

**stichting
mathematisch
centrum**



AFDELING INFORMATICA
(DEPARTMENT OF COMPUTER SCIENCE)

IW 209/82

SEPTEMBER

J.W. DE BAKKER & J.I. ZUCKER

PROCESSES AND THE DENOTATIONAL SEMANTICS OF CONCURRENCY

Preprint

kruislaan 413 1098 SJ amsterdam

Printed at the Mathematical Centre, 413 Kruislaan, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O.).

1980 Mathematics subject classification: 68B10, 68C05

1982 CR. Categories: D.1.3, F.1.2, F.3.2, F.3.3

Processes and the denotational semantics of concurrency^{*)}

by

J.W. de Bakker & J.I. Zucker^{**)}

ABSTRACT

A framework allowing a unified and rigorous definition of the semantics of concurrency is proposed. The mathematical model introduces processes as elements of process domains which are obtained as solutions of domain equations in the sense of Scott and Plotkin. Techniques of metric topology as proposed, e.g., by Nivat are used to solve such equations. Processes are then used as meanings of statements in languages with concurrency. Three main concepts are treated, viz. parallelism - arbitrary interleaving of sequences of elementary actions -, synchronization, and communication. These notions are embedded in languages which also feature classical sequential concepts such as assignment, tests, iteration or recursion, and guarded commands. In the definitions, a sequence of process domains of increasing complexity is used. The languages discussed include Milner's calculus for communicating systems and Hoare's communicating sequential processes. The paper concludes with a section with brief remarks on miscellaneous notions in concurrency, and two appendices with mathematical details.

KEY WORDS & PHRASES: *concurrency, parallelism, merge, synchronization, communication, recursion, denotational semantics, metric topology, Hausdorff metric, contractions, Banach's fixed point theorem*

^{*)} This report will be submitted for publication elsewhere.

^{**)} Department of Computer Science, State University of New York, Buffalo, U.S.A. Part of the work of this author was performed during his stay at the Department of Mathematics and Computer Science, Bar-Ilan University, Ramat Gan, Israel.

1. INTRODUCTION

The aim of this paper is to present a mathematical study of the semantics of a variety of language concepts in the area of *concurrency*. We shall be concerned with three fundamental notions in this field: *parallel composition*, *synchronization*, and *communication*, and we shall develop a general framework in which definitions and properties of these notions can be discussed in a systematic way.

The emphasis in the paper is on *definitions* - rather than on pragmatic use - of language concepts. We shall use the methodology of *denotational* semantics. "Denotational" should be contrasted here with "operational": The key idea of the former approach is that expressions in a programming language denote values in mathematical domains equipped with an appropriate structure, whereas in the latter the operations as prescribed by the language constructs are modelled by steps performed by some suitable abstract machine.

In the denotational semantics of sequential programming concepts, a central role is played by the notion of (state-transforming) *function*. Let us use Σ , with elements σ , for the set of *states*. For the present purposes, it suffices to define a state as a mapping from program variables x, y, \dots to values such as $0, 1, \dots$. The denotational meaning of a simple command such as the assignment statement $x := x+1$ is a function $\phi: \Sigma \rightarrow \Sigma$, defined by $\phi(\sigma) = \sigma'$, where $\sigma'(x) = \sigma(x)+1$, and $\sigma'(y) = \sigma(y)$ for all $y \neq x$. Also, the meaning of a composite command, formed by sequential composition ";", such as $x := x+1; y := x+y$ is obtained by forming the function composition $\phi_2 \circ \phi_1$, where ϕ_1 and ϕ_2 are the meanings of the statements $x := x+1$ and $y := x+y$, respectively. When we admit nondeterminacy, the situation changes

somewhat in that the meaning of a statement is now a function from states to sets of states with a certain structure. Using P for "power set of", we now use functions $\phi: \Sigma \rightarrow P(\Sigma)$. Here as well, composition is easy to define: $\phi_1 \circ \phi_2 = \lambda \sigma. \{\sigma' \mid \sigma' \in \phi_1(\sigma'') \text{ for some } \sigma'' \in \phi_2(\sigma)\}$, and no *essential* extension of the traditional view of a statement having a state transformation as its meaning is necessary. A fundamental change in this view is needed, however, for the denotational treatment of *parallel* composition. Let $S_1 \parallel S_2$ denote parallel execution of S_1 and S_2 : Statements S_1 and S_2 - in the example allowed to share their variables - are executed by arbitrary interleaving of the constituent elementary actions of S_1 and S_2 . Consider, for example, a simple program $(*)$: $(A_1; A_2) \parallel (B_1; B_2)$, with A_i, B_i elementary actions (such as $x := x+1$), and let ϕ_i, ψ_i be the respective meanings of A_i, B_i . Now what happens if we take the ϕ_i, ψ_i simply as functions: $\Sigma \rightarrow \Sigma$? We form the compositions $\phi = \phi_2 \circ \phi_1$, $\psi = \psi_2 \circ \psi_1$ and try to define a resulting function *merge* (ϕ, ψ) . Here we are stuck, since having formed the compositions ϕ, ψ , we no longer have available their respective operands ϕ_i, ψ_i . (Remember that what we want as resulting function is the union of the (six) possibilities $\phi_2 \circ \phi_1 \circ \psi_2 \circ \psi_1$, $\phi_2 \circ \psi_2 \circ \phi_1 \circ \psi_1, \dots, \psi_2 \circ \psi_1 \circ \phi_2 \circ \phi_1$.) In an operational approach, the problem does not arise in this form: A *trace* is kept of the computation, e.g. in the form of the (set of the) sequence(s) of elementary actions generated while executing the program, and the meaning of $S_1 \parallel S_2$ is simply the shuffle (in the language theoretic sense) of the traces corresponding to S_1 and S_2 . (Other operational approaches are also possible, see e.g. [31, 49]. However, they all involve suitably structured sequences of elementary steps.) This preserving of intermediate information in order to be able to describe the final result of interleaving is crucial for a proper treatment of parallelism, and is in fact what we shall do as well in our denotational approach. The basic idea is to extend the notion of function to that of *process*. Here "process" is a generic term, referring to a variety of mathematical objects which have one important property in common, viz. that they are constituted in some way from (possibly infinite) sets of (possibly infinite) sequences. For the example language considered above, the corresponding notion of process is an extension of that of state-transforming function in that it is still a function but now includes the information on how it was built up from the - possibly infinite - sequences of its elementary components. In this introduction we shall not be more precise about the notion of process. What we do underline is that in our theory a process is a semantic rather than a syntactic notion: it is a

feature of the mathematical model rather than of the program text

Section 2 of the paper presents the notion of process in some detail. A rigorous treatment of this requires some mathematical machinery involving tools from metric topology. A fundamental role is played by *equations* for *domains* of processes. Such equations are solved essentially by completion techniques - reminiscent of the way Cantor constructed the real numbers from the rationals. Next, the central *operations* upon processes are defined. We consider the convenience in formulating these definitions as an important accomplishment of the theory of processes. Processes are finite or infinite. Defining the operations for the finite cases requires specific attention; the infinite ones are each time obtained in a standard way by continuity arguments. Some of the more tedious mathematical arguments are relegated to the appendices; in section 2 we concentrate on those results which are necessary for an understanding of the central sections of our paper. For the reader who wants to skip *all* mathematical details we provide a brief summary of the relevant results at the end of the section. Sections 3 to 8 constitute the applied part of the paper. In these, it is shown how a rigorous and concise semantics can be designed for certain central notions in concurrency, by an appropriate synthesis of the use of processes with that of more traditional ideas of denotational semantics. Section 3 concentrates on flow of control: It considers a simple language with elementary actions, sequential composition and nondeterministic choice, and iteration or recursion. Adding parallel composition (" \parallel ") to this requires for its semantics a rather simple process domain, the so-called *uniform* processes. Iteration and recursion are dealt with in a relatively straightforward way by certain limit constructions. We already mention that an appeal to Banach's fixed point theorem will replace the familiar least fixed point approach of denotational semantics based on complete partially ordered sets. The section also discusses how the *yield* of a uniform process p can be derived from the set of all *paths* in p .

In section 4 we add *synchronization* to the language(s) of section 3. Synchronization restricts the set of all possible interleavings of sequences of elementary actions, and a general mechanism to model this is studied. Section 5 refines the theory by introducing the notion of state - suppressed in sections 3 and 4 - and assignments, and discusses the required extensions to the notions of processes and their yields. Processes are no longer uniform, but depend on the state as an argument, and the previous definitions have to be modified accordingly. As special feature we mention

that unbounded nondeterminacy can be dealt with without any additional measures. Section 6 combines the ideas of sections 4 and 5, in that synchronization is now considered for non-uniform processes. Among the topics studied are deadlock, and synchronization through guards in guarded commands. Section 7 extends synchronization to *communication*: At points of synchronization in the parallel execution values are passed from one process to another. A further extension of the notion of process is needed to deal with this. Two major examples of languages with communication are treated: Hoare's Communicating Sequential Processes ([34]), and Milner's Calculus for Communicating Systems ([44]). In section 8 we finally discuss some miscellaneous notions in concurrency, without providing a full treatment as was done in the preceding sections. In the appendices a number of mathematical details omitted in section 2 are filled in.

There is a vast amount of literature on concurrency, and a good part of these papers involve some discussion of the operational semantics of the notion(s) in concurrency. Our understanding of concurrency has been profoundly influenced by the work of R. Milner, starting with [42], continued in papers such as [30,40,43], and culminating in [44]. Though the latter work is primarily operational in spirit, there is still a lot in it which recalls its author's denotational period. Also, for an intuitive understanding of the central notions in concurrency it is an invaluable source. The various notions of process to be studied below will be introduced as solutions of domain equations. The introduction of equations of this type is due to D.Scott - dating back to perhaps the most famous equation for *reflexive* domains: $D = D \rightarrow D$ - and has been treated extensively in, e.g., [54] or, more recently, in [55]. A very nice textbook on denotational semantics in general and domain equations in particular is Stoy [57]. (A more introductory text on denotational semantics is Gordon [28]; many advanced topics are treated in Milne & Strachey [41].) Scott's theory did not include non-determinacy or concurrency, and an extension of his theory dealing with these concepts was proposed by Plotkin ([48]), later simplified somewhat by Smyth ([56]; c.f. also [39]). The first time we saw a domain equation intended to be used for modelling concurrency was in Bekic [12]. In the work of Plotkin and Smyth, domain equations are solved by category-theoretic methods which may be somewhat demanding for the uninitiated reader. We prefer to use other tools, viz. those of metric topology. The use of these has been advocated in recent years by M. Nivat and his colleagues, and applied successfully in a variety of applications having to do with infinite words or

infinite trees modelling infinite computations and the semantics of recursive program schemes with nondeterminacy [5,6,45,46]. The mathematical foundations of our work - as described in section 2 - owes a considerable debt to the work of Nivat's school - though the specific way we use topological completion techniques to solve equations seems to be new.

Our own first venture into the realm of (infinite) processes was De Bakker [9]. Lacking in that paper was a sound mathematical basis for the notion of process. The present topological treatment was first described in De Bakker & Zucker [11], reporting on research which was started during a most enjoyable stay of the first author at Bar-Ilan University and the Weizmann Institute during the summer of 1981.

Further references to the literature - in particular concerned with the various concepts in concurrency we shall encounter in these notes - will be given as we go along.

A preliminary version of this paper was used as lecture notes for the Fourth Advanced Course on Foundations of Computer Science, Amsterdam, June 1982. We are indebted to the students of this course for various questions and comments. We also acknowledge discussions with J.A. Bergstra, J.W. Klop, R. Kuiper, L. Lamport, J.J.Ch.Meyer, and G. Plotkin.

2. PROCESSES

In this section we show how processes p can be introduced as elements of domains P which are obtained as solutions of domain equations of the form $(*)$: $P = T(P)$. The techniques used to solve $(*)$ are taken from metric topology. A variety of equations $(*)$ is considered, determining a variety of process domains of increasing complexity. Furthermore, a number of operations upon processes are defined, viz. composition $(p_1 \circ p_2)$, union $(p_1 \cup p_2)$, and merge $(p_1 \parallel p_2)$, and various properties of these operations are presented. A few of the proofs of the supporting mathematical facts are not contained in this section but can be found in the appendix. A brief summary of the relevant results is given at the end of the section.

We begin by recalling a few basic facts from metric topology. We assume known the notions of metric space, Cauchy sequence (CS) in a metric space, limits and closed sets, completeness of a metric space, and the theorem stating that each metric space (M, d) can be completed to (i.e. isometrically embedded in) a complete metric space. Throughout our paper, we shall only consider spaces (M, d) such that the metric d has values in the interval $[0, 1]$.

These notions are sufficient to solve the first domain equation for processes. This equation is very simple, and introduced only for the sake of illustrating the method used in solving such equations. Let A be any set. We consider the equation

$$(2.1) \quad P = \{p_0\} \cup (A \times P)$$

where p_0 is the *nil* process, and " \times " is the usual cartesian product. Intuitively, it is not difficult to see that the (greatest) solution set P should consist of p_0 , all finite sequences of the form $\langle a_1, \langle a_2, \dots, \langle a_n, p_0 \rangle \dots \rangle \rangle$, for $n \geq 1$, together with all infinite sequences $\langle a_1, \langle a_2, \dots \rangle \rangle$. The role of the nil process p_0 may be somewhat unusual in this equation, in that it replaces the more familiar empty sequence. However, it will remain with us all through the paper, and we ask the reader to exercise some patience in trying to appreciate its use.

We now obtain the solution of (2.1) in a more rigorous manner:

DEFINITION 2.1. Let (P_n, d_n) , $n = 0, 1, \dots$, be a collection of metric spaces defined inductively by: $P_0 = \{p_0\}$, $d_0(p', p'') = 0$ (since

$p', p'' \in P_0 \iff p' = p'' = p_0$, $P_{n+1} = \{p_0\} \cup (A \times P_n)$, d_{n+1} is given by:
 $d_{n+1}(p', p'') = 0$ if $p' = p'' = p_0$, $d_{n+1}(p', p'') = 1$ if $p' = p_0$, $p'' \neq p_0$ or $p' \neq p_0$,
 $p'' = p_0$. Otherwise, $p' = \langle a_1, p_1 \rangle$, $p'' = \langle a_2, p_2 \rangle$ for some $a_1, a_2 \in A$, $p_1, p_2 \in P_n$, and we put

$$d_{n+1}(p', p'') = d_{n+1}(\langle a_1, p_1 \rangle, \langle a_2, p_2 \rangle) = \begin{cases} 1, & \text{if } a_1 \neq a_2 \\ \frac{1}{2} d_n(p_1, p_2), & \text{if } a_1 = a_2 \end{cases}.$$

It is not difficult to verify that d_n is indeed a metric on P_n . As next step, we define $P_\omega \stackrel{\text{df.}}{=} \bigcup_n P_n$ and $d \stackrel{\text{df.}}{=} \bigcup_n d_n$. E.g., take $p' = \langle a_1, \langle a_2, \langle a_3, p_0 \rangle \rangle \rangle$, $p'' = \langle a_1, \langle a_2, \langle a_3, \langle a_4, p_0 \rangle \rangle \rangle \rangle$. Then $d(p', p'') = d_m(p', p'')$ (any $m \geq 4$) $= \frac{1}{2} d_{m-1}(\langle a_2, \langle a_3, p_0 \rangle \rangle, \langle a_2, \langle a_3, \langle a_4, p_0 \rangle \rangle \rangle) = \dots = \frac{1}{8} d_{m-3}(p_0, \langle a_4, p_0 \rangle) = \frac{1}{8} * 1 = \frac{1}{8}$.

DEFINITION 2.2.

- $P_\omega = \bigcup_n P_n$, $d = \bigcup_n d_n$
- (P, d) is the *completion* of (P_ω, d) .

Standard properties of the completion technique yield that we may take P as consisting of P_ω together with all limit points $p = \lim_n p_n$, with $\langle p_n \rangle_n$ a Cauchy sequence such that $p_n \in P_n$. It is now straightforward to show that

LEMMA 2.3. P satisfies (2.1).

Proof. Let $P' \stackrel{\text{df.}}{=} \{p_0\} \cup (A \times P)$. We define isometries $\phi: P \rightarrow P'$, $\psi: P' \rightarrow P$ in the following manner. First we consider ϕ . If $p = p_0$, we take $\phi(p) = p_0$; clearly, $\phi(p) \in P'$ in that case. Otherwise, $p = \lim_n p_n$ with $\langle p_n \rangle_n$ a CS (if $p \in P_n$, for some $n \geq 1$, p is identified with a CS which is eventually constant), and we may assume without lack of generality that $p_n = \langle a, q_n \rangle$, for some a and all n , such that $\langle q_n \rangle_n$ is also a CS. Now let $q = \lim_n q_n$. We take $\phi(p) = \langle a, q \rangle$. We leave the definition of ψ , and verification that ϕ, ψ are indeed isometries (distance-preserving bijections) to the reader. \square

The trouble taken to solve (2.1) may seem somewhat inordinate. It was done this way to familiarize the reader with this style of argument - which will pay off later - rather than for the solution of this problem in its own right.

Processes p which are elements of sets P as defined (e.g.) by equation (2.1) have a *degree*, written as $\deg(p)$, and defined in

DEFINITION 2.4. $\deg(p_0) = 0$, $\deg(p) = n$ if $p \in P_n \setminus P_{n-1}$, for some $n \geq 1$, and $\deg(p) = \infty$, otherwise.

For processes p, q in P as defined in (2.1) we now give the definition of their *composition* $p \circ q$:

DEFINITION 2.5. $p \circ q$ is defined (by induction on $\deg(q)$)

- a. $p \circ p_0 = p$, $p \circ \langle a, q' \rangle = \langle a, p \circ q' \rangle$ if $\deg(\langle a, q' \rangle) < \infty$
- b. $p \circ \lim_i q_i = \lim_i (p \circ q_i)$, for q_i finite

Example: $\langle a_1, \langle a_2, p_0 \rangle \rangle \circ \langle a_3, p_0 \rangle = \langle a_3, \langle a_1, \langle a_2, p_0 \rangle \rangle \rangle$. We see that composition is (almost) concatenation in reverse order.

LEMMA 2.6.

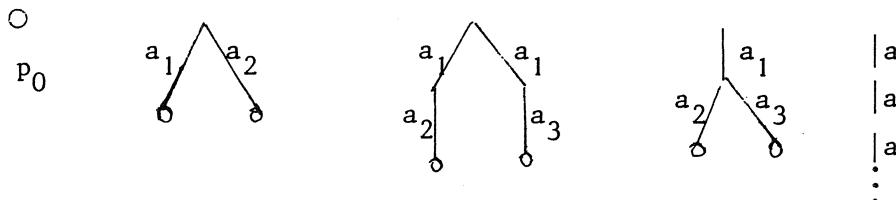
- a. If $\langle q_i \rangle_i$ is a CS then so are $\langle p \circ q_i \rangle_i$ (this justifies definition 2.5b) and $\langle q_i \circ p \rangle_i$.
- b. " \circ " is continuous in both arguments, i.e., $(\lim_i p_i) \circ q = \lim_i (p_i \circ q)$, and $p \circ \lim_i q_i = \lim_i (p \circ q_i)$, for all p_i, q_i such that $\langle p_i \rangle_i, \langle q_i \rangle_i$ are CS.
- c. " \circ " is associative


Proof. This lemma being a special case of later results, we omit its proof. \square

We now turn to the solution of a more interesting equation. The resulting processes are not simply (finite or infinite) sequences, but - roughly, a precise statement follows - *sets* of such sequences. We want to solve

$$(2.2) \quad P = \{p_0\} \cup P_c(A \times P)$$

where $P(\cdot)$ denotes all subsets of (\cdot) , and $P_c(\cdot)$ all *closed* subsets of (\cdot) (closed with respect to the metric to be introduced in a moment). Before going into the mathematical details, we consider a few simple examples. Possible elements of P are p_0 , $\{\langle a_1, p_0 \rangle, \langle a_2, p_0 \rangle\}$, $\{\langle a_1, \{\langle a_2, p_0 \rangle\} \rangle, \langle a_1, \{\langle a_3, p_0 \rangle\} \rangle\}$, $\{\langle a_1, \{\langle a_2, p_0 \rangle, \langle a_3, p_0 \rangle\} \rangle\}$, or $\{\langle a, \{\langle a, \{\langle a, \dots \rangle\} \rangle\} \rangle\}$. In pictures, these processes may be represented by



We see that these processes closely resemble (unordered) trees. However, as essential difference we have that "nodes" in a process have a set - rather than a multiset - of successors: A tree  has no corresponding process.

The topological treatment of the solution of (2.2) requires some preparations. Firstly, we extend distances d as follows:

DEFINITION 2.7. Let (M, d) be a metric space and let X, Y be subsets of M .

We define

$$a. d(x, Y) = \inf_{y \in Y} d(x, y)$$

$$b. d(X, Y) = \max\left(\sup_{x \in X} d(x, Y), \sup_{y \in Y} d(y, X)\right)$$

(By convention, $\inf \emptyset = 1$, $\sup \emptyset = 0$.)

Remark. The distance $d(X, Y)$ is the Hausdorff distance between sets. It should be distinguished from $d'(X, Y) = \inf_{x \in X, y \in Y} d(x, y)$, which does not determine a metric.

For the Hausdorff distance we have

LEMMA 2.8. Let (M, d) be a metric space, and let $P_c(M)$ be the collection of all closed subsets of M . Then $(P_c(M), d)$ is a metric space.

Proof. See [19] or [22]. \square

Remark. Given a metric space (M, d) , d is said to be an *ultrametric* on M if it satisfies the "strong triangle inequality" $\forall x, y, z \in M [d(x, z) \leq \max(d(x, y), d(y, z))]$. It is easy to see that if d is an ultrametric on M , then so is the induced Hausdorff metric on $P_c(M)$. It will follow (as can easily be shown) that every process domain P considered in this article will have an ultrametric with, moreover, $\max \{d(p, q) \mid p, q \in P\} = 1$.

An important technical result which plays a central role in the theory developed below is the following theorem of Hahn [29](cf. also [22]):

THEOREM 2.9. If (M, d) is complete then so is $(P_c(M), d)$. Also, for $\langle X_n \rangle_n$ a CS in $P_c(M)$, we have that

$$\lim_n X_n = \{x \mid x = \lim_n x_n, x_n \in X_n, \langle x_n \rangle_n \text{ a CS in } M\}.$$

Proof. See Appendix A.

We now proceed with the construction solving (2.2). We introduce metric spaces (P_n, d_n) , extending the techniques as applied before with sets and their (Hausdorff) distances:

DEFINITION 2.10. The collection of metric spaces (P_n, d_n) , $n = 0, 1, \dots$, is defined by $P_0 = \{p_0\}$, $d_0(p', p'') = 0$, $P_{n+1} = \{p_0\} \cup P(A \times P_n)$, $d_{n+1}(p', p'')$ is as before for $p' = p_0$ or $p'' = p_0$. Otherwise, $p' = X \subseteq A \times P_n$, $p'' = Y \subseteq A \times P_n$, and we take $d_{n+1}(X, Y)$ as the Hausdorff distance induced by the distance between points $d_{n+1}(x, y)$, where (as before), for $x = \langle a_1, p_1 \rangle$, $y = \langle a_2, p_2 \rangle$

$$d_{n+1}(x, y) = \begin{cases} 1, & \text{if } a_1 \neq a_2 \\ \frac{1}{2} d_n(p_1, p_2), & \text{if } a_1 = a_2 \end{cases}.$$

Example. Take $a_2 \neq a_3$. Then $d_2(\{\langle a_1, \{\langle a_2, p_0 \rangle, \langle a_3, p_0 \rangle\} \rangle\}, \{\langle a_1, \{\langle a_2, p_0 \rangle\} \rangle, \langle a_1, \{\langle a_3, p_0 \rangle\} \rangle\}) = \frac{1}{2}$.

As before, we take $P_\omega = \bigcup_n P_n$, $d = \bigcup_n d_n$, and (P, d) is defined as the completion of (P_ω, d) . We have

THEOREM 2.11. $P = \{p_0\} \cup P_c(A \times P)$, where $P_c(\cdot)$ stands for all subsets of (\cdot) which are closed with respect to the metric d .

The proof needs a definition and a lemma.

DEFINITION 2.12.

- Let $p \in P_\omega$. We define $p^{(n)}$, $n = 0, 1, \dots$, by: If $p = p_0$ then $p^{(n)} = p_0$, $n = 0, 1, \dots$. Otherwise, $p^{(0)} = p_0$, $p^{(n+1)} = \{\langle a, q^{(n)} \rangle \mid \langle a, q \rangle \in p\}$.
- Let $p \in P \setminus P_\omega$. Then $p = \lim_i p_i$, $p_i \in P_i$, $\langle p_i \rangle_i$ a CS. We then put $p^{(n)} = \lim_i p_i^{(n)}$.
- For $X \subseteq A \times P$ we put $X^{(n+1)} = \{\langle a, p^{(n)} \rangle \mid \langle a, p \rangle \in X\}$, $n = 0, 1, \dots$.

LEMMA 2.13.

- For each p , $p = \lim_n p^{(n)}$
- For $X \subseteq A \times P$, $\langle X^{(n)} \rangle_n$ is a CS and $\lim_n X^{(n)} = \bar{X}$, where \bar{X} is the closure of X . Hence, for X closed, $X = \lim_n X^{(n)}$

Proof. We only prove part b. Clearly, for $m < n$, $d(X^{(n)}, X^{(m)}) \leq 1/2^m$, and we see that $\langle X^{(n)} \rangle_n$ is a CS. We now show that $X \subseteq \lim_n X^{(n)}$. Let $\langle a, p \rangle \in X$. Then $\langle a, p \rangle = \langle a, \lim_n p^{(n)} \rangle = \lim_n \langle a, p^{(n)} \rangle \in \lim_n X^{(n)}$. Each $X^{(n)}$ is closed

in P_{n+1} (all subsets of each P_n are closed, since distances between points are at least $1/2^n$ and so there are no non-trivial CS in P_n); hence, $\lim_n X^{(n)}$ exists and is closed. From this and $X \subseteq \lim_n X^{(n)}$ it follows that $\bar{X} \subseteq \lim_n X^{(n)}$. Conversely, let $p \in \lim_n X^{(n)}$. By theorem 2.9, $p = \lim_n p_n$, where $p_n \in X^{(n)}$, $\langle p_n \rangle_n$ a CS. Hence, $p_n = q_n^{(n)}$ for some $q_n \in X$. Then $p = \lim_n q_n$, i.e., p belongs to the closure \bar{X} of X .

We now prove theorem 2.11. Similarly to what we did in the proof of lemma 2.3, we show that P satisfies (2.2) by establishing an isometry between the spaces P and $P' \stackrel{\text{df}}{=} \{p_0\} \cup P_c(A \times P)$. We define two bijections $\phi: P \rightarrow P'$, $\psi: P' \rightarrow P$, as follows:

- (i) If $p = p_0$, then $\phi(p) = p_0$. Otherwise, $p = \lim_n p_n$, $p_n \in P_n$, $\langle p_n \rangle_n$ a CS, $p_n \neq p_0$ for n sufficiently large. For these n , by the definition of P_n we have that p_n is a subset of $A \times P_{n-1}$, hence closed in $A \times P$; thus, $\langle p_n \rangle_n$ is a CS of closed sets in $A \times P$. We now take for $\phi(p)$ the closed subset of $A \times P$ which equals $\lim_n p_n$.
- (ii) If $p' = p_0$ then $\psi(p') = p_0$. Otherwise, take $p' = X \in P_c(A \times P)$. By Lemma 2.13b, $X = \lim_n X^{(n)}$. For each $n > 0$, put $p_n = X^{(n)} \in P_n$. Since $\langle X^{(n)} \rangle_n$ is a CS in P' , $\langle p_n \rangle_n$ is a CS in P . So we define $\psi(p') = \lim_n p_n$.

We leave it to the reader to verify that ϕ, ψ are the required isometric mappings. This concludes the proof of theorem 2.11. \square

We proceed with the introduction of the operations " \circ ", " \cup ", " \parallel " for processes p in P solving (2.2). By the preceding theory we know that for each process p , either p is p_0 , or p is finite and $p = X \in P_c(A \times P)$, or p is infinite and $p = \lim_i p^{(i)}$, $\langle p^{(i)} \rangle_i$ a CS, with $p^{(i)} \in P_i$, $i = 0, 1, \dots$.

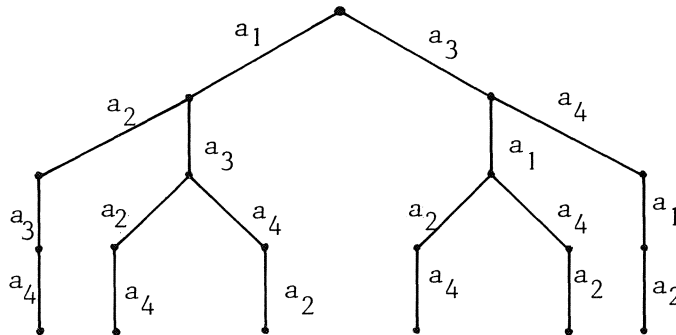
DEFINITION 2.14. Let $X, Y \in P_c(A \times P)$ with $\deg(X), \deg(Y) < \infty$.

- a. (composition) $p \circ p_0 = p$, $p \circ X = \{p \circ x \mid x \in X\}$, $p \circ \langle a, q \rangle = \langle a, p \circ q \rangle$, and $p \circ \lim_i q^{(i)} = \lim_i (p \circ q^{(i)})$.
- b. (union) $p_0 \cup p = p \cup p_0 = p$, $X \cup Y$ is the set-theoretic union of the two sets X, Y . Also, $(\lim_i p^{(i)}) \cup (\lim_j q^{(j)}) = \lim_k (p^{(k)} \cup q^{(k)})$.
- c. (merge) $p \parallel p_0 = p_0 \parallel p = p$, $X \parallel Y = \{X \parallel y \mid y \in Y\} \cup \{x \parallel Y \mid x \in X\}$, $X \parallel \langle a, p \rangle = \langle a, X \parallel p \rangle$, $\langle a, p \rangle \parallel X = \langle a, p \parallel X \rangle$, and $(\lim_i p^{(i)}) \parallel (\lim_j q^{(j)}) = \lim_k (p^{(k)} \parallel q^{(k)})$.

$$\begin{aligned} \text{Example. } p_1 \parallel p_2 &\stackrel{\text{df.}}{=} \{ \langle a_1, \{ \langle a_2, p_0 \rangle \} \rangle \parallel \{ \langle a_3, \{ \langle a_4, p_0 \rangle \} \} \} = \\ &= \{ \langle a_1, \{ \langle a_2, p_0 \rangle \} \parallel p_2 \rangle \} \cup \{ \langle a_3, p_1 \parallel \{ \langle a_4, p_0 \rangle \} \rangle \} = \\ &= \{ \langle a_1, \{ \langle a_2, p_2 \rangle \} \} \cup \{ \langle a_3, \{ \langle a_2, p_0 \rangle \} \parallel \{ \langle a_4, p_0 \rangle \} \rangle \}, \\ &\quad \langle a_3, \{ \langle a_4, p_1 \rangle \} \} \cup \{ \langle a_1, \{ \langle a_2, p_0 \rangle \} \parallel \{ \langle a_4, p_0 \rangle \} \rangle \rangle \} = \dots = \\ &= \{ \langle a_1, \{ \langle a_2, \{ \langle a_3, \{ \langle a_4, p_0 \rangle \} \rangle \} \rangle \rangle \rangle, \\ &\quad \langle a_3, \{ \langle a_2, \{ \langle a_4, p_0 \rangle \} \rangle, \langle a_4, \{ \langle a_2, p_0 \rangle \} \rangle \rangle \rangle \rangle, \\ &\quad \langle a_3, \dots \rangle \}. \end{aligned}$$

(The reader should compare this with the (language-theoretic) *shuffle* of two words a_1a_2 and a_3a_4 , yielding a set of six words $\{a_1a_2a_3a_4, a_1a_3a_2a_4, \dots, a_3a_4a_1a_2\}$.)

The following picture describes the result:



Definition 2.14 is justified in

LEMMA 2.15.

- a. For finite $q, q', d(p \circ q, p \circ q') \leq d(q, q')$
- b. For finite q_n , if $\langle q_n \rangle_n$ is a CS then so is $\langle p \circ q_n \rangle_n$
(Hence, the definition $p \circ q = \lim_n (p \circ q^{(n)})$ is well-formed)
- c. Part a holds for all q, q'
- d. If $q_n \rightarrow q$ then $p \circ q_n \rightarrow p \circ q$ (" \circ " is continuous in its second argument)
- e. For finite $p, q, p', q', d(p \cup p', q \cup q') \leq \max(d(p, q), d(p', q'))$
- f. For finite p_n, q_n , if $\langle p_n \rangle_n, \langle q_n \rangle_n$ are CS, then so is $\langle p_n \cup q_n \rangle_n$
(Hence, the definition $p \cup q = \lim_n (p^{(n)} \cup q^{(n)})$ is well-formed)
- g. Part f holds for all p, p', q, q'
- h. If $p_n \rightarrow p, q_n \rightarrow q$ then $p_n \cup q_n \rightarrow p \cup q$ (" \cup " is continuous in both arguments)
- i. For finite $p, q, q', p', d(p \parallel q, p' \parallel q') \leq \max(d(p, p'), d(q, q'))$
- j-l. Similarly to f-h for \parallel
- m. " \circ " is continuous in its first argument
- n. " \circ ", " \cup ", " \parallel " are associative, " \cup " and " \parallel " are commutative.

Proof. See Appendix B. \square

We continue with the consideration of domain equations which determine more complex processes. Calling processes in (2.2) *uniform*, we consider the non-uniform processes defined in

$$(2.3) \quad P = \{p_0\} \cup (A \rightarrow P_c(B \times P))$$

Processes p are now (either p_0 or) *functions*, such that for each a , $p(a)$ is a closed set $\{\dots, \langle b_i, p_i \rangle, \dots\}_{i \in I}$, where the index set I depends on a : $I = I(a)$. The solution of (2.3) is very similar to the ones given above. A new element is the distance between functions. We give

DEFINITION 2.16. The collection of spaces (P_n, d_n) , $n = 0, 1, \dots$, is defined as follows: P_0 and d_0 are as before. $P_{n+1} = \{p_0\} \cup (A \rightarrow P(B \times P_n))$, $d_{n+1}(p', p'')$ is as before for $p' = p_0$ or $p'' = p_0$. Otherwise, $d_{n+1}(p', p'') = \sup_{a \in A} d_{n+1}(p'(a), p''(a))$, where the distance between the sets $p'(a), p''(a)$ is the usual Hausdorff distance induced by the distance between points $d_{n+1}(\langle b_1, p_1 \rangle, \langle b_2, p_2 \rangle)$ given by

$$d_{n+1}(\langle b_1, p_1 \rangle, \langle b_2, p_2 \rangle) = \begin{cases} 1, & \text{if } b_1 \neq b_2 \\ \frac{1}{2} d_n(p_1, p_2), & \text{if } b_1 = b_2. \end{cases}$$

As before, d_n determines a metric on P_n , P_ω is defined as $\bigcup_n P_n$, $d = \bigcup_n d_n$, and (P, d) is the completion of (P_ω, d) . We have

THEOREM 2.17. $P = \{p_0\} \cup (A \rightarrow P_c(B \times P))$.

Proof. By appropriately adapting the proof of theorem 2.11. For example, we treat the isometry $\phi: P \rightarrow P'$, where $P' \stackrel{\text{df}}{=} \{p_0\} \cup (A \rightarrow P_c(B \times P))$. Let $p = \lim_n p_n$, $\langle p_n \rangle_n$ a CS in P . We indicate how to obtain $\phi(p)$ as a function in $(A \rightarrow P_c(B \times P))$. Take any $a \in A$. Since $\langle p_n \rangle_n$ is a CS, so is $\langle p_n(a) \rangle_n$. As CS of closed sets, $\langle p_n(a) \rangle_n$ has as limit a closed set, say X_a , where $X_a \subseteq B \times P$. Now put $\phi(p) = \lambda a. X_a$. We have to check (i) ϕ is well defined, i.e., if $(p =) \lim_n p_n = \lim_n q_n$, then $\lim_n p_n(a) = \lim_n q_n(a)$, (ii) ϕ is 1-1, i.e., $\phi(p) = \phi(q) \Rightarrow p = q$, (iii) ϕ is onto, and (iv) ϕ preserves distances. We treat only (ii). Assume that, for all a , $\lim_n p_n(a) = \lim_n q_n(a)$. To

show $p = q$, i.e., $\lim_n p_n = \lim_n q_n$. Since $\langle p_n \rangle_n, \langle q_n \rangle_n$ are CS, we have $\forall \epsilon \exists N \forall m, n \geq N [d(p_m, p_n) < \epsilon/2, d(q_m, q_n) < \epsilon/2]$. Thus, (*) $\forall m, n \geq N \forall a [d(p_m(a), p_n(a)) < \epsilon/2]$, (**) $\forall m, n \geq N \forall a [d(q_m(a), q_n(a)) < \epsilon/2]$. Letting $m \rightarrow \infty$ in (*), (**) we have $p_m(a) \rightarrow p(a), q_m(a) \rightarrow q(a)$. Thus $\forall n \geq N \forall a [d(p_n(a), p(a)) \leq \epsilon/2, d(q_n(a), q(a)) \leq \epsilon/2]$. From this, since $p(a) = q(a)$, we obtain $\forall n \geq N [d(p_n(a), q_n(a)) \leq \epsilon]$. Taking sup over all a we get $\forall n \geq N [d(p_n, q_n) \leq \epsilon]$. By a standard argument then $d(p, q) \leq \epsilon$. Since this holds for any ϵ we conclude that $p = q$. \square

The operations " \circ ", " \cup ", " \parallel " can be extended to non-uniform processes.

DEFINITION 2.18. We only consider processes of finite nonzero degree, the treatment of the remaining cases being the usual one.

- a. (composition) $p \circ \lambda a.X = \lambda a.(p \circ X)$, where $p \circ X = \{p \circ x \mid x \in X\}$, and $p \circ \langle b, q \rangle = \langle b, p \circ q \rangle$
- b. (union) $(\lambda a.X) \cup (\lambda a.Y) = \lambda a.(X \cup Y)$
- c. (merge) $(\lambda a.X) \parallel (\lambda a.Y) = \lambda a.(\{x \parallel (\lambda a.Y) \mid x \in X\} \cup \{\lambda a.X \parallel y \mid y \in Y\})$
where $\langle b, p \rangle \parallel (\lambda a.Y) = \langle b, p \parallel \lambda a.Y \rangle$, and $(\lambda a.X) \parallel \langle b, q \rangle = \langle b, (\lambda a.X) \parallel q \rangle$

Remark. Observe the difference between clauses b and c, in that we do *not* put $(\lambda a.X) \parallel (\lambda a.Y) = \lambda a.(X \parallel Y)$ (with $X \parallel Y$ defined appropriately).

In other words, though we have, for $p, q \neq p_0$, that $p \cup q = \lambda a.(p(a) \cup q(a))$, for $p \parallel q$ we do not have $p \parallel q = \lambda a.(p(a) \parallel q(a))$ but, instead, $p \parallel q = \lambda a.((p(a) \parallel q) \cup (p \parallel q(a)))$.

Operations " \circ ", " \cup " and " \parallel " for non-uniform processes satisfy the natural extension of Lemma 2.15:

LEMMA 2.19. As Lemma 2.15, but now for the operations as given in definition 2.18.

Proof. Left to the reader.

The last equation in the list of domain equations is

$$(2.4) \quad P = \{p_0\} \cup (A \rightarrow P_c((B \times P) \cup (C \rightarrow P))).$$

We only give the definition of the metric spaces (P_n, d_n) , leaving elaboration of the details concerning the isometries necessary to establish (2.4) to the reader. We have

DEFINITION 2.20. The metric spaces (P_n, d_n) , $n = 0, 1, \dots$, are defined by: P_0, d_0 are as before, $P_{n+1} = \{p_0\} \cup (A \rightarrow P((B \times P_n) \cup (C \rightarrow P_n)))$, $d_{n+1}(p', p'')$ is as before for $p' = p_0$ or $p'' = p_0$. Otherwise, $d_{n+1}(p', p'') = \sup_{a \in A} d_{n+1}(p'(a), p''(a))$, where $d_{n+1}(X, Y)$ is the Hausdorff distance between sets induced by the distance between points $d_{n+1}(x, y)$, where $d_{n+1}(\langle b, p \rangle, \lambda c.p') = 1 = d_{n+1}(\lambda c.p', \langle b, p \rangle)$, $d_{n+1}(\langle b_1, p_1 \rangle, \langle b_2, p_2 \rangle)$ is as usual, and $d_{n+1}(\lambda c.p_1, \lambda c.p_2) = \sup_{c \in C} d_n(p_1, p_2)$.

The operations for $p \in P$, with P solving (2.4) are given in

DEFINITION 2.21. We only consider processes of finite nonzero degree.

- a. $p \circ \lambda a.X = \lambda a.(p \circ X)$, $p \circ X = \{p \circ x \mid x \in X\}$, $p \circ \langle b, q \rangle = \langle b, p \circ q \rangle$, $p \circ \lambda c.p' = \lambda c.(p \circ p')$
- b. \cup : Omitted.
- c. $(\lambda a.X) \parallel (\lambda a.Y) = \lambda a.(\{x \parallel (\lambda a.Y) \mid x \in X\} \cup \{(\lambda a.X) \parallel y \mid y \in Y\})$, where $\langle b, p \rangle \parallel \lambda a.Y = \langle b, p \parallel \lambda a.Y \rangle$ and similarly for $(\lambda a.X) \parallel \langle b, p \rangle$, $(\lambda c.p') \parallel (\lambda a.Y) = \lambda c.(p' \parallel \lambda a.Y)$, and similarly for $(\lambda a.X) \parallel (\lambda c.p')$.

As the last lemma of this section we claim

LEMMA 2.22. The operations " \circ ", " \cup ", " \parallel " have the usual properties.

Proof. Omitted. \square

Having arrived at the end of this section, we summarize the main results:

1. Process domains P are obtained as solutions of equations of the form

- a. $P = \{p_0\} \cup (A \times P)$
- b. $P = \{p_0\} \cup P_c(A \times P)$, where $P_c(\cdot)$ stands for all closed subsets of (\cdot)
- c. $P = \{p_0\} \cup (A \rightarrow P_c(B \times P))$ (idem)
- d. $P = \{p_0\} \cup (A \rightarrow P_c((B \times P) \cup (C \rightarrow P)))$ (idem)

2. Processes p are either $\text{nil}(p_0)$, or finite and of finite degree $\text{deg}(p)$, or infinite and (topological) limit of a sequence $\langle p^{(i)} \rangle_i$ with $p^{(i)}$ finite. (For the definitions of the $p^{(i)}$ see point 5 below.)

3. Operations upon processes are composition (" \circ "), union (" \cup ") and merge (" \parallel "). They are defined as follows (\cup, \parallel only for process domains solving b, c, d above; X, Y are always finite elements of $P_c(\cdot)$):

3.1. $p \circ q$ is defined by induction on $\deg(q)$:

$$p \circ p_0 = p, \quad p \circ X = \{p \circ x \mid x \in X\}, \quad p \circ \langle a, q \rangle = \langle a, p \circ q \rangle, \quad p \circ \lambda a.X = \lambda a.(p \circ X), \\ p \circ \langle b, q \rangle = \langle b, p \circ q \rangle, \quad p \circ \lambda c.q = \lambda c.p \circ q, \quad p \circ \lim_i q^{(i)} = \lim_i (p \circ q^{(i)})$$

3.2. $p \cup q$ is defined by

$$p \cup p_0 = p_0 \cup p = p, \quad X \cup Y \text{ is the set-theoretic union of } X \text{ and } Y, \\ (\lambda a.X) \cup (\lambda a.Y) = \lambda a.(X \cup Y), \quad (\lim_i p^{(i)}) \cup (\lim_j q^{(j)}) = \lim_k (p^{(k)} \cup q^{(k)})$$

3.3. $p \parallel q$ is defined by induction on $\deg(p) + \deg(q)$:

$$p \parallel p_0 = p_0 \parallel p = p, \quad X \parallel Y = \{x \parallel y \mid x \in X\} \cup \{X \parallel y \mid y \in Y\}, \\ (\lambda a.X) \parallel (\lambda a.Y) = \lambda a.(\{x \parallel \lambda a.Y \mid x \in X\} \cup \{\lambda a.X \parallel y \mid y \in Y\}), \\ \langle a, p \rangle \parallel Y = \langle a, p \parallel Y \rangle, \quad Y \parallel \langle a, p \rangle = \langle a, Y \parallel p \rangle, \\ \langle b, p \rangle \parallel (\lambda a.Y) = \langle b, p \parallel \lambda a.Y \rangle, \text{ and similarly for } (\lambda a.Y) \parallel \langle b, p \rangle \\ (\lambda c.q) \parallel (\lambda a.Y) = \lambda c.(q \parallel (\lambda a.Y)), \text{ and similarly for } (\lambda a.Y) \parallel (\lambda c.q), \\ (\lim_i p^{(i)}) \parallel (\lim_j q^{(j)}) = \lim_k (p^{(k)} \parallel q^{(k)}).$$

4. The above operations are continuous and satisfy the usual properties such as commutativity (\cup, \parallel) , associativity (\circ, \cup, \parallel) , etc.

5. With respect to each of the equations a to d, $p_0^{(n)} = p_0$, $n = 0, 1, \dots$, and, for $p \neq p_0$, $p^{(0)} = p_0$.

Moreover, for $n = 0, 1, \dots$,

$$(\text{For a}) \quad p^{(n+1)} = \langle a, q^{(n)} \rangle, \text{ where } p = \langle a, q \rangle$$

$$(\text{For b}) \quad p^{(n+1)} = \{\langle a, q^{(n)} \rangle \mid \langle a, q \rangle \in p\}$$

$$(\text{For c}) \quad p^{(n+1)} = \lambda a. \{\langle b, q^{(n)} \rangle \mid \langle b, q \rangle \in p(a)\}$$

$$(\text{For d}) \quad p^{(n+1)} = \lambda a. (\{\langle b, q^{(n)} \rangle \mid \langle b, q \rangle \in p(a)\} \cup \{\lambda c.q^{(n)} \mid \lambda c.q \in p(a)\}).$$

3. FLOW OF CONTROL: MERGE WITH ITERATION OR RECURSION

In this section we introduce the first two of the series of languages studied in sections 3-8. Both languages have elementary actions, sequential composition, nondeterministic choice and (arbitrary, i.e. not synchronized) merge. Language L_0 has moreover iteration (*), and language L_1 has recursion. We shall use \underline{A} , with typical elements \underline{a} , for the class of elementary (atomic) actions. In later refinements of the theory, actions \underline{a} will be replaced by assignment statements. Throughout the paper, we use a self-explanatory variant of BNF for syntactic definitions.

DEFINITION 3.1. The language L_0 (regular flow of control + merge) with elements S , is defined by

$$S ::= \underline{a} \mid \underline{\text{skip}} \mid S_1; S_2 \mid S_1 \cup S_2 \mid S_1 \parallel S_2 \mid S^*.$$

For the definition of the semantics of L_0 we use a domain of uniform processes P_0 . We assume that its constituent set A is a (possibly infinite) alphabet such that for each elementary action $\underline{a} \in \underline{A}$ there is a corresponding $a \in A$. Let, moreover, ϵ be the empty word (with respect to the alphabet A). We give

DEFINITION 3.2. The domain P_0 is given as solution of

$$P_0 = \{p_0\} \cup P_c((A \cup \{\epsilon\}) \times P_0).$$

Remark. Properly speaking, this requires adaptation of the definitions of section 2 for uniform processes with the convention that $a \in A \cup \{\epsilon\}$, together with natural definitions such as: $a_1 = a_2$ if a_1 and a_2 are both ϵ , or denote the same element of A .

We now define the semantics of L_0 by providing a mapping $M: L_0 \rightarrow P_0$. Thus, M determines for each language element S a corresponding process p . (Mappings such as M are often called *valuations* in denotational semantics. They serve to associate *meaning* - mathematical objects - to the syntactic constructs in a certain class (here L_0), and in this way embody the heart of a denotational semantics definition.)

DEFINITION 3.3. The valuation $M: L_0 \rightarrow P_0$ is defined by

- a. $M(\underline{a}) = \{\langle a, p_0 \rangle\}$, where a corresponds to \underline{a} , $M(\underline{\text{skip}}) = \{\langle \epsilon, p_0 \rangle\}$
- b. $M(S_1; S_2) = M(S_2) \circ M(S_1)$, $M(S_1 \cup S_2) = M(S_1) \cup M(S_2)$, $M(S_1 \parallel S_2) = M(S_1) \parallel M(S_2)$
- c. $M(S^*) = \lim_i p_i$, where $(p_0 = p_0$ and)

$$p_{i+1} = (p_i \circ M(S)) \cup \{\langle \epsilon, p_0 \rangle\}.$$

Remarks.

1. Since the elementary actions are left unspecified, there is not much we can do with them in the semantic definition. Therefore, we simply map them onto some corresponding elementary process.
2. The simplicity of clause b is a reward of our preparatory work in section 2. Operations upon (uniform) processes " \circ ", " \cup ", " \parallel " have become available, and they can be used directly to model the corresponding syntactic composition rules.
3. In order to understand the definition of S^* , recall the equivalence $S^* = S; S^* \cup \underline{\text{skip}}$. Now define a mapping $T: P_0 \rightarrow P_0$ by putting $T = \lambda p. ((p \circ M(S)) \cup \{\langle \epsilon, p_0 \rangle\})$. Here $\{\langle \epsilon, p_0 \rangle\}$ is the dummy process, i.e., the semantic equivalent of the syntactic skip action. It follows from general properties of the operations " \circ ", " \cup " (see Appendix B) that the mapping T is *contracting*, viz. that, for all p', p'' , $d(T(p'), T(p'')) \leq \frac{1}{2}d(p', p'')$ (this uses that $M(S) \neq P_0$ for all S). By a classical result in metric topology (the Banach fixed point theorem) we may then conclude that the sequence $p_0, T(p_0), T^2(p_0), \dots$ is a Cauchy sequence which converges to a limit p satisfying $p = T(p)$. (In fact, this limit is independent of the starting process p_0 , and yields the *unique* fixed point of T .)

Examples

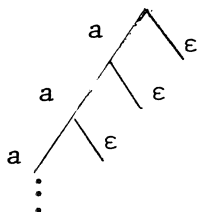
1. $M(\underline{a}_1; \underline{a}_2) = M(\underline{a}_2) \circ M(\underline{a}_1) = \{\langle a_2, p_0 \rangle\} \circ \{\langle a_1, p_0 \rangle\} = \{\langle a_1, \{\langle a_2, p_0 \rangle\} \rangle\}$.
2. $M((\underline{a}_1; \underline{a}_2) \parallel (\underline{a}_3; \underline{a}_4)) = \{\langle a_1, \{\langle a_2, p_0 \rangle\} \rangle\} \parallel \{\langle a_3, \{\langle a_4, p_0 \rangle\} \rangle\} = \dots =$
 $\{\langle a_1, \{\langle a_2, \{\langle a_3, \{\langle a_4, p_0 \rangle\} \rangle\} \rangle\} \rangle,$
 $\langle a_3, \{\langle a_2, \{\langle a_4, p_0 \rangle\} \rangle, \langle a_4, \{\langle a_2, p_0 \rangle\} \rangle \rangle \rangle,$
 $\langle a_3, \dots \rangle\}$

(Cf. the example after definition 2.14).

3. $M(\underline{a}^*) = p = \lim_i p_i$, where $p_{i+1} = (p_i \circ \{\langle a, p_0 \rangle\}) \cup \{\langle \epsilon, p_0 \rangle\}$.

Hence, $p = \{ \langle \epsilon, p_0 \rangle, \langle a, \{ \langle \epsilon, p_0 \rangle, \langle a, \{ \langle \epsilon, p_0 \rangle, \langle a, \dots \rangle \} \rangle \} \}$

In a picture, $M(\underline{a}^*)$ is described by



We observe that \underline{a}^* means executing \underline{a} zero or more times, including infinite repetition of \underline{a} .

We next turn to the recursive case. We shall employ the notation of the μ -calculus for recursion (see, e.g. [10,32]). For the reader who has not seen this before, the following explanation may help: Think of a parameterless recursive procedure Q in some Algol-like language. Q has a declaration of the form, say, $Q \Leftarrow \dots Q \dots Q \dots$, where $\dots Q \dots Q \dots$ is the procedure body with two recursive calls of Q . We note that the procedure variable Q is *bound* in this declaration (systematically renaming it would make no difference). A call of Q in the main program corresponds in the notation of the μ -calculus to the statement $\mu\xi[\dots\xi\dots\xi\dots]$, where the bound variable ξ is from some alphabet of procedure variables X . In this way, procedure declarations disappear, and inner calls are taken care of by the bound variable mechanism.

DEFINITION 3.4. Let X , with elements ξ , be the set of procedure variables. The language L_1 (general recursion with merge) is defined by: Let $S \in L_1$. Then

$$S ::= \underline{a} \mid \underline{\text{skip}} \mid S_1 ; S_2 \mid S_1 \cup S_2 \mid S_1 \parallel S_2 \mid \xi \mid \mu\xi[S].$$

For the semantics of L_1 we take process domain P_1 equal to P_0 . In order to handle the variables ξ , we introduce an environment E , with elements η , defined by $E = X \rightarrow P_1$, and we define the meaning of a statement $S \in L_1$ with respect to E . In other words, we take $M: L_1 \rightarrow (E \rightarrow P_1)$; its definition is given in

DEFINITION 3.5.

- a. $M(\underline{a})(\eta) = \{\langle a, p_0 \rangle\}$, $M(\underline{\text{skip}})(\eta) = \{\langle \epsilon, p_0 \rangle\}$
 b. $M(S_1; S_2)(\eta) = M(S_2)(\eta) \circ M(S_1)(\eta)$
 $M(S_1 \cup S_2)(\eta) = M(S_1)(\eta) \cup M(S_2)(\eta)$
 $M(S_1 \parallel S_2)(\eta) = M(S_1)(\eta) \parallel M(S_2)(\eta)$
 c. $M(\xi)(\eta) = \eta(\xi)$
 $M(\mu\xi[S])(\eta) = \lim_i p_i$, where $(p_0 = p_0 \text{ and })$
 $p_{i+1} = \{\langle \epsilon, M(S)(\eta\{p_i/\xi\}) \rangle\}.$

Remarks.

1. Clauses a and b are exactly as in definition 3.3, apart from the extra argument η which is just carried along.
2. In the definition of the meaning of the μ -construct we observe a complication. The reader who is familiar with the treatment of (sequential) recursive procedures in denotational semantics would probably have expected the definition $p_{i+1} = M(S)(\eta\{p_i/\xi\})$. (Note that this specializes to the previous treatment of iteration by taking $S^* \equiv \mu\xi[S; \xi \cup \underline{\text{skip}}]$.) This may work as well, but we have not been able to prove that, defining the mapping $T' = \lambda p. M(S)(\eta\{p/\xi\})$, the sequence $\langle T'^i(p_0) \rangle_i$ is a CS for arbitrary $S \in L_1$. (Bergstra & Klop [13] prove that $\langle T'^i(q) \rangle_i$ is a CS for each q . However, the resulting limit depends, in general, on q , and the problem remains which q to choose.) Therefore, we have introduced an extra step in defining $T = \lambda p. \{\langle \epsilon, M(S)(\eta\{p/\xi\}) \rangle\}$. This indeed ensures that T is contracting and, as before, $\lim_i T^i(p_0)$ exists and equals the unique fixed point of T . Operationally, the ϵ -step may be seen as reflecting the action of procedure entry. By way of example we obtain that $M(\mu\xi[\xi])(\eta) = \{\langle \epsilon, \{\langle \epsilon, \{\langle \epsilon, \dots \rangle\} \rangle\} \rangle\}$ (an infinite sequence of empty steps). C.f. also the discussion in [17].

In definitions 3.3 and 3.5 we have shown how to associate a process p with statements $S \in L_0$ or $S \in L_1$. In case one is interested only in the set of all possible sequences of elementary actions determined by executing S - rather than in its meaning $p = M(S)$ as a whole; note that a process contains more information than the set of its constituent paths - we apply a new (unary) operation upon process p , determining its *yield* p^+ . For this, we need the auxiliary definition of *path* of a process:

DEFINITION 3.6. Let $p \in P_0$, and let $a, a_i \in A \cup \{\epsilon\}$. A *path* for p is a (finite or infinite) sequence $(*) : \langle a_1, p_1 \rangle, \langle a_2, p_2 \rangle, \dots, \langle a_i, p_i \rangle, \dots$ such that

- (i) $\langle a_1, p_1 \rangle \in p$ and $\langle a_{i+1}, p_{i+1} \rangle \in p_i$, $i = 1, 2, \dots$,
- (ii) sequence $(*)$ is either infinite or, when finite, terminates with $\langle a_n, p_n \rangle$ ($n \geq 1$), with $p_n = p_0$.

Remark. Note that, by this definition, p_0 has no paths. Moreover, note that we do not allow a finite path terminating in $\langle a_n, p_n \rangle$ with $p_n = \emptyset$ (the empty set is also a process!)

Now let $A^\infty \stackrel{\text{df.}}{=} A^* \cup A^\omega$, i.e., A^∞ is the set of all finite (possibly empty) and infinite sequences of elements in A . Also, let $"."$ denote concatenation of words over A . We put

DEFINITION 3.7. $p^+ \subseteq A^\infty$ is defined to consist of all words $w \in A^\omega$ such that either $w = a_1 \cdot a_2 \cdot \dots \cdot a_n$, where $\langle a_1, p_1 \rangle, \langle a_2, p_2 \rangle, \dots, \langle a_n, p_n \rangle$ is a finite path for p , or $w = a_1 \cdot a_2 \cdot \dots \cdot a_i \cdot \dots$, where $\langle a_1, p_1 \rangle, \langle a_2, p_2 \rangle, \dots, \langle a_i, p_i \rangle, \dots$ is an infinite path for p .

Remark. Remember that $a_i \in A \cup \{\epsilon\}$. Thus, the a_i occurring in the above equations for w may disappear in the resulting concatenation in case $a_i = \epsilon$.

Examples. $p_0^+ = \emptyset$, $\{\epsilon, p_0\}^+ = \{\epsilon\}$, $\{\langle a_1, \{\epsilon, \{\langle a_2, p_0 \rangle\} \rangle\} \rangle\}^+ = \{a_1 \cdot \epsilon \cdot a_2\} = \{a_1 a_2\}$, $\{\langle a_1, \{\langle a_2, p_0 \rangle\} \rangle, \langle a_1, \{\langle a_3, p_0 \rangle\} \rangle\}^+ = \{\langle a_1, \{\langle a_2, p_0 \rangle, \langle a_3, p_0 \rangle\} \rangle\}^+ = \{a_1 a_2, a_1 a_3\}$.

From the last example we conclude that, for $p_1 \neq p_2$, we may have that $p_1^+ = p_2^+$. We may define $p_1 \sim p_2 \stackrel{\text{df.}}{\iff} p_1^+ = p_2^+$, and study properties of this equivalence relation. (A more refined equivalence relation is Milner's observation equivalence, cf.[44].)

Finally, one may use the yield operation in the semantics of languages such as L_0 or L_1 , by investigating the mapping M^+ defined by $M^+(S) = M(S)^+$. This mapping obtains the sequences of elementary actions prescribed by the execution of S . For example, $M^+((S_1; S_2) \cup (S_1; S_3)) = M^+(S_1; (S_2 \cup S_3))$, whereas M differs on these two arguments. For languages such as L_0, L_1 , consideration of the yield $M(S)^+$ is probably not very fruitful. Later (section 5) we shall encounter languages where the role of the yield operation is more important.

4. SYNCHRONIZATION

We add a synchronization construct to the language L_0 - leaving to the reader a similar extension of L_1 . This section owes much to the pioneering studies of Milner on the nature of synchronization [40,42,43,44].

We introduce the language L_2 as an extension of L_0 by adding a class of synchronization commands C, \bar{C} . Synchronization commands always appear in pairs such that \bar{C} corresponds to C (and C to \bar{C}). Before trying to explain their meaning, we first give the syntax for L_2 .

DEFINITION 4.1. The language L_2 , with elements S , is defined by

$$S ::= \underline{a} \mid \underline{\text{skip}} \mid C \mid \bar{C} \mid S_1; S_2 \mid S_1 \cup S_2 \mid S_1 \parallel S_2 \mid S^* \mid S \setminus C.$$

In order to obtain some understanding for the meaning of these synchronization commands, let us take $S' \equiv S_1; C; S_2$, $S'' \equiv S_3; \bar{C}; S_4$, and let us consider $(S' \parallel S'') \setminus C$. Its intended meaning is that the merge of S' and S'' is synchronized by the pair C, \bar{C} such that, instead of the full merge of $S_1; S_2$ with $S_3; S_4$, we only retain $(S_1 \parallel S_3); (S_2 \parallel S_4)$. The role of the *restriction* operation $S \setminus C$ may be phrased roughly as deleting from the execution of S all execution sequences which contain C and \bar{C} in a way where synchronization failed. In an example such as $S' \equiv \underline{a}_1; C; \underline{a}_2$, $S'' \equiv \underline{a}_3; \bar{C}; \underline{a}_4$, one such failing sequence is, e.g., $\underline{a}_1; C; \underline{a}_2; \underline{a}_3; \bar{C}; \underline{a}_4$.

For the definition of the semantics of L_2 we introduce the domain P_2 as given by the equation

$$(4.1) \quad P_2 = \{p_0\} \cup P_c((A \cup \Gamma \cup \{\epsilon\}) \times P_2)$$

Here, as before, for each $\underline{a} \in L_2$ there is a corresponding $a \in A$. Moreover, for $C, \bar{C} \in L_2$ there are corresponding elements $\gamma, \bar{\gamma}$ in Γ . An arbitrary element of the set $A \cup \Gamma \cup \{\epsilon\}$ will in the sequel be denoted by β .

Remark. Processes in P_2 are close to Milner's synchronization trees. An important difference, however, is our use of sets rather than multisets, for the collection of "successors" of the "nodes" in a process.

We now give

We conclude with a few words on the yield operation in this case. For $p \in P_2$, p^+ determines a set of (finite or infinite) paths over the alphabet $A \cup \Gamma$. In case $p^+ \subseteq A^\infty$, one might call p *proper*. E.g., for p such that $p = M(S \setminus C)$, where synchronization in S only uses C, \bar{C} , we expect that $p^+ \subseteq A^\infty$. This expresses that unsuccessful attempts at synchronization do not contribute to p^+ , since there is no contribution to p^+ from paths in p terminating in the empty process (cf. the remark following definition 3.6).

5. STATES AND ASSIGNMENT

Until now, our languages contained only elementary actions the meaning of which was left unspecified. We next introduce the notion of state, extend the syntax of our languages with assignment and tests, and discuss the corresponding extension for the processes used in their semantics. First we present some preliminary definitions, introducing simple expressions, tests, and their meanings with respect to some state.

DEFINITION 5.1.

- a. Let Var , with elements $\underline{x}, \underline{y}, \dots$ be the class of simple variables. Let V be some domain of values (\mathbb{Z} might be an example) and let $\Sigma \stackrel{\text{df.}}{=} Var \rightarrow V$. Let $W = \{tt, ff\}$ be the set of truth-values.
- b. Let $v \in V$, $\sigma \in \Sigma$, $\underline{x} \in Var$. We define the *variant* notation, turning state σ into a state $\sigma\{v/\underline{x}\}$, by putting

$$\sigma\{v/\underline{x}\}(\underline{y}) = \begin{cases} v, & \text{if } \underline{x} \equiv \underline{y} \\ \sigma(\underline{y}) & \text{if } \underline{x} \neq \underline{y}. \end{cases}$$

- c. We introduce the classes Exp , with elements s, t , of *expressions* and $Test$, with elements b , of *logical expressions*. We assume given valuations $V: Exp \rightarrow (\Sigma \rightarrow V)$ and $W: Test \rightarrow (\Sigma \rightarrow W)$.
(The precise nature of Exp and $Test$ does not concern us here; all we require is that their evaluation always terminates. In a specific instance, taking, e.g., \mathbb{Z} for V , one might think of expressions such as $x+(y*z)$, and tests such as $x > y+z$.)

We continue with the definition of the syntax of language L_3 . It extends L_0 with assignment and tests. Synchronization will reappear in section 6 (this postponement is only for reasons of presentation).

DEFINITION 5.2. The language L_3 , with elements S , is defined by

$$S ::= \underline{x} := s \mid \underline{\text{skip}} \mid b \mid S_1; S_2 \mid S_1 \cup S_2 \mid S_1 \parallel S_2 \mid S^* \mid \underline{x} := ?$$

Remarks.

1. The intuitive meaning of $\underline{x} := s$, $\underline{\text{skip}}$, $S_1; S_2$, $S_1 \cup S_2$, $S_1 \parallel S_2$, S^* should be clear.

2. A test statement b may succeed or fail, depending on whether the test b evaluates to tt or ff in the current state. More familiar constructions such as if b then S_1 else S_2 fi or while b do S od are expressed in L_3 by $(b;S_1) \cup (\neg b;S_2)$ or $(b;S)^* ; \neg b$, respectively.
3. $\underline{x} := ?$ is the *random assignment*, introduced not so much because it is our favorite language concept, but rather to illustrate that semantics using processes can deal with it without any technical problems (contrary to the situation in traditional denotational semantics, cf. [4,7,10,20]).

For the semantics of L_3 we introduce the class of processes P_3 . This involves an essential extension of the processes as considered upto now, in that a process $p(\neq p_0)$ is now a function depending on Σ .

DEFINITION 5.3. The class of process P_3 is defined as solution of the domain equation

$$(5.1) \quad P_3 \equiv \{p_0\} \cup (\Sigma \rightarrow P_c(\Sigma \times P_3)).$$

We observe that (5.1) is an equation for a domain of non-uniform processes of the type considered in section 2, equation (2.3). By the general theory as developed there, operations $p_1 \circ p_2$, $p_1 \cup p_2$, $p_1 \parallel p_2$ are again meaningful (the latter, for the moment, without the synchronization refinement).

We define the valuation $M : L_3 \rightarrow P_3$ in

DEFINITION 5.4. The semantics of L_3 is given by

- a. $M(\underline{x}:=s) = \lambda\sigma. \{ \langle \sigma \{V(s)(\sigma)/\underline{x}, p_0 \rangle \}, M(\underline{\text{skip}}) = \lambda\sigma. \{ \langle \sigma, p_0 \rangle \}$
- b. $M(b) = \lambda\sigma. \text{ if } W(b)(\sigma) \text{ then } \{ \langle \sigma, p_0 \rangle \} \text{ else } \emptyset \text{ fi}$
- c. $M(S_1;S_2) = M(S_2) \circ M(S_1)$, $M(S_1 \cup S_2) = M(S_1) \cup M(S_2)$, $M(S_1 \parallel S_2) = M(S_1) \parallel M(S_2)$.
- d. $M(S^*) = \lim_i p_i$, where $(p_0 = p_0 \text{ and})$
 $p_{i+1} = (p_i \circ M(S)) \cup \lambda\sigma. \{ \langle \sigma, p_0 \rangle \}$
- e. $M(\underline{x}:=?) = \lambda\sigma. \{ \langle \sigma \{v/\underline{x}\}, p_0 \rangle \mid v \in V \}$.

Remarks.

1. Note how the dummy process, previously represented by $\{ \langle \epsilon, p_0 \rangle \}$, is now replaced by $\lambda\sigma. \{ \langle \sigma, p_0 \rangle \}$.

2. Note that, in clause e, the set $X = \{\langle \sigma\{v/\underline{x}\}, p_0 \rangle \mid v \in V\}$ is a subset of $P_{3,1} \stackrel{\text{df.}}{=} \{p_0\} \cup (\Sigma \rightarrow P(\Sigma \times \{p_0\}))$; that X is closed requires no more argument than the observation that *all* subsets of each of $P_{3,k}$ (where $P_{3,0} = \{p_0\}$, $P_{3,k+1} = \{p_0\} \cup (\Sigma \rightarrow P_c(\Sigma \times P_{3,k}))$, for $k \geq 0$) are (trivially) closed: distances between points are at least $1/2^{k-1}$ (for $k \geq 1$), and no nontrivial CS exists in $P_{3,k}$. Thus, we see how unbounded nondeterminacy fits smoothly into our theory.

Examples.

1. $M(\underline{x}:=0; \underline{y}:=\underline{x}+1) = \lambda\sigma. \{\langle \sigma\{0/\underline{x}\}, \lambda\bar{\sigma}. \{\langle \bar{\sigma}\{V(\underline{x}+1)(\bar{\sigma})/\underline{y}\}, p_0 \rangle\} \rangle\}$
2. $M((\underline{x}:=0; \underline{y}:=\underline{x}+1) \parallel \underline{x}:=1) =$
 $\lambda\sigma. \{\langle \sigma\{0/\underline{x}\}, \lambda\bar{\sigma}. \{\langle \bar{\sigma}\{V(\underline{x}+1)(\bar{\sigma})/\underline{y}\}, p_0 \rangle\} \rangle\} \parallel \lambda\bar{\sigma}. \{\langle \bar{\sigma}\{1/\underline{x}\}, p_0 \rangle\} = \dots =$
 $\lambda\sigma. \{\langle \sigma\{1/\underline{x}\}, \lambda\bar{\sigma}. \{\langle \bar{\sigma}\{0/\underline{x}\}, \lambda\bar{\sigma}. \{\langle \bar{\sigma}\{V(\underline{x}+1)(\bar{\sigma})/\underline{y}\}, p_0 \rangle\} \rangle\} \rangle,$
 $\langle \sigma\{0/\underline{x}\}, \lambda\bar{\sigma}. \{\langle \bar{\sigma}\{1/\underline{x}\}, \lambda\bar{\sigma}. \{\langle \bar{\sigma}\{V(\underline{x}+1)(\bar{\sigma})/\underline{y}\}, p_0 \rangle\} \rangle\} \rangle,$
 $\langle \sigma\{0/\underline{x}\}, \lambda\bar{\sigma}. \{\langle \bar{\sigma}\{V(\underline{x}+1)(\bar{\sigma})/\underline{y}\}, \lambda\bar{\sigma}. \{\langle \bar{\sigma}\{1/\underline{x}\}, p_0 \rangle\} \rangle\} \rangle\}.$
3. $M((\underline{x}=\underline{y}); \underline{z}:=1 \cup (\underline{x} \neq \underline{y}); \underline{z}:=2) =$
 $\lambda\sigma. (\text{if } \sigma(\underline{x}) = \sigma(\underline{y}) \text{ then } \{\langle \sigma, \lambda\bar{\sigma}. \{\langle \bar{\sigma}\{1/\underline{z}\}, p_0 \rangle\} \rangle \text{ else } \emptyset \text{ fi} \cup$
 $\text{if } \sigma(\underline{x}) \neq \sigma(\underline{y}) \text{ then } \{\langle \sigma, \lambda\bar{\sigma}. \{\langle \bar{\sigma}\{2/\underline{z}\}, p_0 \rangle\} \rangle \text{ else } \emptyset \text{ fi})$

Contrary to the situation in the previous sections, it is now of some importance to study the notion of yield for $p \in P_3$. We need the following definitions:

DEFINITION 5.5 (*paths for $\langle \sigma, p \rangle$*).

A (finite or infinite) sequence $\langle \sigma_1, p_1 \rangle, \langle \sigma_2, p_2 \rangle, \dots$, is a path for $\langle \sigma, p \rangle$ whenever

- (i) $\langle \sigma_1, p_1 \rangle = \langle \sigma, p \rangle$, and $\langle \sigma_{i+1}, p_{i+1} \rangle \in p_i(\sigma_i)$, $i = 1, 2, \dots$
- (ii) the sequence is either infinite or, when finite, terminates with $\langle \sigma_n, p_n \rangle$, $n \geq 1$, such that $p_n = p_0$.

The yield of a non-uniform process p may intuitively be understood as follows: Supply p with an argument σ . The pair $\langle \sigma, p \rangle$ determines the set of all paths for $\langle \sigma, p \rangle$. Terminating paths have leaves $\bar{\sigma}$ which are included in the output set, nonterminating paths are reflected by the appearance of \perp in the output set. Here \perp is the undefined state corresponding to nonterminating computations. Its role is fundamental in traditional denotational semantics, but rather less so in our theory.

DEFINITION 5.6. For $p \in P_3$ we define $p^+ : \Sigma \rightarrow P(\Sigma \cup \{\perp\})$ by putting $p_0^+ = \lambda\sigma.\emptyset$, and, for $p \neq p_0$, $p^+ = \lambda\sigma.(\langle\sigma, p\rangle)^+$, where $\langle\sigma, p\rangle^+$ is given by

$$\langle\sigma, p\rangle^+ = \{\bar{\sigma} \mid \text{there exists a path for } \langle\sigma, p\rangle \text{ terminating with } \langle\bar{\sigma}, p_0\rangle\} \\ \cup \{ \text{if } \langle\sigma, p\rangle \text{ has infinite paths then } \{\perp\} \text{ else } \emptyset \text{ fi} \}.$$

Example. Consider the processes $p_1 = M(\underline{x}:=0; \underline{y}:=\underline{x}+1) \parallel \underline{x}:=1$ discussed in the example following definition 5.4. First consider p_1 . The pair $\langle\sigma, p\rangle$ has as (only) path the sequence $\langle\sigma, p\rangle, \langle\sigma\{0/\underline{x}\}, \lambda\bar{\sigma}.\{\bar{\sigma}\{V(\underline{x}+1)(\bar{\sigma})/\underline{y}\}, p_0\rangle\rangle, \langle\sigma\{0/\underline{x}\}\{V(\underline{x}+1)(\sigma\{0/\underline{x}\})/\underline{y}\}, p_0\rangle$, and we see that $p_1^+ = \lambda\sigma.\{\sigma\{0/\underline{x}\}\{1/\underline{y}\}\}$. For p_2 we obtain in a similar fashion $p_2^+ = \lambda\sigma.\{\sigma\{0/\underline{x}\}\{1/\underline{y}\}, \sigma\{1/\underline{x}\}\{2/\underline{y}\}, \sigma\{1/\underline{y}\}\{1/\underline{x}\}\}$.

We now consider what happens when we extend L_3 with recursion. We only supply the pertinent definitions which should be sufficient for the reader who has understood L_1 :

DEFINITION 5.7 (recursion).

a. (syntax) Let $S \in L_4$. We define (omitting $\underline{x} := ?$ for simplicity):

$$S ::= \underline{x} := s \mid b \mid S_1; S_2 \mid S_1 \cup S_2 \mid S_1 \parallel S_2 \mid \xi \mid \mu\xi[S].$$

b. Let $P_4 \stackrel{\text{df.}}{=} P_3$, and let $E = X \rightarrow P_4$, with $\eta \in E$. We define

$M: L_4 \rightarrow (E \rightarrow P_4)$ by

$$M(\underline{x}:=s)(\eta) = \lambda\sigma.\{\sigma\{V(s)(\sigma)/\underline{x}\}, p_0\},$$

$$M(b)(\eta) = \lambda\sigma. \text{if } W(b)(\sigma) \text{ then } \{\sigma, p_0\} \text{ else } \emptyset \text{ fi}$$

$$M(S_1; S_2)(\eta) = M(S_2)(\eta) \circ M(S_1)(\eta), \dots,$$

$$M(\xi)(\eta) = \eta(\xi),$$

$$M(\mu\xi[S])(\eta) = \lim_i p_i, \text{ where } (p_0 = p_0 \text{ and})$$

$$p_{i+1} = \lambda\sigma.\{\sigma, M(S)(\eta\{p_i/\xi\})\}.$$

Thus, apart from the use of $\lambda\sigma.\{\sigma, \dots\}$ instead of $\{\epsilon, \dots\}$ - as we saw in definition 3.5 - the definitions are a straightforward continuation of the preceding theory.

6. STATES, ASSIGNMENT AND SYNCHRONIZATION

We now extend the language L_3 introduced in the previous section with synchronization commands. We proceed in two stages: Firstly, we add to L_3 commands C, \bar{C} as considered previously in section 4. Secondly, we further extend L_3 with *guarded commands* and, in particular, with guards establishing synchronization. (For simplicity, we return to L_3 rather than extending L_4 .)

DEFINITION 6.1 (L_3 with synchronization). The language L_5 with elements S , is defined by

$$S ::= \underline{x} := s \mid \underline{\text{skip}} \mid b \mid S_1; S_2 \mid S_1 \cup S_2 \mid S_1 \parallel S_2 \mid S^* \mid C \mid \bar{C} \mid S \setminus_1 C \mid S \setminus_2 C \mid \Delta.$$

We observe two restriction operations \setminus_1 and \setminus_2 . The former is the direct counterpart of the \setminus - operation in the uniform case (section 4); the latter is aimed at modelling *deadlock*. In our interpretation, this occurs in a situation where a failing attempt at synchronization has no alternative. This phenomenon is then signalled by the appearance of the *dead* process in the result. The statement Δ is the *abort* statement. We assume from now on that Σ contains the special dead state δ .

Next, we introduce the process domain P_5 :

DEFINITION 6.2. Process domain P_5 satisfies the equation

$$P_5 = \{p_0\} \cup (\Sigma \rightarrow P_c((\Sigma \cup \Gamma) \times P_5))$$

We define the semantics of L_5 in

DEFINITION 6.3. The valuation $M: L_5 \rightarrow P_5$ is given by

$$\begin{aligned} M(\underline{x} := s) &= \lambda \sigma. \underline{\text{if}} \sigma = \delta \underline{\text{then}} \{ \langle \delta, p_0 \rangle \} \underline{\text{else}} \{ \langle \sigma \{ V(s)(\sigma) / \underline{x} \rangle, p_0 \rangle \} \underline{\text{fi}} \\ M(\underline{\text{skip}}) &= \lambda \sigma. \{ \langle \sigma, p_0 \rangle \} \\ M(b) &= \lambda \sigma. \underline{\text{if}} \sigma = \delta \underline{\text{then}} \{ \langle \delta, p_0 \rangle \} \underline{\text{else if}} W(b)(\sigma) \underline{\text{then}} \{ \langle \sigma, p_0 \rangle \} \underline{\text{else}} \emptyset \underline{\text{fi fi}} \\ M(S_1; S_2) &= M(S_2) \circ M(S_1), M(S_1 \cup S_2) = M(S_1) \cup M(S_2), \\ M(S_1 \parallel S_2) &= M(S_1) \parallel M(S_2), \text{ with "||" defined below} \\ M(S^*) &= \lim_i p_i, \text{ with } (p_0 = p_0 \text{ and}) \\ p_{i+1} &= (p_i \circ M(S)) \cup \lambda \sigma. \{ \langle \sigma, p_0 \rangle \} \\ M(C) &= \lambda \sigma. \underline{\text{if}} \sigma = \delta \underline{\text{then}} \{ \langle \delta, p_0 \rangle \} \underline{\text{else}} \{ \langle \gamma, p_0 \rangle \} \underline{\text{fi}}, \text{ and similarly for } M(\bar{C}) \end{aligned}$$

$$M(S \setminus_i C) = M(S) \setminus_i \gamma, \text{ with } \setminus_i \text{ to be defined below, } i = 1, 2$$

$$M(\Delta) = \lambda\sigma.\{\langle\delta, p_0\rangle\}$$

The definitions of " \parallel ", " \setminus_i " are given in

DEFINITION 6.4. Let β range over $\Sigma \cup \Gamma$. We only give the definitions for p, q of finite nonzero degree:

- a. $(\lambda\sigma.X) \parallel (\lambda\sigma.Y) =$
 $\lambda\sigma.(\{x \parallel \lambda\sigma.Y \mid x \in X\} \cup \{\lambda\sigma.Y \parallel y \mid y \in Y\} \cup \{\langle\sigma, p' \parallel q'\rangle \mid \langle\gamma, p'\rangle \in X, \langle\bar{\gamma}, q'\rangle \in Y\});$
 here $\langle\beta, p'\rangle \parallel \lambda\sigma.Y = \langle\beta, p' \parallel \lambda\sigma.Y\rangle$, $\lambda\sigma.X \parallel \langle\beta, q'\rangle = \langle\beta, \lambda\sigma.X \parallel q'\rangle$
- b. $p \setminus_1 \gamma = \lambda\sigma.\{\langle\beta, p' \setminus_1 \gamma\rangle \mid \langle\beta, p'\rangle \in p(\sigma), \beta \neq \gamma, \bar{\gamma}\}$
 $p \setminus_2 \gamma = \lambda\sigma.\{\langle\beta, p' \setminus_2 \gamma\rangle \mid \langle\beta, p'\rangle \in p(\sigma), \beta \neq \gamma, \bar{\gamma}\} \stackrel{\text{df.}}{=} X$
 $\text{u}(\text{if } (p(\sigma) \neq \emptyset) \wedge (X = \emptyset) \text{ then } \{\langle\delta, p_0\rangle\} \text{ else } \emptyset \text{ fi})$

We see that in $S \setminus_1 C$, failed attempts at synchronization through C, \bar{C} are not signalled (pairs $\langle\gamma, p'\rangle, \langle\bar{\gamma}, p''\rangle$ are simply deleted), whereas in $S \setminus_2 C$ the failed attempts at synchronization are signalled when they are without alternatives (i.e. in case the set X , obtained from $p(\sigma)$ by deleting pairs $\langle\gamma, p'\rangle, \langle\bar{\gamma}, p''\rangle$, equals \emptyset).

Examples

1. We determine $M(S)$, where $S \equiv (\underline{x}:=0; C; \underline{x}:=1) \parallel (\underline{y}:=2; \bar{C}; \underline{y}:=3) \setminus_2 C$.
 Let $M(\underline{x}:=1 \parallel \underline{y}:=3) \stackrel{\text{df.}}{=} p$. Then $M(S) =$ (omitting dead states for simplicity)
 $\lambda\sigma.\{\langle\sigma\{0/\underline{x}\}, \lambda\bar{\sigma}.\{\langle\bar{\sigma}\{2/\underline{y}\}, \lambda\bar{\sigma}.\{\langle\bar{\sigma}, p\rangle\}\rangle\},$
 $\langle\sigma\{2/\underline{y}\}, \lambda\bar{\sigma}.\{\langle\bar{\sigma}\{0/\underline{x}\}, \lambda\bar{\sigma}.\{\langle\bar{\sigma}, p\rangle\}\rangle\}\rangle\}$

Here the $\lambda\bar{\sigma}.\{\langle\bar{\sigma}, \dots\rangle\}$ terms result from the synchronization of the $\langle\gamma, p'\rangle, \langle\bar{\gamma}, p''\rangle$ terms; also, all pairs $\langle\gamma, \dots\rangle, \langle\bar{\gamma}, \dots\rangle$ are deleted by the restriction operation (no dead states are introduced; \setminus_1 and \setminus_2 are indistinguishable in this example). Cf. also the example after definition 4.3.

2. Let $p_1 \stackrel{\text{df.}}{=} \lambda\sigma.\{\langle\sigma_1, \lambda\bar{\sigma}.\{\langle\gamma, p'_2\rangle, \langle\sigma_3, p_0\rangle\}\rangle\},$
 $p_2 \stackrel{\text{df.}}{=} \lambda\sigma.\{\langle\sigma_1, \lambda\bar{\sigma}.\{\langle\gamma, p'_2\rangle\}, \langle\sigma_3, p_0\rangle\}\}$
 Then $p_1 \setminus_1 \gamma = p_1 \setminus_2 \gamma = \lambda\sigma.\{\langle\sigma_1, \lambda\bar{\sigma}.\{\langle\sigma_3, p_0\rangle\}\rangle\}$
 $p_2 \setminus_1 \gamma = \lambda\sigma.\{\langle\sigma_2, \lambda\bar{\sigma}.\emptyset, \langle\sigma_3, p_0\rangle\}\},$
 $p_2 \setminus_2 \gamma = \lambda\sigma.\{\langle\sigma_1, \lambda\bar{\sigma}.\{\langle\delta, p_0\rangle\}\rangle, \langle\sigma_3, p_0\rangle\}$

We see that in process p_2 its subprocess $\lambda\bar{\sigma}.\{\langle\gamma, p'_2\rangle\}$ has no alternatives for synchronization through γ ; hence, deadlock is signalled as the result of restriction.

3. Consider the program $S \equiv ((C \cup (\underline{x}:=1)) \parallel \bar{C}) \setminus_2 C$

Let $\sigma_1 = \sigma\{1/\underline{x}\}$. We obtain for $M(S)$ - again ignoring dead states:

$$\begin{aligned} & (\lambda\sigma.\{\langle\gamma, p_0\rangle, \langle\sigma_1, p_0\rangle\} \parallel \lambda\bar{\sigma}.\{\langle\bar{\gamma}, p_0\rangle\}) \setminus_2 \gamma = \\ & \lambda\sigma.\{\langle\sigma, p_0\rangle, \langle\gamma, \lambda\bar{\sigma}.\{\langle\bar{\gamma}, p_0\rangle\}\rangle, \langle\sigma_1, \lambda\bar{\sigma}.\{\langle\bar{\gamma}, p_0\rangle\}\rangle, \\ & \quad \langle\bar{\gamma}, \lambda\bar{\sigma}.\{\langle\gamma, p_0\rangle, \langle\bar{\sigma}_1, p_0\rangle\}\rangle\} \setminus_2 \gamma = \\ & \lambda\sigma.\{\langle\sigma, p_0\rangle, \langle\sigma_1, \lambda\bar{\sigma}.\{\langle\delta, p_0\rangle\}\rangle\} \end{aligned}$$

We see that S amounts to either the skip statement, or setting \underline{x} to 1 after which deadlock occurs.

We conclude this part with a few words on p^+ for $p \in P_5$. Let, as usual, β range over $\Sigma \cup \Gamma$. We say that a (finite or infinite) sequence

(*) : $\langle\beta_1, p_1\rangle, \langle\beta_2, p_2\rangle, \dots$ is a *path* for $\langle\sigma, p\rangle$ whenever

- (i) $\langle\beta_1, p_1\rangle = \langle\sigma, p\rangle, \beta_i \in \Sigma$ and $\langle\beta_{i+1}, p_{i+1}\rangle \in p_i(\beta_i)$, $i = 1, 2, \dots$, and
- (ii) the sequence (*) is either infinite or, when finite, terminates in

$$\langle\beta_n, p_n\rangle \text{ with } p_n = p_0 \text{ or } \beta_n \in \Gamma.$$

We now define $p^+ : \Sigma \rightarrow \mathcal{P}(\Sigma \cup \Gamma \cup \{\perp\})$ by putting $p_0^+ = \lambda\sigma.\emptyset$ and, for $p \neq p_0$, $p^+ = \lambda\sigma.(\langle\sigma, p\rangle^+)$, where $\langle\sigma, p\rangle^+$ is given by

$$\begin{aligned} \{\beta_n \mid & \text{there exists some terminating path for } \langle\sigma, p\rangle \text{ with } \langle\beta_n, p_n\rangle \\ & = \langle\beta_n, p_0\rangle \text{ or } \langle\beta_n, p_n\rangle = \langle\gamma, p_n\rangle\} \end{aligned}$$

$$\cup (\text{if } \langle\sigma, p\rangle \text{ has an infinite path then } \{\perp\} \text{ else } \emptyset \text{ fi})$$

Note that the definition of p^+ assumes the possibility of γ -leaves in the process tree. Normally, this will only occur as the result of some error, since suitable use of the \setminus_i operations will have deleted all occurrences of any γ from processes p obtained as meaning $M(S)$ of some $S \in L_5$.

We now turn to the consideration of guarded commands and, in particular, of synchronization through guards. We introduce L_6 in

DEFINITION 6.5 (L_5 with guarded commands). The language L_6 is defined by

$$\begin{aligned} S := \underline{x} := s \mid \dots \mid \Delta \mid & \quad (\dots \text{ as in definition 6.1}) \\ \text{if } b_1 \rightarrow S_1 \square \dots \square b_n \rightarrow S_n \text{ fi} \mid & \text{do } b_1 \rightarrow S_1 \square \dots \square b_n \rightarrow S_n \text{ od} \mid \\ \text{if } b_1; C_1 \rightarrow S_1 \square \dots \square b_n; C_n \rightarrow S_n \text{ fi} \mid & \\ \text{do } b_1; C_1 \rightarrow S_1 \square \dots \square b_n; C_n \rightarrow S_n \text{ od} & \end{aligned}$$

The constructs if ... fi and do ... od with simple tests as guards are as in Dijkstra [20]; the constructs $b_i; C_i \rightarrow S_i$ (synchronization through guards) are a simple case of Hoare's CSP (see next section). The meaning of the first two constructs is easy to define: We take $P_6 = P_5$, and define $M: L_6 \rightarrow P_6$ by (omitting the clauses which are as in definition 6.3):

DEFINITION 6.6.

- a. $M(\underline{\text{if}}\ b_1 \rightarrow S_1 \ \square \dots \square b_n \rightarrow S_n \ \underline{\text{fi}}) =$
 $M(b_1; S_1 \cup \dots \cup b_n; S_n \cup \neg b_1 \wedge \dots \wedge \neg b_n; \Delta)$
- b. $M(\underline{\text{do}}\ b_1 \rightarrow S_1 \ \square \dots \square b_n \rightarrow S_n \ \underline{\text{od}}) =$
 $M((b_1; S_1 \cup \dots \cup b_n; S_n)^* ; (\neg b_1 \wedge \dots \wedge \neg b_n))$

Remarks

1. Note how, for the if ... fi command, if all guards fail Δ is executed; abortion is thus modelled - just as deadlock - by delivering the dead state
2. Definition 6.6b expresses that do ... od is equivalent to while $b_1 \vee \dots \vee b_n$ do $b_1; S_1 \cup \dots \cup b_n; S_n$ od
3. For a remark on a possible different interpretation of $b_i \rightarrow S_i$ in guarded commands see remark 8.2.

The definition of the other two cases is more involved:

DEFINITION 6.7.

$$M(\underline{\text{if}}\ b_1; C_1 \rightarrow S_1 \ \square \dots \square b_n; C_n \rightarrow S_n \ \underline{\text{fi}}) =$$

$$\lambda\sigma. \underline{\text{if}}\ \sigma = \delta \ \underline{\text{then}}\ \{\langle \delta, p_0 \rangle\} \ \underline{\text{else}}$$

$$(\underline{\text{if}}\ W(b_1)(\sigma) \ \underline{\text{then}}\ \{\langle \gamma_1, M(S_1) \rangle\} \ \underline{\text{else}}\ \emptyset \ \underline{\text{fi}} \cup \dots \cup$$

$$\underline{\text{if}}\ W(b_n)(\sigma) \ \underline{\text{then}}\ \{\langle \gamma_n, M(S_n) \rangle\} \ \underline{\text{else}}\ \emptyset \ \underline{\text{fi}} \cup$$

$$\underline{\text{if}}\ W(\neg b_1 \wedge \dots \wedge \neg b_n)(\sigma) \ \underline{\text{then}}\ \{\langle \delta, p_0 \rangle\} \ \underline{\text{else}}\ \emptyset \ \underline{\text{fi}}) \ \underline{\text{fi}}$$

Definition 6.7 is perhaps best understood by discussing an example. We use a slight variation on the official syntax, by allowing an if ... fi construct with both $b;C$ and b -type of guards. Also, the guard true ; C is abbreviated to C . Let

$$S_1 \equiv (\underline{\text{if}}\ C \rightarrow \text{skip} \ \square \ \text{true} \rightarrow x:=1 \ \underline{\text{fi}} \parallel x:=2) \setminus_2 C$$

$$S_2 \equiv (\underline{\text{if}}\ \text{true} \rightarrow C \ \square \ \text{true} \rightarrow x:=1 \ \underline{\text{fi}} \parallel x:=2) \setminus_2 C$$

We show that the deadlock behaviour of these two cases differs. In fact, putting $\sigma_1 = \sigma\{1/x\}$, $\sigma_2 = \sigma\{2/x\}$, $p_e = \lambda\sigma. \{\langle \sigma, p_0 \rangle\}$, $p_1 = \lambda\sigma. \{\langle \sigma_1, p_0 \rangle\}$, $p_2 = \lambda\sigma. \{\langle \sigma_2, p_0 \rangle\}$ (and ignoring the case $\sigma = \delta$ for simplicity), we obtain

$$M(S_1) = (\lambda\sigma. \{\langle \gamma, p_e \rangle, \langle \sigma, p_1 \rangle\} \parallel p_2) \setminus_2 \gamma$$

$$M(S_2) = (\lambda\sigma. \{\langle \sigma, \lambda\bar{\sigma}. \{\langle \gamma, p_0 \rangle\} \rangle, \langle \sigma, p_1 \rangle\} \parallel p_2) \setminus_2 \gamma$$

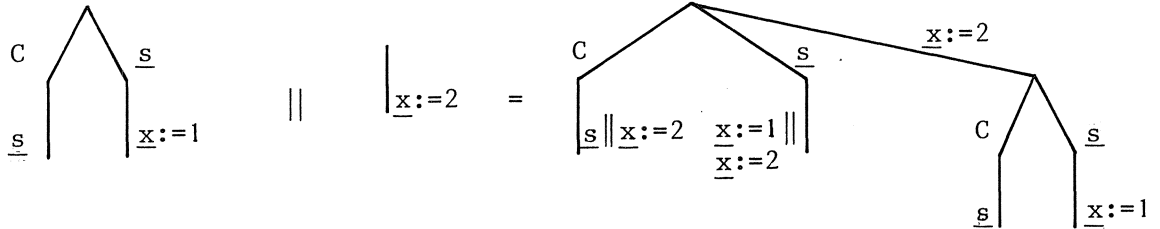
Hence,

$$\begin{aligned}
 M(S_1) &= \lambda\sigma.\{\langle\gamma, p_\epsilon \parallel p_2\rangle, \langle\sigma, p_1 \parallel p_2\rangle, \\
 &\quad \langle\sigma_2, \lambda\bar{\sigma}.\{\langle\gamma, p_\epsilon\rangle, \langle\bar{\sigma}, p_1\rangle\}\rangle\}_2 \gamma \\
 M(S_2) &= \lambda\sigma.\{\langle\sigma, \lambda\bar{\sigma}.\{\langle\gamma, p_0\rangle\} \parallel p_2\rangle, \langle\sigma, p_1 \parallel p_2\rangle, \\
 &\quad \langle\sigma_2, \lambda\bar{\sigma}.\{\langle\bar{\sigma}, \lambda\bar{\sigma}.\{\langle\gamma, p_0\rangle\}\rangle, \langle\bar{\sigma}, p_1\rangle\}\rangle\}_2 \gamma \\
 &= \lambda\sigma.\{\langle\sigma, \lambda\bar{\sigma}.\{\langle\gamma, p_2\rangle, \langle\sigma_2, \lambda\bar{\sigma}.\{\langle\gamma, p_0\rangle\}\rangle\}\rangle, \langle\sigma, p_1 \parallel p_2\rangle, \\
 &\quad \langle\sigma_2, \lambda\bar{\sigma}.\{\langle\bar{\sigma}, \lambda\bar{\sigma}.\{\langle\gamma, p_0\rangle\}\rangle, \langle\bar{\sigma}, p_1\rangle\}\rangle\}_2 \gamma
 \end{aligned}$$

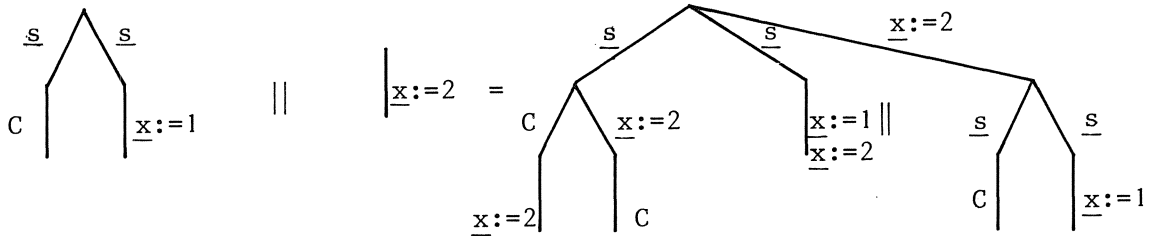
Thus,

$$\begin{aligned}
 M(S_1) &= \lambda\sigma.\{\langle\sigma, p_1 \parallel p_2\rangle, \langle\sigma_2, \lambda\bar{\sigma}.\{\langle\bar{\sigma}, p_1\rangle\}\rangle\} \\
 &\quad (M(S_1) \text{ shows no deadlock}) \\
 M(S_2) &= \lambda\sigma.\{\langle\sigma, \lambda\bar{\sigma}.\{\langle\sigma_2, \lambda\bar{\sigma}.\{\langle\delta, p_0\rangle\}\rangle\}\rangle, \langle\sigma, p_1 \parallel p_2\rangle, \\
 &\quad \langle\sigma_2, \lambda\bar{\sigma}.\{\langle\bar{\sigma}, \lambda\bar{\sigma}.\{\langle\delta, p_0\rangle\}\rangle, \langle\bar{\sigma}, p_1\rangle\}\rangle\} \\
 &\quad (M(S_2) \text{ has two possibilities of deadlock})
 \end{aligned}$$

Some pictures may clarify the situation. Let s be short for skip (or, equivalently, for true). S_1 may be pictured as



and S_2 as



In the first resulting picture, the two branches labelled by C both have an alternative. In the second resulting picture, there are two C - branches without alternative which are turned into dead branches by $\setminus_2 C$.

We conclude this section with the definition of the semantics of the construct $(*)$: $\underline{\text{do}}\ b_1 ; C_1 \rightarrow S_1 \square \dots \square b_n ; C_n \rightarrow S_n \underline{\text{od}}$. Defining the meaning of $(*)$ turns out to be fairly involved - at least, we have not been able to come up with a simpler treatment. The problem we have is best explained by comparing statements $\underline{\text{do}}\ b \rightarrow S \underline{\text{od}}$ and $\underline{\text{do}}\ C \rightarrow S \underline{\text{od}}$. For the former we have the equivalent construct $(b;S)^* ; \neg b$ - iterate $b;S$ as long as b is true - and for the latter we would like to be able to write, by analogy, something like $(C;S)^* ; \neg C$. This is not well-defined in L_6 . However, it suggests the following approach for dealing with $(*)$: Introduce, besides synchronization elements $\gamma, \bar{\gamma} \in \Gamma$ also elements $\neg\gamma, \neg\bar{\gamma}$ in a set $\neg\Gamma$. The function of $\neg\gamma$ or $\neg\bar{\gamma}$ is, roughly, to express commitment *not* to use the possibility of a $\gamma, \bar{\gamma}$ synchronization - and, instead, deliver the equivalent of a *skip* -statement. We (once more) redefine " \parallel ". The essence of the new definition consists in

- (i) $\langle \neg\gamma, \dots \rangle$ encountering some $\langle \bar{\gamma}, \dots \rangle$ gives no contribution to the result,
- and (ii) remaining occurrences of $\neg\gamma$ in the result are turned into skip steps by the restriction operation \setminus_2 . By way of example we consider the merge of the following two sets (returning for a moment to the uniform case for easier notation):

$$(6.1) \quad \{ \langle \gamma, p_1 \rangle, \langle \neg\gamma, p_0 \rangle \} \parallel \{ \langle \bar{\gamma}, p_2 \rangle, \langle \beta, p_0 \rangle \} \setminus_2 \gamma$$

We want the outcome of (6.1) to consist of the following parts:

- (i) $\langle \gamma, \dots \rangle$ and $\langle \bar{\gamma}, \dots \rangle$, to be deleted by $\setminus_2 \gamma$
- (ii) $\langle \epsilon, (p_1 \parallel p_2) \setminus_2 \gamma \rangle$, achieved as a result of successful synchronization between $\langle \gamma, p_1 \rangle$ and $\langle \bar{\gamma}, p_2 \rangle$
- (iii) $\langle \neg\gamma, \{ \langle \beta, p_0 \rangle \}, \langle \beta, \{ \langle \gamma, p_1 \rangle, \langle \neg\gamma, p_0 \rangle \} \rangle$ as intermediate result, turned by the redefined \setminus_2 into $\langle \epsilon, \{ \langle \beta, p_0 \rangle \}, \langle \beta, \{ \langle \epsilon, p_0 \rangle \} \rangle$
- (iv) no pairs as result of the merge of $\langle \neg\gamma, p_0 \rangle$ with $\langle \bar{\gamma}, p_2 \rangle$

Formally, the various parts of the definition are collected in

DEFINITION 6.8.

- a. $P'_6 = \{p_0\} \cup (\Sigma \rightarrow P_c((\Sigma \cup \Gamma \cup \neg\Gamma) \times P'_6))$
- b. For $p, q \in P'_6$ we defined $p \parallel q$ (for $p = \lambda\sigma.X, q = \lambda\sigma.Y$ of finite non-zero degree) as follows: Let β range over $\Sigma \cup \Gamma \cup \neg\Gamma$.

$$(\lambda\sigma.X) \parallel (\lambda\sigma.Y) =$$

$$\lambda\sigma.(\{(x \parallel \lambda\sigma.Y) \mid x \in X\} \cup \{(\lambda\sigma.X \parallel y) \mid y \in Y\} \cup \{\langle\sigma, p' \parallel p''\rangle \mid \\ \langle\gamma, p'\rangle \in X, \langle\bar{\gamma}, p''\rangle \in Y\})$$

$$\text{where } \langle\beta, p\rangle \parallel \lambda\sigma.Y = \langle\beta, p \parallel \lambda\sigma.Y'\rangle$$

$$\text{where } Y' = \begin{cases} Y, & \text{if } \beta \notin \neg\Gamma \\ Y \setminus \{\langle\beta', p'\rangle \mid \beta = \neg\gamma \text{ and } \beta' = \bar{\gamma} \text{ for some } \gamma \in \Gamma\} & \text{if } \beta \in \neg\Gamma \end{cases}$$

$$\text{and similarly for } (\lambda\sigma.X) \parallel \langle\beta, p\rangle$$

$$\begin{aligned} \text{c. } p \setminus_2 \gamma &= \lambda\sigma. \{ \langle\beta', p' \setminus_2 \gamma\rangle \mid \langle\beta', p'\rangle \in p(\sigma), \beta \neq \gamma, \bar{\gamma} \} \stackrel{\text{df.}}{=} X_1 \cup \\ &\quad \{ \langle\sigma, p'' \setminus_2 \gamma\rangle \mid \langle\neg\gamma, p''\rangle \in p(\sigma) \text{ or } \langle\neg\bar{\gamma}, p''\rangle \in p(\sigma) \} \stackrel{\text{df.}}{=} X_2 \cup \\ &\quad \text{if } (X_1 = \emptyset) \wedge (X_2 = \emptyset) \wedge p(\sigma) \neq \emptyset \text{ then } \{\langle\delta, p_0\rangle\} \text{ else } \emptyset \text{ fi} \end{aligned}$$

$$\begin{aligned} \text{d. } M(\text{do } b_1; C_1 \rightarrow S_1 \square \dots \square b_n; C_n \rightarrow S_n \text{ od}) &= \lim_i p_i, \text{ where } (p_0 = p_0 \text{ and}) \\ p_{i+1} &= \lambda\sigma. \text{ if } \sigma = \delta \text{ then } \{\langle\delta, p_0\rangle\} \text{ else} \\ &\quad \{ \langle\gamma_k, p_i \circ M(S_k)\rangle, \langle\neg\gamma_k, q_{K_\sigma \setminus \{k\}}\rangle \mid k \in K_\sigma \} \cup \\ &\quad \text{if } K_\sigma = \emptyset \text{ then } \{\langle\sigma, p_0\rangle\} \text{ else } \emptyset \text{ fi} \end{aligned}$$

$$\text{where } K_\sigma = \{k \mid 1 \leq k \leq n, W(b_k)(\sigma) = tt\}$$

$$q_\emptyset = p_0$$

$$q_{K'} = \lambda\sigma. \{ \langle\neg\gamma_{k'}, q_{K' \setminus \{k'\}}\rangle \mid k' \in K' \}, K' \subseteq \{1, \dots, n\}, K' \neq \emptyset.$$

Clause d of the definition combines a number of ideas. Firstly, the iteration aspect is best understood by comparing it with a similarly structured definition of the simple do ... od construct $S' \equiv \text{do } b_1 \rightarrow S_1 \square \dots \square b_n \rightarrow S_n \text{ od}$. For this we can take $M(S') = \lim_i p_i$, where $(p_0 = p_0 \text{ and})$

$p_{i+1} = \lambda\sigma. \text{ if } \sigma = \delta \text{ then } \{\langle\delta, p_0\rangle\} \text{ else } (\{p_i \circ M(S_k) \mid k \in K_\sigma\} \cup (\text{if } K_\sigma = \emptyset \text{ then } \{\langle\sigma, p_0\rangle\} \text{ else } \emptyset \text{ fi}))$, where $K_\sigma = \{k \mid 1 \leq k \leq n, W(b_k)(\sigma) = tt\}$. Secondly, it contains synchronization elements γ_k prefixed to $p_i \circ M(S_k)$ similarly to the use of γ_k prefixing $M(S_k)$ in definition 6.7. Thirdly, the

$\langle\neg\gamma_k, q_{K_\sigma \setminus \{k\}}\rangle$ parts are based on the ideas on the use of $\neg\gamma$'s discussed above. For $K_\sigma = \{1, 2, 3\}$ we obtain for $\langle\neg\gamma_1, q_{K_\sigma \setminus \{1\}}\rangle$ the following pair: $\langle\neg\gamma_1, \lambda\sigma. \{ \langle\neg\gamma_2, \lambda\sigma. \{ \langle\neg\gamma_3, p_0\rangle \} \rangle, \langle\neg\gamma_3, \lambda\sigma. \{ \langle\neg\gamma_2, p_0\rangle \} \} \rangle \rangle$.

The reason for the accumulation of the $\neg\gamma_k$, is that only if *all* synchronization through γ_k , - for which the corresponding b_k , is true - fails should skip be the outcome of $M(S)$. The last part of clause d ensures that if all b_k are false, $M(S)$ equals skip.

Example. We determine $M(S)$, for

$$S \equiv \{ \underline{\text{do}} \ \bar{C} \rightarrow \underline{a_1} \ \underline{\text{od}} \ \parallel \ (\bar{C}; \underline{a_2} \cup \underline{a_3}) \} \setminus_2 C,$$

where we have returned to the uniform case for simplicity. We obtain, successively, for $M(S)$:

$$\begin{aligned} & ((\lim_i p_{i+1}) \parallel \{ \langle \bar{\gamma}, \{ \langle a_2, p_0 \rangle \} \rangle, \langle a_3, p_0 \rangle \}) \\ &= \\ & \lim_i ((p_{i+1} \parallel \{ \langle \bar{\gamma}, \{ \langle a_2, p_0 \rangle \} \rangle, \langle a_3, p_0 \rangle \}) \setminus_2 \gamma) \\ &= \\ & \lim_i ((\langle \{ \gamma, p_i \} \circ \{ \langle a_1, p_0 \rangle \} \rangle, \langle \neg \gamma, p_0 \rangle \} \parallel \{ \langle \bar{\gamma}, \{ \langle a_2, p_0 \rangle \} \rangle, \langle a_3, p_0 \rangle \}) \setminus_2 \gamma) \\ &= \\ & \lim_i \{ \langle \epsilon, \{ \langle a_1, p_i \rangle \} \parallel \{ \langle a_2, p_0 \rangle \} \setminus_2 \gamma \rangle, \langle \epsilon, \{ \langle a_3, p_0 \rangle \} \rangle, \langle a_3, \{ \langle \epsilon, p_0 \rangle \} \rangle \} \\ &= \\ & \lim_{i \geq 1} \{ \langle \epsilon, \{ \langle a_1, \{ \langle \gamma, \dots \rangle, \langle \neg \gamma, p_0 \rangle \} \parallel \{ \langle a_2, p_0 \rangle \} \setminus_2 \gamma \rangle, \\ & \quad \langle a_2, \{ \langle a_1, p_i \rangle \setminus_2 \gamma \} \rangle \rangle, \\ & \quad \langle \epsilon, \{ \langle a_3, p_0 \rangle \} \rangle, \langle a_3, \{ \langle \epsilon, p_0 \rangle \} \rangle \} \\ &= \\ & \{ \langle \epsilon, \{ \langle a_1, \{ \langle \epsilon, \{ \langle a_2, p_0 \rangle \} \rangle, \langle a_2, \{ \langle \epsilon, p_0 \rangle \} \rangle \rangle, \\ & \quad \langle a_2, \{ \langle a_1, \{ \langle \epsilon, p_0 \rangle \} \rangle \rangle \rangle, \\ & \quad \langle \epsilon, \{ \langle a_3, p_0 \rangle \} \rangle, \langle a_3, \{ \langle \epsilon, p_0 \rangle \} \rangle \} \end{aligned}$$

(where in the final process we have dropped the $\lim_{i \geq 1}$ -prefix, since it is independent of i).

7. COMMUNICATION: CSP AND CCS

In this section we define the semantics of two languages where communication is a central concept, viz. Hoare's Communicating Sequential Processes (CSP) [33], and Milner's Calculus for Communicating Systems (CCS) [44].

We begin with CSP, and use the following syntax for a somewhat abstracted version of it:

DEFINITION 7.1 (a version of CSP). The language L_7 , with elements S , is defined by

$$S ::= \underline{x} := s \mid \text{skip} \mid b \mid S_1; S_2 \mid S_1 \cup S_2 \mid S_1 \parallel S_2 \mid S^* \mid C? \underline{x} \mid C!s \mid S \setminus C \mid \Delta.$$

To clarify the correspondence between L_7 and CSP, we consider a number of constructs in the syntax of CSP proper:

1. $\{P_1; \dots; P_2? \underline{x} \dots \parallel P_2; \dots; P_1!s \dots\}$. This corresponds in L_7 to $\{(\dots C? \underline{x} \dots) \parallel (\dots C!s \dots)\} \setminus C$. We see firstly that " \parallel " in L_7 and CSP correspond. Furthermore, communication over the "channel" $P_1 \leftrightarrow P_2$ (using the matching pair $P_2? \underline{x}$ occurring in P_1 and $P_1!s$ occurring in P_2) is mirrored by the pair of communication commands $C? \underline{x}$, $C!s$. (In general, there will be one pair $C? \dots, C! \dots$ for each channel $P_i \leftrightarrow P_j$; at the \dots , varying arguments may appear.) Moreover, a restriction construct $S \setminus C$ - with the same meaning as the $S \setminus_2 C$ construct of section 6 - is used. In general, there will be as many restrictions $((S \setminus C) \setminus C') \dots$ as there are channels C, C', \dots in the program.
2. Full CSP has constructs of the form $b; C? \underline{x}$ or $b; C!s$ appearing as guards in if...fi or do...od commands. The treatment of these requires a combination of the techniques described in the previous section with those for communication described below. We leave it to the reader to work out the details of this.

We have made no attempt at modelling the distributed termination convention of CSP.

For the definition of the semantics of L_7 we need a new class of processes. The set V is used - as before - for the set of values to be assigned to the variables \underline{x} , \underline{y} of the program, as well as for the values communicated over the channels C .

DEFINITION 7.2. The domain P_7 is defined by the equation

$$P_7 = \{p_0\} \cup (\Sigma \rightarrow P_c((\Sigma \cup \Gamma) \times (P_7 \cup (V \times P_7) \cup (V \rightarrow P_7))))$$

We observe in $P_c(\cdot)$ an extension of the definition as used for P_5 :

$P_c((\Sigma \cup \Gamma) \times P_5)$ is replaced by $P_c((\Sigma \cup \Gamma) \times (P_7 \cup (V \times P_7) \cup (V \rightarrow P_7)))$. The domain we now consider is a variation on the process domain of the general format as discussed in section 2, equation (2.4). We leave the details of the necessary modifications of the underlying mathematics to the reader. We shall use π for a typical element of the set $V \rightarrow P_7$. As before, we assume that Σ contains a dead state δ , and that for each pair $C?$, $C!$ in the language there is a corresponding pair $\gamma, \bar{\gamma}$ in Γ .

The semantics of L_7 is described in

DEFINITION 7.3. The valuation $M: L_7 \rightarrow P_7$ is given by

- a. $M(\underline{x}:=s), \dots, M(S^*)$ are as in definition 5.4. In particular,
 $M(S_1 \parallel S_2) = M(S_1) \parallel M(S_2)$. For processes p_1, p_2 in P_7 , $p_1 \parallel p_2$ will be re-defined below.
- b. $M(C?x) = \lambda \sigma. \{ \langle \gamma, \lambda v. \lambda \bar{\sigma}. \{ \langle \bar{\sigma}\{v/\underline{x}\}, p_0 \rangle \} \rangle \}$
 $M(C!s) = \lambda \sigma. \{ \langle \bar{\gamma}, \langle V(s)(\sigma), p_0 \rangle \rangle \}$
- c. $M(S \setminus C) = M(S) \setminus \gamma$
- d. $M(\Delta) = \lambda \sigma. \{ \langle \delta, p_0 \rangle \}$.

Clause b is the crucial one; it should be understood with respect to the new definition of " \parallel " contained in

DEFINITION 7.4.

- a. $p \parallel q$ is defined as usual for p or q equal p_0 or of infinite degree.
 Otherwise, $p = \lambda \sigma. X, q = \lambda \sigma. Y$, and we put

$$(\lambda\sigma.X) \parallel (\lambda\sigma.Y) = \lambda\sigma.(\{(x \parallel (\lambda\sigma.Y)) \mid x \in X\} \cup \{((\lambda\sigma.X) \parallel y) \mid y \in Y\} \cup \{\langle\sigma, p' \parallel p'' \rangle \mid \langle\gamma, \pi \rangle \in X, \langle\bar{\gamma}, \langle v, p'' \rangle \rangle \in Y, \pi(v) = p'\}).$$

Let β be a typical element of $\Sigma \cup \Gamma$, and π of $V \rightarrow P_7$. We put

$$\begin{aligned} \langle\beta, p \rangle \parallel \lambda\sigma.Y &= \langle\beta, p \parallel \lambda\sigma.Y \rangle, \langle\beta, \langle v, p \rangle \rangle \parallel \lambda\sigma.Y = \langle\beta, \langle v, p \parallel \lambda\sigma.Y \rangle \rangle, \langle\beta, \pi \rangle \parallel \lambda\sigma.Y = \\ &= \langle\beta, \lambda w. \pi(w) \rangle \parallel \lambda\sigma.Y = \langle\beta, \lambda w. (\pi(w) \parallel \lambda\sigma.Y) \rangle, \text{ and similarly for } \\ \lambda\sigma.X \parallel \langle\beta, p \rangle, \text{ etc.} \end{aligned}$$

b. $p \setminus \gamma = p \setminus_2 \gamma$, with \setminus_2 as in definition 6.4.

The heart of the definition is the third term on the right-hand side of the formula for $(\lambda\sigma.X) \parallel (\lambda\sigma.Y)$. Here the value v is transmitted between $p = \lambda\sigma.X = \lambda\sigma.\{\dots, \langle\gamma, \pi \rangle, \dots\}$ and $q = \lambda\sigma.Y = \lambda\sigma.\{\dots, \langle\bar{\gamma}, \langle v, p'' \rangle \rangle, \dots\}$, determining as possible candidate for continuation in the process $(p \parallel q)$ when applied to σ , the process $p' \parallel p''$, with $p' = \pi(v)$: At the synchronization point corresponding to the pair $\langle\gamma, \bar{\gamma}\rangle$, the value v is supplied to the function π determining process $p' = \pi(v)$ as part of the continuation $p' \parallel p''$. Let us apply definitions 7.3, 7.4 to the simple example $S \equiv (C; \underline{x} \parallel C!1) \setminus C$. We obtain $M(S) = M((C; \underline{x} \parallel C!1) \setminus C) = p \setminus \gamma$, where $p = M((C; \underline{x} \parallel C!1))$. By definition 7.3, we obtain for p : $p = \lambda\sigma.\{\langle\gamma, \lambda v. \lambda\bar{\sigma}.\{\langle\bar{\sigma}\{v/\underline{x}\}, p_0 \rangle\} \rangle \parallel \lambda\bar{\sigma}.\{\langle\bar{\gamma}, \langle 1, p_0 \rangle \rangle\} = (\text{by def. 7.4})$
 $\lambda\sigma.(\langle\gamma, \dots \rangle, \langle\bar{\gamma}, \dots \rangle, \langle\sigma, [\lambda v. \lambda\bar{\sigma}.\{\langle\bar{\sigma}\{v/\underline{x}\}, p_0 \rangle\}](1) \parallel p_0 \rangle) =$
 $\lambda\sigma.(\dots, \langle\sigma, \lambda\bar{\sigma}.\{\langle\bar{\sigma}\{1/\underline{x}\}, p_0 \rangle\} \rangle)$. Applying the definition of $\setminus \gamma$ to this results in deletion of the \dots , and we obtain as final result for $M(S)$: $\lambda\sigma.\{\langle\sigma, \lambda\bar{\sigma}.\{\langle\bar{\sigma}\{1/\underline{x}\}, p_0 \rangle\} \rangle\}$ which is, indeed, a (somewhat elaborate) way of setting \underline{x} to 1.

Definition 7.4 owes a lot to the ideas of Milner [40,44]. Also, it is close to the approach to CSP semantics as described in [24]. The main difference lies in our use of processes as underlying mathematical structure rather than a denotational system with power domains (as in [40]) or with infinite trees (as in [24]). From the variety of operational approaches to CSP semantics we mention [18,25,34,49,53]. Applications of semantics to proof theory (in proving the soundness of a proof system) are studied in [1], cf. also [2].

We close our treatment of CSP with a few words on the definition of yield for $p \in P_7$. In fact, the same definitions both for a *path* for $\langle\sigma, p \rangle$ and for p^+ can be used as in section 6. Observe, however, that this implies that *only* pairs $\langle\beta_{i+1}, p_{i+1} \rangle \in p_i(\beta_i)$ (for $\beta_i \in \Sigma$) contribute to such paths, whereas pairs $\langle v, p \rangle$ or functions π do not appear in any path.

We now turn to the definition of Milner's CCS. Contrary to the previous languages, CCS is an expression based language. Synchronization and communication are very similar to CSP, but there is no notion of assignment or sequential composition as we had previously. Also, CCS features recursion rather than iteration. In the syntax we shall give for L_8 we have introduced a deviation from CCS in that we have separated λ -abstraction ($\lambda x \dots$) from synchronization ($\langle c, \dots \rangle, \langle \bar{c}, \dots \rangle$); in CCS, these notions are combined in the notation $\alpha x \dots$ or $\bar{\alpha} v \dots$. We first give a simple version of L_8 , where recursive declarations are parameterless:

DEFINITION 7.5 (a version of CCS). The language L_8 , with elements s, \dots is defined by

$$s ::= \underline{\text{nil}} \mid \langle e, s \rangle \mid \langle c, s \rangle \mid \langle \bar{c}, s \rangle \mid s_1 \text{ us } s_2 \mid s_1 \parallel s_2 \mid s \setminus c \mid \xi \mid \mu \xi[s] \mid \lambda x. s$$

where s in $\mu \xi[s]$ is restricted as stated below.

Remarks.

1. In the construct $\langle e, s \rangle$, e is a *simple* expression, defined for example by $e ::= \underline{x} \mid f(e_1, \dots, e_n)$, for f an n -ary function symbol. We assume that evaluation of e always terminates, delivering a value $v \in V$.
2. Expressions s replace statements S ; synchronization prefixes $\langle c, \dots \rangle$, $\langle \bar{c}, \dots \rangle$ replace commands C, \bar{C} as used above.
3. CCS's construct $\alpha x. B$ is written as $\langle c, \lambda x. s_B \rangle$, with s_B the construct in L_8 corresponding to B .
4. We have not taken the trouble to incorporate the relabelling feature of CCS.
5. The recursive construct $\mu \xi[s]$ corresponds to a "call" of some b defined by $b \Leftarrow B$ in CCS. Moreover, the s in $\mu \xi[s]$ is - for the moment - assumed to be of ground type (i.e., not of the form $\lambda x. s'$).

The process domain for L_8 is introduced in

DEFINITION 7.6. The process domain P_8 is defined by

$$P_8 = \{p_0\} \cup P_c((\Gamma \cup V \cup \{\varepsilon\}) \times (P_8 \cup (V \rightarrow P_8))).$$

For the semantics of L_8 we need a class of environments $E = E_1 \times E_2$, where $E_1 = \text{Var} \rightarrow V$, $E_2 = X \rightarrow P_7$. (X is the set of variables ξ used in recursive definitions.) Thus, taking $\eta = \langle \eta_1, \eta_2 \rangle \in E$, $\eta_1(x) = v$ and $\eta_2(\xi) = p$ are meaningful equations. As before, V is the valuation for simple expressions e , yielding results $V(e)(\eta_1) = v$.

DEFINITION 7.7. The valuation $M: L_8 \rightarrow (E \rightarrow P_8)$ is defined by

- a. $M(\underline{\text{nil}})(\eta) = p_0$
- b. $M(\langle e, s \rangle)(\eta) = \{ \langle V(e)(\eta_1), M(s)(\eta) \rangle \}$
- c. $M(\langle c, s \rangle)(\eta) = \{ \langle \gamma, M(s)(\eta) \rangle \}$, where γ corresponds to c , and similarly for $\langle \bar{c}, s \rangle$.
- d. $M(s_1 \text{ us } s_2)(\eta) = M(s_1)(\eta) \cup M(s_2)(\eta)$, $M(s_1 \parallel s_2)(\eta) = M(s_1)(\eta) \parallel M(s_2)(\eta)$, with " \parallel " to be defined below
- e. $M(s \setminus c)(\eta) = M(s)(\eta) \setminus \gamma$
- f. $M(\xi)(\eta) = \eta_2(\xi)$, $M(\mu \xi[s])(\eta) = \lim_i p_i$, where $(p_0 = p_0 \text{ and } p_{i+1} = \{ \langle \epsilon, M(s)(\eta\{p_i/\xi\}) \rangle \})$
- g. $M(\lambda x.s)(\eta) = \{ \lambda v.M(s)(\eta\{v/x\}) \}$.

Here $p \setminus \gamma$ is as $p \setminus_1 \gamma$ in definition 6.4 (in order to use \setminus_2 , we would have to extend Γ with a dead symbol δ). Furthermore, the definition of " \parallel " is very similar to the one used in the CSP definition, as can be seen from **DEFINITION 7.8.** For $X, Y \in P_c(\cdot)$ of finite nonzero degree we put

$$X \parallel Y = \{ x \parallel y \mid x \in X \} \cup \{ x \parallel y \mid y \in Y \} \cup \{ \langle \epsilon, p' \parallel p'' \rangle \mid \langle \gamma, \pi \rangle \in X, \langle \bar{\gamma}, \{ \langle v, p'' \rangle \} \rangle \in Y, \text{ where } \pi(v) = p' \}.$$

Here $\langle \beta, p \rangle \parallel Y = \langle \beta, p \parallel Y \rangle$, $\pi \parallel Y = (\lambda v. \pi(v)) \parallel Y = \lambda v. (\pi(v) \parallel Y)$, etc.

Example. For constructs b_1, b_2 in CCS defined by $b_1 \Leftarrow \alpha x. x+1.b$, and $b_2 \Leftarrow \bar{\alpha} y+3.b_2$, we have as corresponding $s_1, s_2 \in L_8$: $s_1 \equiv \mu \xi[\langle c, \lambda x. \langle x+1, \xi \rangle \rangle]$, $s_2 \equiv \mu \xi[\langle \bar{c}, \langle y+3, \xi \rangle \rangle]$, and for $p_i = M(s_i)(\eta)$ we obtain $p_1 = \lim_i p_1^{(i)}$, where $p_1^{(i+1)} = \{ \langle \epsilon, \{ \langle \gamma, \{ \lambda v. \{ \langle v+1, p_1^{(i)} \rangle \} \} \rangle \} \rangle \}$, $p_2 = \lim_i p_2^{(i)}$, where $p_2^{(i+1)} = \{ \langle \epsilon, \{ \langle \bar{\gamma}, \{ \langle \eta_1(y)+3, p_2^{(i)} \rangle \} \} \rangle \} \}$. Also, it can be shown that $(p_1 \parallel p_2) \setminus \gamma = \lim_i q^{(i)}$, where $q^{(i+1)} = \{ \langle \epsilon, \{ \langle \epsilon, \{ \langle \eta_1(y)+4, q^{(i)} \rangle \} \} \rangle \} \}$.

Remarks.

1. The use of $\langle \epsilon, \dots \rangle$ in the process theory corresponds to the unobservable action τ of CCS.

2. Processes p in P_g are quite close to communication trees (Ch.6 of [44]). Important differences are
- (i) the collection of successors of a node in a communication tree is a multiset rather than a set
 - (ii) the "mathematical sophistication we do not want to be bothered with" (a quotation from [44] referring to the case of infinite trees) is - if our attempts have been successful - present in our theory.
3. Recursive behaviour expressions *with* parameters - of the form $b(x) \Leftarrow B$ in CCS - can be dealt with very similarly to the above treatment of $\mu\xi[s]$. Without going into details, something along the following lines will have to be done: The syntax of L_g is extended with the clause $s ::= \dots | \xi(s_1) \dots (s_n)$. Moreover, the terms s - in particular the variables ξ - are now supposed to be *typed* as in, e.g., the typed lambda-calculus. We drop the requirement that s in $\mu\xi[s]$ be of ground type, and adapt the choice of p_0 - for the zero element of the CS converging to the meaning of $\mu\xi[s]$ - replacing it by $\frac{\lambda v \dots \lambda v. p_0}{\sqrt[n]{x}}$, where n is such that the type of ξ is $V^n \rightarrow V (n \geq 0)$.
4. The use of the $\langle \varepsilon, \dots \rangle$ prefix in definition 7.7 f could be avoided if we were to adopt Milner's requirement that "no behaviour may call itself recursively without passing a guard". Syntactically, this would amount to the requirement that, in a recursive construct $\mu\xi[s]$, ξ occurs in s only within subterms of the form $\langle c, \dots \xi \dots \rangle$ or $\langle \bar{c}, \dots \xi \dots \rangle$ or $\langle e, \dots \xi \dots \rangle$. In this way, the contraction property of $T' = \lambda p. M(s)(\eta\{p/\xi\})$ is guaranteed. In our treatment, the same result is obtained by using the CS of iterates $T^i(p_0)$ for T of the form $T = \lambda p. \{\langle \varepsilon, M(s)(\eta\{p/\xi\}) \rangle\}$. (As remarked already in section 3, we are not sure that this precaution is indeed necessary, but we do not know how to prove that $\langle T^i(p_0) \rangle_i$ is a CS without it.)

This concludes our discussion of CCS semantics. We close with a remark on p^+ for $p \in P_g$. Analogously to what we did in previous sections (3.4), we can define p^+ over the alphabet $V \cup \Gamma$ - where, just as we did for CSP, *paths* are defined such that constituents π of p do not contribute to its paths. Also, we may again put $p \sim q \iff p^+ = q^+$, and investigate properties of " \sim ".

8. MISCELLANEOUS NOTIONS IN CONCURRENCY

There is an astounding variety of notions in concurrency, and only a few of them have been investigated in the preceding sections. In this section we briefly comment upon some additional topics. In most cases we provide some suggestions on how the theory of processes could be linked to the notion concerned. Sometimes, we provide no more than some pointers to problems still to be dealt with.

1. *Critical sections.* Let us extend the language L_3 (section 5) with the construct $[S]$. Thus, as syntax for L_3 we have

$$S ::= \underline{x} := s \mid \underline{\text{skip}} \mid b \mid S_1; S_2 \mid S_1 \cup S_2 \mid S_1 \parallel S_2 \mid S^* \mid [S].$$

Here $[S]$ has as intended meaning that execution of S is not interruptible (S is "locked".) Using P_3 as in section 5, we put $M([S]) = \lambda\sigma.$

$\{\langle\sigma', p_0\rangle \mid \sigma' \in M(S)^+(\sigma)\}$, where p^+ is the (usual) yield of p . This expresses that S is, by $[...]$, turned into an elementary action execution of which cannot be merged at intermediate stages with execution of some parallel statement. Note that σ' in $M(S)^+(\sigma)$ may equal \perp ; strictly speaking, this requires appropriate adaptation of the definition of Σ and of P_3 .

2. *Guarded commands.* In section 6 we encountered a guarded command such

as $\text{if } b_1 \rightarrow S_1 \square b_2 \rightarrow S_2 \text{ fi}$, to be modelled by

$(b_1; S_1) \cup (b_2; S_2) \cup (\neg b_1; \neg b_2; \Delta)$. This correspondence implies the following:

Suppose that, e.g., in state σ it turns out that b_1 is true, and S_1 is selected for execution. Before starting execution of S_1 , an interleaving action of some parallel S' may have changed σ to σ' for which b is no longer true, and we see that we cannot be sure that the first action of S_1 is executed with b_1 true, even though the "branch" $b_1; S_1$ was chosen since b_1 was true for σ . A different interpretation of a guarded command is possible - and may even be the intended one - viz. one in which the first elementary action of S_1 is taken immediately after it was selected on the basis of b_1 being true. Let us write $b \Rightarrow S$ for a construct which, contrary to $b; S$, allows no interleaving actions between b and the first action of S . Including this construct in L_3 requires an extension of M with the clause $M(b \Rightarrow S) = \lambda\sigma. \text{ if } W(b)(\sigma) \text{ then } M(S)(\sigma) \text{ else } \emptyset \text{ fi.}$

(The reader should contrast this with $M(b; S) = M(S) \circ M(b) = \lambda\sigma. \text{ if } W(b)(\sigma) \text{ then } \{\langle\sigma, M(S)\rangle\} \text{ else } \emptyset \text{ fi.}$)

3. *Await statement* [47]. Consider the await statement $(*)$: await b then S . Operationally, when execution reaches $(*)$, if b is true then S is executed as in divisible action, if b is false execution waits. Combining the ideas of 1,2 above, we can model $(*)$ by $b \Rightarrow [S]$.

4. *Indivisible parameter passing*. Extend L_3 with the clause

$$S ::= \dots \mid (\lambda \underline{x}.S)(t)$$

where $(\lambda \underline{x}.S)(t)$ is equivalent to $\underline{x} := t; S$, but allows no concurrent action at the ";". We can deal with this by putting $M((\lambda \underline{x}.S)(t)) = \lambda \sigma. M(S)(\sigma\{V(t)(\sigma)/\underline{x}\})$.

5. *Histories on channels*. Extend L_5 with

$$S ::= \dots \mid \text{read } (\underline{x}, C_k) \mid \text{write } (s, C_k).$$

Here C_1, \dots, C_n are channels (as before), but now may contain *sequences* of values. States are now pairs $\langle \sigma, \rho \rangle$, σ as usual, $\rho = \rho_1, \dots, \rho_n$, each ρ_i a - possibly infinite - sequence of values, the current contents of channel C_i . Let $\epsilon(\rho_i)$ test whether the sequence ρ_i is empty, and let ρ_i^\uparrow be the last element of ρ_i . Also, ρ_i^\downarrow denotes ρ_i with its last element deleted. The central clauses in the definition are

$$\begin{aligned} M(\text{read}(\underline{x}, C_k)) &= \\ \lambda \sigma \rho. \text{ if } \epsilon(\rho_k) \text{ then } \emptyset \text{ else } \{ \langle \sigma\{\rho_k^\uparrow/\underline{x}\}, \rho_k^\downarrow, p_0 \rangle \} \text{ fi} \\ M(\text{write}(s, C_k)) &= \\ \lambda \sigma \rho. \{ \langle \sigma, V(s)(\sigma) \hat{\rho}_k, p_0 \rangle \}. \end{aligned}$$

Here $\hat{}$ denotes concatenation (of sequences over $V^* \cup V^\omega$), and in the ρ -component we have not mentioned the channels which are not referred to (and remain unchanged). Observe that reading from an empty channel results in an empty output. As usual, this captures the operational notion of waiting.

6. *Linking channels*. Let p, q be processes in the domain $P = \{p_0\} \cup P_c((A \cup \Gamma \cup \{\epsilon\}) \times P)$. Previously, synchronization of p, q was achieved through matching pairs $\gamma, \bar{\gamma}$ occurring in p and q respectively. Such matching

can also be "programmed" by using the notation $(p \parallel q)[\gamma:\delta]$ which expresses that δ (in this paragraph standing for some element of Γ rather than for the dead state) now acts as $\bar{\gamma}$, i.e., we define

$$(p \parallel q)[\gamma:\delta] = (p \parallel q) (" \parallel " \text{ as in section 3}) \cup \{ \langle \epsilon, p' \parallel q' \rangle \mid \langle \gamma, p' \rangle \in p, \langle \delta, q' \rangle \in q \}.$$

An operation such as $(p \parallel q)[\gamma:\delta]$ is reminiscent of the use of channel linking in Back & Mannila [8]. Also, it resembles the use of equalities $c_i.a = c_j.b$ in [52], which in a similar manner establish linking between "ports" of processes p, p_1, \dots, p_n occurring in their com...moc construct (albeit that their definition of " \parallel " differs from the one used throughout our paper).

7. *Logic*. Let α be a some formula of, e.g., predicate or temporal logic ([50]). We can distinguish a variety of ways of interpreting α in process p . Let, e.g., $p \in P_3$. We may choose $\alpha(p_0) = tt$ or $\alpha(p_0) = ff$, $\alpha(\lambda\sigma.X) = \lambda\sigma.\alpha(X)$, $\alpha(X) = \bigwedge_{x \in X} \alpha(x)$ or $\alpha(X) = \bigvee_{x \in X} \alpha(x)$, $\alpha(\langle \sigma, p_0 \rangle) = \alpha(\sigma)$, and, for $p \neq p_0$, $\alpha(\langle \sigma, p \rangle) = \alpha(\sigma) \vee \alpha(p(\sigma))$, or $\alpha(\langle \sigma, p \rangle) = \alpha(\sigma) \wedge \alpha(p(\sigma))$. E.g., the combination of definitions $\alpha(X) = \bigwedge_{x \in X} \alpha(x)$ with $\alpha(\langle \sigma, p \rangle) = \alpha(\sigma) \vee \alpha(p(\sigma))$ states that $\alpha(p)$ is true in σ whenever α is true in at least one node along each path for $\langle \sigma, p \rangle$. The implications of these definitions for the model theory of temporal logic deserve further study. We also would like to know whether the results of Emerson & Clarke [19] can be applied in the context of processes.

8. *ADA rendez-vous, distributed processes, data flow*. These notions are mentioned here for the sake of completeness. We have no semantic definitions for them at the moment of writing this. For the ADA rendez-vous this should not be too difficult, because of its close connection with CSP(cf.[26]). For DP([15,27]) and data flow ([14,16,23,35,36,37,38,51,58]) we need further study.

9. *Fairness*. There is a well-known correspondence between fairness and unbounded nondeterminacy (see, e.g., Apt & Olderog [3]). Since our processes allow a smooth treatment of the latter, the question arises as to their role for defining the former. We know how to do this, and we hope to describe it in a future publication (which is not along the lines of the approach sketched in the remark in [11]).

This concludes our discussion of some miscellaneous topics in concurrency, and brings us to the end of this paper.

REFERENCES

- [1] APT, K.R., *Formal justification of a proof system for communicating sequential processes*, preprint, EUR, 1981.
- [2] APT, K.R., N. FRANCEZ & W.P. DE ROEVER, *A proof system for communicating sequential processes*, ACM TOPLAS, 2 (1980) 359-385.
- [3] APT, K.R. & E.R. OLDEROG, *Proof rules dealing with fairness*, Proc. Logic of Programs 1981 (D.Kozen, ed.), 1-9, Lecture Notes in Computer Science 131, Springer, 1982.
- [4] APT, K.R. & G.D. PLOTKIN, *A Cook's tour of countable nondeterminism*, Proc. 8th ICALP (S. Even & O. Kariv, eds), 479-494, Lecture Notes in Computer Science, 115, Springer, 1981.
- [5] ARNOLD, A. & M. NIVAT, *Metric interpretations of infinite trees and semantics of nondeterministic recursive programs*, Theoretical Computer Science, 11 (1980) 181-206.
- [6] ARNOLD, A. & N. NIVAT, *The metric space of infinite trees. Algebraic and topological properties*, Fund. Inf. III, 4 (1980) 445-476.
- [7] BACK, R.J., *Semantics of unbounded nondeterminism*, Proc. 7th ICALP (J.W. de Bakker & J. van Leeuwen, eds), 52-63, Lecture Notes in Computer Science, 85, Springer, 1980.
- [8] BACK, R.J. & N. MANNILA, *A refinement of Kahn's semantics to handle non-determinism and communication*, preprint, University of Helsinki, 1982.
- [9] DE BAKKER, J.W., *Semantics of infinite processes using generalized trees*, Proc. 6th MFCS (J. Gruska, ed.), 240-252, Lecture Notes in Computer Science, 53, Springer, 1977.
- [10] DE BAKKER, J.W., *Mathematical Theory of Program Correctness*, Prentice-Hall International, 1980.
- [11] DE BAKKER, J.W. & J.I. ZUCKER, *Denotational semantics of concurrency*, Proc. 14th ACM Symp. on Theory of Computing, pp. 153-158, 1982.
- [12] BEKIC, H., *Towards a mathematical theory of processes*, Tech. Report TR 25-125, IBM Lab., Vienna, 1971.

- [13] BERGSTRA, J.A. & J.W. KLOP, *Fixed point semantics in process algebras*, Department of Computer Science Technical Report, Mathematisch Centrum, 1982.
- [14] BOUSSINOT, F., *Proposition de sémantique dénotationnelle pour des réseaux de processus avec opérateur de mélange équitable*, Theoretical Computer Science, 18 (1982), 173-206.
- [15] BRINCH-HANSEN, P., *Distributed processes: a concurrent programming concept*, Comm. ACM 21 (1978), 934-941.
- [16] BROCK, J.D. & W.B. ACKERMANN, *Scenarios: a model of non-determinate computation*, Proc. Formalization of Programming concepts (J. Diaz & I. Ramos, eds.), 252-259, Lecture Notes in Computer Science, 107, Springer, 1981.
- [17] DE BRUIN, A., *On the existence of Cook semantics*, Report IW 163/81, Mathematisch Centrum, 1981.
- [18] COUSOT, P. & R. COUSOT, *Semantic analysis of communicating sequential processes*, Proc. 7th ICALP (J.W. de Bakker & J. van Leeuwen, eds.) 119-133, Lecture Notes in Computer Science, 85, Springer, 1980.
- [19] DUGUNDJI, J., *Topology*, Allen & Bacon, 1966.
- [20] DIJKSTRA, E.W., *A Discipline of Programming*, Prentice-Hall, 1976.
- [21] EMERSON, E.A. & E.M. CLARKE, *Characterizing correctness properties of parallel programs using fixpoints*, Proc. 7th ICALP (J.W. de Bakker & J. van Leeuwen, eds.) 169-181, Lecture Notes in Computer Science, 85, Springer, 1980.
- [22] ENGELKING, R., *General Topology*, Polish Scientific Publishers, 1977.
- [23] FAUSTINI, A.A., *An operational semantics for pure data flow*, Proc. 9th ICALP (M.Nielsen & E.M.Schmidt, eds.), 212-224, Lecture Notes Computer Science 140, Springer, 1982.
- [24] FRANCEZ, N., C.A.R. HOARE, D.J. LEHMANN & W.P. DE ROEVER, *Semantics of nondeterminism, concurrency and communication*, J. Comp. Syst. Sciences, 19 (1979), 290-308.
- [25] FRANCEZ, N., D.J. LEHMANN & A. PNUELI, *Linear history semantics for distributed languages*, Proc. 21st Symp. Foundations of Computer Science, IEEE 1980, 143-151.

- [26] GERTH, R., *A sound and complete Hoare axiomatization of the ADA-rendez-vous*, Proc. 9th ICALP (M.Nielsen & E.M.Schmidt, eds.), 252-264, Lecture Notes in Computer Science 140, Springer, 1982.
- [27] GERTH, R., W.P. DE ROEVER & M. RONCKEN, *Procedures and concurrency: a study in proof*, Proc. Int. Symp. on Programming (M. Dezani-Ciancaglini & U. Montanari, eds.), 132-163, Lecture Notes in Computer Science, 137, 1982.
- [28] GORDON, M., *The Denotational Description of Programming Languages*, Springer, 1979.
- [29] HAHN, H., *Reelle Funktionen*, Chelsea, 1948.
- [30] HENNESSY, N. & R. MILNER, *On observing nondeterminism and concurrency*, Proc. 7th ICALP (J.W. de Bakker & J. van Leeuwen, eds.), 299-309, Lecture Notes in Computer Science, 85, 1980.
- [31] HENNESSY, M. & G.D. PLOTKIN, *Full abstraction for a simple parallel programming language*, Proc. 8th MFCS (J. Bečvar^V, ed.), 108-120, Lecture Notes in Computer Science, 74, 1979.
- [32] HITCHCOCK, P. & D. PARK, *Induction rules and termination proofs*, Proc. 1st ICALP (M. Nivat, ed.), 225-251, North-Holland, 1973.
- [33] HOARE, C.A.R., *Communicating sequential processes*, Comm. ACM, 21 (1978), 666-677.
- [34] HOARE, C.A.R., *A model for communicating sequential processes*, Technical Monograph PRG-22, Oxford University, 1981.
- [35] KAHN, G., *The semantics of a simple language for parallel programming*, Proc. IFIP 74, North-Holland, 1974.
- [36] KAHN, G. & D.B. MACQUEEN, *Coroutines and networks of parallel processes*, Proc. IFIP 77, 993-998, North-Holland, 1977.
- [37] KELLER, R.M., *Denotational models for parallel programs with indeterminate operators*, Formal Description of Programming Concepts (E.J. Neuhold, ed.), 337-366, North-Holland, 1978.
- [38] KOSINSKI, P.R., *A straightforward denotational semantics for non-determinate data flow programs*, Conf. Record 5th ACM Symp. Principles of Programming Languages, 1978, 214-221.

- [39] LEHMANN, D.J., *Categories for fixed point semantics*, Proc. 17th IEEE Symp. on Foundations of Computer Science, 122-126, 1976.
- [40] MILNE, G. & R. MILNER, *Concurrent processes and their syntax*, J. ACM, 26 (1979), 302-321.
- [41] MILNE, R. & C. STRACHEY, *A Theory of Programming Language Semantics*, Chapman & Hall, 1977.
- [42] MILNER, R., *Processes; a mathematical model of computing agents*, Proc. Logic Coll. 73, (Rose & Shepherdson, eds.), North-Holland, 1973.
- [43] MILNER, R., *Flow graphs and flow algebras*, J. ACM, 26 (1979), 794-818.
- [44] MILNER, R., *A Calculus for Communicating Systems*, Lecture Notes in Computer Science, 92, 1980.
- [45] NIVAT, M., *Infinite words, infinite trees, infinite computations*, Foundations of Computer Science III. 2 (J.W. de Bakker & J. van Leeuwen, eds.) 3-52, Mathematical Centre Tracts, 109, 1979.
- [46] NIVAT, M., *Synchronization of concurrent processes*, *Formal Language Theory* (R.V. Book, ed.), 429-454, Academic Press, 1980.
- [47] OWICKI, S. & D. GRIES, *Verifying properties of parallel programs, an axiomatic approach*, Comm. ACM, 19 (1976), 279-285.
- [48] PLOTKIN, G.D., *A power domain construction*, SIAM J. on Comp., 5 (1976), 452-487.
- [49] PLOTKIN, G.D., *An operational semantics for CSP*, Proc. IFIP Working Conference on Formal Description of Programming Concepts II (D. Bjørner, ed.), North-Holland, to appear.
- [50] PNUELI, A., *The temporal logic of programs*, Proc. 19th Ann. Symp. on Foundations of Comp. Science, IEEE, 46-57, 1977.
- [51] PRATT, V.R., *On the composition of processes*, Proc. 9th ACM Symp. on Principles of Programming Languages, 213-223, 1982.
- [52] REM, M. & J.L.A. VAN DE SNEPSCHEUT, *Some observations on partially ordered computations*, Preprint, Eindhoven University of Technology, 1981.

- [53] ROUNDS, W.C. & S.D. BROOKES, *Possible futures, acceptances, refusals, and communicating processes*, Proc. 22nd Symp. Foundations of Computer Science, IEEE, 140-149, 1981.
- [54] SCOTT, D., *Data types as lattices*, SIAM J. on Comp., 5 (1976), 522-587.
- [55] SCOTT, D.S., *Domains for denotational semantics*, Proc. 9th ICALP (M.Nielsen & E.M. Schmidt, eds), 577-613, Lecture Notes in Computer Science 140, Springer, 1982.
- [56] SMYTH, M.B., *Power domains*, J. Comp. Syst. Sciences, 16 (1978), 23-36.
- [57] STOY, J., *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory*, MIT Press, 1977.
- [58] WADGE, W., *An extensional treatment of dataflow deadlock*, Theoretical Computer Science, 13 (1981), 3-16.

APPENDIX A. HAHN'S THEOREM

Since the proof of Hahn's theorem (theorem 2.9) is not easily accessible, we present the proof in this appendix. We repeat the theorem as

THEOREM A (= theorem 2.9). If (M, d) is a complete metric space, then so is $(P_c(M), d)$, where $P_c(M)$ denotes the collection of all closed subsets of M , and the distance d for sets is the Hausdorff distance. Moreover, we have, for $\langle X_n \rangle_n$ a CS of closed sets,

$$\lim_n X_n = X \stackrel{\text{df.}}{=} \{x \mid x = \lim_n x_n, \langle x_n \rangle_n \text{ a CS in } M \text{ such that } x_n \in X_n\}.$$

Proof. Clearly, we may assume that $X_n \neq \emptyset$ for almost all n . We show that (i) X is closed, and (ii) $d(X_n, X) \rightarrow 0$.

Ad(i). Let $\langle y_n \rangle_n$ be a CS in X with $y_n \rightarrow y$. We show that $y \in X$. Let, for each n , $\langle x_{i,n} \rangle_i$ be a CS such that $x_{i,n} \in X_n$, and $x_{i,n} \rightarrow y_n$. Consider the diagonal sequence $\langle x_{n,n} \rangle_n$, $x_{n,n} \in X_n$. Then $\langle x_{n,n} \rangle_n$ is a CS, with $x_{n,n} \rightarrow y$. Therefore, by the definition of X , we have that $y \in X$.

Ad(ii). The proof of this fact is more involved. We have to show that $\forall \epsilon \exists N \forall n \geq N [d(X_n, X) < \epsilon]$, i.e.,

$$(A1) \quad \forall \epsilon \exists N \forall n \geq N \forall x_n \in X_n [d(x_n, X) < \epsilon]$$

$$(A2) \quad \forall \epsilon \exists N \forall n \geq N \forall x \in X [d(x, X_n) < \epsilon]$$

or, equivalently,

$$(A3) \quad \forall \epsilon \exists N \forall n \geq N \forall x_n \in X_n \exists x \in X [d(x_n, x) < \epsilon]$$

$$(A4) \quad \forall \epsilon \exists N \forall n \geq N \forall x \in X \exists x_n \in X_n [d(x_n, x) < \epsilon].$$

We first prove (A3). Choose ϵ . Then $(*) : \exists N \forall m, n \geq N [d(X_m, X_n) < \epsilon/2]$.

Now take any $m \geq N$, and any $x_m \in X_m$. We show how to find $x \in X$ such that $d(x_m, x) < \epsilon$. There exists a sequence

$$m = N_0 < N_1 < N_2 < \dots$$

such that $(**): n, n' \geq N_k \Rightarrow d(X_n, X_{n'}) < \epsilon/2^{k+1}$. Now define a sequence $\langle x_n \rangle_n$ as follows: For $n < N_0$, x_n is arbitrary. For $n = N_0$, $x_n = x_{N_0}$ ($= x_m$).

For $N_0 < n \leq N_1$: take any x_n such that $d(x_{N_0}, x_n) < \varepsilon/2$ (by (*))

For $N_1 < n \leq N_2$: take any x_n such that $d(x_{N_1}, x_n) < \varepsilon/4$ (by (**))

...

For $N_k < n \leq N_{k+1}$: take any x_n such that $d(x_{N_k}, x_n) < \varepsilon/2^{k+1}$ (by (**))

...

Then $\langle x_n \rangle$ is a CS, since for, say, $N_k < n \leq N_{k+1}$, and any $m \geq n$, $d(x_m, x_n) \leq d(x_n, x_{N_{k+1}}) + d(x_{N_{k+1}}, x_{N_{k+2}}) + \dots + d(x_{N_{k+\ell}}, x_m)$

$$< \varepsilon/2^{k+1} + \varepsilon/2^{k+2} + \dots < \varepsilon/2^k.$$

So, by completeness of (M, d) , $x_n \rightarrow x$ for some x . Thus, $x \in X$. Furthermore, we have $\forall n > m$, $d(x_m, x_n) < \varepsilon/2 + \varepsilon/4 + \dots$ (by similar reasoning) $< \varepsilon$. Hence, $d(x_m, x) \leq \varepsilon$. Altogether, we have proved (A3). We now prove (A4). Choose some ε . As before, there exists N such that $\forall m, n \geq N [d(x_m, x_n) < \varepsilon/2]$. Let $x \in X$ and $m \geq N$. We show that $d(x, x_m) < \varepsilon$. There exists a CS $\langle x_n \rangle$ such that $x_n \rightarrow x$. We have, for $m \geq N$, $d(x_n, x_m) < \varepsilon/2$, so $d(x_n, x_m) < \varepsilon/2$ for all $n \geq N$. Hence (since $x_n \rightarrow x$) $d(x, x_m) \leq \varepsilon/2 < \varepsilon$, which proves (A4). \square

APPENDIX B

In this appendix, we present a detailed proof of lemma 2.15. The main part consists in the justification of the definitions of $p \circ q$, $p \cup q$ and $p \parallel q$, as provided in theorems B7, B12, B14 and B16 and their corollaries. Preliminary to these theorems there are some general lemmas on the Hausdorff distance. Throughout the Appendix lhs and rhs stand for left-hand side and right-hand side, respectively.

Up to lemma B5 we assume X, Y, \dots are subsets of an arbitrary metric space (M, d) , and assume, moreover:

$$x \in X, x' \in X', y \in Y, y' \in Y'.$$

LEMMA B1. *Given $\ell > 0$ $d(X, X') \leq \ell$ if and only if:*

$$(B1) \quad \forall x \exists x' d(x, x') \leq \ell, \text{ and}$$

$$(B2) \quad \forall x' \exists x d(x, x') \leq \ell$$

Proof. $d(X, X') \leq \ell$

$$\Leftrightarrow \forall x d(x, X') \leq \ell \text{ and } \forall x' d(X, x') \leq \ell$$

$$\Leftrightarrow (B1) \text{ and } (B2). \quad \square$$

We often use a special case of this:

COROLLARY B2. *Suppose there are surjections $f: Y \rightarrow X$, $f': Y \rightarrow X'$ such that $\forall y d(f(y), f'(y)) \leq \ell$. Then $d(X, X') \leq \ell$.*

Proof. Clear from lemma B1. \square

LEMMA B3. *If*

$$(B3) \quad \forall y \exists x \forall x' \exists y' [d(y, y') \leq d(x, x')]$$

$$(B4) \quad \forall y' \exists x' \forall x \exists y [d(y, y') \leq d(x, x')]$$

then $d(Y, Y') \leq d(X, X')$

Proof. (B3) implies, successively,

$$\forall y \exists x \forall x' d(y, Y') \leq d(x, x')$$

$$\forall y \exists x d(y, Y') \leq d(x, X')$$

$$\forall y d(y, Y') \leq d(X, X')$$

$$\sup_y d(y, Y') \leq d(X, X')$$

Similarly, (B4) implies $\sup_y d(Y, y') \leq d(X, X')$. The desired result now follows by taking the maximum of the lhs of the last 2 inequalities. \square

Actually, we only need lemma B3 in the special case of

COROLLARY B4. *Suppose there are surjections $f: X \rightarrow Y$ and $f': X' \rightarrow Y'$ such that $\forall x, x' [d(f(x), f'(x')) \leq d(x, x')]$. Then $d(Y, Y') \leq d(X, X')$.*

Proof. Clear from lemma B3. \square

LEMMA B5.

$$d(X \cup Y, Y' \cup Y') \leq \max(d(X, X'), d(Y, Y')).$$

Proof. $d(x, X' \cup Y') \leq d(x, X') \leq d(X, X') \leq \text{rhs.}$

Hence, $\sup_x d(x, X' \cup Y') \leq \text{rhs.}$

Similarly, $\sup_y d(Y, X' \cup Y') \leq \text{rhs}$

$$\sup_x d(X \cup Y, x') \leq \text{rhs}$$

$$\sup_y d(X \cup Y, y') \leq \text{rhs.}$$

Now take the maximum of the lhs of the last 4 lines. \square

From now on we consider uniform processes, solving equation (2.2).

(See definition 2.10.) We let x, y, \dots range over elements of $A \times P$, and define $\deg(\langle a, p \rangle) = \deg(p)$.

We give one more lemma.

LEMMA B6. *For finite p, p', q, q' :*

$$\text{if } d(q, q') \leq d(p, p')$$

$$\text{then } d(\langle a, q \rangle, \langle a', q' \rangle) \leq d(\langle a, p \rangle, \langle a', p' \rangle).$$

Proof. Clear. \square

THEOREM B7. For finite q, q' :

$$(B5) \quad d(p \circ q, p \circ q') \leq d(q, q')$$

Proof. We prove (B5) simultaneously with

$$(B6) \quad d(p \circ x, p \circ x') \leq d(x, x')$$

by induction on n , where $n = \max(\deg(q), \deg(q'))$ in the case of (B5), and $n = \max(\deg(x), \deg(x'))$ in the case of (B6),

If $q = p_0$ or $q' = p_0$ then (B5) is clear. Otherwise (cf. definition 2.14a)

$$\text{lhs of (B5)} = d(\{p \circ x \mid x \in q\}, \{p \circ x' \mid x' \in q'\}) \leq d(q, q')$$

by the induction hypothesis for (B6) and corollary B4 (taking $f(x) = p \circ x$ and $f'(x') = p \circ x'$). This proves (B5) for the given n . Now (B6) follows for the same n :

$$\begin{aligned} d(p \circ \langle a, q \rangle, p \circ \langle a', q' \rangle) &= d(\langle a, p \circ q \rangle, \langle a', p \circ q' \rangle) \\ &\leq d(\langle a, q \rangle, \langle a', q' \rangle) \end{aligned}$$

by (B5) and lemma B6. \square

COROLLARY B8. For finite q_n , if $\langle q_n \rangle_n$ is a CS, then so is $\langle p \circ q_n \rangle_n$.

Proof. Clear from theorem B7. \square

We observe that corollary B8 justifies the definition $p \circ q = \lim_n (p \circ q^{(n)})$.

COROLLARY B9. Theorem B7 holds for all q, q' .

Proof. For all n , $d(p \circ q^{(n)}, p \circ q'^{(n)}) \leq d(q^{(n)}, q'^{(n)})$, by theorem B7. Now $\text{lhs} \rightarrow d(p \circ q, p \circ q')$, $\text{rhs} \rightarrow d(q, q')$, and we see that $d(p \circ q, p \circ q') \leq d(q, q')$. \square

COROLLARY B10. Corollary B8 holds for all q_n .

Proof. Clear from corollary B9. \square

A more interesting consequence is (for *all* sequences $\langle q_n \rangle_n$):

COROLLARY B11. *If $q_n \rightarrow q$ then $p \circ q_n \rightarrow p \circ q$.*

Proof. $d(p \circ q_n, p \circ q) \leq d(q_n, q) \rightarrow 0$. \square

Note. Corollary B11 states that " \circ " is continuous in its second argument.

THEOREM B12. *For finite p, p', q, q' ,*

$$d(p \cup q, p' \cup q') \leq \max(d(p, p'), d(q, q')).$$

Proof. If any of p, p', q, q' equals p_0 , the result is clear. Otherwise it follows immediately from lemma B5. \square

Again, we have the corollaries

COROLLARY B13.

- a. *For finite p_n, q_n , if $\langle p_n \rangle_n, \langle q_n \rangle_n$ are CS then so is $\langle p_n \cup q_n \rangle_n$.
(This justifies the definition $p \cup q = \lim_n (p^{(n)} \cup q^{(n)})$.)*
- b. *Theorem B12 holds for all p, p', q, q' .*
- c. *Part a holds for all p_n, q_n .*
- d. *If $p_n \rightarrow p, q_n \rightarrow q$ then $p_n \cup q_n \rightarrow p \cup q$ (for all p, q).*
Thus, " \cup " is jointly continuous in both arguments.

Proof. We only prove

- b. For all n , $d(p^{(n)} \cup q^{(n)}, p'^{(n)} \cup q'^{(n)}) \leq \max(d(p^{(n)}, p'^{(n)}), d(q^{(n)}, q'^{(n)}))$.

Now let $n \rightarrow \infty$.

- d. $d(p_n \cup q_n, p \cup q) \leq \max(d(p_n, p), d(q_n, q)) \rightarrow 0$. \square

THEOREM B14. *For finite p, p', q, q' ,*

$$(B7) \quad d(p \parallel q, p' \parallel q') \leq \max(d(p, p'), d(q, q')).$$

Proof. We first prove a special case of (B7), namely with $q = q'$:

$$(B8) \quad d(p \parallel q, p' \parallel q) \leq d(p, p').$$

This is proved simultaneously with

$$(B9) \quad d(p \parallel y, p' \parallel y) \leq d(p, p')$$

$$(B10) \quad d(x \parallel q, x' \parallel q) \leq d(x, x')$$

by induction on n , where $n = \max(\deg(p), \deg(p')) + \deg(q)$ in (B8),
 $n = \max(\deg(p), \deg(p')) + \deg(y)$ in (B9), and
 $n = \max(\deg(x), \deg(x')) + \deg(q)$ in (B10).

Now if any of p, p', q equals p_0 , then (B8) is clear. Otherwise (cf. definition 2.14c):

$$\begin{aligned} \text{lhs of (B8)} &= d(\{p \parallel y \mid y \in q\} \cup \{x \parallel q \mid x \in p\}, \\ &\quad \{p' \parallel y \mid y \in q\} \cup \{x' \parallel q \mid x' \in p'\}) \\ &\leq \max(d_1, d_2) \end{aligned}$$

by lemma B5, where

$$\begin{aligned} d_1 &= d(\{p \parallel y \mid y \in q\}, \{p' \parallel y \mid y \in q\}) \\ d_2 &= d(\{x \parallel q \mid x \in p\}, \{x' \parallel q \mid x' \in p'\}). \end{aligned}$$

Now $d_1 \leq d(p, p')$ by the induction hypothesis for (B9) and corollary B2 (taking $f(y) = p \parallel y$, $f'(y) = p' \parallel y$ and $\ell = d(p, p')$). Also $d_2 \leq d(p, p')$ by the induction hypothesis for (B10) and corollary B4 (taking $f(x) = x \parallel q$ and $f'(x') = x' \parallel q$). This proves (B8) for the given n . Now (B9) and (B10) follow for the same n . For (B9):

$$\begin{aligned} &d(p \parallel \langle a, q \rangle, p' \parallel \langle a, q \rangle) \\ &= d(\langle a, p \parallel q \rangle, \langle a, p' \parallel q \rangle) \\ &= \frac{1}{2}d(p \parallel q, p' \parallel q) \\ &\leq \frac{1}{2}d(p, p') \quad \text{by (B7)} \\ &\leq d(p, p'), \end{aligned}$$

and for (B10):

$$\begin{aligned}
& d(\langle a, p \rangle \parallel q, \langle a', p' \rangle \parallel q) \\
&= d(\langle a, p \parallel q \rangle, \langle a', p' \parallel q \rangle) \\
&\leq d(\langle a, p \rangle, \langle a', p' \rangle)
\end{aligned}$$

by (B7) and lemma B6.

Thus we have proved (B8). Similarly (by a symmetrical argument) we can prove (for finite p, q, q'):

$$(B11) \quad d(p \parallel q, p \parallel q') \leq d(q, q').$$

Finally, from (B10) and (B11), and the strong triangle inequality (see the remark after lemma 2.8) we obtain

$$\begin{aligned}
d(p \parallel q, p' \parallel q') &\leq \max(d(p \parallel q, p' \parallel q), d(p' \parallel q, p' \parallel q')) \\
&\leq \max(d(p, p'), d(q, q')). \quad \square
\end{aligned}$$

As before, we have the corollaries

COROLLARY B15.

- a. For finite p_n, q_n , if $\langle p_n \rangle_n, \langle q_n \rangle_n$ are CS then so is $\langle p_n \parallel q_n \rangle_n$.
(This justifies the definition $p \parallel q = \lim_n (p^{(n)} \parallel q^{(n)})$.)
- b. Theorem B14 holds for all p, p', q, q' .
- c. Part a holds for all p_n, q_n .
- d. If $p_n \rightarrow p, q_n \rightarrow q$ then $p_n \parallel q_n \rightarrow p \parallel q$ (for all p, q).
Thus, " \parallel " is jointly continuous in both arguments.

Proof. Clear. \square

Now the properties of lemma 2.15 q , i.e., associativity of " \circ ", " \cup ", " \parallel ", commutativity of " \cup ", " \parallel ", are easily proved. E.g., for associativity of " \circ ", prove $(p \circ q) \circ r = p \circ (q \circ r)$ first for finite r by induction on $\deg(r)$, and then for all r by taking $r = \lim_n r^{(n)}$, and using corollary B11.

We conclude this appendix with a proof that " \circ " is jointly continuous in both arguments (as yet, we only proved continuity in its second argument).

THEOREM B16. For finite q ,

$$(B12) \quad d(p \circ q, p' \circ q) \leq d(p, p').$$

Proof. We prove (B12) simultaneously with

$$(B13) \quad d(p \circ y, p' \circ y) \leq d(p, p'),$$

by induction on $\deg(q)$ (in (B12)) and $\deg(y)$ (in (B13)). If $q = p_0$ then (B12) is clear. Otherwise

$$\begin{aligned} d(p \circ q, p' \circ q) &= d(\{p \circ y \mid y \in q\}, \{p' \circ y \mid y \in q\}) \\ &\leq d(p, p') \end{aligned}$$

by the induction hypothesis for (B13) and corollary B2. As for (B13):

$$\begin{aligned} &d(p \circ \langle a, q \rangle, p' \circ \langle a, q \rangle) \\ &= d(\langle a, p \circ q \rangle, \langle a, p' \circ q \rangle) \\ &= \tfrac{1}{2} d(p \circ q, p' \circ q) \\ &\leq \tfrac{1}{2} d(p, p') \quad \text{by (B12)} \\ &\leq d(p, p'). \quad \square \end{aligned}$$

Finally, we obtain the corollaries.

COROLLARY B17.

- a. Theorem B16 holds for all q .
- b. If $p_n \rightarrow p$ then $p_n \circ q \rightarrow p \circ q$
- c. If $p_n \rightarrow p$ and $q_n \rightarrow q$ then $p_n \circ q_n \rightarrow p \circ q$
(i.e., " \circ " is jointly continuous in both arguments).

Proof. We prove only part c. We have $d(p_n \circ q_n, p \circ q) \leq \max(d(p_n \circ q_n, p_n \circ q), d(p_n \circ q, p \circ q)) \leq \max(d(q_n, q), d(p_n, p)) \rightarrow 0$, by the strong triangle inequality and corollaries B9 and B17. \square

MC NR

35230

69 D13
69 F12
69 F32
69 F33

ONTVANGEN 14 SEP. 1982