainder of this section addresses that issue. First some intermediate results are described.

The following property shows a relationship between traces belonging to configurations that are in the relation $\rightarrow^*$, with exactly one action occurrence between them.

Property 2.1.9

For system S, $\{(t,T),(t',T')\} \subseteq Conf.S$, and $(a,C) \in Act.S$

$$(t,T) \rightarrow (a,C) \rightarrow (t',T') \;\Rightarrow\; t'\restriction \mathbf{a}T = (t\restriction \mathbf{a}T')a.$$

Proof

From Definition 2.1.7 we have

$$(t,T) \rightarrow (a,C) \;\Leftrightarrow\; (t,T) \in C$$

and

$$(a,C) \rightarrow (t',T') \;\Leftrightarrow\; (\exists u' : t' = u'a : (u',T') \in C).$$

Let $(u',T') \in C$ such that $t' = u'a$. We derive

$$\begin{array}{ll} & (t,T) \in C \wedge (u',T') \in C \\ \Rightarrow & \{\, C \in \mathcal{C}.S.a \,\} \\ & (\exists u : ua \in \mathbf{t}S : u\restriction \mathbf{a}T = t \wedge u\restriction \mathbf{a}T' = u') \wedge a \in \mathbf{a}T \\ \Rightarrow & \{\, \text{Lemma 1.1.15, property of } \restriction \,\} \\ & (\exists u : ua \in \mathbf{t}S : u\restriction (\mathbf{a}T \cap \mathbf{a}T') = t\restriction \mathbf{a}T' \wedge u\restriction (\mathbf{a}T \cap \mathbf{a}T') = u'\restriction \mathbf{a}T) \wedge a \in \mathbf{a}T \\ \Rightarrow & \{\, \text{predicate calculus} \,\} \\ & u'\restriction \mathbf{a}T = t\restriction \mathbf{a}T' \wedge a \in \mathbf{a}T \\ \Rightarrow & \{\, \text{calculus, definition of } \restriction,\ t' = u'a \,\} \\ & t'\restriction \mathbf{a}T = (t\restriction \mathbf{a}T')a. \end{array}$$

□

In the sequel, for subset π of $Conf.S \cup Act.S$ the following abbreviations are used:

$Conf.\pi$ for $\pi \cap Conf.S$, and
$Act.\pi$ for $\pi \cap Act.S$.

The behaviour of a system is characterized by all possible *computations*. Each computation contains only those configurations and action occurrences that can appear in a single full execution (run) of the system, where the choices have been made, and independent actions are all included. They correspond to observations of [Ma1].

We introduce two notions used for defining computations. The first one concerns branching of elements of $Conf.S \cup Act.S$ ([Re1]).

Definition 2.1.10

Let π be a subset of $Conf.S \cup Act.S$, and let $\varphi \in \pi$. We say that φ is

- *backward branched* in π if $(\exists x, y : \{x, y\} \subseteq \pi \wedge x \neq y : x \rightarrow \varphi \wedge y \rightarrow \varphi)$,
- *forward branched* in π if $(\exists x, y : \{x, y\} \subseteq \pi \wedge x \neq y : \varphi \rightarrow x \wedge \varphi \rightarrow y)$.

□

Forward branched configurations in $Conf.S \cup Act.S$ represent choices in processes (nondeterminism) belonging to S. Forward branched action occurrences represent synchronization between processes of S.

Observe that for $\pi = Conf.S \cup Act.S$, and action occurrence α of π

$$\alpha \text{ is forward branched in } \pi \Leftrightarrow \alpha \text{ is backward branched in } \pi.$$

Even stronger, the number of configurations that directly precede α equals that number of configurations that are its direct successors. That is,

$$(\#x : x \in Conf.S : x \rightarrow \alpha) = (\#x : x \in Conf.S : \alpha \rightarrow x).$$

Hence, in the context of action occurrences it is sufficient to simply speak about branching, without further qualification.

In case of our exemplary system S (see page 40 for $\mathcal{O}[\![S]\!]$) we have that

- configuration (ε, T) is forward branched,
- configuration $(bc, \mathcal{U})$ is backward branched,
- action occurrence $(c, 0)$ is forward and backward branched, or simply, branched

in $Conf.\mathcal{S} \cup Act.\mathcal{S}$.

The second notion necessary for the definition of computations characterizes subsets of $Conf.S \cup Act.S$. Subset π of $Conf.S \cup Act.S$ is *admissible* means that, firstly, every configuration (t,T) of it is reachable via a path in π from (ε,T). Secondly, it means that for every action occurrence $(a,C) \in \pi$ action a is enabled in a distributed state D such that $C \subseteq D \subseteq \pi$. This part takes care of the problem described in connection with Example 2.1.8.

Definition 2.1.11

Subset π of $Conf.S \cup Act.S$ is admissible if

$$(\forall T,t,a : (ta,T) \in Conf.\pi : (\exists C : (a,C) \in Act.\pi : (t,T) \in C))$$
$$\wedge\, (\forall a,C : (a,C) \in Act.\pi : (\exists u : ua \in \mathbf{t}S : C \subseteq \{(u\lceil \mathbf{a}T, T) \mid T \in \mathbf{p}S\} \subseteq \pi)).$$

□

Examples of admissible subsets of $Conf.\mathcal{S} \cup Act.\mathcal{S}$ are

- $\emptyset$ and $Conf.\mathcal{S} \cup Act.\mathcal{S}$,
- $\{(\varepsilon,T),(a,0),(c,0),(\varepsilon,\mathcal{U}),(b,0),(b,\mathcal{U})\}$,
- $\{(\varepsilon,T),(a,0),(a,T),(\varepsilon,\mathcal{U}),(b,0),(b,\mathcal{U}),(c,1),(ac,T),(bc,\mathcal{U})\}$.

An alternative way of checking upon forward branched configurations in admissible subsets of $Conf.S \cup Act.S$ is presented in the following property.

Property 2.1.12

For admissible subset π of $Conf.S \cup Act.S$

$$(\forall \varphi : \varphi \in Conf.\pi : \varphi \text{ is not forward branched in } \pi)$$
$$\Leftrightarrow$$
$$(\forall a,a',C,C' : \{(a,C),(a',C')\} \subseteq Act.\pi \wedge C \cap C' \neq \emptyset : (a,C) = (a',C')).$$

Proof

We derive

$$(\forall a,a',C,C' : \{(a,C),(a',C')\} \subseteq Act.\pi \wedge C \cap C' \neq \emptyset : (a,C) = (a',C'))$$
$$\Leftrightarrow \quad \{\pi \text{ is admissible, hence, } (a,C) \in Act.\pi \Rightarrow C \subseteq \pi, \text{ set and predicate}$$

calculus }

$(\forall a, a', C, C' : \{(a, C), (a', C')\} \subseteq Act.\pi$
$\wedge (\exists t, T : (t, T) \in Conf.\pi : (t, T) \in C \cap C')$
$: (a, C) = (a', C'))$

$\Leftrightarrow$ { predicate calculus }

$(\forall a, a', C, C', t, T : \{(a, C), (a', C')\} \subseteq Act.\pi \wedge (t, T) \in Conf.\pi \wedge (t, T) \in C \cap C'$
$: (a, C) = (a', C'))$

$\Leftrightarrow$ { Definition 2.1.10, definition of $\rightarrow$ }

$(\forall \varphi : \varphi \in Conf.\pi : \varphi$ is not forward branched in $\pi)$.

□

By means of computations, formally defined below, full behaviours of systems are described.

Definition 2.1.13

A computation of system S is a maximal (with respect to set inclusion) subset π of $Conf.S \cup Act.S$ such that

- π is admissible,
- no configuration is forward branched in π.

□

A computation is meant to describe the actual behaviour of a mechanism at work, where choices have already been made and actions have actually taken place. That is why we require admissibility and no forward branched configurations in computations. The set V of all admissible subsets of $Conf.S \cup Act.S$ that have no forward branched configurations forms — with set inclusion — a partial order. Since every chain of this set has an upper bound in it, which is proved below, on account of Zorn's lemma ([Wec]), computations exist.

Lemma 2.1.14

Let V be the set of all admissible subsets of $Conf.S \cup Act.S$ that have no forward branched configurations. Let Y be a chain of $(V, \subseteq)$ and $\pi' = (\cup \pi : \pi \in Y : \pi)$. Then

$\pi' \in V$.

Proof

We first prove that π' is admissible (see Definition 2.1.11).
For configurations we derive

$$
\begin{array}{ll}
 & (ta,T) \in Conf.\pi' \\
\Rightarrow & \{\text{definitions of } \pi' \text{ and of } Conf\} \\
 & (\exists \pi : \pi \in Y : (ta,T) \in Conf.\pi) \\
\Rightarrow & \{(\forall \pi : \pi \in Y : \pi \text{ is admissible}), \text{Definition 2.1.11}\} \\
 & (\exists \pi, C : \pi \in Y \wedge (a,C) \in Act.\pi : (t,T) \in C) \\
\Rightarrow & \{\text{definitions of } \pi' \text{ and of } Act\} \\
 & (\exists C : (a,C) \in Act.\pi' : (t,T) \in C).
\end{array}
$$

For action occurrences we derive

$$
\begin{array}{ll}
 & (a,C) \in Act.\pi' \\
\Rightarrow & \{\text{definitions of } \pi' \text{ and of } Act\} \\
 & (\exists \pi : \pi \in Y : (a,C) \in Act.\pi) \\
\Rightarrow & \{(\forall \pi : \pi \in Y : \pi \text{ is admissible}), \text{Definition 2.1.11}\} \\
 & (\exists \pi, u : \pi \in Y \wedge ua \in \mathbf{t}S : C \subseteq \{(u \lceil \mathbf{a}T, T) \mid T \in \mathbf{p}S\} \subseteq \pi) \\
\Rightarrow & \{\text{definition of } \pi'\} \\
 & (\exists u : ua \in \mathbf{t}S : C \subseteq \{(u \lceil \mathbf{a}T, T) \mid T \in \mathbf{p}S\} \subseteq \pi').
\end{array}
$$

Next we prove that π' has no forward branched configurations. We derive

$$
\begin{array}{ll}
 & (a,C) \in Act.\pi' \wedge (a',C') \in Act.\pi' \wedge C \cap C' \neq \emptyset \\
\Rightarrow & \{Y \text{ is a chain}\} \\
 & (\exists \pi : \pi \in Y : (a,C) \in Act.\pi \wedge (a',C') \in Act.\pi \wedge C \cap C' \neq \emptyset) \\
\Rightarrow & \{(\forall \pi : \pi \in Y : \pi \text{ has no forward branched configurations}), \text{Definition 2.1.10}, \\
 & \quad \text{Property 2.1.12, predicate calculus}\} \\
 & (a,C) = (a',C').
\end{array}
$$

Hence, on account of Property 2.1.12, π' has no forward branched configurations.

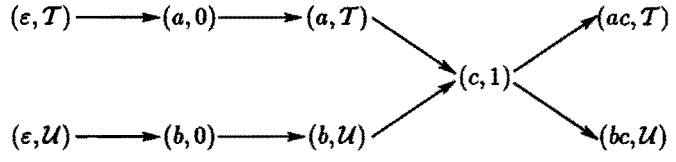
□

As an illustration, we mention computations of $\mathcal{S}$

- $\{(\varepsilon,T),(a,0),(a,T),(\varepsilon,\mathcal{U}),(b,0),(b,\mathcal{U}),(c,1),(ac,T),(bc,\mathcal{U})\}$,

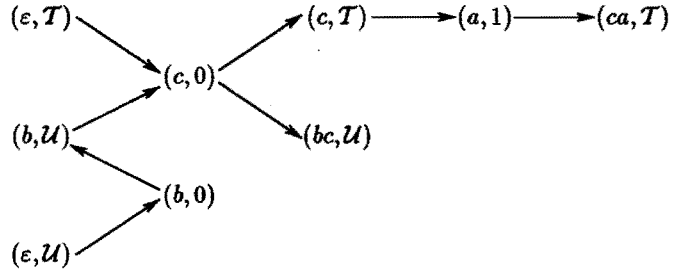
- $\{(\varepsilon,\mathcal{T}),(c,0),(c,\mathcal{T}),(\varepsilon,\mathcal{U}),(b,0),(b,\mathcal{U}),(a,1),(ca,\mathcal{T}),(bc,\mathcal{U})\}$.

A more appealing form of these computations might be a picture. The first computation is then illustrated by



This is a possible behaviour of system $\mathcal{S}$: first, $\mathcal{T}$ and $\mathcal{U}$ independently perform a and b, respectively, after that they synchronize on c.

The second computation is illustrated by



In this computation, first process $\mathcal{U}$ performs b, then $\mathcal{T}$ and $\mathcal{U}$ synchronize on c, and finally, $\mathcal{T}$ performs a.

We define relation $\rightarrow_\pi$ to be the restriction of $\rightarrow$ to the elements of π.

With every subset of $Conf.S \cup Act.S$ a set of traces is associated.

Definition 2.1.15

Let S be a system and π a subset of $Conf.S \cup Act.S$. We define the function $\mathbf{tr}_S : \mathcal{P}.(Conf.S \cup Act.S) \rightarrow \mathcal{P}.\mathbf{t}S$ by

$$\mathbf{tr}_S.\pi = \{t \mid t \in \mathbf{t}S \wedge \{(t\restriction \mathbf{a}T, T) \mid T \in \mathbf{p}S\} \subseteq \pi\}.$$

The set of traces associated with p is then defined to be $\mathbf{tr}_S.\pi$.

□

In the sequel, we usually omit S in $\mathbf{tr}_S$.

Because computations form partially ordered sets (see Theorem A.1.8 of Section A.1 in the Appendix) they are also called *partial-order computations*, and the set of computations of system S is denoted by $poc.S$.

With a non-sequential but deterministic system only one partial-order computation is associated, while there are — in general — many computations resulting from modelling parallelism by means of interleaving. Nondeterministic systems generate many partial-order computations.

Notice that, for the time being, external alphabets of systems are of no importance. This also means that the distributed states, the operational model, and the partial-order computations depend only on the set of processes of the system. This is expressed by the following property.

Property 2.1.16

For set Y of processes, and subsets E and E' of $\mathbf{a}(\mathbf{W}.Y)$

- $\mathcal{D}.\langle E, Y\rangle = \mathcal{D}.\langle E', Y\rangle$,
- $\mathcal{O}[\![\langle E, Y\rangle]\!] = \mathcal{O}[\![\langle E', Y\rangle]\!]$,
- $poc.\langle E, Y\rangle = poc.\langle E', Y\rangle$.

□

The purpose of our operational model is to have a description of the behaviour of a system that maintains full information about concurrency. This information is crucial if one wants to make statements about guaranteed behaviour (progress) of the system, e.g., statements like "after trace t action a will certainly happen". To illustrate the relevance of the introduced operational model to this issue we present the following example.

Example 2.1.17

Let

$$T = \mathbf{pr}(b^* \,;\, e),$$

$$U = \mathbf{pr}((c \mid e) \,;\, a)$$

and

$$S = \langle \{a\}, \{T, U\} \rangle.$$

A possible enumeration f is

$$
\begin{aligned}
&f.a.\{(c,U)\} = 0,\\
&f.a.\{(e,U)\} = 1,\\
&f.c.\{(\varepsilon,U)\} = 0,\\
&f.e.\{(\varepsilon,U),(b^i,T)\} = i, && i \geq 0,\\
&f.b.\{(b^i,T)\} = i, && i \geq 0.
\end{aligned}
$$

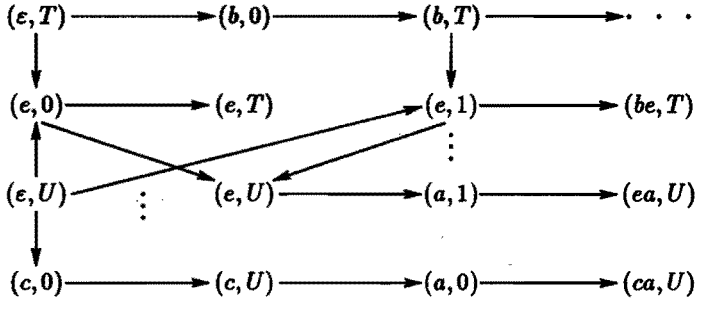
The operational model of S is then defined by

$$\mathcal{O}[S] = (Conf.S, Act.S, \rightarrow),$$

where

- $Conf.S = \{(b^i,T),(b^ie,T) \mid i \geq 0\} \cup \{(\varepsilon,U),(e,U),(ea,U),(c,U),(ca,U)\}$,
- $Act.S = \{(a,0),(a,1),(c,0)\} \cup \{(e,i),(b,i) \mid i \geq 0\}$,
- $(b^i,T) \rightarrow (b,i) \rightarrow (b^{i+1},T)$, $i \geq 0$,
  $(b^i,T) \rightarrow (e,i) \rightarrow (b^ie,T)$, $i \geq 0$,
  $(\varepsilon,U) \rightarrow (e,i) \rightarrow (e,U)$, $i \geq 0$,
  $(e,U) \rightarrow (a,1) \rightarrow (ea,U)$,
  $(\varepsilon,U) \rightarrow (c,0) \rightarrow (c,U) \rightarrow (a,0) \rightarrow (ca,U)$.

In a graphical form we can expose only a part of $\mathcal{O}[S]$.



One can easily check that (ε,T) is a forward branched configuration and that (e,U) is a backward branched configuration.

Examples of partial-order computations are

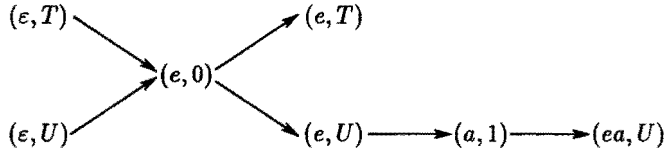
- $\{(b^i, T) \mid i \geq 0\} \cup \{(b, i) \mid i \geq 0\} \cup \{(\varepsilon, U), (c, U), (ca, U)\} \cup \{(c, 0), (a, 0)\}$,
- $\{(\varepsilon, T), (\varepsilon, U), (e, T), (e, U), (ea, U)\} \cup \{(e, 0), (a, 1)\}$.

Graphically, (a part of) the first computation is represented by

$$(\varepsilon, T) \longrightarrow (b, 0) \longrightarrow (b, T) \longrightarrow (b, 1) \longrightarrow \cdots$$

$$(\varepsilon, U) \longrightarrow (c, 0) \longrightarrow (c, U) \longrightarrow (a, 0) \longrightarrow (ca, U)$$

The second computation is represented by



Considering the failures model (of Section 1.4) of system S results in the conclusion that $(\varepsilon, \{a\}) \in \mathbf{f}S$. In connection with progress considerations this is interpreted as the possibility that a may never happen while S is executing. From our operational model of S, intuitively, an opposite conclusion can be drawn. Since in every partial-order computation of S an occurrence of action a is included (there is a trace t associated with that computation for which $t\lceil a = a$ holds), a is guaranteed to happen.

□

The observation concerning occurrences of action a in every partial-order computation of system S from Example 2.1.17 has a direct implication, namely, that each computation has a configuration with a belonging to its trace. This observation gave the inspiration for the definition of acceptances in Section 2.2.

2.2 A characterization of external behaviour — acceptances

Intuitively, a proper characterization of a system's external behaviour should allow us to distinguish between nondeterminism and concurrency. If the distinction between both phenomena can be made, precise statements about guaranteed behaviour of the system can be formulated. To explain this more precisely we present two simple examples of systems that should be distinguished in our opinion.

Example 2.2.1

Consider processes $T = \mathbf{pr}(a)$ and $U = \mathbf{pr}(x^*)$. Let

$$S_0 = \langle \{a\}, \{T \mathbf{w} U\} \rangle,$$

and

$$S_1 = \langle \{a\}, \{T, U\} \rangle.$$

System S_0 consists of only one process. This process can choose to perform an unbounded number of x's; in other words, it suffers from divergence. In system S_1 however, a and x are independent of each other, so even if U keeps on performing x's, external action a will take place. In the failures model of Section 1.4 these systems are identified. In the operational model of Section 2.1 both systems have different sets of computations. System S_0 has infinitely many computations, including one that gives only $\{\varepsilon\}$ if its set of traces is projected on $\{a\}$. On the other hand, S_1 has precisely one computation, viz., the one with $\{\varepsilon, a\}$ resulting from the projection of its set of traces on $\{a\}$. From this result it can be concluded that during the execution of S_0 action a may never happen, whereas it certainly will happen during the execution of S_1. Hence, it is worthwhile to have a closer look at external traces of computations.

□

Example 2.2.2

Consider processes $T = \mathbf{pr}(a)$ and $U = \mathbf{pr}(b)$. Let

$$S_0 = \langle \{a, b\}, \{T \mathbf{w} U\} \rangle,$$

and

$$S_1 = \langle \{a, b\}, \{T, U\} \rangle.$$

Since the choice is nondeterministic, we get two computations in the case of the mechanism specified by system S_0, with associated sets of external traces $\{\varepsilon, a, ab\}$ and $\{\varepsilon, b, ba\}$, respectively. In the case of the mechanism specified by system S_1, a and b are independent of each other, which results in only one computation with which the set $\{\varepsilon, a, b, ab, ba\}$ of external traces is associated. Thus the difference between S_0 and S_1 can also be detected from the sets of external traces of computations.

In the failures model of Section 1.4 both systems are identified.

□

Since we are interested in guaranteed behaviour in every state of S, or more precisely, for every trace $t \in \mathbf{t}(\mathbf{pr}S)$, we consider sets of external traces containing guaranteed extensions of t. The smaller these sets are, the more information about the system we have.

The following example shows that it is important to consider sets of traces, because sets of actions are not sufficient.

Example 2.2.3

Consider systems

$$S_0 = (\{a,b\}, \{\mathbf{pr}(b^*\,;\, a)\}) \qquad \text{and} \qquad S_1 = (\{a,b\}, \{T\}),$$

where process T is defined by

$$T = (\{a,b\} \cup \{x_i \mid i \geq 0\}, \{t \mid (\exists i : i \geq 0 : t \leq x_i b^i a)\}).$$

The external processes of both systems are equal. Consider sets of actions as guaranteed extensions. Then $\{a,b\}$ is the guaranteed extension for every external trace t, such that $t\restriction a = \varepsilon$, in both systems. System S_1 has failures that S_0 does not have, for instance $(\varepsilon, \{a\})$. It is a failure of S_1 because for $u = x_1$ we have (see Section 1.4)

$$u \in \mathbf{t}S_1 \wedge u\restriction\{a,b\} = \varepsilon \wedge \mathit{dis}.u.a.S_1.$$

On the other hand, every failure of S_0 is also a failure of S_1. From this one can conclude that in the failures model S_0 implements S_1. However, considering sets of traces as guaranteed extensions leads to a different conclusion. Namely, as $\{b^i a \mid 0 \leq i\}$ is a guaranteed extension for every external trace t, such that $t\restriction a = \varepsilon$, in S_1 and not in S_0 (S_0 has a computation in which a does not occur), from our point of view S_0 should not be considered as an implementation of S_1.

□

The guaranteed extensions are called *acceptances*. The acceptances as defined here differ from the acceptances of [Hen].

Definition 2.2.4

Let $t \in \mathbf{t}(\mathbf{pr}S)$ and $L \subseteq \mathbf{e}S^*$. Pair (t, L) is an acceptance of system S if

$$(\forall \pi : \pi \in \mathit{poc}.S \wedge t \in \mathbf{tr}.\pi\restriction\mathbf{e}S : \{v \mid tv \in \mathbf{tr}.\pi\restriction\mathbf{e}S\} \cap L \neq \varnothing).$$

□

In the sequel, we denote the set of acceptances of system S by $\mathbf{ac}S$.
For the systems of Example 2.2.1 we have

$$\mathbf{ac}S_0 = \{(t, L) \mid (t = \varepsilon \vee t = a) \wedge L \subseteq \{a\}^* \wedge \varepsilon \in L\}$$

and

$$\mathbf{ac}S_1 = \{(\varepsilon, L) \mid L \subseteq \{a\}^* \wedge (\varepsilon \in L \vee a \in L)\} \cup \{(a, L) \mid L \subseteq \{a\}^* \wedge \varepsilon \in L\}.$$

For the systems of Example 2.2.2 we have

$$\begin{aligned}\mathbf{ac}S_0 =& \{(\varepsilon, L) \mid L \subseteq \{a, b\}^* \wedge L \cap \{\varepsilon, a, ab\} \neq \emptyset \wedge L \cap \{\varepsilon, b, ba\} \neq \emptyset\} \\ & \cup \{(a, L) \mid L \subseteq \{a, b\}^* \wedge (\varepsilon \in L \vee b \in L)\} \\ & \cup \{(b, L) \mid L \subseteq \{a, b\}^* \wedge (\varepsilon \in L \vee a \in L)\} \\ & \cup \{(t, L) \mid (t = ab \vee t = ba) \wedge L \subseteq \{a, b\}^* \wedge \varepsilon \in L\}\end{aligned}$$

and

$$\begin{aligned}\mathbf{ac}S_1 =& \{(\varepsilon, L) \mid L \subseteq \{a, b\}^* \wedge L \cap \{\varepsilon, a, b, ab, ba\} \neq \emptyset\} \\ & \cup \{(a, L) \mid L \subseteq \{a, b\}^* \wedge (\varepsilon \in L \vee b \in L)\} \\ & \cup \{(b, L) \mid L \subseteq \{a, b\}^* \wedge (\varepsilon \in L \vee a \in L)\} \\ & \cup \{(t, L) \mid (t = ab \vee t = ba) \wedge L \subseteq \{a, b\}^* \wedge \varepsilon \in L\}.\end{aligned}$$

The following property expresses, among other things, the fact that it is sufficient to consider minimal acceptances.

Property 2.2.5

For system S, $t \in \mathbf{t}(\mathbf{pr}S)$, and $L \subseteq \mathbf{e}S^*$

- $(t, \{\varepsilon\})$ and $(t, \mathbf{t}(\mathit{after}.t.\mathbf{pr}S))$ are acceptances of every system S,
- $(t, \emptyset)$ is not an acceptance of S,
- $(t, L) \in \mathbf{ac}S \Rightarrow (\forall L' : L \subseteq L' \subseteq \mathbf{e}S^* : (t, L') \in \mathbf{ac}S)$,
- $S' \approx S \Rightarrow \mathbf{ac}S' = \mathbf{ac}S$.

□

Because $\mathbf{ac}S' = \mathbf{ac}S$, for $S' \approx S$, the following definition is sound.

Definition 2.2.6

For system S

$$\mathbf{ac}[S] = \mathbf{ac}S.$$

□

Equipped with additional tools to characterize systems, we return to Example 2.1.17 of the previous section. After having calculated the set of partial-order computations of S, we are able to conclude that $(\epsilon, \{a\})$ is an acceptance of that system. An interpretation of this result is that this particular action certainly will take place during the execution of system S (in the initial state). In general, it would not be possible to prove a similar assertion in an interleaving operational model (as, e.g., in [Paw] and [Hen]) without employing fairness assumptions (see Section A.3 in the Appendix).

Next, a few examples are presented to show the relationship between acceptances and a characterization of systems by means of the failures model of Section 1.4.

System S of Example 2.1.17 is non-disabling and divergent, according to Definitions 1.3.2 and 1.3.3. A consequence of the divergence is that $(\epsilon, \{a\})$ is a failure of S, thus a may never happen. Because $(\epsilon, \{a\}) \in \mathbf{ac}S$ (thus a will certainly happen), we conclude that the characterization of S given by the failures model does not agree with its operational model. Thus we can give a more suitable characterization of divergence in our operational model. Notice however, that acceptances alone do not always preserve information about disabling or divergence, as can be observed in the following example.

Example 2.2.7

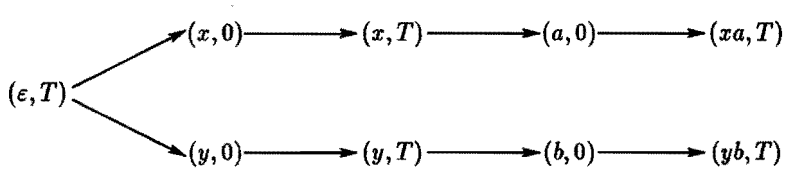
Let

$$T = \mathbf{pr}(x\,;\, a \mid y\,;\, b)$$

and

$$S_0 = \langle \{a, b\}, \{T\} \rangle.$$

Then $\mathcal{O}[\![S_0]\!]$ can be represented by



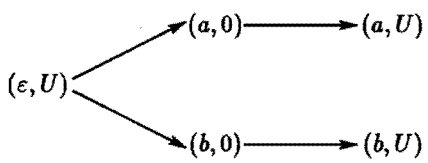
We can compute that $(\varepsilon, \{a, b\}) \in \mathbf{ac}S_0$, $(\varepsilon, \{a\}) \notin \mathbf{ac}S_0$, and $(\varepsilon, \{b\}) \notin \mathbf{ac}S_0$ holds. For a similar system S_1

$$S_1 = \langle \{a, b\}, \{U\} \rangle,$$

where

$$U = \mathbf{pr}(a \mid b),$$

we get the following as the representation of $\mathcal{O}[\![S_1]\!]$



Acceptances of S_1 are equal to those of S_0, thus for S_1 the same conclusions as for S_0 can be drawn. But then, according to the failures model, $(\varepsilon, \{a\})$ and $(\varepsilon, \{b\})$ are both failures of S_0 and not of S_1, while S_0 and S_1 cannot be distinguished considering acceptances only. Information about disabling gets lost in this case.

A similar information loss occurs in the case of system S_2 defined by

$$S_2 = \langle \{a, b\}, \{\mathbf{pr}((x \mid x'; y^*); a), \mathbf{pr}((x' \mid x; z^*); b), \mathbf{pr}(a \mid b)\} \rangle.$$

Acceptances of this system are also equal to that of S_0 (and, of course, of S_1). For failures we have $\mathbf{f}S_0 \subset \mathbf{f}S_2$, because $(\varepsilon, \{a, b\}) \in \mathbf{f}S_2$ and $(\varepsilon, \{a, b\}) \notin \mathbf{f}S_0$. This particular failure of S_2 is caused by divergence as defined in Section 1.4.

□

In the case of the system in the following example, the information about disabling can be found in the set of acceptances as well.

Example 2.2.8

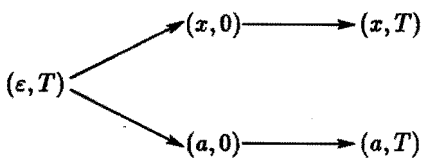
Let

$$S_0 = \langle \{a\}, \{T\} \rangle,$$

where

$$T = \mathbf{pr}(x \mid a).$$

For $\mathcal{O}[\![S_0]\!]$ we then have



We can now compute that $(\varepsilon, \{\varepsilon, a\}) \in \mathbf{ac}S_0$ and $(\varepsilon, \{a\}) \notin \mathbf{ac}S_0$, and from the failures model we deduce that $(\varepsilon, \{a\}) \in \mathbf{f}S_0$, so in this case the same conclusions can be drawn from both characterizations (that a is not guaranteed to happen).

Intuitively, system S_0 cannot be an implementation of S_1 that is defined by

$$S_1 = \langle \{a\}, \{\mathbf{pr}a\} \rangle,$$

where $(\varepsilon, \{a\}) \in \mathbf{ac}S_1$ (and $(\varepsilon, \{a\}) \notin \mathbf{f}S_1$).

□

These small examples make clear why sets of acceptances are not enough to distinguish between systems.

On the other hand, with acceptances the distinction between concurrency and interleaving can easily be made. The next example shows this distinction for systems without divergence and without disabling.

Example 2.2.9

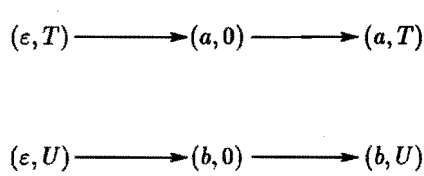
Let

$$T = \mathbf{pr}(a),$$
$$U = \mathbf{pr}(b),$$
$$S_0 = \langle \{a, b\}, \{T, U\} \rangle$$

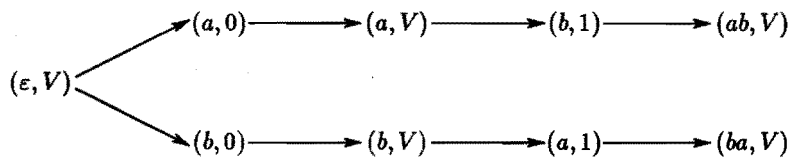
and

$$S_1 = \langle \{a, b\}, \{T \mathbf{w} U\} \rangle.$$

The graphical representation of $\mathcal{O}[\![S_0]\!]$ is the following



We compare $\mathcal{O}[\![S_0]\!]$ with $\mathcal{O}[\![S_1]\!]$ that is represented by



where V stands for $T \mathbf{w} U$.

The difference between $\mathcal{O}[\![S_0]\!]$ and $\mathcal{O}[\![S_1]\!]$ results also in different acceptances. Namely, $\mathbf{ac}S_1 \subset \mathbf{ac}S_0$, because $(\varepsilon, \{a\}) \in \mathbf{ac}S_0 \wedge (\varepsilon, \{a\}) \notin \mathbf{ac}S_1$ (the same holds for $(\varepsilon, \{b\})$).

□

The notion of acceptances we introduced in this section represents those aspects of system behaviour that are of importance in our approach. However, it turned out that taking only them into account causes undesired identifications (Example 2.2.7). In Section 2.4, a system equivalence is defined that is based on acceptances, and that resolves the problem just mentioned.

2.3 A simpler characterization of acceptances

On account of Definition 2.2.4, it is sufficient to consider sets of external traces associated with partial-order computations for the purpose of determining the acceptances of a system. In this section, a different definition of these sets of traces is presented. This definition does not make use of the operational model. By Corollary 2.3.21, which in turn is a consequence of Theorem 2.3.20, we show that both definitions (Definition 2.1.15 and the alternative definition of this section) coincide.

At the end of this section, a few results are included that refer to conservative processes and that are used in Section 2.5.

A few new notions are introduced. The first one is the notion of a *prefix of a computation.*

Definition 2.3.1

A prefix of a partial-order computation of system S is defined to be a subset π of $Conf.S \cup Act.S$ satisfying

- $\{(\varepsilon, T) \mid T \in \mathbf{p}S\} \subseteq \pi$,

- $(\forall T, t, a : (ta, T) \in Conf.\pi : (\exists C : (a, C) \in Act.\pi : (t, T) \in C))$,
- $(\forall a, C : (a, C) \in Act.\pi : (\exists u : ua \in \mathbf{t}S : C \subseteq \{(u\restriction \mathbf{a}T, T) \mid T \in \mathbf{p}S\} \subseteq \pi \wedge \{(ua\restriction \mathbf{a}T, T) \mid T \in \mathbf{p}S\} \subseteq \pi))$,
- $(\forall a, a', C, C' : \{(a, C), (a', C')\} \subseteq Act.\pi \wedge C \cap C' \neq \emptyset : (a, C) = (a', C'))$.

The set of prefixes of partial-order computations of S is denoted by $ppoc.S$.

□

Note that the set of maximal elements of $ppoc.S$ equals $poc.S$.
We refer to the set of traces belonging to a prefix of a partial-order computation as *po-trace*. Definition 2.3.2 provides an alternative formalization of po-traces, which is proved by Corollary 2.3.19.

Definition 2.3.2

A po-trace of system S is defined to be a subset p of $\mathbf{t}S$ that satisfies

- $p \neq \emptyset$,
- $pref.p = p$,
- $(\forall T, t, u, a, b : T \in \mathbf{p}S \wedge \{a, b\} \subseteq \mathbf{a}T \wedge \{ta, ub\} \subseteq p \wedge t\restriction \mathbf{a}T = u\restriction \mathbf{a}T : a = b)$,
- $(\forall t, u : t \in p \wedge u \in \mathbf{t}S \wedge (\forall T : T \in \mathbf{p}S : t\restriction \mathbf{a}T = u\restriction \mathbf{a}T) : u \in p)$.

The set of po-traces of S is denoted by $ppot.S$.
The set of maximal elements of $ppot.S$ is called $pot.S$.

□

The ultimate goal is to prove that $pot.S = \{\mathbf{tr}.p \mid p \in poc.S\}$.
Property 2.3.3 provides an alternative way of specifying the maximal po-traces.

Property 2.3.3

For system S

$$
\begin{array}{l}
\quad p \in pot.S \\
\Leftrightarrow \\
\quad p \in ppot.S \wedge (\forall t, a : ta \in \mathbf{t}S \setminus p \wedge t \in p \\
\qquad\qquad : (\exists T, u, b : T \in \mathbf{p}S \wedge ub \in p \wedge u\restriction \mathbf{a}T = t\restriction \mathbf{a}T \\
\qquad\qquad\quad : \{a, b\} \subseteq \mathbf{a}T \wedge a \neq b)).
\end{array}
$$

□

Observe that for po-trace p of system S and process T of $\mathbf{p}S$, the process $(\mathbf{a}T, p\restriction \mathbf{a}T)$ has at most one symbol in every successor set.

The following results (Lemma 2.3.4 through Corollary 2.3.19) form a preparation for Theorem 2.3.20, which proves that $(ppoc.S, \subseteq)$ is isomorphic to $(ppot.S, \subseteq)$.

For every prefix π of a computation of S, $\mathbf{tr}.\pi$ is a member of $ppot.S$, which is expressed by Corollary 2.3.7 and which is proved by Lemmata 2.3.4, 2.3.5, and 2.3.6.

Lemma 2.3.4

Let π be a prefix of a computation of system S. Then

$$\mathbf{tr}.\pi \neq \emptyset \wedge pref.(\mathbf{tr}.\pi) = \mathbf{tr}.\pi.$$

Proof

It is clear from the definition that $\mathbf{tr}.\pi$ is not empty. We prove that $\mathbf{tr}.\pi$ is prefix-closed. Let $t \in \mathbf{t}S$ and $ta \in \mathbf{tr}.\pi$. We derive

$$
\begin{array}{ll}
& ta \in \mathbf{tr}.\pi \\
\Leftrightarrow & \quad \{\text{Definition 2.1.15}\} \\
& \{(ta\restriction \mathbf{a}T, T) \mid T \in \mathbf{p}S\} \subseteq \pi \\
\Rightarrow & \quad \{\text{Definition 2.3.1, definition of } \restriction\} \\
& \{(t\restriction \mathbf{a}T, T) \mid T \in \mathbf{p}S \wedge a \notin \mathbf{a}T\} \subseteq \pi \\
& \wedge (\forall T : T \in \mathbf{p}S \wedge a \in \mathbf{a}T : (\exists C : (a, C) \in Act.\pi : (t\restriction \mathbf{a}T, T) \in C)) \\
\Rightarrow & \quad \{\text{Definition 2.3.1, predicate calculus}\} \\
& \{(t\restriction \mathbf{a}T, T) \mid T \in \mathbf{p}S \wedge a \notin \mathbf{a}T\} \subseteq \pi \wedge (\forall T : T \in \mathbf{p}S \wedge a \in \mathbf{a}T : (t\restriction \mathbf{a}T, T) \in \pi) \\
\Leftrightarrow & \quad \{\text{calculus}\} \\
& \{(t\restriction \mathbf{a}T, T) \mid T \in \mathbf{p}S\} \subseteq \pi \\
\Leftrightarrow & \quad \{\text{Definition 2.1.15}\} \\
& t \in \mathbf{tr}.\pi.
\end{array}
$$

Hence, $\mathbf{tr}.\pi$ is prefix-closed.

□

Lemma 2.3.5

Let S be a system and π be a prefix of a computation of S. Then for process T of $\mathbf{p}S$, and actions a and b of $\mathbf{a}T$

$$\{ta, ub\} \subseteq \mathbf{tr}.\pi \wedge t\lceil \mathbf{a}T = u\lceil \mathbf{a}T \Rightarrow a = b.$$

Proof

Let $\{ta, ub\} \subseteq \mathbf{tr}.\pi$, such that $t\lceil \mathbf{a}T = u\lceil \mathbf{a}T$. We derive

$$\begin{array}{ll}
& ta \in \mathbf{tr}.\pi \wedge ub \in \mathbf{tr}.\pi \\
\Rightarrow & \{\text{Definition 2.1.15}\} \\
& (ta\lceil \mathbf{a}T, T) \in \pi \wedge (ub\lceil \mathbf{a}T, T) \in \pi \\
\Rightarrow & \{\text{Definition 2.3.1}\} \\
& (\exists C : (a, C) \in Act.\pi : (t\lceil \mathbf{a}T, T) \in C) \wedge (\exists C' : (b, C') \in Act.\pi : (u\lceil \mathbf{a}T, T) \in C') \\
\Leftrightarrow & \{t\lceil \mathbf{a}T = u\lceil \mathbf{a}T\} \\
& (\exists C : (a, C) \in Act.\pi : (t\lceil \mathbf{a}T, T) \in C) \wedge (\exists C' : (b, C') \in Act.\pi : (t\lceil \mathbf{a}T, T) \in C') \\
\Leftrightarrow & \{\text{predicate calculus}\} \\
& (\exists C, C' : (a, C) \in Act.\pi \wedge (b, C') \in Act.\pi : (t\lceil \mathbf{a}T, T) \in C \cap C') \\
\Rightarrow & \{\text{calculus}\} \\
& (\exists C, C' : (a, C) \in Act.\pi \wedge (b, C') \in Act.\pi : C \cap C' \neq \emptyset) \\
\Rightarrow & \{\text{Definition 2.3.1}\} \\
& (\exists C, C' : (a, C) \in Act.\pi \wedge (b, C') \in Act.\pi : (a, C) = (b, C')) \\
\Rightarrow & \{\text{calculus}\} \\
& a = b.
\end{array}$$

□

Lemma 2.3.6

For system S and prefix π of a computation of S

$$(\forall t, u : t \in \mathbf{tr}.\pi \wedge u \in \mathbf{t}S \wedge (\forall T : T \in \mathbf{p}S : t\lceil \mathbf{a}T = u\lceil \mathbf{a}T) : u \in \mathbf{tr}.\pi).$$

Proof

Let $\{t, u\} \subseteq \mathbf{t}S$, such that $(\forall T : T \in \mathbf{p}S : t\lceil \mathbf{a}T = u\lceil \mathbf{a}T)$. We derive

$$
\begin{array}{ll}
& t \in \mathbf{tr}.\pi \\
\Leftrightarrow & \{\text{Definition 2.1.15}\} \\
& \{(t\restriction \mathbf{a}T, T) \mid T \in \mathbf{p}S\} \subseteq \pi \\
\Leftrightarrow & \{(\forall T : T \in \mathbf{p}S : t\restriction \mathbf{a}T = u\restriction \mathbf{a}T)\} \\
& \{(u\restriction \mathbf{a}T, T) \mid T \in \mathbf{p}S\} \subseteq \pi \\
\Leftrightarrow & \{\text{Definition 2.1.15}\} \\
& u \in \mathbf{tr}.\pi.
\end{array}
$$

□

The following corollary expresses the relationship between prefixes of computations and po-traces of a system. It is, on account of Definition 2.3.2, an immediate consequence of the three preceding lemmata.

Corollary 2.3.7

For system S

$$\{\mathbf{tr}.\pi \mid \pi \in ppoc.S\} \subseteq ppot.S.$$

□

From every element p of $ppot.S$, a prefix of a computation of system S can be constructed. This result is expressed by Corollary 2.3.14 and it is proved by Property 2.3.9, Lemmata 2.3.10, 2.3.11, and 2.3.13.

In order to facilitate the proofs, we first introduce some convenient notation.

Definition 2.3.8

For po-trace p of S

- $conf.p = \{(t\restriction \mathbf{a}T, T) \mid t \in p \wedge T \in \mathbf{p}S\}$,
- $act.p = \{(a, \{(t\restriction \mathbf{a}T, T) \mid T \in \mathbf{p}S \wedge a \in \mathbf{a}T\}) \mid ta \in p\}$.

□

Because, on account of Definition 2.3.2, every po-trace p of system S is non-empty and prefix-closed, we have the following property.

Property 2.3.9

Let S be a system. For every po-trace p of S

$$\{(\epsilon, T) \mid T \in \mathbf{p}S\} \subseteq conf.p \cup act.p.$$

□

The above property means that $conf.p \cup act.p$, for a po-trace p of system S, satisfies the first requirement of the Definition 2.3.1. In the next three lemmata we prove that the other requirements of that definition also are satisfied by $conf.p \cup act.p$. Lemmata 2.3.10 and 2.3.11 prove that $conf.p \cup act.p$ is admissible. Lemma 2.3.13 proves that configurations of $conf.p$ are not forward branched.

Lemma 2.3.10

Let S be a system, p a po-trace of S, and $\pi = conf.p \cup act.p$. Then

$$(\forall T, t, a : (ta, T) \in Conf.\pi : (\exists C : (a, C) \in Act.\pi : (t, T) \in C)).$$

Proof

Let $(ta, T) \in conf.p$. Note that $conf.p = Conf.\pi$ and $act.p = Act.\pi$.
We derive

$$\begin{array}{ll} & (ta, T) \in conf.p \\ \Rightarrow & \{\text{definition of } conf.p\} \\ & (\exists u : u \in p : ta = u \restriction \mathbf{a}T) \\ \Rightarrow & \{p \text{ is prefix-closed, Definition 2.3.2, definition of projection}\} \\ & (\exists u : ua \in p : t = u \restriction \mathbf{a}T) \\ \Rightarrow & \{\text{definition of } act.p\} \\ & (\exists u : ua \in p : t = u \restriction \mathbf{a}T \wedge (a, \{(u \restriction \mathbf{a}U, U) \mid U \in \mathbf{p}S \wedge a \in \mathbf{a}U\}) \in act.p) \\ \Rightarrow & \{a \in \mathbf{a}T, \text{calculus}\} \\ & (\exists C : (a, C) \in act.p : (t, T) \in C). \end{array}$$

Hence, because $conf.p = Conf.\pi$ and $act.p = Act.\pi$,

$$(\forall T, t, a : (ta, T) \in Conf.\pi : (\exists C : (a, C) \in Act.\pi : (t, T) \in C)).$$

□

Lemma 2.3.11

Let S be a system, p a po-trace of S, and $\pi = conf.p \cup act.p$. Then for action occurrence (a, C) of $Act.\pi$

$$(\exists u : ua \in \mathbf{t}S : C \subseteq \{(u\restriction \mathbf{a}T, T) \mid T \in \mathbf{p}S\} \subseteq \pi \wedge \{(ua\restriction \mathbf{a}T, T) \mid T \in \mathbf{p}S\} \subseteq \pi).$$

Proof

Let $(a, C) \in act.p$. Note that $conf.p = Conf.\pi$ and $act.p = Act.\pi$.
We derive

$$\begin{array}{ll}
& (a, C) \in act.p \\
\Leftrightarrow & \{\text{definition of } act.p\} \\
& (\exists u : ua \in p : C = \{(u\restriction \mathbf{a}T, T) \mid T \in \mathbf{p}S \wedge a \in \mathbf{a}T\}) \\
\Leftrightarrow & \{\text{definition of } conf.p, \text{Definition 2.3.2 } (p = pref.p)\} \\
& (\exists u : ua \in p : C = \{(u\restriction \mathbf{a}T, T) \mid T \in \mathbf{p}S \wedge a \in \mathbf{a}T\} \\
& \qquad \wedge \{(u\restriction \mathbf{a}T, T) \mid T \in \mathbf{p}S\} \cup \{(ua\restriction \mathbf{a}T, T) \mid T \in \mathbf{p}S\} \subseteq conf.p) \\
\Rightarrow & \{\text{calculus}\} \\
& (\exists u : ua \in p : C \subseteq \{(u\restriction \mathbf{a}T, T) \mid T \in \mathbf{p}S\} \subseteq conf.p \\
& \qquad \wedge \{(ua\restriction \mathbf{a}T, T) \mid T \in \mathbf{p}S\} \subseteq conf.p) \\
\Rightarrow & \{\text{definition of } \pi,\ conf.p = Conf.\pi \text{ and } act.p = Act.\pi,\ p \subseteq \mathbf{t}S\} \\
& (\exists u : ua \in \mathbf{t}S : C \subseteq \{(u\restriction \mathbf{a}T, T) \mid T \in \mathbf{p}S\} \subseteq \pi \wedge \{(ua\restriction \mathbf{a}T, T) \mid T \in \mathbf{p}S\} \subseteq \pi).
\end{array}$$

□

The following result expresses an important property of traces belonging to the same po-trace. It facilitates the proof of Lemma 2.3.13 (and the proof of Property 2.3.15).

Property 2.3.12

For system S and po-trace p of S

$$(\forall T, t, u, a : T \in \mathbf{p}S \wedge a \in \mathbf{a}T \wedge \{ta, ua\} \subseteq p \wedge t\restriction a = u\restriction a : t\restriction \mathbf{a}T = u\restriction \mathbf{a}T).$$

Proof

Let $T \in \mathbf{p}S, a \in \mathbf{a}T$, and $\{ta, ua\} \subseteq p$. We derive

$$\begin{array}{ll}
& t\restriction \mathbf{a}T \neq u\restriction \mathbf{a}T \\
\Rightarrow & \{\text{calculus}\} \\
& ta\restriction \mathbf{a}T \neq ua\restriction \mathbf{a}T
\end{array}$$

$\Leftrightarrow$ {calculus}
$(\exists s, b, c : sb \leq ta\lceil \mathbf{a}T \wedge sc \leq ua\lceil \mathbf{a}T \wedge \{b, c\} \subseteq \mathbf{a}T : b \neq c)$
$\vee\, ta\lceil \mathbf{a}T < ua\lceil \mathbf{a}T \vee ta\lceil \mathbf{a}T > ua\lceil \mathbf{a}T$
$\Leftrightarrow$ {property of projection}
$(\exists s, s', b, c : sb \leq ta \wedge s'c \leq ua \wedge s\lceil \mathbf{a}T = s'\lceil \mathbf{a}T \wedge \{b, c\} \subseteq \mathbf{a}T : b \neq c)$
$\vee\, ta\lceil \mathbf{a}T < ua\lceil \mathbf{a}T \vee ta\lceil \mathbf{a}T > ua\lceil \mathbf{a}T$
$\Leftrightarrow$ {$p \in ppot.S$ and $\{ta, ua\} \subseteq p$, which implies
$(\forall s, s', b, c : sb \leq ta \wedge s'c \leq ua : \{sb, s'c\} \subseteq p)$, Definition 2.3.2}
$ta\lceil \mathbf{a}T < ua\lceil \mathbf{a}T \vee ta\lceil \mathbf{a}T > ua\lceil \mathbf{a}T$
$\Rightarrow$ {property of $\lceil$, $<$ and $\leq$}
$ta\lceil \mathbf{a}T \leq u\lceil \mathbf{a}T \vee t\lceil \mathbf{a}T \geq ua\lceil \mathbf{a}T$
$\Rightarrow$ {$a \in \mathbf{a}T$, property of $\lceil$}
$t\lceil a < u\lceil a \vee t\lceil a > u\lceil a$
$\Rightarrow$ {definition of $<$}
$t\lceil a \neq u\lceil a$.

Hence,

$$(\forall T, t, u, a : T \in \mathbf{p}S \wedge a \in \mathbf{a}T \wedge \{ta, ua\} \subseteq p \wedge t\lceil a = u\lceil a : t\lceil \mathbf{a}T = u\lceil \mathbf{a}T).$$

□

Lemma 2.3.13

Let S be a system, p a po-trace of S, and $\pi = conf.p \cup act.p$. Then no configuration of π is forward branched, that is,

$$(\forall a, a', C, C' : \{(a, C), (a', C')\} \subseteq Act.\pi \wedge C \cap C' \neq \emptyset : (a, C) = (a', C')).$$

Proof

Note that $conf.p = Conf.\pi$ and $act.p = Act.\pi$.
We derive

$\{(a, C), (a', C')\} \subseteq act.p \wedge C \cap C' \neq \emptyset$
$\Leftrightarrow$ {definition of $act.p$, calculus}
$(\exists U, t, t' : U \in \mathbf{p}S \wedge \{a, a'\} \subseteq \mathbf{a}U \wedge \{ta, t'a'\} \subseteq p \wedge t\lceil \mathbf{a}U = t'\lceil \mathbf{a}U$
$: C = \{(t\lceil \mathbf{a}T, T) \mid T \in \mathbf{p}S \wedge a \in \mathbf{a}T\}$
$\wedge C' = \{(t'\lceil \mathbf{a}T, T) \mid T \in \mathbf{p}S \wedge a' \in \mathbf{a}T\})$

$\Rightarrow$ {predicate calculus}
$(\exists U, t, t' : U \in \mathbf{p}S \wedge \{a, a'\} \subseteq \mathbf{a}U \wedge \{ta, t'a'\} \subseteq p : t\restriction \mathbf{a}U = t'\restriction \mathbf{a}U)$
$\Rightarrow$ {$p \in ppot.S$, Definition 2.3.2}
$a = a'$.

Let $\{(a, C), (a', C')\} \subseteq act.p$, and let t and t' satisfy

$$ta \in p \wedge t'a' \in p \wedge C = \{(t\restriction \mathbf{a}T, T) \mid T \in \mathbf{p}S \wedge a \in \mathbf{a}T\}$$

and

$$C' = \{(t'\restriction \mathbf{a}T, T) \mid T \in \mathbf{p}S \wedge a' \in \mathbf{a}T\}.$$

We derive

$C \cap C' \neq \emptyset$
$\Rightarrow$ {definition of t and t', $a = a'$ from the above derivation, calculus}
$(\exists T : T \in \mathbf{p}S \wedge a \in \mathbf{a}T : t\restriction \mathbf{a}T = t'\restriction \mathbf{a}T)$
$\Rightarrow$ {property of $\restriction$}
$t\restriction a = t'\restriction a$
$\Rightarrow$ {Property 2.3.12}
$(\forall T : T \in \mathbf{p}S \wedge a \in \mathbf{a}T : t\restriction \mathbf{a}T = t'\restriction \mathbf{a}T)$
$\Rightarrow$ {definition of t and t'}
$C = C'$.

Hence, because $act.p = Act.\pi$,

$$(\forall a, a', C, C' : \{(a, C), (a', C')\} \subseteq Act.\pi \wedge C \cap C' \neq \emptyset : (a, C) = (a', C')).$$

□

On account of Definition 2.3.1 we have then the result expressed by Corollary 2.3.14. It shows the relationship between po-traces and prefixes of computations of a system.

Corollary 2.3.14

For system S

$$\{conf.p \cup act.p \mid p \in ppot.S\} \subseteq ppoc.S.$$

□

The following property facilitates the proof of Lemma 2.3.16.

Property 2.3.15

For system S, prefix π of a computation of S, and trace ta of $\mathbf{t}S$

$$\{(ta\restriction \mathbf{a}T, T) \mid T \in \mathbf{p}S\} \subseteq \pi \wedge \{(t\restriction \mathbf{a}T, T) \mid T \in \mathbf{p}S\} \subseteq \pi$$
$$\Rightarrow$$
$$(a, \{(t\restriction \mathbf{a}T, T) \mid T \in \mathbf{p}S \wedge a \in \mathbf{a}T\}) \in \pi.$$

Proof

For every $U \in \mathbf{p}S$, such that $a \in \mathbf{a}U$, we have

$$\begin{array}{ll} & ((t\restriction \mathbf{a}U)a, U) \in \pi \\ \Rightarrow & \{\text{Definition 2.3.1}\} \\ & (\exists C : (a, C) \in Act.\pi : (t\restriction \mathbf{a}U, U) \in C) \\ \Rightarrow & \{\text{Definition 2.3.1}\} \\ & (\exists C, u : (a, C) \in Act.\pi \wedge ua \in \mathbf{t}S \wedge (t\restriction \mathbf{a}U, U) \in C \\ & \quad : C \subseteq \{(u\restriction \mathbf{a}T, T) \mid T \in \mathbf{p}S\} \subseteq \pi \wedge \{(ua\restriction \mathbf{a}T, T) \mid T \in \mathbf{p}S\} \subseteq \pi). \end{array}$$

Furthermore, let $C = \{(t\restriction \mathbf{a}T, T) \mid T \in \mathbf{p}S \wedge a \in \mathbf{a}T\}$. We derive

$$\begin{array}{ll} & (a', C') \in Act.\pi \wedge C \cap C' \neq \emptyset \\ \Rightarrow & \{\text{Definition 2.3.1, definition of } C, \text{ calculus}\} \\ & (\exists u, U : ua' \in \mathbf{t}S \wedge U \in \mathbf{p}S \wedge \{a, a'\} \subseteq \mathbf{a}U \wedge (t\restriction \mathbf{a}U, U) \in C' \\ & \quad : C' \subseteq \{(u\restriction \mathbf{a}T, T) \mid T \in \mathbf{p}S\} \subseteq \pi \wedge \{(ua'\restriction \mathbf{a}T, T) \mid T \in \mathbf{p}S\} \subseteq \pi). \\ \Rightarrow & \{\text{Definition 2.1.15, calculus}\} \\ & (\exists u, U : ua' \in \mathbf{tr}.\pi \wedge U \in \mathbf{p}S \wedge \{a, a'\} \subseteq \mathbf{a}U \wedge u\restriction \mathbf{a}U = t\restriction \mathbf{a}U \\ & \quad : C' = \{(u\restriction \mathbf{a}T, T) \mid T \in \mathbf{p}S \wedge a' \in \mathbf{a}T\}) \\ \Rightarrow & \{\text{property of } \restriction, \mathbf{tr}.\pi \in ppot.S \text{ and } ta \in \mathbf{tr}.\pi, \text{ Definition 2.3.2, calculus}\} \\ & a = a' \wedge (\exists u : ua \in \mathbf{tr}.\pi : u\restriction a = t\restriction a \wedge C' = \{(u\restriction \mathbf{a}T, T) \mid T \in \mathbf{p}S \wedge a' \in \mathbf{a}T\}) \\ \Rightarrow & \{\mathbf{tr}.\pi \in ppot.S \text{ and } ta \in \mathbf{tr}.\pi, \text{ Property 2.3.12}\} \\ & a = a' \wedge (\exists u : ua \in \mathbf{tr}.\pi \wedge C' = \{(u\restriction \mathbf{a}T, T) \mid T \in \mathbf{p}S \wedge a' \in \mathbf{a}T\} \\ & \quad : (\forall T : T \in \mathbf{p}S \wedge a \in \mathbf{a}T : u\restriction \mathbf{a}T = t\restriction \mathbf{a}T)) \\ \Rightarrow & \{\text{definition of } C, \text{ calculus}\} \\ & (a, C) = (a', C'). \end{array}$$

Hence, according to Definition 2.3.1,

$$(a, \{(t \restriction \mathbf{a}T, T) \mid T \in \mathbf{p}S \wedge a \in \mathbf{a}T\}) \in \pi.$$

□

We already proved for system S that (Corollary 2.3.7)

$$\{\mathbf{tr}.\pi \mid \pi \in ppoc.S\} \subseteq ppot.S.$$

As we mentioned, the main goal of this section is to prove that $(ppoc.S, \subseteq)$ is isomorphic to $(ppot.S, \subseteq)$ and function $\mathbf{tr}$ is an isomorphism, which constitutes Theorem 2.3.20. To achieve this goal, two intermediate results are needed that are proved by Lemmata 2.3.16 and 2.3.18.

In the following lemma we prove that every prefix π of a computation of system S is equal to $conf.(\mathbf{tr}.\pi) \cup act.(\mathbf{tr}.\pi)$.

Lemma 2.3.16

For system S and prefix π of a computation of S

$$\pi = conf.(\mathbf{tr}.\pi) \cup act.(\mathbf{tr}.\pi).$$

Proof

First, $\pi \supseteq conf.(\mathbf{tr}.\pi) \cup act.(\mathbf{tr}.\pi)$ is proved.
Let $T \in \mathbf{p}S$. We derive

$$\begin{array}{ll} & (t, T) \in conf.(\mathbf{tr}.\pi) \\ \Leftrightarrow & \{\text{definition of } conf\} \\ & (\exists u : u \in \mathbf{tr}.\pi : u \restriction \mathbf{a}T = t) \\ \Rightarrow & \{\text{Definition 2.1.15}\} \\ & (\exists u : \{(u \restriction \mathbf{a}U, U) \mid U \in \mathbf{p}S\} \subseteq \pi : u \restriction \mathbf{a}T = t) \\ \Rightarrow & \{T \in \mathbf{p}S, \text{ set and predicate calculus}\} \\ & (t, T) \in \pi. \end{array}$$

Hence, $conf.(\mathbf{tr}.\pi) \subseteq \pi$.
We derive

$$\begin{array}{ll} & (a, C) \in act.(\mathbf{tr}.\pi) \\ \Leftrightarrow & \{\text{definition of } act\} \end{array}$$

$$
\begin{array}{ll}
 & (\exists u : ua \in \mathbf{tr}.\pi : C = \{(u\lceil \mathbf{a}U, U) \mid U \in \mathbf{p}S \wedge a \in \mathbf{a}U\}) \\
\Rightarrow & \{\, \mathbf{tr}.\pi \text{ is prefix-closed, Definition 2.1.15} \,\} \\
 & (\exists u : ua \in \mathbf{t}S \wedge \{(ua\lceil \mathbf{a}U, U) \mid U \in \mathbf{p}S\} \subseteq \pi \\
 & \quad : C = \{(u\lceil \mathbf{a}U, U) \mid U \in \mathbf{p}S \wedge a \in \mathbf{a}U\} \wedge \{(u\lceil \mathbf{a}U, U) \mid U \in \mathbf{p}S\} \subseteq \pi) \\
\Rightarrow & \{\, \text{Property 2.3.15} \,\} \\
 & (a, C) \in Act.\pi \\
\Rightarrow & \{\, \text{definition of } Act.\pi \,\} \\
 & (a, C) \in \pi.
\end{array}
$$

Hence, $act.(\mathbf{tr}.\pi) \subseteq \pi$.
The two above results prove $\pi \supseteq conf.(\mathbf{tr}.\pi) \cup act.(\mathbf{tr}.\pi)$.

Next, $\pi \subseteq conf.(\mathbf{tr}.\pi) \cup act.(\mathbf{tr}.\pi)$ is proved.
From Definition 2.3.1 we have $\{(\varepsilon, T) \mid T \in \mathbf{p}S\} \subseteq \pi$, which implies $\varepsilon \in \mathbf{tr}.\pi$, and hence — from the definition of $conf$ — $\{(\varepsilon, T) \mid T \in \mathbf{p}S\} \subseteq conf.(\mathbf{tr}.\pi)$.
Furthermore, for $T \in \mathbf{p}S$

$$
\begin{array}{ll}
 & (ta, T) \in Conf.\pi \\
\Rightarrow & \{\, \text{Definition 2.3.1} \,\} \\
 & (\exists C : (a, C) \in Act.\pi : (t, T) \in C) \\
\Rightarrow & \{\, \text{Definition 2.3.1} \,\} \\
 & (\exists C, u : (t, T) \in C \wedge ua \in \mathbf{t}S : C \subseteq \{(u\lceil \mathbf{a}U, U) \mid U \in \mathbf{p}S\} \subseteq \pi \\
 & \qquad\qquad \wedge \{(ua\lceil \mathbf{a}U, U) \mid U \in \mathbf{p}S\} \subseteq \pi) \\
\Rightarrow & \{\, \text{set and predicate calculus, Definition 2.1.15} \,\} \\
 & (\exists u : t = u\lceil \mathbf{a}T : ua \in \mathbf{tr}.\pi) \\
\Rightarrow & \{\, \text{definition of } conf \,\} \\
 & (ta, T) \in conf.(\mathbf{tr}.\pi).
\end{array}
$$

Hence, $Conf.\pi \subseteq conf.(\mathbf{tr}.\pi)$.
We derive

$$
\begin{array}{ll}
 & (a, C) \in Act.\pi \\
\Rightarrow & \{\, \text{Definition 2.3.1} \,\} \\
 & (\exists u : ua \in \mathbf{t}S : C \subseteq \{(u\lceil \mathbf{a}U, U) \mid U \in \mathbf{p}S\} \subseteq \pi \wedge \{(ua\lceil \mathbf{a}U, U) \mid U \in \mathbf{p}S\} \subseteq \pi) \\
\Rightarrow & \{\, \text{Definition 2.1.15, definition of } Act.S \,\} \\
 & (\exists u : ua \in \mathbf{tr}.\pi : C = \{(u\lceil \mathbf{a}U, U) \mid U \in \mathbf{p}S \wedge a \in \mathbf{a}U\})
\end{array}
$$

$\Leftrightarrow$ { definition of act }
$(a, C) \in act.(\mathbf{tr}.\pi)$.

Hence, $Act.\pi \subseteq act.(\mathbf{tr}.\pi)$.
The two above results prove $\pi \subseteq conf.(\mathbf{tr}.\pi) \cup act.(\mathbf{tr}.\pi)$.

□

In the following lemma we prove that for every po-trace p of system S, the trace set associated with $conf.p \cup act.p$ equals p. This lemma facilitates the proof of Lemma 2.3.18.

Lemma 2.3.17

For system S and po-trace p of S

$$p = \mathbf{tr}.(conf.p \cup act.p).$$

Proof

Let $t \in \mathbf{t}S$.
In order to prove $\subseteq$, we derive

$t \in p$
$\Rightarrow$ { definition of $conf$ }
$\{(t\lceil \mathbf{a}T, T) \mid T \in \mathbf{p}S\} \subseteq conf.p$
$\Leftrightarrow$ { Definition 2.1.15 }
$t \in \mathbf{tr}.(conf.p \cup act.p)$.

Hence, $p \subseteq \mathbf{tr}.(conf.p \cup act.p)$.
In order to prove $\supseteq$, we derive

$t \in \mathbf{tr}.(conf.p \cup act.p)$
$\Leftrightarrow$ { Definition 2.1.15 }
$\{(t\lceil \mathbf{a}T, T) \mid T \in \mathbf{p}S\} \subseteq conf.p$
$\Rightarrow$ { definition of $conf$ }
$(\exists u : u \in p : (\forall T : T \in \mathbf{p}S : u\lceil \mathbf{a}T = t\lceil \mathbf{a}T))$
$\Rightarrow$ { $p \in ppot.S$, Definition 2.3.2 }
$t \in p$.

Hence, $\mathbf{tr}.(conf.p \cup act.p) \subseteq p$.

□

The following lemma shows that every element of $ppot.S$ is associated with a prefix of a computation of system S.

Lemma 2.3.18

For system S

$$ppot.S \subseteq \{\mathbf{tr}.\pi \mid \pi \in ppoc.S\}.$$

Proof

We derive

$$\begin{array}{ll} & p \in ppot.S \\ \Rightarrow & \{\text{Corollary 2.3.14}\} \\ & conf.p \cup act.p \in ppoc.S \\ \Rightarrow & \{\text{definition of } \mathbf{tr}\} \\ & \mathbf{tr}.(conf.p \cup act.p) \in \{\mathbf{tr}.\pi \mid \pi \in ppoc.S\} \\ \Leftrightarrow & \{\text{Lemma 2.3.17}\} \\ & p \in \{\mathbf{tr}.\pi \mid \pi \in ppoc.S\}. \end{array}$$

Hence, $ppot.S \subseteq \{\mathbf{tr}.\pi \mid \pi \in ppoc.S\}$.

□

As a consequence of Corollary 2.3.7 and Lemma 2.3.18, we have the equality of the set of po-traces of system S and the image of $ppoc.S$ under $\mathbf{tr}$. This is expressed by the following corollary.

Corollary 2.3.19

For system S

$$ppot.S = \{\mathbf{tr}.\pi \mid \pi \in ppoc.S\}.$$

□

Finally, we present the main theorem of this section. On account of this theorem, the alternative way of determining the acceptances is given further on.

Theorem 2.3.20

Let S be a system. Then $(ppoc.S, \subseteq)$ is isomorphic to $(ppot.S, \subseteq)$, and function $\mathbf{tr}$ is an isomorphism.

Proof

Our proof obligation consists in showing that $\mathbf{tr}$ is a bijection such that for elements π and π' of $ppoc.S$

$$\pi \subseteq \pi' \;\Leftrightarrow\; \mathbf{tr}.\pi \subseteq \mathbf{tr}.\pi'.$$

From Corollary 2.3.19 we already have that $\mathbf{tr}$ is a surjection, that is,

$$\mathbf{tr}.(ppoc.S) = ppot.S.$$

We then prove that $\mathbf{tr}$ is an injection too. In other words, for elements π and π' of $ppoc.S$

$$\mathbf{tr}.\pi = \mathbf{tr}.\pi' \;\Rightarrow\; \pi = \pi'.$$

We derive

$$\begin{array}{ll} & \mathbf{tr}.\pi = \mathbf{tr}.\pi' \\ \Rightarrow & \{\text{definition of } conf \text{ and } act\} \\ & conf.(\mathbf{tr}.\pi) \cup act.(\mathbf{tr}.\pi) = conf.(\mathbf{tr}.\pi') \cup act.(\mathbf{tr}.\pi') \\ \Leftrightarrow & \{\text{Lemma 2.3.16}\} \\ & \pi = \pi'. \end{array}$$

Hence, $\mathbf{tr}$ is a bijection.
Consequently, for elements π and π' of $ppoc.S$ we prove

$$\pi \subseteq \pi' \;\Leftrightarrow\; \mathbf{tr}.\pi \subseteq \mathbf{tr}.\pi'.$$

Assume $\pi \subseteq \pi'$ and $t \in \mathbf{t}S$. We derive

$$\begin{array}{ll} & t \in \mathbf{tr}.\pi \\ \Leftrightarrow & \{\text{Definition 2.1.15}\} \\ & \{(t \restriction \mathbf{a}T, T) \mid T \in \mathbf{p}S\} \subseteq \pi \end{array}$$

$$
\begin{array}{ll}
\Rightarrow & \{\pi \subseteq \pi'\} \\
& \{(t\lceil \mathbf{a}T, T) \mid T \in \mathbf{p}S\} \subseteq \pi' \\
\Leftrightarrow & \{\text{Definition 2.1.15}\} \\
& t \in \mathbf{tr}.\pi'.
\end{array}
$$

Hence, $\pi \subseteq \pi' \Rightarrow \mathbf{tr}.\pi \subseteq \mathbf{tr}.\pi'$.

In order to prove the implication the other way around, assume $\mathbf{tr}.\pi \subseteq \mathbf{tr}.\pi'$.
For configurations we derive

$$
\begin{array}{ll}
& \mathit{conf}.(\mathbf{tr}.\pi) \\
= & \{\text{definition of } \mathit{conf}\} \\
& \{(t\lceil \mathbf{a}T, T) \mid T \in \mathbf{p}S \wedge t \in \mathbf{tr}.\pi\} \\
\subseteq & \{\mathbf{tr}.\pi \subseteq \mathbf{tr}.\pi'\} \\
& \{(t\lceil \mathbf{a}T, T) \mid T \in \mathbf{p}S \wedge t \in \mathbf{tr}.\pi'\} \\
= & \{\text{definition of } \mathit{conf}\} \\
& \mathit{conf}.(\mathbf{tr}.\pi').
\end{array}
$$

For action occurrences we derive

$$
\begin{array}{ll}
& \mathit{act}.(\mathbf{tr}.\pi) \\
= & \{\text{definition of } \mathit{act}\} \\
& \{(a, \{(t\lceil \mathbf{a}T, T) \mid T \in \mathbf{p}S \wedge a \in \mathbf{a}T\}) \mid a \in \mathbf{a}S \wedge ta \in \mathbf{tr}.\pi\} \\
\subseteq & \{\mathbf{tr}.\pi \subseteq \mathbf{tr}.\pi'\} \\
& \{(a, \{(t\lceil \mathbf{a}T, T) \mid T \in \mathbf{p}S \wedge a \in \mathbf{a}T\}) \mid a \in \mathbf{a}S \wedge ta \in \mathbf{tr}.\pi'\} \\
= & \{\text{definition of } \mathit{act}\} \\
& \mathit{act}.(\mathbf{tr}.\pi').
\end{array}
$$

Hence, $\mathit{conf}.(\mathbf{tr}.\pi) \cup \mathit{act}.(\mathbf{tr}.\pi) \subseteq \mathit{conf}.(\mathbf{tr}.\pi') \cup \mathit{act}.(\mathbf{tr}.\pi')$.
Together with Lemma 2.3.16 this gives $\pi \subseteq \pi'$.

□

A standard result for isomorphic partially ordered sets is that maximal elements of one of them coincide with the isomorphic images of maximal elements of the second one. In our case, this means that

π is a maximal element of $\mathit{ppoc}.S \Leftrightarrow \mathbf{tr}.\pi$ is a maximal element of $\mathit{ppot}.S$,

which is expressed by Corollary 2.3.21.

Corollary 2.3.21

For system S

$$pot.S = \{\mathbf{tr}.\pi \mid \pi \in poc.S\}.$$

□

A consequence of the fact that the set of maximal po-traces of system S equals the image of $poc.S$ under $\mathbf{tr}$ is that acceptances of S can be determined using $pot.S \restriction \mathbf{e}S$. Hence, we do not have to use the operational model for this purpose. This result is expressed by the following corollary.

Corollary 2.3.22

For system S, trace t of $\mathbf{t}(\mathbf{pr}S)$, and subset L of $\mathbf{e}S^*$

$$(t, L) \in \mathbf{ac}S \;\Leftrightarrow\; (\forall p : p \in pot.S \wedge t \in p \restriction \mathbf{e}S : \{v \mid tv \in p \restriction \mathbf{e}S\} \cap L \neq \emptyset).$$

An equivalent formulation is

$$(t, L) \in \mathbf{ac}S \;\Leftrightarrow\; (\forall X : X \in pot.S \restriction \mathbf{e}S \wedge t \in X : \{v \mid tv \in X\} \cap L \neq \emptyset).$$

□

The results presented in the sequel relate the elements of $ppot.S$ to conservative processes. This relationship is used in the proof of Lemma 2.5.6 in Section 2.5.
On account of Definitions 1.3.5 and 2.3.2 we have the following property.

Property 2.3.23

For system S, po-trace p of S, and process T of $\mathbf{p}S$

$\langle \mathbf{a}T, p \restriction \mathbf{a}T \rangle$ is a conservative process.

□

Property 2.3.24

For system S and po-trace p of S

$$\langle \mathbf{a}S, p \rangle = (\mathbf{W}T : T \in \mathbf{p}S : \langle \mathbf{a}T, p \restriction \mathbf{a}T \rangle).$$

Proof

The alphabets of both processes are equal, viz., $\mathbf{a}S$. The remaining proof obligation consists in showing that their trace sets are equal too.
In order to prove $\subseteq$, we derive

$$
\begin{array}{ll}
& t \in p \\
\Rightarrow & \{\, p \subseteq (\mathbf{a}S)^*, \text{ definition of } \restriction \,\} \\
& t \in (\mathbf{a}S)^* \wedge (\forall T : T \in \mathbf{p}S : t\restriction \mathbf{a}T \in p\restriction \mathbf{a}T) \\
\Leftrightarrow & \{\, \text{definition of } \mathbf{W} \,\} \\
& t \in \mathbf{t}(\mathbf{W}T : T \in \mathbf{p}S : \langle \mathbf{a}S, p\restriction \mathbf{a}T\rangle).
\end{array}
$$

In order to prove $\supseteq$, we derive

$$
\begin{array}{ll}
& t \in \mathbf{t}(\mathbf{W}T : T \in \mathbf{p}S : \langle \mathbf{a}S, p\restriction \mathbf{a}T\rangle) \\
\Leftrightarrow & \{\, \text{definition of } \mathbf{W} \,\} \\
& t \in (\mathbf{a}S)^* \wedge (\forall T : T \in \mathbf{p}S : t\restriction \mathbf{a}T \in p\restriction \mathbf{a}T) \\
\Leftrightarrow & \{\, p \subseteq \mathbf{t}S, \restriction \text{ is monotonic, definition of } \mathbf{t}S, \text{ Lemma 1.1.20} \,\} \\
& t \in \mathbf{t}S \wedge (\forall T : T \in \mathbf{p}S : t\restriction \mathbf{a}T \in p\restriction \mathbf{a}T) \\
\Rightarrow & \{\, \text{definition of } \restriction \,\} \\
& t \in \mathbf{t}S \wedge (\forall T : T \in \mathbf{p}S : (\exists u : u \in p : t\restriction \mathbf{a}T = u\restriction \mathbf{a}T)) \\
\Rightarrow & \{\, \text{definition of } \mathit{conf} \ (2.3.8) \,\} \\
& t \in \mathbf{t}S \wedge (\forall T : T \in \mathbf{p}S : (t\restriction \mathbf{a}T, T) \in \mathit{conf}.p) \\
\Rightarrow & \{\, p \in \mathit{ppot}.S, \text{ hence on account of Corollary 2.3.14 and Definition 2.3.1,} \\
& \quad \mathit{conf}.p \cup \mathit{act}.p \subseteq \mathit{Conf}.S \cup \mathit{Act}.S, \text{ Definition 2.1.15} \,\} \\
& t \in \mathbf{tr}.(\mathit{conf}.p \cup \mathit{act}.p) \\
\Leftrightarrow & \{\, p \in \mathit{ppot}.S, \text{ Lemma 2.3.17} \,\} \\
& t \in p.
\end{array}
$$

□

By Property 1.3.7 and the two above properties we have the following result.

Corollary 2.3.25

Let S be a system and p a po-trace of S. Then

$\langle \mathbf{a}S, p\rangle$ is a conservative process.

□

2.4 Behavioural relations

Different theories intended to describe and to reason about concurrent and nondeterministic systems are known in the literature. They use various notions of equivalence of systems, e.g., string equivalence, observational equivalence, failure equivalence, testing equivalence. A thorough review of these notions can be found in [De0]. Almost all of them are based on the idea that two systems are equivalent whenever no external observer can distinguish between them. This idea is inspired by the fact that for a given system usually not its internal structure is of interest but its behaviour with respect to the environment in which it is placed (that is, a system with which it is supposed to co-operate).

We decided to compare systems by exposing them to tests and confronting their responses (see, e.g., [Hen], also for further references), because the identifications and distinctions that are made in this way seem to be very reasonable. The formalization of this approach results in behavioural relations embedded in the operational model of Section 2.1. By means of these relations, parallel systems can be distinguished from nondeterministic but sequential systems. We borrowed the general idea of testing from [Hen], however, the relations defined in this section are different from the testing relations described there.

When comparing systems by means of testing, different aspects of their behaviour can be considered. Two of them are of interest to us. For every pair of systems exposed to a test (also a system) their *potential* and *guaranteed* responses are compared. The potential response of system S to test R is defined to be $\mathbf{pr}(R \parallel S)$ — the external process of the parallel composition of R and S. By the guaranteed response of system S to test R we understand the set of acceptances of the parallel composition of R and S. Based on these two aspects three behavioural (testing) relations between systems are defined.

In the sequel, SY represents the set of systems with alphabets contained in Ω, and PR represents the set of processes with alphabets contained in Ω.

The first testing relation considers potential responses of systems to tests.

Definition 2.4.1

Let S_0 and S_1 be systems. Relation **psat** compares systems S_0 and S_1 with respect to their potential responses. It is defined by

$$S_0 \ \mathbf{psat}\ S_1 \ \Leftrightarrow \ (\forall R : R \in SY : \mathbf{pr}(R \parallel S_0) \subseteq \mathbf{pr}(R \parallel S_1)).$$

□

If S_0 **psat** S_1 holds, we say that S_0 potentially satisfies (implements) S_1. That is, S_0 cannot produce other sequences of actions than the ones S_1 can produce too, if they are placed in the same context. In other words, S_0 cannot do anything "wrong" with respect to what S_1 can do. In our model, relation **psat** takes care of the verification of the, so-called, safety properties.

The following lemma shows that there is a simple way to check whether S_0 **psat** S_1 holds. Namely, we can get rid of the universal quantification, which considerably simplifies calculations.

Lemma 2.4.2

For systems S_0 and S_1

$$S_0 \textbf{ psat } S_1 \Leftrightarrow \mathbf{pr}S_0 \subseteq \mathbf{pr}S_1.$$

Proof

We derive

$$\begin{array}{ll} & S_0 \textbf{ psat } S_1 \\ \Leftrightarrow & \{\text{Definition 2.4.1}\} \\ & (\forall R : R \in SY : \mathbf{pr}(R \parallel S_0) \subseteq \mathbf{pr}(R \parallel S_1)) \\ \Leftrightarrow & \{\text{Lemma 1.2.4}\} \\ & (\forall R : R \in SY : (\mathbf{pr}R)\ \mathbf{w}\ (\mathbf{pr}S_0) \subseteq (\mathbf{pr}R)\ \mathbf{w}\ (\mathbf{pr}S_1)) \\ \Leftrightarrow & \{\{\mathbf{pr}R \mid R \in SY\} = PR\} \\ & (\forall T : T \in PR : T\ \mathbf{w}\ (\mathbf{pr}S_0) \subseteq T\ \mathbf{w}\ (\mathbf{pr}S_1)) \\ \Leftrightarrow & \{\text{Lemma 1.1.22}\} \\ & \mathbf{pr}S_0 \subseteq \mathbf{pr}S_1. \end{array}$$

□

The second testing relation considers guaranteed responses of systems to tests.

Definition 2.4.3

Let S_0 and S_1 be systems. Relation **gsat** compares systems S_0 and S_1 from the point of view of guaranteed responses. It is defined by

$$S_0 \textbf{ gsat } S_1 \Leftrightarrow (\forall R : R \in SY : \mathbf{ac}(R \parallel S_0) \supseteq \mathbf{ac}(R \parallel S_1)).$$

□

If S_0 **gsat** S_1, we say that S_0 is guaranteed to satisfy (implement) S_1. That is, everything S_1 can guarantee to execute in a context, will certainly be performed by S_0 in the same context. In our model, relation **gsat** takes care of the verification of the, so-called, liveness properties.

The third testing relation combines the two previous ones.

Definition 2.4.4

Let S_0 and S_1 be systems. Relation **sat** expressing that system S_0 is an implementation of (satisfies) S_1, S_0 **sat** S_1, is defined by

$$S_0 \textbf{ sat } S_1 \Leftrightarrow S_0 \textbf{ psat } S_1 \wedge S_0 \textbf{ gsat } S_1.$$

□

For the systems of Example 2.2.3, $\neg(S_0 \textbf{ sat } S_1)$ holds. To establish this result consider test $R = \langle\{a, b\}, \{\mathbf{pr}(b^*; a)\}\rangle$. Then

$$(\varepsilon, \{b^i a \mid 0 \leq i\}) \in \mathbf{ac}(R \parallel S_1) \qquad \text{and} \qquad (\varepsilon, \{b^i a \mid 0 \leq i\}) \notin \mathbf{ac}(R \parallel S_0).$$

We also have $\neg(S_1 \textbf{ sat } S_0)$, which can be established by considering as a test system $R = \langle\{a, b\}, \{\mathbf{pr}(b; a)\}\rangle$. Then

$$(b, \{a\}) \in \mathbf{ac}(R \parallel S_0) \qquad \text{and} \qquad (b, \{a\}) \notin \mathbf{ac}(R \parallel S_1).$$

Furthermore, two systems are declared to be equivalent, if each of them is an implementation of the other one.

Definition 2.4.5

For systems S_0 and S_1

$$S_0 \textbf{ equ } S_1 \Leftrightarrow S_0 \textbf{ sat } S_1 \wedge S_1 \textbf{ sat } S_0.$$

□

Testing relations solve the problem related to the phenomenon of disabling that is mentioned in Section 2.2. That is, thanks to the fact that systems are exposed to tests, acceptances supply information about the system's external behaviour that is sufficient to distinguish between systems with and without disabling.

Relation **gsat** enables us to distinguish between systems S_0 and S_1 of Example 2.2.7, which have equal external processes. In this case, it is easy to verify that for the test $R = \langle\{a,b\},\{\langle\{a,b\},\{\varepsilon,a\}\rangle\}\rangle$ we have

$$(\varepsilon,\{a\}) \in \mathbf{ac}(R \parallel S_1) \wedge (\varepsilon,\{a\}) \notin \mathbf{ac}(R \parallel S_0).$$

This implies $\neg(S_0\ \mathbf{gsat}\ S_1)$. Hence, S_0 does not implement S_1, i.e., $\neg(S_0\ \mathbf{sat}\ S_1)$. Consequently, S_0 and S_1 are not equivalent, i.e., $\neg(S_0\ \mathbf{equ}\ S_1)$.

In the sequel, some more examples are presented. All of them concern pairs of systems with equal external processes, which implies that they are potential implementations of each other. In these examples we concentrate on showing why they are not equivalent to each other.

Example 2.4.6

Consider systems S_0 and S_1 defined by

$$S_0 = \langle\{a,b,c\},\{\mathbf{pr}(a\,;\,(x \mid c)),\mathbf{pr}(b\,;\,(x \mid c))\}\rangle$$

and

$$S_1 = \langle\{a,b,c\},\{\mathbf{pr}((a\,;\,b \mid b\,;\,a)\,;\,c)\}\rangle.$$

Because for $R = \langle\{a,b,c\},\{\mathbf{pr}(a\,;\,b\,;\,c)\}\rangle$

$$(ab,\{c\}) \in \mathbf{ac}(R \parallel S_1) \qquad \text{and} \qquad (ab,\{c\}) \notin \mathbf{ac}(R \parallel S_0),$$

we have $\neg(S_0\ \mathbf{gsat}\ S_1)$. Hence, S_0 does not implement S_1.
We also have $\neg(S_1\ \mathbf{gsat}\ S_0)$, because for $R = \langle\{a,b,c\},\{\mathbf{pr}a,\mathbf{pr}b,\mathbf{pr}(c^0)\}\rangle$

$$(\varepsilon,\{a\}) \in \mathbf{ac}(R \parallel S_0) \wedge (\varepsilon,\{a\}) \notin \mathbf{ac}(R \parallel S_1).$$

Consequently, S_1 is not a correct implementation of S_0.

□

Example 2.4.7

Let S_0 and S_1 be systems defined by

$$S_0 = \langle\{a\},\{\mathbf{pr}(a^*)\}\rangle$$

and

$$S_1 = \langle\{a\},\{\mathbf{pr}(x_i\,;\,a^i) \mid 1 \leq i\}\rangle.$$

We have $\neg(S_1 \textbf{ gsat } S_0)$, because for $R = \langle\{a\}, \{\mathbf{pr}(a\,;\, a)\}\rangle$

$$(a, \{a\}) \in \mathbf{ac}(R \parallel S_0) \wedge (a, \{a\}) \notin \mathbf{ac}(R \parallel S_1).$$

Hence, S_1 is not a correct implementation of S_0.

□

Example 2.4.8

Consider systems S_0 and S_1 that are defined by

$$S_0 = \langle\{a, b\}, \{\mathbf{pr}(a\,;\, x), \mathbf{pr}((y \mid x)\,;\, b)\}\rangle$$

and

$$S_1 = \langle\{a, b\}, \{\mathbf{pr}a, \mathbf{pr}b\}\rangle.$$

We have $\neg(S_0 \textbf{ gsat } S_1)$, because for $R = \langle\{a, b\}, \{\mathbf{pr}a, \mathbf{pr}b\}\rangle$

$$(\varepsilon, \{b\}) \in \mathbf{ac}(R \parallel S_1) \wedge (\varepsilon, \{b\}) \notin \mathbf{ac}(R \parallel S_0).$$

Consequently, S_0 does not implement S_1.

□

It is obvious from their definitions that **psat**, **gsat** and **sat** are pre-orders. This implies then that **equ** is an equivalence relation (**equ** is reflexive, symmetric and transitive).

A consequence of relation **sat** being a pre-order is that **equ** is a congruence with respect to **sat**. That is, for systems S_0, S_1, T_0, and T_1

$$S_0 \textbf{ equ } T_0 \wedge S_1 \textbf{ equ } T_1 \Rightarrow (S_0 \textbf{ sat } S_1 \Leftrightarrow T_0 \textbf{ sat } T_1).$$

So far we have provided means to compare systems with — for us — satisfactory results. However, to prove that two systems are equivalent requires a lot of effort, because of the universal quantifications over the large system domain.

The purpose of testing is to check how the external behaviour of a system is influenced by all possible environments. In fact, the internal actions of tests cannot influence the behaviour of the tested system. Moreover, tests with too large external processes do not provide more information about the tested system than other tests. More precisely, for system S it is sufficient to consider tests R such that $\mathbf{a}R = \mathbf{e}R = \mathbf{e}S$ and $\mathbf{pr}R \subseteq \mathbf{pr}S$. We claim that

$$\mathit{Test}.S = \{R \mid R \in SY \wedge \mathbf{a}R = \mathbf{e}R = \mathbf{e}S \wedge \mathbf{pr}R \subseteq \mathbf{pr}S\}$$

constitutes the domain of relevant tests for system S, if S stands for the right argument of **sat**. That is, for systems S_0 and S_1

$$S_0 \textbf{ sat } S_1 \Leftrightarrow \textbf{pr}S_0 \subseteq \textbf{pr}S_1 \wedge (\forall R : R \in \mathit{Test}.S_1 : \textbf{ac}(R \parallel S_0) \supseteq \textbf{ac}(R \parallel S_1)).$$

The proof of this statement can be found in Section A.5 of the Appendix.
In the case of systems S_0 and S_1 of Example 2.2.7 the relevant tests are

- $\langle \{a, b\}, \{\textbf{pr}a, \textbf{pr}(b^0)\} \rangle$,
- $\langle \{a, b\}, \{\textbf{pr}(a^0), \textbf{pr}b\} \rangle$,
- $\langle \{a, b\}, \{\textbf{pr}(a \mid b)\} \rangle$.

(Test $\langle \{a, b\}, \{\textbf{stop}.\{a, b\}\} \rangle$ gives no information about the tested system and other tests of $\mathit{Test}.S_0$ are of no relevance, because their behaviours correspond to the behaviours of the above tests.)
For all the above tests R we have $\textbf{ac}(R \parallel S_1) \supseteq \textbf{ac}(R \parallel S_0)$. Together with $\textbf{pr}S_1 \subseteq \textbf{pr}S_0$ this yields S_1 **sat** S_0.
In the following example we show the equivalence of two systems with equal external processes.

Example 2.4.9

Let $S_0 = \langle \{a, b\}, \{\textbf{pr}(a \mid x), \textbf{pr}(b \mid y)\} \rangle$ and

$$S_1 = \langle \{a, b\}, \{\textbf{pr}((a \mid x) ; (b \mid y) \mid (b \mid y) ; (a \mid x))\} \rangle.$$

The relevant tests are

- $\langle \{a, b\}, \{\textbf{pr}a, \textbf{pr}(b^0)\} \rangle$,
- $\langle \{a, b\}, \{\textbf{pr}(a^0), \textbf{pr}b\} \rangle$,
- $\langle \{a, b\}, \{\textbf{pr}(a ; b)\} \rangle$,
- $\langle \{a, b\}, \{\textbf{pr}(b ; a)\} \rangle$,
- $\langle \{a, b\}, \{\textbf{pr}a, \textbf{pr}b\} \rangle$.

It is then easy to verify that S_0 **gsat** S_1 and S_1 **gsat** S_0.

□

The operational view of system behaviour presented in this section results in the testing relations that allow us to compare systems. Systems with and without parallelism are distinguished in our approach. We use the testing relations to justify our choice of the abstract model that is discussed in Chapter 3.

2.5 Properties of behavioural relations

In this section some facts about testing relations are proved. They include monotonicity of parallel composition and projection with respect to **psat**, **gsat**, **sat**, and **equ**. This means that the verification method based on the **sat** relation is compositional with respect to parallel composition and projection. Moreover, as a consequence of the above monotonicity properties, **equ** is a congruence with respect to parallel composition of systems and to projection of systems on alphabets.

Lemma 2.5.1 shows that $\|$ is monotonic with respect to **psat**.

Lemma 2.5.1

For systems S_0, S_1, and T

$$S_0 \textbf{ psat } S_1 \Rightarrow (S_0 \| T) \textbf{ psat } (S_1 \| T).$$

Proof

We derive

$$\begin{array}{ll} & S_0 \textbf{ psat } S_1 \\ \Leftrightarrow & \{\text{Lemma 2.4.2}\} \\ & \textbf{pr}S_0 \subseteq \textbf{pr}S_1 \\ \Rightarrow & \{\text{weave is monotonic, Lemma 1.1.14}\} \\ & (\textbf{pr}S_0) \textbf{ w } (\textbf{pr}T) \subseteq (\textbf{pr}S_1) \textbf{ w } (\textbf{pr}T) \\ \Leftrightarrow & \{\text{Lemma 1.2.4}\} \\ & \textbf{pr}(S_0 \| T) \subseteq \textbf{pr}(S_1 \| T) \\ \Leftrightarrow & \{\text{Lemma 2.4.2}\} \\ & (S_0 \| T) \textbf{ psat } (S_1 \| T). \end{array}$$

□

Lemma 2.5.2 shows that $\|$ is monotonic with respect to **gsat**.

Lemma 2.5.2

For systems S_0, S_1, and T

$$S_0 \textbf{ gsat } S_1 \Rightarrow (S_0 \parallel T) \textbf{ gsat } (S_1 \parallel T).$$

Proof

We derive

$$\begin{array}{ll} & S_0 \textbf{ gsat } S_1 \\ \Leftrightarrow & \{\text{Definition 2.4.3}\} \\ & (\forall R : R \in SY : \textbf{ac}(R \parallel S_0) \supseteq \textbf{ac}(R \parallel S_1)) \\ \Rightarrow & \{\text{predicate calculus (substitute } R' \parallel T \text{ for } R)\} \\ & (\forall R' : R' \in SY : \textbf{ac}(R' \parallel T \parallel S_0) \supseteq \textbf{ac}(R' \parallel T \parallel S_1)) \\ \Leftrightarrow & \{\parallel \text{ is associative and commutative, Definition 2.4.3}\} \\ & (S_0 \parallel T) \textbf{ gsat } (S_1 \parallel T). \end{array}$$

□

As a consequence of the preceding lemmata we have monotonicity of $\parallel$ with respect to **sat** and **equ**.

Corollary 2.5.3

For systems S_0, S_1, and T

- $S_0 \textbf{ sat } S_1 \Rightarrow (S_0 \parallel T) \textbf{ sat } (S_1 \parallel T)$.
- $S_0 \textbf{ equ } S_1 \Rightarrow (S_0 \parallel T) \textbf{ equ } (S_1 \parallel T)$.

□

Theorem 2.5.4 expresses the fact that relation **equ** is a congruence with respect to $\parallel$.

Theorem 2.5.4

For systems S_0, S_1, T_0, and T_1

$$S_0 \textbf{ equ } T_0 \wedge S_1 \textbf{ equ } T_1 \Rightarrow (S_0 \parallel S_1) \textbf{ equ } (T_0 \parallel T_1).$$

Proof

We derive

S_0 **equ** $T_0 \wedge S_1$ **equ** T_1
$\Rightarrow$ {corollary 2.5.3}
$(S_0 \parallel S_1)$ **equ** $(T_0 \parallel S_1) \wedge (S_1 \parallel T_0)$ **equ** $(T_1 \parallel T_0)$
$\Leftrightarrow$ {$\parallel$ is commutative}
$(S_0 \parallel S_1)$ **equ** $(T_0 \parallel S_1) \wedge (T_0 \parallel S_1)$ **equ** $(T_0 \parallel T_1)$
$\Rightarrow$ {**equ** is transitive}
$(S_0 \parallel S_1)$ **equ** $(T_0 \parallel T_1)$.

□

Lemma 2.5.5 shows that projection is monotonic with respect to **psat**.

Lemma 2.5.5

Let S_0 and S_1 be systems, and let A be an alphabet. Then

$$S_0 \textbf{ psat } S_1 \Rightarrow (S_0 \restriction A) \textbf{ psat } (S_1 \restriction A).$$

Proof

We derive

S_0 **psat** S_1
$\Leftrightarrow$ {Lemma 2.4.2}
$\mathbf{pr}S_0 \subseteq \mathbf{pr}S_1$
$\Rightarrow$ {projection is monotonic, Lemma 1.1.14}
$(\mathbf{pr}S_0) \restriction A \subseteq (\mathbf{pr}S_1) \restriction A$
$\Leftrightarrow$ {Property 1.2.5}
$\mathbf{pr}(S_0 \restriction A) \subseteq \mathbf{pr}(S_1 \restriction A)$
$\Leftrightarrow$ {Lemma 2.4.2}
$(S_0 \restriction A)$ **psat** $(S_1 \restriction A)$.

□

The following lemma facilitates the proof of Lemma 2.5.7.

Lemma 2.5.6

For system S, $p \in pot.S$, and alphabet A

$$u \in p \wedge u'v \in p \wedge u \restriction A = u' \restriction A \Rightarrow u(u'v \setminus \flat.u) \in p \wedge v \restriction A = (u'v \setminus \flat.u) \restriction A.$$

Proof

On account of Corollary 2.3.25 and Property 1.3.9, we have

$$u \in p \wedge u'v \in p \;\Rightarrow\; u(u'v \setminus \flat.u) \in p.$$

Assume $u\restriction A = u'\restriction A$. We derive

$$\begin{array}{ll} & (u'v \setminus \flat.u)\restriction A \\ = & \quad \{\text{Property 1.1.1}\} \\ & u'v\restriction A \setminus \flat.(u\restriction A) \\ = & \quad \{\text{property of projection}\} \\ & (u'\restriction A)(v\restriction A) \setminus \flat.(u\restriction A) \\ = & \quad \{u\restriction A = u'\restriction A\} \\ & (u\restriction A)(v\restriction A) \setminus \flat.(u\restriction A) \\ = & \quad \{\text{Property 1.1.1}\} \\ & v\restriction A \setminus \flat.\varepsilon \\ = & \quad \{\flat.\varepsilon = \emptyset \text{ and } s \setminus \emptyset = s, \text{ for trace } s\} \\ & v\restriction A. \end{array}$$

Hence,

$$u \in p \wedge u'v \in p \wedge u\restriction A = u'\restriction A \;\Rightarrow\; u(u'v \setminus \flat.u) \in p \wedge v\restriction A = (u'v \setminus \flat.u)\restriction A.$$

□

Lemma 2.5.7 shows that projection is monotonic with respect to **gsat**.

Lemma 2.5.7

Let S_0 and S_1 be systems, such that $\mathbf{pr}S_0 \subseteq \mathbf{pr}S_1$, and let A be an alphabet. Then

$$S_0 \;\mathbf{gsat}\; S_1 \;\Rightarrow\; (S_0\restriction A)\;\mathbf{gsat}\;(S_1\restriction A).$$

Proof

Assume

$$(\forall R : R \in SY : \mathbf{ac}(R \parallel S_0) \supseteq \mathbf{ac}(R \parallel S_1)).$$

Let R be a system. For the sake of brevity (see Definition 1.2.1) we assume that $\mathbf{a}R \cap \mathbf{a}S_0 = \mathbf{e}R \cap \mathbf{e}S_0 \cap A$ and $\mathbf{a}R \cap \mathbf{a}S_1 = \mathbf{e}R \cap \mathbf{e}S_1 \cap A$.
Let $t \in \mathbf{t}(\mathbf{pr}(R \parallel (S_1\restriction A)))$ and $L \subseteq (\mathbf{e}R \cup (\mathbf{e}S_1 \cap A))^*$.
Define

$$L' = (\cup p, u, v : p \in pot.(R \parallel S_1) \wedge uv \in p\restriction(\mathbf{e}R \cup \mathbf{e}S_1) \\ \wedge u\restriction(\mathbf{e}R \cup (\mathbf{e}S_1 \cap A)) = t \wedge v\restriction(\mathbf{e}R \cup (\mathbf{e}S_1 \cap A)) \in L \\ : v) \\ \cup L \setminus \{v \mid tv \notin \mathbf{t}(\mathbf{pr}(R \parallel (S_1\restriction A)))\}.$$

We derive

$$\begin{array}{ll} & (t, L) \in \mathbf{ac}(R \parallel (S_1\restriction A)) \\ \Leftrightarrow & \{\text{Corollary 2.3.22}\} \\ & (\forall p : p \in pot.(R \parallel (S_1\restriction A)) \wedge t \in p\restriction(\mathbf{e}R \cup (\mathbf{e}S_1 \cap A)) \\ & \quad : \{v \mid tv \in p\restriction(\mathbf{e}R \cup (\mathbf{e}S_1 \cap A))\}) \cap L \neq \emptyset) \\ \Leftrightarrow & \{(\mathbf{e}S \setminus A) \cap \mathbf{e}R = \emptyset \Rightarrow R \parallel (S\restriction A) = \langle \mathbf{e}R \cup (\mathbf{e}S \setminus A), \mathbf{p}R \cup \mathbf{p}S\rangle, \\ & \quad \text{Property 2.1.16, Corollary 2.3.21, predicate calculus}\} \\ & (\forall p : p \in pot.(R \parallel S_1) \wedge t \in p\restriction(\mathbf{e}R \cup (\mathbf{e}S_1 \cap A)) \\ & \quad : (\exists v : v \in L : tv \in p\restriction(\mathbf{e}R \cup (\mathbf{e}S_1 \cap A)))) \\ \Leftrightarrow & \{\text{property of projection, predicate calculus}\} \\ & (\forall p, u : p \in pot.(R \parallel S_1) \wedge u \in p \wedge u\restriction(\mathbf{e}R \cup (\mathbf{e}S_1 \cap A)) = t \\ & \quad : (\exists u', v : v\restriction(\mathbf{e}R \cup (\mathbf{e}S_1 \cap A)) \in L : u'v \in p \wedge u'\restriction(\mathbf{e}R \cup (\mathbf{e}S_1 \cap A)) = t)) \\ \Leftrightarrow & \{\Rightarrow\text{: Lemma 2.5.6 with } S \text{ substituted by } R \parallel S_1 \text{ and } A \text{ substituted by} \\ & \quad \mathbf{e}R \cup (\mathbf{e}S_1 \cap A), \text{ predicate calculus; } \Leftarrow\text{: predicate calculus}\} \\ & (\forall p, u : p \in pot.(R \parallel S_1) \wedge u \in p \wedge u\restriction(\mathbf{e}R \cup (\mathbf{e}S_1 \cap A)) = t \\ & \quad : (\exists v : v\restriction(\mathbf{e}R \cup (\mathbf{e}S_1 \cap A)) \in L : uv \in p)) \\ \Leftrightarrow & \{\text{definition of } L', \text{ set and predicate calculus}\} \\ & (\forall p, u : p \in pot.(R \parallel S_1) \wedge u \in p \wedge u\restriction(\mathbf{e}R \cup (\mathbf{e}S_1 \cap A)) = t \\ & \quad : (\exists v : v\restriction(\mathbf{e}R \cup \mathbf{e}S_1) \in L' : uv \in p)) \\ \Leftrightarrow & \{\text{property of projection}\} \\ & (\forall p, u : p \in pot.(R \parallel S_1) \wedge u \in p\restriction(\mathbf{e}R \cup \mathbf{e}S_1) \wedge u\restriction(\mathbf{e}R \cup (\mathbf{e}S_1 \cap A)) = t \\ & \quad : (\exists v : v \in L' : uv \in p\restriction(\mathbf{e}R \cup \mathbf{e}S_1))) \\ \Leftrightarrow & \{\text{predicate calculus, } u \in p\restriction(\mathbf{e}R \cup \mathbf{e}S_1) \Rightarrow u \in \mathbf{t}(\mathbf{pr}(R \parallel S_1)), \\ & \quad \text{Definition 2.2.4}\} \\ & (\forall u : u \in \mathbf{t}(\mathbf{pr}(R \parallel S_1)) \wedge u\restriction(\mathbf{e}R \cup (\mathbf{e}S_1 \cap A)) = t : (u, L') \in \mathbf{ac}(R \parallel S_1)) \\ \Rightarrow & \{\mathbf{ac}(R \parallel S_1) \subseteq \mathbf{ac}(R \parallel S_0), \mathbf{pr}S_0 \subseteq \mathbf{pr}S_1\} \\ & (\forall u : u \in \mathbf{t}(\mathbf{pr}(R \parallel S_0)) \wedge u\restriction(\mathbf{e}R \cup (\mathbf{e}S_0 \cap A)) = t : (u, L') \in \mathbf{ac}(R \parallel S_0)) \\ \Leftrightarrow & \{\text{repetition of the first seven steps of this derivation in the reverse order,} \\ & \quad L = L'\restriction(\mathbf{e}R \cup (\mathbf{e}S_0 \cap A))\} \\ & (t, L) \in \mathbf{ac}(R \parallel (S_0\restriction A)). \end{array}$$

Hence,

$$S_0 \textbf{ gsat } S_1 \Rightarrow (S_0 \restriction A) \textbf{ gsat } (S_1 \restriction A).$$

□

From Lemmata 2.5.5 and 2.5.7 immediately follows that projection is monotonic with respect to **sat** and **equ**. (In the case of the **equ** relation, this also means that **equ** is a congruence with respect to projection.)

Corollary 2.5.8

For systems S_0 and S_1, and for alphabet A

- $S_0 \textbf{ sat } S_1 \Rightarrow (S_0 \restriction A) \textbf{ sat } (S_1 \restriction A)$.
- $S_0 \textbf{ equ } S_1 \Rightarrow (S_0 \restriction A) \textbf{ equ } (S_1 \restriction A)$.

□

3 Abstract model of systems

In the previous chapter, testing relations are introduced to formalize an operational view on how to compare systems. It is easy to show that two systems are related (or are not related) via **psat** — on account of Lemma 2.4.2 it is sufficient to compare their external processes. To show that S_0 is not guaranteed to implement S_1 it suffices, according to Definition 2.4.3, to point out a test R for which $\mathbf{ac}(R \parallel S_1)$ is not a subset of $\mathbf{ac}(R \parallel S_0)$. However, to establish that two systems are related via **gsat** all possible tests must be considered. In this case, a nontrivial reasoning about the large system domain is involved.

This chapter provides an abstract model of systems that is compositional (with respect to projection and parallel composition), and in which the outcome of comparisons is the same as in the operational model, while the calculations are much simpler here. At the same time the domain of systems is restricted. Only systems with nondeterministic choices that do not depend on external actions are considered. To give an informal explanation of what is meant by this, we mention two simple examples of systems that do, and do not satisfy this requirement. For instance, system $\langle\{a,b\},\{\mathbf{pr}(a \mid b)\}\rangle$ is excluded, and system $\langle\{a,b\},\{\mathbf{pr}(x\,;\, a \mid y\,;\, b)\}\rangle$ — which has a nondeterministic choice between a and b — may be considered. Moreover, system $\langle\{a,b,c\},\{\mathbf{pr}(x\,;\, a \mid y\,;\, b), \mathbf{pr}(c\,;\, y)\}\rangle$ is excluded from our considerations, whereas system $\langle\{a,b,c\},\{\mathbf{pr}(x\,;\, a \mid y\,;\, b), \mathbf{pr}c\}\rangle$ is not. The reason for the exclusion of the first system is an important change in its behaviour when it is exposed to the test $\langle\{c\},\{\mathbf{pr}(c^0)\}\rangle$. Whereas in the same context the nondeterministic choice of the second system stays intact, in the case of the first system there is no choice anymore. The nondeterministic choice of the second system depends on the occurrence of the external action c, which can be influenced by a test. We want to exclude this kind of influence by the testing environment. Although these exclusions genuinely restrict the application domain, it is still sufficient to cover — among other things — the semantics of Tangram programs, as is shown in Chapter 4.

In Section 3.1, a few results concerning computations of parallel composition of systems are presented. Section 3.2 contains the formal definition of the abstract model for the restricted system domain. This model is compositional with respect to parallel composition and projection. Finally, in Section 3.3, we show that for the restricted system domain both — operational and abstract — models give the same identifications and

differentiations. As a consequence, verification in the abstract model can be realized hierarchically.

3.1 Preliminary results

This section contains results that allow us to relate sets of traces associated with computations of a parallel composition to sets of traces associated with computations of its components. The first two lemmata present intermediate results associated with po-traces.

Lemma 3.1.1

For system S, po-trace p of S, trace t of p, and trace u of $\mathbf{t}S$

$$(\forall T : T \in \mathbf{p}S : u\lceil \mathbf{a}T \leq t\lceil \mathbf{a}T) \;\Rightarrow\; u \in p.$$

Proof

Assume $(\forall T : T \in \mathbf{p}S : u\lceil \mathbf{a}T \leq t\lceil \mathbf{a}T)$. We derive

$$\begin{array}{ll}
& t \in p \\
\Leftrightarrow & \{\text{Lemma 2.3.17}\} \\
& t \in \mathbf{tr}.(\mathit{conf}.p \cup \mathit{act}.p) \\
\Leftrightarrow & \{t \in p \wedge p \in \mathit{ppot}.S \Rightarrow t \in \mathbf{t}S, \text{Definition 2.1.15}\} \\
& \{(t\lceil \mathbf{a}T, T) \mid T \in \mathbf{p}S\} \subseteq \mathit{conf}.p \cup \mathit{act}.p \\
\Rightarrow & \{\text{Corollary 2.3.14, from Definition 2.3.1 follows that } (v,T) \in \mathit{conf}.p \\
& \quad \text{implies } (\forall v' : v' \leq v : (v',T) \in \mathit{conf}.p), \text{assumption}\} \\
& \{(u\lceil \mathbf{a}T, T) \mid T \in \mathbf{p}S\} \subseteq \mathit{conf}.p \cup \mathit{act}.p \\
\Leftrightarrow & \{u \in \mathbf{t}S, \text{Definition 2.1.15}\} \\
& u \in \mathbf{tr}.(\mathit{conf}.p \cup \mathit{act}.p) \\
\Leftrightarrow & \{\text{Lemma 2.3.17}\} \\
& u \in p.
\end{array}$$

□

Lemma 3.1.2

For system S, po-trace p of S, and traces t and u of p

$$(\exists v : v \in p : (\forall T : T \in \mathbf{p}S : t\lceil \mathbf{a}T \leq v\lceil \mathbf{a}T \wedge u\lceil \mathbf{a}T \leq v\lceil \mathbf{a}T)).$$

Proof

We derive

$$
\begin{array}{ll}
& t \in p \wedge u \in p \\
\Rightarrow & \{\text{Corollary 2.3.25, Property 1.3.9, definition of } \leq \text{ for traces}\} \\
& t(u \setminus \flat.t) \in p \wedge t \leq t(u \setminus \flat.t) \\
\Rightarrow & \{\text{Lemma 1.1.15}\} \\
& t(u \setminus \flat.t) \in p \wedge (\forall T : T \in \mathbf{p}S : t\restriction \mathbf{a}T \leq t(u \setminus \flat.t)\restriction \mathbf{a}T).
\end{array}
$$

For traces t and u of p we have (compare this with Corollary A.1.6)

$$(\forall T : T \in \mathbf{p}S : t\restriction \mathbf{a}T \leq u\restriction \mathbf{a}T \vee u\restriction \mathbf{a}T \leq t\restriction \mathbf{a}T).$$

Let $T \in \mathbf{p}S$. We distinguish two cases.

Case 0 $\quad u\restriction \mathbf{a}T \leq t\restriction \mathbf{a}T$

Then

$$
\begin{array}{ll}
& t\restriction \mathbf{a}T \leq t(u \setminus \flat.t)\restriction \mathbf{a}T \\
\Rightarrow & \{u\restriction \mathbf{a}T \leq t\restriction \mathbf{a}T, \leq \text{ is transitive}\} \\
& u\restriction \mathbf{a}T \leq t(u \setminus \flat.t)\restriction \mathbf{a}T.
\end{array}
$$

Case 1 $\quad t\restriction \mathbf{a}T \leq u\restriction \mathbf{a}T$

Let $v \in (\mathbf{a}T)^*$, such that $u\restriction \mathbf{a}T = (t\restriction \mathbf{a}T)v$. We derive

$$
\begin{array}{ll}
& t(u \setminus \flat.t)\restriction \mathbf{a}T \\
= & \{\text{Property 1.1.1}\} \\
& (t\restriction \mathbf{a}T)(u\restriction \mathbf{a}T \setminus \flat.(t\restriction \mathbf{a}T)) \\
= & \{\text{definition of } v\} \\
& (t\restriction \mathbf{a}T)((t\restriction \mathbf{a}T)v \setminus \flat.(t\restriction \mathbf{a}T)) \\
= & \{\text{Property 1.1.1}\} \\
& (t\restriction \mathbf{a}T)(v \setminus \flat.\varepsilon) \\
= & \{\flat.\varepsilon = \emptyset,\ v \setminus \emptyset = v\} \\
& (t\restriction \mathbf{a}T)v \\
= & \{\text{definition of } v\} \\
& u\restriction \mathbf{a}T.
\end{array}
$$

Hence, $u\lceil \mathbf{a}T \leq t(u \setminus \flat.t)\lceil \mathbf{a}T$.
On account of the above derivations we have

$$(\forall T : T \in \mathbf{p}S : t\lceil \mathbf{a}T \leq t(u \setminus \flat.t)\lceil \mathbf{a}T \wedge u\lceil \mathbf{a}T \leq t(u \setminus \flat.t)\lceil \mathbf{a}T).$$

□

In the sequel, S_0 and S_1 are systems. Moreover, for $p_0 \in ppot.S_0$ and $p_1 \in ppot.S_1$,

p_0 **w** p_1 is used to denote $\mathbf{t}(\langle \mathbf{a}S_0, p_0\rangle \mathbf{w} \langle \mathbf{a}S_1, p_1\rangle)$.

The alphabets that play a rôle in p_0 **w** p_1 should be inferred from the context (i.e., from the definitions of S_0 and S_1).
By means of the following properties and lemmata we prove

$$ppot.(S_0 \parallel S_1) = \{p_0 \mathbf{w} p_1 \mid p_0 \in ppot.S_0 \wedge p_1 \in ppot.S_1\},$$

which is expressed by Corollaries 3.1.6 and 3.1.11.
First, $\supseteq$ is proved by Property 3.1.3, and Lemmata 3.1.4 and 3.1.5.
Property 1.1.7 and the definition of **w** yield that p_0 **w** p_1 is a non-empty and prefix-closed subset of $\mathbf{t}(S_0 \parallel S_1)$. This is expressed by Property 3.1.3.

Property 3.1.3

Let $p_0 \in ppot.S_0$ and $p_1 \in ppot.S_1$. Then

$$p_0 \mathbf{w} p_1 \neq \emptyset \wedge pref.(p_0 \mathbf{w} p_1) = p_0 \mathbf{w} p_1 \wedge p_0 \mathbf{w} p_1 \subseteq \mathbf{t}(S_0 \parallel S_1).$$

□

Lemmata 3.1.4 and 3.1.5 prove that p_0 **w** p_1 also satisfies the last two requirements of Definition 2.3.2.

Lemma 3.1.4

For $p_0 \in ppot.S_0$ and $p_1 \in ppot.S_1$

$$(\forall t, u, a, b, T : T \in \mathbf{p}S_0 \cup \mathbf{p}S_1 \wedge \{a, b\} \subseteq \mathbf{a}T \wedge \{ta, ub\} \subseteq p_0 \mathbf{w} p_1 : t\lceil \mathbf{a}T = u\lceil \mathbf{a}T \Rightarrow a = b).$$

Proof

Let $T \in \mathbf{p}S_0$, $\{a, b\} \subseteq \mathbf{a}T$, and $t\lceil \mathbf{a}T = u\lceil \mathbf{a}T$. We derive

$\{ta, ub\} \subseteq p_0 \mathbf{w} p_1$
$\Rightarrow$ { definition of $\mathbf{w}$, predicate calculus }
$ta\restriction \mathbf{a}S_0 \in p_0 \wedge ub\restriction \mathbf{a}S_0 \in p_0$
$\Leftrightarrow$ { $T \in \mathbf{p}S_0$, $\{a, b\} \subseteq \mathbf{a}T$, definition of $\restriction$ }
$(t\restriction \mathbf{a}S_0)a \in p_0 \wedge (u\restriction \mathbf{a}S_0)b \in p_0$
$\Rightarrow$ { $T \in \mathbf{p}S$, hence, $(t\restriction \mathbf{a}S_0)\restriction \mathbf{a}T = t\restriction \mathbf{a}T$ and $(u\restriction \mathbf{a}S_0)\restriction \mathbf{a}T = u\restriction \mathbf{a}T$, $t\restriction \mathbf{a}T = u\restriction \mathbf{a}T$, $p_0 \in ppot.S_0$ }
$a = b$.

In the same way the derivation for $T \in \mathbf{p}S_1$ and p_1 can be given.

□

Lemma 3.1.5

For $p_0 \in ppot.S_0$ and $p_1 \in ppot.S_1$

$$(\forall t, u : t \in p_0 \mathbf{w} p_1 \wedge u \in \mathbf{t}(S_0 \parallel S_1) \wedge (\forall T : T \in \mathbf{p}S_0 \cup \mathbf{p}S_1 : t\restriction \mathbf{a}T = u\restriction \mathbf{a}T) : u \in p_0 \mathbf{w} p_1).$$

Proof

Let $t \in p_0 \mathbf{w} p_1$ and $u \in (\mathbf{a}S_0 \cup \mathbf{a}S_1)^*$ satisfy $(\forall T : T \in \mathbf{p}S_0 \cup \mathbf{p}S_1 : t\restriction \mathbf{a}T = u\restriction \mathbf{a}T)$. We derive

$u \in \mathbf{t}(S_0 \parallel S_1)$
$\Leftrightarrow$ { $t \in p_0 \mathbf{w} p_1$, $u \in (\mathbf{a}S_0 \cup \mathbf{a}S_1)^*$, definition of $\mathbf{w}$ }
$t\restriction \mathbf{a}S_0 \in p_0 \wedge t\restriction \mathbf{a}S_1 \in p_1 \wedge u\restriction \mathbf{a}S_0 \in \mathbf{t}S_0 \wedge u\restriction \mathbf{a}S_1 \in \mathbf{t}S_1$
$\Rightarrow$ { $T \in \mathbf{p}S_0$, hence, $(t\restriction \mathbf{a}S_0)\restriction \mathbf{a}T = t\restriction \mathbf{a}T$ and $t\restriction \mathbf{a}T = u\restriction \mathbf{a}T$, $p_0 \in ppot.S_0$, similar arguments hold for $T \in \mathbf{p}S_1$ and p_1 }
$u\restriction \mathbf{a}S_0 \in p_0 \wedge u\restriction \mathbf{a}S_1 \in p_1$
$\Leftrightarrow$ { definition of $\mathbf{w}$, $u \in (\mathbf{a}S_0 \cup \mathbf{a}S_1)^*$ }
$u \in p_0 \mathbf{w} p_1$.

□

The following corollary is a consequence of Property 3.1.3, and Lemmata 3.1.4 and 3.1.5. It shows that the weave of po-traces of systems S_0 and S_1 is a po-trace of their parallel composition.

Corollary 3.1.6

For systems S_0 and S_1

$$ppot.(S_0 \parallel S_1) \supseteq \{p_0 \mathbf{w} p_1 \mid p_0 \in ppot.S_0 \wedge p_1 \in ppot.S_1\}.$$

□

Next, $\subseteq$ is proved by Corollary 3.1.7, Property 3.1.9, and Lemmata 3.1.8 and 3.1.10. In the sequel, p is a po-trace of $S_0 \parallel S_1$. Moreover, for $i \in \{0,1\}$, we define

$$p_i = \{t \mid t \in \mathbf{t}S_i \wedge (\exists u : u \in p\lceil \mathbf{a}S_i : (\forall T : T \in \mathbf{p}S_i : t\lceil \mathbf{a}T \leq u\lceil \mathbf{a}T))\}.$$

In the above definition, we could equally well use $=$ instead of $\leq$. However, it is much easier to prove statements about $\leq$ than about $=$.
Corollary 3.1.7 follows from Property 1.1.3, and the definitions of $\mathbf{w}$, p_0, and p_1. It shows that, for $i \in \{0,1\}$, p_i is a non-empty prefix-closed subset of $\mathbf{t}S_i$.

Corollary 3.1.7

For $i = 0$ or $i = 1$ we have

$$p_i \neq \emptyset \wedge pref.p_i = p_i \wedge p_i \subseteq \mathbf{t}S_i.$$

□

The following lemma proves that, for $i \in \{0,1\}$, p_i satisfies the third condition of Definition 2.3.2.

Lemma 3.1.8

Let $i \in \{0,1\}$. Then

$$(\forall t, u, a, b, T : T \in \mathbf{p}S_i \wedge \{a,b\} \subseteq \mathbf{a}T \wedge \{ta, ub\} \subseteq p_i \wedge t\lceil \mathbf{a}T = u\lceil \mathbf{a}T : a = b).$$

Proof

Let $T \in \mathbf{p}S_i$, $\{a,b\} \subseteq \mathbf{a}T$, and $\{ta,ub\} \subseteq p_i$. We derive

$$\begin{array}{ll} & t\lceil \mathbf{a}T = u\lceil \mathbf{a}T \\ \Leftrightarrow & \{\text{definition of } p_i,\ \{ta,ub\} \subseteq p_i\} \\ & (\exists t' : t' \in p\lceil \mathbf{a}S_i : (\forall U : U \in \mathbf{p}S_i : ta\lceil \mathbf{a}U \leq t'\lceil \mathbf{a}U)) \wedge \end{array}$$

$(\exists u' : u' \in p\restriction \mathbf{a}S_i : (\forall U : U \in \mathbf{p}S_i : ub\restriction \mathbf{a}U \leq u'\restriction \mathbf{a}U)) \wedge t\restriction \mathbf{a}T = u\restriction \mathbf{a}T$

$\Rightarrow$ {predicate calculus, $T \in \mathbf{p}S_i$}

$(\exists t', u' : \{t', u'\} \subseteq p\restriction \mathbf{a}S_i : ta\restriction \mathbf{a}T \leq t'\restriction \mathbf{a}T \wedge ub\restriction \mathbf{a}T \leq u'\restriction \mathbf{a}T) \wedge t\restriction \mathbf{a}T = u\restriction \mathbf{a}T$

$\Rightarrow$ {$(\mathbf{a}S_0 \cup \mathbf{a}S_1, p)$ is a process, Property 1.1.3, $\{a, b\} \subseteq \mathbf{a}T$, definition of $\restriction$}

$(t\restriction \mathbf{a}T)a \in p\restriction \mathbf{a}S_i \wedge (u\restriction \mathbf{a}T)b \in p\restriction \mathbf{a}S_i \wedge t\restriction \mathbf{a}T = u\restriction \mathbf{a}T$

$\Rightarrow$ {predicate calculus}

$(\exists t', u' : \{t'a, u'b\} \subseteq p\restriction \mathbf{a}S_i : t'\restriction \mathbf{a}T = u'\restriction \mathbf{a}T)$

$\Rightarrow$ {property of $\restriction$, $p \in ppot.(S_0 \parallel S_1)$, hence, p is prefix-closed, calculus}

$(\exists t', u' : \{t'a, u'b\} \subseteq p : t'\restriction \mathbf{a}T = u'\restriction \mathbf{a}T)$

$\Rightarrow$ {$p \in ppot.(S_0 \parallel S_1)$}

$a = b.$

□

The following property follows from the definition of p_i. It shows that, for $i \in \{0,1\}$, p_i satisfies the forth requirement of Definition 2.3.2.

Property 3.1.9

For $i = 0$ or $i = 1$

$$(\forall t, u : t \in p_i \wedge u \in \mathbf{t}S_i \wedge (\forall T : T \in \mathbf{p}S_i : t\restriction \mathbf{a}T = u\restriction \mathbf{a}T) : u \in p_i).$$

□

Lemma 3.1.10

Let p be a po-trace of $S_0 \parallel S_1$. For p_0 and p_1 as defined above, we have

$$p = p_0 \mathbf{w} p_1.$$

Proof

We derive

$t \in p$

$\Rightarrow$ {$p \subseteq \mathbf{t}(S_0 \parallel S_1)$, property of $\restriction$}

$t \in (\mathbf{a}S_0 \cup \mathbf{a}S_1)^* \wedge t\restriction \mathbf{a}S_0 \in p\restriction \mathbf{a}S_0 \wedge t\restriction \mathbf{a}S_1 \in p\restriction \mathbf{a}S_1$

$\Rightarrow$ {definition of p_0 and p_1}

$t \in (\mathbf{a}S_0 \cup \mathbf{a}S_1)^* \wedge t\restriction \mathbf{a}S_0 \in p_0 \wedge t\restriction \mathbf{a}S_1 \in p_1$

$\Leftrightarrow$ { definition of **w** }
$t \in p_0 \mathbf{w} p_1$.

Hence, $p \subseteq p_0 \mathbf{w} p_1$.
For the inclusion the other way around we derive

$t \in p_0 \mathbf{w} p_1$
$\Rightarrow$ { definition of **w** }
$t\lceil \mathbf{a}S_0 \in p_0 \wedge t\lceil \mathbf{a}S_1 \in p_1$
$\Leftrightarrow$ { definition of p_0 and p_1 }
$(\exists u_0 : u_0 \in p\lceil \mathbf{a}S_0 : (\forall T : T \in \mathbf{p}S_0 : (t\lceil \mathbf{a}S_0)\lceil \mathbf{a}T \leq u_0\lceil \mathbf{a}T)) \wedge$
$(\exists u_1 : u_1 \in p\lceil \mathbf{a}S_1 : (\forall T : T \in \mathbf{p}S_1 : (t\lceil \mathbf{a}S_1)\lceil \mathbf{a}T \leq u_1\lceil \mathbf{a}T))$
$\Rightarrow$ { Property 1.1.19, from $T \in \mathbf{p}S$ follows $\mathbf{a}S \cap \mathbf{a}T = \mathbf{a}T$, property of $\lceil$ }
$(\exists u_0 : u_0 \in p : (\forall T : T \in \mathbf{p}S_0 : t\lceil \mathbf{a}T \leq u_0\lceil \mathbf{a}T)) \wedge$
$(\exists u_1 : u_1 \in p : (\forall T : T \in \mathbf{p}S_1 : t\lceil \mathbf{a}T \leq u_1\lceil \mathbf{a}T))$
$\Rightarrow$ { Lemma 3.1.2, $\leq$ is transitive, predicate calculus }
$(\exists v : v \in p : (\forall T : T \in \mathbf{p}S_0 \cup \mathbf{p}S_1 : t\lceil \mathbf{a}T \leq v\lceil \mathbf{a}T))$
$\Rightarrow$ { Lemma 3.1.1 }
$t \in p$.

Hence, $p \supseteq p_0 \mathbf{w} p_1$.

□

Corollary 3.1.7, Property 3.1.9, and Lemma 3.1.8 together prove that, on account of Definition 2.3.2, p_0 is a po-trace of S_0 and p_1 is a po-trace of S_1. Moreover, together with Lemma 3.1.10 this gives the following result expressing that every po-trace of the parallel composition can be obtained as the weave of po-traces of its components.

Corollary 3.1.11

For systems S_0 and S_1

$$ppot.(S_0 \parallel S_1) \subseteq \{p_0 \mathbf{w} p_1 \mid p_0 \in ppot.S_0 \wedge p_1 \in ppot.S_1\}.$$

□

Combining Corollaries 3.1.6 and 3.1.11 we get the equality of both sets.

Corollary 3.1.12

For systems S_0 and S_1

$$ppot.(S_0 \parallel S_1) = \{p_0 \mathbf{w} p_1 \mid p_0 \in ppot.S_0 \wedge p_1 \in ppot.S_1\}.$$

□

An immediate consequence is that $pot.(S_0 \parallel S_1)$ is equal to the set of maximal elements of $\{p_0 \mathbf{w} p_1 \mid p_0 \in pot.S_0 \wedge p_1 \in pot.S_1\}$. This is why we only have $\subseteq$ in Corollary 3.1.13.

Corollary 3.1.13

For systems S_0 and S_1

$$pot.(S_0 \parallel S_1) \subseteq \{p_0 \mathbf{w} p_1 \mid p_0 \in pot.S_0 \wedge p_1 \in pot.S_1\}.$$

□

As an example, consider systems

$$S_0 = \langle \{a, b\}, \{\mathbf{pr}(a \mid b)\} \rangle$$

and

$$S_1 = \langle \{a, b\}, \{\mathbf{pr}a\} \rangle.$$

Then $p_0 = \{\varepsilon, b\}$ belongs to $pot.S_0$, and $p_1 = \{\varepsilon, a\}$ belongs to $pot.S_1$, but $p_0 \mathbf{w} p_1 = \{\varepsilon\}$ does not belong to $pot.(S_0 \parallel S_1)$. This is a consequence of the fact that in the parallel composition of S_0 and S_1 the choice between a and b present in S_0 is reduced by S_1 to a only. Hence, the only element of $pot.(S_0 \parallel S_1)$ is $\{\varepsilon, a\}$.

However, for systems with disjoint external alphabets, we do have the equality of the sets from Corollary 3.1.13.

Lemma 3.1.14

For systems S_0 and S_1, such that $\mathbf{e}S_0 \cap \mathbf{e}S_1 = \emptyset$,

$$pot.(S_0 \parallel S_1) \supseteq \{p_0 \mathbf{w} p_1 \mid p_0 \in pot.S_0 \wedge p_1 \in pot.S_1\}.$$

Proof

We assume that $\mathbf{a}S_0 \cap \mathbf{a}S_1 = \mathbf{e}S_0 \cap \mathbf{e}S_1$ $(= \emptyset)$ holds.

Let $p_0 \in pot.S_0$ and $p_1 \in pot.S_1$. Then, on account of Corollary 3.1.12, $p_0 \mathbf{w} p_1$ is an element of $ppot.(S_0 \parallel S_1)$. Consequently, by Property 2.3.3, it is sufficient to prove

$$(\forall t, a : ta \in \mathbf{t}(S_0 \parallel S_1) \setminus p_0 \mathbf{w} p_1 \wedge t \in p_0 \mathbf{w} p_1 \\ : (\exists T, u, b : T \in \mathbf{p}(S_0 \parallel S_1) \wedge ub \in p_0 \mathbf{w} p_1 \wedge u\restriction \mathbf{a}T = t\restriction \mathbf{a}T \\ : \{a, b\} \subseteq \mathbf{a}T \wedge a \neq b)).$$

We derive

$$\begin{array}{ll} & ta \in \mathbf{t}(S_0 \parallel S_1) \setminus p_0 \mathbf{w} p_1 \wedge t \in p_0 \mathbf{w} p_1 \\ \Leftrightarrow & \{\text{definition of } \mathbf{w} \text{ and } \mathbf{t}S\} \\ & ta \in (\mathbf{a}S_0 \cup \mathbf{a}S_1)^* \wedge ta\restriction \mathbf{a}S_0 \in \mathbf{t}S_0 \wedge ta\restriction \mathbf{a}S_1 \in \mathbf{t}S_1 \wedge \\ & (ta\restriction \mathbf{a}S_0 \notin p_0 \vee ta\restriction \mathbf{a}S_1 \notin p_1) \wedge t\restriction \mathbf{a}S_0 \in p_0 \wedge t\restriction \mathbf{a}S_1 \in p_1 \\ \Rightarrow & \{\mathbf{a}S_0 \cap \mathbf{a}S_1 = \varnothing, \text{ set and predicate calculus}\} \\ & (a \in \mathbf{a}S_0 \setminus \mathbf{a}S_1 \vee a \in \mathbf{a}S_1 \setminus \mathbf{a}S_0) \wedge ta\restriction \mathbf{a}S_0 \in \mathbf{t}S_0 \wedge ta\restriction \mathbf{a}S_1 \in \mathbf{t}S_1 \wedge \\ & (ta\restriction \mathbf{a}S_0 \notin p_0 \vee ta\restriction \mathbf{a}S_1 \notin p_1) \wedge t\restriction \mathbf{a}S_0 \in p_0 \wedge t\restriction \mathbf{a}S_1 \in p_1 \\ \Rightarrow & \{\text{definition of } \restriction, \text{ predicate calculus}\} \\ & (a \in \mathbf{a}S_0 \setminus \mathbf{a}S_1 \wedge (t\restriction \mathbf{a}S_0)a \in \mathbf{t}S_0 \setminus p_0 \wedge t\restriction \mathbf{a}S_0 \in p_0) \\ & \vee (a \in \mathbf{a}S_1 \setminus \mathbf{a}S_0 \wedge (t\restriction \mathbf{a}S_1)a \in \mathbf{t}S_1 \setminus p_1 \wedge t\restriction \mathbf{a}S_1 \in p_1) \\ \Rightarrow & \{p_0 \in pot.S_0 \text{ and } p_1 \in pot.S_1, \text{ Property 2.3.3, furthermore,} \\ & \quad \text{by Property 1.1.19, } T \in \mathbf{p}S_i \Rightarrow t\restriction \mathbf{a}S_i\restriction \mathbf{a}T = t\restriction \mathbf{a}T, \text{ for } i = 0 \text{ and } i = 1\} \\ & (\exists T, u_0, b : T \in \mathbf{p}S_0 \wedge u_0 b \in p_0 \wedge u_0\restriction \mathbf{a}T = t\restriction \mathbf{a}T : \{a, b\} \subseteq \mathbf{a}T \wedge a \neq b) \\ & \vee (\exists T, u_1, b : T \in \mathbf{p}S_1 \wedge u_1 b \in p_1 \wedge u_1\restriction \mathbf{a}T = t\restriction \mathbf{a}T : \{a, b\} \subseteq \mathbf{a}T \wedge a \neq b) \\ \Rightarrow & \{\mathbf{a}S_0 \cap \mathbf{a}S_1 = \varnothing, \text{ hence on account of Lemma 1.1.21, } (p_0 \mathbf{w} p_1)\restriction \mathbf{a}S_i = p_i, \\ & \quad \text{for } i = 0 \text{ and } i = 1\} \\ & (\exists T, u, b : T \in \mathbf{p}S_0 \wedge ub \in p_0 \mathbf{w} p_1 \wedge u\restriction \mathbf{a}T = t\restriction \mathbf{a}T : \{a, b\} \subseteq \mathbf{a}T \wedge a \neq b) \\ & \vee (\exists T, u, b : T \in \mathbf{p}S_1 \wedge ub \in p_0 \mathbf{w} p_1 \wedge u\restriction \mathbf{a}T = t\restriction \mathbf{a}T : \{a, b\} \subseteq \mathbf{a}T \wedge a \neq b) \\ \Leftrightarrow & \{\text{predicate calculus}\} \\ & (\exists T, u, b : T \in \mathbf{p}S_0 \cup \mathbf{p}S_1 \wedge ub \in p_0 \mathbf{w} p_1 \wedge u\restriction \mathbf{a}T = t\restriction \mathbf{a}T \\ & \quad : \{a, b\} \subseteq \mathbf{a}T \wedge a \neq b). \end{array}$$

Hence, on account of Property 2.3.3, $p_0 \mathbf{w} p_1$ belongs to $pot.(S_0 \parallel S_1)$.

□

In the sequel, for $X_0 \in pot.S_0\restriction \mathbf{e}S_0$ and $X_1 \in pot.S_1\restriction \mathbf{e}S_1$,

$X_0 \mathbf{w} X_1$ is used to denote $\mathbf{t}(\langle \mathbf{e}S_0, X_0\rangle \mathbf{w} \langle \mathbf{e}S_1, X_1\rangle)$.

The alphabets that play a rôle in $X_0 \mathbf{w} X_1$ are $\mathbf{e}S_0$ and $\mathbf{e}S_1$, respectively.

Because p_0 **w** p_1 does not necessarily belong to $pot.(S_0 \parallel S_1)$, for $p_0 \in pot.S_0$ and $p_1 \in pot.S_1$, in general, we also do not have the equality of the sets from Corollary 3.1.13 projected on the union of the external alphabets. That is, in general,

$$pot.(S_0 \parallel S_1)\restriction(\mathbf{e}S_0 \cup \mathbf{e}S_1) = \{X_0 \mathbf{w} X_1 \mid X_0 \in pot.S_0\restriction\mathbf{e}S_0 \wedge X_1 \in pot.S_1\restriction\mathbf{e}S_1\}$$

does not necessarily hold. Since in the example discussed above $\mathbf{a}S_0 = \mathbf{e}S_0$ and $\mathbf{a}S_1 = \mathbf{e}S_1$, exactly the same arguments can be used to justify this statement.

However, we do have one inclusion, namely $\subseteq$, which is expressed by the following lemma.

Lemma 3.1.15

For systems S_0 and S_1

$$pot.(S_0 \parallel S_1)\restriction(\mathbf{e}S_0 \cup \mathbf{e}S_1) \subseteq \{X_0 \mathbf{w} X_1 \mid X_0 \in pot.S_0\restriction\mathbf{e}S_0 \wedge X_1 \in pot.S_1\restriction\mathbf{e}S_1\}.$$

Proof

Assume $\mathbf{a}S_0 \cap \mathbf{a}S_1 = \mathbf{e}S_0 \cap \mathbf{e}S_1$. We derive

$$\begin{array}{ll} & X \in pot.(S_0 \parallel S_1)\restriction(\mathbf{e}S_0 \cup \mathbf{e}S_1) \\ \Leftrightarrow & \{\text{definition of projection}\} \\ & (\exists p : p \in pot.(S_0 \parallel S_1) : X = p\restriction(\mathbf{e}S_0 \cup \mathbf{e}S_1)) \\ \Rightarrow & \{\text{Corollary 3.1.13}\} \\ & (\exists p_0, p_1 : p_0 \in pot.S_0 \wedge p_1 \in pot.S_1 : X = (p_0 \mathbf{w} p_1)\restriction(\mathbf{e}S_0 \cup \mathbf{e}S_1)) \\ \Leftrightarrow & \{\text{Lemma 1.1.13, } \mathbf{a}S_0 \cap \mathbf{a}S_1 = \mathbf{e}S_0 \cap \mathbf{e}S_1\} \\ & (\exists p_0, p_1 : p_0 \in pot.S_0 \wedge p_1 \in pot.S_1 : X = p_0\restriction\mathbf{e}S_0 \mathbf{w} p_1\restriction\mathbf{e}S_1) \\ \Leftrightarrow & \{\text{predicate calculus, definition of projection}\} \\ & (\exists X_0, X_1 : X_0 \in pot.S_0\restriction\mathbf{e}S_0 \wedge X_1 \in pot.S_1\restriction\mathbf{e}S_1 : X = X_0 \mathbf{w} X_1). \end{array}$$

□

Lemma 3.1.16 shows that for systems with disjoint external alphabets we also have the other inclusion.

Lemma 3.1.16

For systems S_0 and S_1, such that $\mathbf{e}S_0 \cap \mathbf{e}S_1 = \emptyset$,

$$pot.(S_0 \parallel S_1)\restriction(\mathbf{e}S_0 \cup \mathbf{e}S_1) \supseteq \{X_0 \mathbf{w} X_1 \mid X_0 \in pot.S_0\restriction\mathbf{e}S_0 \wedge X_1 \in pot.S_1\restriction\mathbf{e}S_1\}.$$

Proof

Assume $\mathbf{a}S_0 \cap \mathbf{a}S_1 = \mathbf{e}S_0 \cap \mathbf{e}S_1$ $(= \emptyset)$. We derive

$$
\begin{array}{ll}
& X_0 \in pot.S_0 \restriction \mathbf{e}S_0 \wedge X_1 \in pot.S_1 \restriction \mathbf{e}S_1 \\
\Leftrightarrow & \{\text{definition of projection}\} \\
& (\exists p_0, p_1 : p_0 \in pot.S_0 \wedge p_1 \in pot.S_1 : X_0 = p_0 \restriction \mathbf{e}S_0 \wedge X_1 = p_1 \restriction \mathbf{e}S_1) \\
\Rightarrow & \{\mathbf{e}S_0 \cap \mathbf{e}S_1 = \emptyset, \text{Lemma 3.1.14}\} \\
& (\exists p_0, p_1 : p_0 \in pot.S_0 \wedge p_1 \in pot.S_1 \wedge p_0 \mathbf{w} p_1 \in pot.(S_0 \parallel S_1) \\
& \qquad : X_0 = p_0 \restriction \mathbf{e}S_0 \wedge X_1 = p_1 \restriction \mathbf{e}S_1) \\
\Leftrightarrow & \{\mathbf{a}S_0 \cap \mathbf{a}S_1 = \emptyset, \text{hence on account of Lemma 1.1.21, } (p_0 \mathbf{w} p_1) \restriction \mathbf{a}S_i = p_i; \\
& \quad \mathbf{a}S_i \cap \mathbf{e}S_i = \mathbf{e}S_i, \text{Property 1.1.19, for } i = 0 \text{ and } i = 1\} \\
& (\exists p_0, p_1 : p_0 \in pot.S_0 \wedge p_1 \in pot.S_1 \wedge p_0 \mathbf{w} p_1 \in pot.(S_0 \parallel S_1) \\
& \qquad : X_0 = (p_0 \mathbf{w} p_1) \restriction \mathbf{e}S_0 \wedge X_1 = (p_0 \mathbf{w} p_1) \restriction \mathbf{e}S_1) \\
\Leftrightarrow & \{\mathbf{a}S_0 \cap \mathbf{a}S_1 = \mathbf{e}S_0 \cap \mathbf{e}S_1, \text{Lemma 1.1.13, predicate calculus}\} \\
& (\exists p_0, p_1 : (p_0 \mathbf{w} p_1) \in pot.(S_0 \parallel S_1) : (p_0 \mathbf{w} p_1) \restriction (\mathbf{e}S_0 \cup \mathbf{e}S_1) = X_0 \mathbf{w} X_1) \\
\Rightarrow & \{\text{predicate calculus}\} \\
& (\exists p : p \in pot.(S_0 \parallel S_1) : p \restriction (\mathbf{e}S_0 \cup \mathbf{e}S_1) = X_0 \mathbf{w} X_1) \\
\Leftrightarrow & \{\text{definition of projection}\} \\
& X_0 \mathbf{w} X_1 \in pot.(S_0 \parallel S_1) \restriction (\mathbf{e}S_0 \cup \mathbf{e}S_1).
\end{array}
$$

□

3.2 Abstract model

The operational model gives a view of how systems behave, and testing relations serve the purpose of identifying (or distinguishing between) them. In the sequel we define an abstract model of systems that yields a more straightforward way of establishing the same results for a restricted system domain.

In the sequel, the following abbreviation is used.

Definition 3.2.1

For system S

$$\mathbf{c}S = pot.S \restriction \mathbf{e}S.$$

□

From now on we use the name *computation* for the elements of $\mathbf{c}S$.

On account of Corollary 2.3.22, using $\mathbf{c}S$ we can rewrite the definition of acceptances as follows.

Corollary 3.2.2

For $t \in \mathbf{t}(\mathbf{pr}S)$ and $L \subseteq \mathbf{e}S^*$

$$(t, L) \in \mathbf{ac}S \;\Leftrightarrow\; (\forall X : X \in \mathbf{c}S \wedge t \in X : \{v \mid tv \in X\} \cap L \neq \emptyset).$$

□

The reason for introducing testing relations is the fact that otherwise we would not be able to distinguish two systems, one with a nondeterministic choice and another one with deterministic choice between two external actions (see Example 2.2.7), only by means of acceptances. A first guess for a suitable abstract model of system S, denoted by $\mathcal{A}[\![S]\!]$, would then be

$$\mathcal{A}[\![S]\!] = (\mathbf{e}S, \mathbf{f}S, \mathbf{ac}S).$$

Relation $\sqsupseteq$ on abstract models of systems S_0 and S_1 would then be defined by

$$\mathcal{A}[\![S_0]\!] \sqsupseteq \mathcal{A}[\![S_1]\!] \;\Leftrightarrow\; \mathbf{e}S_0 = \mathbf{e}S_1 \wedge \mathbf{f}S_0 \subseteq \mathbf{f}S_1 \wedge \mathbf{ac}S_0 \supseteq \mathbf{ac}S_1.$$

Unfortunately, this idea was unsuccessful, because

- we were not able to derive suitable expressions for failures and acceptances in order to achieve compositionality with respect to parallel composition and projection,
- systems like S_0 and S_2 of Example 2.2.7 would be different according to the above relation ($\mathcal{A}[\![S_2]\!] \not\sqsupseteq \mathcal{A}[\![S_0]\!]$), whereas they have the same external behaviour (S_0 **equ** S_2).

The latter is a consequence of the fact that in the definition of *div* the information about concurrency is neglected. We have to admit that it is not easy to formalize the notion of divergences using concurrency, especially if compositionality with respect to $\|$ and $\restriction$ is required. An attempt at such a formalization may be found in [Klo].

In the next attempt, $\mathbf{ac}S$ was changed into $\mathbf{c}S$ in the definition of $\mathcal{A}[\![S]\!]$. Then one problem disappeared: computations of $S \restriction A$ can be expressed in terms of computations of S. But we were not able to find a suitable expression for computations of parallel composition. In the meantime we had discovered that the last problem can be solved

for a certain class of systems. This class is called *RD* (from *Restricted system Domain*) and it contains systems with nondeterministic choices that do not depend on external actions (as informally described at the beginning of this chapter). Class *RD* of systems is closed under parallel composition and projection, which is proved by Lemmata 3.2.5 and 3.2.6. Formally, *RD* is defined as follows.

Definition 3.2.3

Let S be a system. Then

$$S \in RD \;\Leftrightarrow\; (\forall t,p,a : p \in pot.S \wedge t \in p \wedge ta \in \mathbf{t}S \setminus p : (\exists T,b : T \in \mathbf{p}S \wedge tb \in p : \{a,b\} \subseteq \mathbf{a}T \setminus \mathbf{e}S)).$$

□

The following property supplies an intermediate result used in the proofs of Lemmata 3.2.5 and 3.2.16.

Property 3.2.4

Let $S_0 \in RD$, $S_1 \in RD$, $p_0 \in pot.S_0$, and $p_1 \in pot.S_1$. Then

$$t \in p_0 \,\mathbf{w}\, p_1 \wedge ta \in \mathbf{t}(S_0 \parallel S_1) \setminus p_0 \,\mathbf{w}\, p_1$$
$$\Rightarrow$$
$$(\exists T,b : T \in \mathbf{p}S_0 \cup \mathbf{p}S_1 \wedge tb \in p_0 \,\mathbf{w}\, p_1 : \{a,b\} \subseteq \mathbf{a}T \setminus (\mathbf{e}S_0 \cup \mathbf{e}S_1)).$$

Proof

Assume that $\mathbf{a}S_0 \cap \mathbf{a}S_1 = \mathbf{e}S_0 \cap \mathbf{e}S_1$.
Let $t \in (\mathbf{a}S_0 \cup \mathbf{a}S_1)^*$. We derive

$$\begin{array}{ll}
& t \in p_0 \,\mathbf{w}\, p_1 \wedge ta \notin p_0 \,\mathbf{w}\, p_1 \wedge ta \in \mathbf{t}(S_0 \parallel S_1) \\
\Leftrightarrow & \quad \{\, t \in (\mathbf{a}S_0 \cup \mathbf{a}S_1)^*, \text{ definition of } \mathbf{w} \,\} \\
& t\restriction \mathbf{a}S_0 \in p_0 \wedge t\restriction \mathbf{a}S_1 \in p_1 \wedge ((ta)\restriction \mathbf{a}S_0 \notin p_0 \vee (ta)\restriction \mathbf{a}S_1 \notin p_1) \wedge \\
& (ta)\restriction \mathbf{a}S_0 \in \mathbf{t}S_0 \wedge (ta)\restriction \mathbf{a}S_1 \in \mathbf{t}S_1 \wedge a \in \mathbf{a}S_0 \cup \mathbf{a}S_1 \\
\Leftrightarrow & \quad \{\, \text{predicate and set calculus} \,\} \\
& t\restriction \mathbf{a}S_0 \in p_0 \wedge t\restriction \mathbf{a}S_1 \in p_1 \wedge (a \in \mathbf{a}S_0 \setminus \mathbf{a}S_1 \vee a \in \mathbf{a}S_0 \cap \mathbf{a}S_1 \vee a \in \mathbf{a}S_1 \setminus \mathbf{a}S_0) \wedge \\
& (((ta)\restriction \mathbf{a}S_0 \in \mathbf{t}S_0 \setminus p_0 \wedge (ta)\restriction \mathbf{a}S_1 \in \mathbf{t}S_1) \vee ((ta)\restriction \mathbf{a}S_1 \in \mathbf{t}S_1 \setminus p_1 \wedge (ta)\restriction \mathbf{a}S_0 \in \mathbf{t}S_0)) \\
\Leftrightarrow & \quad \{\, \text{definition of } \restriction, \text{ set and predicate calculus} \,\} \\
& t\restriction \mathbf{a}S_0 \in p_0 \wedge t\restriction \mathbf{a}S_1 \in p_1 \wedge
\end{array}$$

$$
\begin{array}{ll}
& ((a \in \mathbf{a}S_0 \setminus \mathbf{a}S_1 \wedge (t\restriction \mathbf{a}S_0)a \in \mathbf{t}S_0 \setminus p_0) \\
& \quad \vee (a \in \mathbf{a}S_0 \cap \mathbf{a}S_1 \wedge (t\restriction \mathbf{a}S_0)a \in \mathbf{t}S_0 \setminus p_0 \wedge (t\restriction \mathbf{a}S_1)a \in \mathbf{t}S_1) \\
& \quad \vee (a \in \mathbf{a}S_0 \cap \mathbf{a}S_1 \wedge (t\restriction \mathbf{a}S_0)a \in \mathbf{t}S_0 \wedge (t\restriction \mathbf{a}S_1)a \in \mathbf{t}S_1 \setminus p_1) \\
& \quad \vee (a \in \mathbf{a}S_1 \setminus \mathbf{a}S_0 \wedge (t\restriction \mathbf{a}S_1)a \in \mathbf{t}S_1 \setminus p_1)) \\
\Leftrightarrow & \quad \{\text{since } S_0 \in RD,\ S_1 \in RD,\ p_0 \in pot.S_0, \text{ and } p_1 \in pot.S_1, \text{ on account of} \\
& \quad\ \text{Definition 3.2.3 we have } t\restriction \mathbf{a}S_i \in p_i \wedge (t\restriction \mathbf{a}S_i)a \in \mathbf{t}S_i \setminus p_i \Rightarrow a \notin \mathbf{e}S_i, \\
& \quad\ \text{for } i = 0 \text{ and } i = 1,\ \mathbf{a}S_0 \cap \mathbf{a}S_1 = \mathbf{e}S_0 \cap \mathbf{e}S_1\} \\
& t\restriction \mathbf{a}S_0 \in p_0 \wedge t\restriction \mathbf{a}S_1 \in p_1 \wedge \\
& ((a \in \mathbf{a}S_0 \setminus \mathbf{a}S_1 \wedge (t\restriction \mathbf{a}S_0)a \in \mathbf{t}S_0 \setminus p_0) \vee (a \in \mathbf{a}S_1 \setminus \mathbf{a}S_0 \wedge (t\restriction \mathbf{a}S_1)a \in \mathbf{t}S_1 \setminus p_1)) \\
\Rightarrow & \quad \{S_0 \in RD \text{ and } S_1 \in RD,\ p_0 \in pot.S_0 \text{ and } p_1 \in pot.S_1, \text{ predicate calculus}\} \\
& t\restriction \mathbf{a}S_0 \in p_0 \wedge t\restriction \mathbf{a}S_1 \in p_1 \wedge \\
& ((\exists T, b : T \in \mathbf{p}S_0 \wedge (t\restriction \mathbf{a}S_0)b \in p_0 : \{a, b\} \subseteq \mathbf{a}T \setminus \mathbf{e}S_0) \\
& \quad \vee (\exists T, b : T \in \mathbf{p}S_1 \wedge (t\restriction \mathbf{a}S_1)b \in p_1 : \{a, b\} \subseteq \mathbf{a}T \setminus \mathbf{e}S_1)) \\
\Rightarrow & \quad \{\text{definition of } \mathbf{w},\ t \in (\mathbf{a}S_0 \cup \mathbf{a}S_1)^*,\ \mathbf{a}S_0 \cap \mathbf{a}S_1 = \mathbf{e}S_0 \cap \mathbf{e}S_1\} \\
& (\exists T, b : T \in \mathbf{p}S_0 \wedge tb \in p_0 \,\mathbf{w}\, p_1 : \{a, b\} \subseteq \mathbf{a}T \setminus \mathbf{e}S_0) \\
& \quad \vee (\exists T, b : T \in \mathbf{p}S_1 \wedge tb \in p_0 \,\mathbf{w}\, p_1 : \{a, b\} \subseteq \mathbf{a}T \setminus \mathbf{e}S_1) \\
\Rightarrow & \quad \{\mathbf{a}S_0 \cap \mathbf{a}S_1 = \mathbf{e}S_0 \cap \mathbf{e}S_1, \text{ predicate calculus}\} \\
& (\exists T, b : T \in \mathbf{p}S_0 \cup \mathbf{p}S_1 \wedge tb \in p_0 \,\mathbf{w}\, p_1 : \{a, b\} \subseteq \mathbf{a}T \setminus (\mathbf{e}S_0 \cup \mathbf{e}S_1)).
\end{array}
$$

□

The following two lemmata express that set RD is closed under parallel composition and projection.

Lemma 3.2.5

For systems S_0 and S_1 of RD

$$S_0 \parallel S_1 \in RD.$$

Proof

We derive

$$
\begin{array}{ll}
& p \in pot.(S_0 \parallel S_1) \\
\Rightarrow & \quad \{\text{Corollary 3.1.11}\} \\
& (\exists p_0, p_1 : p_0 \in pot.S_0 \wedge p_1 \in pot.S_1 : p = p_0 \,\mathbf{w}\, p_1) \\
\Leftrightarrow & \quad \{S_0 \in RD \text{ and } S_1 \in RD, \text{ Property 3.2.4}\} \\
& (\exists p_0, p_1 : p_0 \in pot.S_0 \wedge p_1 \in pot.S_1 \wedge p = p_0 \,\mathbf{w}\, p_1
\end{array}
$$

$$
\begin{array}{l}
\quad : (\forall t, a : t \in p_0 \,\mathbf{w}\, p_1 \wedge ta \in \mathbf{t}(S_0 \parallel S_1) \setminus p_0 \,\mathbf{w}\, p_1 \\
\qquad : (\exists T, b : T \in \mathbf{p}S_0 \cup \mathbf{p}S_1 \wedge tb \in p_0 \,\mathbf{w}\, p_1 \\
\qquad\quad : \{a, b\} \subseteq \mathbf{a}T \setminus (\mathbf{e}S_0 \cup \mathbf{e}S_1)))) \\
\Rightarrow \quad \{\text{predicate calculus}\} \\
(\forall t, a : t \in p \wedge ta \in \mathbf{t}(S_0 \parallel S_1) \setminus p \\
\qquad : (\exists T, b : T \in \mathbf{p}S_0 \cup \mathbf{p}S_1 \wedge tb \in p : \{a, b\} \subseteq \mathbf{a}T \setminus (\mathbf{e}S_0 \cup \mathbf{e}S_1))).
\end{array}
$$

□

Lemma 3.2.6

For system S of RD and alphabet A

$$S\!\uparrow\!A \in RD.$$

Proof

We derive

$$
\begin{array}{l}
S \in RD \\
\Leftrightarrow \quad \{\text{Definition 3.2.3}\} \\
(\forall t, p, a : p \in pot.S \wedge t \in p \wedge ta \in \mathbf{t}S \setminus p \\
\qquad : (\exists T, b : T \in \mathbf{p}S \wedge tb \in p : \{a, b\} \subseteq \mathbf{a}T \setminus \mathbf{e}S)) \\
\Rightarrow \quad \{pot.S = pot.(S\!\uparrow\!A),\ a \notin \mathbf{e}S \Rightarrow a \notin \mathbf{e}S \cap A,\ \mathbf{e}(S\!\uparrow\!A) = \mathbf{e}S \cap A\} \\
(\forall t, p, a : p \in pot.(S\!\uparrow\!A) \wedge t \in p \wedge ta \in \mathbf{t}S \setminus p \\
\qquad : (\exists T, b : T \in \mathbf{p}S \wedge tb \in p : \{a, b\} \subseteq \mathbf{a}T \setminus \mathbf{e}(S\!\uparrow\!A))) \\
\Leftrightarrow \quad \{\text{Definition 3.2.3}\} \\
S\!\uparrow\!A \in RD.
\end{array}
$$

□

A very satisfactory consequence of considering only systems of RD is the fact that failures become unnecessary in the abstract model. In Section 3.3 we prove that the compositional abstract model introduced below is consistent with the testing relations of Chapter 2.

This concludes our explanation for the restriction of the system domain and for the choice of the abstract model. In the sequel, formal definitions and proofs of compositionality with respect to parallel composition and projection are given.

$$
\begin{array}{l}
\text{Let } \Sigma = \{(T, \mathcal{X}) \mid T \in PR \wedge \mathbf{t}T = (\cup X : X \in \mathcal{X} : X) \\
\qquad\qquad \wedge (\forall X : X \in \mathcal{X} : \langle \mathbf{a}T, X\rangle \text{ is a conservative process})\}.
\end{array}
$$

For system S of RD we have then that $(\mathbf{pr}S, \mathbf{c}S)$ is an element of Σ.
We use the following abbreviation.

Definition 3.2.7

Let $(T, \mathcal{X}) \in \Sigma$ and $(U, \mathcal{Y}) \in \Sigma$. For $X \in \mathcal{X}$ and $Y \in \mathcal{Y}$

$$X \mathbf{w} Y = \mathbf{t}(\langle \mathbf{a}T, X\rangle \mathbf{w} \langle \mathbf{a}U, Y\rangle).$$

□

On systems of RD, the semantic function $\mathcal{A}$ is defined.

Definition 3.2.8

Let S be a system belonging to RD. We define the function $\mathcal{A} : RD \to \Sigma$ by

$$\mathcal{A}[\![S]\!] = (\mathbf{pr}S, \mathbf{c}S).$$

Then $\mathcal{A}[\![S]\!]$ is called the abstract model of S.

□

To see why $\mathcal{A}$ cannot be defined on the whole system domain, consider again systems S_0 and S_1 from Example 2.2.7. They have equal external processes ($\mathbf{pr}(a \mid b)$) and the same computations ($\{\varepsilon, a\}$ and $\{\varepsilon, b\}$). As a consequence, they would be mapped by $\mathcal{A}$ on the same object of Σ. However, their external behaviour is different, which can be verified by applying the **sat** relation.
The notion of acceptances (see the characterization given by Corollary 3.2.2) is generalized to elements of Σ in the obvious way.

Definition 3.2.9

For $(T, \mathcal{X}) \in \Sigma$

$$\mathbf{ac}(T, \mathcal{X}) = \{(t, L) \mid t \in \mathbf{t}T \wedge L \subseteq \mathbf{a}T^* \\ \wedge (\forall X : X \in \mathcal{X} \wedge t \in X : \{v \mid tv \in X\} \cap L \neq \emptyset)\}.$$

□

Furthermore, on pairs of elements of Σ, relation $\sqsupseteq$ is defined.

Definition 3.2.10

For $(T,\mathcal{X}) \in \Sigma$ and $(U,\mathcal{Y}) \in \Sigma$

$$(T,\mathcal{X}) \sqsupseteq (U,\mathcal{Y}) \;\Leftrightarrow\; T \subseteq U \wedge \mathbf{ac}(T,\mathcal{X}) \supseteq \mathbf{ac}(U,\mathcal{Y}).$$

□

We use $\sqsupseteq$ to emphasize the contribution of the acceptances. This symbol also agrees with the following interpretation of $\mathcal{A}[\![S_0]\!] \sqsupseteq \mathcal{A}[\![S_1]\!]$: from the point of view of an external observer, S_0 is at least as deterministic as S_1.

The above definition implies that relation $\sqsupseteq$ is reflexive and transitive. Observe that relation $\sqsupseteq$ is not antisymmetric (hence, not a partial order), that is, the following implication

$$\mathcal{A}[\![S_0]\!] \sqsupseteq \mathcal{A}[\![S_1]\!] \wedge \mathcal{A}[\![S_0]\!] \sqsubseteq \mathcal{A}[\![S_1]\!] \;\Rightarrow\; \mathcal{A}[\![S_0]\!] = \mathcal{A}[\![S_1]\!]$$

is not necessarily true. For instance, systems S_0 and S_1 of Example 2.4.9 do not satisfy this implication. Namely, we have $\mathbf{pr}S_0 = \mathbf{pr}S_1$ and $\mathbf{ac}S_0 = \mathbf{ac}S_1$, but $\mathbf{c}S_0 \neq \mathbf{c}S_1$.

For the elements of Σ we have the following lemma and two properties.

Lemma 3.2.11

Let $(T,\mathcal{X}) \in \Sigma$ and $(U,\mathcal{Y}) \in \Sigma$, such that $\mathbf{a}T = \mathbf{a}U$ and $\mathcal{X} = \mathcal{Y}$. Then

- $T = U$,
- $(T,\mathcal{X}) \sqsupseteq (U,\mathcal{Y}) \wedge (U,\mathcal{Y}) \sqsupseteq (T,\mathcal{X})$.

Proof

We derive

$$\begin{array}{ll} & \mathbf{t}T \\ = & \quad \{(T,\mathcal{X}) \in \Sigma\} \\ & (\cup X : X \in \mathcal{X} : X) \\ = & \quad \{\mathcal{X} = \mathcal{Y}\} \\ & (\cup X : X \in \mathcal{Y} : X) \\ = & \quad \{(U,\mathcal{Y}) \in \Sigma\} \\ & \mathbf{t}U. \end{array}$$

Hence, $T = U$.

Let $t \in \mathbf{t}U$ and $L \subseteq \mathbf{a}U^*$. We derive

$$
\begin{array}{ll}
& (t, L) \in \mathbf{ac}(U, \mathcal{Y}) \\
\Leftrightarrow & \{\text{Definition 3.2.9, } t \in \mathbf{t}U \text{ and } L \subseteq \mathbf{a}U^*\} \\
& (\forall X : X \in \mathcal{Y} \wedge t \in X : \{v \mid tv \in X\} \cap L \neq \emptyset) \\
\Leftrightarrow & \{\mathcal{X} = \mathcal{Y}\} \\
& (\forall X : X \in \mathcal{X} \wedge t \in X : \{v \mid tv \in X\} \cap L \neq \emptyset) \\
\Leftrightarrow & \{T = U, \text{ Definition 3.2.9}\} \\
& (t, L) \in \mathbf{ac}(T, \mathcal{X}).
\end{array}
$$

On account of the above derivations and Definition 3.2.10 can be concluded that

$$(T, \mathcal{X}) \sqsupseteq (U, \mathcal{Y}) \wedge (U, \mathcal{Y}) \sqsupseteq (T, \mathcal{X}).$$

□

Property 3.2.12

For $(T, \mathcal{X}) \in \Sigma$ and $(U, \mathcal{Y}) \in \Sigma$

$$(T \mathbf{w} U, \{X \mathbf{w} Y \mid X \in \mathcal{X} \wedge Y \in \mathcal{Y}\}) \in \Sigma.$$

□

Property 3.2.13

For alphabet A and $(T, \mathcal{X}) \in \Sigma$

$$(T\restriction A, \mathcal{X}\restriction A) \in \Sigma.$$

□

We define two operators on Σ: binary operator $\|$ and unary operator $\restriction A$, for alphabet A.

Definition 3.2.14

Let $(T, \mathcal{X}) \in \Sigma$ and $(U, \mathcal{Y}) \in \Sigma$. Operator $\|: \Sigma \times \Sigma \to \Sigma$ is defined by

$$(T, \mathcal{X}) \,\|\, (U, \mathcal{Y}) = (T \mathbf{w} U, \{X \mathbf{w} Y \mid X \in \mathcal{X} \wedge Y \in \mathcal{Y}\}).$$

□

Definition 3.2.15

Let A be an alphabet and $(T, \mathcal{X}) \in \Sigma$. Operator $\restriction A : \Sigma \to \Sigma$ is defined by

$$(T, \mathcal{X})\restriction A = (T\restriction A, \mathcal{X}\restriction A).$$

□

For compositionality the equality of $\mathcal{A}[\![S_0 \parallel S_1]\!]$ and $\mathcal{A}[\![S_0]\!] \parallel \mathcal{A}[\![S_1]\!]$ is needed.
On account of Lemma 1.2.4 we already have

$$\mathbf{pr}(S_0 \parallel S_1) = \mathbf{pr}S_0 \mathbin{\mathbf{w}} \mathbf{pr}S_1.$$

We are left with the proof obligation concerning the computation part. For the actual proof an intermediate result is necessary.

Lemma 3.2.16

For systems S_0 and S_1 of RD

$$(\forall p_0, p_1 : p_0 \in pot.S_0 \wedge p_1 \in pot.S_1 : p_0 \mathbin{\mathbf{w}} p_1 \in pot.(S_0 \parallel S_1)).$$

Proof

Let $p_0 \in pot.S_0$ and $p_1 \in pot.S_1$. On account of Corollary 3.1.6 we have

$$p_0 \mathbin{\mathbf{w}} p_1 \in ppot.(S_0 \parallel S_1).$$

We prove that $p_0 \mathbin{\mathbf{w}} p_1$ is a maximal element of $ppot.(S_0 \parallel S_1)$.
Assume $t \in (\mathbf{a}S_0 \cup \mathbf{a}S_1)^*$ and $ta \notin p_0 \mathbin{\mathbf{w}} p_1$. We derive

$$\begin{array}{ll} & t \in p_0 \mathbin{\mathbf{w}} p_1 \wedge ta \in \mathbf{t}(S_0 \parallel S_1) \\ \Rightarrow & \{ta \notin p_0 \mathbin{\mathbf{w}} p_1, \text{ Property 3.2.4}\} \\ & (\exists T, b : T \in \mathbf{p}S_0 \cup \mathbf{p}S_1 \wedge tb \in p_0 \mathbin{\mathbf{w}} p_1 : \{a, b\} \subseteq \mathbf{a}T \setminus (\mathbf{e}S_0 \cup \mathbf{e}S_1)) \\ \Rightarrow & \{ta \notin p_0 \mathbin{\mathbf{w}} p_1, \text{ predicate calculus}\} \\ & (\exists T, u, b : T \in \mathbf{p}S_0 \cup \mathbf{p}S_1 \wedge \{a, b\} \subseteq \mathbf{a}T \wedge ub \in p_0 \mathbin{\mathbf{w}} p_1 \wedge u\restriction\mathbf{a}T = t\restriction\mathbf{a}T \\ & \qquad : a \neq b). \end{array}$$

Hence, we have

$$
\begin{array}{l}
p_0 \mathbf{w} p_1 \in ppot.(S_0 \parallel S_1) \\
\wedge \\
(\forall t, a : t \in p_0 \mathbf{w} p_1 \wedge ta \in \mathbf{t}(S_0 \parallel S_1) \setminus p_0 \mathbf{w} p_1 \\
\quad : (\exists T, u, b : T \in \mathbf{p}S_0 \cup \mathbf{p}S_1 \wedge \{a, b\} \subseteq \mathbf{a}T \wedge ub \in p_0 \mathbf{w} p_1 \wedge u\restriction \mathbf{a}T = t\restriction \mathbf{a}T \\
\qquad : a \neq b)).
\end{array}
$$

This gives, on account of Property 2.3.3,

$$p_0 \mathbf{w} p_1 \in pot.(S_0 \parallel S_1).$$

□

The following theorem shows that computations of $\mathcal{A}[\![S_0 \parallel S_1]\!]$ and $\mathcal{A}[\![S_0]\!] \parallel \mathcal{A}[\![S_1]\!]$ are equal.

Theorem 3.2.17

For systems S_0 and S_1 of RD

$$\mathbf{c}(S_0 \parallel S_1) = \{X_0 \mathbf{w} X_1 \mid X_0 \in \mathbf{c}S_0 \wedge X_1 \in \mathbf{c}S_1\}.$$

Proof

Assume for convenience that $\mathbf{a}S_0 \cap \mathbf{a}S_1 = \mathbf{e}S_0 \cap \mathbf{e}S_1$.

The inclusion $\subseteq$ is already proved by Lemma 3.1.15. Here a proof for the inclusion the other way around is supplied.

We derive

$$
\begin{array}{ll}
& X_0 \in \mathbf{c}S_0 \wedge X_1 \in \mathbf{c}S_1 \\
\Leftrightarrow & \{\text{Definition 3.2.1}\} \\
& (\exists p_0, p_1 : p_0 \in pot.S_0 \wedge p_1 \in pot.S_1 : p_0\restriction \mathbf{e}S_0 = X_0 \wedge p_1\restriction \mathbf{e}S_1 = X_1) \\
\Leftrightarrow & \{\text{Lemma 3.2.16}\} \\
& (\exists p_0, p_1 : p_0 \in pot.S_0 \wedge p_1 \in pot.S_1 \wedge p_0 \mathbf{w} p_1 \in pot.(S_0 \parallel S_1) \\
& \qquad : p_0\restriction \mathbf{e}S_0 = X_0 \wedge p_1\restriction \mathbf{e}S_1 = X_1) \\
\Rightarrow & \{\mathbf{w} \text{ is monotonic, } \mathbf{a}S_0 \cap \mathbf{a}S_1 = \mathbf{e}S_0 \cap \mathbf{e}S_1, \text{ Lemma 1.1.13}\} \\
& (\exists p_0, p_1 : p_0 \in pot.S_0 \wedge p_1 \in pot.S_1 \wedge p_0 \mathbf{w} p_1 \in pot.(S_0 \parallel S_1) \\
& \qquad : X_0 \mathbf{w} X_1 = (p_0 \mathbf{w} p_1)\restriction (\mathbf{e}S_0 \cup \mathbf{e}S_1)) \\
\Rightarrow & \{\text{predicate calculus}\} \\
& (\exists p : p \in pot.(S_0 \parallel S_1) : X_0 \mathbf{w} X_1 = p\restriction (\mathbf{e}S_0 \cup \mathbf{e}S_1))
\end{array}
$$

$\Leftrightarrow$ { Definition 3.2.1 }
$X_0 \mathbf{w} X_1 \in \mathbf{c}(S_0 \parallel S_1)$.

□

In order to achieve compositionality the equality of $\mathcal{A}[\![S\restriction A]\!]$ and $\mathcal{A}[\![S]\!]\restriction A$ has to be proved.
On account of Lemma 1.2.5 we already have

$$\mathbf{pr}(S\restriction A) = \mathbf{pr}S\restriction A.$$

Theorem 3.2.18 supplies the proof for the computation part.

Theorem 3.2.18

For system S and alphabet A

$$\mathbf{c}(S\restriction A) = \mathbf{c}S\restriction A.$$

Proof

We derive

$\mathbf{c}(S\restriction A)$
$=$ { Definition 3.2.1, $\mathbf{e}(S\restriction A) = \mathbf{e}S \cap A$ }
$pot.(S\restriction A)\restriction(\mathbf{e}S \cap A)$
$=$ { $pot.(S\restriction A) = pot.S$, Property 1.1.19 }
$(pot.S\restriction \mathbf{e}S)\restriction A$
$=$ { Definition 3.2.1 }
$\mathbf{c}S\restriction A$.

□

3.3 Relationship with the operational model

The abstract model of systems is introduced as an alternative to the testing relations of Chapter 2. It gives an easier method of establishing whether or not two systems of RD are related via **sat**, not in terms of how they can effect other systems but in terms of their own external processes and acceptances. This statement is formalized and proved by Theorem 3.3.4. First we supply some intermediate results.

The following lemma shows an alternative way of establishing the relationship between acceptance sets in the case of systems with equal external processes.

Lemma 3.3.1

For systems S_0 and S_1, such that $\mathbf{pr}S_0 = \mathbf{pr}S_1$,

$$\begin{array}{l} \mathbf{ac}S_0 \supseteq \mathbf{ac}S_1 \\ \Leftrightarrow \\ (\forall X_0, t : X_0 \in \mathbf{c}S_0 \wedge t \in X_0 \\ \qquad : (\exists X_1 : X_1 \in \mathbf{c}S_1 \wedge t \in X_1 : \{v \mid tv \in X_1\} \subseteq \{v \mid tv \in X_0\})). \end{array}$$

Proof

Let $X_0 \in \mathbf{c}S_0$ and $t \in X_0$. Define L by

$$L = \{v \mid (\exists X_1 : X_1 \in \mathbf{c}S_1 : tv \in X_1) \wedge tv \notin X_0\}.$$

We derive

$$\begin{array}{ll} & (\forall X_1 : X_1 \in \mathbf{c}S_1 \wedge t \in X_1 : \{v \mid tv \in X_1\} \not\subseteq \{v \mid tv \in X_0\}) \\ \Leftrightarrow & \{\text{set and predicate calculus}\} \\ & (\forall X_1 : X_1 \in \mathbf{c}S_1 \wedge t \in X_1 : (\exists v : tv \in X_1 : tv \notin X_0)) \\ \Rightarrow & \{\text{definition of } L,\ L \neq \emptyset,\ \mathbf{pr}S_0 = \mathbf{pr}S_1 \text{ implies } t \in \mathbf{t}(\mathbf{pr}S_1), \text{ definition of} \\ & \quad \text{acceptances}\} \\ & (t, L) \in \mathbf{ac}S_1 \wedge (t, L) \notin \mathbf{ac}S_0. \end{array}$$

Hence,

$$\begin{array}{l} \mathbf{ac}S_0 \supseteq \mathbf{ac}S_1 \\ \Rightarrow \\ (\forall X_0, t : X_0 \in \mathbf{c}S_0 \wedge t \in X_0 \\ \qquad : (\exists X_1 : X_1 \in \mathbf{c}S_1 \wedge t \in X_1 : \{v \mid tv \in X_1\} \subseteq \{v \mid tv \in X_0\})). \end{array}$$

Let $t \in \mathbf{t}(\mathbf{pr}S_1)$ and $L \subseteq \mathbf{e}S_1^*$. Assume

$$\begin{array}{l} (\forall X_0, t : X_0 \in \mathbf{c}S_0 \wedge t \in X_0 \\ \qquad : (\exists X_1 : X_1 \in \mathbf{c}S_1 \wedge t \in X_1 : \{v \mid tv \in X_1\} \subseteq \{v \mid tv \in X_0\})). \end{array}$$

We derive

$$
\begin{array}{ll}
& (t, L) \in \mathbf{ac}S_1 \\
\Leftrightarrow & \{\text{definition of acceptances, } t \in \mathbf{t}(\mathbf{pr}S_1),\ L \subseteq \mathbf{e}S_1^*\} \\
& (\forall X_1 : X_1 \in \mathbf{c}S_1 \wedge t \in X_1 : \{v \mid tv \in X_1\} \cap L \neq \emptyset) \\
\Rightarrow & \{\text{assumption, set calculus}\} \\
& (\forall X_0 : X_0 \in \mathbf{c}S_0 \wedge t \in X_0 : \{v \mid tv \in X_0\} \cap L \neq \emptyset) \\
\Leftrightarrow & \{\text{definition of acceptances, } \mathbf{pr}S_0 = \mathbf{pr}S_1 \text{ implies } L \subseteq \mathbf{e}S_0^* \text{ and } t \in \mathbf{t}(\mathbf{pr}S_0)\} \\
& (t, L) \in \mathbf{ac}S_0.
\end{array}
$$

Hence,

$$
\begin{array}{l}
\quad \mathbf{ac}S_0 \supseteq \mathbf{ac}S_1 \\
\Leftarrow \\
\quad (\forall X_0, t : X_0 \in \mathbf{c}S_0 \wedge t \in X_0 \\
\qquad : (\exists X_1 : X_1 \in \mathbf{c}S_1 \wedge t \in X_1 : \{v \mid tv \in X_1\} \subseteq \{v \mid tv \in X_0\})).
\end{array}
$$

□

The following lemma facilitates the proof of Lemma 3.3.3, which states that for system R, systems S_0 and S_1 of RD such that $\mathbf{pr}S_0 = \mathbf{pr}S_1$,

$$\mathbf{ac}S_0 \supseteq \mathbf{ac}S_1 \ \Rightarrow\ \mathbf{ac}(R \parallel S_0) \supseteq \mathbf{ac}(R \parallel S_1).$$

Lemma 3.3.2

Let S_0 and S_1 be systems of RD, such that $\mathbf{pr}S_0 = \mathbf{pr}S_1$ and $\mathbf{ac}S_0 \supseteq \mathbf{ac}S_1$. Let, additionally, R be a system, $X_0 \in \mathbf{c}S_0$, and $X \in \mathbf{c}R$. Then

$$
\begin{array}{l}
\quad X \mathbf{\ w\ } X_0 \in \mathbf{c}(R \parallel S_0) \wedge t \in X \mathbf{\ w\ } X_0 \\
\Rightarrow \\
\quad (\exists X_1 : X_1 \in \mathbf{c}S_1 \wedge t \restriction \mathbf{e}S_1 \in X_1 \wedge X \mathbf{\ w\ } X_1 \in \mathbf{c}(R \parallel S_1) \\
\qquad : \{v \mid (t \restriction \mathbf{e}S_1)v \in X_1\} \subseteq \{v \mid (t \restriction \mathbf{e}S_0)v \in X_0\}).
\end{array}
$$

Proof

Assume for convenience that $\mathbf{a}S_0 \cap \mathbf{a}R = \mathbf{e}S_0 \cap \mathbf{e}R = \mathbf{a}S_1 \cap \mathbf{a}R$.

Let $X \mathbf{\ w\ } X_0 \in \mathbf{c}(R \parallel S_0)$ and $t \in X \mathbf{\ w\ } X_0$.

On account of Definition 3.2.1 and Corollary 3.1.13, we have

$$
\begin{array}{l}
\quad X \mathbf{w} X_0 \in \mathbf{c}(R \parallel S_0) \\
\Leftrightarrow \\
\quad (\exists p_0, p : p_0 \in pot.S_0 \wedge p_0 \lceil \mathbf{e} S_0 = X_0 \wedge p \in pot.R \wedge p \lceil \mathbf{e} R = X \\
\qquad : p \mathbf{w} p_0 \in pot.(R \parallel S_0)).
\end{array}
$$

Let $p_0 \in pot.S_0$ and $p \in pot.R$, such that $p_0 \lceil \mathbf{e} S_0 = X_0$, $p \lceil \mathbf{e} R = X$, and $p \mathbf{w} p_0$ belongs to $pot.(R \parallel S_0)$.
Furthermore,

$$
\begin{array}{ll}
& t \in X \mathbf{w} X_0 \\
\Rightarrow & \{ \text{definition of } \mathbf{w} \} \\
& t \lceil \mathbf{e} S_0 \in X_0 \\
\Rightarrow & \{ \text{assumption} \} \\
& (\exists X_1 : X_1 \in \mathbf{c} S_1 \wedge t \lceil \mathbf{e} S_1 \in X_1 : \{v \mid (t \lceil \mathbf{e} S_1) v \in X_1\} \subseteq \{v \mid (t \lceil \mathbf{e} S_0) v \in X_0\}).
\end{array}
$$

The set of all X_1's that satisfy the above requirements is called $\mathcal{X}$. Because $\mathbf{pr} S_0$ is equal to $\mathbf{pr} S_1$, $X \mathbf{w} X_0$ is a computation of $R \parallel S_0$, and both systems (S_0 and S_1) belong to RD, it is possible to find in $\mathcal{X}$ a (minimal) computation that matches X with respect to choices between common actions (action of $\mathbf{e} R \cap \mathbf{e} S_1$). Let X_1 be such a computation, which is formally expressed by

$$
(\forall Y : Y \in \mathcal{X} \wedge X_1 \not\subseteq Y : X \mathbf{w} X_1 \not\subseteq X \mathbf{w} Y).
$$

By the definition of computations we have then

$$
(\exists p_1 : p_1 \in pot.S_1 : p_1 \lceil \mathbf{e} S_1 = X_1).
$$

Let p_1 be as specified by the above predicate. In the sequel we prove that $p \mathbf{w} p_1$ belongs to $pot.(R \parallel S_1)$ by applying Property 2.3.3. We already know that $p \mathbf{w} p_1$ is a po-trace of $R \parallel S_1$.
Let $ua \notin p \mathbf{w} p_1$ and $u \in (\mathbf{a} R \cup \mathbf{a} S_1)^*$. We derive

$$
\begin{array}{ll}
& u \in p \mathbf{w} p_1 \wedge ua \in \mathbf{t}(R \parallel S_1) \\
\Rightarrow & \{ ua \notin p \mathbf{w} p_1 \text{ and } u \in (\mathbf{a} R \cup \mathbf{a} S_1)^*, \text{ definitions of } \mathbf{w} \text{ and } \lceil \} \\
& u \lceil \mathbf{a} R \in p \wedge u \lceil \mathbf{a} S_1 \in p_1 \wedge \\
& ((a \in \mathbf{a} S_1 \setminus \mathbf{a} R \wedge (u \lceil \mathbf{a} S_1) a \in \mathbf{t} S_1 \setminus p_1) \vee \\
& \ (a \in \mathbf{a} S_1 \cap \mathbf{a} R \wedge (u \lceil \mathbf{a} S_1) a \in \mathbf{t} S_1 \setminus p_1 \wedge (u \lceil \mathbf{a} R) a \in \mathbf{t} R) \vee \\
& \ (a \in \mathbf{a} S_1 \cap \mathbf{a} R \wedge (u \lceil \mathbf{a} S_1) a \in \mathbf{t} S_1 \cap p_1 \wedge (u \lceil \mathbf{a} R) a \in \mathbf{t} R \setminus p) \vee \\
& \ (a \in \mathbf{a} R \setminus \mathbf{a} S_1 \wedge (u \lceil \mathbf{a} R) a \in \mathbf{t} R \setminus p))
\end{array}
$$

$\Leftrightarrow$ { $S_1 \in RD$ and $\mathbf{a}S_1 \cap \mathbf{a}R = \mathbf{e}S_1 \cap \mathbf{e}R$ imply that the second disjunct in the above formula is false }

$u\restriction \mathbf{a}R \in p \wedge u\restriction \mathbf{a}S_1 \in p_1 \wedge$
$((a \in \mathbf{a}S_1 \setminus \mathbf{a}R \wedge (u\restriction \mathbf{a}S_1)a \in \mathbf{t}S_1 \setminus p_1) \vee$
$(a \in \mathbf{a}S_1 \cap \mathbf{a}R \wedge (u\restriction \mathbf{a}S_1)a \in \mathbf{t}S_1 \cap p_1 \wedge (u\restriction \mathbf{a}R)a \in \mathbf{t}R \setminus p) \vee$
$(a \in \mathbf{a}R \setminus \mathbf{a}S_1 \wedge (u\restriction \mathbf{a}R)a \in \mathbf{t}R \setminus p))$

$\Rightarrow$ { $S_1 \in RD$ and $p_1 \in pot.S_1$ }

$u\restriction \mathbf{a}R \in p \wedge u\restriction \mathbf{a}S_1 \in p_1 \wedge$
$((\exists T, b : T \in \mathbf{p}S_1 \wedge (u\restriction \mathbf{a}S_1)b \in p_1 : \{a, b\} \subseteq \mathbf{a}T \setminus \mathbf{e}S_1) \vee$
$(a \in \mathbf{a}S_1 \cap \mathbf{a}R \wedge (u\restriction \mathbf{a}S_1)a \in \mathbf{t}S_1 \cap p_1 \wedge (u\restriction \mathbf{a}R)a \in \mathbf{t}R \setminus p) \vee$
$(a \in \mathbf{a}R \setminus \mathbf{a}S_1 \wedge (u\restriction \mathbf{a}R)a \in \mathbf{t}R \setminus p))$

$\Rightarrow$ { $u \in (\mathbf{a}R \cup \mathbf{a}S_1)^*$, $\mathbf{a}S_1 \cap \mathbf{a}R = \mathbf{e}S_1 \cap \mathbf{e}R$, definition of $\mathbf{w}$, $p \in pot.R$, Property 2.3.3 }

$(\exists T, b : T \in \mathbf{p}S_1 \wedge ub \in p \ \mathbf{w}\ p_1 : \{a, b\} \subseteq \mathbf{a}T \setminus \mathbf{e}S_1) \vee$
$(u\restriction \mathbf{a}R \in p \wedge u\restriction \mathbf{a}S_1 \in p_1 \wedge$
$((a \in \mathbf{a}S_1 \cap \mathbf{a}R \wedge (u\restriction \mathbf{a}S_1)a \in \mathbf{t}S_1 \cap p_1) \vee a \in \mathbf{a}R \setminus \mathbf{a}S_1) \wedge$
$(\exists T, u', b : T \in \mathbf{p}R \wedge \{a, b\} \subseteq \mathbf{a}T \wedge u'b \in p \wedge u'\restriction \mathbf{a}T = u\restriction \mathbf{a}T : a \neq b))$

$\Rightarrow$ { $\mathbf{ac}S_0 \supseteq \mathbf{ac}S_1$, $\mathbf{pr}S_0 = \mathbf{pr}S_1$, assumptions about p_1 and p, properties of computations and of conservative processes }

$(\exists T, u', b : T \in \mathbf{p}S_1 \wedge \{a, b\} \subseteq \mathbf{a}T \wedge u'b \in p \ \mathbf{w}\ p_1 \wedge u'\restriction \mathbf{a}T = u\restriction \mathbf{a}T : a \neq b) \vee$
$(\exists T, u', b : T \in \mathbf{p}R \wedge \{a, b\} \subseteq \mathbf{a}T \wedge u'b \in p \ \mathbf{w}\ p_1 \wedge u'\restriction \mathbf{a}T = u\restriction \mathbf{a}T : a \neq b)$

$\Leftrightarrow$ { predicate calculus }

$(\exists T, u', b : T \in \mathbf{p}R \cup \mathbf{p}S_1 \wedge \{a, b\} \subseteq \mathbf{a}T \wedge u'b \in p \ \mathbf{w}\ p_1 \wedge u'\restriction \mathbf{a}T = u\restriction \mathbf{a}T$
$: a \neq b)$.

By Corollary 3.1.6 and Property 2.3.3, the above derivation proves that $p \ \mathbf{w}\ p_1$ belongs to $pot.(R \parallel S_1)$. Then, on account of Definition 3.2.1, $X \ \mathbf{w}\ p_1\restriction \mathbf{e}S_1$ belongs to $\mathbf{c}(R \parallel S_1)$. Hence,

$(\exists X_1 : X_1 \in \mathbf{c}S_1 \wedge t\restriction \mathbf{e}S_1 \in X_1 \wedge X \ \mathbf{w}\ X_1 \in \mathbf{c}(R \parallel S_1)$
$: \{v \mid (t\restriction \mathbf{e}S_1)v \in X_1\} \subseteq \{v \mid (t\restriction \mathbf{e}S_0)v \in X_0\})$.

□

Lemma 3.3.3

For systems S_0 and S_1 of RD, such that $\mathbf{pr}S_0 = \mathbf{pr}S_1$, and for system R

$\mathbf{ac}S_0 \supseteq \mathbf{ac}S_1 \Rightarrow \mathbf{ac}(R \parallel S_0) \supseteq \mathbf{ac}(R \parallel S_1)$.

Proof

By Lemma 3.3.1 we have

$$\begin{array}{ll} & \mathbf{ac}S_0 \supseteq \mathbf{ac}S_1 \\ \Leftrightarrow & \\ & (\forall X_0, t : X_0 \in \mathbf{c}S_0 \wedge t \in X_0 \\ & \qquad : (\exists X_1 : X_1 \in \mathbf{c}S_1 \wedge t \in X_1 : \{v \mid tv \in X_1\} \subseteq \{v \mid tv \in X_0\})). \end{array}$$

Assume

$$\begin{array}{l} (\forall X_0, t : X_0 \in \mathbf{c}S_0 \wedge t \in X_0 \\ \qquad : (\exists X_1 : X_1 \in \mathbf{c}S_1 \wedge t \in X_1 : \{v \mid tv \in X_1\} \subseteq \{v \mid tv \in X_0\})). \end{array}$$

Let $X_0 \in \mathbf{c}S_0$ and $X \in \mathbf{c}R$. We derive

$$\begin{array}{ll} & X \mathbf{\ w\ } X_0 \in \mathbf{c}(R \parallel S_0) \wedge t \in X \mathbf{\ w\ } X_0 \\ \Rightarrow & \{\text{Lemma 3.3.2}\} \\ & (\exists X_1 : X_1 \in \mathbf{c}S_1 \wedge t\restriction \mathbf{e}S_1 \in X_1 \wedge X \mathbf{\ w\ } X_1 \in \mathbf{c}(R \parallel S_1) \\ & \qquad : \{v \mid (t\restriction \mathbf{e}S_1)v \in X_1\} \subseteq \{v \mid (t\restriction \mathbf{e}S_0)v \in X_0\}) \\ & \wedge t \in X \mathbf{\ w\ } X_0 \\ \Leftrightarrow & \{\text{definition of } \mathbf{w}\} \\ & (\exists X_1 : X_1 \in \mathbf{c}S_1 \wedge t\restriction \mathbf{e}S_1 \in X_1 \wedge X \mathbf{\ w\ } X_1 \in \mathbf{c}(R \parallel S_1) \\ & \qquad : \{v \mid (t\restriction \mathbf{e}S_1)v \in X_1\} \subseteq \{v \mid (t\restriction \mathbf{e}S_0)v \in X_0\}) \\ & \wedge t \in (\mathbf{e}S_0 \cup \mathbf{e}R)^* \wedge t\restriction \mathbf{e}R \in X \wedge t\restriction \mathbf{e}S_0 \in X_0 \\ \Rightarrow & \{\text{predicate calculus, } \mathbf{pr}S_0 = \mathbf{pr}S_1, \text{ definition of } \mathbf{w}, \mathbf{w} \text{ is monotonic}\} \\ & (\exists X_1 : X_1 \in \mathbf{c}S_1 \wedge t \in X \mathbf{\ w\ } X_1 \wedge X \mathbf{\ w\ } X_1 \in \mathbf{c}(R \parallel S_1) \\ & \qquad : \{v \mid tv \in X \mathbf{\ w\ } X_1\} \subseteq \{v \mid tv \in X \mathbf{\ w\ } X_0\}) \\ \Rightarrow & \{\text{predicate calculus}\} \\ & (\exists X_1 : X_1 \in \mathbf{c}(R \parallel S_1) \wedge t \in X_1 : \{v \mid tv \in X_1\} \subseteq \{v \mid tv \in X \mathbf{\ w\ } X_0\}). \end{array}$$

Furthermore,

$$\begin{array}{ll} & \mathbf{ac}S_0 \supseteq \mathbf{ac}S_1 \\ \Leftrightarrow & \{\text{Lemma 3.3.1}\} \\ & (\forall X_0, t : X_0 \in \mathbf{c}S_0 \wedge t \in X_0 \\ & \qquad : (\exists X_1 : X_1 \in \mathbf{c}S_1 \wedge t \in X_1 : \{v \mid tv \in X_1\} \subseteq \{v \mid tv \in X_0\})) \\ \Rightarrow & \{\text{Lemma 3.1.15, the above derivation}\} \\ & (\forall X_0, t : X_0 \in \mathbf{c}(R \parallel S_0) \wedge t \in X_0 \end{array}$$

$$: (\exists X_1 : X_1 \in \mathbf{c}(R \parallel S_1) \wedge t \in X_1 : \{v \mid tv \in X_1\} \subseteq \{v \mid tv \in X_0\}))$$

$\Leftrightarrow$ { Lemma 3.3.1 }

$\mathbf{ac}(R \parallel S_0) \supseteq \mathbf{ac}(R \parallel S_1)$.

□

The following theorem shows that verification of systems in the operational model of Chapter 2 (by means of **sat** relation) gives the same results as in the abstract model (i.e., **sat** is consistent with $\sqsupseteq$).

Theorem 3.3.4

For systems S_0 and S_1 of RD

$$S_0 \textbf{ sat } S_1 \Leftrightarrow \mathcal{A}[\![S_0]\!] \sqsupseteq \mathcal{A}[\![S_1]\!].$$

Proof

We derive

$S_0 \textbf{ sat } S_1$

$\Leftrightarrow$ { Definition 2.4.4 }

$S_0 \textbf{ psat } S_1 \wedge S_0 \textbf{ gsat } S_1$

$\Leftrightarrow$ { Lemma 2.4.2, Definition 2.4.3 }

$\mathbf{pr}S_0 \subseteq \mathbf{pr}S_1 \wedge (\forall R : R \in SY : \mathbf{ac}(R \parallel S_0) \supseteq \mathbf{ac}(R \parallel S_1))$

$\Rightarrow$ { predicate calculus, $\langle \emptyset, \emptyset \rangle \in SY$ }

$\mathbf{pr}S_0 \subseteq \mathbf{pr}S_1 \wedge \mathbf{ac}(\langle \emptyset, \emptyset \rangle \parallel S_0) \supseteq \mathbf{ac}(\langle \emptyset, \emptyset \rangle \parallel S_1)$

$\Leftrightarrow$ { Property 1.2.2 }

$\mathbf{pr}S_0 \subseteq \mathbf{pr}S_1 \wedge \mathbf{ac}S_0 \supseteq \mathbf{ac}S_1$

$\Leftrightarrow$ { Definitions 3.2.8 and 3.2.10 }

$\mathcal{A}[\![S_0]\!] \sqsupseteq \mathcal{A}[\![S_1]\!]$.

For the implication the other way around we derive

$\mathcal{A}[\![S_0]\!] \sqsupseteq \mathcal{A}[\![S_1]\!]$

$\Leftrightarrow$ { Definitions 3.2.8 and 3.2.10 }

$\mathbf{pr}S_0 \subseteq \mathbf{pr}S_1 \wedge \mathbf{ac}S_0 \supseteq \mathbf{ac}S_1$

$\Leftrightarrow$ { Property 2.2.5 }

$\mathbf{pr}S_0 = \mathbf{pr}S_1 \wedge \mathbf{ac}S_0 \supseteq \mathbf{ac}S_1$

$\Rightarrow$ { Lemma 3.3.3, S_0 and S_1 are elements of RD }
$\mathbf{pr}S_0 = \mathbf{pr}S_1 \wedge (\forall R : R \in SY : \mathbf{ac}(R \parallel S_0) \supseteq \mathbf{ac}(R \parallel S_1))$
$\Leftrightarrow$ { Lemma 2.4.2, Definition 2.4.3 }
S_0 **psat** $S_1 \wedge S_0$ **gsat** S_1
$\Leftrightarrow$ { Definition 2.4.4 }
S_0 **sat** S_1.

□

Since relation $\sqsupseteq$ is not antisymmetric, as mentioned in the previous section, the abstract model is not fully abstract ([Mi0] and [Hen]) with respect to the operational model (to the **sat** relation). Another abstract model, similar to the above one — in which computations are replaced by acceptances, is fully abstract to the operational model. Whether it would also be compositional, however, is still an unsolved problem.

4 An application

In this chapter, we use the abstract model of Chapter 3 to discuss communication behaviours of Tangram programs ([Be2]). Tangram is a simple CSP-based ([Hoa]) programming language for describing VLSI designs. A more extensive description of Tangram can be found in [Be2]. Tangram programs can be automatically translated into VLSI circuits ([Be1]). In Section 4.1, an intermediate implementation of (a subset of) Tangram programs by means of networks of handshake processes ([Be0]) is described. A semantics of Tangram programs based on the abstract model of Section 3.2 is defined in Section 4.2. A formal proof of the relationship between the semantics of Tangram and its implementation is presented. Section 4.3 consists of a few examples of equivalent and different programs.

4.1 C-Tangram

In this section, a subset of Tangram is described. Only undirected communication actions without value passing are considered. This restricted version of Tangram is called in the sequel c-Tangram. In addition we introduce a restriction on the names of communication actions. To explain this restriction, we first discuss the structure of the universe Ω of symbols ([Zwa]). We define Ω_s to be a set of *simple* symbols. Furthermore, for $n > 0$, the set Ω_s^n is defined to be the set of n-tuples of symbols from Ω_s. Our universe of symbols, Ω, is then defined by

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

The elements of Ω that are not in Ω_s are called *compound* symbols. For $n > 0$, the n-tuple $(a_0, a_1, \ldots, a_{n-1}) \in \Omega$ is denoted by $a_0 \cdot a_1 \cdot \ldots \cdot a_{n-1}$. In particular, (a) is denoted by a.

In c-Tangram programs only simple symbols, like a and b, are used.

Furthermore, we define

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

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

An essential element in Tangram programs is the *command*. We give an inductive definition of the type of commands that are allowed in our restricted version.

- a is a command, for $a \in \Omega_s \setminus \{\surd\}$,
- if C is a command, then so is $\#[C]$ — infinite repetition (it never stops),
- if N is a natural number ($N \geq 0$) and C is a command, then $\#N[C]$ — finite repetition (N times) — is a command as well,
- if C_0 and C_1 are commands, then so are
  - $C_0 \parallel C_1$ — parallel composition,
  - $C_0 ; C_1$ — sequential composition,
  - $C_0 \sqcap C_1$ — selection (nondeterministic choice).

The parallel composition operator has the highest priority and the selection operator has the lowest priority.

For finite concatenations of the same command an abbreviation is introduced. For command C and natural number $N \geq 1$

$$
\begin{aligned}
C^1 &= C, \\
C^{N+1} &= C ; C^N.
\end{aligned}
$$

The alphabet of command C, $\mathbf{a}C$, is the set of symbols from Ω_s that appear in C.

A c-Tangram program is a construct of the form $A.C$, where C is a command and $A \subseteq \mathbf{a}C$.

Commands are implemented by networks of basic components co-operating by two-phase handshake signalling ([Be2], [Be0]). For the purpose of this thesis these basic components are described by trace theory systems.

In the sequel, 0 and 1 are fixed, different from each other elements of $\Omega_s \setminus \{\surd\}$.

The *sequencer*, $\mathbf{seq}(a, b, c)$, defined by

$$\mathbf{seq}(a, b, c) = \langle \{0{\cdot}a, 1{\cdot}a, 0{\cdot}b, 1{\cdot}b, 0{\cdot}c, 1{\cdot}c\}, \{\mathbf{pr}((0{\cdot}a\,;\, 0{\cdot}b\,;\, 1{\cdot}b\,;\, 0{\cdot}c\,;\, 1{\cdot}c\,;\, 1{\cdot}a)^*)\}\rangle,$$

is a basic component with three ports a, b and c. After activation along a, first the communication along b (following the handshake protocol) and then along c takes place, after which the communication along a is completed. The component may subsequently be activated again. In a similar way the operational explanation of the remaining basic components can be given.

Besides the sequencer we have the following basic components

- the *repeater*,
  $\mathbf{rep}(a,b) = \langle\{0{\cdot}a, 1{\cdot}a, 0{\cdot}b, 1{\cdot}b\}, \{\mathbf{pr}(0{\cdot}a\,;\,(0{\cdot}b\,;\,1{\cdot}b)^*))\ \mathbf{w}\ \mathbf{stop}.\{1{\cdot}a\}\}\rangle$,
- the *N-repeater*, for $N \geq 0$,
  $\mathbf{rep}_N(a,b) = \langle\{0{\cdot}a, 1{\cdot}a, 0{\cdot}b, 1{\cdot}b\}, \{\mathbf{pr}((0{\cdot}a\,;\,(0{\cdot}b\,;\,1{\cdot}b)^N\,;\,1{\cdot}a)^*)\}\rangle$,
- the *mixer*,
  $\mathbf{mix}(a,b,c) = \langle\{0{\cdot}a, 1{\cdot}a, 0{\cdot}b, 1{\cdot}b, 0{\cdot}c, 1{\cdot}c\},$
  $\{\mathbf{pr}((0{\cdot}a\,;\,0{\cdot}c\,;\,1{\cdot}c\,;\,1{\cdot}a \mid 0{\cdot}b\,;\,0{\cdot}c\,;\,1{\cdot}c\,;\,1{\cdot}b)^*)\}\rangle$,
- the *connector*,
  $\mathbf{con}(a,b) = \langle\{0{\cdot}a, 1{\cdot}a, 0{\cdot}b, 1{\cdot}b\}, \{\mathbf{pr}((0{\cdot}a\,;\,0{\cdot}b\,;\,1{\cdot}b\,;\,1{\cdot}a)^*)\}\rangle$,
- the *selector*,
  $\mathbf{sel}(a,b,c) = \langle\{0{\cdot}a, 1{\cdot}a, 0{\cdot}b, 1{\cdot}b, 0{\cdot}c, 1{\cdot}c\}, \{\mathbf{pr}((0{\cdot}a\,;\,(0{\cdot}b\,;\,1{\cdot}b \mid 0{\cdot}c\,;\,1{\cdot}c)\,;\,1{\cdot}a)^*)\}\rangle$.

Note that the repeater can be activated only once, and that the only purpose of $\mathbf{stop}.\{1{\cdot}a\}$ in the definition of the repeater is to include $1{\cdot}a$ in the external alphabet (see Section 1.2).

Command C is implemented by the network of handshake processes represented by trace theory system $sys.C$. We give an inductive definition for $sys.C$. In the sequel, l and r are fixed, different from each other (and from 0 and 1) elements of $\Omega_* \setminus \{\surd\}$. For commands C, C_0, and C_1

$$
\begin{aligned}
sys.a &= \mathbf{con}(\surd, a)\\
sys.\#[C] &= \mathbf{rep}(\surd, l{\cdot}\surd) \parallel (\parallel a : a \in \mathbf{a}C : \mathbf{con}(l{\cdot}a, a)) \parallel l{\cdot}(sys.C)\\
sys.\#N[C] &= \mathbf{rep}_N(\surd, l{\cdot}\surd) \parallel (\parallel a : a \in \mathbf{a}C : \mathbf{con}(l{\cdot}a, a)) \parallel l{\cdot}(sys.C)\\
sys.(C_0\,;\,C_1) &= \mathbf{seq}(\surd, l{\cdot}\surd, r{\cdot}\surd) \parallel (\parallel a : a \in \mathbf{a}C_0 \cap \mathbf{a}C_1 : \mathbf{mix}(l{\cdot}a, r{\cdot}a, a))\\
&\qquad \parallel (\parallel a : a \in \mathbf{a}C_0 \setminus \mathbf{a}C_1 : \mathbf{con}(l{\cdot}a, a))\\
&\qquad \parallel (\parallel a : a \in \mathbf{a}C_1 \setminus \mathbf{a}C_0 : \mathbf{con}(r{\cdot}a, a)) \parallel l{\cdot}(sys.C_0) \parallel r{\cdot}(sys.C_1)\\
sys.(C_0 \parallel C_1) &= \mathbf{con}(\surd, l{\cdot}\surd) \parallel \mathbf{con}(\surd, r{\cdot}\surd) \parallel (\parallel a : a \in \mathbf{a}C_0 : \mathbf{con}(l{\cdot}a, a))\\
&\qquad \parallel (\parallel a : a \in \mathbf{a}C_1 : \mathbf{con}(r{\cdot}a, a)) \parallel l{\cdot}(sys.C_0) \parallel r{\cdot}(sys.C_1)\\
sys.(C_0 \sqcap C_1) &= \mathbf{sel}(\surd, l{\cdot}\surd, r{\cdot}\surd) \parallel (\parallel a : a \in \mathbf{a}C_0 \cap \mathbf{a}C_1 : \mathbf{mix}(l{\cdot}a, r{\cdot}a, a)
\end{aligned}
$$

$$\| (\| a : a \in \mathbf{a}C_0 \setminus \mathbf{a}C_1 : \mathbf{con}(l{\cdot}a, a))$$
$$\| (\| a : a \in \mathbf{a}C_1 \setminus \mathbf{a}C_0 : \mathbf{con}(r{\cdot}a, a)) \| l{\cdot}(sys.C_0) \| r{\cdot}(sys.C_1).$$

Moreover, we define

$$\mathbf{tick} = \langle \{0{\cdot}\surd, 1{\cdot}\surd\}, \{\mathbf{pr}(0{\cdot}\surd\,;\, 1{\cdot}\surd)\}\rangle$$

and, for $a \in \mathbf{a}C$,

$$\mathbf{trans}(a) = \langle \{0{\cdot}a, 1{\cdot}a, a\}, \{\mathbf{pr}((0{\cdot}a\,;\, a\,;\, 1{\cdot}a)^*)\}\rangle.$$

For $a \in \mathbf{a}C$, $\mathbf{trans}(a)$ serves the purpose of translating the two-phase handshake on a into the occurrence of a.

With program $P = A.C$, system $sys.P$ is associated, which is defined by

$$sys.P = (sys.C \| (\| a : a \in \mathbf{a}C : \mathbf{trans}(a)) \| \mathbf{tick})\restriction A.$$

Observe that $\mathbf{e}(sys.P) = A$.

The execution of program $A.C$ corresponds to a single execution of command C. This is modelled by the parallel composition of $sys.C$ and **tick**.

Note that for every program P, $\mathbf{p}(sys.P)$ forms a set of regular processes, and hence, $\mathbf{pr}(sys.P)$ is a regular process (Properties 1.1.9 and 1.1.6).

Let $P = A.C$. The remainder of this section is devoted to the proof of the fact that $sys.C$ and $sys.P$ both belong to RD (so that it makes sense to compare programs using acceptances only). The fact that $sys.C$ belongs to RD is also of importance for the next section.

The proof goes by induction on C.

Components $\mathbf{seq}(a,b,c)$, $\mathbf{rep}(a,b)$, $\mathbf{rep}_N(a,b)$, $\mathbf{con}(a,b)$, $\mathbf{trans}(a)$, and **tick** all have only one po-trace. Hence, they all belong to RD. Then, on account of Lemmata 3.2.5 and 3.2.6 it can be concluded that for program P with a command of the form: a, $\#[C]$, $\#N[C]$, or $C_0 \| C_1$

$$sys.C \in RD \qquad \text{and} \qquad sys.P \in RD.$$

Command a forms the basis for the induction; for the remaining commands we use $l{\cdot}(sys.C) \in RD$ (or $l{\cdot}(sys.C_0) \in RD$ and $r{\cdot}(sys.C_1) \in RD$) as induction hypothesis.

In the remaining two cases (sequential composition and nondeterministic choice) components $\mathbf{mix}(l{\cdot}a, r{\cdot}a, a)$ and $\mathbf{sel}(\surd, l{\cdot}\surd, r{\cdot}\surd)$ that do not belong to RD are used, so in this case Lemmata 3.2.5 and 3.2.6 alone do not help. However, in the case of sequential composition the choice present in **mix** is ruled out by **seq** — first the actions with names preceded by $l{\cdot}$ occur, after which occurrences of actions with names preceded

by $r\cdot$ are allowed. In the case of selection either symbols preceded by $l\cdot$ or symbols preceded by $r\cdot$ (and not both) are allowed, depending on the choices in **sel**. Because of the specific form of possible traces the conditions of Definition 3.2.3 are fulfilled. Hence, also for program P with a command of the form $C_0\,;\, C_1$ or $C_0 \sqcap C_1$

$$sys.C \in RD$$

and, on account of Lemmata 3.2.5 and 3.2.6 (**trans**(a) and **tick** belong to RD)

$$sys.P \in RD.$$

4.2 Semantics of c-Tangram

The commands of c-Tangram of the previous section are given meanings in the sequel. The semantics is partially based on the abstract model of Section 3.2 and partially on [Hoa]. From the latter the idea of successfully terminated computations is taken. This concept is crucial for the definition of the semantics of sequential composition. The computations that are either non-terminating (infinite) or unsuccessfully terminated are in the sequel called deadlocked computations. The computations that are successfully terminated are in the sequel called completed computations. In the (direct) semantics of Tangram presented in [Be0], which maps commands into handshake processes, $a \parallel b$ does not implement $a\,;\, b \sqcap b\,;\, a$. In our semantic model, $a \parallel b$ implements $a\,;\, b \sqcap b\,;\, a$.

Before we give the definition of the semantics of commands, we introduce some auxiliary notations and some new notions.

For sets X and X' of traces we define the concatenation of X and X', denoted as $X +\!\!+ X'$, by

$$X +\!\!+ X' = \{tu \mid t \in X \wedge u \in X'\}.$$

Observe that $+\!\!+$ is associative.

For set A of sets of traces we define A^i, for $i \geq 0$, by

$$\begin{aligned} A^0 &= \{\{\varepsilon\}\}, \\ A^{i+1} &= \{X +\!\!+ X' \mid X \in A^i \wedge X' \in A\}. \end{aligned}$$

Observe that, for $N \geq 1$, $\varnothing^N = \varnothing$.

Then $A^* = (\cup i : i \geq 0 : A^i)$ denotes the set of finite concatenations of elements of A. Furthermore, $A^\infty = \{X_0 +\!\!+ X_1 +\!\!+ \ldots \mid (\forall i : i \geq 0 : X_i \in A)\}$ denotes the set of infinite concatenations of elements of A . For the empty set we have $\varnothing^\infty = \varnothing$ and $\varnothing^* = \{\{\varepsilon\}\}$.

For set X of traces, $\mu.X$ is the set of maximal traces of X. Formally,

$$\mu.X = \{t \mid t \in X \wedge (\forall u : u \neq \varepsilon : tu \notin X)\}.$$

For set A of sets of traces we also define

$$\mu.A = \{\mu.X \mid X \in A\}$$

For subset A of simple symbols, $\delta.A = \{i{\cdot}a \mid a \in A \wedge (i = 0 \vee i = 1)\}$.
The set of c-Tangram commands is called *Com*. The set of triples $(A, \mathcal{X}, \mathcal{Y})$ such that

$$((A, (\cup X : X \in \mathcal{X} \cup \mathcal{Y} : X)), \mathcal{X} \cup \mathcal{Y}) \in \Sigma$$

is called Γ.
On Γ, two unary and three binary operators are defined.

Definition 4.2.1

Let $(A, \mathcal{X}, \mathcal{Y})$, $(A_0, \mathcal{X}_0, \mathcal{Y}_0)$, and $(A_1, \mathcal{X}_1, \mathcal{Y}_1)$ be elements of Γ.

- Operator $\# : \Gamma \to \Gamma$ is defined by

$$\begin{array}{l} \#(A, \mathcal{X}, \mathcal{Y}) \\ = \\ (A, \varnothing, \{pref.X \mid X \in (\mu.\mathcal{X})^{\infty}\} \cup \{pref.(X +\!\!+ Y) \mid X \in (\mu.\mathcal{X})^{*} \wedge Y \in \mathcal{Y}\}). \end{array}$$

- Let $N \geq 0$. Operator $\#N : \Gamma \to \Gamma$ is defined by

$$\begin{array}{l} \#N(A, \mathcal{X}, \mathcal{Y}) \\ = \\ (A, \{pref.X \mid X \in (\mu.\mathcal{X})^{N}\} \\ \quad , \{pref.(X +\!\!+ Y) \mid (\exists i : 0 \leq i < N : X \in (\mu.\mathcal{X})^{i}) \wedge Y \in \mathcal{Y}\}). \end{array}$$

- Operator $; : \Gamma \times \Gamma \to \Gamma$ is defined by

$$\begin{array}{l} (A_0, \mathcal{X}_0, \mathcal{Y}_0) ; (A_1, \mathcal{X}_1, \mathcal{Y}_1) \\ = \\ (A_0 \cup A_1, \{pref.(X_0 +\!\!+ X_1) \mid X_0 \in \mu.\mathcal{X}_0 \wedge X_1 \in \mu.\mathcal{X}_1\} \\ \qquad , \mathcal{Y}_0 \cup \{pref.(X_0 +\!\!+ Y_1) \mid X_0 \in \mu.\mathcal{X}_0 \wedge Y_1 \in \mathcal{Y}_1\}). \end{array}$$

- Operator $\| : \Gamma \times \Gamma \to \Gamma$ is defined by

$$
\begin{aligned}
&(A_0, \mathcal{X}_0, \mathcal{Y}_0) \parallel (A_1, \mathcal{X}_1, \mathcal{Y}_1) \\
=\ &(A_0 \cup A_1 \\
&\ ,\{X_0 \mathbf{w} X_1 \mid X_0 \in \mathcal{X}_0 \wedge X_1 \in \mathcal{X}_1 \wedge \\
&\qquad (\exists t : t \in X_0 \mathbf{w} X_1 : t\lceil A_0 \in \mu.X_0 \wedge t\lceil A_1 \in \mu.X_1)\} \\
&\ ,\{X_0 \mathbf{w} X_1 \mid (X_0 \in \mathcal{Y}_0 \wedge X_1 \in \mathcal{X}_1 \cup \mathcal{Y}_1) \vee (X_0 \in \mathcal{X}_0 \cup \mathcal{Y}_0 \wedge X_1 \in \mathcal{Y}_1) \\
&\qquad \vee (X_0 \in \mathcal{X}_0 \wedge X_1 \in \mathcal{X}_1 \wedge \\
&\qquad\quad (\forall t : t \in X_0 \mathbf{w} X_1 : t\lceil A_0 \notin \mu.X_0 \vee t\lceil A_1 \notin \mu.X_1))\}),
\end{aligned}
$$

where for $X_0 \in \mathcal{X}_0$ and $X_1 \in \mathcal{X}_1$

$$X_0 \mathbf{w} X_1 = \mathbf{t}(\langle A_0, X_0\rangle \mathbf{w} \langle A_1, X_1\rangle).$$

- Operator $\sqcap : \Gamma \times \Gamma \to \Gamma$ is defined by

$$(A_0, \mathcal{X}_0, \mathcal{Y}_0) \sqcap (A_1, \mathcal{X}_1, \mathcal{Y}_1) = (A_0 \cup A_1, \mathcal{X}_0 \cup \mathcal{X}_1, \mathcal{Y}_0 \cup \mathcal{Y}_1).$$

□

Assume $a \in \Omega_* \setminus \{\surd\}$ and $N \geq 0$. Let C, C_0, and C_1 be c-Tangram commands. The semantic function $\mathcal{M} : Com \to \Gamma$ is inductively defined by

- $\mathcal{M}[\![a]\!] = (\{0{\cdot}a, 1{\cdot}a\}, \{pref.\{0{\cdot}a\ 1{\cdot}a\}\}, \varnothing)$,
- $\mathcal{M}[\![\#[C]]\!] = \#\mathcal{M}[\![C]\!]$,
- $\mathcal{M}[\![\#N[C]]\!] = \#N\mathcal{M}[\![C]\!]$,
- $\mathcal{M}[\![C_0\,;\, C_1]\!] = \mathcal{M}[\![C_0]\!]\,;\, \mathcal{M}[\![C_1]\!]$,
- $\mathcal{M}[\![C_0 \parallel C_1]\!] = \mathcal{M}[\![C_0]\!] \parallel \mathcal{M}[\![C_1]\!]$,
- $\mathcal{M}[\![C_0 \sqcap C_1]\!] = \mathcal{M}[\![C_0]\!] \sqcap \mathcal{M}[\![C_1]\!]$.

For command C we have that the first element of the triple $\mathcal{M}[\![C]\!]$ equals $\delta.\mathbf{a}C$. The second element and the third element of that triple are called completed and deadlocked computations of C, respectively. The completed computations of C are denoted by $\mathbf{c}C$ and the deadlocked computations of C are denoted by $\mathbf{d}C$. Hence,

$$\mathcal{M}[\![C]\!] = (\delta.\mathbf{a}C, \mathbf{c}C, \mathbf{d}C).$$

It can be proved by induction that $\mathbf{c}C$ is a finite set of finite prefix-closed sets of traces, for all commands C. For $X \in \mathbf{c}C$ we have that X is fully characterized by the set of its maximal traces, i.e.,

$$X = \mathit{pref}.(\mu.X).$$

It can also be proved by induction that, for command C and $X \in \mathbf{c}C$, $\langle \delta.\mathbf{a}C, X\rangle$ is a conservative process.

It can be proved by induction that, for $X \in \mathbf{d}C$, $\langle \delta.\mathbf{a}C, X\rangle$ is a conservative process.

Properties 4.2.2 and 4.2.3 follow directly from Definition 4.2.1 and the definition of $\mathcal{M}$.

Property 4.2.2

Assume $a \in \Omega_a \setminus \{\surd\}$ and $N \geq 0$. Let C, C_0, and C_1 be c-Tangram commands. For the completed computations of commands we have

$$\begin{aligned}
&\mathbf{c}a = \{\mathit{pref}.\{0{\cdot}a\ 1{\cdot}a\}\},\\
&\mathbf{c}\#[C] = \varnothing,\\
&\mathbf{c}\#N[C] = \{\mathit{pref}.X \mid X \in (\mu.\mathbf{c}C)^N\},\\
&\mathbf{c}(C_0\,;\, C_1) = \{\mathit{pref}.(X_0 +\!\!+ X_1) \mid X_0 \in \mu.\mathbf{c}C_0 \wedge X_1 \in \mu.\mathbf{c}C_1\},\\
&\mathbf{c}(C_0 \parallel C_1) = \{X_0\ \mathbf{w}\ X_1 \mid X_0 \in \mathbf{c}C_0 \wedge X_1 \in \mathbf{c}C_1 \wedge\\
&\qquad\qquad (\exists t : t \in X_0\ \mathbf{w}\ X_1 : t\restriction\delta.\mathbf{a}C_0 \in \mu.X_0 \wedge t\restriction\delta.\mathbf{a}C_1 \in \mu.X_1)\},\\
&\mathbf{c}(C_0 \sqcap C_1) = \mathbf{c}C_0 \cup \mathbf{c}C_1,
\end{aligned}$$

□

Property 4.2.3

Assume $a \in \Omega_a \setminus \{\surd\}$ and $N \geq 0$. Let C, C_0, and C_1 be c-Tangram commands. For the deadlocked computations of commands we have

$$\begin{aligned}
&\mathbf{d}a = \varnothing,\\
&\mathbf{d}\#[C] = \{\mathit{pref}.X \mid X \in (\mu.\mathbf{c}C)^\infty\}\\
&\qquad\quad \cup \{\mathit{pref}.(X +\!\!+ X') \mid X \in (\mu.\mathbf{c}C)^* \wedge X' \in \mathbf{d}C\},\\
&\mathbf{d}\#N[C] = \{\mathit{pref}.(X +\!\!+ X') \mid (\exists i : 0 \leq i < N : X \in (\mu.\mathbf{c}C)^i) \wedge X' \in \mathbf{d}C\},\\
&\mathbf{d}(C_0\,;\, C_1) = \mathbf{d}C_0 \cup \{\mathit{pref}.(X_0 +\!\!+ X_1) \mid X_0 \in \mu.\mathbf{c}C_0 \wedge X_1 \in \mathbf{d}C_1\},\\
&\mathbf{d}(C_0 \parallel C_1) = \{X_0\ \mathbf{w}\ X_1\\
&\qquad\qquad |(X_0 \in \mathbf{d}C_0 \wedge X_1 \in \mathbf{c}C_1 \cup \mathbf{d}C_1) \vee (X_0 \in \mathbf{c}C_0 \cup \mathbf{d}C_0 \wedge X_1 \in \mathbf{d}C_1)\\
&\qquad\qquad \vee (X_0 \in \mathbf{c}C_0 \wedge X_1 \in \mathbf{c}C_1 \wedge
\end{aligned}$$

$$(\forall t : t \in X_0 \mathbf{w} X_1 : t\lceil\delta.\mathbf{a}C_0 \notin \mu.X_0 \vee t\lceil\delta.\mathbf{a}C_1 \notin \mu.X_1))\},$$
$$\mathbf{d}(C_0 \sqcap C_1) = \mathbf{d}C_0 \cup \mathbf{d}C_1.$$

□

Furthermore, we have the following two properties.

Property 4.2.4

For commands C_0 and C_1

$$\mu.\mathbf{c}(C_0\,;\, C_1) = \{X_0 + X_1 \mid X_0 \in \mu.\mathbf{c}C_0 \wedge X_1 \in \mu.\mathbf{c}C_1\}.$$

□

Property 4.2.5

Let C_0 and C_1 be commands, $X_0 \in \mathbf{c}C_0$, and $X_1 \in \mathbf{c}C_1$, such that

$$(\exists t' : t' \in X_0 \mathbf{w} X_1 : t'\lceil\delta.\mathbf{a}C_0 \in \mu.X_0 \wedge t'\lceil\delta.\mathbf{a}C_1 \in \mu.X_1).$$

Then for $t \in X_0 \mathbf{w} X_1$ we have

$$t \in \mu.(X_0 \mathbf{w} X_1) \;\Leftrightarrow\; t\lceil\delta.\mathbf{a}C_0 \in \mu.X_0 \wedge t\lceil\delta.\mathbf{a}C_1 \in \mu.X_1.$$

Proof

To prove $\Leftarrow$ we derive

$$\begin{array}{ll} & t\lceil\delta.\mathbf{a}C_0 \in \mu.X_0 \wedge t\lceil\delta.\mathbf{a}C_1 \in \mu.X_1 \\ \Leftrightarrow & \{\text{definition of } \mu.X\} \\ & t\lceil\delta.\mathbf{a}C_0 \in X_0 \wedge (\forall u : u \neq \varepsilon : (t\lceil\delta.\mathbf{a}C_0)u \notin X_0) \wedge \\ & t\lceil\delta.\mathbf{a}C_1 \in X_1 \wedge (\forall u : u \neq \varepsilon : (t\lceil\delta.\mathbf{a}C_1)u \notin X_1) \\ \Rightarrow & \{t \in X_0 \mathbf{w} X_1, \text{definition of } \mathbf{w}, \text{Property } 1.1.16\} \\ & (\forall u : u \neq \varepsilon : tu \notin X_0 \mathbf{w} X_1) \\ \Leftrightarrow & \{t \in X_0 \mathbf{w} X_1, \text{definition of } \mu.X\} \\ & t \in \mu.(X_0 \mathbf{w} X_1). \end{array}$$

Let $t' \in X_0 \mathbf{w} X_1$, such that $t'\lceil\delta.\mathbf{a}C_0 \in \mu.X_0$ and $t'\lceil\delta.\mathbf{a}C_1 \in \mu.X_1$. Then we also have $t' \in \mu.(X_0 \mathbf{w} X_1)$. To prove $\Rightarrow$ we derive

$t \in \mu.(X_0 \mathbf{w} X_1)$

$\Leftrightarrow$ {$t' \in \mu.(X_0 \mathbf{w} X_1)$, $\langle \delta.(\mathbf{a}C_0 \cup \mathbf{a}C_1), X_0 \mathbf{w} X_1 \rangle$ is a process, definition of $\mu.X$}

$t \in X_0 \mathbf{w} X_1 \wedge suc.t.\langle \delta.(\mathbf{a}C_0 \cup \mathbf{a}C_1), X_0 \mathbf{w} X_1 \rangle = \emptyset \wedge$
$t' \in X_0 \mathbf{w} X_1 \wedge suc.t'.\langle \delta.(\mathbf{a}C_0 \cup \mathbf{a}C_1), X_0 \mathbf{w} X_1 \rangle = \emptyset$

$\Rightarrow$ {definition of $\mathbf{w}$, $\langle \delta.(\mathbf{a}C_0 \cup \mathbf{a}C_1), X_0 \mathbf{w} X_1 \rangle$ is a conservative process, Property 1.3.6}

$t\restriction\delta.\mathbf{a}C_0 \in X_0 \wedge t'\restriction\delta.\mathbf{a}C_0 \in X_0 \wedge \flat.t = \flat.t'$

$\Rightarrow$ {for traces u, v and set A we have that $\flat.u = \flat.v \Rightarrow \flat.(u\restriction A) = \flat.(v\restriction A)$}

$t\restriction\delta.\mathbf{a}C_0 \in X_0 \wedge t'\restriction\delta.\mathbf{a}C_0 \in X_0 \wedge \flat.(t\restriction\delta.\mathbf{a}C_0) = \flat.(t'\restriction\delta.\mathbf{a}C_0)$

$\Rightarrow$ {$\langle \delta.\mathbf{a}C_0, X_0 \rangle$ is a conservative process, Property 1.3.6}

$[t\restriction\delta.\mathbf{a}C_0]_{\langle \delta.\mathbf{a}C_0, X_0 \rangle} = [t'\restriction\delta.\mathbf{a}C_0]_{\langle \delta.\mathbf{a}C_0, X_0 \rangle}$

$\Rightarrow$ {definition of $[u]_T$ and $\overset{T}{\sim}$}

$t\restriction\delta.\mathbf{a}C_0 \in X_0 \wedge suc.(t\restriction\delta.\mathbf{a}C_0).\langle \delta.\mathbf{a}C_0, X_0 \rangle = suc.(t'\restriction\delta.\mathbf{a}C_0).\langle \delta.\mathbf{a}C_0, X_0 \rangle \wedge$
$t'\restriction\delta.\mathbf{a}C_0 \in X_0$

$\Rightarrow$ {$t'\restriction\delta.\mathbf{a}C_0 \in \mu.X_0 \Rightarrow suc.(t'\restriction\delta.\mathbf{a}C_0).\langle \delta.\mathbf{a}C_0, X_0 \rangle = \emptyset$, predicate calculus}

$t\restriction\delta.\mathbf{a}C_0 \in X_0 \wedge suc.(t\restriction\delta.\mathbf{a}C_0).\langle \delta.\mathbf{a}C_0, X_0 \rangle = \emptyset$

$\Leftrightarrow$ {definition of $\mu.X$}

$t\restriction\delta.\mathbf{a}C_0 \in \mu.X_0$.

By the above derivation we proved

$$t \in \mu.(X_0 \mathbf{w} X_1) \Rightarrow t\restriction\delta.\mathbf{a}C_0 \in \mu.X_0.$$

On account of symmetry we also have

$$t \in \mu.(X_0 \mathbf{w} X_1) \Rightarrow t\restriction\delta.\mathbf{a}C_1 \in \mu.X_1.$$

□

Directly from Properties 4.2.2 and 4.2.3 we have

$\mathbf{c}(C_0 \sqcap C_1) = \mathbf{c}(C_1 \sqcap C_0)$,
$\mathbf{d}(C_0 \sqcap C_1) = \mathbf{d}(C_1 \sqcap C_0)$,
$\mathbf{c}((C_0 \sqcap C_1) \sqcap C_2) = \mathbf{c}(C_0 \sqcap (C_1 \sqcap C_2))$,
$\mathbf{d}((C_0 \sqcap C_1) \sqcap C_2) = \mathbf{d}(C_0 \sqcap (C_1 \sqcap C_2))$,

hence, operator $\sqcap$ on c-Tangram commands is commutative and associative.
The following lemma shows that the ; operator (for c-Tangram commands) is associative.

Lemma 4.2.6

For commands C_0, C_1, and C_2

$$\mathbf{c}((C_0\,;\,C_1)\,;\,C_2) = \mathbf{c}(C_0\,;\,(C_1\,;\,C_2))$$

and

$$\mathbf{d}((C_0\,;\,C_1)\,;\,C_2) = \mathbf{d}(C_0\,;\,(C_1\,;\,C_2)).$$

Proof

For completed computations we derive

$$\begin{array}{ll} & \mathbf{c}((C_0\,;\,C_1)\,;\,C_2) \\ = & \quad \{\text{Property 4.2.2}\} \\ & \{pref.(X \mathbin{+\!\!+} X_2) \mid X \in \mu.\mathbf{c}(C_0\,;\,C_1) \wedge X_2 \in \mu.\mathbf{c}C_2\} \\ = & \quad \{\text{Property 4.2.4, } \mathbin{+\!\!+} \text{ is associative}\} \\ & \{pref.(X_0 \mathbin{+\!\!+} X_1 \mathbin{+\!\!+} X_2) \mid X_0 \in \mu.\mathbf{c}C_0 \wedge X_1 \in \mu.\mathbf{c}C_1 \wedge X_2 \in \mu.\mathbf{c}C_2\} \\ = & \quad \{\text{Property 4.2.4, } \mathbin{+\!\!+} \text{ is associative}\} \\ & \{pref.(X_0 \mathbin{+\!\!+} X) \mid X_0 \in \mu.\mathbf{c}C_0 \wedge X \in \mu.\mathbf{c}(C_1\,;\,C_2)\} \\ = & \quad \{\text{Property 4.2.2}\} \\ & \mathbf{c}(C_0\,;\,(C_1\,;\,C_2)). \end{array}$$

For deadlocked computations we derive

$$\begin{array}{ll} & \mathbf{d}((C_0\,;\,C_1)\,;\,C_2) \\ = & \quad \{\text{Property 4.2.3}\} \\ & \mathbf{d}(C_0\,;\,C_1) \cup \{pref.(X \mathbin{+\!\!+} X_2) \mid X \in \mu.\mathbf{c}(C_0\,;\,C_1) \wedge X_2 \in \mathbf{d}C_2\} \\ = & \quad \{\text{Property 4.2.3, Property 4.2.4, } \mathbin{+\!\!+} \text{ is associative}\} \\ & \mathbf{d}C_0 \cup \{pref.(X_0 \mathbin{+\!\!+} X_1) \mid X_0 \in \mu.\mathbf{c}C_0 \wedge X_1 \in \mathbf{d}C_1\} \cup \\ & \{pref.(X_0 \mathbin{+\!\!+} X_1 \mathbin{+\!\!+} X_2) \mid X_0 \in \mu.\mathbf{c}C_0 \wedge X_1 \in \mu.\mathbf{c}C_1 \wedge X_2 \in \mathbf{d}C_2\} \\ = & \quad \{pref.(X \mathbin{+\!\!+} Y) = pref.(X \mathbin{+\!\!+} pref.Y), \text{ set calculus}\} \\ & \mathbf{d}C_0 \cup \\ & \{pref.(X_0 \mathbin{+\!\!+} X) \mid X_0 \in \mu.\mathbf{c}C_0 \wedge \\ & \qquad\qquad X \in \mathbf{d}C_1 \cup \{pref.(X_1 \mathbin{+\!\!+} X_2) \mid X_1 \in \mu.\mathbf{c}C_1 \wedge X_2 \in \mathbf{d}C_2\}\} \end{array}$$

$$
\begin{aligned}
&= \quad \{\text{Property 4.2.3}\} \\
&\quad \mathbf{d}C_0 \cup \{pref.(X_0 +\!\!+ X) \mid X_0 \in \mu.\mathbf{c}C_0 \wedge X \in \mathbf{d}(C_1\,;\, C_2)\} \\
&= \quad \{\text{Property 4.2.3}\} \\
&\quad \mathbf{d}(C_0\,;\,(C_1\,;\,C_2)).
\end{aligned}
$$

□

Since weaving is commutative (Property 1.1.8), it follows from Properties 4.2.2 and 4.2.3 that operator $\|$ on c-Tangram commands is commutative as well. The following lemma establishes the associativity of $\|$ (for c-Tangram commands).

Lemma 4.2.7

For commands C_0, C_1, and C_2

$$\mathbf{c}((C_0 \,\|\, C_1) \,\|\, C_2) = \mathbf{c}(C_0 \,\|\, (C_1 \,\|\, C_2))$$

and

$$\mathbf{d}((C_0 \,\|\, C_1) \,\|\, C_2) = \mathbf{d}(C_0 \,\|\, (C_1 \,\|\, C_2)).$$

Proof

For completed computations we derive

$$
\begin{aligned}
&\mathbf{c}((C_0 \,\|\, C_1) \,\|\, C_2) \\
&= \quad \{\text{Property 4.2.2}\} \\
&\quad \{X \;\mathbf{w}\; X_2 \mid X \in \mathbf{c}(C_0 \,\|\, C_1) \wedge X_2 \in \mathbf{c}C_2 \wedge \\
&\qquad\qquad (\exists t : t \in X \;\mathbf{w}\; X_2 : t\lceil\delta.(\mathbf{a}C_0 \cup \mathbf{a}C_1) \in \mu.X \wedge t\lceil\delta.\mathbf{a}C_2 \in \mu.X_2)\} \\
&= \quad \{\text{Property 4.2.2, w is associative}\} \\
&\quad \{X_0 \;\mathbf{w}\; X_1 \;\mathbf{w}\; X_2 \mid X_0 \in \mathbf{c}C_0 \wedge X_1 \in \mathbf{c}C_1 \wedge X_2 \in \mathbf{c}C_2 \wedge \\
&\qquad\qquad (\exists t : t \in X_0 \;\mathbf{w}\; X_1 : t\lceil\delta.\mathbf{a}C_0 \in \mu.X_0 \wedge t\lceil\delta.\mathbf{a}C_1 \in \mu.X_1) \wedge \\
&\qquad\qquad (\exists t : t \in X_0 \;\mathbf{w}\; X_1 \;\mathbf{w}\; X_2 : t\lceil\delta.(\mathbf{a}C_0 \cup \mathbf{a}C_1) \in \mu.(X_0 \;\mathbf{w}\; X_1) \\
&\qquad\qquad\qquad\qquad \wedge t\lceil\delta.\mathbf{a}C_2 \in \mu.X_2)\} \\
&= \quad \{\text{Property 4.2.5, definition of w, property of } \lceil\,\} \\
&\quad \{X_0 \;\mathbf{w}\; X_1 \;\mathbf{w}\; X_2 \mid X_0 \in \mathbf{c}C_0 \wedge X_1 \in \mathbf{c}C_1 \wedge X_2 \in \mathbf{c}C_2 \wedge \\
&\qquad\qquad (\exists t : t \in X_0 \;\mathbf{w}\; X_1 : t\lceil\delta.\mathbf{a}C_0 \in \mu.X_0 \wedge t\lceil\delta.\mathbf{a}C_1 \in \mu.X_1) \wedge \\
&\qquad\qquad (\exists t : t \in X_0 \;\mathbf{w}\; X_1 \;\mathbf{w}\; X_2 : t\lceil\delta.\mathbf{a}C_0 \in \mu.X_0 \wedge t\lceil\delta.\mathbf{a}C_1 \in \mu.X_1 \\
&\qquad\qquad\qquad\qquad \wedge t\lceil\delta.\mathbf{a}C_2 \in \mu.X_2)\} \\
&= \quad \{\text{definition of w, predicate calculus}\}
\end{aligned}
$$

$$
\begin{aligned}
&\{X_0 \,\mathbf{w}\, X_1 \,\mathbf{w}\, X_2 \mid X_0 \in \mathbf{c}C_0 \wedge X_1 \in \mathbf{c}C_1 \wedge X_2 \in \mathbf{c}C_2 \wedge \\
&\qquad (\exists t : t \in X_0 \,\mathbf{w}\, X_1 \,\mathbf{w}\, X_2 : t\restriction\delta.\mathbf{a}C_0 \in \mu.X_0 \wedge t\restriction\delta.\mathbf{a}C_1 \in \mu.X_1 \\
&\qquad\qquad \wedge t\restriction\delta.\mathbf{a}C_2 \in \mu.X_2)\} \\
=\quad &\{\text{definition of } \mathbf{w}\text{, predicate calculus}\} \\
&\{X_0 \,\mathbf{w}\, X_1 \,\mathbf{w}\, X_2 \mid X_0 \in \mathbf{c}C_0 \wedge X_1 \in \mathbf{c}C_1 \wedge X_2 \in \mathbf{c}C_2 \wedge \\
&\qquad (\exists t : t \in X_1 \,\mathbf{w}\, X_2 : t\restriction\delta.\mathbf{a}C_1 \in \mu.X_1 \wedge t\restriction\delta.\mathbf{a}C_2 \in \mu.X_2) \wedge \\
&\qquad (\exists t : t \in X_0 \,\mathbf{w}\, X_1 \,\mathbf{w}\, X_2 : t\restriction\delta.\mathbf{a}C_0 \in \mu.X_0 \wedge t\restriction\delta.\mathbf{a}C_1 \in \mu.X_1 \\
&\qquad\qquad \wedge t\restriction\delta.\mathbf{a}C_2 \in \mu.X_2)\} \\
=\quad &\{\text{Property 4.2.5, definition of } \mathbf{w}\text{, property of } \restriction\} \\
&\{X_0 \,\mathbf{w}\, X_1 \,\mathbf{w}\, X_2 \mid X_0 \in \mathbf{c}C_0 \wedge X_1 \in \mathbf{c}C_1 \wedge X_2 \in \mathbf{c}C_2 \wedge \\
&\qquad (\exists t : t \in X_1 \,\mathbf{w}\, X_2 : t\restriction\delta.\mathbf{a}C_1 \in \mu.X_1 \wedge t\restriction\delta.\mathbf{a}C_2 \in \mu.X_2) \wedge \\
&\qquad (\exists t : t \in X_0 \,\mathbf{w}\, X_1 \,\mathbf{w}\, X_2 : t\restriction\delta.\mathbf{a}C_0 \in \mu.X_0 \\
&\qquad\qquad \wedge t\restriction\delta.(\mathbf{a}C_1 \cup \mathbf{a}C_2) \in \mu.(X_1 \,\mathbf{w}\, X_2))\} \\
=\quad &\{\text{Property 4.2.2, } \mathbf{w} \text{ is associative}\} \\
&\{X_0 \,\mathbf{w}\, X \mid X_0 \in \mathbf{c}C_0 \wedge X \in \mathbf{c}(C_1 \parallel C_2) \wedge \\
&\qquad (\exists t : t \in X_0 \,\mathbf{w}\, X : t\restriction\delta.\mathbf{a}C_0 \in \mu.X_0 \wedge t\restriction\delta.(\mathbf{a}C_1 \cup \mathbf{a}C_2) \in \mu.X)\} \\
=\quad &\{\text{Property 4.2.2}\} \\
&\mathbf{c}(C_0 \parallel (C_1 \parallel C_2)).
\end{aligned}
$$

For deadlocked computations we derive

$$
\begin{aligned}
&\mathbf{d}((C_0 \parallel C_1) \parallel C_2) \\
=\quad &\{\text{Property 4.2.3}\} \\
&\{X \,\mathbf{w}\, X_2 \mid (X \in \mathbf{d}(C_0 \parallel C_1) \wedge X_2 \in \mathbf{c}C_2 \cup \mathbf{d}C_2) \vee \\
&\qquad (X \in \mathbf{c}(C_0 \parallel C_1) \cup \mathbf{d}(C_0 \parallel C_1) \wedge X_2 \in \mathbf{d}C_2) \vee \\
&\qquad (X \in \mathbf{c}(C_0 \parallel C_1) \wedge X_2 \in \mathbf{c}C_2 \wedge \\
&\qquad\ (\forall t : t \in X \,\mathbf{w}\, X_2 : t\restriction\delta.(\mathbf{a}C_0 \cup \mathbf{a}C_1) \notin \mu.X \vee t\restriction\delta.\mathbf{a}C_2 \notin \mu.X_2))\} \\
=\quad &\{\text{Properties 4.2.2 and 4.2.3, } \mathbf{w} \text{ is associative, predicate calculus}\} \\
&\{X_0 \,\mathbf{w}\, X_1 \,\mathbf{w}\, X_2 \\
&\ \mid (X_0 \in \mathbf{d}C_0 \wedge X_1 \in \mathbf{c}C_1 \cup \mathbf{d}C_1 \wedge X_2 \in \mathbf{c}C_2 \cup \mathbf{d}C_2) \vee \\
&\ \ (X_0 \in \mathbf{c}C_0 \cup \mathbf{d}C_0 \wedge X_1 \in \mathbf{d}C_1 \wedge X_2 \in \mathbf{c}C_2 \cup \mathbf{d}C_2) \vee \\
&\ \ (X_0 \in \mathbf{c}C_0 \wedge X_1 \in \mathbf{c}C_1 \wedge X_2 \in \mathbf{c}C_2 \cup \mathbf{d}C_2 \wedge \\
&\ \ \ (\forall t : t \in X_0 \,\mathbf{w}\, X_1 : t\restriction\delta.\mathbf{a}C_0 \notin \mu.X_0 \vee t\restriction\delta.\mathbf{a}C_1 \notin \mu.X_1)) \vee \\
&\ \ (X_0 \in \mathbf{c}C_0 \wedge X_1 \in \mathbf{c}C_1 \wedge X_2 \in \mathbf{d}C_2) \vee \\
&\ \ (X_0 \in \mathbf{d}C_0 \wedge X_1 \in \mathbf{c}C_1 \cup \mathbf{d}C_1 \wedge X_2 \in \mathbf{d}C_2) \vee \\
&\ \ (X_0 \in \mathbf{c}C_0 \cup \mathbf{d}C_0 \wedge X_1 \in \mathbf{d}C_1 \wedge X_2 \in \mathbf{d}C_2) \vee
\end{aligned}
$$

$$
\begin{array}{ll}
 & (X_0 \in \mathbf{c}C_0 \wedge X_1 \in \mathbf{c}C_1 \wedge X_2 \in \mathbf{c}C_2 \wedge \\
 & (\exists t : t \in X_0 \mathbf{w} X_1 : t\restriction\delta.\mathbf{a}C_0 \in \mu.X_0 \wedge t\restriction\delta.\mathbf{a}C_1 \in \mu.X_1) \wedge \\
 & (\forall t : t \in X_0 \mathbf{w} X_1 \mathbf{w} X_2 : t\restriction\delta.(\mathbf{a}C_0 \cup \mathbf{a}C_1) \notin \mu.(X_0 \mathbf{w} X_1) \vee t\restriction\delta.\mathbf{a}C_2 \notin \mu.X_2))\} \\
= & \quad \{\text{predicate calculus}\} \\
 & \{X_0 \mathbf{w} X_1 \mathbf{w} X_2 \\
 & |(X_0 \in \mathbf{d}C_0 \wedge X_1 \in \mathbf{c}C_1 \cup \mathbf{d}C_1 \wedge X_2 \in \mathbf{c}C_2 \cup \mathbf{d}C_2) \vee \\
 & (X_0 \in \mathbf{c}C_0 \cup \mathbf{d}C_0 \wedge X_1 \in \mathbf{d}C_1 \wedge X_2 \in \mathbf{c}C_2 \cup \mathbf{d}C_2) \vee \\
 & (X_0 \in \mathbf{c}C_0 \wedge X_1 \in \mathbf{c}C_1 \wedge X_2 \in \mathbf{d}C_2) \vee \\
 & (X_0 \in \mathbf{d}C_0 \wedge X_1 \in \mathbf{c}C_1 \cup \mathbf{d}C_1 \wedge X_2 \in \mathbf{d}C_2) \vee \\
 & (X_0 \in \mathbf{c}C_0 \cup \mathbf{d}C_0 \wedge X_1 \in \mathbf{d}C_1 \wedge X_2 \in \mathbf{d}C_2) \vee \\
 & (X_0 \in \mathbf{c}C_0 \wedge X_1 \in \mathbf{c}C_1 \wedge X_2 \in \mathbf{c}C_2 \wedge \\
 & (\forall t : t \in X_0 \mathbf{w} X_1 : t\restriction\delta.\mathbf{a}C_0 \notin \mu.X_0 \vee t\restriction\delta.\mathbf{a}C_1 \notin \mu.X_1)) \vee \\
 & (X_0 \in \mathbf{c}C_0 \wedge X_1 \in \mathbf{c}C_1 \wedge X_2 \in \mathbf{c}C_2 \wedge \\
 & (\exists t : t \in X_0 \mathbf{w} X_1 : t\restriction\delta.\mathbf{a}C_0 \in \mu.X_0 \wedge t\restriction\delta.\mathbf{a}C_1 \in \mu.X_1) \wedge \\
 & (\forall t : t \in X_0 \mathbf{w} X_1 \mathbf{w} X_2 : t\restriction\delta.(\mathbf{a}C_0 \cup \mathbf{a}C_1) \notin \mu.(X_0 \mathbf{w} X_1) \vee t\restriction\delta.\mathbf{a}C_2 \notin \mu.X_2))\} \\
= & \quad \{\text{Property 4.2.5, definition of } \mathbf{w}\text{, predicate calculus}\} \\
 & \{X_0 \mathbf{w} X_1 \mathbf{w} X_2 \\
 & |(X_0 \in \mathbf{d}C_0 \wedge X_1 \in \mathbf{c}C_1 \cup \mathbf{d}C_1 \wedge X_2 \in \mathbf{c}C_2 \cup \mathbf{d}C_2) \vee \\
 & (X_0 \in \mathbf{c}C_0 \cup \mathbf{d}C_0 \wedge X_1 \in \mathbf{d}C_1 \wedge X_2 \in \mathbf{c}C_2 \cup \mathbf{d}C_2) \vee \\
 & (X_0 \in \mathbf{c}C_0 \wedge X_1 \in \mathbf{c}C_1 \wedge X_2 \in \mathbf{d}C_2) \vee \\
 & (X_0 \in \mathbf{d}C_0 \wedge X_1 \in \mathbf{c}C_1 \cup \mathbf{d}C_1 \wedge X_2 \in \mathbf{d}C_2) \vee \\
 & (X_0 \in \mathbf{c}C_0 \cup \mathbf{d}C_0 \wedge X_1 \in \mathbf{d}C_1 \wedge X_2 \in \mathbf{d}C_2) \vee \\
 & (X_0 \in \mathbf{c}C_0 \wedge X_1 \in \mathbf{c}C_1 \wedge X_2 \in \mathbf{c}C_2 \wedge \\
 & (\forall t : t \in X_0 \mathbf{w} X_1 \mathbf{w} X_2 : t\restriction\delta.\mathbf{a}C_0 \notin \mu.X_0 \vee t\restriction\delta.\mathbf{a}C_1 \notin \mu.X_1 \vee \\
 & \qquad\qquad t\restriction\delta.\mathbf{a}C_2 \notin \mu.X_2))\} \\
= & \quad \{\text{Property 4.2.5, definition of } \mathbf{w}\text{, predicate calculus}\} \\
 & \{X_0 \mathbf{w} X_1 \mathbf{w} X_2 \\
 & |(X_0 \in \mathbf{c}C_0 \cup \mathbf{d}C_0 \wedge X_1 \in \mathbf{d}C_1 \wedge X_2 \in \mathbf{c}C_2 \cup \mathbf{d}C_2) \vee \\
 & (X_0 \in \mathbf{c}C_0 \cup \mathbf{d}C_0 \wedge X_1 \in \mathbf{c}C_1 \cup \mathbf{d}C_1 \wedge X_2 \in \mathbf{d}C_2) \vee \\
 & (X_0 \in \mathbf{c}C_0 \cup \mathbf{d}C_0 \wedge X_1 \in \mathbf{c}C_1 \wedge X_2 \in \mathbf{c}C_2 \wedge \\
 & (\forall t : t \in X_1 \mathbf{w} X_2 : t\restriction\delta.\mathbf{a}C_1 \notin \mu.X_1 \vee t\restriction\delta.\mathbf{a}C_2 \notin \mu.X_2)) \vee \\
 & (X_0 \in \mathbf{d}C_0 \wedge X_1 \in \mathbf{c}C_1 \wedge X_2 \in \mathbf{c}C_2) \vee \\
 & (X_0 \in \mathbf{d}C_0 \wedge X_1 \in \mathbf{d}C_1 \wedge X_2 \in \mathbf{c}C_2 \cup \mathbf{d}C_2) \vee \\
 & (X_0 \in \mathbf{d}C_0 \wedge X_1 \in \mathbf{c}C_1 \cup \mathbf{d}C_1 \wedge X_2 \in \mathbf{d}C_2) \vee \\
 & (X_0 \in \mathbf{c}C_0 \wedge X_1 \in \mathbf{c}C_1 \wedge X_2 \in \mathbf{c}C_2 \wedge
\end{array}
$$

$$(\exists t : t \in X_1 \mathbin{\mathbf{w}} X_2 : t\restriction\delta.\mathbf{a}C_1 \in \mu.X_1 \wedge t\restriction\delta.\mathbf{a}C_2 \in \mu.X_2) \wedge$$
$$(\forall t : t \in X_0 \mathbin{\mathbf{w}} X_1 \mathbin{\mathbf{w}} X_2 : t\restriction\delta.\mathbf{a}C_0 \notin \mu.X_0 \vee t\restriction\delta.(\mathbf{a}C_1 \cup \mathbf{a}C_2) \notin \mu.(X_1 \mathbin{\mathbf{w}} X_2)))\}$$
$$= \quad \{\text{Properties 4.2.2 and 4.2.3}\}$$
$$\mathbf{d}(C_0 \parallel (C_1 \parallel C_2)).$$

□

We say that command C is implemented by system S if $\mathbf{a}S \supseteq \delta.(\mathbf{a}C \cup \{\surd\})$ and

$$\mathbf{c}C = \{p\restriction\delta.\mathbf{a}C \mid p \in \mathbf{c}(S\restriction\delta.(\mathbf{a}C \cup \{\surd\}) \parallel \mathbf{tick}) \wedge p = pref.(\mu.p)$$
$$\wedge (\forall t : t \in \mu.p : (\exists u :: t = u\,1{\cdot}\surd))\}$$

and

$$\mathbf{d}C = \{p\restriction\delta.\mathbf{a}C \mid p \in \mathbf{c}(S\restriction\delta.(\mathbf{a}C \cup \{\surd\}) \parallel \mathbf{tick}) \wedge p\restriction 1{\cdot}\surd = \{\varepsilon\}\}.$$

In order to prove that $sys.C$ as defined in the previous section implements command C an intermediate result is needed that is expressed by the following lemma.

Lemma 4.2.8

Let C be a command and let $\gamma.C = \{\{0{\cdot}\surd\} +\!\!+ X +\!\!+ \{1{\cdot}\surd\} \mid X \in \mu.\mathbf{c}C\}$. Then

$$\mathbf{c}(sys.C\restriction\delta.(\mathbf{a}C \cup \{\surd\}))$$
$$=$$
$$\{pref.X \mid X \in (\gamma.C)^{\infty}\} \cup \{pref.(X +\!\!+ \{0{\cdot}\surd\} +\!\!+ Y) \mid X \in (\gamma.C)^{*} \wedge Y \in \mathbf{d}C\}.$$

Proof

This proof goes by induction on the structure of C.

Base $C = a$

Then $\gamma.a = \{\{0{\cdot}\surd\ 0{\cdot}a\ 1{\cdot}a\ 1{\cdot}\surd\}\}$ and

$$\mathbf{c}(sys.a\restriction\{0{\cdot}a, 1{\cdot}a, 0{\cdot}\surd, 1{\cdot}\surd\})$$
$$= \quad \{\text{definition of } sys.a\}$$
$$\mathbf{c}(\mathbf{con}.\surd.a)$$
$$= \quad \{\text{definitions of } \mathbf{con}, \text{ and of computations}\}$$
$$\{\mathbf{t}(\mathbf{pr}((0{\cdot}\surd;\ 0{\cdot}a;\ 1{\cdot}a;\ 1{\cdot}\surd)^{*}))\}$$
$$= \quad \{\text{definitions of } A^{\infty}, \text{ and of } \gamma.a\}$$

$$
\begin{array}{ll}
& \{pref.X \mid X \in (\gamma.a)^{\infty}\} \\
= & \quad \{\mathbf{d}a = \emptyset, \text{ set calculus}\} \\
& \{pref.X \mid X \in (\gamma.a)^{\infty}\} \cup \{pref.(X +\!\!+ \{0{\cdot}\surd\} +\!\!+ Y) \mid X \in (\gamma.a)^{*} \wedge Y \in \mathbf{d}a\}.
\end{array}
$$

Step

- $C = \#[C']$

Then $\gamma.\#[C'] = \emptyset$, which follows from the fact that $\mathbf{c}\#[C'] = \emptyset$. Also $\mathbf{a}C = \mathbf{a}C'$. We derive

$$
\begin{array}{ll}
& \mathbf{c}(sys.\#[C']\restriction\delta.(\mathbf{a}C' \cup \{\surd\})) \\
= & \quad \{\text{definitions of } sys.\#[C'], \mathbf{rep}, \mathbf{con}, \text{ and of computations}\} \\
& \{(\mathbf{t}(\mathbf{pr}((0{\cdot}\surd;\ (l{\cdot}0{\cdot}\surd;\ l{\cdot}1{\cdot}\surd)^{*}))\ \mathbf{w}\ \mathbf{stop}.\{1{\cdot}\surd\} \\
& \qquad \mathbf{w}\ (\mathbf{W}a : a \in \mathbf{a}C' : \mathbf{pr}((l{\cdot}0{\cdot}a;\ 0{\cdot}a;\ 1{\cdot}a;\ l{\cdot}1{\cdot}a)^{*}))) \\
& \quad \mathbf{w}\ X)\restriction\delta.(\mathbf{a}C' \cup \{\surd\}) \mid X \in l{\cdot}\mathbf{c}(sys.C')\} \\
= & \quad \{\text{properties of projection, set calculus, Lemma 1.1.13}\} \\
& \{(\mathbf{t}(\mathbf{pr}((0{\cdot}\surd;\ (l{\cdot}0{\cdot}\surd;\ l{\cdot}1{\cdot}\surd)^{*}))\ \mathbf{w}\ \mathbf{stop}.\{1{\cdot}\surd\} \\
& \qquad \mathbf{w}\ (\mathbf{W}a : a \in \mathbf{a}C' : \mathbf{pr}((l{\cdot}0{\cdot}a;\ 0{\cdot}a;\ 1{\cdot}a;\ l{\cdot}1{\cdot}a)^{*}))) \\
& \quad \mathbf{w}\ X)\restriction\delta.(\mathbf{a}C' \cup \{\surd\}) \mid X \in l{\cdot}\mathbf{c}(sys.C'\restriction\delta.(\mathbf{a}C' \cup \{\surd\}))\} \\
= & \quad \{\text{induction hypothesis}\} \\
& \{pref.(\{0{\cdot}\surd\} +\!\!+ X) \mid X \in (\mu.\mathbf{c}C')^{\infty}\} \cup \\
& \{pref.(\{0{\cdot}\surd\} +\!\!+ X +\!\!+ Y) \mid X \in (\mu.\mathbf{c}C')^{*} \wedge Y \in \mathbf{d}C'\} \\
= & \quad \{\text{Property 4.2.3 — the part concerning } \mathbf{d}\#[C]\} \\
& \{pref.(\{0{\cdot}\surd\} +\!\!+ X) \mid X \in \mathbf{d}\#[C']\} \\
= & \quad \{\gamma.\#[C'] = \emptyset,\ \emptyset^{\infty} = \emptyset,\ \emptyset^{*} = \{\{\varepsilon\}\}\} \\
& \{pref.X \mid X \in (\gamma.\#[C'])^{\infty}\} \cup \\
& \{pref.(X +\!\!+ \{0{\cdot}\surd\} +\!\!+ Y) \mid X \in (\gamma.\#[C'])^{*} \wedge Y \in \mathbf{d}\#[C']\}.
\end{array}
$$

- $C = \#N[C']$

Then $\gamma.\#N[C'] = \{\{0{\cdot}\surd\} +\!\!+ X^{N} +\!\!+ \{1{\cdot}\surd\} \mid X \in \mu.\mathbf{c}C'\}$ and $\mathbf{a}C = \mathbf{a}C'$. We derive

$$
\begin{array}{ll}
& \mathbf{c}(sys.\#N[C']\restriction\delta.(\mathbf{a}C' \cup \{\surd\})) \\
= & \quad \{\text{definitions of } sys.\#N[C'], \mathbf{rep}_{N}, \mathbf{con}, \text{ and of computations}\} \\
& \{(\mathbf{t}(\mathbf{pr}((0{\cdot}\surd;\ (l{\cdot}0{\cdot}\surd;\ l{\cdot}1{\cdot}\surd)^{N};\ 1{\cdot}\surd)^{*})
\end{array}
$$

$$
\begin{array}{ll}
 & \mathbf{w}\ (\mathbf{W}a : a \in \mathbf{a}C' : \mathbf{pr}((l{\cdot}0{\cdot}a\,;\, 0{\cdot}a\,;\, 1{\cdot}a\,;\, l{\cdot}1{\cdot}a)^*))) \\
 & \mathbf{w}\ X)\restriction\delta.(\mathbf{a}C' \cup \{\surd\}) \mid X \in l{\cdot}\mathbf{c}(sys.C')\} \\
= & \quad \{\text{properties of projection, set calculus, Lemma 1.1.13}\} \\
 & \{(\mathbf{t}(\mathbf{pr}((0{\cdot}\surd\,;\, (l{\cdot}0{\cdot}\surd\,;\, l{\cdot}1{\cdot}\surd)^N\,;\, 1{\cdot}\surd)^* \\
 & \quad \mathbf{w}\ (\mathbf{W}a : a \in \mathbf{a}C' : \mathbf{pr}((l{\cdot}0{\cdot}a\,;\, 0{\cdot}a\,;\, 1{\cdot}a\,;\, l{\cdot}1{\cdot}a)^*))) \\
 & \mathbf{w}\ X)\restriction\delta.(\mathbf{a}C' \cup \{\surd\}) \mid X \in l{\cdot}\mathbf{c}(sys.C'\restriction\delta.(\mathbf{a}C' \cup \{\surd\}))\} \\
= & \quad \{\text{induction hypothesis, definition of } \gamma.\#N[C]\} \\
 & \{pref.X \mid X \in (\gamma.\#N[C'])^\infty\} \cup \\
 & \{pref.(X +\!\!+ \{0{\cdot}\surd\} +\!\!+ X' +\!\!+ Y) \\
 & \mid X \in (\gamma.\#N[C'])^* \wedge (\exists i : 0 \le i < N : X' \in (\mu.\mathbf{c}C')^i) \wedge Y \in \mathbf{d}C'\} \\
= & \quad \{\text{Properties 4.2.2 and 4.2.3 — the parts concerning } \mathbf{c}\#N[C] \text{ and } \mathbf{d}\#N[C]\} \\
 & \{pref.X \mid X \in (\gamma.\#N[C'])^\infty\} \cup \\
 & \{pref.(X +\!\!+ \{0{\cdot}\surd\} +\!\!+ Y) \mid X \in (\gamma.\#N[C'])^* \wedge Y \in \mathbf{d}\#N[C']\}.
\end{array}
$$

- $C = C_0\,;\, C_1$

In this case we have

$$\gamma.(C_0\,;\, C_1) = \{\{0{\cdot}\surd\} +\!\!+ X_0 +\!\!+ X_1 +\!\!+ \{1{\cdot}\surd\} \mid X_0 \in \mu.\mathbf{c}C_0 \wedge X_1 \in \mu.\mathbf{c}C_1\}.$$

We derive

$$
\begin{array}{ll}
 & \mathbf{c}(sys.(C_0\,;\, C_1)\restriction\delta.(\mathbf{a}C_0 \cup \mathbf{a}C_1 \cup \{\surd\})) \\
= & \quad \{\text{definitions of } sys.(C_0\,;\, C_1), \textbf{seq}, \textbf{mix}, \textbf{con}, \text{ and of computations, the} \\
 & \qquad \text{choices in } \textbf{mix} \text{ are resolved during the execution by the sequencer}\} \\
 & \{(\mathbf{t}(\mathbf{pr}((0{\cdot}\surd\,;\, l{\cdot}0{\cdot}\surd\,;\, l{\cdot}1{\cdot}\surd\,;\, r{\cdot}0{\cdot}\surd\,;\, r{\cdot}1{\cdot}\surd\,;\, 1{\cdot}\surd)^* \\
 & \quad \mathbf{w}\ (\mathbf{W}a : a \in \mathbf{a}C_0 \cap \mathbf{a}C_1 : \mathbf{pr}((l{\cdot}0{\cdot}a\,;\, 0{\cdot}a\,;\, 1{\cdot}a\,;\, l{\cdot}1{\cdot}a \mid r{\cdot}0{\cdot}a\,;\, 0{\cdot}a\,;\, 1{\cdot}a\,;\, r{\cdot}1{\cdot}a)^*)) \\
 & \quad \mathbf{w}\ (\mathbf{W}a : a \in \mathbf{a}C_0 \setminus \mathbf{a}C_1 : \mathbf{pr}((l{\cdot}0{\cdot}a\,;\, 0{\cdot}a\,;\, 1{\cdot}a\,;\, l{\cdot}1{\cdot}a)^*)) \\
 & \quad \mathbf{w}\ (\mathbf{W}a : a \in \mathbf{a}C_1 \setminus \mathbf{a}C_0 : \mathbf{pr}((r{\cdot}0{\cdot}a\,;\, 0{\cdot}a\,;\, 1{\cdot}a\,;\, r{\cdot}1{\cdot}a)^*))) \\
 & \mathbf{w}\ X_0\ \mathbf{w}\ X_1)\restriction\delta.(\mathbf{a}C_0 \cup \mathbf{a}C_1 \cup \{\surd\}) \\
 & \mid X_0 \in l{\cdot}\mathbf{c}(sys.C_0) \wedge X_1 \in r{\cdot}\mathbf{c}(sys.C_1)\} \\
= & \quad \{\text{properties of projection, set calculus, Lemma 1.1.13 (twice)}\} \\
 & \{(\mathbf{t}(\mathbf{pr}((0{\cdot}\surd\,;\, l{\cdot}0{\cdot}\surd\,;\, l{\cdot}1{\cdot}\surd\,;\, r{\cdot}0{\cdot}\surd\,;\, r{\cdot}1{\cdot}\surd\,;\, 1{\cdot}\surd)^* \\
 & \quad \mathbf{w}\ (\mathbf{W}a : a \in \mathbf{a}C_0 \cap \mathbf{a}C_1 : \mathbf{pr}((l{\cdot}0{\cdot}a\,;\, 0{\cdot}a\,;\, 1{\cdot}a\,;\, l{\cdot}1{\cdot}a \mid r{\cdot}0{\cdot}a\,;\, 0{\cdot}a\,;\, 1{\cdot}a\,;\, r{\cdot}1{\cdot}a)^*)) \\
 & \quad \mathbf{w}\ (\mathbf{W}a : a \in \mathbf{a}C_0 \setminus \mathbf{a}C_1 : \mathbf{pr}((l{\cdot}0{\cdot}a\,;\, 0{\cdot}a\,;\, 1{\cdot}a\,;\, l{\cdot}1{\cdot}a)^*)) \\
 & \quad \mathbf{w}\ (\mathbf{W}a : a \in \mathbf{a}C_1 \setminus \mathbf{a}C_0 : \mathbf{pr}((r{\cdot}0{\cdot}a\,;\, 0{\cdot}a\,;\, 1{\cdot}a\,;\, r{\cdot}1{\cdot}a)^*))) \\
 & \mathbf{w}\ X_0\ \mathbf{w}\ X_1)\restriction\delta.(\mathbf{a}C_0 \cup \mathbf{a}C_1 \cup \{\surd\})
\end{array}
$$

$$
\begin{aligned}
& \mid X_0 \in l{\cdot}\mathbf{c}(sys.C_0\restriction\delta.(\mathbf{a}C_0 \cup \{\surd\})) \wedge X_1 \in r{\cdot}\mathbf{c}(sys.C_1\restriction\delta.(\mathbf{a}C_1 \cup \{\surd\}))\} \\
= \quad & \{\text{induction hypothesis, definition of } \gamma.(C_0\,;\,C_1)\} \\
& \{pref.X \mid X \in (\gamma.(C_0\,;\,C_1))^\infty\} \cup \\
& \{pref.(X + \{0{\cdot}\surd\} + X' + Y) \mid X \in (\gamma.(C_0\,;\,C_1))^* \wedge X' \in \mu.\mathbf{c}C_0 \wedge Y \in \mathbf{d}C_1\} \\
& \cup \{pref.(X + \{0{\cdot}\surd\} + Y) \mid X \in (\gamma.(C_0\,;\,C_1))^* \wedge Y \in \mathbf{d}C_1\} \\
= \quad & \{\text{Properties 4.2.2 and 4.2.3 — the parts concerning } \mathbf{c}(C_0\,;\,C_1) \text{ and } \\
& \quad \mathbf{d}(C_0\,;\,C_1)\} \\
& \{pref.X \mid X \in (\gamma.(C_0\,;\,C_1))^\infty\} \cup \\
& \{pref.(X + \{0{\cdot}\surd\} + Y) \mid X \in (\gamma.(C_0\,;\,C_1))^* \wedge Y \in \mathbf{d}(C_0\,;\,C_1)\}.
\end{aligned}
$$

- $C = C_0 \parallel C_1$

In this case

$$
\begin{aligned}
& \gamma.(C_0 \parallel C_1) \\
= \quad & \\
& \{\{0{\cdot}\surd\} + \mu.(pref.X_0 \mathbin{\mathbf{w}} pref.X_1) + \{1{\cdot}\surd\} \\
& \mid X_0 \in \mu.\mathbf{c}C_0 \wedge X_1 \in \mu.\mathbf{c}C_1 \wedge \\
& \quad (\exists t : t \in pref.X_0 \mathbin{\mathbf{w}} pref.X_1 : t\restriction\delta.\mathbf{a}C_0 \in X_0 \wedge t\restriction\delta.\mathbf{a}C_1 \in X_1)\}.
\end{aligned}
$$

We derive

$$
\begin{aligned}
& \mathbf{c}(sys.(C_0 \parallel C_1)\restriction\delta.(\mathbf{a}C_0 \cup \mathbf{a}C_1 \cup \{\surd\})) \\
= \quad & \{\text{definitions of } sys.(C_0 \parallel C_1), \mathbf{con}, \text{ and of computations}\} \\
& \{(\mathbf{t}(\mathbf{pr}((0{\cdot}\surd\,;\, l{\cdot}0{\cdot}\surd\,;\, l{\cdot}1{\cdot}\surd\,;\, 1{\cdot}\surd)^*) \mathbin{\mathbf{w}} \mathbf{t}(\mathbf{pr}((0{\cdot}\surd\,;\, r{\cdot}0{\cdot}\surd\,;\, r{\cdot}1{\cdot}\surd\,;\, 1{\cdot}\surd)^* \\
& \quad \mathbf{w}\ (\mathbf{W}a : a \in \mathbf{a}C_0 : \mathbf{pr}((l{\cdot}0{\cdot}a\,;\, 0{\cdot}a\,;\, 1{\cdot}a\,;\, l{\cdot}1{\cdot}a)^*)) \\
& \quad \mathbf{w}\ (\mathbf{W}a : a \in \mathbf{a}C_1 : \mathbf{pr}((r{\cdot}0{\cdot}a\,;\, 0{\cdot}a\,;\, 1{\cdot}a\,;\, r{\cdot}1{\cdot}a)^*))) \\
& \quad \mathbf{w}\ X_0 \mathbf{w}\ X_1)\restriction\delta.(\mathbf{a}C_0 \cup \mathbf{a}C_1 \cup \{\surd\}) \\
& \mid X_0 \in l{\cdot}\mathbf{c}(sys.C_0) \wedge X_1 \in r{\cdot}\mathbf{c}(sys.C_1)\} \\
= \quad & \{\text{properties of projection, set calculus, Lemma 1.1.13 (twice)}\} \\
& \{(\mathbf{t}(\mathbf{pr}((0{\cdot}\surd\,;\, l{\cdot}0{\cdot}\surd\,;\, l{\cdot}1{\cdot}\surd\,;\, 1{\cdot}\surd)^*) \mathbin{\mathbf{w}} \mathbf{t}(\mathbf{pr}((0{\cdot}\surd\,;\, r{\cdot}0{\cdot}\surd\,;\, r{\cdot}1{\cdot}\surd\,;\, 1{\cdot}\surd)^* \\
& \quad \mathbf{w}\ (\mathbf{W}a : a \in \mathbf{a}C_0 : \mathbf{pr}((l{\cdot}0{\cdot}a\,;\, 0{\cdot}a\,;\, 1{\cdot}a\,;\, l{\cdot}1{\cdot}a)^*)) \\
& \quad \mathbf{w}\ (\mathbf{W}a : a \in \mathbf{a}C_1 : \mathbf{pr}((r{\cdot}0{\cdot}a\,;\, 0{\cdot}a\,;\, 1{\cdot}a\,;\, r{\cdot}1{\cdot}a)^*))) \\
& \quad \mathbf{w}\ X_0 \mathbf{w}\ X_1)\restriction\delta.(\mathbf{a}C_0 \cup \mathbf{a}C_1 \cup \{\surd\}) \\
& \mid X_0 \in l{\cdot}\mathbf{c}(sys.C_0\restriction\delta.(\mathbf{a}C_0 \cup \{\surd\})) \wedge X_1 \in r{\cdot}\mathbf{c}(sys.C_1\restriction\delta.(\mathbf{a}C_0 \cup \{\surd\}))\} \\
= \quad & \{\text{induction hypothesis, definition of } \gamma.(C_0 \parallel C_1)\} \\
& \{pref.X \mid X \in (\gamma.(C_0 \parallel C_1))^\infty\} \cup
\end{aligned}
$$

$$
\begin{aligned}
&\{pref.(X \mathbin{+\!\!+} \{0{\cdot}\surd\} \mathbin{+\!\!+} (X_0 \mathbf{w} X_1)) \\
&\quad \mid X \in (\gamma.(C_0 ; C_1))^* \wedge \\
&\quad\quad ((X_0 \in \mathbf{c}C_0 \cup \mathbf{d}C_0 \wedge X_1 \in \mathbf{d}C_1) \vee (X_0 \in \mathbf{d}C_0 \wedge X_1 \in \mathbf{c}C_1 \cup \mathbf{d}C_1) \\
&\quad\quad\quad \vee (X_0 \in \mathbf{c}C_0 \wedge X_1 \in \mathbf{c}C_1 \wedge \\
&\quad\quad\quad\quad (\forall t : t \in X_0 \mathbf{w} X_1 : t\restriction\delta.\mathbf{a}C_0 \notin \mu.X_0 \vee t\restriction\delta.\mathbf{a}C_1 \notin \mu.X_1))\} \\
=\quad &\{\text{Properties 4.2.2 and 4.2.3 — the parts concerning } \mathbf{c}(C_0 \parallel C_1) \text{ and} \\
&\quad \mathbf{d}(C_0 \parallel C_1)\} \\
&\{pref.X \mid X \in (\gamma.(C_0 \parallel C_1))^\infty\} \cup \\
&\{pref.(X \mathbin{+\!\!+} \{0{\cdot}\surd\} \mathbin{+\!\!+} Y) \mid X \in (\gamma.(C_0 \parallel C_1))^* \wedge Y \in \mathbf{d}(C_0 \parallel C_1)\}.
\end{aligned}
$$

- $C = C_0 \sqcap C_1$

Then we have

$$\gamma.(C_0 \sqcap C_1) = \{\{0{\cdot}\surd\} \mathbin{+\!\!+} X \mathbin{+\!\!+} \{1{\cdot}\surd\} \mid X \in \mu.\mathbf{c}C_0 \cup \mu.\mathbf{c}C_1\}.$$

We derive

$$
\begin{aligned}
&\mathbf{c}(sys.(C_0 \sqcap C_1)\restriction(\delta.(\mathbf{a}C_0 \cup \mathbf{a}C_1) \cup \{0{\cdot}\surd, 1{\cdot}\surd\})) \\
=\quad &\{\text{definitions of } sys.(C_0 \sqcap C_1), \textbf{sel}, \textbf{mix}, \textbf{con}, \text{ and of computations, the} \\
&\quad \text{choices in } \textbf{mix} \text{ are resolved during the execution by the choices made} \\
&\quad \text{in the selector}\} \\
&\{(\mathbf{t}((\mathbf{W}a : a \in \mathbf{a}C_0 \cap \mathbf{a}C_1 : \mathbf{pr}((l{\cdot}0{\cdot}a\,;\, 0{\cdot}a\,;\, 1{\cdot}a\,;\, l{\cdot}1{\cdot}a \mid r{\cdot}0{\cdot}a\,;\, 0{\cdot}a\,;\, 1{\cdot}a\,;\, r{\cdot}1{\cdot}a)^*)) \\
&\quad\quad \mathbf{w}\ (\mathbf{W}a : a \in \mathbf{a}C_0 \setminus \mathbf{a}C_1 : \mathbf{pr}((l{\cdot}0{\cdot}a\,;\, 0{\cdot}a\,;\, 1{\cdot}a\,;\, l{\cdot}1{\cdot}a)^*)) \\
&\quad\quad \mathbf{w}\ (\mathbf{W}a : a \in \mathbf{a}C_1 \setminus \mathbf{a}C_0 : \mathbf{pr}((r{\cdot}0{\cdot}a\,;\, 0{\cdot}a\,;\, 1{\cdot}a\,;\, r{\cdot}1{\cdot}a)^*))) \\
&\quad\ \mathbf{w}\ Y\ \mathbf{w}\ X_0\ \mathbf{w}\ X_1)\restriction\delta.(\mathbf{a}C_0 \cup \mathbf{a}C_1 \cup \{\surd\}) \\
&\quad \mid Y \text{ is a maximal chain of } \mathbf{t}(\mathbf{pr}((0{\cdot}\surd\,;\, (l{\cdot}0{\cdot}\surd\,;\, l{\cdot}1{\cdot}\surd \mid r{\cdot}0{\cdot}\surd\,;\, r{\cdot}1{\cdot}\surd)\,;\, 1{\cdot}\surd)^*)) \\
&\quad\quad \wedge X_0 \in l{\cdot}\mathbf{c}(sys.C_0) \wedge X_1 \in r{\cdot}\mathbf{c}(sys.C_1)\} \\
=\quad &\{\text{properties of projection, set calculus, Lemma 1.1.13 (twice)}\} \\
&\{(\mathbf{t}((\mathbf{W}a : a \in \mathbf{a}C_0 \cap \mathbf{a}C_1 : \mathbf{pr}((l{\cdot}0{\cdot}a\,;\, 0{\cdot}a\,;\, 1{\cdot}a\,;\, l{\cdot}1{\cdot}a \mid r{\cdot}0{\cdot}a\,;\, 0{\cdot}a\,;\, 1{\cdot}a\,;\, r{\cdot}1{\cdot}a)^*)) \\
&\quad\quad \mathbf{w}\ (\mathbf{W}a : a \in \mathbf{a}C_0 \setminus \mathbf{a}C_1 : \mathbf{pr}((l{\cdot}0{\cdot}a\,;\, 0{\cdot}a\,;\, 1{\cdot}a\,;\, l{\cdot}1{\cdot}a)^*)) \\
&\quad\quad \mathbf{w}\ (\mathbf{W}a : a \in \mathbf{a}C_1 \setminus \mathbf{a}C_0 : \mathbf{pr}((r{\cdot}0{\cdot}a\,;\, 0{\cdot}a\,;\, 1{\cdot}a\,;\, r{\cdot}1{\cdot}a)^*))) \\
&\quad\ \mathbf{w}\ Y\ \mathbf{w}\ X_0\ \mathbf{w}\ X_1)\restriction\delta.(\mathbf{a}C_0 \cup \mathbf{a}C_1 \cup \{\surd\}) \\
&\quad \mid Y \text{ is a maximal chain of } \mathbf{t}(\mathbf{pr}((0{\cdot}\surd\,;\, (l{\cdot}0{\cdot}\surd\,;\, l{\cdot}1{\cdot}\surd \mid r{\cdot}0{\cdot}\surd\,;\, r{\cdot}1{\cdot}\surd)\,;\, 1{\cdot}\surd)^*)) \\
&\quad\quad \wedge X_0 \in l{\cdot}\mathbf{c}(sys.C_0\restriction\delta.(\mathbf{a}C_0 \cup \{\surd\})) \wedge X_1 \in r{\cdot}\mathbf{c}(sys.C_1\restriction\delta.(\mathbf{a}C_1 \cup \{\surd\}))\} \\
=\quad &\{\text{induction hypothesis, definition of } \gamma.(C_0 \sqcap C_1)\} \\
&\{pref.X \mid X \in (\gamma.(C_0 \sqcap C_1))^\infty\} \cup
\end{aligned}
$$

$$\begin{array}{ll} & \{pref.(X +\!\!+ \{0{\cdot}\surd\} +\!\!+ Y) \mid X \in (\gamma.(C_0 \sqcap C_1))^* \wedge Y \in \mathbf{d}C_0 \cup \mathbf{d}C_1\} \\ = & \quad \{\text{Properties 4.2.2 and 4.2.3 — the parts concerning } \mathbf{c}(C_0 \sqcap C_1) \text{ and } \mathbf{d}(C_0 \sqcap C_1)\} \\ & \{pref.X \mid X \in (\gamma.(C_0 \sqcap C_1))^\infty\} \cup \\ & \{pref.(X +\!\!+ \{0{\cdot}\surd\} +\!\!+ Y) \mid X \in (\gamma.(C_0 \sqcap C_1))^* \wedge Y \in \mathbf{d}(C_0 \sqcap C_1)\}. \end{array}$$

□

From the previous section we know that, for command C, $sys.C \in RD$. Moreover, on account of Lemma 3.2.6, we have

$$sys.C\restriction\delta.(\mathbf{a}C \cup \{\surd\}) \in RD.$$

Hence, on account of Theorem 3.2.17, Lemma 4.2.8, and the fact that $pref.\{0{\cdot}\surd\, 1{\cdot}\surd\}$ is the only computation of **tick**, we have the following property.

Property 4.2.9

For command C

$$\begin{array}{l} \mathbf{c}(sys.C\restriction\delta.(\mathbf{a}C \cup \{\surd\}) \parallel \mathbf{tick}) \\ = \\ \{pref.(\{0{\cdot}\surd\} +\!\!+ X +\!\!+ \{1{\cdot}\surd\} \mid X \in \mu.\mathbf{c}C\} \cup \{pref.(\{0{\cdot}\surd\} +\!\!+ X) \mid X \in \mathbf{d}C\}. \end{array}$$

□

Theorem 4.2.10

Every command C is implemented by $sys.C$.

Proof

On account of the definitions of $sys.C$ and of all the components, we conclude that

$$\mathbf{a}sys.C \supseteq \delta.(\mathbf{a}C \cup \{\surd\}).$$

We derive

$$\begin{array}{ll} & p \in \mathbf{c}(sys.C\restriction\delta.(\mathbf{a}C \cup \{\surd\}) \parallel \mathbf{tick}) \wedge p = pref.(\mu.p) \wedge \\ & (\forall t : t \in \mu.p : (\exists u :: t = u\, 1{\cdot}\surd)) \\ \Leftrightarrow & \quad \{\text{Property 4.2.9}\} \\ & (\exists X : X \in \mu.\mathbf{c}C : p = pref.(\{0{\cdot}\surd\} +\!\!+ X +\!\!+ \{1{\cdot}\surd\})) \end{array}$$

$$
\begin{array}{ll}
\Rightarrow & \{\text{property of } \restriction, \text{ predicate calculus, } X \in \mu.\mathbf{c}C \;\Rightarrow\; X \in (\delta.\mathbf{a}C)^*,\ 0{\cdot}\surd \notin \delta.\mathbf{a}C \\
& \quad \text{and } 1{\cdot}\surd \notin \delta.\mathbf{a}C\} \\
& (\exists X : X \in \mu.\mathbf{c}C : p\restriction\delta.\mathbf{a}C = pref.X) \\
\Leftrightarrow & \{X \in \mu.\mathbf{c}C \;\Leftrightarrow\; pref.X \in \mathbf{c}C, \text{ predicate calculus}\} \\
& (\exists X' : X' \in \mathbf{c}C : p\restriction\delta.\mathbf{a}C = X').
\end{array}
$$

Hence,

$$
\begin{aligned}
\mathbf{c}C \supseteq \{p\restriction\delta.\mathbf{a}C \mid\ & p \in \mathbf{c}(sys.C\restriction\delta.(\mathbf{a}C \cup \{\surd\}) \parallel \mathbf{tick}) \wedge p = pref.(\mu.p) \\
& \wedge (\forall t : t \in \mu.p : (\exists u :: t = u\ 1{\cdot}\surd))\}.
\end{aligned}
$$

For the inclusion the other way around we derive

$$
\begin{array}{ll}
& X \in \mathbf{c}C \\
\Rightarrow & \{\text{Property 4.2.9, } X \in \mathbf{c}C \;\Leftrightarrow\; \mu.X \in \mu.\mathbf{c}C,\ X \in \mathbf{c}C \;\Rightarrow\; X \in (\delta.\mathbf{a}C)^*\} \\
& (\exists p : p = pref.(\{0{\cdot}\surd\} +\!\!+ \mu.X +\!\!+ \{1{\cdot}\surd\}) \wedge p \in \mathbf{c}(sys.C\restriction\delta.(\mathbf{a}C \cup \{\surd\}) \parallel \mathbf{tick}) \\
& \quad : p\restriction\delta.\mathbf{a}C = X \wedge X = pref.(\mu.X)) \\
\Rightarrow & \{p = pref.(\{0{\cdot}\surd\} +\!\!+ \mu.X +\!\!+ \{1{\cdot}\surd\}) \text{ and } X = pref.(\mu.X) \text{ imply} \\
& \quad \mu.p = \{0{\cdot}\surd\} +\!\!+ \mu.X +\!\!+ \{1{\cdot}\surd\} \text{ and } p = pref.(\mu.p), \text{ predicate calculus}\} \\
& (\exists p : p \in \mathbf{c}(sys.C\restriction\delta.(\mathbf{a}C \cup \{\surd\}) \parallel \mathbf{tick}) \wedge p = pref.(\mu.p) \\
& \quad : (\forall t : t \in \mu.p : (\exists u :: t = u\ 1{\cdot}\surd)) \wedge p\restriction\delta.\mathbf{a}C = X).
\end{array}
$$

For deadlocked computations we derive

$$
\begin{array}{ll}
& p \in \mathbf{c}(sys.C\restriction\delta.(\mathbf{a}C \cup \{\surd\}) \parallel \mathbf{tick}) \wedge p\restriction 1{\cdot}\surd = \{\varepsilon\} \\
\Leftrightarrow & \{\text{Property 4.2.9}\} \\
& (\exists X : X \in \mathbf{d}C : p = pref.(\{0{\cdot}\surd\} +\!\!+ X)) \\
\Rightarrow & \{\text{property of } \restriction, \text{ predicate calculus, } X \in \mathbf{d}C \;\Rightarrow\; X \in (\delta.\mathbf{a}C)^*,\ 0{\cdot}\surd \notin \delta.\mathbf{a}C\} \\
& (\exists X : X \in \mathbf{d}C : p\restriction\delta.\mathbf{a}C = pref.X) \\
\Leftrightarrow & \{X \in \mathbf{d}C \;\Rightarrow\; X = pref.X\} \\
& (\exists X : X \in \mathbf{d}C : p\restriction\delta.\mathbf{a}C = X).
\end{array}
$$

Hence,

$$
\mathbf{d}C \supseteq \{p\restriction\delta.\mathbf{a}C \mid p \in \mathbf{c}(sys.C\restriction\delta.(\mathbf{a}C \cup \{\surd\}) \parallel \mathbf{tick}) \wedge p\restriction 1{\cdot}\surd = \{\varepsilon\}\}.
$$

For the inclusion the other way around we derive

$$
\begin{array}{ll}
& X \in \mathbf{d}C \\
\Rightarrow & \{\text{Property 4.2.9, } X \in \mathbf{d}C \Rightarrow X \in (\delta.\mathbf{a}C)^*,\ 0{\cdot}\surd \notin \delta.\mathbf{a}C \text{ and } 1{\cdot}\surd \notin \delta.\mathbf{a}C, \\
& \quad \text{property of } \restriction\} \\
& (\exists p : p = \mathit{pref}.(\{0{\cdot}\surd\} +\!\!+ X) \wedge p \in \mathbf{c}(\mathit{sys}.C \restriction \delta.(\mathbf{a}C \cup \{\surd\}) \parallel \mathbf{tick}) \\
& \quad : p \restriction \delta.\mathbf{a}C = X \wedge p \restriction 1{\cdot}\surd = \{\varepsilon\}) \\
\Rightarrow & \{\text{predicate calculus}\} \\
& (\exists p : p \in \mathbf{c}(\mathit{sys}.C \restriction \delta.(\mathbf{a}C \cup \{\surd\}) \parallel \mathbf{tick}) : p \restriction 1{\cdot}\surd = \{\varepsilon\} \wedge p \restriction \delta.\mathbf{a}C = X).
\end{array}
$$

□

4.3 Comparing programs

In this section a method of comparing communication behaviours of programs is introduced. This method is based on the abstract model of Section 3.2 and it is illustrated by means of a few examples. Since we are interested in the occurrence of actions, we have to translate the two-phase handshakes into action occurrences.

For $P = A.C$ the process of P is denoted by $\mathbf{pr}P$ and is defined by

$$\mathbf{pr}P = (\langle \delta.\mathbf{a}C, (\cup X : X \in \mathbf{c}C \cup \mathbf{d}C : X)\rangle \mathbf{\ w\ pr}(\parallel a : a \in \mathbf{a}C : \mathbf{trans}.a)) \restriction A.$$

The set of computations of P is denoted by $\mathbf{c}P$ and is defined by

$$\mathbf{c}P = \{(X \mathbf{\ w\ t}(\mathbf{pr}(\parallel a : a \in \mathbf{a}C : \mathbf{trans}.a))) \restriction A \mid X \in \mathbf{c}C \cup \mathbf{d}C\}.$$

The semantics of program P is denoted by $\mathcal{M}[\![P]\!]$ and is defined by

$$\mathcal{M}[\![P]\!] = (\mathbf{pr}P, \mathbf{c}P).$$

The following lemma shows that computations of program P are equal to computations of $\mathit{sys}.P$.

Lemma 4.3.1

For c-Tangram program $P = A.C$

$$\mathbf{c}(\mathit{sys}.P) = \mathbf{c}P.$$

Proof

Let $T = (\parallel a : a \in \mathbf{a}C : \mathbf{trans}(a))$. We derive

$$
\begin{array}{ll}
 & sys.P \\
= & \quad \{\text{definition of } sys.P\} \\
 & (sys.C \parallel T \parallel \mathbf{tick})\restriction A \\
= & \quad \{A \subseteq \mathbf{a}C, \text{ hence } (\delta.(\mathbf{a}C \cup \{\surd\}) \cup \mathbf{a}C) \cap A = A, \text{ Property 1.1.19, } \parallel \text{ is commutative}\} \\
 & (sys.C \parallel \mathbf{tick} \parallel T)\restriction(\delta.(\mathbf{a}C \cup \{\surd\}) \cup \mathbf{a}C)\restriction A \\
= & \quad \{\mathbf{e}(sys.C \parallel \mathbf{tick}) \cap \mathbf{e}T \subseteq \delta.(\mathbf{a}C \cup \{\surd\}) \cup \mathbf{a}C, \text{ Property 1.2.6}\} \\
 & ((sys.C \parallel \mathbf{tick})\restriction(\delta.(\mathbf{a}C \cup \{\surd\}) \cup \mathbf{a}C) \parallel T\restriction(\delta.(\mathbf{a}C \cup \{\surd\}) \cup \mathbf{a}C))\restriction A \\
= & \quad \{\mathbf{a}C \cap \mathbf{e}(sys.C \parallel \mathbf{tick}) = \varnothing, \mathbf{e}T = \delta.\mathbf{a}C \cup \mathbf{a}C, \surd \notin \mathbf{a}C, \text{ definition of projection for systems}\} \\
 & ((sys.C \parallel \mathbf{tick})\restriction\delta.(\mathbf{a}C \cup \{\surd\}) \parallel T)\restriction A \\
= & \quad \{\mathbf{e}(\mathbf{tick}) = \delta.\{\surd\}, \mathbf{e}(sys.C) \cap \mathbf{e}(\mathbf{tick}) \subseteq \delta.(\mathbf{a}C \cup \{\surd\}), \text{ Property 1.2.6, definition of projection for systems}\} \\
 & (sys.C\restriction\delta.(\mathbf{a}C \cup \{\surd\}) \parallel \mathbf{tick} \parallel T)\restriction A.
\end{array}
$$

Furthermore,

$$
\begin{array}{ll}
 & \mathbf{c}((sys.C\restriction\delta.(\mathbf{a}C \cup \{\surd\}) \parallel \mathbf{tick} \parallel T)\restriction A) \\
= & \quad \{\text{Theorem 3.2.18}\} \\
 & (\mathbf{c}(sys.C\restriction\delta.(\mathbf{a}C \cup \{\surd\}) \parallel \mathbf{tick} \parallel T))\restriction A \\
= & \quad \{sys.C \in RD, \mathbf{tick} \in RD, \text{ for all } a \in \mathbf{a}C, \mathbf{trans}(a) \in RD, \text{ Lemmata 3.2.6 and 3.2.5, Theorem 3.2.17, definitions of } T \text{ and of } \mathbf{trans}\} \\
 & \{X \mathbf{\ w\ t}(\mathbf{pr}T) \mid X \in \mathbf{c}(sys.C\restriction\delta.(\mathbf{a}C \cup \{\surd\}) \parallel \mathbf{tick})\}\restriction A \\
= & \quad \{\text{Property 4.2.9, } \mathbf{c}C = \{pref.X \mid X \in \mu.\mathbf{c}C\}\} \\
 & \{X \mathbf{\ w\ t}(\mathbf{pr}T) \mid X \in \mathbf{c}C \cup \mathbf{d}C\}\restriction A \\
= & \quad \{\text{definitions of projection and of } \mathbf{c}P\} \\
 & \mathbf{c}P.
\end{array}
$$

□

Additionally, since $\mathbf{a}(\mathbf{pr}P) = A$ and $\mathbf{a}(\mathbf{pr}(sys.P)) = A$, on account of Lemma 3.2.11 we have

$$\mathcal{M}[\![P]\!] = \mathcal{A}[\![sys.P]\!].$$

Hence, $sys.P$ is a correct implementation of P.

Observe that $(\mathbf{pr}P, \mathbf{c}P)$ belongs to Σ. Additionally, we define

$$\mathbf{ac}P = \mathbf{ac}(\mathbf{pr}P, \mathbf{c}P).$$

On programs a relation is defined that is based on the abstract model of systems described in Chapter 3.
We say that program P implements (is at least as good as) program Q, which is denoted by $P \sqsupseteq Q$, if

$$\mathcal{M}[\![P]\!] \sqsupseteq \mathcal{M}[\![Q]\!].$$

We say that program P is equivalent to program Q if

$$P \sqsupseteq Q \wedge Q \sqsupseteq P.$$

An alternative definition of $\mathbf{pr}P$ and $\mathbf{c}P$ takes as starting point the slightly changed definitions of completed and deadlocked computations of the previous section. Namely, $0{\cdot}a\ 1{\cdot}a$ is replaced by a in the definition of $\mathbf{c}a$. Furthermore, in the definitions of the completed and deadlocked computations of $C_0 \parallel C_1$, $\delta.\mathbf{a}C_0$ and $\delta.\mathbf{a}C_1$ are replaced by $\mathbf{a}C_0$ and $\mathbf{a}C_1$, respectively. Then

$$\mathbf{pr}P = \langle A, (\cup X : X \in \mathbf{c}C \cup \mathbf{d}C : X\restriction A)\rangle.$$

and

$$\mathbf{c}P = \{X\restriction A \mid X \in \mathbf{c}C \cup \mathbf{d}C\}.$$

These expressions define exactly the same processes and the same sets of sets of traces as those at the beginning of this section, and they are much easier in use. The original definitions are, however, necessary to achieve the results of the previous section.
An immediate consequence of the above definitions is the following property.

Property 4.3.2

For programs $P = A.C$ and $Q = A.C'$, such that $\mathbf{c}C \cup \mathbf{d}C = \mathbf{c}C' \cup \mathbf{d}C'$,

$$\mathbf{c}P = \mathbf{c}Q.$$

□

The above property and Lemma 3.2.11 yield Corollary 4.3.3.

Corollary 4.3.3

Programs $A.C$ and $A.C'$, such that $\mathbf{c}C \cup \mathbf{d}C = \mathbf{c}C' \cup \mathbf{d}C'$, are equivalent.

□

In the following examples we compare a few Tangram programs.

Example 4.3.4

Consider two programs $P = \{a, b\}.(a \sqcap b) \parallel a$ and $Q = \{a, b\}.a \sqcap b$. We show that P is equivalent to Q.
For P we derive

$$
\begin{aligned}
& \mathbf{c}((a \sqcap b) \parallel a) \\
= \quad & \{\text{parallel composition}\} \\
& \{X_0 \mathbin{\mathbf{w}} X_1 \mid X_0 \in \mathbf{c}(a \sqcap b) \wedge X_1 \in \mathbf{c}a \wedge \\
& \qquad (\exists t : t \in X_0 \mathbin{\mathbf{w}} X_1 : t\restriction\{a, b\} \in \mu.X_0 \wedge t\restriction a \in \mu.X_1)\} \\
= \quad & \{\text{selection, } \mathbf{c}a = \{\{\varepsilon, a\}\},\ \mu.\{\varepsilon, a\} = \{a\}\} \\
& \{X_0 \mathbin{\mathbf{w}} \{\varepsilon, a\} \mid (X_0 = \{\varepsilon, a\} \vee X_0 = \{\varepsilon, b\}) \wedge \\
& \qquad (\exists t : t \in X_0 \mathbin{\mathbf{w}} \{\varepsilon, a\} : t\restriction\{a, b\} \in \mu.X_0 \wedge t\restriction a = a)\} \\
= \quad & \{\text{definition } \mathbf{w} \text{ for computations, } \mathbf{a}(a \sqcap b) = \{a, b\}\} \\
& \{\{\varepsilon, a\}\}
\end{aligned}
$$

and

$$
\begin{aligned}
& \mathbf{d}((a \sqcap b) \parallel a) \\
= \quad & \{\text{parallel composition, } \mathbf{d}a = \varnothing \text{ and } \mathbf{d}b = \varnothing, \text{ hence } \mathbf{d}(a \sqcap b) = \varnothing\} \\
& \{X_0 \mathbin{\mathbf{w}} X_1 \mid X_0 \in \mathbf{c}(a \sqcap b) \wedge X_1 \in \mathbf{c}a \wedge \\
& \qquad (\forall t : t \in X_0 \mathbin{\mathbf{w}} X_1 : t\restriction\{a, b\} \notin \mu.X_0 \vee t\restriction a \notin \mu.X_1)\} \\
= \quad & \{\text{selection, } \mathbf{c}a = \{\{\varepsilon, a\}\},\ \mu.\{\varepsilon, a\} = \{a\}\} \\
& \{X_0 \mathbin{\mathbf{w}} \{\varepsilon, a\} \mid (X_0 = \{\varepsilon, a\} \vee X_0 = \{\varepsilon, b\}) \wedge \\
& \qquad (\forall t : t \in X_0 \mathbin{\mathbf{w}} \{\varepsilon, a\} : t\restriction\{a, b\} \notin \mu.X_0 \vee t\restriction a \neq a)\} \\
= \quad & \{\text{definition } \mathbf{w} \text{ for computations, } \mathbf{a}(a \sqcap b) = \{a, b\}\} \\
& \{\{\varepsilon, b\}\}.
\end{aligned}
$$

For Q we derive

$$
\begin{aligned}
& \mathbf{c}(a \sqcap b) \\
= & \quad \{\text{selection}\} \\
& \mathbf{c}a \cup \mathbf{c}b \\
= & \quad \{\mathbf{c}a = \{\{\epsilon, a\}\}, \mathbf{c}b = \{\{\epsilon, b\}\}\} \\
& \{\{\epsilon, a\}, \{\epsilon, b\}\}
\end{aligned}
$$

and

$$
\begin{aligned}
& \mathbf{d}(a \sqcap b) \\
= & \quad \{\text{selection}\} \\
& \mathbf{d}a \cup \mathbf{d}b \\
= & \quad \{\mathbf{d}a = \emptyset, \mathbf{d}b = \emptyset\} \\
& \emptyset.
\end{aligned}
$$

Hence,

$$\mathbf{c}((a \sqcap b) \parallel a) \cup \mathbf{d}((a \sqcap b) \parallel a) = \mathbf{c}(a \sqcap b) \cup \mathbf{d}(a \sqcap b).$$

Since P and Q also have equal alphabets, on account of Corollary 4.3.3, P is equivalent to Q.

□

Example 4.3.5

We establish that for $N \geq 1$ programs

$$P = A.\#N[\#[C]]$$

and

$$Q = A.\#[C]$$

are equivalent.

For completed computations we derive

$$
\begin{aligned}
& \mathbf{c}\#N[\#[C]] \\
= & \quad \{\text{finite repetition}\} \\
& \{pref.X \mid X \in (\mu.\mathbf{c}\#[C])^N\} \\
= & \quad \{\mathbf{c}\#[C] = \emptyset, \text{ hence also } \mu.\mathbf{c}\#[C] = \emptyset,\ N \geq 1 \Rightarrow \emptyset^N = \emptyset\} \\
& \emptyset \\
= & \quad \{\text{infinite repetition}\} \\
& \mathbf{c}\#[C].
\end{aligned}
$$

For deadlocked computations we derive

$$
\begin{array}{ll}
 & \mathbf{d}\#N[\#[C]] \\
= & \quad \{\text{finite repetition}\} \\
 & \{pref.(X +\!\!+ X') \mid (\exists i : 0 \leq i < N : X \in (\mu.\mathbf{c}\#[C])^i \wedge X' \in \mathbf{d}\#[C]\} \\
= & \quad \{\mu.\mathbf{c}\#[C] = \varnothing, \text{ hence the only possibility is } i = 0, \varnothing^0 = \{\{\varepsilon\}\}\} \\
 & \{pref.X \mid X \in \mathbf{d}\#[C]\} \\
= & \quad \{\text{deadlocked computations are prefix-closed}\} \\
 & \mathbf{d}\#[C].
\end{array}
$$

Because the alphabets of P and Q are equal as well, the programs are equivalent (on account of Corollary 4.3.3).

□

Example 4.3.6

We prove by induction that for a natural number $N \geq 1$,

$$P = A.\#N[C] \qquad \text{and} \qquad Q = A.C^N$$

are equivalent. We actually prove that

$$\mathbf{c}\#N[C] = \mathbf{c}(C^N)$$

and

$$\mathbf{d}\#N[C] = \mathbf{d}(C^N),$$

for $N \geq 1$.

Base $N = 1$

We derive

$$
\begin{array}{ll}
 & \mathbf{c}\#1[C] \\
= & \quad \{\text{finite repetition}\} \\
 & \{pref.X \mid X \in \mu.\mathbf{c}C\} \\
= & \quad \{\text{for } X \in \mathbf{c}C \text{ we have } X = pref.(\mu.X)\} \\
 & \mathbf{c}C
\end{array}
$$

and

$$
\begin{array}{ll}
& \mathbf{d}\#1[C] \\
= & \quad \{\text{finite repetition}\} \\
& \{pref.X \mid X \in \mathbf{d}C\} \\
= & \quad \{\text{deadlocked computations are prefix-closed}\} \\
& \mathbf{d}C.
\end{array}
$$

(Hence, on account of Corollary 4.3.3, $A.\#1[C]$ is equivalent to $A.C$).

Step $N \geq 1$

The induction hypothesis is

$$\mathbf{c}\#N[C] = \mathbf{c}(C^N)$$

and

$$\mathbf{d}\#N[C] = \mathbf{d}(C^N).$$

For completed computations we derive

$$
\begin{array}{ll}
& \mathbf{c}\#(N+1)[C] \\
= & \quad \{\text{finite repetition}\} \\
& \{pref.X \mid X \in (\mu.\mathbf{c}C)^{N+1}\} \\
= & \quad \{\text{definition } A^i\} \\
& \{pref.(X +\!\!+ X') \mid X \in (\mu.\mathbf{c}C)^N \wedge X' \in \mu.\mathbf{c}C\} \\
= & \quad \{(\mu.\mathbf{c}C)^N = \mu.\mathbf{c}\#N[C]\} \\
& \{pref.(X +\!\!+ X') \mid X \in \mu.\mathbf{c}\#N[C] \wedge X' \in \mu.\mathbf{c}C\} \\
= & \quad \{\text{induction hypothesis (holds also for sets of maximal traces)}\} \\
& \{pref.(X +\!\!+ X') \mid X \in \mu.\mathbf{c}(C^N) \wedge X' \in \mu.\mathbf{c}C\} \\
= & \quad \{\text{sequential composition}\} \\
& \mathbf{c}(C^N ; C) \\
= & \quad \{\text{associativity of } ;\} \\
& \mathbf{c}(C^{N+1}).
\end{array}
$$

For deadlocked computations we derive

$$
\begin{array}{ll}
& \mathbf{d}\#(N+1)[C] \\
= & \quad \{\text{finite repetition}\} \\
& \{pref.(X +\!\!+ X') \mid (\exists i : 0 \leq i < N+1 : X \in (\mu.\mathbf{c}C)^i) \wedge X' \in \mathbf{d}C\}
\end{array}
$$

$$
\begin{array}{ll}
= & \{\text{predicate calculus}\} \\
& \{pref.(X +\!\!+ X') \mid ((\exists i : 0 \le i < N : X \in (\mu.\mathbf{c}C)^i) \wedge X' \in \mathbf{d}C) \\
& \qquad\qquad \vee (X \in (\mu.\mathbf{c}C)^N \wedge X' \in \mathbf{d}C)\} \\
= & \{\text{predicate and set calculus}\} \\
& \{pref.(X +\!\!+ X') \mid (\exists i : 0 \le i < N : X \in (\mu.\mathbf{c}C)^i) \wedge X' \in \mathbf{d}C\} \\
& \cup \{pref.(X +\!\!+ X') \mid X \in (\mu.\mathbf{c}C)^N \wedge X' \in \mathbf{d}C\} \\
= & \{\text{finite repetition}\} \\
& \mathbf{d}\#N[C] \cup \{pref.(X +\!\!+ X') \mid X \in (\mu.\mathbf{c}C)^N \wedge X' \in \mathbf{d}C\} \\
= & \{\text{induction hypothesis (for deadlocked computations),} \\
& \quad (\mu.\mathbf{c}C)^N = \mu.\mathbf{c}\#N[C]\} \\
& \mathbf{d}(C^N) \cup \{pref.(X +\!\!+ X') \mid X \in \mu.\mathbf{c}\#N[C] \wedge X' \in \mathbf{d}C\} \\
= & \{\text{induction hypothesis (for completed computations, holds also for sets} \\
& \quad \text{of maximal traces)}\} \\
& \mathbf{d}(C^N) \cup \{pref.(X +\!\!+ X') \mid X \in \mu.\mathbf{c}(C^N) \wedge X' \in \mathbf{d}C\} \\
= & \{\text{sequential composition}\} \\
& \mathbf{d}(C^N ; C) \\
= & \{\text{associativity of } ;\} \\
& \mathbf{d}(C^{N+1}).
\end{array}
$$

(Hence, on account of Corollary 4.3.3, $A.\#(N+1)[C]$ and $A.C^{N+1}$ are equivalent.)

□

The next example shows that parallel composition is not modelled by interleaving.

Example 4.3.7

Consider two programs $P = \{a, b\}.a \parallel b$ and $Q = \{a, b\}.a ; b \sqcap b ; a$.
For P we derive

$$
\begin{array}{ll}
& \mathbf{c}(a \parallel b) \\
= & \{\text{parallel composition}\} \\
& \{X_0 \mathbf{w} X_1 \mid X_0 \in \mathbf{c}a \wedge X_1 \in \mathbf{c}b \wedge \\
& \qquad (\exists t : t \in X_0 \mathbf{w} X_1 : t\lceil a \in \mu.X_0 \wedge t\lceil b \in \mu.X_1)\} \\
= & \{\mathbf{c}a = \{\{\varepsilon, a\}\} \text{ and } \mathbf{c}b = \{\{\varepsilon, b\}\}, \text{ definition } \mathbf{w} \text{ for computations}, \\
& \quad \mathbf{a}a = \{a\} \text{ and } \mathbf{a}b = \{b\}\} \\
& \{pref.\{ab, ba\}\}
\end{array}
$$

and

$$
\begin{array}{ll}
& \mathbf{d}(a \parallel b) \\
= & \quad \{\text{parallel composition, } \mathbf{d}a = \emptyset \text{ and } \mathbf{d}b = \emptyset\} \\
& \{X_0 \mathbf{w} X_1 \mid X_0 \in \mathbf{c}a \wedge X_1 \in \mathbf{c}b \wedge \\
& \qquad (\forall t : t \in X_0 \mathbf{w} X_1 : t\lceil a \notin \mu.X_0 \vee t\lceil b \notin \mu.X_1)\} \\
= & \quad \{\mathbf{c}a = \{\{\varepsilon, a\}\} \text{ and } \mathbf{c}b = \{\{\varepsilon, b\}\}, \text{ definition } \mathbf{w} \text{ for computations}, \\
& \quad \mathbf{a}a \cap \mathbf{a}b = \emptyset\} \\
& \emptyset.
\end{array}
$$

For Q we derive

$$
\begin{array}{ll}
& \mathbf{c}(a\,;\, b \sqcap b\,;\, a) \\
= & \quad \{\text{selection}\} \\
& \mathbf{c}(a\,;\, b) \cup \mathbf{c}(b\,;\, a) \\
= & \quad \{\mathbf{c}(a\,;\, b) = \{pref.\{ab\}\},\ \mathbf{c}(b\,;\, a) = \{pref.\{ba\}\}\} \\
& \{pref.\{ab\}, pref.\{ba\}\}
\end{array}
$$

and

$$
\begin{array}{ll}
& \mathbf{d}(a\,;\, b \sqcap b\,;\, a) \\
= & \quad \{\text{selection}\} \\
& \mathbf{d}(a\,;\, b) \cup \mathbf{d}(b\,;\, a) \\
= & \quad \{\mathbf{d}(a\,;\, b) = \emptyset,\ \mathbf{d}(b\,;\, a) = \emptyset\} \\
& \emptyset.
\end{array}
$$

Then we have

$$\mathbf{pr}P = \langle \{a, b\}, pref.\{ab, ba\} \rangle = \mathbf{pr}Q$$

and

$$\mathbf{ac}P \supseteq \mathbf{ac}Q \qquad \text{and} \qquad \mathbf{ac}Q \not\supseteq \mathbf{ac}P.$$

Hence, on account of the definition of $\mathcal{M}[\![P]\!]$ and Definition 3.2.10, $P \sqsupseteq Q$ and $Q \not\sqsupseteq P$.

□

5 External behaviour and fairness assumptions

In the preceding chapters two models of trace theory systems have been introduced allowing reasoning about concurrency and nondeterminism. In particular, the distinction between these phenomena is made explicit in both models. A direct consequence of this distinction is that in many cases we can prove desired properties of systems without referring to fairness. In fact, our partial order computations from Chapter 2 are always fair according to [Mal]. We claim that fairness assumptions are necessary if properties have to be proved that depend not only on the structure of systems but also on other aspects, such as the treatment of choices in the system components (processes). In this chapter we present a model of systems that allows explicit assumptions about fairness (or justice) and their impact on external behaviour of systems. This is an abstract and not compositional model for systems of the restricted class defined in Section 3.2.

In Section 5.1, we shortly discuss fairness problems. Section 5.2 presents our abstract model incorporating fairness. In Section 5.3, the classical example of the Alternating Bit Protocol is addressed.

5.1 A short introduction to fairness

The concept of fairness is connected to that of nondeterminism. Fairness properties can be seen as restrictions on infinite executions of systems: if there is a recurring choice out of a few alternatives, none of them should be constantly ignored. Fairness is a convenient abstraction from (irrelevant) implementation details.

Fairness properties (also known as fairness assumptions) are used for two purposes: in the definition of the semantics of programming languages and in the descriptions of the behaviour of systems. A review of the literature on fairness together with a classification of fairness properties is presented in [Paw]. In [Fra], the use of fairness assumptions for proving termination of programs is extensively explored.

In [Paw], two classes of fairness concepts are distinguished: general and specific fairness properties. General fairness properties are mostly associated with constructs of programming languages. An example of such a property can be the assumption that any guard of a program being enabled permanently is chosen infinitely often. This property

is called weak fairness, or justice. If "permanently" is replaced by "infinitely often", we get the property known under the name strong fairness. General fairness properties are used for defining models and proof systems for programming languages. Specific fairness properties are related to the behaviour of a specific system or program under study. An example might be the assumption that the system is fair with respect to a certain action in the sense that it cannot persistently choose other actions if this action is permanently enabled. In this thesis we are concerned with specific fairness properties only, although in this context general fairness would be a special case of specific fairness. Namely, when a system is regarded to be fair with respect to all its actions. We refer to the above specific fairness property as justice. Again, if "permanently" is replaced by "infinitely often", we get a specific fairness property referred to as strong fairness, or simply fairness. The choice of these names for the two mentioned specific fairness properties is motivated by the direct association with the general concept of justice and strong fairness.

5.2 Abstract model

In our abstract model with fairness we concentrate on the question of what the external behaviour of a system is if the system is fair with respect to some explicitly chosen actions. To specify the set of fairly treated actions an extended notion of system is introduced.

An *f-system* (fair system) is a triple $\langle A, Y, F\rangle$, where $\langle A, Y\rangle \in RD$ and $F \subseteq \mathbf{a}(\mathbf{W}.Y)$. As before, A is the external alphabet of the f-system. Set F contains actions that are treated fairly by the f-system and it is called *fairness set* of the f-system. Notice that we do not require $F \subseteq A$. For f-system S we denote its external alphabet by $\mathbf{e}S$, its set of processes by $\mathbf{p}S$ and its fairness set by $\mathbf{fs}S$.

For f-system S we define

$$pot.S = pot.\langle \mathbf{e}S, \mathbf{p}S\rangle.$$

In Section 3.2 we defined computations of system S. They are derived from $pot.S$. In this chapter we present two abstract models of f-systems for two kinds of fairness, namely weak fairness (justice) and strong fairness. In the sequel we write fairness for strong fairness.

In the abstract model with justice *just computations* are of importance. Just computations of f-system S are derived from just elements of $pot.S$. We first explain what unjust means with respect to elements of $pot.S$. Element p of $pot.S$ is unjust if there is a trace t in p and an action a in $\mathbf{fs}S$ which is permanently enabled in p after t — expressed by

$$(\forall u : u \in p \wedge t \leq u : ua \in \mathbf{t}S),$$

and which does not occur in p after t — expressed by

$$(\forall u : u \in p \wedge t \leq u : ua \notin p).$$

We are interested in those elements of $pot.S$ that are not unjust. The set of just elements of $pot.S$, denoted by $jpot.S$, is defined as follows.

Definition 5.2.1

Let S be an f-system and let $p \in pot.S$. Then

$$p \in jpot.S$$
$$\Leftrightarrow$$
$$(\forall t, a : t \in p \wedge a \in \mathbf{fs}S : (\exists u : u \in p \wedge t \leq u : ua \notin \mathbf{t}S \vee ua \in p)).$$

□

From $jpot.S$ just computations, denoted by $\mathbf{jc}S$, are derived.

Definition 5.2.2

For f-system S

$$\mathbf{jc}S = jpot.S\restriction \mathbf{e}S.$$

□

The abstract model with justice of f-system S is defined to be the pair $(\mathbf{pr}S, \mathbf{jc}S)$ and it is denoted by $\mathcal{J}[\![S]\!]$.

Similarly, in the abstract model with fairness *fair computations* are of importance. Fair computations of f-system S are derived from fair elements of $pot.S$. As before, we explain what the opposite means. Element p of $pot.S$ is unfair if there is a trace t in p and an action a in $\mathbf{fs}S$ which is infinitely often enabled in p after t — expressed by

$$(\forall u : u \in p \wedge t \leq u : (\exists v : v \in p \wedge u \leq v : va \in \mathbf{t}S)),$$

and which does not occur in p after t. We concentrate on those elements of $pot.S$ that are not unfair. The set of fair elements of $pot.S$ is denoted by $fpot.S$, and it is defined as follows.

Definition 5.2.3

Let S be an f-system and let $p \in pot.S$. Then

$$
\begin{array}{l}
\quad p \in \mathit{fpot}.S \\
\Leftrightarrow \\
\quad (\forall t, a : t \in p \wedge a \in \mathbf{fs}S \\
\qquad : (\exists u : u \in p \wedge t \leq u : (\forall v : v \in p \wedge u \leq v : va \notin \mathbf{t}S) \vee ua \in p)).
\end{array}
$$

□

From $\mathit{fpot}.S$ fair computations, denoted by $\mathbf{fc}S$, are derived.

Definition 5.2.4

For f-system S

$$\mathbf{fc}S = \mathit{fpot}.S \restriction \mathbf{e}S.$$

□

The abstract model with fairness of f-system S is defined to be the pair $(\mathbf{pr}S, \mathbf{fc}S)$ and it is denoted by $\mathcal{F}[\![S]\!]$.
For f-system S we have $\mathcal{J}[\![S]\!] \in \Sigma$ and $\mathcal{F}[\![S]\!] \in \Sigma$.
We say that f-system S_0 is a just implementation of system S_1 if

$$\mathcal{J}[\![S_0]\!] \sqsupseteq \mathcal{A}[\![S_1]\!].$$

Respectively, we say that f-system S_0 is a fair implementation of system S_1 if

$$\mathcal{F}[\![S_0]\!] \sqsupseteq \mathcal{A}[\![S_1]\!].$$

As one would expect, all finite computations are fair and just.
The following property proves that every fair computation is also just.

Property 5.2.5

For f-system S

$$\mathit{fpot}.S \subseteq \mathit{jpot}.S \subseteq \mathit{pot}.S.$$

Proof

Let $p \in \mathit{pot}.S$. From Definition 5.2.1 we have $\mathit{jpot}.S \subseteq \mathit{pot}.S$.
We derive

$$
\begin{array}{ll}
& p \in \mathit{fpot}.S \\
\Leftrightarrow & \{\text{Definition 5.2.3}\} \\
& (\forall t, a : t \in p \wedge a \in \mathbf{fs}S \\
& \quad : (\exists u : u \in p \wedge t \leq u : (\forall v : v \in p \wedge u \leq v : va \notin \mathbf{t}S) \vee ua \in p)) \\
\Rightarrow & \{\text{predicate calculus}\} \\
& (\forall t, a : t \in p \wedge a \in \mathbf{fs}S : (\exists u : u \in p \wedge t \leq u : ua \notin \mathbf{t}S \vee ua \in p)) \\
\Leftrightarrow & \{\text{Definition 5.2.1}\} \\
& p \in \mathit{jpot}.S.
\end{array}
$$

□

As a consequence, we have for f-system S

$$\mathbf{fc}S \subseteq \mathbf{jc}S.$$

If the fairness set is empty, all computations are fair and just. Hence the following property.

Property 5.2.6

For system S

$$\mathcal{A}[\![S]\!] = \mathcal{J}[\![(\mathbf{e}S, \mathbf{p}S, \emptyset)]\!],$$

$$\mathcal{A}[\![S]\!] = \mathcal{F}[\![(\mathbf{e}S, \mathbf{p}S, \emptyset)]\!].$$

□

We illustrate our abstract model with incorporated justice by Example 5.2.7. The model with fairness is illustrated by Example 5.2.8. It is also used in the next section for the verification of a communication protocol.

Example 5.2.7

Consider system

$$S_0 = (\{a\}, \{\mathbf{pr}a\})$$

and f-system

$$S_1 = (\{a\}, \{\mathbf{pr}(x^*; x'; a)\}, \{x'\}).$$

We show that S_1 is a just implementation of S_0.
Both systems belong to RD. External processes of S_0 and S_1 are both equal to $\mathbf{pr}a$. System S_0 has only one computation: $\{\varepsilon, a\}$. Maximal po-traces of S_1 are

$$pot.S_1 = \{\mathbf{t}(\mathbf{pr}(x^i\,;\, x'\,;\, a)) \mid i \geq 0\} \cup \{\mathbf{t}(\mathbf{pr}(x^*))\}.$$

The only not just element of $pot.S_1$ is $p = \mathbf{t}(\mathbf{pr}(x^*))$, because

$$(\forall u : u \in p : ux' \in \mathbf{t}S_1 \wedge ux' \notin p).$$

(Note that, on account of Property 5.2.5, this particular p is also not fair.) Hence, the set of just elements of $pot.S_1$ equals

$$jpot.S_1 = \{\mathbf{t}(\mathbf{pr}(x^i\,;\, x'\,;\, a)) \mid i \geq 0\}.$$

As a consequence, the set of just computations consists of one element: $\{\varepsilon, a\}$. (Every element of $jpot.S_1$ also satisfies the requirements of being fair. Hence, $\mathbf{fc}S_1 = \mathbf{jc}S_1$.) As $\mathbf{pr}S_1 = \mathbf{pr}S_0$ and $\mathbf{jc}S_1 = \mathbf{c}S_0$ (and $\mathbf{fc}S_1 = \mathbf{c}S_0$) we can conclude that S_1 is a just (and fair) implementation of S_0. In every just computation action a is guaranteed to occur.

□

Example 5.2.8

Consider processes

$$\begin{aligned} T = \langle &\{q, n, y, x, x', z, z'\} \cup \{x_i \mid i \geq 0\}, \\ &\{t \mid (\exists i, j : i \geq 0 \wedge j \geq 0 : t \leq z'x_i(qx'n)^i x(qy)^j) \\ &\quad \vee (\exists i, j : i \geq 0 \wedge j \geq 0 : t \leq z'x_{i+1}(qx'n)^i qxx'n(qy)^j) \\ &\quad \vee (\exists i : i \geq 0 : t \leq z(qn)^i)\}\rangle, \\ U = \mathbf{pr}(z \mid z'\,;\, a\,;\, x) & \end{aligned}$$

and system

$$S = \langle \{a, q, n, y\}, \{T, U\}\rangle.$$

Process T models a synchronizer. By means of q the environment (which can be some other process or system) can inquire whether event x already took place. The answer is positive (y) if event x occurred prior to q, otherwise it is negative (n). (Note that T is not regular.)
Process U determines whether event x will happen. It makes a choice between z and z'. If z is chosen event x will not occur. If z' is chosen, after a event x will take place.

System S belongs to RD, because every choice is between internal actions (x' is introduced only for this purpose).

Computations of S are

$$\mathbf{c}S = \{\mathbf{t}(\mathbf{pr}((q\,;\,n)^*))\} \cup \{\mathbf{t}(\mathbf{pr}(a\,,\,(q\,;\,n)^i\,;\,(q\,;\,y)^*)) \mid i \geq 0\}.$$

Let $T' = \mathbf{pr}(((x''\,;\,q\,;\,x'\,;\,n)^*\,;\,x \mid x''\,;\,q\,;\,(x'\,;\,n\,;\,x''\,;\,q)^*\,;\,x\,;\,n)\,;\,(q\,;\,y)^*)$. In the sequel we show that f-system

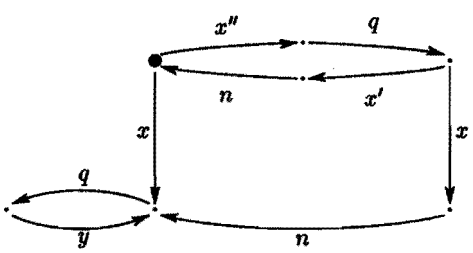
$$S' = \langle\{a, q, n, y\}, \{T', U\}, \{x\}\rangle$$

is a fair implementation of S.

System S' belongs to RD, because every choice is between internal actions (x'' and x' are introduced only for this purpose).

Process T' is a kind of a regular approximation of the synchronizer. It has the same set of traces when actions x, q, n, and y are concerned. It has the advantage of being regular and the disadvantage of being divergent with respect to x.

The state graph of T' is



In the context of T', if a is executed by process U, occurrence of event x depends on the behaviour of T' with respect to the choice between x and x'', and between x and x'. We are interested in showing that the occurrence of a will eventually lead to the answer y under the assumption that T' is fair with respect to x. The fairness set of S' is chosen to equal $\{x\}$ to formalize this assumption.

External processes of S and S' are equal.

Sets of traces belonging to partial order computations of S' are members of

$$\begin{aligned} pot.S' = \{&\mathbf{t}(\mathbf{pr}(z\,,\,(x''\,;\,q\,;\,x'\,;\,n)^*)), \mathbf{t}(\mathbf{pr}((z'\,;\,a)\,,\,(x''\,;\,q\,;\,x'\,;\,n)^*))\} \\ &\cup \{\mathbf{t}(\mathbf{pr}((z'\,;\,a)\,,\,(x''\,;\,q\,;\,x'\,;\,n)^i\,;\,x\,;\,(q\,;\,y)^*)) \mid i \geq 0\} \\ &\cup \{\mathbf{t}(\mathbf{pr}((z'\,;\,a)\,,\,(x''\,;\,q\,;\,(x'\,;\,n\,;\,x''\,;\,q)^i\,;\,x\,;\,n\,;\,(q\,;\,y)^*)) \mid i \geq 0\}. \end{aligned}$$

The only not fair element of $pot.S'$ is $p = \mathbf{t}(\mathbf{pr}((z';a),(x'';q;x';n)^*))$. This is because $z'a \in p$ and

$$(\forall u : u \in p \wedge z'a \leq u : (\exists v : v \in p \wedge u \leq v : vx \in \mathbf{t}S') \wedge ux \notin p).$$

All other elements of $pot.S'$ do satisfy the requirement of Definition 5.2.3. Hence, the set of fair elements of $pot.S'$ is

$$fpot.S' = pot.S' \setminus \{\mathbf{t}(\mathbf{pr}((z';a),(x'';q;x';n)^*))\}.$$

The set of fair computations is

$$\mathbf{fc}S' = \{\mathbf{t}(\mathbf{pr}((q;n)^*))\} \cup \{\mathbf{t}(\mathbf{pr}(a,(q;n)^i;(q;y)^*)) \mid i \geq 0\}.$$

As $\mathbf{pr}S' = \mathbf{pr}S$ and $\mathbf{fc}S' = \mathbf{c}S$ we can conclude that S' is a fair implementation of S. In every fair computation containing a action y is guaranteed to occur (after a finite number of occurrences of n).

This analysis makes critical use of our fairness assumption. In the abstract model of Chapter 3 we would find computation $\mathbf{t}(\mathbf{pr}(a,(q;n)^*))$ in $\mathbf{c}S'$. In that model we, consequently, could not show that the execution of a will eventually lead to the answer y.

□

5.3 Alternating Bit Protocol

The Alternating Bit Protocol was introduced in [Bar] and has been extensively studied in the literature as a test example for various description formalisms. It is a simple data link protocol for error-free one-way communication over a medium that may corrupt messages. In this section we are only concerned with the communication behaviour of the protocol. Our presentation is essentially based on [Paw], with the exclusion of timeouts (we assume that the communication medium cannot lose messages). This simplification is not necessary because timeouts can be modelled by additional actions. However, the absence of timeouts makes the discussion less complex.

The external communication behaviour of the Alternating Bit Protocol is specified by

$$T = \langle \{s,r\}, \{\mathbf{pr}((s;r)^*)\} \rangle,$$

where s and r model the send and receive actions, respectively. System T belongs to RD.

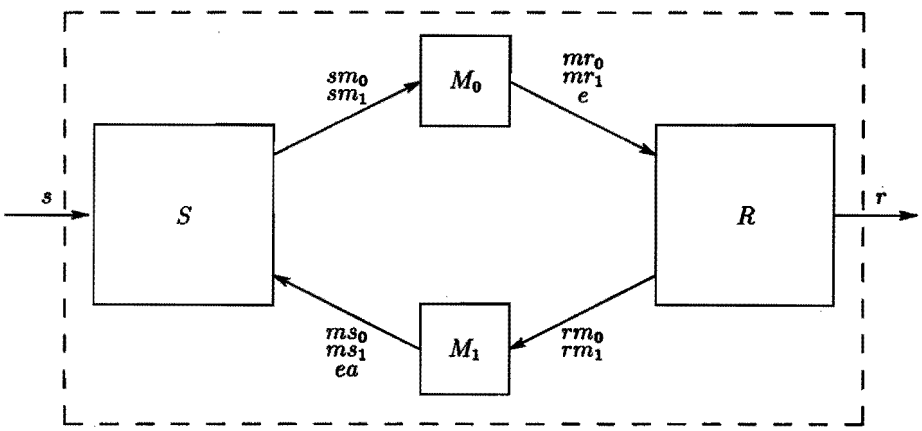


Figure 5.1: A scheme of the implementation of the Alternating Bit Protocol.

We have

$$\mathcal{A}[T] = (\mathbf{pr}((s\,;r)^*), \{\mathbf{t}(\mathbf{pr}((s\,;r)^*))\}).$$

The proposed implementation of the protocol consists of four components: sender S, receiver R and two communication media M_0 and M_1 (see Fig. 5.1). The sender and the receiver communicate via M_0 and M_1. Both media may corrupt but not lose or re-order messages. The sender accepts a message for transmission, which is modelled by action s, and adds one bit to it (starting with 0). The extended message is transmitted via M_0 to the receiver. This transmission is modelled by action sm_0. After reception of the correct acknowledgement ms_0 (with the same sequence bit) via M_1 a new message is accepted for transmission. This message gets the inverted sequence bit and the procedure is repeated with actions sm_1 and ms_1, respectively. If the sender receives an acknowledgement with the wrong sequence bit (ms_1 in the case when the preceding transmission action was sm_0, and ms_0 in the another case) or a corrupt acknowledgement (modelled by ea) it retransmits the message.

The formal specification of S is

$$S = \mathbf{pr}((s\,;\, sm_0\,;\, ((ms_1 \mid ea)\,;\, sm_0)^*\,;\, ms_0\,;\, s\,;\, sm_1\,;\, ((ms_0 \mid ea)\,;\, sm_1)^*\,;\, ms_1)^*).$$

The receiver R accepts messages from M_0, which is modelled by actions mr_0 and mr_1. Every message with the sequence bit different from that of the preceding message is delivered to the environment via r. Each arriving message is acknowledged via M_1 by

an acknowledge action (rm_0 or rm_1) consistent with the sequence bit of the message. If a corrupt message arrives (e), the last acknowledgement is retransmitted.
The formal specification of R is

$$R = \mathbf{pr}((mr_0\,;\, r\,;\, rm_0\,;\, ((mr_0 \mid e)\,;\, rm_0)^*\,;\, mr_1\,;\, r\,;\, rm_1\,;\, ((mr_1 \mid e)\,;\, rm_1)^*)^*).$$

(Notice that the main repetitions in the specifications of S and R consist of two parts that differ only in the interchange of zeros and ones.)
The medium M_0 accepts a message for transmission (sm_0 and sm_1), after which the message is delivered intact (mr_0 and mr_1) or corrupt (e). The medium M_1 has a similar behaviour with respect to acknowledgements. Accepting of acknowledgements for transmission is modelled by rm_0 and rm_1. Delivering of proper acknowledgements to the sender is modelled by ms_0 and ms_1; for corrupt acknowledgements action ea is used.
Formally, M_0 and M_1 are specified by

$$M_0 = \mathbf{pr}((sm_0\,;\, (mr_0 \mid e) \mid sm_1\,;\, (mr_1 \mid e))^*)$$

and

$$M_1 = \mathbf{pr}((rm_0\,;\, (ms_0 \mid ea) \mid rm_1\,;\, (ms_1 \mid ea))^*).$$

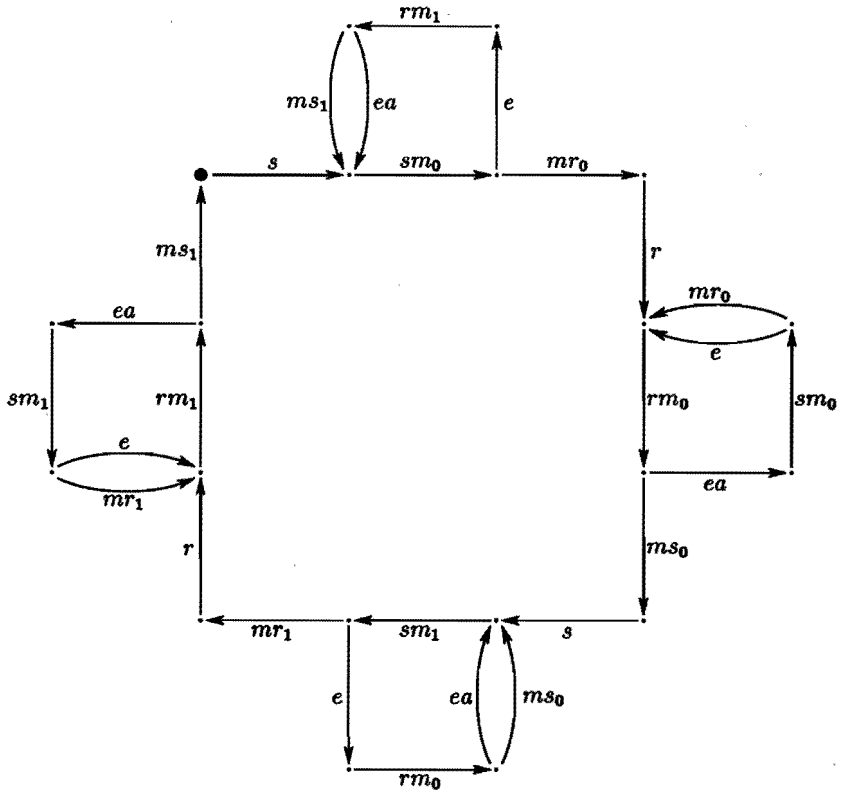
(To include timeouts, action to should be introduced as an additional alternative:

- in the sender S, in $(ms_1 \mid ea)$ and $(ms_0 \mid ea)$,
- in the medium M_0, in $(mr_0 \mid e)$ and $(mr_1 \mid e)$,
- in the medium M_1, in $(ms_0 \mid ea)$ and $(ms_1 \mid ea)$.)

For a start we consider the implementation of the Alternating Bit Protocol without any fairness assumptions.
Let $T' = \langle\{s,r\},\{S,R,M_0,M_1\}\rangle$. In order to facilitate further calculations we first determine the weave of S, R, M_0, and M_1, which is

$$\begin{aligned}\mathbf{pr}((\ & s\,;\, sm_0\,;\, (e\,;\, rm_1\,;\, (ms_1 \mid ea)\,;\, sm_0)^*\,;\, mr_0\\ & r\,;\, rm_0\,;\, (ea\,;\, sm_0\,;\, (mr_0 \mid e)\,;\, rm_0)^*\,;\, ms_0\\ & s\,;\, sm_1\,;\, (e\,;\, rm_0\,;\, (ms_0 \mid ea)\,;\, sm_1)^*\,;\, mr_1\\ & r\,;\, rm_1\,;\, (ea\,;\, sm_1\,;\, (mr_1 \mid e)\,;\, rm_1)^*\,;\, ms_1)^*).\end{aligned}$$

The state graph of the weave is



Note that all choices in the weave are only between internal actions (thus $T' \in RD$) that belong to the same process, viz., M_0 and M_1. These choices model the possibility that a medium corrupts a message or delivers it intact. External process of T' is equal to that of T.

In order to characterize $pot.T'$ we label four particular states of the weave

- $\alpha_0 = [s\ sm_0]$,
- $\alpha_1 = [s\ sm_0\ mr_0\ r\ rm_0]$,
- $\alpha_2 = [s\ sm_0\ mr_0\ r\ rm_0\ ms_0\ s\ sm_1]$,
- $\alpha_3 = [s\ sm_0\ mr_0\ r\ rm_0\ ms_0\ s\ sm_1\ mr_1\ r\ rm_1]$,

and define

- $E_0 = \{e\,rm_1\,ms_1\,sm_0, e\,rm_1\,ea\,sm_0\}$,
- $E_1 = \{ea\,sm_0\,mr_0\,rm_0, ea\,sm_0\,e\,rm_0\}$,
- $E_2 = \{e\,rm_0\,ms_0\,sm_1, e\,rm_0\,ea\,sm_1\}$,
- $E_3 = \{ea\,sm_1\,mr_1\,rm_1, ea\,sm_1\,e\,rm_1\}$,
- $E = \{t \mid t \in [\varepsilon]\}$.

Furthermore, for set X of traces, by $MC.X$ we denote the set of maximal chains of $(X, \leq)$. That is,

$$\begin{array}{l} Y \in MC.X \\ \Leftrightarrow \\ Y \subseteq X \wedge (\forall t,u : t \in Y \wedge u \in Y : t \leq u \vee u \leq t) \\ \wedge (\forall t : t \in X \setminus Y : (\exists u : u \in Y : \neg(t \leq u \vee u \leq t))). \end{array}$$

Then one can check that

$$\begin{array}{l} pot.T' = \{pref.\{ty \mid y \in K\} \mid 0 \leq j < 4 \wedge t \in \alpha_j \wedge K \in MC.E_j^*\} \cup \\ \qquad \{pref.K \mid K \in MC.E\}. \end{array}$$

According to Definition 3.2.10 we have that T' does not implement T. In particular,

$$\begin{array}{l} \mathcal{A}[\![T']\!] = (\mathbf{pr}((s\,;r)^*), \{\mathbf{t}(\mathbf{pr}((s\,;r)^i)) \mid i \geq 0\} \cup \{\mathbf{t}(\mathbf{pr}(s\,;(r\,;s)^i)) \mid i \geq 0\} \\ \qquad\qquad \cup \{\mathbf{t}(\mathbf{pr}((s\,;r)^*))\}). \end{array}$$

Hence, $\{\varepsilon\} \in \mathbf{c}T'$. This, in turn, implies that $(\varepsilon, \{s\}) \notin \mathbf{ac}T'$. For T we have that $(\varepsilon, \{s\})$ does belong to $\mathbf{ac}T$ and, hence, $\mathcal{A}[\![T']\!] \not\supseteq \mathcal{A}[\![T]\!]$.

We now discuss an implementation of the protocol that guarantees fairness of choices with respect to the actions mr_0, ms_0, mr_1, and ms_1. That is, we consider f-system T'' defined by

$$T'' = \langle \{s, r\}, \{S, R, M_0, M_1\}, \{mr_0, ms_0, mr_1, ms_1\} \rangle.$$

External processes of T'' and T are equal.

On account of Definition 5.2.3 and with the help of the state graph from the previous page, can be concluded that only infinite traces in which the initial state is repeatedly passed, belong to $fpot.T''$. Formally,

$$fpot.T'' = \{pref.K \mid K \in MC.E\}.$$

Hence, $\mathbf{fc}T'' = \{\mathbf{t}(\mathbf{pr}((s\,;\,r)^*))\}$. Furthermore, we have

$$\mathcal{F}[\![T'']\!] = (\mathbf{pr}((s\,;\,r)^*), \{\mathbf{t}(\mathbf{pr}((s\,;\,r)^*))\}).$$

Hence, we have that T'' is a fair implementation of T.
On account of Property 5.2.5 we have

$$\{pref.K \mid K \in MC.E\} \subseteq jpot.T''.$$

For every other $p \in pot.T''$ holds

$$(\forall t, a : t \in p \wedge a \in \mathbf{fs}T'' : (\exists u : u \in p \wedge t \leq u : ua \notin \mathbf{t}T''))$$

(see the state graph of S **w** R **w** M_0 **w** M_1). A conclusion is then that $jpot.T'' = pot.T''$. Hence, T'' is not a just implementation of T (see the argument for $\mathcal{A}[\![T']\!] \not\supseteq \mathcal{A}[\![T]\!]$).
The f-system T'' has four events (mr_0, ms_0, mr_1, and ms_1) in its fairness set. The fairness set of T'' expresses the property that in the infinite executions of T'' infinitely many messages with 0-bit and infinitely many messages with 1-bit are delivered to the receiver, and infinitely many acknowledgements with 0-bit and infinitely many acknowledgements with 1-bit are delivered to the sender. In [Paw], a different fairness property is used in the verification of the Alternating Bit Protocol. Namely, that infinitely many messages with 0-bit or infinitely many messages with 1-bit are delivered to the receiver, and infinitely many acknowledgements with 0-bit or infinitely many acknowledgements with 1-bit are delivered to the sender. These properties are associated with the communication media M_0 and M_1 instead of with the whole system, which is the case in our method. This is possible because the verification method of [Paw] is compositional.

Conclusions

This thesis proposes two new methods for verifying designs with respect to their specifications. Both specifications and designs are given in terms of trace theory systems.

The first method is operational. It comprises both safety and liveness considerations, and it distinguishes nondeterministic, but sequential systems from parallel ones. It is compositional in the following sense:

- if S is an implementation of T then $S{\uparrow}A$ is an implementation of $T{\uparrow}A$,
- if, additionally, S' is an implementation of T' then $S \parallel S'$ is an implementation of $T \parallel T'$.

However, the model on which this method is based is not compositional. To be more precise, we did not succeed in expressing $\mathcal{O}[\![S \parallel S']\!]$ in terms of $\mathcal{O}[\![S]\!]$ and $\mathcal{O}[\![S']\!]$. By Property 2.1.16 we know, on the other hand, that $\mathcal{O}[\![S{\uparrow}A]\!]$ equals $\mathcal{O}[\![S]\!]$.

The second method is restricted to a certain class of systems. As a consequence, the abstract model on which this method is based is compositional (i.e., $\mathcal{A}[\![S \parallel S']\!] = \mathcal{A}[\![S]\!] \parallel \mathcal{A}[\![S']\!]$ and $\mathcal{O}[\![S{\uparrow}A]\!] = \mathcal{A}[\![S]\!]{\uparrow}A$). The second method provides the same comparisons as the first one for the restricted system domain, hence, it also is compositional in the above sense. In the second method calculations are easier than in the first one. The model on which the second method is based turns out to be not fully abstract with respect to the operational model, because relation $\sqsupseteq$ is not antisymmetric. This model is not very much different from the failures model and yet it can express more liveness properties. Although the abstract model applies to a restricted system domain, it is rich enough to be used as a basis for defining a semantics of CSP-like languages, as it is shown for c-Tangram.

In the approach presented in this thesis parallelism is explicitly present in models used for comparing systems. The construction of the models follows, in broad outlines, that of [Hen]. However, the theory of [Hen] explains parallelism in terms of nondeterminism and interleaving. This is also the case in [Hoa], [Mil], and in related work. As a consequence, these theories identify systems like S_0 and S_1 discussed in the Introduction.

Additionally, the second method is extended — in a non-compositional way — to f-systems, which are systems with incorporated fairness requirements. In this manner,

the concept of fairness is strictly separated from the concept of parallelism. In this thesis, f-systems are used to describe only designs, but they can equally well be used to describe specifications.

Finally, we mention two major open questions.

- Is there another compositional model that is fully abstract with respect to the operational model on the whole system domain?
- Is there a compositional model for a system domain that includes f-systems?

A Appendix

This appendix contains some additional results in connection with the operational model of Chapter 2. Section A.1 describes several properties of partial-order computations. In Section A.2, the relationship between our operational model and nets is investigated. In Section A.3, a comparison of our model and an interleaving model of trace theory systems is presented. Section A.4 contains a non-trivial example. In Section A.5, a statement from Section 2.4 is proved.

A.1 Properties of partial-order computations

The following property shows that if (t,T) belongs to computation π then so does every configuration of process T with the trace being a prefix of t. Moreover, there is a path inside π from such a configuration to (t,T).

Property A.1.1

Let π be a computation of system S. Then

$$(t,T) \in Conf.\pi \;\Rightarrow\; (\forall t' : t' < t : (t',T) \in Conf.\pi \wedge (t',T) \rightarrow_\pi^* (t,T)).$$

Proof

The property is proved by induction on $\ell.t$ for $t \in \mathbf{t}T$.

Base $\quad \ell.t = 0$

Then $t = \epsilon$ and the universal quantification is trivially true.

Step $\quad \ell.t \neq 0$

Then $t \neq \epsilon$. Let $t = ua$. We derive

$$\begin{array}{ll} & (ua,T) \in Conf.\pi \\ \Rightarrow & \{\text{Definitions 2.1.13 and 2.1.11 — } \pi \text{ is admissible}\} \\ & (\exists C : (a,C) \in Act.\pi : (u,T) \in C \wedge C \subseteq \pi) \end{array}$$

$\Rightarrow$ { Definitions 2.1.7 and 2.1.5, set and predicate calculus }
$(\exists C : (a, C) \in Act.\pi : (u, T) \rightarrow_\pi (a, C) \rightarrow_\pi (ua, T)) \wedge (u, T) \in Conf.\pi$
$\Rightarrow$ { definition of $\rightarrow_\pi^*$, predicate calculus }
$(u, T) \rightarrow_\pi^* (ua, T) \wedge (u, T) \in Conf.\pi$
$\Rightarrow$ { induction hypothesis }
$(\forall t' : t' < u : (t', T) \in Conf.\pi \wedge (t', T) \rightarrow_\pi^* (u, T)) \wedge (u, T) \rightarrow_\pi^* (ua, T)$
$\Rightarrow$ { $\rightarrow_\pi^*$ is transitive, predicate calculus }
$(\forall t' : t' < ua : (t', T) \in Conf.\pi \wedge (t', T) \rightarrow_\pi^* (ua, T))$.

□

The next lemma shows that traces from two configurations of the same process that belong to one computation are related to each other. Namely, one of them is a prefix of the other one.

Lemma A.1.2

For computation π of system S, and configurations (t, T) and (u, T) of π

$$t \leq u \vee u \leq t.$$

Proof

The proof consists in showing that $(\forall v, a, b : va \leq t \wedge vb \leq u : a = b)$.
We derive

$va \leq t \wedge vb \leq u$
$\Rightarrow$ { Property A.1.1 }
$\{(va, T), (vb, T)\} \subseteq Conf.\pi$
$\Rightarrow$ { Definitions 2.1.13 and 2.1.11 — π is admissible }
$(\exists C, C' : \{(a, C), (b, C')\} \subseteq Act.\pi : (v, T) \in C \cap C')$
$\Rightarrow$ { Definition 2.1.13 — no forward branched configurations in π, Property 2.1.12 }
$a = b$.

□

A consequence of Property A.1.1 and Lemma A.1.2 is that in a computation configurations belonging to the same process are totally ordered, which is expressed in the following corollary.

Corollary A.1.3

For computation π of system S, and configurations (t,T) and (u,T) of π

$$(t,T) \rightarrow_\pi^* (u,T) \vee (u,T) \rightarrow_\pi^* (t,T).$$

□

Property A.1.4 shows that the set of traces associated with a computation is prefix-closed.

Property A.1.4

For computation π of system S, $\mathbf{tr}.\pi$ is prefix-closed.

Proof

We derive

$$\begin{array}{ll} & ua \in \mathbf{tr}.\pi \\ \Rightarrow & \{\text{Definition 2.1.15}\} \\ & (\forall T : T \in \mathbf{p}S : (ua\lceil \mathbf{a}T, T) \in \pi) \\ \Rightarrow & \{\text{Property A.1.1, definition of } \lceil\} \\ & (\forall T : T \in \mathbf{p}S : (u\lceil \mathbf{a}T, T) \in \pi) \\ \Rightarrow & \{\text{Definition 2.1.15}\} \\ & u \in \mathbf{tr}.\pi. \end{array}$$

Hence, $\mathbf{tr}.\pi$ is prefix-closed.

□

The configurations of a computation are precisely those configurations that contribute to the traces associated with it (this results from the second conjunct in Definition 2.1.11). This yields the following property.

Property A.1.5

For computation π of system S

$$Conf.\pi = \{(u\lceil \mathbf{a}T, T) \mid u \in \mathbf{tr}.\pi \wedge T \in \mathbf{p}S\}.$$

Proof

For $T \in \mathbf{p}S$ we prove that $(t,T) \in Conf.\pi \Leftrightarrow (\exists u : u \in \mathbf{tr}.\pi : u\lceil \mathbf{a}T = t)$.

Case $\ell.t = 0$

Then $t = \varepsilon$. On account of Definition 2.1.15 and the fact that $\mathbf{tr}.\pi$ is prefix-closed we conclude

$$(\varepsilon, T) \in \mathit{Conf}.\pi \;\Leftrightarrow\; (\exists u : u \in \mathbf{tr}.\pi : u \restriction \mathbf{a}T = \varepsilon).$$

Case $\ell.t \neq 0$

Then $t \neq \varepsilon$. Let $t = va$. We derive

$$\begin{array}{ll}
& (va, T) \in \mathit{Conf}.\pi \\
\Rightarrow & \{\, \pi \text{ is admissible (Definition 2.1.11)} \,\} \\
& (\exists C, u' : (a, C) \in \mathit{Act}.\pi \wedge (v, T) \in C \wedge u'a \in \mathbf{t}S \\
& \qquad : C \subseteq \{(u' \restriction \mathbf{a}T', T') \mid T' \in \mathbf{p}S\} \subseteq \pi) \\
\Rightarrow & \{\, \text{predicate and set calculus, definition of } \mathit{C.S.a} \,\} \\
& (\exists u' : u'a \in \mathbf{t}S : \{(u' \restriction \mathbf{a}T', T') \mid T' \in \mathbf{p}S\} \subseteq \pi \wedge u' \restriction \mathbf{a}T = v \wedge a \in \mathbf{a}T) \\
\Rightarrow & \{\, \pi \text{ is maximal, Definition 2.1.13, } s = s' \;\Rightarrow\; sb = s'b \,\} \\
& (\exists u' : u'a \in \mathbf{t}S : \{(u' \restriction \mathbf{a}T', T') \mid T' \in \mathbf{p}S\} \subseteq \pi \wedge (u' \restriction \mathbf{a}T)a = va \wedge a \in \mathbf{a}T \\
& \qquad\qquad \wedge \{((u' \restriction \mathbf{a}T')a, T') \mid T' \in \mathbf{p}S \wedge a \in \mathbf{a}T'\} \subseteq \pi) \\
\Rightarrow & \{\, \text{predicate and set calculus, definition of } \restriction \,\} \\
& (\exists u' : u'a \in \mathbf{t}S : \{(u'a \restriction \mathbf{a}T', T') \mid T' \in \mathbf{p}S\} \subseteq \pi \wedge u'a \restriction \mathbf{a}T = va) \\
\Leftrightarrow & \{\, \text{predicate calculus, Definition 2.1.15} \,\} \\
& (\exists u : u \in \mathbf{tr}.\pi : u \restriction \mathbf{a}T = va).
\end{array}$$

For the implication the other way around we derive

$$\begin{array}{ll}
& (\exists u : u \in \mathbf{tr}.\pi : u \restriction \mathbf{a}T = va) \\
\Rightarrow & \{\, \mathbf{tr}.\pi \text{ is prefix-closed — Property A.1.4, definition of } \restriction \,\} \\
& (\exists u' : u'a \in \mathbf{tr}.\pi : u'a \restriction \mathbf{a}T = va) \\
\Leftrightarrow & \{\, \text{Definition 2.1.15} \,\} \\
& (\exists u' : u'a \in \mathbf{t}S : \{(u'a \restriction \mathbf{a}T', T') \mid T' \in \mathbf{p}S\} \subseteq \pi \wedge u'a \restriction \mathbf{a}T = va) \\
\Rightarrow & \{\, \text{predicate and set calculus, } T \in \mathbf{p}S \,\} \\
& (va, T) \in \pi \\
\Leftrightarrow & \{\, \text{for } T \in \mathbf{p}S,\ (s, T) \in \pi \;\Leftrightarrow\; (s, T) \in \mathit{Conf}.\pi \,\} \\
& (va, T) \in \mathit{Conf}.\pi.
\end{array}$$

□

Lemma A.1.2 and Property A.1.5 yield the following corollary, showing that projections of traces associated with a computation on the alphabet of one process are totally ordered by the prefix relation.

Corollary A.1.6

For computation π of system S, traces t, u of $\mathbf{tr}.\pi$, and process T of S

$$t\restriction \mathbf{a}T \leq u\restriction \mathbf{a}T \vee u\restriction \mathbf{a}T \leq t\restriction \mathbf{a}T.$$

□

Lemma A.1.7 facilitates the proof of Theorem A.1.8.

Lemma A.1.7

For computation π of system S, process U of S, $u \in \mathbf{tr}.\pi$, configuration (t, T), and action occurrence (a, C)

$$(t,T) \rightarrow_\pi (a,C) \rightarrow_\pi (u\restriction \mathbf{a}U, U) \Rightarrow t < u\restriction \mathbf{a}T.$$

Proof

We derive

$(t,T) \rightarrow_\pi (a,C) \rightarrow_\pi (u\restriction \mathbf{a}U, U)$

$\Rightarrow$ {Property 2.1.9, Corollary A.1.6, and Definition 2.1.15}

$(t\restriction \mathbf{a}U)a = u\restriction(\mathbf{a}U \cap \mathbf{a}T) \wedge (t \leq u\restriction \mathbf{a}T \vee t \geq u\restriction \mathbf{a}T)$

$\Rightarrow$ {Lemma 1.1.15, from the first conjunct follows that $a \in \mathbf{a}U \cap \mathbf{a}T$}

$(ta)\restriction a = u\restriction a \wedge (t \leq u\restriction \mathbf{a}T \vee t \geq u\restriction \mathbf{a}T)$

$\Rightarrow$ {calculus}

$t\restriction a < u\restriction a \wedge (t \leq u\restriction \mathbf{a}T \vee t \geq u\restriction \mathbf{a}T)$

$\Rightarrow$ {$t \leq u\restriction \mathbf{a}T \Rightarrow t\restriction a \leq u\restriction a$ and $t \geq u\restriction \mathbf{a}T \Rightarrow t\restriction a \geq u\restriction a$, predicate calculus}

$t < u\restriction \mathbf{a}T.$

□

Theorem A.1.8 shows that in contrast to $\rightarrow^*$, relation $\rightarrow_\pi^*$ is a partial order.

Theorem A.1.8

If π is a computation of system S, relation $\rightarrow_\pi^*$ is a partial order.

Proof

From the definition (of $*$) we have that $\rightarrow_\pi^*$ is reflexive and transitive. We prove that $\rightarrow_\pi^*$ is antisymmetric, i.e., that there are no cycles in π. It is sufficient to show that no configuration of π occurs in a cycle in π. That is, on account of Property A.1.5,

$$(\forall u, t, T : u \in \mathbf{tr}.\pi \wedge T \in \mathbf{p}S \wedge t \leq u\lceil \mathbf{a}T : (t, T) \text{ not in a cycle in } \pi).$$

This is proved by induction on $\ell.u$, for $u \in \mathbf{tr}.\pi$. Let $T \in \mathbf{p}S$.

Base $\ell.u = 0$

Then $u = \varepsilon$. It is obvious from Definition 2.1.7 that

$$(\forall \alpha : \alpha \in Act.S : \neg(\alpha \rightarrow (\varepsilon, T))).$$

Hence, (ε, T) not in a cycle in π.

Step $\ell.u \neq 0$

Then $u \neq \varepsilon$. Let $u = va$. We derive

$$\begin{array}{ll} & (va\lceil \mathbf{a}T, T) \in Conf.\pi \\ \Rightarrow & \{\text{Lemma A.1.7}\} \\ & (\forall t, C, U : (t, U) \rightarrow_\pi (a, C) \rightarrow_\pi (va\lceil \mathbf{a}T, T) : t < va\lceil \mathbf{a}U) \\ \Rightarrow & \{\text{property of } \lceil\} \\ & (\forall t, C, U : (t, U) \rightarrow_\pi (a, C) \rightarrow_\pi (va\lceil \mathbf{a}T, T) : t \leq v\lceil \mathbf{a}U) \\ \Rightarrow & \{\text{induction hypothesis, calculus}\} \\ & (va\lceil \mathbf{a}T, T) \text{ not in a cycle in } \pi. \end{array}$$

□

The distributed states contained in computation π are the maximal antichains of $(Conf.\pi, \rightarrow_\pi^*)$, which is proved by Theorems A.1.9 and A.1.10.

The fact that subset X of $Conf.\pi$ is a maximal antichain of $(Conf.\pi, \rightarrow_\pi^*)$ is formally expressed by

$$\begin{array}{l} (\forall \varphi' : \varphi' \in Conf.\pi \setminus X : (\exists \varphi : \varphi \in X : \varphi \rightarrow_\pi^* \varphi' \vee \varphi' \rightarrow_\pi^* \varphi)) \\ \wedge\, (\forall \varphi, \varphi' : \{\varphi, \varphi'\} \subseteq X \wedge \varphi \neq \varphi' : \neg(\varphi \rightarrow_\pi^* \varphi')). \end{array}$$

In Theorem A.1.9 we prove that every distributed state of computation π is a maximal antichain of $(Conf.\pi, \rightarrow_\pi^*)$.

Theorem A.1.9

For computation π of system S and trace $t \in \mathbf{tr}.\pi$

$\{(t\restriction \mathbf{a}T, T) \mid T \in \mathbf{p}S\}$ is a maximal antichain $(Conf.\pi, \rightarrow_\pi^*)$.

Proof

Set $\{(t\restriction \mathbf{a}T, T) \mid T \in \mathbf{p}S\}$ cannot be extended with other configurations because on account of Corollary A.1.3 we have

$$(\forall u, T : (u, T) \in Conf.\pi \wedge u \neq t\restriction \mathbf{a}T \\ : (u, T) \rightarrow_\pi^* (t\restriction \mathbf{a}T, T) \vee (t\restriction \mathbf{a}T, T) \rightarrow_\pi^* (u, T)).$$

We show that $\{(t\restriction \mathbf{a}T, T) \mid T \in \mathbf{p}S\}$ is unordered by proving that

$$(\forall v, U, T : (v, U) \in Conf.\pi \wedge T \in \mathbf{p}S \wedge (v, U) \rightarrow_\pi^+ (t\restriction \mathbf{a}T, T) : v < t\restriction \mathbf{a}U).$$

This is proved by induction on $\ell.t$.

Let (v, U) be a configuration of π and let T be a process of S.

Base $\ell.t = 0$

Then $t = \varepsilon$. From Definition 2.1.7 we have then

$$(\forall \varphi : \varphi \in Conf.\pi : \neg(\varphi \rightarrow_\pi^+ (\varepsilon, T))).$$

Hence, the domain of the universal quantification to prove is empty, which means that the quantification itself holds.

Step $\ell.t \neq 0$

Let $t = ua$. We derive

$$\begin{array}{ll} & (v, U) \rightarrow_\pi^+ (ua\restriction \mathbf{a}T, T) \\ \Leftrightarrow & \{\text{definition of } \rightarrow \text{ and } \rightarrow_\pi^+\} \\ & (\exists C : (a, C) \in Act.\pi : (v, U) \rightarrow_\pi^+ (a, C) \rightarrow_\pi (ua\restriction \mathbf{a}T, T)) \\ \Rightarrow & \{\text{definition of } \rightarrow\} \\ & (\exists C : (a, C) \in Act.\pi : (v, U) \rightarrow_\pi^* (u\restriction \mathbf{a}U, U) \rightarrow_\pi (a, C) \rightarrow_\pi (ua\restriction \mathbf{a}T, T)) \end{array}$$

$\Rightarrow$ { case analysis: $v = u\lceil \mathbf{a}U$ — Lemma A.1.7, $v \neq u\lceil \mathbf{a}U$ — induction hypothesis; calculus }

$v < ua\lceil \mathbf{a}U$.

□

In the following theorem we prove for computation π that every maximal antichain of $(Conf.\pi, \rightarrow_\pi^*)$ forms a distributed state of π.

Theorem A.1.10

Let π be a computation of system S, and let D be a maximal antichain of $(Conf.\pi, \rightarrow_\pi^*)$. Then

$$(\exists t : t \in \mathbf{tr}.\pi : D = \{(t\lceil \mathbf{a}T, T) \mid T \in \mathbf{p}S\}).$$

Proof

Let $V_0 = \{\{(\varepsilon, T) \mid T \in \mathbf{p}S\}\}$ and, for $i \geq 0$,

$$V_{i+1} = \{(D \setminus C) \cup \{(va, T) \mid (v, T) \in C\} \mid (a, C) \in Act.\pi \wedge D \in V_i \wedge C \subseteq D\}.$$

Using Definitions 2.1.7 and 2.1.13 an inductive proof can be given for the fact that the set of maximal antichains of $(Conf.\pi, \rightarrow_\pi^*)$ equals $(\cup i : i \geq 0 : V_i)$. This proof is not included here. It consists of two parts. Firstly, for $D \in V_i$ and $i \geq 0$ one has to prove (by induction on i) that D is a maximal antichain of $(Conf.\pi, \rightarrow_\pi^*)$. Secondly, one has to prove that there are no other maximal antichains of $(Conf.\pi, \rightarrow_\pi^*)$, i.e., for every maximal antichain X of $(Conf.\pi, \rightarrow_\pi^*)$ there exists an $i \geq 0$ such that $X \in V_i$ (this part also goes by induction on i).

We prove by induction on i that $D \in V_i$, for $i \geq 0$, implies

$$(\exists t : t \in \mathbf{tr}.\pi : D = \{(t\lceil \mathbf{a}T, T) \mid T \in \mathbf{p}S\}).$$

Base $i = 0$

The only element of V_0 is $\{(\varepsilon, T) \mid T \in \mathbf{p}S\}$. Moreover, from Definition 2.1.15, we have that $\varepsilon \in \mathbf{tr}.\pi$. Hence, the above existential quantification holds for every D of V_0.

Step $i \geq 0$

Let $(a, C) \in Act.\pi$ and $D \in V_i$, such that $C \subseteq D$. Let, furthermore, t be a trace of $\mathbf{tr}.\pi$ satisfying $D = \{(t\lceil \mathbf{a}T, T) \mid T \in \mathbf{p}S\}$ (such t exists on account of induction hypothesis). We derive

$$
\begin{array}{ll}
& (D \setminus C) \cup \{(va, T) \mid (v, T) \in C\} \\
= & \quad \{\text{Definition 2.1.5, } C \subseteq D, \text{ for } T \in \mathbf{p}S \text{ we have } v = t\lceil \mathbf{a}T\} \\
& \{(t\lceil \mathbf{a}T, T) \mid T \in \mathbf{p}S \wedge a \notin \mathbf{a}T\} \cup \{((t\lceil \mathbf{a}T)a, T) \mid T \in \mathbf{p}S \wedge a \in \mathbf{a}T\} \\
= & \quad \{\text{definition of } \lceil, \text{ set calculus}\} \\
& \{(ta\lceil \mathbf{a}T, T) \mid T \in \mathbf{p}S\} \\
\subseteq & \quad \{\text{Definition 2.1.13}\} \\
& \pi.
\end{array}
$$

Moreover, from Definition 2.1.15, $ta \in \mathbf{tr}.\pi$. Hence, the above existential quantification holds for every D of V_{i+1}.

□

In a computation of S no forward branched configurations occur, according to Definition 2.1.13. A consequence of this, thanks to the structure of configurations of S, is that no configuration of a computation of S is backward branched, which is proved in the next lemma.

Lemma A.1.11

No configuration of a computation of a system is backward branched.

Proof

Let π be a computation of system S. Note that (ϵ, T), for any $T \in \mathbf{p}S$, is not backward branched in π. Let $\{(a, C), (a', C')\} \subseteq Act.\pi$ and $(ta, T) \in Conf.\pi$. We derive

$$
\begin{array}{ll}
& (a, C) \rightarrow (ta, T) \wedge (a', C') \rightarrow (ta, T) \\
\Leftrightarrow & \quad \{\text{Definition 2.1.7}\} \\
& (\exists u : ta = ua : (u, T) \in C) \wedge (\exists u : ta = u'a : (u, T) \in C') \\
\Leftrightarrow & \quad \{\text{calculus}\} \\
& (t, T) \in C \wedge a = a' \wedge (t, T) \in C' \\
\Rightarrow & \quad \{\text{set and predicate calculus}\} \\
& C \cap C' \neq \emptyset \\
\Rightarrow & \quad \{\text{Property 2.1.12}\} \\
& (a, C) = (a', C').
\end{array}
$$

Hence, according to Definition 2.1.10, (ta, T) is not backward branched in π.

□

A.2 Relationship between the operational model and nets

The elements underlying our operational model show a great similarity to nets. In this section we expose this relationship. For this purpose the basic notions of net theory are recalled after [Gol]. First the general notion of a net is defined.

Definition A.2.1

Let $\mathcal{S}$ and $\mathcal{T}$ be disjoint sets and let $\mathcal{F}$ be a relation, $\mathcal{F} \subseteq (\mathcal{S} \times \mathcal{T}) \cup (\mathcal{T} \times \mathcal{S})$. The triple $(\mathcal{S}, \mathcal{T}, \mathcal{F})$ is called a net if

$$(\forall t : t \in \mathcal{T} : (\exists s : s \in \mathcal{S} : (t,s) \in \mathcal{F} \vee (s,t) \in \mathcal{F})).$$

□

The following notation is used often in the context of nets:

$$\bullet x = \{y \mid y \in X \wedge (y,x) \in \mathcal{F}\},$$
$$x\bullet = \{y \mid y \in X \wedge (x,y) \in \mathcal{F}\},$$

for $x \in \mathcal{T}$ and $X = \mathcal{S}$, or $x \in \mathcal{S}$ and $X = \mathcal{T}$.

There are many different kinds of nets. They may have additional structure compared to Definition A.2.1 (for instance *marked nets*), or they may be special cases (for instance *occurrence nets*). An overview of different kinds of nets together with related notions can be found in [Gol]. We restrict ourselves to a special kind of net, viz., occurrence nets, because of their similarity to partial-order computations of systems.

Definition A.2.2

Net $(\mathcal{S}, \mathcal{T}, \mathcal{F})$ is called an occurrence net if

$\mathcal{F}^*$ is antisymmetric

and

$$(\forall s,t,t' : s \in \mathcal{S} \wedge ((t,s) \in \mathcal{F} \wedge (t',s) \in \mathcal{F}) \vee ((s,t) \in \mathcal{F} \wedge (s,t') \in \mathcal{F}) : t = t').$$

□

Let us now consider $\mathcal{O}[\![S]\!] = (Conf.S, Act.S, \rightarrow)$ being the operational model of system S. Then it is easy to check that $(Conf.S, Act.S, \rightarrow)$ is a net. First of all, $Conf.S$ and $Act.S$ are disjoint sets. Furthermore, $\rightarrow \subseteq (Conf.S \times Act.S) \cup (Act.S \times Conf.S)$ and

$$(\forall \alpha : \alpha \in Act.S : (\exists \varphi : \varphi \in Conf.S : \alpha \rightarrow \varphi \vee \varphi \rightarrow \alpha)).$$

All conditions of $(Conf.S, Act.S, \rightarrow)$ being a net are thus satisfied.

Partial-order computations are related to occurrence nets. Let π be a partial-order computation of system S. Then $(Conf.\pi, Act.\pi, \rightarrow_\pi)$ is an occurrence net. Firstly, it is a net, which is easy to verify. Secondly, relation $\rightarrow_\pi^*$ is antisymmetric (it is a partial order, Theorem A.1.8). Furthermore, from Definition 2.1.13 it follows that no element of $Conf.\pi$ is forward branched, and from Lemma A.1.11 we know that no element of $Conf.\pi$ is backward branched.

Nets corresponding to $\mathcal{O}[\![S]\!]$ and to computations of S have a few specific properties. The following definitions come from [Roz].

Definition A.2.3

Net $(\mathcal{S}, \mathcal{T}, \mathcal{F})$ is *finite* if $\mathcal{S} \cup \mathcal{T}$ is finite.

□

Definition A.2.4

Net $(\mathcal{S}, \mathcal{T}, \mathcal{F})$ is *pure* if

$$(\forall x : x \in \mathcal{S} \cup \mathcal{T} : {\bullet x} \cap x{\bullet} = \emptyset).$$

□

Definition A.2.5

Net $(\mathcal{S}, \mathcal{T}, \mathcal{F})$ is *simple* if

$$(\forall x, y : \{x, y\} \subseteq \mathcal{S} \cup \mathcal{T} : {\bullet x} = {\bullet y} \wedge x{\bullet} = y{\bullet} \Rightarrow x = y).$$

□

Nets corresponding to $\mathcal{O}[\![S]\!]$ and to computations of S are not necessarily finite or simple. All of them are pure.

A.3 Comparison of the operational model with an interleaving model

The usual way of defining the operational semantics of mechanisms is by labelled transition systems ([Hen], and others). They are often used as an operational model that underlies many existing models of concurrency as well. Labelled transition systems are

based on the notions of global state and indivisible actions that cause state transitions. Basically, this results in an interleaving semantics and therefore usually leads to an inappropriate treatment of divergence problems.

Definition A.3.1

A labelled transition system ([De0]) is a quadruple $(\mathcal{S}, \mathcal{A}, \rightarrow, s_0)$, where

- $\mathcal{S}$ is a set of (global) states,
- $\mathcal{A}$ is a set of actions, $\mathcal{S} \cap \mathcal{A} = \emptyset$,
- $\rightarrow$ is a transition relation, $\rightarrow \subseteq \mathcal{S} \times \mathcal{A} \times \mathcal{S}$,
- $s_0 \in \mathcal{S}$ is the initial state.

□

For states s and s' of $\mathcal{S}$, and for action a of $\mathcal{A}$, we usually write $s \xrightarrow{a} s'$ instead of $(s, a, s') \in \rightarrow$. Furthermore, relations $\xrightarrow{a}$, for $a \in \mathcal{A}$, are extended to relations $\xrightarrow{t}$, for any $t \in \mathcal{A}^*$, as follows

- $s \xrightarrow{\epsilon} s'$ if $s = s'$,
- $s \xrightarrow{at} s'$ if $(\exists s'' : s'' \in \mathcal{S} : s \xrightarrow{a} s'' \wedge s'' \xrightarrow{t} s')$.

We provide an interleaving operational model of the same systems as in the previous chapter. The interleaving model of system S is defined to be the labelled transition system

$$(\mathbf{s}S, \mathbf{a}S, \rightarrow, s_0),$$

where

- $\mathbf{s}S$ is the set of states, which are mappings s from $\mathbf{p}S$ into $\mathbf{a}S^*$, such that ([Ver]) $(\forall T : T \in \mathbf{p}S : s.T \in \mathbf{t}T) \wedge (\exists t : t \in \mathbf{t}S : (\forall T : T \in \mathbf{p}S : t \restriction \mathbf{a}T = s.T))$,
- s_0 is the initial state, such that $(\forall T : T \in \mathbf{p}S : s_0.T = \epsilon)$,
- $\rightarrow$ is the transition relation, such that for two states s and s' and $a \in \mathbf{a}S$ $s \xrightarrow{a} s'$ if $(\forall T : T \in \mathbf{p}S : (a \in \mathbf{a}T \Rightarrow (s.T)a = s'.T) \wedge (a \notin \mathbf{a}T \Rightarrow s.T = s'.T))$.

Note that, for state s, trace $t \in \mathbf{t}S$ such that $(\forall T : T \in \mathbf{p}S : t\lceil \mathbf{a}T = s.T)$ is not necessarily unique.

States of S can be represented by tuples of traces, if we fix the order of processes.

The behaviour of a system in the interleaving model is characterized by *sequential (interleaving) computations.*

Definition A.3.2

A sequential (interleaving) computation of system S is defined to be a subset p of $\mathbf{s}S$ that satisfies the following conditions

- $s_0 \in p$,
- p is admissible, that is, $(\forall s' : s' \in p \wedge s' \neq s_0 : (\exists s, a : s \in p \wedge a \in \mathbf{a}S : s \xrightarrow{a} s'))$,
- every state of p has at most one successor in p, i.e.,
  $(\forall s, a, s', a', s'' : \{s, s', s''\} \subseteq p \wedge s \xrightarrow{a} s' \wedge s \xrightarrow{a'} s'' : s' = s'' \wedge a = a')$,
- p is non-extendable, that is,
  $(\forall s, s', a : s' \in \mathbf{s}S \setminus p \wedge s \in p \wedge s \xrightarrow{a} s' : (\exists s'', b : s'' \in p \wedge s \xrightarrow{b} s'' : a \neq b))$.

□

With every state s in a given interleaving computation the unique (global) trace $tr.s$ is associated (as a result of the requirement that all states in a computation have at most one successor). It is defined inductively by

$$\begin{aligned} tr.s_0 &= \varepsilon \\ tr.s_{i+1} &= (tr.s_i)a, \qquad \text{for } i \geq 0, a \in \mathbf{a}S, \text{ and } s_i \xrightarrow{a} s_{i+1}. \end{aligned}$$

Trace $tr.s$ can be calculated in a finite number of steps because traces are finite by definition.

Sequential computations are in fact the complete traces (infinite traces or traces with empty successor sets), and we have already observed that they do not characterize systems properly.

In order to compare the interleaving operational model just introduced with the operational model of Section 2.1, the notion of acceptances is defined. In this case acceptances do not have the ability to reflect parallelism.

Definition A.3.3

Let $t \in \mathbf{t}(\mathbf{pr}S)$ and $L \subseteq \mathbf{e}S^*$. Pair (t, L) is an acceptance of system S (in the interleaving model) if

$$(\forall p, s : p \in \mathit{ilc}.S \wedge s \in p \wedge t = \mathit{tr}.s\lceil \mathbf{e}S \\ : (\exists s', v : s' \in p \wedge v \in L : tv = \mathit{tr}.s'\lceil \mathbf{e}S)),$$

where $\mathit{ilc}.S$ denotes the set of sequential computations of S.

□

According to the interleaving model of system S of Example 2.1.17, $(\epsilon, \{a\})$ is not an acceptance of S, which is because $\{(b^i, \epsilon) \mid i \geq 0\}$ (the first trace in every pair belongs to T and the second one to U) is a sequential computation. Essentially this means, from an interpretational point of view, that a will not necessarily be executed even if the environment is willing to participate in that action. This result is due to global states and interleaving. To avoid this problem one usually restricts the computations to fair sequential computations (e.g., [Mal]). On account of Definition 2.2.4, $(\epsilon, \{a\})$ does belong to the acceptances of S, which means that a is guaranteed to occur.

The set of acceptances of the interleaving model of system S equals $\mathbf{ac}\langle \mathbf{e}S, \{\mathbf{W}.\mathbf{p}S\}\rangle$. This statement is implied by

$$\{\{\mathit{tr}.s \mid s \in p\} \mid p \in \mathit{ilc}.S\} = \{p' \mid p' \in \mathit{pot}.\langle \mathbf{e}S, \{\mathbf{W}.\mathbf{p}S\}\rangle\}.$$

The remainder of this section consists of the proof of the above equality. Note that $\langle \mathbf{e}S, \{\mathbf{W}.\mathbf{p}S\}\rangle$ has only one process.

Let $p \in \mathit{ilc}.S$. We prove that $\{\mathit{tr}.s \mid s \in p\}$ belongs to $\mathit{pot}.\langle \mathbf{e}S, \{\mathbf{W}.\mathbf{p}S\}\rangle$ (see Definition 2.3.2).

- $\{\mathit{tr}.s \mid s \in p\} \neq \emptyset$

Proof

We derive

$p \in \mathit{ilc}.S$
$\Rightarrow$ { Definition A.3.2 }
$s_0 \in p$
$\Rightarrow$ { definition of tr }
$\epsilon \in \{\mathit{tr}.s \mid s \in p\}$
$\Rightarrow$ { set calculus }
$\{\mathit{tr}.s \mid s \in p\} \neq \emptyset$.

- $\{\mathit{tr}.s \mid s \in p\}$ is prefix-closed

Proof

We derive

$$
\begin{array}{ll}
& ta \in \{tr.s \mid s \in p\} \\
\Leftrightarrow & \{\text{set calculus, } tr.s_0 = \varepsilon\} \\
& (\exists s' : s' \in p \wedge s' \neq s_0 : tr.s' = ta) \\
\Rightarrow & \{\text{Definition A.3.2 — } p \text{ is admissible, predicate calculus}\} \\
& (\exists s', s, a' : s \in p \wedge a' \in \mathbf{a}S : s \xrightarrow{a'} s' \wedge tr.s' = ta) \\
\Rightarrow & \{\text{definition of } tr, \text{ predicate calculus}\} \\
& (\exists s, a' : s \in p \wedge a' \in \mathbf{a}S : (tr.s)a' = ta) \\
\Rightarrow & \{\text{predicate calculus}\} \\
& (\exists s : s \in p : tr.s = t) \\
\Leftrightarrow & \{\text{set calculus}\} \\
& t \in \{tr.s \mid s \in p\}.
\end{array}
$$

- $(\forall t, a, b : \{a, b\} \subseteq \mathbf{a}S \wedge \{ta, tb\} \subseteq \{tr.s \mid s \in p\} : a = b)$

Proof

Let $\{a, b\} \subseteq \mathbf{a}S$. We derive

$$
\begin{array}{ll}
& \{ta, tb\} \subseteq \{tr.s \mid s \in p\} \\
\Leftrightarrow & \{\text{set calculus}\} \\
& (\exists s', s'' : \{s', s''\} \subseteq p : tr.s' = ta \wedge tr.s'' = tb) \\
\Rightarrow & \{\text{definition of } tr\} \\
& (\exists s, s', s'' : \{s, s', s''\} \subseteq p : s \xrightarrow{a} s' \wedge s \xrightarrow{b} s'') \\
\Rightarrow & \{\text{Definition A.3.2 — every state of } p \text{ has at most one successor in } p\} \\
& a = b.
\end{array}
$$

- $(\forall t, u : t \in \{tr.s \mid s \in p\} \wedge u \in \mathbf{t}S \wedge t = u : u \in p)$ is trivially true.

- $\{tr.s \mid s \in p\}$ is a maximal subset of $\mathbf{t}S$ that satisfies the above four conditions.

Proof

On account of Property 2.3.3 it is sufficient to prove that

$$
\begin{array}{l}
(\forall t, a : ta \in \mathbf{t}S \setminus \{tr.s \mid s \in p\} \wedge t \in \{tr.s \mid s \in p\} \\
\qquad : (\exists b : tb \in \{tr.s \mid s \in p\} : a \neq b)).
\end{array}
$$

Let $ta \in \mathbf{t}S$ and $t \in \{tr.s'' \mid s'' \in p\}$. We define s to be an element of p such that $tr.s = t$, and s' to be an element of $\mathbf{s}S$ such that $tr.s' = ta$. We derive

$ta \notin \{tr.s'' \mid s'' \in p\}$
$\Rightarrow$ {definitions of tr, s', and $\rightarrow$, $tr.s = t$}
$s' \notin p \wedge s \xrightarrow{a} s'$
$\Rightarrow$ {$s \in p$, Definition A.3.2 — p is non-extendable}
$(\exists s'', b : s'' \in p \wedge s \xrightarrow{b} s'' : a \neq b)$
$\Rightarrow$ {definition of tr, $tr.s = t$, set and predicate calculus}
$(\exists b : tb \in \{tr.s'' \mid s'' \in p\} : a \neq b)$.

Now let $p' \in pot.\langle \mathbf{e}S, \{\mathbf{W}.\mathbf{p}S\}\rangle$. We prove that $p = \{s \mid s \in \mathbf{s}S \wedge (\exists t : t \in p' : tr.s = t)\}$ belongs to $ilc.S$ (see Definition A.3.2).

- $s_0 \in p$

Proof

We derive

$p' \in pot.\langle \mathbf{e}S, \{\mathbf{W}.\mathbf{p}S\}\rangle$
$\Rightarrow$ {Definition 2.3.2 — p' is non-empty and prefix-closed}
$\varepsilon \in p'$
$\Rightarrow$ {definitions of tr and of p}
$s_0 \in p$.

- $(\forall s' : s' \in p \wedge s' \neq s_0 : (\exists s, a : s \in p \wedge a \in \mathbf{a}S : s \xrightarrow{a} s'))$

Proof

Let $s' \in p$. We derive

$s' \neq s_0$
$\Rightarrow$ {$s' \in p$, definitions of tr and of p}
$(\exists t, a : ta \in p' : tr.s' = ta)$
$\Rightarrow$ {p' is prefix-closed, definitions of tr and p}
$(\exists t, a, s : ta \in p' \wedge t \in p' \wedge s \in p : tr.s' = ta \wedge tr.s = t)$
$\Rightarrow$ {definition of $\rightarrow$, $ta \in p' \Rightarrow a\mathbf{a}S$, predicate calculus}
$(\exists s, a : s \in p \wedge a \in \mathbf{a}S : s \xrightarrow{a} s')$.

- $(\forall s, a, s', a', s'' : \{s, s', s''\} \subseteq p \wedge s \xrightarrow{a} s' \wedge s \xrightarrow{a'} s'' : s' = s'' \wedge a = a')$

Proof

Let $\{s, s', s''\} \subseteq p$. We derive

$$
\begin{array}{ll}
& s \xrightarrow{a} s' \wedge s \xrightarrow{a'} s'' \\
\Leftrightarrow & \{\{s', s''\} \subseteq p, \text{ definitions of } p \text{ and } tr, \text{ predicate calculus}\} \\
& s \xrightarrow{a} s' \wedge s \xrightarrow{a'} s'' \wedge (\exists t : \{ta, ta'\} \subseteq p' : tr.s' = ta \wedge tr.s'' = ta') \\
\Rightarrow & \{\text{on account of Definition 2.3.2 and the fact that } \langle \mathbf{e}S, \{\mathbf{W}.\mathbf{p}S\}\rangle \text{ has only} \\
& \quad \text{one process, we have } (\forall u, b, b' : \{ub, ub'\} \subseteq p' : b = b'), \text{ predicate calculus}\} \\
& s \xrightarrow{a} s' \wedge s \xrightarrow{a'} s'' \wedge (\exists t : ta \in p' : tr.s' = ta \wedge tr.s'' = ta) \wedge a = a' \\
\Rightarrow & \{\{s', s''\} \subseteq p, \text{ definitions of } p \text{ and } \rightarrow\} \\
& s' = s'' \wedge a = a'.
\end{array}
$$

- $(\forall s, s', a : s' \in \mathbf{s}S \setminus p \wedge s \in p \wedge s \xrightarrow{a} s' : (\exists s'', b : s'' \in p \wedge s \xrightarrow{b} s'' : a \neq b))$

Proof

Let $s' \in \mathbf{s}S$ and $s \in p$. We derive

$$
\begin{array}{ll}
& s' \notin p \wedge s \xrightarrow{a} s' \\
\Rightarrow & \{s \in p, \text{ definitions of } p \text{ and } tr, \text{ set and predicate calculus}\} \\
& tr.s \in p' \wedge (tr.s)a \notin p' \\
\Rightarrow & \{\text{Property 2.3.3, } \langle \mathbf{e}S, \{\mathbf{W}.\mathbf{p}S\}\rangle \text{ has only one process}\} \\
& (\exists b : (tr.s)b \in p' : a \neq b) \\
\Rightarrow & \{\text{definitions of } p, tr, \text{ and } \rightarrow\} \\
& (\exists s'', b : s'' \in p \wedge s \xrightarrow{b} s'' : a \neq b).
\end{array}
$$

The conclusion is then that

$$\{\{tr.s \mid s \in p\} \mid p \in ilc.S\} = \{p' \mid p' \in pot.\langle \mathbf{e}S, \{\mathbf{W}.\mathbf{p}S\}\rangle\},$$

which completes the proof.

A.4 An example: three buffers

The example is presented along with a program notation ([Kal]) that serves the purpose of denoting systems ([Zwa]). For the discussion of the structure of the universe of

symbols, Ω, see Section 4.1. In programs simple symbols, like a and b, as well as compound symbols, like $p{\cdot}a$ and $p{\cdot}b$, are used.

Every program — also called a component — defines a system. We distinguish three classes of components: simple components, non-recursive components, and recursive components.

A simple component describes a sequential mechanism without internal actions. It is of the form

$$\textbf{com } c(A) : \mathcal{C} \textbf{ moc}.$$

The name of the component is c, A is its external alphabet and $\mathcal{C}$ is its command. Such a program has to satisfy the following restrictions

- A is finite and consists of simple symbols only,
- $A = \mathbf{a}(\mathbf{pr}\mathcal{C})$.

The system corresponding to component c is denoted by $sys.c$ and is defined by

$$sys.c = \langle A, \{\mathbf{pr}\mathcal{C}\}\rangle.$$

A process can also be associated with a component, as a means to describe the sequentialized behaviour of the component. The process associated with component c is denoted by $\mathbf{pr}c$ and it is defined by $\mathbf{pr}c = \mathbf{pr}(sys.c)$. For simple components c defined above we have $\mathbf{pr}c = \mathbf{pr}\mathcal{C}$.

Example A.4.1

Component buf_1 is defined by

$$\textbf{com } buf_1(\{a,b\}) : (a\,;\,b)^* \textbf{ moc}.$$

The system associated with this component is

$$sys.buf_1 = \langle \{a,b\}, \{\mathbf{buf}_1.a.b\}\rangle.$$

The process associated with buf_1 is

$$\mathbf{pr}buf_1 = \mathbf{buf}_1.a.b.$$

□

A non-recursive component describes a parallel composition of sequential mechanisms. It has the following general form

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

Here again c is the name of the component, A is its external alphabet, and C is its command. Between **sub** and **bus**, the subcomponents are declared. In this declaration, $c_0, c_1, \ldots, c_{n-1}$ are previously defined components, and $p_0, p_1, \ldots, p_{n-1}$ are the subcomponents of c of the types $c_0, c_1, \ldots, c_{n-1}$, respectively. If $p_0, p_1, \ldots, p_i$ have the same type (that is, $c_0 = c_1 = \ldots = c_i$), the abbreviation $p_0, p_1, \ldots, p_i : c_0$ is used. With subcomponent p_i, system $p_i{\cdot}sys.c_i$ is associated. In the sequel, B denotes the union of the sets of external symbols of all subcomponents, i.e.,

$$B = (\cup i : 0 \leq i < n : \mathbf{ep}_i{\cdot}sys.c_i).$$

Observe that external alphabets of systems $p_i{\cdot}sys.c_i$ are pairwise disjoint. The intersection of the external alphabet of $p_i{\cdot}sys.c_i$ and A, for $0 \leq i \leq n$, is empty as well. That is,

$$(\forall i, j : 0 \leq i < j < n : \mathbf{ep}_i{\cdot}sys.c_i \cap \mathbf{ep}_j{\cdot}sys.c_j = \emptyset)$$
$$\wedge\, (\forall i : 0 \leq i < n : \mathbf{ep}_i{\cdot}sys.c_i \cap A = \emptyset).$$

The set of internal symbols of c equals $B \setminus \{x_j \mid 0 \leq j < m\}$.
The equalities represent connections between two subcomponents and give the links between internal and external symbols. Equality $x_j = y_j$ results in renaming x_j to y_j in each $p_i{\cdot}sys.c_i$ that has x_j in its alphabet, for $0 \leq j < m$ and $0 \leq i < n$.
The requirements for this case are

- A is finite and contains simple symbols only,
- $p_0, p_1, \ldots, p_{n-1}$ are n distinct and simple symbols,
- no external symbols of the same subcomponent are connected either directly or indirectly, that is
  - $(\forall j : 0 \leq j < m : x_j \in B)$,
  - $(\forall j : 0 \leq j < m : y_j \in B \cup A)$,
  - $|\{x_j \mid 0 \leq j < m\}| = m$,
  - $\{x_j \mid 0 \leq j < m\} \cap \{y_j \mid 0 \leq j < m\} = \emptyset$,

- for all j, $0 \leq j < m$, symbols x_j and y_j belong to two different external alphabets of the subcomponents, i.e.,
  $(\forall i : 0 \leq i < n : \{x_j, y_j\} \not\subseteq ep_i{\cdot}sys.c_i)$, (note that, because of the first restriction, $\{x_j, y_j\} \not\subseteq A$),
- for all j and k, $0 \leq j < k < m$, such that $y_j = y_k$, symbols x_j and x_k belong to two different external alphabets of the subcomponents, i.e.,
  $(\forall i : 0 \leq i < n : \{x_j, x_k\} \not\subseteq ep_i{\cdot}sys.c_i)$, (note that, because of the first restriction, $\{x_j, x_k\} \not\subseteq A$),

- every external symbol of the component appears in command C or is associated with an internal symbol, that is, $A \subseteq \mathbf{a}(\mathbf{pr}C) \cup \{y_j \mid 0 \leq j < m\}$,
- the alphabet of command C consists of external symbols and internal symbols that are not in $\{x_j \mid 0 \leq j < m\}$, i.e., $\mathbf{a}(\mathbf{pr}C) \subseteq A \cup B \setminus \{x_j \mid 0 \leq j < m\}$.

The system corresponding to component c is defined by

$$sys.c = ((\| i : 0 \leq i < n : (p_i{\cdot}sys.c_i)^{x_0,x_1,\ldots,x_{m-1}}_{y_0,y_1,\ldots,y_{m-1}}) \parallel \langle \mathbf{a}(\mathbf{pr}C), \{\mathbf{pr}C\}\rangle)\restriction A,$$

where $(p_i{\cdot}sys.c_i)^{x_0,x_1,\ldots,x_{m-1}}_{y_0,y_1,\ldots,y_{m-1}}$ denotes system $p_i{\cdot}sys.c_i$ in which every occurrence of symbol x_j is replaced by y_j, for $0 \leq j < m$. Note that due to the above restrictions $sys.c$ is well-defined.

The process associated with c is ([Zwa])

$$\mathbf{pr}c = ((\mathbf{W}i : 0 \leq i < n : (p_i{\cdot}\mathbf{pr}c_i)^{x_0,x_1,\ldots,x_{m-1}}_{y_0,y_1,\ldots,y_{m-1}}) \mathbf{\ w\ } \mathbf{pr}C)\restriction A.$$

Example A.4.2

Component buf_3 is defined by

com $buf_3(\{a, b\})$:
  sub $p, q : buf_1$ **bus**
  $[p{\cdot}a = a, q{\cdot}b = b]$
  $(p{\cdot}b\,;\, q{\cdot}a)^*$
moc.

The system associated with component buf_3 is

$$sys.buf_3 = \langle \{a, b\}, \{\mathbf{buf}_1.a.(p{\cdot}b), \mathbf{buf}_1.(p{\cdot}b).(q{\cdot}a), \mathbf{buf}_1.(q{\cdot}a).b\}\rangle.$$

The process associated with buf_3 is

$\mathbf{pr}buf_3$

$=$ { definition of the process associated with a component }

$(\mathbf{buf}_1.a.(p{\cdot}b)\ \mathbf{w}\ \mathbf{buf}_1.(p{\cdot}b).(q{\cdot}a)\ \mathbf{w}\ \mathbf{buf}_1.(q{\cdot}a).b)\restriction\{a,b\}$

$=$ { Lemma 1.1.13, set calculus, Property 1.1.18 }

$((\mathbf{buf}_1.a.(p{\cdot}b)\ \mathbf{w}\ \mathbf{buf}_1.(p{\cdot}b).(q{\cdot}a))\restriction\{a,q{\cdot}a\}\ \mathbf{w}\ \mathbf{buf}_1.(q{\cdot}a).b)\restriction\{a,b\}$

$=$ { Corollary 1.1.25 with $A=\{a\}$, $B=\{p{\cdot}b\}$ and $C=\{q{\cdot}a\}$ }

$(\mathbf{buf}_2.a.(q{\cdot}a)\ \mathbf{w}\ \mathbf{buf}_1.(q{\cdot}a).b)\restriction\{a,b\}$

$=$ { Corollary 1.1.25 with $A=\{a\}$, $B=\{q{\cdot}a\}$ and $C=\{b\}$ }

$\mathbf{buf}_3.a.b.$

□

We discuss only direct recursive components. A recursive component with name c, external alphabet A and command $\mathcal{C}$ has the following form

com $c(A)$:
  sub $p:c$ **bus**
  $\mathcal{C}$
moc.

The restrictions imposed on such a program are

- A is a finite alphabet consisting of simple symbols only,
- p is a simple symbol,
- $\mathbf{a}(\mathbf{pr}\mathcal{C}) = A \cup p{\cdot}A$.

The system associated with component c is defined by

$$sys.c = \langle A, \{(p{\cdot})^i\mathbf{pr}\mathcal{C} \mid i \geq 0\}\rangle.$$

The process associated with component c is

$$\mathbf{pr}c = (\mathbf{W}i : i \geq 0 : (p{\cdot})^i\mathbf{pr}\mathcal{C})\restriction A.$$

From [Zwa] we know that $\mathbf{pr}c$ equals the least fixpoint of function $f : \mathcal{T}.A \to \mathcal{T}.A$ defined by $f.T = (p{\cdot}T\ \mathbf{w}\ \mathbf{pr}\mathcal{C})\restriction A$, for $T \in \mathcal{T}.A$. This least fixpoint is

$$(\cup i : i \geq 0 : f^i.(\mathbf{stop}.A)).$$

Example A.4.3

Component *buf* is defined by

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

The system corresponding to component *buf* is

$$sys.buf = \langle \{a,b\}, \{(p\cdot)^i \mathbf{buf}_1.\{a,p{\cdot}b\}.\{p{\cdot}a,b\} \mid i \geq 0\}\rangle.$$

The process corresponding to *buf* equals the least fixpoint of function f defined by

$$f.T = (p{\cdot}T \mathbin{\mathbf{w}} \mathbf{buf}_1.\{a,p{\cdot}b\}.\{p{\cdot}a,b\})\restriction\{a,b\},$$

for process $T \in \mathcal{T}.\{a,b\}$.
We have $f^0.(\mathbf{stop}.\{a,b\}) = \mathbf{stop}.\{a,b\}$ and

$$\begin{array}{ll} & f^1.(\mathbf{stop}.\{a,b\}) \\ = & \quad \{\text{definition of } f\} \\ & (p{\cdot}\mathbf{stop}.\{a,b\} \mathbin{\mathbf{w}} \mathbf{buf}_1.\{a,p{\cdot}b\}.\{p{\cdot}a,b\})\restriction\{a,b\} \\ = & \quad \{\text{definition of } p{\cdot}\} \\ & (\mathbf{stop}.\{p{\cdot}a,p{\cdot}b\} \mathbin{\mathbf{w}} \mathbf{buf}_1.\{a,p{\cdot}b\}.\{p{\cdot}a,b\})\restriction\{a,b\} \\ = & \quad \{\text{calculus}\} \\ & \mathbf{buf}_1.a.b. \end{array}$$

We prove by induction that $f^i.(\mathbf{stop}.\{a,b\}) = \mathbf{buf}_i.a.b$, for $i \geq 1$.

Base $i = 1$
From the above derivation we have $f^1.(\mathbf{stop}.\{a,b\}) = \mathbf{buf}_1.a.b$.

Step $i \geq 1$
We derive

$$\begin{array}{ll} & f^{i+1}.(\mathbf{stop}.\{a,b\}) \\ = & \quad \{\text{calculus}\} \\ & f.(f^i.(\mathbf{stop}.\{a,b\})) \end{array}$$

$$
\begin{array}{ll}
= & \{\text{induction hypothesis}\} \\
& f.(\mathbf{buf}_i.a.b) \\
= & \{\text{definition of } f\} \\
& (p{\cdot}\mathbf{buf}_i.a.b \ \mathbf{w}\ \mathbf{buf}_1.\{a, p{\cdot}b\}.\{p{\cdot}a, b\})\restriction\{a, b\} \\
= & \{\text{definition of } p{\cdot}\} \\
& (\mathbf{buf}_i.(p{\cdot}a).(p{\cdot}b)\ \mathbf{w}\ \mathbf{buf}_1.\{a, p{\cdot}b\}.\{p{\cdot}a, b\})\restriction\{a, b\} \\
= & \{\text{correspondence } \mathbf{buf}_i - \mathbf{sync}_{i,0}\text{, Lemma 1.1.23 with } A = \{p{\cdot}a\},\ B = \{p{\cdot}b\}, \\
& \quad C = \{a, p{\cdot}b\} \text{ and } D = \{p{\cdot}a, b\}\} \\
& \mathbf{buf}_{i+1}.a.b.
\end{array}
$$

Furthermore,

$$
\begin{array}{ll}
& (\cup i : i \geq 0 : f^i.(\mathbf{stop}.\{a, b\})) \\
= & \{\text{calculus}\} \\
& \mathbf{stop}.\{a, b\} \cup (\cup i : i \geq 1 : f^i.(\mathbf{stop}.\{a, b\})) \\
= & \{\mathbf{stop}.\{a, b\} = \mathbf{buf}_0.a.b\text{, the above derivation}\} \\
& (\cup i : i \geq 0 : \mathbf{buf}_i.a.b) \\
= & \{\text{definition of } \mathbf{buf}\} \\
& \mathbf{buf}.a.b.
\end{array}
$$

Hence, $\mathbf{pr}\mathit{buf} = \mathbf{buf}.a.b$.

□

We discuss three programs in the notation introduced above as possible implementations of $\mathbf{buf}.a.b = (\cup k : k \geq 0 : \mathbf{buf}_k.a.b)$.

The first two programs come from [Kal]. The third one was inspired by [Kal].

For this purpose system $S = \langle\{a, b\}, \{\mathbf{buf}.a.b\}\rangle$ is compared to systems associated with these programs by means of testing relations. Observe that system S has no divergence and that no action of its alphabet can ever be disabled.

The first program is of the following form:

com $c_0(\{a, b\})$:
  sub $p : c_0$ **bus**
  $((a \mid p{\cdot}b)\,;\,(p{\cdot}a \mid b))^*$
moc.

The system associated with c_0 is

$$sys.c_0 = \langle \{a, b\}, \{(p\cdot)^i \mathbf{pr}(((a \mid p{\cdot}b)\,;\,(p{\cdot}a \mid b))^*) \mid i \geq 0\rangle.$$

The process associated with c_0 is $\mathbf{buf}.a.b$ (see Example A.4.3 for the calculation).
The second program is of the following form:

com $c_1(\{a, b\})$:
  sub $p : c_1$ **bus**
  $(a\,;\,p{\cdot}a \mid a\,;\,b \mid p{\cdot}b\,;\,b)^*$
moc.

The system associated with c_1 is

$$sys.c_1 = \langle \{a, b\}, \{(p\cdot)^i \mathbf{pr}((a\,;\,p{\cdot}a \mid a\,;\,b \mid p{\cdot}b\,;\,b)^*) \mid i \geq 0\rangle.$$

The process associated with c_1 is the least fixpoint of function f defined for $T \in \mathcal{T}.\{a, b\}$ by (see above)

$$f.T = (p{\cdot}T \mathbf{\ w\ pr}((a\,;\,p{\cdot}a \mid a\,;\,b \mid p{\cdot}b\,;\,b)^*))\restriction\{a, b\}.$$

We have $f^0.(\mathbf{stop}.\{a, b\}) = \mathbf{stop}.\{a, b\}$ and

$$\begin{aligned}
& f^1.(\mathbf{stop}.\{a, b\}) \\
= \quad & \{\text{ definition of } f\ \} \\
& (p{\cdot}\mathbf{stop}.\{a, b\} \mathbf{\ w\ pr}((a\,;\,p{\cdot}a \mid a\,;\,b \mid p{\cdot}b\,;\,b)^*))\restriction\{a, b\} \\
= \quad & \{\text{ definition of } p{\cdot}\ \} \\
& (\mathbf{stop}.\{p{\cdot}a, p{\cdot}b\} \mathbf{\ w\ pr}((a\,;\,p{\cdot}a \mid a\,;\,b \mid p{\cdot}b\,;\,b)^*))\restriction\{a, b\} \\
= \quad & \{\text{ definition of } \mathbf{w}, \text{ calculus}\ \} \\
& \mathbf{buf}_1.a.b.
\end{aligned}$$

We prove by induction that $f^i.(\mathbf{stop}.\{a, b\}) = \mathbf{buf}_i.a.b$, for $i \geq 1$.

Base $i = 1$
From the above derivation we have $f^1.(\mathbf{stop}.\{a, b\}) = \mathbf{buf}_1.a.b$.

Step $i \geq 1$
We derive

$$
\begin{array}{ll}
 & f^{i+1}.(\mathbf{stop}.\{a,b\}) \\
= & \quad \{\text{calculus}\} \\
 & f.(f^{i}.(\mathbf{stop}.\{a,b\})) \\
= & \quad \{\text{induction hypothesis}\} \\
 & f.(\mathbf{buf}_i.a.b) \\
= & \quad \{\text{definition of } f\} \\
 & (p{\cdot}\mathbf{buf}_i.a.b \ \mathbf{w}\ \mathbf{pr}((a\,;\, p{\cdot}a \mid a\,;\, b \mid p{\cdot}b\,;\, b)^*))\restriction\{a,b\} \\
= & \quad \{\text{definition of } p{\cdot}\} \\
 & (\mathbf{buf}_i.(p{\cdot}a).(p{\cdot}b)\ \mathbf{w}\ \mathbf{pr}((a\,;\, p{\cdot}a \mid a\,;\, b \mid p{\cdot}b\,;\, b)^*))\restriction\{a,b\} \\
= & \quad \{\text{by definition of } \mathbf{buf}_i \text{ we know that } p{\cdot}b\text{'s cannot precede } p{\cdot}a\text{'s and} \\
 & \quad \text{maximally } i \ p{\cdot}a\text{'s can precede a } p{\cdot}b, \text{ hence, by the definition of } \mathbf{w}, \\
 & \quad b\text{'s cannot precede } a\text{'s and maximally } i+1 \ a\text{'s can precede a } b\} \\
 & \mathbf{buf}_{i+1}.a.b.
\end{array}
$$

Furthermore,

$$
\begin{array}{ll}
 & (\cup i : i \geq 0 : f^{i}.(\mathbf{stop}.\{a,b\})) \\
= & \quad \{\text{calculus}\} \\
 & \mathbf{stop}.\{a,b\} \cup (\cup i : i \geq 1 : f^{i}.(\mathbf{stop}.\{a,b\})) \\
= & \quad \{\mathbf{stop}.\{a,b\} = \mathbf{buf}_0.a.b, \text{ the above derivation}\} \\
 & (\cup i : i \geq 0 : \mathbf{buf}_i.a.b) \\
= & \quad \{\text{definition of } \mathbf{buf}\} \\
 & \mathbf{buf}.a.b.
\end{array}
$$

Hence, $\mathbf{prc}_1 = \mathbf{buf}.a.b$.
The third program is of the following form:

$$
\begin{array}{l}
\mathbf{com}\ \ c_2(\{a,b\}) : \\
\qquad \mathbf{sub}\ p : c_2\ \ \mathbf{bus} \\
\qquad ((a\,;\, p{\cdot}a)^*\,;\, a\,;\, b\,;\, (p{\cdot}b\,;\, b)^*)^* \\
\mathbf{moc}.
\end{array}
$$

The system associated with c_2 is

$$sys.c_2 = \langle \{a,b\}, \{(p{\cdot})^i \mathbf{pr}(((a\,;\, p{\cdot}a)^*\,;\, a\,;\, b\,;\, (p{\cdot}b\,;\, b)^*)^*) \mid i \geq 0\rangle.$$

In the similar way as for c_1, it can be derived that $\mathbf{prc}_2 = \mathbf{buf}.a.b$.

Applying definitions from Section 2.4 we have, for $i \in \{0,1,2\}$,

$$sys.c_i \textbf{ psat } S.$$

Thus, all three programs are potential implementations of S.
But none of them is guaranteed to implement S. In the case of c_0 consider test

$$R_0 = \langle \{a,b\}, \{\mathbf{pr}(a\,;\, b)\} \rangle.$$

On account of the definition of $sys.c$ for a recursive c, we have that in the case of c_0:

- in the initial state only a is possible,
- after trace a, the infinite repetition of $p{\cdot}a\ p{\cdot}b$ excludes the execution of $(p{\cdot})^2a$ and $(p{\cdot})^2b$, and therefore also the execution of $(p{\cdot})^ia$ and $(p{\cdot})^ib$, for $i \leq 3$.

This implies that $\{\varepsilon\} \cup \{at \mid t \in \mathbf{t}(p{\cdot}\mathbf{buf}_1.a.b)\} \in pot.(sys.c_0 \parallel R_0)$. Thus we can calculate that

$$(a,\{b\}) \notin \mathbf{ac}(sys.c_0 \parallel R_0) \qquad \text{and} \qquad (a,\{b\}) \in \mathbf{ac}(S \parallel R_0).$$

This yields, on account of Definition 2.4.3,

$$\neg(sys.c_0 \textbf{ gsat } S).$$

The fact that $(a,\{b\})$ is not an acceptance of $sys.c_0 \parallel R_0$ is a consequence of the divergence in $sys.c_0$ after trace a. Observe, however, that b is not disabled in $sys.c_0$ after a.
In the case of c_1 consider test

$$R_1 = \langle \{a,b\}, \{\mathbf{pr}(a\,;\, a)\} \rangle.$$

On account of the definition of $sys.c$ for a recursive c, we have that in the case of c_1:

- in the initial state only a is possible,
- after trace a, the choice between b and $p{\cdot}a$ must be made,
- after trace $a\ p{\cdot}a$, the choice between $p{\cdot}b$ and $(p{\cdot})^2a$ must be made,
- after trace $a\ p{\cdot}a\ p{\cdot}b$, only b is possible.

This implies that $\{\varepsilon, a, a\ p{\cdot}a, a\ p{\cdot}a\ p{\cdot}b\} \in pot.(sys.c_1 \parallel R_1)$. Hence, we can conclude that

$$(a, \{a\}) \notin \mathbf{ac}(sys.c_1 \parallel R_1) \qquad \text{and} \qquad (a, \{a\}) \in \mathbf{ac}(S \parallel R_1).$$

On account of Definition 2.4.3, this implies

$$\neg(sys.c_1 \ \mathbf{gsat}\ S).$$

In this case, $(a, \{a\})$ is not an acceptance of $sys.c_1 \parallel R_1$, because a can be disabled in $sys.c_1$ after trace a.

Finally, in the case of c_2 consider, again, test R_0. We define

- $X_0 = \{a\}$,
- $X_{i+1} = \{t(p\cdot)^{i+1}a \mid t \in X_i\}$, for $0 \leq i$.

From the definition of $sys.c$ for a recursive c, we can conclude that in the case of c_2:

- in the initial state only a is possible,
- after trace t of X_i, for $i \geq 0$, the choice between $(p\cdot)^{i+1}a$ and $(p\cdot)^i b$ must be made.

We have then $\{\varepsilon\} \cup (\cup i : 0 \leq i : X_i) \in pot.(sys.c_2 \parallel R_0)$, and

$$(a, \{b\}) \notin \mathbf{ac}(sys.c_2 \parallel R_0) \qquad \text{and} \qquad (a, \{b\}) \in \mathbf{ac}(S \parallel R_0).$$

On account of Definition 2.4.3, this implies

$$\neg(sys.c_2 \ \mathbf{gsat}\ S).$$

Action b can be disabled in $sys.c_2$ after trace a and this causes that $(a, \{b\})$ is not an acceptance of $sys.c_2 \parallel R_0$.

Hence, none of the presented programs implements

$$\mathbf{buf}.a.b = (\cup k : k \geq 0 : \mathbf{buf}_k.a.b).$$

Whether a program that is a correct implementation of **buf** exists is still an open question (although we strongly suspect that the answer is negative).

We additionally compare the programs with each other. The calculations in the sequel follow from the reasoning similar to the one described above. We only shortly present the essential results.

Consider, again, test R_0. For this test we have

$$(a, \{b\}) \in \mathbf{ac}(sys.c_1 \parallel R_0),$$
$$(a, \{b\}) \notin \mathbf{ac}(sys.c_0 \parallel R_0), \text{ and}$$
$$(a, \{b\}) \notin \mathbf{ac}(sys.c_2 \parallel R_0).$$

Hence, on account of Definition 2.4.3,

$$\neg(sys.c_0 \textbf{ gsat } sys.c_1) \qquad \text{and} \qquad \neg(sys.c_2 \textbf{ gsat } sys.c_1).$$

When R_1 is considered, the following results are obtained

$(a, \{a\}) \in \mathbf{ac}(sys.c_2 \parallel R_1)$,
$(a, \{a\}) \notin \mathbf{ac}(sys.c_0 \parallel R_1)$, and
$(a, \{a\}) \notin \mathbf{ac}(sys.c_1 \parallel R_1)$.

Hence, on account of Definition 2.4.3,

$$\neg(sys.c_0 \textbf{ gsat } sys.c_2) \qquad \text{and} \qquad \neg(sys.c_1 \textbf{ gsat } sys.c_2).$$

Programs c_1 and c_2 are thus not comparable.
Furthermore, let R be a member of $Test.(sys.c_0)$, such that $\mathbf{pr}R \neq \mathbf{stop}.\{a, b\}$. Then we have

$(\varepsilon, \{a\}) \in \mathbf{ac}(sys.c_0 \parallel R)$,
$(\varepsilon, \{a\}) \in \mathbf{ac}(sys.c_1 \parallel R)$, and
$(\varepsilon, \{a\}) \in \mathbf{ac}(sys.c_2 \parallel R)$.

Let $t \in \mathbf{t}(sys.c_0 \parallel R)$, such that t is different from ϵ, and let $L \subseteq \{a, b\}^*$. Then

$$(t, L) \in \mathbf{ac}(sys.c_0 \parallel R) \Leftrightarrow \epsilon \in L,$$

and, on account of Property 2.2.5,

$$\begin{array}{l} \quad (t, L) \in \mathbf{ac}(sys.c_0 \parallel R) \\ \Rightarrow \\ \quad (t, L) \in \mathbf{ac}(sys.c_1 \parallel R) \wedge (t, L) \in \mathbf{ac}(sys.c_2 \parallel R). \end{array}$$

Hence, on account of Definition 2.4.3,

$$sys.c_1 \textbf{ gsat } sys.c_0 \qquad \text{and} \qquad sys.c_2 \textbf{ gsat } sys.c_0.$$

Neither program c_1 nor program c_2 is an implementation of c_0. This is not surprising, because at any time after the first a, c_0 can refuse both a and b by engaging in an unlimited internal activity (divergence), whereas c_1 and c_2 may only refuse either a or b, but not both (no divergence, only disabling).

A.5 A proof

In this section, the proof is presented of a statement made in Section 2.4 and related to the **sat** relation.
In Section 2.4, we defined for system S the set of relevant tests by

$$Test.S = \{R \mid R \in SY \wedge \mathbf{a}R = \mathbf{e}R = \mathbf{e}S \wedge \mathbf{pr}R \subseteq \mathbf{pr}S\}.$$

The following properties are used in our proof.

Property A.5.1

Let R, S_0, and S_1 be systems such that $\mathbf{pr}S_0 = \mathbf{pr}S_1$. Let $t \in \mathbf{t}(\mathbf{pr}(R \parallel S_0))$. Then

$$\begin{array}{l} \{X \mid X \in pot.R \wedge (\exists X_0 : X_0 \in pot.S_0 \wedge X \mathbf{w} X_0 \in pot.(R \parallel S_0) \\ \qquad\qquad : t \in X\restriction \mathbf{e}R \mathbf{w} X_0\restriction \mathbf{e}S_0)\} \\ = \\ \{X \mid X \in pot.R \wedge (\exists X_1 : X_1 \in pot.S_1 \wedge X \mathbf{w} X_1 \in pot.(R \parallel S_1) \\ \qquad\qquad : t \in X\restriction \mathbf{e}R \mathbf{w} X_1\restriction \mathbf{e}S_1)\}. \end{array}$$

□

Property A.5.2

Let R and S be systems. For $t \in \mathbf{t}(\mathbf{pr}(R \parallel S))$ and $L \subseteq (\mathbf{e}R \cup \mathbf{e}S)^*$ we have

$$\begin{array}{l} (\forall X' : X' \in pot.(R \parallel S)\restriction(\mathbf{e}R \cup \mathbf{e}S) \wedge t \in X' : \{v \mid tv \in X'\} \cap L \neq \emptyset) \\ \Rightarrow \\ (\forall X_0, X_1 : X_0 \in pot.R \wedge X_1 \in pot.S \wedge X_0 \mathbf{w} X_1 \in pot.(R \parallel S) \wedge \\ \qquad t \in X_0\restriction \mathbf{e}R \mathbf{w} X_1\restriction \mathbf{e}S \\ \qquad : \{v \mid (t\restriction \mathbf{e}R)v \in X\restriction \mathbf{e}R\} \cap L\restriction \mathbf{e}R \neq \emptyset). \end{array}$$

□

In the sequel, we prove that for systems S_0 and S_1

$$S_0 \mathbf{sat} S_1 \Leftrightarrow \mathbf{pr}S_0 \subseteq \mathbf{pr}S_1 \wedge (\forall R : R \in Test.S_1 : \mathbf{ac}(R \parallel S_0) \supseteq \mathbf{ac}(R \parallel S_1)).$$

By Definition 2.4.4 and Lemma 2.4.2 we immediately have the "implies"-part. We are left with the proof for the "follows from"-part.
On account of Property 2.2.5, we have

$$\begin{array}{l} \quad \mathbf{pr}S_0 \subseteq \mathbf{pr}S_1 \wedge (\forall R : R \in \mathit{Test}.S_1 : \mathbf{ac}(R \parallel S_0) \supseteq \mathbf{ac}(R \parallel S_1)) \\ \Leftrightarrow \\ \quad \mathbf{pr}S_0 = \mathbf{pr}S_1 \wedge (\forall R : R \in \mathit{Test}.S_1 : \mathbf{ac}(R \parallel S_0) \supseteq \mathbf{ac}(R \parallel S_1)). \end{array}$$

Finally, we show that for systems R, S_0, and S_1 such that $\mathbf{pr}S_0 = \mathbf{pr}S_1$,

$$\begin{array}{l} \quad (\forall R : R \in \mathit{Test}.S_1 : \mathbf{ac}(R \parallel S_0) \supseteq \mathbf{ac}(R \parallel S_1)) \\ \Rightarrow \\ \quad (\forall R : R \in SY : \mathbf{ac}(R \parallel S_0) \supseteq \mathbf{ac}(R \parallel S_1)), \end{array}$$

which completes our proof.

Assume for convenience that $\mathbf{a}R \cap \mathbf{a}S_i = \mathbf{e}R \cap \mathbf{e}S_i$, for $i = 0$ and $i = 1$.
Define $R' = \langle \mathbf{e}S_1, \{T \restriction \mathbf{e}S_1 \mid T \in \mathbf{p}R\} \rangle$.
Assume $(\forall R : R \in \mathit{Test}.S_1 : \mathbf{ac}(R \parallel S_0) \supseteq \mathbf{ac}(R \parallel S_1))$.
Let $t \in \mathbf{t}(\mathbf{pr}(R \parallel S_1)$ and $L \subseteq (\mathbf{e}R \cup \mathbf{e}S_1)^*$.
We derive

$$\begin{array}{ll} & (\forall X : X \in \mathit{pot}.(R \parallel S_1) \restriction (\mathbf{e}R \cup \mathbf{e}S_1) \wedge t \in X : \{v \mid tv \in X\} \cap L \neq \emptyset) \\ \Leftrightarrow & \{\text{definition of } \restriction\} \\ & (\forall X : X \in \mathit{pot}.(R \parallel S_1) \wedge t \in X \restriction (\mathbf{e}R \cup \mathbf{e}S_1) \\ & \quad : \{v \mid tv \in X \restriction (\mathbf{e}R \cup \mathbf{e}S_1)\} \cap L \neq \emptyset) \\ \Rightarrow & \{\text{definition of } R', \text{property of } \restriction, \text{set and predicate calculus}\} \\ & (\forall X' : X' \in \mathit{pot}.(R' \parallel S_1) \wedge t \restriction \mathbf{e}S_1 \in X' \restriction \mathbf{e}S_1 \\ & \quad : \{v \mid (t \restriction \mathbf{e}S_1)v \in X' \restriction \mathbf{e}S_1\} \cap L \restriction \mathbf{e}S_1 \neq \emptyset) \\ \Rightarrow & \{\text{assumption, Definition 2.2.4, } \mathbf{e}S_0 = \mathbf{e}S_1\} \\ & (\forall X' : X' \in \mathit{pot}.(R' \parallel S_0) \wedge t \restriction \mathbf{e}S_0 \in X' \restriction \mathbf{e}S_0 \\ & \quad : \{v \mid (t \restriction \mathbf{e}S_0)v \in X' \restriction \mathbf{e}S_0\} \cap L \restriction \mathbf{e}S_0 \neq \emptyset) \\ \Leftrightarrow & \{\text{Corollary 3.1.12}\} \\ & (\forall X', X_0 : X' \in \mathit{pot}.R' \wedge X_0 \in \mathit{pot}.S_0 \wedge X' \mathbf{w} X_0 \in \mathit{pot}.(R' \parallel S_0) \wedge \\ & \qquad t \restriction \mathbf{e}S_0 \in (X' \mathbf{w} X_0) \restriction \mathbf{e}S_0 \\ & \qquad : \{v \mid (t \restriction \mathbf{e}S_0)v \in (X' \mathbf{w} X_0) \restriction \mathbf{e}S_0\} \cap L \restriction \mathbf{e}S_0 \neq \emptyset) \\ \Leftrightarrow & \{\text{from the definition of } R' \text{ and because } \mathbf{e}S_0 = \mathbf{e}S_1, \text{ we have } \mathbf{a}R' \cap \mathbf{a}S_0 = \mathbf{e}S_0 \\ & \quad \text{and } \mathbf{a}R' = \mathbf{e}S_0, \text{ Lemma 1.1.12 and Property 1.1.17}\} \\ & (\forall X', X_0 : X' \in \mathit{pot}.R' \wedge X_0 \in \mathit{pot}.S_0 \wedge X' \mathbf{w} X_0 \in \mathit{pot}.(R' \parallel S_0) \wedge \\ & \qquad t \restriction \mathbf{e}S_0 \in X' \mathbf{w} X_0 \restriction \mathbf{e}S_0 \\ & \qquad : \{v \mid (t \restriction \mathbf{e}S_0)v \in X' \mathbf{w} X_0 \restriction \mathbf{e}S_0\} \cap L \restriction \mathbf{e}S_0 \neq \emptyset) \\ \Rightarrow & \{\text{Properties A.5.1 and A.5.2, properties of } \restriction \text{ and } \mathbf{w}\} \end{array}$$

$$
\begin{array}{ll}
& (\forall X, X_0 : X \in pot.R \wedge X_0 \in pot.S_0 \wedge X \mathbf{w} X_0 \in pot.(R \parallel S_0) \wedge \\
& \qquad t \in X \restriction \mathbf{e}R \mathbf{w} X_0 \restriction \mathbf{e}S_0 \\
& \qquad : \{v \mid tv \in X \restriction \mathbf{e}R \mathbf{w} X_0 \restriction \mathbf{e}S_0\} \cap L \neq \emptyset) \\
\Leftrightarrow & \quad \{\mathbf{a}R \cap \mathbf{a}S_0 = \mathbf{e}R \cap \mathbf{e}S_0, \text{ Lemma } 1.1.13\} \\
& (\forall X, X_0 : X \in pot.R \wedge X_0 \in pot.S_0 \wedge X \mathbf{w} X_0 \in pot.(R \parallel S_0) \wedge \\
& \qquad t \in (X \mathbf{w} X_0) \restriction (\mathbf{e}R \cup \mathbf{e}S_0) \\
& \qquad : \{v \mid tv \in (X \mathbf{w} X_0) \restriction (\mathbf{e}R \cup \mathbf{e}S_0)\} \cap L \neq \emptyset) \\
\Leftrightarrow & \quad \{\text{Corollary } 3.1.12\} \\
& (\forall X : X \in pot.(R \parallel S_0) \restriction (\mathbf{e}R \cup \mathbf{e}S_0) \wedge t \in X : \{v \mid tv \in X\} \cap L \neq \emptyset).
\end{array}
$$

Hence, on account of Definition 2.2.4,

$$\mathbf{ac}(R \parallel S_0) \supseteq \mathbf{ac}(R \parallel S_1).$$

References

[Alp] Alpern, B. and F.B. Schneider. Defining Liveness. *Information Processing Letters*, 21, 1985, pp. 181 – 185.

[Ash] Ashcroft, E.A. Proving Assertions about Parallel Programs. *Journal of Computer and System Sciences*, 10, 1975, pp. 110 – 135.

[Bar] Bartlett, K.A., R.A. Scantlebury, and P.T. Wilkinson. A Note on Reliable Full-Duplex Transmission over Half-Duplex Links. *Communications of the ACM*, 12 (5), 1969, pp. 260 – 261.

[Ber] Bergstra, J.A. and J.W. Klop. *Fixed Point Semantics in Process Algebras.* Report IW 206, Mathematisch Centrum, Amsterdam, 1982.

[Be0] Van Berkel, C.H. *Handshake Circuits: an Intermediary between Communicating Processes and VLSI.* Ph.D. thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology, 1992.

[Be1] Van Berkel, C.H. and R.W.J.J. Saeijs. Compilation of Communicating Processes into Delay-Insensitive Circuits. In *Proceedings of the 1988 IEEE International Conference on Computer Design: VLSI in Computers & Processors*, IEEE Computer Society, Washington DC, 1988, pp. 157 – 162.

[Be2] Van Berkel, C.H., J. Kessels, M. Roncken, R.W.J.J. Saeijs, and F. Schalij. The VLSI-programming Language Tangram and its Translation into Handshake Circuits. In *Proceedings of the 1991 European Design Automation Conference*, IEEE Computer Society, Los Alamitos, California, 1991, pp. 384 – 389.

[Bir] Birkhoff, G. *Lattice Theory.* American Mathematical Society, Providence, 1967 (AMS Colloquium Publications: vol. 25).

[Brz] Brzozowski, J.A. Derivatives of Regular Expressions. *Journal of the Association for Computing Machinery*, 11 (4), 1964, pp. 481 – 494.

[De0] De Nicola, R. Extensional Equivalences for Transition Systems. *Acta Informatica*, 24, 1987, pp. 211 – 237.

[De1] De Nicola, R. and M. Hennessy. Testing Equivalences for Processes. *Theoretical Computer Science*, 34, 1983, pp. 83 – 133.

[Dij] Dijkstra, E.W. and W.H.J. Feijen. *A Method of Programming.* Addison-Wesley, Reading, Massachusetts, 1988.

[Elr] Elrad, T. and N. Francez. Decomposition of Distributed Programs into Communication-Closed Layers. *Science of Computer Programming*, 2, 1982, pp. 155 – 173.

[Fra] Francez, N. *Fairness.* Springer-Verlag, New York, 1986.

[Gol] Goltz, U. and W. Reisig. The Non-sequential Behaviour of Petri Nets. *Information and Control*, 57, 1983, pp. 125 – 147.

[Hen] Hennessy, M. *Algebraic Theory of Processes.* The MIT Press, Cambridge, Massachusetts, 1988.

[Hoa] Hoare, C.A.R. *Communicating Sequential Processes.* Prentice Hall, New York, 1985.

[Hop] Hopcroft, J.E. and J.D. Ullman. *Formal Languages and their Relation to Automata.* Addison-Wesley, New York, 1969.

[Kal] Kaldewaij, A. *A Formalism for Concurrent Processes.* Ph.D. thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology, 1986.

[Klo] Kloosterhuis, W. *Livelock in Concurrent Processes.* Master's thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology, 1987.

[Lam] Lamport, L. Proving the Correctness of Multiprocess Programs. *IEEE Transactions on Software Engineering*, 3 (2), 1977, pp. 125 – 143.

[Leh] Lehmann, D., A. Pnueli, and J. Stavi. Impartiality, Justice and Fairness: the Ethics of Concurrent Termination. In O. Kariv, S. Even (eds.), *Proceedings of the 8th International Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science, vol. 115, Springer-Verlag, Berlin, 1981, pp. 264 – 277.

[Man] Manna, Z. and A. Pnuelli. *The Temporal Logic of Reactive and Concurrent Systems.* Springer-Verlag, New York, 1992.

[Ma0] Mazurkiewicz, A. *Semantics of Concurrent Systems: a Modular Fixed Point Trace Approach.* In G. Rozenberg (ed.), *Advances in Petri Nets*, Lecture Notes in Computer Science, vol. 188, Springer-Verlag, Berlin, 1985, pp. 353 – 375.

[Ma1] Mazurkiewicz, A., E. Ochmański, and W. Penczek. Concurrent Systems and Inevitability. *Theoretical Computer Science*, 64, 1989, pp. 281 – 304.

[Mi0] Milner, R. Fully Abstract Models of Typed λ-Calculi. *Theoretical Computer Science*, 4, 1977, pp. 1 – 22.

[Mi1] Milner, R. *A Calculus of Communicating Systems.* Lecture Notes in Computer Science, vol. 92, Springer-Verlag, New York, 1980.

[Old] Olderog, E.-R. and C.A.R. Hoare. Specification-Oriented Semantics for Communicating Processes. *Acta Informatica*, 23, 1986, pp. 9 – 66.

[Owi] Owicki, S. and L. Lamport. Proving Liveness Properties of Concurrent Programs. *ACM Transactions on Programming Languages and Systems*, 4 (3), 1982, pp. 455 – 495.

[Par] Park, D.M.R. *Concurrency and Automata on Infinite Streams.* In P. Deussen (ed.), *Proceedings of the 5th GI Conference on Theoretical Computer Science*, Lecture Notes in Computer Science, vol. 104, Springer-Verlag, Berlin, 1981, pp. 167 – 183.

[Paw] Parrow, J. *Fairness Properties in Process Algebra with Applications in Communication Protocol Verification.* Ph.D. thesis, Department of Computer Science, Uppsala University, 1985.

[Pet] Petri, C.A. *Non-sequential Processes.* ISF Report 77-05, Gesellschaft für Mathematik und Datenverarbeitung, Bonn, 1977.

[Plo] Plotkin, G. An Operational Semantics for CSP. In D. Björner (ed.), *Formal Description of Programming Concepts – II*, pp. 199 – 223, North-Holland Publishing Company, Amsterdam, 1983.

[Pra] Pratt, V. Modelling Concurrency with Partial Orders. *International Journal of Parallel Programming*, 15 (1), 1986, pp. 33 – 71.

[Rab] Rabin, M.O. and D. Scott. Finite Automata and their Decision Problems. *IBM Journal of Research and Development*, 3, 1959, pp. 114 – 125.

[Re0] Reisig, W. *Petri Nets.* EATCS Monograph on Theoretical Computer Science, Springer-Verlag, Berlin, 1985.

[Rel] Reisig, W. Partial Order Semantics versus Interleaving Semantics for CSP-like Languages and its Impact on Fairness. In Paredaens, J. (ed.), *Proceedings of the 11th International Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science, vol. 172, Springer-Verlag, Berlin, 1984, pp. 403 – 413.

[Rem] Rem, M. Concurrent Computations and VLSI Circuits. In Broy, M. (ed.), *Control Flow and Data Flow : Concepts of Distributed Programming*, Springer-Verlag, Berlin, 1985, pp. 399 – 437.

[Roz] Rozenberg, G. and P.S. Thiagarajan. Petri Nets : Basic Notions, Structure, Behaviour. In J.W. de Bakker, W.P. de Roever, and G. Rozenberg (eds.), *Proceedings of the ESPRIT/LPC Advanced School on Current Trends in Concurrency: Overviews and Tutorials*, Lecture Notes in Computer Science, vol. 224, Springer-Verlag, Berlin, 1986, pp. 585 – 668.

[Sne] Snepscheut, J.L.A. van de. *Trace Theory and VLSI Design*. Lecture Notes in Computer Science, vol. 200, Springer-Verlag, Berlin, 1985.

[Ver] Verhoeff, T. Eindhoven University of Technology. Private communication.

[Wec] Wechler, W. *Universal Algebra for Computer Scientists*. Springer-Verlag, Berlin, 1992.

[Win] Winskel, G. Event Structure Semantics for CCS and Related Languages. In M. Nielsen and E.M. Schmidt (eds.), *Proceedings of the 9th International Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science, vol. 140, Springer-Verlag, Berlin, 1982, pp. 561 – 576.

[Zwa] Zwaan, G. *Parallel Computations*. Ph.D. thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology, 1989.

Index

Samenvatting

Dit proefschrift behandelt twee methoden voor het verifiëren van de correctheid van een ontwerp ten opzichte van zijn specificatie. Ontwerpen en specificaties die wij beschouwen zijn gedefinieerd in termen van tracetheorie ([Rem],[Sne], en [Kal]). Ze worden gemodelleerd door *systemen* ([Klo] en [Zwa]). Een systeem beschrijft een mechanisme bestaande uit componenten die onderling en met de omgeving kunnen communiceren. Binnen een systeem wordt onderscheid gemaakt tussen interne en externe communicatieacties. De interne structuur van een systeem kan heel gecompliceerd zijn, maar voor zijn gebruiker is alleen het waarneembare (externe) gedrag van belang. Daar de interne structuur het externe gedrag kan beïnvloeden is het belangrijk dat dit tot uitdrukking komt in de specificatie. Het externe gedrag van een systeem wordt gespecificeerd door

- de verzameling van toegestane externe communicatieacties,
- de verzameling van alle *mogelijke* rijen communicatieacties (*safety*), en
- de verzameling van *gegarandeerde* voortzettingen van elke rij (*liveness*).

Een systeem kan verschillende liveness eigenschappen hebben, afhankelijk van de manier waarop het uitvoeren van parallelle acties wordt gemodelleerd. We spreken van *interleaving* als er ordeningen op de parallelle acties worden geïntroduceerd. Als de acties niet geordend worden spreken we van *true concurrency*. Om de formele verificatie mogelijk te maken wordt op systemen een partiële ordeningsrelatie geïntroduceerd, die alleen rekening houdt met het externe gedrag van systemen.

De eerste verificatiemethode is operationeel. Daarin wordt het externe gedrag van een systeem gekarakteriseerd door zijn werking in verschillende omgevingen uit te drukken in termen van toestanden en transities tussen de toestanden. Deze methode is compositioneel ten opzichte van de operatoren waarmee samengestelde systemen gemaakt kunnen worden: $\|$ (*parallelle samenstelling*) en $\restriction$ (*projectie*). Dat wil zeggen:

- als S een implementatie is van T dan is $S\restriction A$ een implementatie van $T\restriction A$,
- als bovendien S' is een implementatie van T' dan is $S \parallel S'$ een implementatie van $T \parallel T'$.

Het model waarop deze methode is gebaseerd is niet compositioneel. Bovendien, omdat de verificatie het testen in alle mogelijke omgevingen inhoudt, vereist het aantonen dat S een implementatie is van T niet-triviale argumenten.

Deze twee nadelen worden vermeden in de tweede verificatiemethode, ten koste echter van een beperking van het systeemdomein. In het model waarop deze methode is gebaseerd worden systemen op paren afgebeeld, die safety en liveness karakteriseren. Op deze paren wordt een partiële ordeningsrelatie gedefinieerd. De tweede methode komt overeen met de eerste voor het beperkte systeemdomein en is dus compositioneel in de hierboven genoemde zin.

Dankzij de keuze voor true concurrency, kan met behulp van beide verificatiemethoden onderscheid gemaakt worden tussen sequentiële en parallelle systemen.

De structuur van het proefschrift ziet er kort samengevat als volgt uit. In hoofdstuk 1 wordt een overzicht van de tracetheorie gegeven. Hier komen de voor dit proefschrift relevante begrippen *disabling* en *divergence* aan de orde, evenals een aanpassing van het *failures model* voor systemen. Disabling en divergence hebben te maken met de interne structuur van systemen. In het failures model wordt het parallellisme gemodelleerd door interleaving waardoor het onderscheid tussen sequentiële en parallelle systemen niet te maken is.

Beide verificatiemethoden en de modellen waarop deze zijn gebaseerd worden behandeld in hoofdstukken 2 en 3.

Hoewel voor het abstracte model waarop de tweede verificatiemethode is gebaseerd het systeemdomein moest worden beperkt, kan het model worden gebruikt voor het definiëren van de semantiek van CSP-achtige programmeertalen. In hoofdstuk 4 is dit gedaan voor c-Tangram.

In hoofdstuk 5 is de tweede verificatiemethode uitgebreid voor f-systemen, dat wil zeggen systemen met expliciete aannamen over *fairness*. Deze uitbreiding is echter niet compositioneel. Door de introductie van f-systemen kan fairness onafhankelijk van het parallellisme beschouwd worden.

In de appendix worden op hoofdstuk 2 aanvullende resultaten gepresenteerd: onder andere een vergelijking van het operationele model met een interleaving model en een bespreking van drie mogelijke implementaties van een buffer.

Curriculum vitae

Asia van de Mortel-Fronczak werd op 10 december 1958 geboren te Pińczów, Polen. Eveneens in Pińczów, behaalde zij in juni 1977 haar Atheneum B-diploma.

Een paar maanden later, in oktober 1977, begon haar informaticastudie aan de Stanisław Staszic Universiteit te Kraków (Krakau). In februari 1982 studeerde zij met lof af. Het afstudeeronderwerp was het ontwikkelen van software hulpmiddelen voor het ontwerpen en testen van microcomputer systemen.

Aan dezelfde universiteit was zij als assistent onderzoeker werkzaam bij vakgroep Informatica vanaf maart 1982 tot april 1985. In die periode bracht zij in het kader van de IAESTE drie maanden (van december 1982 tot maart 1983) door bij vakgroep Telecommunicatie aan de Technische Universiteit in Eindhoven. Onder de leiding van prof.dr. J. Arnbak werkte zij aan een analytische en numerieke oplossing voor een datacommunicatie probleem.

Van april 1985 tot juli 1990 was zij als assistent onderzoeker en van januari 1991 tot oktober 1992 als universitair docent werkzaam bij vakgroep Informatica aan de Technische Universiteit Eindhoven. Het in deze tijd uitgevoerde promotieonderzoek onder leiding van prof.dr. M. Rem heeft geleid tot dit proefschrift.

Stellingen

behorend bij het proefschrift

Models of Trace Theory Systems

van

Asia van de Mortel-Fronczak

Technische Universiteit Eindhoven

4 mei 1993

1. Voor systemen die divergentievrij en sequentieel zijn komt de 'implements'-relatie van het failures model overeen met de **sat**-relatie gedefinieerd in dit proefschrift.

2. Transitiesystemen lenen zich bij uitstek voor het definiëren van de operationele semantiek van CCS/CSP-achtige programmeertalen.

   lit. – Hennessy, M. *Algebraic Theory of Processes.* The MIT Press, Cambridge, Massachusetts, 1988.

   – Olderog, E.-R. and C.A.R. Hoare. Specification-Oriented Semantics for Communicating Processes. *Acta Informatica*, 23, 1986, pp. 9 – 66.

3. Door hun compositionaliteit sluiten de verificatiemethoden beschreven in dit proefschrift op natuurlijke wijze aan bij een bottom-up ontwerp van systemen.

4. Het tijdens executie laten testen van de geldigheid van asserties kan het formeel verifiëren van programma's niet vervangen.

   lit. – Meyer, B. From Structured Programming to Object-Oriented Design: the Road to Eiffel. *Structured Programming*, 1, 1989, pp. 19 – 39.

5. De definitie van de functies *div* en *mod* op p.40 in het boek "Verification of Sequential and Concurrent Programs" van Apt en Olderog is fout.

   lit. – Apt, K.R. and E.-R. Olderog. *Verification of Sequential and Concurrent Programs.* Springer-Verlag, New York, 1991.

6. Het is onjuist het breken van de Enigmacode uitsluitend aan Engelse wiskundigen toe te schrijven.

   lit. – Bertrand, G. *Enigma ou les Plus Grande Énigmes de la Guerre 1939 - 1945.* Plon, Paris, 1974.

   – Winterbotham, F.W. *The Ultra Secret.* Weidenfeld & Nicolson, London, 1974.

   – Hodges, A. *Alan Turing: the Enigma.* Burnett Books, London, 1983.

7. Het belang van het ontwikkelen van schriftelijke uitdrukkingsvaardigheden wordt door technische opleidingen niet voldoende onderkend.

8. Ten behoeve van het verkrijgen van vaardigheid in het afleiden van programma's zou in het wiskunde-onderwijs meer aandacht besteed moeten worden aan het afleiden van recurrente betrekkingen.

9. Een taal die samenstellingen van woorden toelaat noemen wij compositioneel. Niet-compositionele talen (zoals het Russisch) zijn minder geschikt voor het maken van cryptogrammen dan compositionele talen (zoals het Nederlands).

   lit. – Dik, S.C. en J.G. Kooij. *Algemene Taalwetenschap.* Uitgeverij Het Spectrum, Utrecht, 1979.

   – Verschuyl, H.J. *Cryptogrammatica: het Cryptogram als Taalspel.* Uitgeverij Kosmos, Utrecht, 1990.

10. Van de Nederlandse uitdrukkingen die betrekking hebben op Polen heeft "nog is Polen niet verloren" de minst negatieve betekenis. Dit geeft aan dat positieve kenmerken van Polen (in het verleden) weinig indruk gemaakt hebben op Nederlanders.

    lit. – Geerts, G. en H. Heestermans (redactie). *Van Dale Groot Woordenboek der Nederlandse Taal.* Van Dale Lexicografie bv, Utrecht, 1984.