Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

J.W. de Bakker

Designing concurrency semantics

Computer Science/Department of Software Technology Report CS-R8902 March

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Research (N.W.O.).

Designing Concurrency Semantics

J.W. de Bakker
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands
Department of Mathematics and Computer Science
Free University of Amsterdam, the Netherlands

Based on our experience in the design of semantics for several families of parallel languages, we present an expository account of a selection of four topics in concurrency semantics. These illustrate both foundational issues such as the connection with infinitary formal language theory and domain theory, and problems arising from the modelling of notions such as process creation and rendez-vous.

1980 Mathematics Subject Classification: 68Q55, 68N15.
1987 CR Categories: F.3.2, F.3.3, D.3.3.
Key Words & Phrases: metric semantics, infinitary format languages, synchronous step concurrency, process creation, rendez-vous, resumptions.
Note: The research of the author was partially supported by ESPRIT project 415, Parallel Architectures and Languages for AIP.
Note: This report contains the text of an invited address to be presented at IFIP Congress '89.

1. Introduction

The design of semantic models for parallel programming languages poses various problems which are not (so much) encountered in the study of sequential languages. Our aim in this lecture is to discuss a sample of these problems: we have selected four topics which illustrate some of the notions and techniques of concurrency semantics. Partly, these are of a foundational or comparative nature, partly they stem from our experience with the semantic modelling of 'real-life' languages. The latter owes much to our involvement in ESPRIT's project 415: Parallel Architectures and Languages (cf. [BNT1,2, B2]).

The first theme concerns the modelling of possibly infinite behaviour using tools from (infinitary) formal language theory. We investigate the rather natural question: what happens when one extends formal language theory with (a version of) parallel composition. Our main (and fairly recent) result may be phrased either as a natural generalization of an old theorem of SCHUTZENBERGER ([Sc], in turn generalized in [Ni1]), or, if one prefers, as an equivalence result for the operational and denotational semantics for such a language.

The second part exemplifies the central role of domain theory in concurrency semantics. We introduce two kinds of distinctions: *linear time* versus *branching time*, and *interleaving* versus *synchronous step* concurrency. The induced four types of concurrency are embedded into four languages, and each of these is provided with operational and denotational semantics. Thanks to the use of a somewhat advanced definitional machinery, we are able to obtain a unified proof of the equivalence of the two models in all four cases.

The next section deals with some problems of a different nature: we discuss how to design semantic models for two interesting notions from present-day concurrent languages, viz. *process creation* and *rendez-vous*. As always in this lecture, we illustrate the problem by focusing on a stripped-down version of the phenomenon at hand, referring for a discussion of the full version to work appearing elsewhere.

The three topics mentioned so far are all characterized by the property that they are expressed in terms of uninterpreted or schematic elementary actions (atoms are just symbols). In the final section,

Report CS-R8902
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

we discuss the complications arising when articulating the elementary actions. It is no longer clear how to satisfy the requirement that denotational models should be defined compositionally. The solution we present is based on Plotkin's resumption model.

In several papers of our group (references in the concluding section) we have used a combination of the concepts and tools to be outlined below in the design of semantics for a range of parallel languages, including imperative, dataflow, object-oriented and logic programming styles.

Acknowledgement. I am grateful to all colleagues in Amsterdam (at the Centre for Mathematics and Computer Science and at the Free University) and elsewhere (notably in Eindhoven, Kiel and Buffalo) with whom I have cooperated over the years. I am indebted to Jan Rutten for discussions concerning the selection of topics for this lecture.

2. Notation and mathematical preliminaries

The phrase 'let $(x \in)X$ be...' introduces a set X with variable x ranging over X. The notation $x(\in X) ::= 1|2| \cdots$ introduces the syntactic class X with elements x specified by the *BNF*-syntax $1|2| \cdots$. $\mathcal{P}(X)$ or $\mathcal{P}_\pi(X)$ denotes the set of all subsets (all subsets with property π) of X. For $f:X \rightarrow Y$ a function, $f[y/x]$ denotes the function which equals f in arguments $\neq x$, and which satisfies $f[y/x](x) = y$. For $f:X \rightarrow X$, any $x \in X$ such that $f(x) = x$ is called a fixpoint of f. In case f has precisely one fixpoint, we denote it by $fix(f)$.

We assume as known the notions of (complete) metric space (M,d), of closed or compact subsets of M, and of isometry between metric spaces. We also assume as known how to construct composed spaces (product, disjoint union, function space, all closed subsets of...) from (a) given metric space(s) (cf. [AR]). A function $f:(M_1,d_1) \rightarrow (M_2,d_2)$ is called *contracting* if, for some $\alpha \in [0,1)$ and all $x,y \in M_1$, we have $d_2(f(x),f(y)) \leqslant \alpha \cdot d_1(x,y)$. According to Banach's theorem, contracting functions on complete metric spaces have unique fixpoints, a property which we exhaustively exploit below.

3. Languages with infinite words

Infinite or perpetual processes frequently occur in (theory and practice of) distributed, in particular embedded, systems. Rather than concentrating on input / output behaviour, one focuses here on (observable abstractions from) the interactions between a reactive ([Pn]) system and its environment. Such behaviour is reflected in various mathematical models incorporating some notion of (possibly infinite) *history*. For many purposes, it is sufficient to take as a starting point the framework of (infinitary) formal language theory. In fact, concurrency semantics has profited extensively from the machinery of formal language theory (see, e.g., several chapters in [BRR 1,2]), and we shall be able to touch only briefly upon these connections.

Our starting point is the theory of context free languages with infinite words. We rephrase a theorem of Nivat [Ni1] (which itself extends a classical theorem of Schutzenberger [Sc]) as stating an equivalence result for operational and denotational semantics of a simple (sequential) language. Next, we indicate how this theorem is preserved under the addition of parallel composition (in the form of interleaving or *shuffle*) as fundamental operator.

3.1. Languages with infinite words: the context free case

Let $(a \in)A$ be a finite alphabet, and let $(x \in)Var$ be a set of *variables*. Let $(u,v,w \in)A^\infty = A^* \cup A^\omega$ be the set of all finite (A^*) or infinite (A^ω) words over A. We introduce a *metric* d on A^∞ as follows: For each u and integer $n \geqslant 0, u(n)$ denotes the prefix of u of length n, if this exists; otherwise, $u(n) = u$. We put $d(u,v) = 2^{-k}$, where $k = sup\{n : u(n) = v(n)\}$ (and $2^{-\infty} = 0$). Let $(X,Y \in)\mathbb{S} = \mathcal{P}_{nc}(A^\infty)$ be the set of all nonempty closed subsets of A^∞.

The language $(s \in)L_{cf}$ - *cf* for context free - has the syntax given in

Definition 3.1.

a. $s(\in L_{cf}) ::= a | x | s_1;s_2 | s_1+s_2$

b. *guarded* statements $g(\in L_{cf}^{g}) ::= a \mid g;s \mid g_1 + g_2$

c A declaration D is a finite set of pairs $<x,g>$, and a program t is of the form $<D,s>$.

Remark. Programming terminology differs here from formal language terminology. A construct $t = <D,s>$ is (isomorphic with) a context free grammar in Greibach normal form, and the meaning of s with respect to $D(\mathcal{O}_D[\![s]\!]$ or $\mathcal{D}_D[\![s]\!]$, to be introduced in a moment) yields a context free language in language theoretic terminology.

Conventions

1. We do not bother about syntactic ambiguities. The reader may add parentheses, if desired.
2. All variables occurring in a program $t = <\{<x_i,g_i>\}_{i=1}^{n},s>$ (i.e., in the g_i and in s) are among $x_1,...,x_n$.
3. Usually, we suppress explicit mentioning of D, and write 'the statement s' rather than 'the program $<D,s>$', etc.

We proceed with the definition of the *operational semantics*, i.e. of the mapping $\mathcal{O}_D:L_{cf} \to \mathbb{S}$. Let $E\,(E \notin L_{cf})$ be a special symbol denoting the 'terminated statement'. We first introduce a *transition system* T_{cf} (in the style of Plotkin's SOS [HP,Pl2,Pl3]), with transitions $s \xrightarrow{a}_D s'$ or $s \xrightarrow{a}_D E$. (We shall use $1 \to 2 \mid 3$ as shorthand for $1 \to 2$ and $1 \to 3$.)

DEFINITION 3.2 (T_{cf}). Assume some fixed D.

1. $a \xrightarrow{a}_D E$
2. If $g \xrightarrow{a}_D s \mid E$ then $x \xrightarrow{a}_D s \mid E$, for $<x,g>$ in D
3. If $s \xrightarrow{a}_D s' \mid E$ then $s;\bar{s} \xrightarrow{a}_D s';\bar{s} \mid \bar{s}$,$s+\bar{s} \xrightarrow{a}_D s' \mid E$, $\bar{s}+s \xrightarrow{a}_D s' \mid E$
   (both in 2 and 3, the choices are to be made consistently)
4. $w \in \mathcal{O}_D$ $[\![s]\!]$ if either
   $w = a_1a_2...a_n$ and $s \xrightarrow{a_1}_D s_1 \to \cdots \xrightarrow{a_{n-1}}_D s_{n-1} \xrightarrow{a_n}_D E$, or
   $w = a_1a_2 \cdots$, and $s \xrightarrow{a_1}_D s_1 \xrightarrow{a_2}_D s_2 \to \cdots$

>From, e.g., [Ni2] we know that $\mathcal{O}_D$ $[\![s]\!]$ is indeed closed (even compact).

The denotational definition $\mathcal{D}$ uses an auxiliary argument from the set $(\gamma\in)\Gamma = Var \to_{fin} \mathbb{S}$, where a finite mapping $\gamma:x_i \to X_i, i = 1,...,n$ is abbreviated to $<X_i/x_i>_i$, or even to $<\cdot>$. We have as definition of $\mathcal{D}:L_{cf} \to \Gamma \to \mathbb{S}$ and of $\mathcal{D}_D:L_{cf} \to \mathbb{S}$:

DEFINITION 3.3. $\mathcal{D}[\![a]\!]<\cdot> = \{a\}$, $\mathcal{D}[\![x]\!]<X_i/x_i>_i = X_j$, where $x = x_j$, $\mathcal{D}[\![s_1;s_2]\!]<\cdot> = \mathcal{D}[\![s_1]\!]<\cdot>.\mathcal{D}[\![s_2]\!]<\cdot>$ (. is concatenation), $\mathcal{D}[\![s_1+s_2]\!]<\cdot> = \mathcal{D}[\![s_1]\!]<\cdot> \cup \mathcal{D}[\![s_2]\!]<\cdot>$, and $\mathcal{D}_D[\![s]\!] = \mathcal{D}[\![s]\!]<Y_i/x_i>_i$, where $<Y_1,...,Y_n> = fix(<\Phi_1, \ldots, \Phi_n>)$, with $\Phi_j = \lambda Y'_1 \cdot ... \lambda Y'_n.\mathcal{D}[\![g_j]\!]<Y'_i/x_i>_i, j=1,...,n$ (and $D = \{<x_i,g_i>_i\}$).

Remarks.

1. The fixpoints exist due to the contractivity of the Φ_j; this is in turn a consequence of the guardedness requirement on the g_j (cf., e.g., [BM2]).
2. Nivat considers greatest fixpoints in the domain $\mathcal{P}_c(A^\infty)$ which does include the empty set. For our purposes, this coincides with unique fixpoints in the domain $\mathcal{P}_{nc}(A^\infty)$, equipped with the Hausdorff metric (cf. [Ni2]) induced by the metric d on A^∞.

A central theorem for L_{cf} is

THEOREM 3.4. *For each* $<D,s>$, $\mathcal{O}_D[\![s]\!] = \mathcal{D}_D[\![s]\!]$.

The proof of this theorem (in [Ni1]) requires substantial language theoretic manipulations. We shall introduce some more advanced tools, facilitating the definition of $\mathcal{O}_D$ and $\mathcal{D}_D$, and the proof of

$\mathcal{O}_D = \mathcal{D}_D$. In fact, the tools be introduced in a moment pervade all semantic equivalence proofs to be discussed below. The key idea (from [KR]) is to use higher-order (contracting) mappings which have semantic operators or even semantic meaning functions such as $\mathcal{O}_D$ or $\mathcal{D}_D$ as (unique) fixpoint.

>From now on, we shall systematically omit the subscript D on $\mathcal{O}$ or $\mathcal{D}$, on $\xrightarrow{a}$, etc.: all considerations have to be supplemented, where relevant, by reference to some given D. By way of further simplification, we often drop parentheses around arguments of functions.

Let $(F\in)M = L_{cf}\rightarrow\mathbb{S}$, and let $\Psi: M\rightarrow M$ be defined in

DEFINITION 3.5. $\Psi Fs = \bigcup\{a\cdot Fs' : s\xrightarrow{a}s'\}\cup\{a : s\xrightarrow{a}E\}$

LEMMA 3.6. $\mathcal{O} = fix(\Psi)$.

Moreover, we define $\Phi: M\rightarrow M$ in

DEFINITION 3.7. $\Phi Fa = \{a\}, \Phi Fx = \Phi Fg$ (with $<x,g>$ in D), $\Phi F(s_1;s_2) = \Phi Fs_1\cdot Fs_2$, $\Phi F(s_1+s_2) = \Phi Fs_1\cup\Phi Fs_2$.

Using induction on the complexity of, first g, then any s, one may show that Φ is well-defined for all (F and) s. Moreover, we have that $\Phi\mathcal{D} = \mathcal{D}$. Finally, one may show (cf. [KR,BM2]) that $\Psi\mathcal{D} = \mathcal{D}$, hence yielding $\mathcal{D} = \mathcal{O}$ by the contractivity of Ψ. Thus, an alternative (and shorter) proof of theorem 3.4 is obtained.

3.2. Languages with infinite words: adding parallel composition

We extend L_{cf} by adding the parallel composition operator ($\|$), an operator yielding the interleaving or shuffle of the elementary actions making up its operands. To simplify our considerations, we omit discussion of synchronization features (which may be added without undue effort to the framework to be developed, cf. [BMOZ] or [KR]). For finite u,v we assume the definition of $u\|v$ as known (e.g. $ab\|c = \{abc,acb,cab\}$). If at least one of u,v is infinite, one may take as definition (cf. [Me]) $u\|v = \{w\in A^\omega \mid \exists w'[w\in h^{-1}(w')\wedge(h_1(w')\leqslant u)\wedge(h_2(w')\leqslant v)]\}$, where $\leqslant$ is the prefix order, and the homomorphisms h,h_1,h_2 are defined by $h_1(\overline{a}) = h_2(\overline{\overline{a}}) = a$, $h_1(\overline{\overline{a}}) = h_2(\overline{a}) = \epsilon, h(\overline{a}) = h(\overline{\overline{a}}) = a$. (Note that our $\|$ is not fair.) An alternative definition which is advantageous when variations on parallel composition are considered (cf. Section 4) is the following approach, again exploiting the idea of a higher-order mapping: Let $(\phi\in)\mathcal{T} = \mathbb{S}\times\mathbb{S}\rightarrow\mathbb{S}$, and let the mappings $\mathcal{K}_\circ, \mathcal{K}_\| : \mathcal{T}\rightarrow\mathcal{T}$ be given by

$$\mathcal{K}_\circ(\phi)(X)(Y) = \bigcup\{\tilde{\phi}(u)(v) : u\in X,\ v\in Y\}$$

$$\tilde{\phi}(\epsilon)(v) = \{v\},\ \tilde{\phi}(a.u)(v) = a.\phi(\{u\})(\{v\})$$

$$\mathcal{K}_\|(\phi)(X)(Y) = \mathcal{K}_\circ(\phi)(X)(Y)\cup\mathcal{K}_\circ(\phi)(Y)(X).$$

We put $\circ = fix(\mathcal{K}_\circ)$, $\| = fix(\mathcal{K}_\|)$, $\lfloor\!\lfloor = \mathcal{K}_\circ(\|)$.

Now that we have defined $\|$ (and the auxiliary $\lfloor\!\lfloor$), let us extend L_{cf} to L_{sh}:

DEFINITION 3.8. $s(\in L_{sh}) ::= a\mid x\mid s_1;s_2\mid s_1+s_2\mid s_1\|s_2, g(\in L_{sh}^g) ::= a\mid g;s\mid g_1+g_2\mid g_1\|g_2$ (and the induced extensions to D and $t = <D,s>$).

The operational semantics $\mathcal{O}: L_{sh}\rightarrow\mathbb{S}$ is given in

DEFINITION 3.9.

1. Add to T_{cf} the rule
   - if $s_1\xrightarrow{a}s_2\mid E$, then $s\|s_1\xrightarrow{a}s\|s_2\mid s$ and $s_1\|s\xrightarrow{a}s_2\|s\mid s$

2. Define Ψ as before (def.3.5), now with respect to T_{sh}.
3. Put $\mathcal{O} = \mathit{fix}(\Psi)$

The denotational definition is even more simply extended: Add to definition 3.7 the clause $\Phi F(s_1 \| s_2) = \Phi F s_1 \| \Phi F s_2$. We have

THEOREM 3.10. *For each* $s \in L_{sh}$, $\mathcal{O}[\![s]\!] = \mathcal{D}[\![s]\!]$.

Theorem 3.10 is in fact a quite recent result. Announced in [BKMOZ], in [BMOZ] it is obtained by an intricate analysis, applied in particular to the behaviour of recursive constructs and to the derivation of $\mathcal{O}[\![s_1 \| s_2]\!] = \mathcal{O}[\![s_1]\!] \| \mathcal{O}[\![s_2]\!]$ (for $\|$ involving synchronization). A substantial simplification was obtained in [KR] for a language with nested recursion (with μ-constructs) and in [BM2] for a language with simultaneous procedure declarations (as our D). Denotational variations may be induced by modifying the underlying framework. In [BM1], the metric model is compared with a model with cpo's, where the order is the Smyth order on sets of streams (cf. [Me]). In [BMO], the Smyth order is in turn compared to a model with (sets of) finite observations, a model representative of a class of models due to the Oxford school (cf. [OH]). Finally, in [BBKM] the simple order of reverse set inclusion is used, among others to compare what is called *linear time* semantics for L_{sh} with *branching time* semantics, a notion discussed further in the next section.

4. FOUR DOMAINS FOR CONCURRENCY

Whereas Section 3 addresses issues where language theory and concurrency semantics encounter each other, we now focus on some topics situated at the interface of concurrency semantics and domain theory. We shall vary our language L_{sh} in two ways. First, we shall replace the (normal) sequential composition (;) by the, not so customary, commit operator (:, an operator from parallel logic programming) which influences the failure (or deadlock) behaviour of a program. Secondly, we shall replace the interleaving semantics by a modest approximation to the notion of 'true concurrency': we shall replace '$\|$' by '$\times$', and assign it the meaning of *synchronous step* concurrency (cf. [TV]; our semantics was inspired by [MV]). Combining the two variations introduces *four* concurrent languages, and we shall introduce four corresponding domains (replacing $\mathbb{S}$ from Section 3) the elements of which are used as meanings for the statements in these four languages. More specifically, we introduce the languages $L_{li}, L_{bi}, L_{ls}, L_{bs}$ (*l*-linear, *b*-branching, *i*-interleaving, *s*-step). In each language, a primitive syntactic construct **fail** is included which we employ to bring out the differences between the linear and branching cases.

DEFINITION 4.1.

a. $s(\in L_{li}) ::= a \mid x \mid \mathbf{fail} \mid s_1;s_2 \mid s_1+s_2 \mid s_1 \| s_2$
b. $s(\in L_{bi}) ::= a \mid x \mid \mathbf{fail} \mid s_1:s_2 \mid s_1+s_2 \mid s_1 \| s_2$
c. $s(\in L_{ls}) ::= a \mid x \mid \mathbf{fail} \mid s_1;s_2 \mid s_1+s_2 \mid s_1 \times s_2$
d. $s(\in L_{bs}) ::= a \mid x \mid \mathbf{fail} \mid s_1:s_2 \mid s_1+s_2 \mid s_1 \times s_2$

In concurrency semantics, often a distinction is made as well between internal / external or local / global choice (e.g. in [BMOZ]). Since this has been investigated extensively elsewhere, we feel free to ignore the distinction here (a modest version appears in Section 5.2).

We assume the naturally induced definitions of $L^g_{li}, \ldots,$ and of declarations and programs. The crucial clauses for g are $g ::= g;s$ and $g ::= g:s$ for the linear and branching cases, respectively. All other composed constructs pass the guardedness requirement on to both components.

We now discuss the four domains used to assign semantics to the four languages. The domains are obtained as solutions to *domain equations* as introduced by SCOTT ([Gi] is a comprehensive reference) or PLOTKIN ([Pl1]). We have introduced our own metric version of the techniques to solve such equations, described in [BZ1] and essentially generalized in [AR]. The domain equations to be discussed are in fact isometries between complete metric spaces, but lack of (another kind of) space does not permit our treating any of the mathematical details here. There is one point too important to be omitted: all domains include - thanks to the underlying mathematical machinery - besides the finite objects specified by the structure of the equation also the corresponding *infinite* objects. For example, in equation *(li)* below we have that Q_{li} contains all elements a in A and all elements in $A \times Q_{li}$, i.e., all $a \in A$, all finite sequences $<a_1, <a_2, ..., <a_n, a>...>>$ *and* all infinite sequences $<a_1, <a_2, ..., <a_n, ...>...>>$. We now exhibit the equations for $P_{li}, P_{bi}, P_{ls}, P_{bs}$, each expressed with the aid of an auxiliary equation for a corresponding $Q_{..}$. In the linear case we also include a special element δ for failure, and in the step semantics we encounter $(\alpha \in)\mathcal{P}_n(A)$, the set of nonempty subsets of A. Of course, we have to provide evidence that the domains to be defined serve their purpose. We shall do this by designing operational and denotational semantics employing the four domains, and by establishing (without proof) the expected equivalence results.

(li): $Q_{li} = A \cup (A \times Q_{li}) \cup \{\delta\}$, $P_{li} = \mathcal{P}_{nc}(Q_{li})$

(bi): $Q_{bi} = A \cup (A \times P_{bi})$, $P_{bi} = \mathcal{P}_c(Q_{bi})$

(ls): $Q_{ls} = \mathcal{P}_n(A) \cup (\mathcal{P}_n(A) \times Q_{ls}) \cup \{\delta\}, P_{ls} = \mathcal{P}_{nc}(Q_{ls})$

(bs): $Q_{bs} = \mathcal{P}_n(A) \cup (\mathcal{P}_n(A) \times P_{bs}), P_{bs} = \mathcal{P}_c(Q_{bs})$

We present several examples of elements in the respective domains, each also obtained as meaning (denoted, for the moment, by $[\![\cdot]\!]$) of some s:

(li): $[\![a \| b]\!] = \{<a,b>, <b,a>\}$, $[\![a;(b+c)]\!] = [\![(a;b)+(a;c)]\!] = \{<a,b>, <a,c>\}$

(bi): $[\![a:(b+c)]\!] = \{<a, \{b,c\}>\}$, $[\![(a:b)+(a:c)]\!] = \{<a,\{b\}>, <a,\{c\}>\}$

(ls): $[\![(a \times b);(c \times d)]\!] = \{<\{a,b\},\{c,d\}>\}$

(bs): $[\![(a \times b):(c+d)]\!] = \{<\{a,b\},\{\{c\},\{d\}\}>\}$

We proceed with the semantic definitions proper.

DEFINITION 4.2 (operational semantics).
(li). T_{li} is as T_{sh} (thus, no axiom or rule is introduced for **fail**)
(bi). T_{bi} is as T_{sh}, with ':' replacing ';'
(ls). T_{ls} is as T_{sh}, with the rule for $\|$ replaced by

· if $s_1 \xrightarrow{\alpha_1} s', s_2 \xrightarrow{\alpha_2} s''$ then $s_1 \times s_2 \xrightarrow{\alpha_1 \cup \alpha_2} s' \times s''$

(and three simpler rules in case s' or s'' equals E)

(bs). T_{bs} is as T_{ls}, with ':' replacing ';'.

Before indicating how $\mathcal{O}_{..}$ is obtained from $T_{..}$, we first define the auxiliary reduction operator $red: P_{li} \rightarrow P_{li}$ (or $P_{ls} \rightarrow P_{ls}$) which inductively applies the simplification $\{\delta\} \cup p = p$, wherever possible. We use the auxiliary notation $p_a = \{q: <a,q> \in p\}$ and $p_\alpha = \{q: <\alpha,q> \in p\}$. Note that p_a or p_α may be empty.

DEFINITION 4.3 (reduction).
(li). $red(\{\delta\}) = \{\delta\}$. For $p \neq \{\delta\}$

$red(p) = \{a : a \in p\} \cup \{<a,q'> : p_a \neq \emptyset$ and $q' \in red(p_a)\}$

(ls). as *(li)*, with α replacing a.

Remark. Use of higher-order functions may make this definition rigorous.

We next define the various $\mathcal{O}_{..}$:

DEFINITION 4.4. Let $(F \in) M_{li} = L_{li} \rightarrow P_{li}$, and similarly for the other indices. We define $\Psi_{..}: M_{..} \rightarrow M_{..}$ in

(li). $\Psi_{li} Fs = red(\{<a,q'> \mid s \xrightarrow{a} s'$ and $q' \in Fs'\} \cup \{a \mid s \xrightarrow{a} E\})$, if the argument of $red \neq \emptyset$

$= \{\delta\}$, otherwise (with $\rightarrow$ according to T_{li})

(bi). $\Psi_{bi}Fs = \{<a,Fs'>|s\xrightarrow{a}s'\} \cup \{a|s\xrightarrow{a}E\}$ ($\rightarrow$ from T_{bi})

(ls). Ψ_{ls} is as Ψ_{li}, with α replacing a

(bs). Ψ_{bs} is as Ψ_{bi}, with α replacing a

Moreover, $\mathcal{O}_{li} = fix(\Psi_{li})$, and similarly for the other indices.

Next, we define the operators which are used in the denotational semantics. In all (relevant) cases, '$\cup$' denotes set-theoretic union, and $\cdot + \cdot = red(\cdot \cup \cdot)$. We now define '$\circ$', '$:$', '$||$', and '$\times$' in

DEFINITION 4.5.

(li). Let $(\phi \in) R_{li} = P_{li} \times P_{li} \rightarrow P_{li}$. We define $\Phi_{\circ}, \Phi_{||}: R_{li} \rightarrow R_{li}$ by

$$\Phi_{\circ}(\phi)(p_1)(p_2) = red(\bigcup \{\tilde{\phi}(q_1)(q_2): q_1 \in p_1, q_2 \in p_2\})$$

$$\tilde{\phi}(a)(q) = \{<a,q>\}, \tilde{\phi}(\delta)(q) = \{\delta\}$$

$$\tilde{\phi}(<a,q_1>)(q_2) = \{<a,\bar{q}>: \bar{q} \in \phi(\{q_1\})(\{q_2\})$$

$$\Phi_{||}(\phi)(p_1)(p_2) = \Phi_{\circ}(\phi)(p_1)(p_2) + \Phi_{\circ}(\phi)(p_2)(p_1)$$

(bi). Let $\phi, \Phi_{:}, \Phi_{||}$ have the appropriate types. We put

$$\Phi_{:}(\phi)(p_1)(p_2) = \{\tilde{\phi}(q_1)(p_2): q_1 \in p_1\}$$

$$\tilde{\phi}(a)(p) = <a,p>, \tilde{\phi}(<a,p_1>)(p_2) = <a, \phi(p_1)(p_2)>$$

$$\Phi_{||}(\phi)(p_1)(p_2) = \Phi_{:}(\phi)(p_1)(p_2) \cup \Phi_{:}(\phi)(p_2)(p_1)$$

(ls). $\Phi_{\circ}$ is as $\Phi_{\circ}$ for *(li)*, with α replacing a. Next, we define $\Phi_{\times}$:

$$\Phi_{\times}(\phi)(p_1)(p_2) = red(\bigcup \{\tilde{\phi}(q_1)(q_2): q_1 \in p_1, q_2 \in p_2\})$$

$$\tilde{\phi}(\alpha_1)(\alpha_2) = \{\alpha_1 \cup \alpha_2\}, \tilde{\phi}(\alpha)(\delta) = \tilde{\phi}(\delta)(\alpha) = \{\delta\}$$

$$\tilde{\phi}(\alpha_1)(<\alpha_2,q>) = \{<\alpha_1 \cup \alpha_2, q>\}, \text{ and symmetric}$$

$$\tilde{\phi}(<\alpha_1,q_1>)(<\alpha_2,q_2>) = \{<\alpha_1 \cup \alpha_2, q>: q \in \phi(\{q_1\})(\{q_2\})\}$$

(bs). $\Phi_{:}$ is as $\Phi_{:}$ for *(bi)*, with α replacing a. Next, we define $\Phi_{\times}$:

$$\Phi_{\times}(\phi)(p_1)(p_2) = \{\tilde{\phi}(q_1)(q_2): q_1 \in p_1, q_2 \in p_2\}$$

$$\tilde{\phi}(\alpha_1)(\alpha_2) = \alpha_1 \cup \alpha_2,\ \tilde{\phi}(<\alpha_1,p_1>)(\alpha_2) = <\alpha_1 \cup \alpha_2, p_1> \text{ and symmetric}$$

$$\tilde{\phi}(<\alpha_1,p_1>)(<\alpha_2,p_2>) = <\alpha_1 \cup \alpha_2, \phi(p_1)(p_2)>$$

Finally, we put $\circ = fix(\Phi_{\circ})$, etc.

The definitions of $\mathcal{D}$ for L_{li},... are now straightforward. For $s \equiv$ **fail**, we define $\mathcal{D}[\![\mathbf{fail}]\!] = \{\delta\}$ for the linear time, and $\mathcal{D}[\![\mathbf{fail}]\!] = \varnothing$ for the branching time case. All other definitions are as expected. E.g., following the strategy as in definition 3.7, we put $\Phi F(s_1:s_2) = \Phi Fs_1 : Fs_2$, etc. Recursion is again dealt with in the clause $\Phi Fx = \Phi Fg$, for $<x,g>$ in D. Thanks to our preparatory work, in particular the unified style in treating the four different cases, it is now not difficult to prove the

THEOREM 4.6. *For each* $s \in L_j, j \in \{li, bi, ls, bs\}$, we have $\mathcal{O}[\![s]\!] = \mathcal{D}[\![s]\!]$.

5. PROCESS CREATION AND RENDEZ-VOUS

After having devoted two sections to foundational questions in concurrency semantics, we now spend two sections on application - oriented topics. In the present section we discuss how to model two programming concepts which are important in 'real-life' concurrent languages, viz. process creation and rendez-vous. In the next section we turn to the treatment of so-called nonuniform phenomena: the elementary actions are no longer left schematic but are interpreted as assignments, send and receive actions and the like. Additional machinery is then necessary to develop the semantic definitions.

In the present section, everything remains uninterpreted or 'uniform'.

5.1. Process creation

We return to the semantic universe of $\mathbb{S} = \mathcal{P}_{nc}(A^{\infty})$, and we introduce L_{pc}, a small but essential modification of L_{sh} which incorporates the interesting notion of process creation. (This section presents a fragment of [AB,BM2], papers in turn inspired by our research on the semantics of POOL, Philips' parallel object-oriented language. POOL is defined in [A]; see [ABKR1,2,R1] for its semantics.) The syntax for L_{pc} is presented in

DEFINITION 5.1.

a. $s(\in L_{pc}) ::= a \mid x \mid s_1;s_2 \mid s_1+s_2 \mid \mathbf{new}(s)$

b. $g(\in L_{pc}^{g}) ::= h \mid g_1+g_2 \mid \mathbf{new}(g)$
   $h(\in L_{pc}^{h}) ::= a \mid h;s \mid h_1+h_2$

c. Declarations with pairs $<x,g>$ and programs for L_{pc} are as usual.

Remarks 1. The guardedness requirement now involves the auxiliary h. This is necessary since we do not want a construct such as $\mathbf{new}(a)$; x to qualify as guarded (since $\mathbf{new}(a);x$ is to have the same effect as the L_{sh}-statement $a \| x$).

2. Note that '$\|$' has disappeared from the syntax.

Before providing the formal semantic definitions, we first present an informal explanation of process creation. The execution of some $s \in L_{pc}$ is described in terms of a dynamically growing number of processes which execute statements in parallel in the following manner:

1. Set an auxiliary variable i to 1 and set s_1 to s, the statement to be executed. A process, numbered 1, is created to execute s.
2. Processes 1 to i execute in parallel. Process j executes s_j $(1 \leqslant j \leqslant i)$ in the usual way in case s_j does not begin with some $\mathbf{new}(s')$ statement.
3. If some process j $(1 \leqslant j \leqslant i)$ has to execute a statement of the form $\mathbf{new}(s')$, then the variable i is set to $i+1$, s_i is set to s', and a new process with number i is created to execute s_i. Process j will continue to execute the part after the $\mathbf{new}(s')$ statement. Go to step 2.
4. Execution terminates if all processes have terminated their execution.

We proceed with the formal definitions. We use a transition system T_{pc} expressed in terms of constructs $(r \in)Seq$ and $(\rho \in)Par$ defined as $r ::= E \mid s;r$ and $\rho ::= r \mid \rho, r \mid r, \rho$. Transitions are now constructs of the form $\rho \xrightarrow{a} \rho'$. The transition relation is specified in

DEFINITION 5.2.

a. $a;r \xrightarrow{a} r$

b. if $g;r \xrightarrow{a} \rho$ then $x;r \xrightarrow{a} \rho$, with $<x,g>$ in D
   if $s_1;(s_2;r) \xrightarrow{a} \rho$ then $(s_1;s_2);r \xrightarrow{a} \rho$
   if $s;r \xrightarrow{a} \rho$ then $(s+\bar{s});r \xrightarrow{a} \rho$ and $(\bar{s}+s);r \xrightarrow{a} \rho$
   if $r,(s;E) \xrightarrow{a} \rho$ then $\mathbf{new}(s);r \xrightarrow{a} \rho$
   if $\rho_1 \xrightarrow{a} \rho_2$ then $r,\rho_1 \xrightarrow{a} r,\rho_2$ and $\rho_1,r \xrightarrow{a} \rho_2,r$

c. Let $(F \in)M = Par \to \mathbb{S}$, and let $\Psi: M \to M$ be defined by

$$\Psi F \rho = \epsilon, \text{ if } \rho = E,E,\ldots,E$$
$$= \bigcup \{a.F(\rho') \mid \rho \xrightarrow{a} \rho'\}, \text{ otherwise}$$

d. Let $\mathcal{O} = fix(\Psi)$.

The denotational semantics for L_{pc} employs the familiar tool of *continuations*. Let $(X\in)Cont \stackrel{df.}{=} \mathbb{S}$. Let $N = Seq \to Cont \to \mathbb{S}$. We define $\Phi: N \to N$ in

DEFINITION 5.3. $\Phi FaX = a.X, \Phi FxX = \Phi FgX$, with $<x,g>$ in D, $\Phi F(s_1;s_2)X = \Phi Fs_1(Fs_2X)$, $\Phi F(s_1+s_2)X = (\Phi Fs_1X)\cup(\Phi Fs_2X)$, and the essential clause

$$\Phi F\ \mathbf{new}(s)X = (\Phi Fs\{\epsilon\})\|X$$

Let $\mathcal{D} = fix(\Phi)$.

Remark. Though '$\|$' is not in the syntax for L_{pc}, we assume it available as semantic operator: $\mathbb{S}\times\mathbb{S}\to\mathbb{S}$ (cf. Section 3).

We can now prove

THEOREM 5.4.

a. *Let* $\mathcal{E}: Par \to \mathbb{S}$ *be given by* $\mathcal{E}[\![E]\!] = \{\epsilon\}, \mathcal{E}[\![s;r]\!] = \mathcal{D}[\![s]\!]\mathcal{E}[\![r]\!], \mathcal{E}[\![r,\rho]\!] = \mathcal{E}[\![r]\!]\|\mathcal{E}[\![\rho]\!]$, *and symmetric. We then have* $\Psi(\mathcal{E}) = \mathcal{E}$.

b. *Putting* $\mathcal{O}[\![s]\!] \stackrel{df.}{=} \mathcal{O}[\![s;E]\!]$, *we have, for all* $s \in L_{pc}$, $\mathcal{O}[\![s]\!] = \mathcal{D}[\![s]\!]\{\epsilon\}$.

Proof. See [BM2] for part a. Part b is direct from part a. □

Remark. L_{pc} and L_{sh} are incomparable: it has been proved by [AA] that there exists $s \in L_{sh}$ such that for no $s' \in L_{pc}$ we have $\mathcal{O}[\![s]\!] = \mathcal{O}[\![s']\!]$, and vice versa.

3.2. Rendez-vous

Rendez-vous is a concept of concurrent languages such as ADA or POOL. We shall be concerned with a primitive version of this notion, were it only since we take it in a uniform setting, without facing problems such as parameter passing and the like. Stripped down to its essentials, rendez-vous is an extension of synchronization or communication such as, e.g., in CCS[Mi] or ACP[BeK]. In CCS, synchronized execution of the actions c and $\bar{c}$ delivers $c|\bar{c} = \tau$. In ACP, synchronized execution (or communication) of a and b delivers $a|b = c$. The situation we shall deal with involves synchronized execution of some $m?$ and $m!$ (m for *method*, as in POOL), which then results in the execution of the associated (guarded) statement $(m?|m!=)g_m$. The information which g_m belongs to m is contained in the (extended) set of declarations.

As syntax for L_{rv} we use

DEFINITION 5.5. Let $(m\in)\mathcal{M}$ be the set of method names, let $M? = \{m?: m\in\mathcal{M}\}$, $M! = \{m!: m\in\mathcal{M}\}, M = M?\cup M!$, and let $(e\in)E = A\cup M$.

1. $s(\in L_{rv}) ::= e\,|\,x\,|\,s_1;s_2\,|\,s_1+s_2\,|\,s_1\|s_2$
2. $g(\in L_{rv}^g) ::= a\,|\,g;s\,|\,g_1+g_2\,|\,g_1\|g_2$
3. A declaration D consists of finite sets of pairs $<x,g>$ and $<m,g>$. Programs are as usual.

Next, we introduce the semantic domains. For $\mathcal{O}$, we use $\mathbb{S}_\delta$ defined as follows: Let $A_\delta^\infty = A^*\cup A^\omega\cup A^*.\delta$. Then $\mathbb{S}_\delta = \mathcal{P}_{nc}(A_\delta^\infty)$. For $\mathcal{D}$ we shall introduce a new domain P_{rv} in a moment. We already announce that no simple equivalence $\mathcal{O} = \mathcal{D}$ will hold. Rather, an abstraction mapping $abs: P_{rv}\to\mathbb{S}_\delta$ will be defined, and we shall assert that $\mathcal{O} = abs\circ\mathcal{D}$.

The operational definitions are based upon the transition system T_{rv} which resembles T_{sh}, though there are also important differences.

DEFINITION 5.6 ($T_{rv}, \mathcal{O}$ for L_{rv}).

a. $e \xrightarrow{e} E$

- if $g \xrightarrow{e} s \mid E$ then $x \xrightarrow{e} s \mid E$, with $<x,g>$ in D
- if $s \xrightarrow{e} s' \mid E$ then $s;\bar{s} \xrightarrow{e} s';\bar{s} \mid \bar{s}$, $s \| \bar{s} \xrightarrow{e} s';\bar{s} \mid \bar{s}$, and $\bar{s} \| s \xrightarrow{e} \bar{s} \| s' \mid \bar{s}$
- if $s \xrightarrow{a} s' \mid E$ then $\bar{s} + s \xrightarrow{a} s' \mid E$ and $s + \bar{s} \xrightarrow{a} s' \mid E$
- if $s_1 \xrightarrow{m?} s', s_2 \xrightarrow{m!} s''$, $<m,g>$ in D, and $g \xrightarrow{c} \bar{s}$, then $s_1 \| s_2 \xrightarrow{c} \bar{s};(s' \| s'')$

(and a number of simpler cases if E replaces s', s'' or $\bar{s}$).

b. Let $(F \in) M = L_{rv} \to \mathbb{S}_\delta$, and let $\Psi: M \to M$ be defined by

$$\Psi FS = \bigcup \{a.Fs' \mid s \xrightarrow{a} s'\} \cup \{a \mid s \xrightarrow{a} E\} \text{ if this set is nonempty}$$
$$= \{\delta\}, \text{ otherwise.}$$

Let $\mathcal{O}: L_{rv} \to \mathbb{S}_\delta$ be defined by $\mathcal{O} = fix(\Psi)$.

Remark. Note that $\mathcal{O}[\![a;(b+m?)]\!] = \{ab\} \neq \{ab, a\delta\} = \mathcal{O}[\![(a;b)+(a;m?)]\!]$. This is a consequence of the rule for '+' and the fact that only $\xrightarrow{a}$-steps contribute to $\mathcal{O}$.

The denotational domain P_{rv} is, for convenience, defined in terms of a set $(\sigma \in) Step$ consisting of atoms a or $m?$, or pairs $<m!,p>$, where p will be used to store the meaning of g_m (with $<m,g_m>$ in D). We put

$$Step = A \cup M? \cup (M! \times P_{rv})$$
$$P_{rv} = \mathcal{P}_c(Step \cup (Step \times P_{rv}))$$

The definition of $\mathcal{D}$ follows the usual pattern. New are the clauses $\Phi Fm? = \{m?\}$, $\Phi Fm! = \{<m!, Fg_m>\}$ ($<m,g_m>$ in D). Also, the definitions of $\Phi F(s_1;s_2)$ and $\Phi F(s_1 \| s_2)$ require the operators '$\circ$', '$\|$': $P_{rv} \times P_{rv} \to P_{rv}$. The essential clauses (which may be embedded in a complete definition using the techniques of Section 3) are the following:

$$\sigma \circ p = <\sigma,p>, <\sigma,p_1> \circ p_2 = <\sigma, p_1 \circ p_2>$$
$$p_1 \| p_2 = (p_1 \mathbin{\|\!\_} p_2) \cup (p_2 \mathbin{\|\!\_} p_1) \cup (p_1 \mid p_2), p_1 \mathbin{\|\!\_} p_2 = \{q \mathbin{\|\!\_} p_2 : q \in p_1\}$$
$$\sigma \mathbin{\|\!\_} p = <\sigma,p>, <\sigma,p_1> \mathbin{\|\!\_} p_2 = <\sigma, p_1 \| p_2>$$

and the crucial rules for the rendez-vous

$$p_1 \mid p_2 = \bigcup \{q_1 \mid q_2 : q_1 \in p_1, q_2 \in p_2\}$$
$$<m?,p_1> \mid <<m!,p>,p_2> = p \circ (p_1 \| p_2),$$

(and three simpler rules if p_1 or p_2 is missing)

and $q_1 \mid q_2 = \varnothing$ for q_1, q_2 not of one of the above forms.

We conclude with the formulation of the relationship between $\mathcal{O}$ and $\mathcal{D}$. Let $(\phi \in) L = (P_{rv} \to \mathbb{S}_\delta)$, and let $\Delta: L \to L$ be given by

$$\Delta \phi p = \{a : a \in p\} \cup \{<a, \phi p'> : <a,p'> \in p\}, \text{ if } p \neq \varnothing$$
$$= \{\delta\}, \text{ if } p = \varnothing$$

Let $abs = fix(\Delta)$. Note that $abs(p)$ deletes all $<<m?,...>$ or $<m!,...>$ steps from p. We have

THEOREM 5.7. *For each $s \in L_{rv}, \mathcal{O}[\![s]\!] = (abs \circ \mathcal{D})[\![s]\!]$.*

6. Interpreting the atoms

We discuss two nonuniform languages L_{as} and L_{co}. In L_{as}, we interpret the elementary actions of L_{sh} as assignments (and introduce a conditional construct). In L_{co} we add to L_{as} a simple version of (CSP-like) communication.

6.1. *States and resumptions*

Let $(v \in)Ivar$ be the syntactic class of *individual variables*, and let $(f \in)Exp$ and $(b \in)Bexp$ be the classes of *expressions* and *booleans*, respectively. We omit specification of a syntax for f and b. In the syntax for L_{as} (definition 6.1) we do not introduce a subclass of guarded statements. A remark explaining this follows below.

Definition 6.1.

a. $s(\in L_{as}) ::= v:=f \mid x \mid s_1;s_2 \mid$ **if** b **then** s_1 **else** s_2 **fi** $\mid s_1 \| s_2$

b. A declaration D is a finite set of pairs $<x,s>$. A program is as usual.

Let $(\alpha \in)V$ and $(\beta \in)W$ be the sets of *values* or *truth-values*. Let $(\sigma \in)\Sigma = Ivar \rightarrow V$ be the set of *states*. We assume as given two functions $[\![\cdot]\!]:Exp \rightarrow \Sigma \rightarrow V$ and $[\![\cdot]\!]:Bexp \rightarrow \Sigma \rightarrow W$. The operational semantics for L_{as} is defined in terms of transitions $<s,\sigma> \rightarrow <s',\sigma'> \mid <E,\sigma'>$. T_{as} is defined in

Definition 6.2. a. $<v:=f,\sigma> \rightarrow <E,\sigma[\alpha/v]>$, with $\alpha = [\![f]\!](\sigma)$.

b. $<x,\sigma> \rightarrow <s,\sigma>$, for $<x,s>$ in D

c. If $<s,\sigma> \rightarrow <s',\sigma'> \mid <E,\sigma'>$ then
- $<s;\bar{s},\sigma> \rightarrow <s';\bar{s},\sigma'> \mid <\bar{s},\sigma'>$
- $<s \| \bar{s},\sigma> \rightarrow <s' \| \bar{s},\sigma'> \mid <\bar{s},\sigma'>$
- $<\bar{s} \| s,\sigma> \rightarrow <\bar{s} \| s',\sigma'> \mid <\bar{s},\sigma'>$

d. **if** - **fi** case omitted.

$\mathcal{O}:L_{as} \rightarrow \Sigma \rightarrow \mathcal{P}_{nc}(\Sigma^\infty)$ is 'defined' in

$$\mathcal{O}[\![s]\!](\sigma) = \{\sigma' \mid <s,\sigma> \rightarrow <E,\sigma'>\} \cup \bigcup \{\sigma'.\mathcal{O}[\![s']\!](\sigma'): <s,\sigma> \rightarrow <s',\sigma'>\}$$

Remark. A consequence of this definition of $\mathcal{O}$ is that we do not have, in general, that $\mathcal{O}[\![s_1 \| s_2]\!] = \mathcal{O}[\![s_1]\!] \| \mathcal{O}[\![s_2]\!]$ (assuming a suitable semantic '$\|$'). We shall see in a moment how (the compositionality of) $\mathcal{D}$ requires a more complex domain:

Let $(p \in)P_{as}, (q \in)Q_{as}$ be domains solving the equations

$$P_{as} = \Sigma \rightarrow Q_{as}$$

$$Q_{as} = \mathcal{P}_{closed}(\Sigma \cup (\Sigma \times P_{as}))$$

Processes p in P_{as} are functions delivering, for input state σ, *sets* $p(\sigma)$ of results of the form σ' or $<\sigma',p'>$. In $<\sigma',p'>$ the new state σ' is delivered together with the new process p', a *resumption* of p (these ideas are from [Pl1]). The semantic operators '$\circ$' and '$\|$' on P_{as} are defined as follows (essential clauses only, and with some abuse of notation): $p_1 \circ p_2 = \lambda\sigma.p_1(\sigma) \circ p_2$, $p_1 \| p_2 = \lambda\sigma.(p_1(\sigma) \| p_2) \cup (p_2(\sigma) \| p_1)$. Also, $q \circ p = \{y \circ p : y \in q\}, \sigma \circ p = <\sigma,p>, <\sigma,p'> \circ p = <\sigma,p' \circ p>$, and $q \| p = \{y \| p : y \in q\}, \sigma \| p = <\sigma,p>, <\sigma,p'> \| p = <\sigma,p' \| p>$. Next, we define $\mathcal{D}$. Let $(F \in)M = L_{as} \rightarrow P_{as}$. We put $\mathcal{D} = fix(\Phi)$, with $\Phi:M \rightarrow M$ given in

Definition 6.3. $\Phi F(v:=f) = \lambda\sigma.\{\sigma[\alpha/v]\}$, with $\alpha = [\![f]\!](\sigma), \Phi F($**if**.**fi**$)$: omitted. $\Phi F(s_1;s_2) = (\Phi F s_1) \circ (\Phi F s_2)$, and similarly for $\|, \Phi F x = \lambda\sigma.\{<\sigma,Fs>\}$, with $<x,s>$ in D.

Remark. A procedure call x is defined in terms of a skip (mapping σ to σ) followed by execution of its body s. This device obviates the need for a syntactic guardedness requirement.

We now discuss how to relate $\mathcal{O}$ and $\mathcal{D}$. We cannot expect a simple equivalence, since the relevant (co) domains differ. We define another abstraction $trace: P \rightarrow \Sigma \rightarrow \mathcal{P}_{nc}(\Sigma^\infty)$. The function of $trace$ is twofold: First, it linearizes the tree (-like) structure of processes p. Moreover, in pairs $<\sigma,p>$ - to be interpreted as 'p is ready to execute σ' it ensures that p is indeed applied to σ. Let $(F\in)N = P\to\Sigma\to\mathcal{P}_{nc}(\Sigma^\infty)$. We define $\Gamma: N\to N$ by

$$\Gamma Fp\sigma = \{\sigma':\sigma'\in p(\sigma)\} \cup \bigcup\{\sigma'.Fp'\sigma': <\sigma',p'>\in p(\sigma)\}.$$

Putting $trace = fix(\Gamma)$, it may now be shown that

THEOREM 6.4. *For $s\in L_{as}$, $\mathcal{O}[\![s]\!] = (trace\circ\mathcal{D})[\![s]\!]$.*

6.2. Communication

Let $(c\in)Chan$ be a set of *channels*. We extend L_{as} to L_{co} by putting $s(\in L_{co})::=c?v\,|\,c!f\,|$ (as for L_{as}). Synchronized execution of $c?v$ and $c!f$ in a parallel construct $s_1\|s_2$ induces the 'handshake' communication $v := f$. In this section we concentrate on $\mathcal{D}$, and do not discuss how to define $\mathcal{O}$ (which requires small variations on earlier techniques). In order to define $\mathcal{D}$ for L_{co} we need another domain P_{co}, obtained as follows: let $(\eta\in)H = \{c?v: c\in Chan, v\in Ivar\}\cup\{c!\alpha: c\in Chan, \alpha\in V\}$, and let $(\tau\in)T = \Sigma\cup H$. We define $(p\in)P_{co}, (q\in)Q_{co}$ as solution of

$$P_{co} = \Sigma\to Q_{co}$$

$$Q_{co} = \mathcal{P}_{compact}(T\cup(T\times P_{co}))$$

The definition of '$\circ$' on P_{co} is almost as that for P_{as}. For '$\|$' we put $p_1\|p_2 = \lambda\sigma.(p_1(\sigma)\|p_2)\cup(p_2(\sigma)\|p_1)\cup(p_1(\sigma)\,|_\sigma p_2(\sigma))$. The definition of $q\|p$ is similar to that in Section 6.1. The term $p_1(\sigma)\,|_\sigma p_2(\sigma)$ is new. We define it in the clause

$$\begin{aligned} q_1\,|_\sigma q_2 = \{&<\sigma[\alpha/v],p_1\|p_2>:<c?v,p_1>\in q_1, <c!\alpha,p_2>\in q_2 \text{ or vice versa,}\\ &<\sigma[\alpha/v],p_1>:<c?v,p_1>\in q_1, c!\alpha\in q_2 \text{ or vice versa,}\\ &<\sigma[\alpha/v],p_2>:c?v\in q_1, <c!\alpha,p_2>\in q_2 \text{ or vice versa,}\\ &\sigma[\alpha/v]:c?v\in q_1, c!\alpha\in q_2, \text{ or vice versa}\}\end{aligned}$$

The definition of $\mathcal{D}$ for L_{co} is now an immediate extension of that for L_{as}. In particular, it contains (what amounts to) the clauses $\mathcal{D}[\![c?v]\!] = \lambda\sigma.\{c?v\}$, and $\mathcal{D}[\![c!f]\!] = \lambda\sigma.\{c!\alpha\}$, with $\alpha = [\![f]\!](\sigma)$. Finally, we mention that it is, again, possible to define a suitable abstraction operator *abs* - this time combining features of *trace* and of a restriction operator throwing away unsuccessful (i.e. one-sided) communications, cf. *abs* for L_{rv} - such that $\mathcal{O} = abs\circ\mathcal{D}$.

7. CONCLUSION

We have discussed a variety of fundamental techniques which may be (and have been) applied in the design of semantics for concurrent languages. We mention a few distinguishing features:

- the metric framework which allows a simple treatment of infinite behaviour, effective use of higher - order techniques, and a smooth transition towards formal language theory and domain theory
- a unified method of comparing operational and denotational semantics
- the option of modelling language concepts at the schematic (uninterpreted) or interpreted level, together with the choice to switch from linear time to branching time (as well as intermediate, cf. [R2]) models.

In recent years, we have collected evidence that the tools illustrated in our lecture may fruitfully be applied to (parallel versions of) imperative languages ([BZ2, BMOZ]), object-oriented languages ([ABKR1,2,R1]), dataflow ([K]) and logic programming languages [B1,BK,BoKPR,Vi]).

We conclude with a list of three challenges for future work:

- perform a systematic study of *full abstraction* (has $\mathcal{D}$ the right level of detail w.r.t. $\mathcal{O}$?, cf. [HP,R2]) for (all) the languages discussed;

- (for logic programming) investigate the relationship between its 'declarative' semantics and the kind of models outlined above;
- exploit semantic knowledge in the design and justification of logical systems for parallel languages.

REFERENCES

[AA] IJ.J. AALBERSBERG, P. AMERICA, *personal communication.*

[A] P. AMERICA, *Definition of POOL 2, a parallel object-oriented language*, ESPRIT project 415, doc. 0364, Philips Research, Eindhoven, 1988.

[AB] P. AMERICA, J.W. DE BAKKER, *Designing equivalent semantic models for process creation*, Theoretical Computer Science 60 (1988), 109-176.

[ABKR1] P. AMERICA, J.W. DE BAKKER, J.N. KOK, J.J.M.M. RUTTEN, *Operational semantics of a parallel object-oriented language*, 13th ACM Symposium on Principles of Programming Languages, St. Petersburg, Florida, January 13-15, 1986, pp. 194-208.

[ABKR2] P. AMERICA, J.W. DE BAKKER, J.N. KOK, J.J.M.M. RUTTEN, *A denotational semantics of a parallel object-oriented language*, Report CS-R8626 Centre for Mathematics and Computer Science, Amsterdam (1986), to appear in Information and Computation.

[AR] P. AMERICA, J.J.M.M. RUTTEN, *Solving reflexive domain equations in a category of complete metric spaces*, in Proc. of the Third Workshop on Mathematical Foundations of Programming Language Semantics, (M. Main, A. Melton, M. Mislove, D. Schmidt, eds.) Lecture Notes in Computer Science, Vol. 298, Springer (1988), pp. 254-288.

[B1] J.W. DE BAKKER, *Comparative semantics for flow of control in logic programming without logic*, Report CS-R8840, Centre for Mathematics and Computer Science, Amsterdam (1988).

[B2] J.W. DE BAKKER (ed.), *Languages for Parallel Architectures: Design, Semantics, Implementation Models*, Wiley, to appear in 1989.

[BBKM] J.W. DE BAKKER, J.A. BERGSTRA, J.W. KLOP, J.-J.CH. MEYER, *Linear time and branching time semantics for recursion with merge*, TCS 34 (1984) 135-156.

[BK] J.W. DE BAKKER, J.N. KOK, *Uniform abstraction, atomicity and contractions in the comparative semantics of Concurrent Prolog*, Report CS-R8834, Centre for Mathematics and Computer Science, Amsterdam, extended abstract in Proc. Fifth Generation Computer Systems, Tokyo, 1988, pp. 347-355.

[BKMOZ] J.W. DE BAKKER, J.N. KOK, J.-J.CH. MEYER, E.-R. OLDEROG, J.I. ZUCKER, *Contrasting themes in the semantics of imperative concurrency*, in Current Trends in Concurrency: Overviews and Tutorials (J.W. de Bakker, W.P. de Roever, G. Rozenberg, eds.), Lecture Notes in Computer Science, Vol. 224, Springer (1986) 51-121.

[BM1] J.W. DE BAKKER, J.-J.CH. MEYER, *Order and metric in the stream semantics for elemental concurrency*, Acta Informatica 24 (1987) 491-511.

[BM2] J.W. DE BAKKER, J.-J.CH. MEYER, *Metric semantics for concurrency*, BIT 28 (1988), 504-529.

[BMO] J.W. DE BAKKER, J.-J.CH. MEYER, E.-R. OLDEROG, *Infinite streams and finite observations in the semantics of uniform concurrency*, Theoretical Computer Science 49 (1987) 87-112.

[BMOZ] J.W. DE BAKKER, J.-J.CH. MEYER, E.-R. OLDEROG, J.I. ZUCKER, *Transition systems, metric spaces and ready sets in the semantics of uniform concurrency*, Journal of Comp. Syst. Sc. 36 (1988), 158-224.

[BNT1] J.W. DE BAKKER, A.J. NIJMAN, P.C. TRELEAVEN (eds.) Proceedings *PARLE Parallel Architectures and Languages Europe*, Vol. 1, *Parallel Architectures*, Lecture Notes in Computer Science 258, Springer, 1987.

[BNT2] J.W. DE BAKKER, A.J. NIJMAN, P.C. TRELEAVEN (eds.) Proceedings *PARLE Parallel Architectures and Languages Europe*, Vol. 2, *Parallel Languages*, Lecture Notes in

Computer Science 259, Springer, 1987.

[BRR1] J.W. DE BAKKER, W.P. DE ROEVER, G. ROZENBERG (eds.,) *Current Trends in Concurrency*, Proc. LPC/ESPRIT Advanced School, Springer Lecture Notes in Computer Science 224, (1986).

[BRR2] J.W. DE BAKKER, W.P. DE ROEVER, G. ROZENBERG (eds.), *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, Proc. REX Workshop, Lecture Notes in Computer Science, Springer, to appear.

[BZ1] J.W. DE BAKKER, J.I. ZUCKER, *Processes and the denotational semantics of concurrency*, Inform. and Control 54 (1982) 70-120.

[BZ2] J.W. DE BAKKER, J.I. ZUCKER, *Processes and a fair semantics for the ADA rendez-vous*, in: Proc. 10th ICALP (J. Diaz, ed.) Lecture Notes in Computer Science, Vol. 154, Springer (1983) 52-66.

[BeK] J.A. BERGSTRA, J.W. KLOP, *Process algebra: specification and verification in bisimulation semantics*, in Mathematics and Computer Science II (M. Hazewinkel, J.K. Lenstra, L.G.L.T. Meertens, eds.), CWI Monographs 4, North-Holland (1986) 61-94.

[BoKPR] F.S. DE BOER, J.N. KOK, C. PALAMIDESSI, J.J.M.M. RUTTEN, *Control flow versus logic: a denotational and a declarative model for Guarded Horn Clauses*, Report CS-R89.., Centre for Mathematics and Computer Science, Amsterdam, to appear.

[Gi] G. GIERZ, K.H. HOFMANN, K. KEIMEL, J.D. LAWSON, M. MISLOVE, D.S. SCOTT, *A compendium of continuous lattices*, Springer-Verlag, 1980.

[HP] M. HENNESSY, G.D. PLOTKIN, *Full abstraction for a simple parallel programming language*, in: Proceedings 8th MFCS (J. Becvar ed.), Lecture Notes in Computer Science, Vol. 74 Springer (1979) 108-120.

[Ho] C.A.R. HOARE, *Communicating Sequential Processes*, Prentice Hall Int., Englewood Cliffs, New Jersey (1985).

[K] J.N. KOK, *Dataflow semantics*, Report CS-R8835, Centre for Mathematics and Computer Science, Amsterdam (1988).

[KR] J.N. KOK, J.J.M.M. RUTTEN, *Contractions in comparing concurrency semantics*, in Proc. 15th ICALP (T. Lepistö, A. Salomaa, eds.), Lecture Notes in Computer Science, Vol. 317, Springer (1988) 317-332.

[Me] J.-J.CH. MEYER, *Merging regular processes by means of fixed point theory*, Theoretical Computer Science 45 (1986) 193-260.

[MV] J.-J.CH. MEYER, E.P. DE VINK, *Step semantics for 'true' concurrency with recursion*, Technical Report IR 138, Department of Computer Science, Free University Amsterdam, 1988.

[Mi] R. MILNER, *A Calculus for Communicating Systems*, Lecture Notes in Computer Science, Vol. 92 Springer (1980).

[Ni1] M. NIVAT, *Mots infinis engendrés par une grammaire algebrique*, RAIRO Informatique Théorique 11 (1977) pp. 311-327.

[Ni2] M. NIVAT, *Infinite words, infinite trees, infinite computations*, Foundations of Computer Science III.2, Mathematical Centre Tracts 109 (1979) 3-52.

[OH] E.-R. OLDEROG, C.A.R. HOARE, *Specification-oriented semantics for communicating processes*, Acta Informatica 23 (1986) 9-66.

[P11] G.D. PLOTKIN, *A powerdomain construction*, SIAM Journal of Computing (1976) 452-487.

[P12] G.D. PLOTKIN, *A structural approach to operational semantics*, Report DAIMI FN-19, Comp.Sci.Dept., Aarhus Univ. 1981.

[P13] G.D. PLOTKIN, *An operational semantics for CSP*, in: D. Bjørner (ed.): Formal description of programming concepts II, North-Holland (1983) 199-223.

[Pn] A. PNUELI, *Linear and branching structures in the semantics and logics of reactive systems*, in: W. Brauer (ed.): Proc. of the 12th Int. Coll on Automata, Languages and

Programming (ICALP), Nafplion, Greece, 1985, Lecture Notes in Computer Science, Vol. 194, Springer (1986) 15-32.

[R1] J.J.M.M. RUTTEN, *Semantic correctness for a parallel object-oriented language*, Report CS-R8843, Centre for Mathematics and Computer Science, Amsterdam (1988).

[R2] J.J.M.M. RUTTEN, *Correctness and full abstraction of metric semantics for concurrency*, in Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency (J.W. de Bakker, W.P. de Roever, G. Rozenberg, eds.), Lecture Notes in Computer Science, Springer, to appear.

[Sc] M.P. SCHUTZENBERGER, *Push-down automata and context free languages*, Inf. and Control, 6 (1963) 246-264.

[TV] D.A. TAUBNER, W. VOGLER, *The step failure semantics*, in Proc. STACS 1987 (F.-J. Brandenburg, G. Vidal-Naquet, M. Wirsing, eds.), Lecture Notes in Computer Science, Vol 247, Springer (1987) 348-359.

[Vi] E.P. DE VINK, *Equivalence of an operational and denotational semantics for a Prolog-like language with cut*, Technical Report IR 151, Free University, Amsterdam, 1988.