stichting
mathematisch
centrum

MC

AFDELING INFORMATICA IW 212/82 NOVEMBER
(DEPARTMENT OF COMPUTER SCIENCE)

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

PROCESSES AND A FAIR SEMANTICS FOR THE ADA RENDEZ-VOUS

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.3.1, F.3.2, F.3.3.

Processes and a fair semantics for the ADA rendez-vous*)

by

J.W. de Bakker & J.I. Zucker**)

ABSTRACT

Processes are mathematical objects which are elements of domains in the sense of Scott and Plotkin. Process domains are obtained as solutions of equations solved by techniques from metric topology as advocated by Nivat. We discuss how such processes can be used to assign meanings to languages with concurrency, culminating in a definition of the ADA rendez-vous. An important intermediate step is a version of Hoare's CSP for which we describe a process semantics and which is used, following Gerth, as target for the translation of the ADA fragment. Furthermore, some ideas will be presented on a mathematically tractable treatment of fairness in the general framework of processes.

KEY WORDS & PHRASES: *Concurrency*, ADA *rendez-vous*, *fairness*, *denotational semantics*, *communicating processes*, *metric topology*

---

*) This report has been submitted for publication elsewhere.

**) Department of Computer Science, State University of New York, Buffalo, U.S.A.

1. INTRODUCTION

This paper presents a case study in the area of the semantics of concurrency. In the initial years of the theory of concurrency, most of the attention was devoted to notions such as composition and *synchronization* of parallel processes - often established through suitably restricted interleaving of the elementary actions of the components, and mostly referring to a shared variable model. More recently there has been a considerable increase in the interest for *communication* between processes - often referring to a model where the individual processes have disjoint variables which interact only through the respective communication mechanisms. Instrumental in this development have been the studies of BRINCH HANSEN [6], HOARE [10] and MILNER [15], where a variety of forms of communication was proposed and embedded in a language design or studied with the tools of operational and denotational semantics. The incorporation of the notions of tasking and rendez-vous in the language ADA ([1]) provides additional motivation for the study of communication, and it is the latter notion in particular which we have chosen as the topic of our investigation.

The main purpose of our paper is firstly to provide a rigorous definition for the ADA rendez-vous with the tools of *denotational* semantics, and secondly to introduce a mathematically tractable approach to fairness which is applicable in general in various situations where choices have to be made on a fair basis, and in particular to the ADA rendez-vous definition.

The general framework we apply in our paper was first outlined in DE BAKKER & ZUCKER [3], and later described in detail in DE BAKKER & ZUCKER [4]. In order to keep the present paper self-contained, we shall provide a summary description of the main points of the latter paper, without going into much mathematical detail. Our approach to the ADA rendez-vous and to fairness owes much to two contributions to ICALP 82. In GERTH [8] the idea of translating the ADA fragment to a version of CSP was proposed; the same approach will be applied by us in section 6. In PLOTKIN [19], the fundamental idea of specifying a fair merge through suitable use of - essentially - an appropriate sucession of random choices was proposed and embedded in a category - theoretic setting. (The suggestion of applying a version of such random choice in the framework of processes arose in a discussion with Plotkin during an IFIP WG 2.2 meeting.)

The structure of the paper is the following. After this introduction we present in section 2 an outline of the underlying semantic framework, though without most of the mathematics. In denotational semantics, language constructs are provided with mathematical objects (functions, operators, etc.) as their meanings. In the present paper, these meanings are so-called *processes* (in our paper a technical term for certain mathematical objects rather than for -syntactic- components of a program). Processes are elements of domains in the general sense as introduced by SCOTT [21,22]. Technically, domains of processes are obtained as solutions of *domain equations*. The solution of such equations in a context with nondeterminacy and concurrency was first studied in detail by PLOTKIN [18] (see [4] for more recent references). We have based our solution techniques on completion methods in metric topology (as advocated recently by Nivat and his school, see e.g [16]). In the appendix we summarize the topological definitions; the full story is given in [4]. Throughout our paper, we shall introduce a variety of processes, corresponding to a variety of programming concepts we encounter on the way to our understanding of the ADA rendez-vous. In section 2, processes are still simple. We call them *uniform*, and they bear a close resemblance to trees - though there are also a few crucial differences. Section 2 further introduces various operations upon processes - which will undergo successive refinements in later section. We moreover illustrate uniform processes by using them in the semantics of a very simple language with parallel merge as its only concurrent notion. In section 3 we use uniform processes as a vehicle to explain the key idea of our approach to fairness, viz. suitable alternation of random choices. (Ultimately, this idea may be traced back to the use of *oracles* to handle fairness. Fundamental studies of the semantics of fairness were made by PARK [17]; proof - theoretic investigations are described, e.g., in [2,11,12,20].) Section 4 describes a number of ways of providing processes with additional structure. Firstly, we enrich them with a synchronization mechanism in the sense of MILNER'S ports ([15]). We then obtain structures which are close to his synchronization trees. Next, we add a functional flavor to uniform processes, and obtain objects which have PLOTKIN'S resumptions ([18]) as forerunners. Finally, we add a communication feature to processes yielding a counterpart for Milner's communication trees ([15]). Whereas in section 4 we introduce each extension independently, we need their combination in section 5 to define the semantics of a language with both parallel merge, (synchronization through) communication, and a version of Milner's restriction operator. This language is an abstraction of HOARE's CSP ([10]), and we use it to provide a translation of the ADA fragment featuring its rendez-vouz concept ([1], chapter 9) in section 6. Section 7, finally, extends the fairness-definition ideas of section 3 to a situation with communication. We emphasize that the definitions in sections 6 and 7 concentrate solely on the concurrency and communication aspects of ADA - together with a few standard sequential concepts to render some verisimilitude to the ADA fragment. Accordingly, we omit all treatment of further ADA notions which, though interacting with the rendez-vous concept in the full language, would detract from the understanding of the central topic of our paper. References to *operational* approaches to the ADA rendez-vous include [9,13]. As final remark we add that the reader who is interested only in the ADA rendez-vous without fairness considerations may just skip sections 3 and 7.

2. UNIFORM PROCESSES AND A SIMPLE LANGUAGE WITH MERGE

A uniform process is a variation on the notion of tree. It is used, e.g., to assign meaning to a program when one is primarily interested in the structure of the sequences of elementary actions generated during its execution, rather than in the relation between input and output states of the program. Processes (and trees) constitute a more refined tool than just sets of sequences: we distinguish between the two objects

which have the same associated sets of sequences {ab,ac}. Also, uniform processes are only the first on a list of gradually more complex constructs to be studied in subsequent sections.

Let A be any alphabet; for the moment we do not care whether A is finite or infinite. Let a,b,... be elements of A. Uniform processes p,q,... will be described as certain constructs over the alphabet A.

We introduce

1. The *nil process* p_0. Roughly, its role is that of neutral element for various operations; also, it may be seen as label of the leaves of a process in case this is viewed as a tree-like construct.
2. The set of all *finite* processes $P_\omega \overset{\text{df.}}{=} \bigcup_n P_n$, where P_n, n = 0,1,..., are given by

$$P_0 = \{p_0\}$$

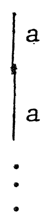
$$P_{n+1} = \mathcal{P}(A \times P_n)$$

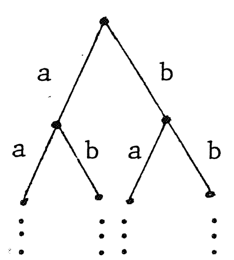
   and $\mathcal{P}\,(\cdot)$ denotes all subsets of $(\cdot)$

   Finite processes are for example $p_0, \{\langle a,p_0\rangle, \langle b,p_0\rangle\}$, or $\{\langle a,\{\langle b,p_0\rangle,\langle c,p_0\rangle\}\rangle\}$ and $\{\langle a,\{\langle b,p_0\rangle\}\rangle,\langle a,\{\langle c,p_0\rangle\}\rangle\}$. Note that these examples are elements of P_0, P_1, P_2 and P_2. Note also that the latter two processes correspond to the pictures at the beginning of the section.
3. The set of all finite or infinite processes (over A) as solution of the domain equation

$$P = \{p_0\} \cup \mathcal{P}_c(A \times P) \tag{2.1}$$

We shall not give the full explanation here, but restrict ourselves to the following (more - though not all - details appear in the Appendix): We may introduce a distance or *metric* d on the space P_ω of all finite processes, and consider the *completion* of P_ω with respect to this metric (cf. Cantor's completion of the set of rationals to that of the reals). Essentially, this amounts to adding to the space P_ω all limits of so-called Cauchy sequences (sequences $\langle p_n\rangle_{n=0}^{\infty}$ with $p_n \in P_n$, such that distances between elements get arbitrarily small with increasing index). E.g., infinite objects such as $\{\langle a,\{\langle a,\{\langle a,\ldots\rangle\}\rangle\}\rangle\}$ or $\{\langle a,\{\langle a,\ldots\rangle,\langle b,\ldots\rangle\}\rangle,\langle b,\{\langle a,\ldots\rangle,\langle b,\ldots\rangle\}\rangle\}$, corresponding to the pictures





belong to P. Furthermore, $P_c(\cdot)$ now stands for the collection of all *closed* - with respect to the metric - subsets of $(\cdot)$, and one can show that for P the completion of P_ω, it indeed satisfies equation (2.1). In summary, each proces is either finite and element of some P_n, or infinite and limit of a Cauchy sequence $\langle p_n \rangle_n$, with $p_n \in P_n$. Throughout the paper, we shall pay little attention to the infinite case, not because we want to ignore it but rather since, based on the firm foundation of (2.1) - or similar equations below, the infinite case - e.g. for the operations to be defined below - always follows straightforwardly from the finite case.

The reader should observe the difference between processes and trees. Firstly, in processes we have nor order. Trees $^a\!\bigwedge^b$ and $^b\!\bigwedge^a$ are different; as processes they are both equal to $\{\langle a,p_0\rangle,\langle b,p_0\rangle\}$. Secondly, processes have no multiple occurrences of elements. The trees $|^a$ and $^a\!\bigwedge^a$ are different, but as processes they coincide (non-nil processes are sets, not multisets).

We continue with the definition of the main *operations* on processes. Throughout the paper, we shall distinguish the cases of the nil process p_0, finite processes p,q,... which are *sets* $X,Y \in P(A\times P_n)$ for some n, and infinite processes $\lim_n p_n$, with $p_n \in P_n$. Observe that elements x,y of sets X,Y are *pairs* $\langle a,p'\rangle,\langle b,q'\rangle$, etc. We now define three important operations on processes.

<u>DEFINITION 2.1.</u>

a. *Composition* "$\circ$" is defined by

$p\circ p_0 = p$, $p\circ X = \{p\circ x \mid x \in X\}$, $p\circ\langle a,q\rangle = \langle a,p\circ q\rangle$,

$p\circ \lim_n q_n = \lim_n (p\circ q_n)$

b. *Union* "$\cup$" is defined by

$p \cup p_0 = p_0 \cup p = p$, and, for $p,q \neq p_0$, $p \cup q$ is the set - theoretic union of the sets p,q

c. *Merge* "$||$" is defined by

$p \,||\, p_0 = p_0 \,||\, p = p$, $\quad X \,||\, Y = (X \,||_L\, Y) \cup (X \,||_R\, Y)$,

$X \,||_L\, Y = \{x \,||\, Y \mid x \in X\}$, $X \,||_R\, Y = \{X \,||\, y \mid y \in Y\}$,

$\langle a,p\rangle \,||\, Y = \langle a,p \,||\, Y\rangle$, $\quad X \,||\, \langle b,q\rangle = \langle b,X \,||\, q\rangle$,

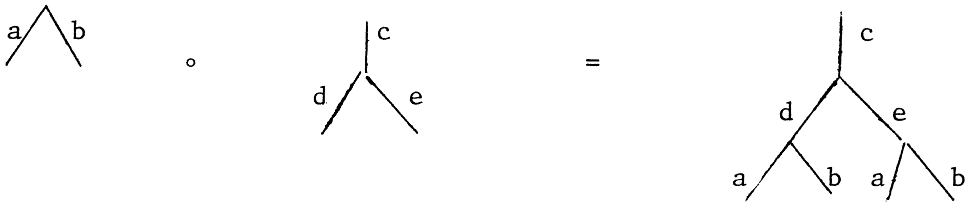
$(\lim_i p_i) \,||\, (\lim_j q_j) = \lim_k (p_k \,||\, q_k)$

<u>LEMMA 2.2</u>. *The above operations are all well-defined and associative,* $\cup$, $||$ *are commutative, and they all have the usual continuity properties (see Appendix for definition).*

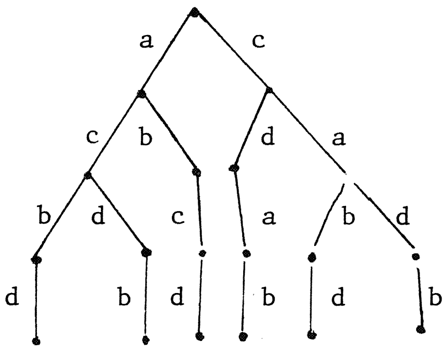
Proof. See [4]. □

Examples

1. Let $p = \{\langle a,p_0\rangle,\langle b,p_0\rangle\}$, $q = \{\langle c,\{\langle d,p_0\rangle,\langle e,p_0\rangle\}\rangle\}$.
   Then $p\circ q = \{\langle c,\{\langle d,p\rangle,\langle e,p\rangle\}\rangle\}$. In pictures we have



2. $\{\langle a,\{\langle b,p_0\rangle\}\rangle\} \parallel \{\langle c,\{\langle d,p_0\rangle\}\rangle\}$
   $=$



We next introduce the important notion of a so-called *contracting* mapping. A mapping $T: P\to P$ is called contracting whenever, for all p',p'', $d(T(p'),T(p'')) \le c\times d(p',p'')$, with $0 \le c < 1$. (The mapping T brings points closer to each other.) We have a classical theorem which will be very useful in the sequel:

THEOREM 2.3. *Let* T *be continuous and contracting, and let* q *be an arbitrary element of* P. *Then the sequence* $q, T(q), T^2(q),\ldots$ *is a Cauchy sequence which converges to the unique fixed point of* T.

Proof. This is Banach's fixed point theorem. □

Remarks

1. Observe that $\lim_n T^n(q)$ is independent of q.
2. Convergence of the sequence $q, T(q),\ldots$ in cases where T is not necessarily contracting is studied in [5].

We close this section with the introduction of a simple language with parallel composition (i.e. the merge "||") as its only non-sequential concept and we show how uniform processes can be used to define its semantics. More specifically, the simple language L_1 has elementary actions (for simplicity taken from the alphabet A), sequential composition, nondeterministic choice, merge, and (finite or infinite) iteration. In BNF like notation, its syntax is given in

<u>DEFINITION 2.4</u>. Statements $S \in L_1$ are defined by

$$S ::= a \mid \underline{skip} \mid S_1;S_2 \mid S_1 \cup S_2 \mid S_1 \,||\, S_2 \mid S^*$$

Remark. We ignore possible syntactic ambiguities.

Let τ be a special element added to A (Milner would call it the unobservable action), and let P_1 solve the equation

$$(2.2) \qquad P_1 = \{p_0\} \cup P_c((A \cup \{\tau\}) \times P_1).$$

We define the semantic mapping M: $L_1 \to P_1$ in

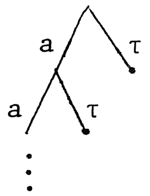
<u>DEFINITION 2.5</u>. The mapping M: $L_1 \to P_1$ is given by
$M(a) = \{<a,p_0>\}$, $M(\underline{skip}) = \{<\tau,p_0>\}$,
$M(S_1;S_2) = M(S_2) \circ M(S_1)$, $M(S_1 \cup S_2) = M(S_1) \cup M(S_2)$, $M(S_1 || S_2) = M(S_1) || M(S_2)$, and
$M(S^*) = \lim_i p_i$, where p_0 is the nil process and $p_{i+1} = (p_i \circ M(S)) \cup \{<\tau,p_0>\}$.

Remarks

1. We see that the syntactic operations ;, $\cup$, $||$ are mapped directly onto the semantic operations $\circ$, $\cup$, $||$.
2. The definition of S^* is explained by observing the intended equivalence $S^* = (S;S^*) \cup \underline{skip}$. Semantically, we have, by theorem 2.3, for $p \overset{df.}{=} \lim_i p_i$, the fixed point property $p = (p \circ M(S)) \cup \{<\tau,p_0>\}$.

Examples.

1. $M(a_1;a_2) = M(a_2) \circ M(a_1) = \{<a_2,p_0>\} \circ \{<a_1,p_0>\} = \{<a_1,\{<a_2,p_0>\}>\}$
2. $M(a_1;(a_2 \cup a_3)) = \{<a_1,\{<a_2,p_0>,<a_3,p_0>\}>\} \neq M((a_1;a_2) \cup (a_1;a_3))$
3. $M(a^*) = \{<\tau,p_0>,<a,\{<\tau,p_0>,<a,...>\}>\}$. Cf. the picture



(Note that, by closedness, we know that in this tree we have to "include" the infinite path a^ω!)

The reader should observe that L_1 could also be provided with a semantics in terms of sets of sequences rather than of processes. In this case $a_1;(a_2 \cup a_3)$ and $(a_1;a_2) \cup (a_1;a_3)$ - and also statements such as $(a \cup b) \| c$ and $(a\| c) \cup (b \| c)$ - would obtain the same meaning. In subsequent applications we shall be able to profit from the more refined process structure, which is why we already used them for providing meaning to L_1.

3. FAIRNESS FOR UNIFORM PROCESSES

We present a definition of fair merge for uniform processes which is based essentially on the well-known idea of implementing fair scheduling through systematic alternation of random choice (see [2] and, in particular, [19]). We first discuss the idea using a simple example (in which it is convenient to use sequences rather than processes). Consider the two infinite sequences of actions a^ω and b^ω, and suppose we want to write a program scheduling their fair merge $a\|_f b$ (which should therefore exclude sequences with almost all a's or almost all b's). Now this is achieved by the following program with random assignments - where $x := ?$ means that x is assigned an arbitrary non-negative integer:

$x_1 := ?;\ x_2 := ?;$

$L_1 : a;\ \underline{\text{if}}\ x_1 > 0\ \underline{\text{then}}\ x_1 := x_1 - 1;\ \underline{\text{goto}}\ L_1\ \underline{\text{else}}\ x_2 := ?;\ \underline{\text{goto}}\ L_2\ \underline{\text{fi}}$

$\cup$

$L_2 : b;\ \underline{\text{if}}\ x_2 > 0\ \underline{\text{then}}\ x_2 := x_2 - 1;\ \underline{\text{goto}}\ L_2\ \underline{\text{else}}\ x_1 := ?;\ \underline{\text{goto}}\ L_1\ \underline{\text{fi}}$

Observe that this program determines $a^\omega \|_f b^\omega$ as an infinite sequence of *either* subsequences of $x_1^{(i)}$ a's and then $x_2^{(i)}$ b's, $i = 1,2,\ldots$, $x_1^{(i)}$ and $x_2^{(i)}$ successive results of the random choices $x_1 := ?$ and $x_2 := ?$, *or* of a similar sequence of subsequences of $x_2^{(j)}$ b's and $x_1^{(j)}$ a's, $j = 1,2,\ldots$.

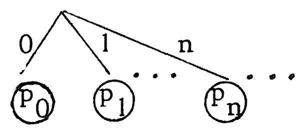
In PLOTKIN [19], this idea was embedded in the settting of category theory. What we shall describe here is the same approach in the framework of process theory. At first sight, the random assignment is an extraneous element for the process notion. However, there is a natural way to link it to the process framework. We start with the observation that the *infinite* union $\bigcup_n p_n$, for processes $p_n \in P$, is, in general, not well-defined (technically, this is the case because the infinite union of a family of closed sets is not necessarily closed). What we can do, however, is to extend P in the following way. Let $\mathbb{N}$ be the set of natural numbers. Now instead of using equation (2.1) we take process domain P_f as solution of

$$(3.1) \qquad P_f = \{p_0\} \cup \mathcal{P}_c((A \cup \mathbb{N}) \times P_f)$$

Within P_f we can define a new construct $\bigsqcup_n p_n$ by the definition

$$\bigsqcup_n p_n = \{<n,p_n> \mid n \in \mathbb{N}\}$$

(In this expression, p_0 is some arbitrary process rather than the nil process.) In a picture we have for $\bigsqcup_n p_n$:



which simulates a random choice between the p_n. It can be verified that $\bigsqcup_n p_n$ is a well-defined element of P_f (since by the definitions in the Appendix, the only non-trivial Cauchy sequences must be wholly within some p_n). We are now sufficiently prepared for

<u>DEFINITION 3.1</u> (fair merge). Let $p,q \in P_f$, and let, as usual, X,Y be finite processes. Let b range over $B \overset{df.}{=} A \cup \mathbb{N}$. We shall define $p \|_f q$ in terms of a number of auxiliary constructs $p \|_\alpha q$, for α any of the subscripts of $\|$ occurring in the clauses below.

a. $p \|_\alpha p_0 = p_0 \|_\alpha p = p$

b. $X \|_f Y = (X \|_L Y) \cup (X \|_R Y)$

c. $X \|_L Y = \{<n, X \|_{L,n} Y> \mid n \in \mathbb{N}\}$, and similarly for $X \|_R Y$

d. $X \|_{L,n} Y = \{x \|_{L,n} Y \mid x \in X\}$, and similarly for $X \|_{R,n} Y$

e. $<b,p> \|_{L,n+1} Y = <b, p \|_{L,n} Y>$, and symmetric

f. $<b,p> \|_{L,0} Y = <b, p \|_R Y>$, and symmetric

g. $(\lim_i p_i) \|_f (\lim_j q_j) = \lim_k (p_k \|_f q_k)$

LEMMA 3.2. *The above definition of* $\|_f$ *is well-formed (e.g., if* $<p_i>_i$ *and* $<q_j>_j$ *are Cauchy sequences, then so is* $<p_k \|_f q_k>_k$*, etc.)*

Proof. The proof is a reasonably straightforward extension of the results in Appendix B of [4], and omitted here. □

Remark. The reader who has understood the program in the beginning of this section will recognize that definition 3.1 is the exact counterpart of that program, with the random choice $x_i := ?$,i = 1,2, replaced by a choice $<n,\ldots>$ for some $n \in \mathbb{N}$.

We need additional study to link the notion of fair merge of two processes to that of a "fair process". The following definitions and property seem plausible here (though we have no full supporting proofs):

1. Let $p \in P_f$, and let b range over $B \stackrel{df.}{=} A \cup \mathbb{N}$. A *path* for p is a (finite or infinite) sequence

   $$(*) \qquad <b_1,p_1> \ , \ <b_2,p_2>,\ldots$$

   such that $<b_1,p_1> \in p$, and $<b_{i+1},p_{i+1}> \in p_i$, i = 1,2,... .
2. b is *enabled* in (*) whenever, for some i and q,$<b,q> \in p_i$.
   b *occurs* in (*) whenever, for some i, $b = b_i$.
3. A path (*) is *fair* with respect to some $B' \subseteq B$ whenever, for all $b' \in B'$, if b' is infinitely often enabled in (*), it infinitely often occurs in (*). Process p is called fair with respect to B' whenever all its paths are fair with respect to B'.
4. We conjecture that, for p,q fair with respect to A, $p \|_f q$ is fair with respect to A.

The above ideas can be modified for *regular* processes. Without going into details, let us call a process regular whenever it has only finitely many different subprocesses. We expect that results extending the above can be obtained for regular processes, where the above definitions are replaced by conditions imposed upon "moves" of pairs $<b,q>$ rather than simply of elementary actions b.

4. PROCESSES WITH ADDITIONAL STRUCTURE

In this section, we discuss three ways in which to extend the uniform processes of section 2. We shall deal with

- processes exhibiting synchronization
- processes which are (also) functions
- processes which communicate.

We begin with synchronization. (The ideas for this stem from MILNER's CCS [15]). Let Γ be a set of *ports*, the elements of which appear in pairs $\gamma,\bar{\gamma},\ldots$ (pairs are symmetric in the sense that $\bar{\bar{\gamma}} = \gamma$). We introduce processes with synchronization as elements of the set P_s which solves

$$(4.1) \qquad P_s = \{p_0\} \cup \mathcal{P}_c((A \cup \{\tau\} \cup \Gamma) \times P_s)$$

Let β range over $A \cup \{\tau\} \cup \Gamma$. We define the operations of section 2, together with the new operation of restriction $p\backslash\gamma$, in

DEFINITION 4.1

a. $p \circ p_0$, $p \circ X$, $p \circ \lim_n p_n$ are as before, and $p \circ \langle\beta,q\rangle = \langle\beta,p\circ q\rangle$.

b. $\cup$ is defined as before

c. $p \,||\, q$ is defined as before, except for the (central) clause

$X \,||\, Y = (X \,||_L Y) \cup (X \,||_R Y) \cup (X \,||_s Y)$, where $||_L$ and $||_R$ are as in def. 2.1, and

$X \,||_s\, Y = \{\langle\tau,p' \,||\, p''\rangle \mid \langle\gamma,p'\rangle \in X, \langle\bar{\gamma},p''\rangle \in Y$, for some pair of corresponding ports $\gamma,\bar{\gamma}\}$

d. $p\backslash\gamma$ is defined by: $p_0\backslash\gamma = p_0$, $(\lim_n p_n)\backslash\gamma = \lim_n (p_n\backslash\gamma)$, and

$$X \backslash \gamma = \{\langle\beta,p'\backslash\gamma\rangle \mid \langle\beta,p'\rangle \in X,\ \beta \neq \gamma,\bar{\gamma}\}$$

Remarks.

1. The definition of $p \,||\, q$ is the essential new element for synchronizing processes. Succesful synchronization of p,q results from pairs $\langle\gamma,p'\rangle$, $\langle\bar{\gamma},p''\rangle$ in p and q, respectively, and the outcome of composing these pairs yields an invisible τ, followed by $p' \,||\, p''$. $X \,||\, Y$ also includes the full merge $(X \,||_L Y) \cup (X \,||_R Y)$ as introduced in definition 2.1. Pairs $\langle\gamma,\ldots\rangle$ and $\langle\bar{\gamma},\ldots\rangle$ in this full merge can be removed by applying the $\backslash\gamma$ operation. (All this is extensively discussed in [15].)
2. In [4] we discuss how the "$\backslash\gamma$" operation can be defined to model deadlock.

In our view, this appears in situations where applying the "$\backslash\gamma$" operation would yield an empty set as outcome; in that case, the refined restriction operator yields a "dead process" as result. We omit further discussion of this.

LEMMA 4.2. *The operations* $\circ$, $\cup$, $||$, $\backslash$ *are well-defined, and satisfy (where relevant) the usual properties such as associativity, continuity etc.*

Proof. Omitted. □

We continue with the treatment of *functional* processes. Let A,B be two (arbitrary) sets. We take P_{fn} as solution of

$$P_{fn} = \{p_0\} \cup (A \to \mathcal{P}_c(B \times P_{fn})) \tag{4.2}$$

The various definitions of operations on P_{fn} are collected in the next definition (where we omit the standard cases when the operands are nil or infinite). We use the lambda - notation $\lambda a \cdot \ldots a \ldots$ for the function which maps a to ...a...

DEFINITION 4.3.

a. $p \circ q = \lambda a.(p \circ q(a))$, $p \circ X = \{p \circ x \mid x \in X\}$, $p \circ \langle b,q \rangle = \langle b, p \circ q \rangle$

b. $p \cup q = \lambda a.(p(a) \cup q(a))$

c. $p || q = \lambda a.((p || q(a)) \cup (p(a) || q))$

$X || q = \{x || q \mid x \in X\}$, $p || Y = \{p || y \mid y \in Y\}$

$\langle b,p \rangle || q = \langle b, p || q \rangle$, $p || \langle b,q \rangle = \langle b, p || q \rangle$

Remark. Note the (essential) difference between clauses b and c, in that $p || q$ is *not* defined as $\lambda a.\ (p(a) || q(a))$.

LEMMA 4.4. *The operations of definition* 4.3 *have the usual properties.*

Proof. Omitted. □

We conclude with the introduction of processes with *communication*. We take P_c as solution of

$$P_c = \{p_0\} \cup \mathcal{P}_c((B \times P_c) \cup (B \to P_c)) \tag{4.b}$$

Let π range over the set $B \to P_c$. The operations on P_c are given in

DEFINITION 4.5.

a. $p \circ X = \{p \circ x \mid x \in X\}$, $p \circ \langle b,q \rangle = \langle b, p \circ q \rangle$,

$p \circ \pi = \lambda b.\ (p \circ \pi(b))$

b. $\cup$ is as usual

c. $X \| Y = (X \|_L Y) \cup (X \|_R Y) \cup (X \|_c Y)$

$X \|_L Y$, $X \|_R Y$ are as usual. Moreover,

$$X \|_c Y = \{\pi(b) \| p' \mid \pi \in X, \langle b,p' \rangle \in Y\} \cup \{ p'' \| \pi(b) \mid \langle b,p'' \rangle \in X, \pi \in Y\}.$$

Remark. A process p may *communicate* with process q in case p contains some $\langle b,p' \rangle$, and q some function π (or vice versa). The process $\pi(b)$ is than used to continue the operation with the merge $\pi(b) \| p'$ (or symmetric). Applications of this idea (which we first saw in [14]) appear in the next section. The operations of definition 4.5 have the usual properties.

5. A CSP LIKE LANGUAGE

We introduce syntax and semantics of a CSP - like language (CSP for Hoare's Communicating Sequential Processes [10]). In the next section we shall use this language as target for the translation of the ADA fragment containing the rendez-vous construct. In the CSP - like language L_2 we articulate the elementary actions of L_1 to assignments and tests; in the next section we explain how tests are used in selection and while statements. L_2 moreover has the same program-forming operations as in section 2 ($;, \cup, \|, *$), and communication commands c?x and c!s. Here c is a *channel*, c?x means that variable x is to receive a value from channel c, and c!s means that the current value of expression s is to be transmitted over the channel c. The actual "hand-shake" communication over the channel c takes place provided (i) c?x and c!s appear as substatements in the statements S_1 and S_2 of some parallel composition $S \equiv S_1 \| S_2$, and (ii) in the execution of S, the flow of control in S_1 has arrived at c?x, and control in S_2 has arrived at c!s. The result of the communication is then equivalent to the assignment x:=s. Besides the communication commands we also have in L_2 a restriction S\c which enables us to delete unsuccesful attempts at communication from (the process which is the meaning of) S, and a special construct $b \Rightarrow S$ which in case test b is true will initiate execution of S without allowing a possible interleaving action from some parallel S' (which might change the value of b to false).

The precise definition of the syntax of L_2 is given in

DEFINITION 5.1.

a. Let x,y,... be variables in a set Var, s,t,... expressions, b,... boolean expressions, and c,... channels. (We omit specifying a syntax for boolean) expressions.)

b. Let $(S \in)$ L_2 be the class of statements defined by

$$S ::= x{:=}s \;|\; \underline{skip} \;|\; b \;|\; S_1;S_2 \;|\; S_1 \cup S_2 \;|\; S_1 \,\|\, S_2 \;|\; S^* \;|$$
$$c?x \;|\; c!s \;|\; S\backslash c \;|\; b \Rightarrow S$$

Remark. We hope no confusion will arise from our using $x \in Var$ in the syntax, and $x \in X$ in the semantics.

We next turn to the semantics of L_2. Let V be a set of *values* (meanings of variables and expressions), and let α range over V. Let $\Sigma = Var \to V$ be the set of *states*, with elements $\sigma \in \Sigma$, and let $\sigma\{\alpha/x\}$ be a state which is like σ, but for its value in x which equals α. Let V, W be functions which for each s, b and σ determine values $V(s)(\sigma) \in V$, and $W(b)(\sigma)$ in $\{tt,ff\}$ (the set of truth-values). We take P_2 as solution of

$$(5.1) \qquad P_2 = \{p_0\} \cup (\Sigma \to P_c((\Sigma \times P_2) \cup (\Gamma \times V \times \Sigma \times P_2) \cup (\Gamma \times (V \to (\Sigma \times P_2))))).$$

In this process domain equation we recognize elements of the three types of extensions discussed separately in section 4. Firstly, terms $\Gamma \times \ldots$ reflect synchronization ports ($\gamma \in \Gamma$ will correspond to channels c in the syntax). Secondly, the $\Sigma \to P_c(\cdot)$ term indicates that processes in P_2 are functional. Thirdly, terms $V \times (\Sigma \times P)$ combined with $V \to (\Sigma \times P)$ correspond to terms $B \times P$ together with $B \to P$ in the case of communicating processes treated previously. In fact, certain variations on equation (5.1) would also lead to a feasible semantics. However, we have chosen the present from since it provides the best model for our fairness considerations in section 7.

We use P_2 in the definition of the semantics for L_2:

<u>DEFINITION 5.2</u>. The mapping M: $L_2 \to P_2$ is defined by

a. $M(x{:=}s) = \lambda\sigma.\{<\sigma\{V(s)(\sigma)/x\},p_0>\}$

$M(\underline{skip}) = \lambda\sigma.\{<\sigma,p_0>\}$

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

b. $M(S_1;S_2) = M(S_2) \circ M(S_1)$

$M(S_1 \cup S_2) = M(S_1) \cup M(S_2)$

$M(S_1 || S_2) = M(S_1) \,||\, M(S_2)$, with $||$ to be defined in definition 5.3

$M(S^*) = \lim_i p_i$, with p_0 as always, and $p_{i+1} =$

$(p_i \circ M(S)) \cup \lambda\sigma.\{<\sigma,p_0>\}$

c. $M(c?x) = \lambda\sigma.\{<\gamma,\ \lambda\alpha.\ <\sigma\{\alpha/x\},p_0>>\}$

$M(c!s) = \lambda\sigma.\{<\bar{\gamma},\ V(s)(\sigma),\sigma,p_0>\}$

d. $M(S\backslash c) = M(S)\backslash\gamma$, with $\backslash$ to be defined in definition 5.3

$M(b \Rightarrow S) = \lambda\sigma.\ \underline{if}\ W(b)(\sigma)\ \underline{then}\ M(S)(\sigma)\ \underline{else}\ \emptyset\ \underline{fi}$

Remarks.

1. We use associativity of tupling, and identify constructs such as $<1,2,<3,4>>$, $<1,2,3,4>$, etc.
2. In part c, note that $M(c?x)(\sigma) \in \Gamma \times (V \to (\Sigma\times P))$, $M(c!s)(\sigma) \in \Gamma\times V\times\Sigma\times P$.
3. The definition of $M(b \Rightarrow S)$ should be contrasted with the result of $M(b;S)$: $M(b;S) = \lambda\sigma.\ \underline{if}\ W(b)(\sigma)\ \underline{then}\ \{<\sigma,M(S)>\}\ \underline{else}\ \emptyset\ \underline{fi}$. The reader should ponder the reasons why the latter semantics indeed allows what amounts to an interleaving action at the ";" in b;S, contrary to what is the case for $b \Rightarrow S$.

Definition 5.2 assumes the definition of $||$ and $\backslash$ in (omitting the nil and infinite cases as usual):

<u>DEFINITION 5.3</u>.

a. $p \,||\, q = \lambda\sigma.((p(\sigma) \,||\, q) \cup (p \,||\, q(\sigma)) \cup (p(\sigma) \,||_c\, q(\sigma)))$

$X \,||\, q = \{x \,||\, q \mid x \in X\}$, $\pi \,||\, q = \lambda\alpha.(\pi(\alpha) \,||\, q)$

$<\sigma,p> \,||\, q = <\sigma, p||q>$

$<\bar{\gamma},\alpha,\sigma,p> \,||\, q = <\bar{\gamma},\alpha,\sigma,p \,||\, q>$, $<\gamma,\pi> \,||\, q = <\gamma, \pi \,||\, q>$

(and, for the last three lines, the symmetric cases)

$X \,||_c\, Y = \{\pi(\alpha) \,||\, p' \mid <\gamma,\pi> \in X,\ <\bar{\gamma},\alpha,\sigma,p'> \in Y\} \cup$

$\{p'' \,||\, \pi(\alpha) \mid <\bar{\gamma},\alpha,\sigma,p''> \in X,\ <\gamma,\pi> \in Y\}$

b. $p \backslash \gamma = \lambda\sigma.\ (p(\sigma) \backslash \gamma)$, $\pi \backslash \gamma = \lambda\alpha.\ (\pi(\alpha) \backslash \gamma)$, $x \backslash \gamma = \{x\} \backslash \gamma$,

$X \backslash \gamma = \{<\sigma,p'\backslash\gamma> \mid <\sigma,p'> \in X\} \cup$

$\{<\gamma', \pi \backslash \gamma> \mid <\gamma', \pi> \in X,\ \gamma' \neq \gamma,\bar{\gamma}\} \cup$

$\{<\gamma',\alpha,\sigma,p'\backslash\gamma> \mid <\gamma',\alpha,\sigma,p'> \in X,\ \gamma' \neq \gamma,\bar{\gamma}\}$

Example. We evaluate $M((c?x \mid\mid c!1) \backslash c)$. We obtain

$M(c?x \mid\mid c!s) \overset{\text{df.}}{=} p =$

$\lambda\sigma.\{<\gamma, \lambda\alpha.<\sigma\{\alpha/x\},p_0>>\} \mid\mid \lambda\sigma.\{<\bar{\gamma}, 1, \sigma, p_0>\} =$

$\lambda\sigma.\{<\gamma,\ldots>,<\bar{\gamma},\ldots>, \lambda\alpha.(<\sigma\{\alpha/x\},p_0>)(1) \parallel p_0\} =$

$\lambda\sigma.\{<\gamma,\ldots>,<\bar{\gamma},\ldots>, \sigma\{1/x\},p_0 \parallel p_0>\}$.

Hence, $M(c?x \mid\mid c!1)\backslash c) = p\backslash\gamma = \lambda\sigma.\{<\sigma\{1/x\},p_0>\}$, which is, indeed, the same process as $M(x:=1)$.

Note that in the above definitions the role of σ in fourtuples $<\bar{\gamma},\alpha,\sigma,p>$ is in fact superfluous. However, we have included it to facilitate the definition of *path* in section 7.

6. THE ADA RENDEZ-VOUS

We consider an ADA fragment which centers around the notion of rendez-vous between (calls and accepts of) entries occurring in ADA tasks, and we exhibit a denotational semantics for the fragment by establishing a translation to L_2. We begin with the syntax:

<u>DEFINITION 6.1.</u>

a. (programs formed from tasks). Programs $S \in L_A$ are defined by

$S ::= \quad T_1 \parallel T_2 \parallel \ldots \parallel T_m$

b. (tasks). Tasks T are defined by

$$\begin{array}{ll} T ::= & x:=s \mid \underline{\text{skip}} \mid \underline{\text{if}}\ b\ \underline{\text{then}}\ T_1\ \underline{\text{else}}\ T_2\ \underline{\text{fi}} \mid \underline{\text{while}}\ b\ \underline{\text{do}}\ T\ \underline{\text{od}} \mid \\ & e(s,z) \mid T_1; T_2 \mid \\ & \underline{\text{accept}}\ e(x,y)\ \underline{\text{do}}\ T\ \underline{\text{end}} \mid \\ & \underline{\text{select}}\ \ b_1 \rightarrow \underline{\text{accept}}\ e_1(x_1,y_1)\ \underline{\text{do}}\ T_1'\ \underline{\text{end}};\ T_1'' \ \Box\ldots\Box \\ & \qquad\quad\ b_n \rightarrow \underline{\text{accept}}\ e_n(x_n,y_n)\ \underline{\text{do}}\ T_n'\ \underline{\text{end}};\ T_n'' \\ & \underline{\text{end}} \end{array}$$

Remarks.

1. e(s,z) is an entry call statement, with actual parameters s and z. Also, <u>accept</u> e(x,y) <u>do</u> T <u>end</u> is an entry accept statement. At the moment of a (succesful) rendez-vous, statement T is executed with actuals s and z corresponding to the formals x,y. The "hand-shake" communication follows the CSP principle. The select statement allows a nondeterministic choice between the guarded accept branches as listed.

2. To avoid problems of naming and scope, we assume a fixed number of distinct entry names $e_1,\dots,e_s$ occurring in the tasks $T_1,\dots,T_m$ of program S. Thus, we ignore the notion of entry declarations; neither do we deal with the selected component notation $T_i.e$.
3. In entry calls e(s,z) we encounter - for simplicity's sake - only two actual parameters, viz. expression s and variable z. We also ignore complications arising from parameter passing, and concentrate our interest on cases where the parameter mechanism is equivalent to call-by-value for s and to call-by-value-result - the definition of which is implied by the clauses in part c of definition 6.2 - for the parameter z.

We now present a translation from the statements (and tasks) in L_A to those in L_2. (The idea of such a translation is due to GERTH [8]; the main difference between our approach and [8] is that the latter paper ultimately considers an operational rather than a denotational semantics.) For $S \in L_A$ and tasks T we define their translation S°, $T^\circ \in L_2$. Compared to L_2 as introduced in section 5 we have, in fact, a few minor amendments. We use e rather than c for channels (to stick more closely to the convention for entries in L_A); moreover, we use a version of *simultaneous* restriction $S\backslash\{e_1,\dots,e_s\}$ with the obvious meaning. Furthermore, we introduce an error statement Δ to be used to indicate failure when all guards in a select statement have the value false. The meaning of Δ is given by $M(\Delta) = \lambda\sigma.\{<\delta,p_0>\}$, where δ is a special *dead* state (to be accompanied by natural definitions such as $M(x:=s)(\delta) = \{<\delta,p_0>\}$, etc.). The translation from L_A to L_2 is given in

DEFINITION 6.2.

a. $(x:=s)^\circ \equiv (x:=s)$, $\underline{\text{skip}}^\circ \equiv \underline{\text{skip}}$, $(T_1;T_2)^\circ \equiv (T_1^\circ;T_2^\circ)$

b. $(\underline{\text{if}}\ b\ \underline{\text{then}}\ T_1\ \underline{\text{else}}\ T_2\ \underline{\text{fi}})^\circ \equiv (b;T_1^\circ) \cup (\neg b;T_2^\circ)$

$(\underline{\text{while}}\ b\ \underline{\text{do}}\ T\ \underline{\text{od}})^\circ \equiv (b;T^\circ)^*;\ \neg b$

c. $e(s,z)^\circ \equiv e!s;\ e!z;\ e?z$

$(\underline{\text{accept}}\ e(x,y)\ \underline{\text{do}}\ T\ \underline{\text{end}})^\circ \equiv e?x;\ e?y;\ T^\circ;\ e!y$

$(\underline{\text{select}}\ \dots\ \underline{\text{end}})^\circ \equiv$

$$\bigcup_{i=1}^{n}(b_i \Rightarrow e_i?\,x_i;\ e_i?\,y_i;\ (T_i')^\circ\ ;\ e_i!\,y_i\,;(T_i'')^\circ)$$
$$\cup$$
$$(\neg b_1 \wedge\dots\wedge \neg b_n);\ \Delta$$

d. $S^\circ \equiv (T_1^\circ \parallel \dots \parallel T_m^\circ) \backslash \{e_1,\dots,e_s\}$

where $e_1,\dots,e_s$ are all names of entries appearing in the tasks $T_1,\dots T_m$.

The reader will be able to convince himself that, indeed, the translation results in elements of L_2. Since L_2 obtained a denotational semantics in section 5, we have now established a denotational semantics - situated in the process framework - for the ADA fragment as well. What remains to be done is to develop a *fair* semantics, and this we shall present in the next section.

7. A FAIR SEMANTICS FOR THE ADA RENDEZ-VOUS

This section brings the final result of the paper: a fair semantics for the ADA rendez-vous concept. Since the ADA reference manual does not mention the word fair, let us explain why we are interested in such a semantics. We distinguish two aspects concerning the proper execution of a number of ADA tasks. Firstly, following the argument from PNUELI & DE ROEVER [20], such execution should be what they call *just*, i.e., it should satisfy the requirement that every task which is continuously enabled from a certain point in the computation should *move* infinitely often in that computation. (For the notions "enabled" and "move" cf. our notions of "enabled" and "occur" described in section 3; refinements for the present context follow soon.) It is this justice property which is achieved by the fair merge schedule to be defined below. Basically, it is motivated by the idea that modelling simultaneous execution of a number of parallel processors by an interleaving of their constituent individual actions should imply that each process should contribute eventually each of its enabled moves to this interleaving. Secondly, the manual stipulates a scheduling which honours different calls for the same entry in their order of arrival. Now one of the benefits of our treatment is that this requirement is met automatically. The crucial property here is that interleavings of the elementary actions where the synchronization does not fit - which in an operational approach leads to extension of the queue of calls for the entry concerned - in the denotational approach disappear through the restriction operator; hence, no special measures to impose the right queuing discipline are in order.

We proceed with the definition proper of $T_1 \|_f T_2$ - which is all that remains for the fair semantics of the ADA rendez-vous. Firstly, we have to extend the process domain in a fashion similar to the construction in section 3: we add a suitable $\mathbb{N} \times (\ldots)$ term:

$$P_A = \{p_0\} \cup (\Sigma \to \mathcal{P}_c((\Sigma \times P_A) \cup (\Gamma \times V \times \Sigma \times P_A) \cup (\Gamma \times (V \to \Sigma \times P_A)) \cup \mathbb{N} \times ((\Sigma \times P_A) \cup (\Gamma \times V \times \Sigma \times P_A) \cup (\Gamma \times (V \to \Sigma \times P_A)))))$$

Next, we give the definition of $p \|_f q$ for $p,q \in P_A$.

DEFINITION 7.1. As before, we define $p \|_\beta q$, for β as encountered below, and we omit treatment of the nil and infinite cases.

a. $p \|_f q = (p \|_L q) \cup (p \|_R q)$

b. $p \|_L q = \lambda\sigma.((p(\sigma) \|_L q) \cup (p(\sigma) \|_f q(\sigma)))$

$p \|_{L,n} q = \lambda\sigma.((p(\sigma) \|_{L,n} q) \cup (p(\sigma) \|_f q(\sigma)))$

c. $X \|_L q = \{\langle n, x \|_{L,n} q\rangle \mid x \in X,\ n \in \mathbb{N}\}$

$X \|_{L,n} q = \{x \|_{L,n} q \mid x \in X\}$

d. $\langle\sigma,p\rangle \|_{L,n+1} q = \langle\sigma, p \|_{L,n} q\rangle, \langle\sigma,p\rangle \|_{L,0} q = \langle\sigma, p \|_R q\rangle$

$\langle\bar{\gamma},\alpha,\sigma,p\rangle \|_{L,n+1} q = \langle\bar{\gamma},\alpha,\sigma,p \|_{L,n} q\rangle, \langle\bar{\gamma},\alpha,\sigma,p \|_{L,0} q = \langle\bar{\gamma},\alpha,\sigma,p \|_R q\rangle$

e. $\langle\gamma,\pi\rangle \|_{L,n} q = \langle\gamma,\pi \|_{L,n} q\rangle$

$\langle m,x\rangle \|_{L,n} q = \langle m,x \|_{L,n} q\rangle$

$\pi \|_{L,n} q = \lambda\alpha.(\pi(\alpha) \|_{L,n} q)$

f. $X \|_f Y = \{\pi(\alpha) \|_f q \mid \langle\gamma,\pi\rangle \in X,\ \langle\bar{\gamma},\alpha,\sigma,q\rangle \in Y\} \cup$

$\{q \|_f \pi(\alpha) \mid \langle\bar{\gamma},\alpha,\sigma,q\rangle \in X,\ \langle\gamma,\pi\rangle \in Y\}$

$\langle\sigma,p\rangle \|_f q = \langle\sigma,p \|_f q\rangle$

(We omit symmetric clauses for $\|_R$ and $\|_{R,n}$.)

We see that the definition is based on the same L/R alternation of random choices, but now embedded in a more complex setting due to the increased complexity of P_A.

Next, we make some remarks on the question - again generalizing section 3 - as to whether fair merge preserves fair processes. The following definitions and properties seem plausible here:

1. Let $\sigma, \sigma' \in \Sigma$, $p,p' \in P_A$. We say that the relationship $\langle\sigma,p\rangle \to \langle\sigma',p'\rangle$ holds whenever one of the following four cases applies:
   (i) $\langle\sigma',p'\rangle \in p(\sigma)$
   (ii) $\langle\bar{\gamma},\alpha,\sigma',p'\rangle \in p(\sigma)$, for some $\bar{\gamma},\alpha$
   (iii) $\langle\gamma,\pi\rangle \in p(\sigma)$, and $\langle\sigma',p'\rangle \in \pi(\alpha)$, for some γ,π,α

(iv) $<n,x> \in p(\sigma)$ for some n, and $<\sigma',p'>$ can be derived from x according to (i) to (iii) above.

Note how clause (ii) is only meaningful due to the presence of σ' in the fourtuple on the left-hand side.

2. Let $\sigma \in \Sigma$, $p \in P_A$. A *path* for p and σ is a (finite or infinite) sequence
   (*) $<\sigma_1,p_1>,<\sigma_2,p_2>,\ldots$
   such that $<\sigma_1,p_1> = <\sigma,p>$, and $<\sigma_i,p_i> \to <\sigma_{i+1},p_{i+1}>$, $i = 1,2,\ldots$
3. Let $\phi \in \Sigma \to \Sigma$. We say that ϕ is *enabled* in (*) whenever there exist i,σ and p such that $<\sigma_i,p_i> \to <\sigma,p>$, and σ is $\phi(\sigma_i)$.
4. We call a path (*) *fair* with respect to ϕ whenever, if ϕ is infinitely often enabled in (*), it infinitely often occurs in (*). We say that p is fair with respect to a collection Φ of functions ϕ whenever, for all σ and $\phi \in \Phi$, all paths for σ and p are fair with respect to ϕ.
   Now we conjecture that
5. If p,q are fair with respect to Φ then so is $p \,||_f\, q$.
6. (The meaning of) each program S of the ADA fragment (with syntax as in definition 6.1) is fair with respect to the collection of functions Φ defined as follows: (i) Φ contains the identity function $\lambda\sigma.\sigma$ and the error function $\lambda\sigma.\delta$. (ii). For each x := s occurring in S, Φ contains the function $\lambda\sigma.\sigma\{V(s)(\sigma)/x\}$. (iii). For each (syntactically) matching pair e?y, e!t occurring in S,Φ contains the function $\lambda\sigma.\sigma\{V(t)(\sigma)/y\}$.

By way of final remark let us add that the fairnees notion appearing in ADA is only one out of a large number of variations on the theme of fairness. We have some ideas on how to apply techniques resembling those of sections 3 and 7 to, e.g., fair iteration in a framework of guarded commands ([2]) or fair communication as discussed in KUIPER & DE ROEVER ([11]). We hope to describe these techniques in a future publication.

REFERENCES

[1] ADA, *The Programming Language* ADA, Reference Manual, LNCS 106, Springer, 1981.

[2] APT, R.R. & E.R. OLDEROG, *Proof rules dealing with fairness*, Proc. Logic of Programs 1981 (D. Kozen,ed.), 1-9, LNCS 131, Springer, 1982.

[3] DE BAKKER, J.W. & J.I. ZUCKER, *Denotational semantics of concurrency*, Proc 14^{th} ACM Symp. on Theory of Computing, pp. 153-158, 1982.

[4] DE BAKKER, J.W. & J.I. ZUCKER, *Processes and the denotational semantics of concurrency*, Report IW 209/82, Mathematisch Centrum, 1982.

[5] BERGSTRA, J.A. & J.W. KLOP, *Fixed point semantics in process algebras*, Report IW 206/82, Mathematisch Centrum, 1982.

[6] BRINCH-HANSEN, P., *Distributed processes: a concurrent programming concept*, C. ACM 21 (1978), 934-941.

[7] DUGUNDJI, J., *Topology*, Allen & Bacon, 1966.

[8] GERTH, R., *A sound and complete Hoare-like axiomatization of the* ADA *rendez-vous*, Proc. 9^{th} ICALP (M.Nielsen & E.M. Schmidt,eds.), 252-264, LNCS 140, Springer, 1982.

[9] HENNESSY, M.& W.LI, *Translating a subset of Ada into* CCS, Proc. IFIP Working Conference on Formal Description of Programming Concepts II (D.Bjørner,ed.), North-Holland, to appear.

[10] HOARE, C.A.R., *Communication sequential processes*, C. ACM 21 (1978), 666-677.

[11] KUIPER, R. & W.P. DE ROEVER, *Fairness assumptions for* CSP *in a temporal logic framework*, Proc. IFIP Working Conference on Formal Description of Programming Concepts, II (D. Bjørner,ed.), North-Holland, to appear.

[12] LEHMANN, D. A.PNUELI & J. STAVI, *Impartiality, justice and fairness: the ethics of concurrent termination*, Proc. 8^{th} ICALP (S. Even & O. Kariv, eds.), 264-277, LNCS 115, Springer, 1981.

[13] LI, W., *An operational semantics of tasking and exception handling in* ADA, Proc. ACM ADA Tec and Tutorial Conference, to appear.

[14] MILNE, G. & R. MILNER, *Concurrent processes and their syntax*, J. ACM 26 (1979), 302-321.

[15] MILNER, R., *A Calculus for Communicating Systems*, LNCS 92, Springer, 1980.

[16] 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.

[17] PARK, D., *On the semantics of fair parallelism,* in Abstract Software Specifications (D. Bjørner,ed.), pp. 504-526, LNCS 86, Springer, 1980.

[18] PLOTKIN, G.D., *A power domain construction,* SIAM J. on Comp. 5 (1976), 452-487.

[19] PLOTKIN, G.D., *A power domain for countable nondeterminism,* Proc. 9th ICALP (M. Nielsen & E.M. Schmidt, eds.), 418-428, LNCS 140, Springer, 1982.

[20] PNUELI, A. & W.P. DE ROEVER, *Rendez-vous with* ADA, *a proof theoretical view*, Proc. ACM ADA Tec and Tutorial Conference, to appear.

[21] SCOTT, D.S., *Data types as lattices,* SIAM J. on Comp., 5 (1976), 522-587.

[22] SCOTT, D.S., *Domains for denotational semantics,* Proc. 9th ICALP (M. Nielsen & E.M. Schmidt, eds.), 577-613, LNCS 140, Springer, 1982.

APPENDIX

We list some definitions and theorems concerning the topological background of the processes introduced above. Proofs are omitted; they can all be found in [4]. We assume known the notions of metric space, Cauchy sequences and limits in a metric space, closed sets, and completion of a metric space, (see, e.g., DUGUNDJI [7]). We shall only be concerned with metrics with values in [0,1].

DEFINITION A.1. Let (M,d) be a metric space, and let $X,Y \subseteq M$.
We define
a. $d(x,Y) = \inf \{d(x,y) \mid y \in Y\}$
b. $d(X,Y) = \max \{\sup\{d(x,Y) \mid x \in X\}, \sup \{d(y,X) \mid y \in Y\})$

The distance between sets in definition A1 is the so-called Hausdorff distance. By convention, $\inf \phi = 1$, $\sup \phi = 0$.

LEMMA A2. *Let* (M,d) *be a metric space, and let* $P_c(M)$ *be the collection of all closed subsets of* M. *Then* $(P_c(M), \tilde{d})$ *is a metric space, for* $\tilde{d}$ *the Hausdorff distance on* $P_c(M)$.

THEOREM A3. (Hahn). *If* (M,d) *is a compete metric space, then so is* $(P_c(M),\tilde{d})$.

Now let A be any set, and p_0 some object not in A. We define

DEFINITION A4. $P_0 = \{p_0\}$, $P_{n+1} = \{p_0\} \cup P\,(A\times P_n)$. d_0 is defined by $d_0(p',p'') = 0$ for $p',p'' \in P_0$. d_{n+1} is defined by: $d_{n+1}(p_0,p) = d_{n+1}(p,p_0) = 1$ for $p \neq p_0$, $d_{n+1}(p_0,p_0) = 0$, and, for $p',p'' \neq p_0$, $d_{n+1}(p',p'')$ is the Hausdorff distance between the sets p',p'' (subsets of $A\times P_n$) induced by the distance between "points" $d_{n+1}(\langle a_1,p_1\rangle,\langle a_2,p_2\rangle)$ given by

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

LEMMA A5. (P_n,d_n) *is a metric space for each n.*

DEFINITION A6. Let $P_\omega = \bigcup_n P_n$, $d = \bigcup_n d_n$ (with the natural meaning for $\bigcup_n d_n$). Let (P,d) be the completion of (P_ω,d).

THEOREM A7. $P \cong \{p_0\} \cup P_c(A\times P)$
(*Here* $\cong$ *means isometry between* ℓhs *and* rhs.)

DEFINITION A8. A mapping $T: P \to P$ is called *continuous* if, for each Cauchy sequence $\langle p_i \rangle_i$, we have that $\langle T(p_i) \rangle_i$ is again a Cauchy sequence, and $T(\lim_i p_i) = \lim_i T(p_i)$. Similarly we define continuity in $n \geq 1$ arguments.

LEMMA A9. *The operations* $\circ$, $\cup$, $\|$ *are continuous in both arguments.*

The above definitions and results can be extended in a natural way to processes with additional structure. Take, e.g., the case of process domain equation (4.2). We take $P_0 = \{p_0\}$, $P_{n+1} = \{p_0\} \cup (A \to \mathcal{P}(B \times P_n))$, and define $d_{n+1}(p', p'')$, for $p', p'' \neq p_0$, by $d_{n+1}(p', p'') = \sup_{a \in A} d_{n+1}(p'(a), p''(a))$, where $p'(a), p''(a)$ are sets to which the Hausdorff distance definition applies.

36466

69D41
69D42
69D43