Semantics of Communicating Processes

MANFRED BROY

Fakultät für Informatik, Universität Passau, Postfach 25 40, D–8390 Passau, West Germany

The semantics of a simple language for describing tightly coupled "synchronous" systems is defined. An operational semantics is given by term rewriting rules and a consistent fully abstract denotational semantics is defined based on the concept of observable behavior and advanced fixed point theory. Particular properties of the language are analysed and especially algebraic laws of the language are discussed. Finally some aspects and problems of the formal definition of the semantics of such a language are treated also comparing them to other approaches found in the literature. © 1984 Academic Press, Inc.

1. INTRODUCTION

A simple version of a language for describing synchronous concurrent systems is introduced. This language called CP can be seen as a simplified version of Milner's calculus of communicating systems (CCS, cf. Milner, 1980a,b) or Hoare's communicating sequential processes (CSP, cf. Hoare, 1978). Apart from some not very significant details it is identical with a language introduced by Hoare to talk about semantic aspects of CSP. Its operational semantics is defined by term-rewriting rules. Its denotational semantics is defined based on the idea of nondeterministic stream processing functions. The special representation of the denotational semantics is motivated by an experimental environment. This gives a clear notion of processes as deterministic or "nondeterministic" (set-valued) stream processing functions. The fixed point theory for communicating processes is based on the ideas of (Broy, 1982), where the meaning of a recursive definition of a stream processing function is defined by a particular fixed point that can be characterized by combining distinct power domains. Thus a fully abstract denotational semantics is derived for CP, which is proved to be consistent with the operational semantics.

Afterwards algebraic properties of the language are analysed and discussed. Finally other aspects and problems arising when giving a formal semantics to CP are investigated. In particular a number of further possibilities for the semantic representation of processes are discussed. Problems arising there are related to abstractness of the representation and fixed point theory.
2. Syntax

We use a very simple language CP following (Hoare et al. 1981). It can be seen as the simplest version of a language for describing synchronized, concurrent systems. Its syntax in BNF reads:

\[
\langle \text{agent} \rangle ::= \text{skip} \quad \text{immediately terminating process}
\]

| \text{abort} \quad \text{nonterminating process} |
| \langle \text{action} \rangle \to \langle \text{agent} \rangle \quad \text{process starting with the resp. action} |
| \langle \text{agent} \rangle \lor \langle \text{agent} \rangle \quad \text{simple nondeterministic choice} |
| \langle \text{agent} \rangle \mid \langle \text{agent} \rangle \quad \text{guarded choice} |
| \langle \text{agent} \rangle \parallel \langle \text{action set} \rangle \langle \text{agent} \rangle \quad \text{parallel composition} |
| \langle \text{agent} \rangle ; \langle \text{agent} \rangle \quad \text{sequential composition} |
| \langle \text{agent} \rangle \setminus \langle \text{action} \rangle \quad \text{hiding of actions} |
| \langle \text{id} \rangle \quad \text{identifier for agent} |
| \langle \text{id} \rangle :: \langle \text{agent} \rangle \quad \text{recursively defined agent} |

Here we assume that a countable set ACTION is given for the objects taken from the syntactic entity \langle \text{action} \rangle and a set \langle \text{id} \rangle of identifiers for agents. By \langle \text{action set} \rangle we denote the set consisting of arbitrary sets of actions from \langle \text{action} \rangle.

3. Operational Semantics

As a first step we give an operational semantics. This is done for several reasons. First, an operational semantics does not lead into the complex problems of powerdomains that have to be dealt with in mathematical semantics. Second, formalized operational semantics guarantees that the concepts incorporated in the language actually can be implemented. Third, it serves as a complementary definition such that the “correctness” of mathematical (“denotational”) semantics can be verified. Fourth, it helps to understand where the particular problems of mathematical semantics come from, since the “abstraction” mapping the operational semantics into the mathematical one can be made explicit.

3.1. Term Rewriting Rules for CP

Operational semantics is defined by the introduction of a ternary partial relation \( OP \), i.e., a partial predicate

\[
OP: \langle \text{agent} \rangle \times (\langle \text{action} \rangle \cup \{ \varepsilon \}) \times \langle \text{agent} \rangle \to \{ \text{true}, \text{false} \}.
\]

We assume, that \( \varepsilon \) is an element that does not occur in the set ACTION. This predicate is defined as a least fixed point of a set of conditional
equations. For better readability we write

\[ p_1 \xrightarrow{a} p_2 \]

for \( \text{OP}(p_1, a, p_2) = \text{true} \). Intuitively this means that the agent \( p_1 \) may perform the action \( a \) and then behave like the agent \( p_2 \). If \( a = \varepsilon \) then we speak of a silent transition. The relation \( \text{OP} \) is a "rewriting rule with output" or a "rewriting rule indicating an event."

Now we define \( \text{OP} \) by induction on the terms of sort agent (where \( x \subseteq \text{ACTION} \) or \( x = \varepsilon \)):

\begin{align*}
\text{abort} & \xrightarrow{\varepsilon} \text{abort} , \\
x & \rightarrow p \xrightarrow{x} p , \\
p_1 \text{ or } p_2 & \xrightarrow{\varepsilon} p_1 , \\
p_1 \text{ or } p_2 & \xrightarrow{\varepsilon} p_2 , \\
p_1 & \xrightarrow{x} p \land x \neq \varepsilon \Rightarrow (p_1 \mid p_2) \xrightarrow{x} p , \\
p_1 & \xrightarrow{x} p \land x \neq \varepsilon \Rightarrow (p_2 \mid p_1) \xrightarrow{x} p , \\
p_1 & \xrightarrow{\varepsilon} p \Rightarrow p_1 \mid p_2 \xrightarrow{\varepsilon} p \mid p_2 , \\
p_1 & \xrightarrow{\varepsilon} p \Rightarrow p_2 \mid p_1 \xrightarrow{\varepsilon} p_2 \mid p , \\
\text{skip } | a & \xrightarrow{\varepsilon} a , \\
a | \text{skip} & \xrightarrow{\varepsilon} a , \\
p_1 & \xrightarrow{x} p_3 \land p_2 \xrightarrow{x} p_4 \land x \in c \Rightarrow p_1 \parallel c \cdot p_2 \xrightarrow{x} p_3 \parallel c \cdot p_4 , \\
p_1 & \xrightarrow{x} p_3 \land \neg(x \in c) \Rightarrow p_1 \parallel c \cdot p_2 \xrightarrow{x} p_3 \parallel c \cdot p_2 , \\
p_1 & \xrightarrow{x} p_3 \land \neg(x \in c) \Rightarrow p_2 \parallel c \cdot p_1 \xrightarrow{x} p_2 \parallel c \cdot p_3 , \\
\text{skip } \parallel c \cdot \text{skip} & \xrightarrow{\varepsilon} \text{skip} , \\
p_1 & \xrightarrow{x} p_3 \Rightarrow p_1 ; p_2 \xrightarrow{x} p_3 ; p_2 , \\
\text{skip } ; p & \xrightarrow{\varepsilon} p , \\
q : p & \xrightarrow{\varepsilon} p[(q : p)/q] , \\
q \in \langle \text{id} \rangle & \Rightarrow q \xrightarrow{\varepsilon} q , \\
p_1 & \xrightarrow{x} p_2 \land x \neq a \Rightarrow p_1 \backslash a \xrightarrow{x} p_2 \backslash a , \\
p_1 & \xrightarrow{\varepsilon} p_2 \Rightarrow p_1 \backslash a \xrightarrow{\varepsilon} p_2 \backslash a , \\
\text{skip } \backslash a & \xrightarrow{\varepsilon} \text{skip} ,
\end{align*}
The only agent that is terminal in a defined way in the relation $\xrightarrow{\cdot}$, i.e., that definitely cannot be reduced, is the agent \texttt{skip}. We use the axiom:

$$OP(\texttt{skip}, x, p) = \text{false}.$$ 

So the agent \texttt{skip} represents the only agent where we definitely say that there is no successor. So it represents the process that has properly terminated.

Here $p[p_1/q]$ denotes the agent arising from $p$ by replacing all free occurrences of $q$ by $p_1$. So the reduction rule for recursively defined agents simply corresponds to $\beta$-reduction. Of course, we assume that name clashes between bound identifiers and identifiers occurring freely in $p_1$ are resolved by appropriate renaming of bound identifiers ($\alpha$-reduction).

We take for $OP$ the least defined relation, i.e., the least fixed point, that fulfills the laws above. According to (Broy and Wirsing, 1982) such a relation exists. Note, that for a deadlock (with $a \neq b$) such as

$$(a \rightarrow \texttt{skip} \| c \rightarrow b \rightarrow \texttt{skip}),$$

where $c = \{a, b\}$, the relation

$$(a \rightarrow \texttt{skip} \| c \rightarrow b \rightarrow \texttt{skip}) \xrightarrow{\cdot} p$$

is not defined, or more precisely,

$$OP((a \rightarrow \texttt{skip} \| c \rightarrow b \rightarrow \texttt{skip}), x, p)$$

is not defined for all agents $p$ and all actions $x$ not for $x = \varepsilon$. According to the fact that $OP$ is a partial relation, the assertion

$$p_1 \xrightarrow{\cdot} p_2$$

(or more precisely, $OP(p_1, x, p_2)$) can be true, false, or not defined. If for some agent $p_1$ this relation is false for all actions $x$ and all agents $p_2$, then we say that the agent $p_1$ has properly terminated. In our language \texttt{skip} is the only agent that has properly terminated. If the relation is undefined for all actions $x$ and for $x = \varepsilon$ and all agents $p_2$, then $p_1$ is called a deadlock, and we write

$$\text{deadlock}(p_1).$$

If there is a sequence of agents $\{p_i\}_{i \in \mathbb{N}}$ with $p_0 = p_1$ and

$$p_i \xrightarrow{\cdot} p_{i+1},$$
then we say that $p1$ has the possibility to diverge and write

$$\text{diverge}(p1).$$

3.2. Streams

For giving a more abstract semantics to agents we introduce the notion of streams. Basically a stream is defined by a sequence of atoms. So let $\text{ATOM}$ be some given set of atomic values (including the natural numbers and the truth values, for instance). By $\text{ATOM}^{\perp}$ the classical flat domain over $\text{ATOM}$ is denoted, i.e., the partially ordered set with just $\perp$ as the least element and all other elements incomparable. By $\text{ATOM}^*$ we denote the finite sequences of elements from $\text{ATOM}$, by $\text{ATOM}^{\infty}$ the infinite sequences. Then the set of streams is defined by

$$\text{STREAM}(\text{ATOM}) =: (\text{ATOM}^* \times \{\perp\}) \cup \text{ATOM}^* \cup \text{ATOM}^{\infty}$$

This definition represents the union of all finite partial streams with all finite total and all infinite (and total) streams, if we use the ordering for streams $s_1, s_2$, defined by

$$s_1 \sqsubseteq s_2 \iff s_1 = s_2 \quad \text{or} \quad s_1 = s_3 \circ (\perp) \text{ and } s_3 \text{ is prefix of } s_2.$$

By $\langle a \rangle$ the one-element sequence is denoted consisting just of the atom $a$ from $\text{ATOM}^{\perp}$. The symbol $\circ$ is used to denote the usual concatenation of sequences to sequences. By "&" we denote the operator adding an atomic element as first element to a stream, i.e.,

$$a \& s = \langle a \rangle \circ s \quad \text{for} \quad a \in \text{ATOM}, \quad \perp \& s = \langle \perp \rangle,$$

is assumed. Note, that & is leftstrict. By $\varepsilon$ the empty stream is denoted. For streams we use the two functions

$$\text{first}: \text{STREAM}(\text{ATOM}) \rightarrow \text{ATOM}^{\perp},$$

$$\text{rest}: \text{STREAM}(\text{ATOM}) \rightarrow \text{STREAM}(\text{ATOM})$$

which are defined by

$$\text{first}(\varepsilon) = \text{first}(\langle \perp \rangle) = \perp, \quad \text{first}(a \& s) = a,$$

$$\text{rest}(\varepsilon) = \text{rest}(\langle \perp \rangle) = \langle \perp \rangle, \quad \text{rest}(a \& s) = s.$$

With these definitions we immediately may prove

**LEMMa.** \text{STREAM}(\text{ATOM}) forms an algebraic domain; the functions first, "&", and rest are monotonic and continuous.

As pointed out in (Broy, 1982) streams are a very basic notion in
concurrent communicating systems. In systems based on shared memory one has to consider streams of states, in tightly coupled systems one has to consider streams of actions.

3.3. Transitive Closure of the Rewriting Relation

Since we want to abstract from the "single" steps in computations we introduce a second partial relation $OP^*$,

$$OP^* : \langle \text{agent} \rangle \times \text{ACTION}^* \times \langle \text{agent} \rangle \rightarrow \{\text{true}, \text{false}\},$$

and write

$$p_1 \xrightarrow{\xi} p_2 \quad \text{for} \quad OP^*(p_1, s, p_2) = \text{true}.$$ 

It is defined by the least defined relation that fulfills

$$p_1 \xrightarrow{x} p_2 \Rightarrow p_1 \xrightarrow{(x)} p_2,$$

$$p_1 \xrightarrow{e} p_2 \Rightarrow p_1 \xrightarrow{e} p_2,$$

$$p_1 \xrightarrow{s_1} p_2 \land p_2 \xrightarrow{s_2} p_3 \Rightarrow p_1 \xrightarrow{s_1 \circ s_2} p_3.$$ 

Moreover we introduce a set-valued function

$$\text{cap} : \langle \text{agent} \rangle \rightarrow P(\text{STREAM(ACTION)}),$$

defined by

$$\text{cap}(p) = \{s \in \text{ACTION}^* : p \xrightarrow{s} \text{skip} \land \text{lub}(s_i \circ \langle \bot \rangle) : \{s_i\}_{i \in N} \subseteq \text{ACTION}^* \land \exists \{a_i\}_{i \in N} \subseteq \text{ACTION}^* : \forall k \in N : \exists p_1, p_2 \in \langle \text{agent} \rangle : p \xrightarrow{s} p_1 \land (p_1 \xrightarrow{a_i} p_2 \land s_{k+1} = s_k \circ \langle a_k \rangle) \lor ((\text{deadlock}(p_1) \lor \text{diverge}(p_1)) \land s_k = s_{k+1}).$$

A stream $s \in \text{cap}(p)$ is called a run of $p$. Note, that this particular definition has been chosen according to the principle of finite observability. The set of capabilities is determined by the set of finite runs (runs with a bounded amount of "time") of the system. This definition nontermination and deadlocks are treated the same way. Based on these notions we can now discuss appropriate abstractions for the operational semantics.
3.4. Abstractions from Operational Semantics

The operational semantics defined in the preceding sections can be perfectly taken as definition of the meaning of CP. However, such a semantics is not very abstract. And abstraction from unimportant details is badly needed if one wants to cope with the enormous combinatorial complexity found in a communicating, concurrent system. So now we are going to study several possibilities of abstraction.

The operational semantics defined in the preceding section defines for every term $p$ of the syntactic unit $\langle$agent$\rangle$ a finitary (i.e., finitely branching) finite or infinite tree, with terms of sort $\langle$agent$\rangle$ as labels in the nodes and arcs labelled by actions. This tree will be denoted by $T(p)$. If we forget about the labels in the nodes and just keep the tree-structure the arcs of which are labelled with the actions (or with $\epsilon$) we obtain a first simple abstraction from the operational semantics. These trees will be denoted by $AT(p)$. Formally one has

$$T: \langle\text{agent}\rangle \rightarrow \text{TREE},$$

$$AT: \langle\text{agent}\rangle \rightarrow \text{ACTIONTREE}.$$  

Of course one could give a "denotational semantics" to CP formalizing $AT$ (cf. Francez et al., 1978; and de Bakker and Zucker, 1982).

However, the abstraction obtained by $AT$ is not very strong. It just abstracts from the particular way in which a recursive agent is defined (such as bound identifiers) and from sequential notation. This leads to the question: which abstraction is strong enough? And we have to be careful, too, not to abstract too much and to forget about important properties of an agent. But what are the important properties of an agent and what are the unimportant ones? For solving this question we adopt a simple notion of observable behavior.

Given an agent $p$ the one thing we assume that we can observe are the action that the agent executes if we start it. Then $p$ performs a run, i.e., we obtain a stream $s$ with

$$s \in \text{cap}(p).$$

Moreover, we can use $p$ for building other agents, i.e., we may construct an agent $C[p]$ such that $p$ is a subterm of $C[p]$ and then observe the actions that are executed if we start $C[p]$. Thus we obtain a run $s_1$ of $C[p]$ with

$$s_1 \in \text{cap}(C[p]).$$

This means, that the semantic identity of an agent $p$ is not only determined by $\text{cap}(p)$ but also by the behavior of $C[p]$, i.e., by $\text{cap}(C[p])$ for arbitrary
contexts $C[\cdot]$. This means that we should be able not only to observe the behavior of $p$ itself, but also the effects of $p$ within any system $C[p]$, i.e., what happens if we put $p$ in some given context $C[\cdot]$. This fixes our notion of observability (cf. Broy and Wirsing, 1981). Note, that it is essential that $\varepsilon$-transitions are not observable here nor the particular shape of an agent after having performed a sequence of actions.

Based on this notion of observability we can now talk about the “abstractness” of a semantic model. Assume we have found a semantic representation for agents by mapping them into some set $\text{PROCESS}$. Then the semantic mapping
\[ S: \langle \text{agent} \rangle \rightarrow \text{PROCESS} \]

has to have the property that it is a homomorphism from the term-algebra (the set of terms or programs of the syntactic unit $\langle \text{agent} \rangle$) into the semantic model $\text{PROCESS}$. So for every language construct in CP, for instance, parallel composition $\|_c$, there has to be a semantic operation, such as
\[ \text{par}:: \langle \text{ACTION SET} \rangle \rightarrow \text{PROCESS} \times \text{PROCESS} \rightarrow \text{PROCESS} \]

with
\[ S[a_1 \|_c a_2] = \text{par}(\varepsilon)(S[a_1], S[a_2]); \]

$S$ has to keep all the relevant information for an agent for using the agent as subcomponent.

In addition the abstraction of the operational behavior should be kept by $S$. Accordingly $S$ is called extensionally equivalent (or consistent) to (the behavior specified by) the operational semantics, if there is some function
\[ \text{CAP}: \text{PROCESS} \rightarrow \text{P}(\text{STREAM(ACTIONS)}) \]

with
\[ \text{CAP}(S[p]) = \text{cap}(p) \]

for all agents $p$. This means that Fig. 1 has to commute.
However we are not satisfied by some $S$ that keeps too much information about a process. We want $S$ to keep exactly as much information as necessary for $S$ to be consistent and fulfilling the composition property above. The composition property can nicely be expressed by contexts. For a program term $C[a]$ of the syntactic unit (agent), where $a$ occurs at a particular position, $C$ is called a context. The semantics $S$ is called abstract (w.r.t. the extensional behavior defined by the operational semantics described above) iff for two agents $p_1$, $p_2$ we have

$$S[p_1] = S[p_2] \text{ if for all contexts } C: \text{cap}(C[p_1]) = \text{cap}(C[p_2]).$$

Note, that for this notion of observability it is essential not to forget about the contexts, i.e., about the possibility of observing the behavior of a system of processes after integrating the given process into a system of other processes (cf. Hennessy and Milner, 1980). A semantics $S$ is called fully abstract iff $S$ is consistent and abstract.

Including the consideration of contexts is sufficient to obtain models for CP. One possibility to obtain concrete models is to define term-oriented representations (cf. Hennessy and Plotkin, 1980) by specifying an appropriate congruence relation on the set of terms of the syntactic unit (agent). We even may construct a fully abstract semantic model in the form of the behavior term algebra along the lines of (Broy and Wirsing, 1981). However, term-oriented semantic models generally have the severe drawback that they do not support the intuition very much. So we adopt a notion of experiment in the following section that leads to a denotational semantic model and finally to a fully abstract one much more helping our understanding.

### 4. Denotational Semantics

According to what has been said above, a process $P$ cannot simply be represented by its capabilities, because in certain contexts $C$ agents $P1$ and $P2$ with identical capabilities may nevertheless lead to agents $C[P1]$ and $C[P2]$ with distinct capabilities. This shows that also other properties of the process $P$ are important (its "incapabilities") to define the capabilities of $C[P]$. According to this we choose a particular idea of "experiments."

#### 4.1. Experiments

For a better understanding and also for defining a mathematical semantics for CP based on the idea of powerdomains, it is important to develop a clear notion of a determinate process. Since in CP there are concepts of global nondeterminism (expressed by $\|$) as well as concepts of local nondeterminism (expressed by $\text{or}$), such a notion has to be chosen very carefully.
For explaining our choice we adopt a particular model of experiment for testing the identity of a process. The test machinery consists of a processing unit in which the CP program is loaded and of a display together with a keyboard on which actions can be chosen. After loading a CP program an experiment proceeds as follows:

0) The machine may display "terminated." Then the experiment is finished.

1) The experimenter chooses an arbitrary action and presses the resp. button on the keyboard.

2) The machine displays either "rejected" or "accepted" or it may not give an answer at all (divergence, nondetermination).

3) The experimenter may finish the experiment or (s)he may continue according to 0.

If the machine displayed "terminated," then further input according to (1) cannot change this behavior.

According to these experiments a process is called determinate, if for repeated experiments with the same process and a chosen sequence of actions it shows always the same behavior, otherwise (the behavior of) the process is called nondeterminate. Note, that with this model of experiments neither \(\varepsilon\)-transitions nor the internal state of the machinery are observable.

4.2. Denotational Semantics of Nonrecursively Defined Agents

According to the experimental philosophy described above a determinate process is a function from

\[
\text{STREAM}(\text{ACTION}) \rightarrow \text{STREAM}(\{A, R\}).
\]

An object from \(\text{STREAM}(\text{ACTION})\) is called an experiment, an object from \(\text{STREAM}(\{A, R\})\) is called a behavior. The relationship to the experimental environment above is as follows: termination is expressed by the empty stream \(\varepsilon\), nontermination by \(\perp\), acceptance or rejection by \(A\) or \(R\), respectively. Note, that for our purposes it is sufficient to consider just partial streams as experiments, but it is important to have both partial and total streams as behaviors. A nondeterminate process then simply is a function with a set of possible behaviors for every experiment, namely an object from

\[
\text{PROCESS} = [\text{STREAM}(\text{ACTION}) \rightarrow P(\text{STREAM}(\{A, R\}))].
\]

For defining the meaning of terms of the syntactic unit \(\langle\text{agent}\rangle\) with free identifiers we use the classical technique of environments:

\[
\text{ENV} = [\langle\text{id}\rangle \rightarrow \text{PROCESS}].
\]
As usual the replacement of a process given under identifier $q$ in an environment $E$ by some other process $p$ is denoted by $E[p/q]$. In particular, we have

$$q = q_1 \Rightarrow E[p/q](q_1) = p,$$
$$q \neq q_1 \Rightarrow E[p/q](q_1) = E(q_1).$$

Now we are going to define the meaning of agents by translating them into nondeterminate stream-processing functions. This translation is represented by the semantic function $S$,

$$S: \langle \text{agent} \rangle \to \text{ENV} \to \text{PROCESS}.$$  

The function $S$ is defined inductively on the terms:

*The Immediately Terminating Agent:* The agent $\text{skip}$ immediately terminates.

$$S[\text{skip}](E)(s) = \{\varepsilon\},$$

*The Never Terminating Agent:* The agent $\text{abort}$ immediately enters a nonterminating loop.

$$S[\text{abort}](E)(s) = \{\bot\},$$

*The Agent Performing a Determinate Action:* The agent $a \rightarrow p$ has just the capability to accept the action $a$ (all other actions are rejected) and then behaves like $p$,

$$S[a \rightarrow p](E)(s) = \{f(q,s,a): q \in S[p](E)(h(s,a))\},$$

where $f$ and $h$ are stream-processing functions defined by the least fixed points of the equations

$$f(q, a \& s, a) = A \& q,$$
$$a \neq a' \Rightarrow f(q, a' \& s, a) = R \& f(q, s, a),$$
$$h(a \& s, a) = s,$$
$$a \neq a' \Rightarrow h(a' \& s, a) = h(s, a).$$

So the agent $a \rightarrow p$ rejects all actions in the experiment $s$ until the first occurrence of the action $a$. This action is accepted and for the remaining stream $h(s, a)$ of experiments the agent behaves like $p$.

*Nondeterministic Choice:* The agent $p_1$ or $p_2$ has the free choice to behave like $p_1$ or like $p_2$,

$$S[p_1 \text{ or } p_2](E)(s) = S[p_1](E)(s) \cup S[p_2](E)(s).$$
So the set of behaviors of for $p_1$ or $p_2$ is just the union of the behaviors of $p_1$ and $p_2$.

**Joining Capabilities:** The agent $p_1 | p_2$ behaves like $p_1$ or $p_2$, however, if some action is offered, then it may only be rejected if both processes are ready to do so,

$$S[p_1 | p_2](E)(s) = \{ g(q_1, q_2): q_1 \in S[p_1](E)(s) \land q_2 \in S[p_2](E)(s) \lor q_2 \in S[p_1](E)(s) \land q_1 \in S[p_2](E)(s) \},$$

where $g$ is the stream-processing function defined by the least fixed point of the equations

- $g(A \& q, q_2) = A \& q_2$,
- $g(q, e) = q = g(e, q)$,
- $g(R \& q_1, R \& q_2) = R \& g(q_1, q_2)$,
- $g(R \& q_1, A \& q_2) = A \& q_2$.

The agent $p_1 | p_2$ joins the capabilities of agent $p_1$ and $p_2$ with respect to accept actions, but can only reject an action if both $p_1$ and $p_2$ are ready to do so. So a defined behavior of $p_1$ is only a feasible behavior of $p_1 | p_2$ if it starts with $A$ for accepted or if there is a behavior of $p_2$ which is empty or starts with a $R$ for rejected. This is expressed by the function $g$.

**Sequential Composition:** The agent $p_1; p_2$ denotes the sequential composition of $p_1$ and $p_2$; as soon as $p_1$ has terminated the agent continues behaving according to $p_2$:

$$S[p_1; p_2](E)(s) = \{ h_1(q_1, q_2): q_1 \in S[p_1](E)(s) \land q_2 \in S[p_2](E)(h_2(q_1, s)) \},$$

where $h_1$ and $h_2$ are the least fixed points of the equations

- $h_1(e, q_2) = q_2$,
- $h_1(X \& q_1, q_2) = X \& h_1(q_1, q_2)$,
- $h_2(e, s) = s$,
- $h_2(X \& q, s) = h_2(q, \text{rest}(s))$.

So the prefix of the experiment $s$ is given to the agent $p_1$ until $p_1$ has properly terminated with the behavior $q_1$ and the remaining experiment $h_2(q_1, s)$ is given to $p_2$. The produced behavior of the agent $p_1; p_2$ then simply is obtained by the “left-strict” concatenation of $q_1$ to the behavior produced by $p_2$.

**Parallel Synchronized Composition:** The agent $p_1 \parallel_c p_2$ defines the synchronized (with respect to $c$) parallel composition of agents $p_1$ and $p_2$. For actions from $c$ the processes join their possibilities to reject, but they can only accept the action if both are ready to do so. For actions not from $c$ they
join their capabilities to accept actions, but they may only reject an action if both are ready to do so:

\[
S[p1 ||_c p2](E)(s) = \\
\{ \text{par}(q1, q2, d, s): \\
q1 \in S[p1](E)(\text{fork}(s, d, 1)) \wedge q2 \in S[p2](E)(\text{fork}(s, d, 2)) \\
\wedge d \in \{0, 1, 2\}^\infty \wedge \text{pos}(q1, q2, d, s) \equiv \text{true}, \}
\]

where fork, par, and pos are the least fixed points of the equations

\[
\text{fork}(\epsilon, d, n) = \epsilon, \\
\text{fork}(x \& s, 0 \& d, n) = \text{fork}(x \& s, n \& d, n) = x \& \text{fork}(s, d, n), \\
m \neq n \wedge m \neq 0 \Rightarrow \text{fork}(x \& s, m \& d, n) = \text{fork}(s, d, n),
\]

\[
\text{par}(\epsilon, e, d, s) = e, \\
(x \in c \wedge X = R) \vee (- (x \in c) \wedge X = A) \\
\Rightarrow \text{par}(q1, X \& q2, 0 \& d, x \& s) = X \& \text{par}(q1, q2, 1 \& d, x \& s) = X \& \text{par}(q1, q2, d, s), \\
(X = A \Rightarrow x \in c) \Rightarrow \text{par}(X \& q1, X \& q2, 0 \& d, x \& s) = X \& \text{par}(q1, q2, d, s), \\
\text{par}(R \& q1, e, 0 \& d, x \& s) = R \& \text{par}(q1, e, d, s), \\
\text{par}(e, R \& q2, 0 \& d, x \& s) = R \& \text{par}(e, q2, d, s),
\]

\[
\text{pos}(\epsilon, e, d, s) = \text{true}, \\
\text{pos}(\epsilon, X \& q, 1 \& d, s) = \text{pos}(X \& q, \epsilon, 2 \& d, s) = \text{false}, \\
\text{pos}(R \& q1, e, 0 \& d, x \& s) = \text{pos}(q1, e, d, s), \\
\text{pos}(e, R \& q2, 0 \& d, x \& s) = \text{pos}(\epsilon, q2, d, s), \\
(x \in c \wedge X = R) \vee (- (x \in c) \wedge X = A) \\
\Rightarrow \text{pos}(q1, X \& q2, 0 \& d, x \& s) = X \& \text{pos}(q1, q2, d, s), \\
\text{pos}(X \& q1, q2, 1 \& d, x \& s) = \text{pos}(q1, q2, d, s), \\
(X = A \Rightarrow x \in c) \Rightarrow \text{pos}(X \& q1, X \& q2, 0 \& d, x \& s) = \text{pos}(q1, q2, d, s), \\
\text{pos}(R \& q1, R \& q2, 0 \& d, x \& s) = \text{pos}(R \& q1, A \& q2, 0 \& d, x \& s) = \text{false},
\]

In a parallel composition \( p1 \parallel_c p2 \) the two agents can do rejections of actions that are not in \( c \) only together, i.e., they join their capabilities for accepting those actions. For actions in \( c \), however, they can do acceptances only together synchronously, but they join the capabilities for rejecting those actions. In the formal definition above \( d \) can be seen as the “scheduler” that distributes the actions of the experiment to one of the processes or to both of them. The partial predicate pos indicates whether this distribution is actually feasible.
We have chosen this lengthy definition of parallel composition, because we found it more systematic than a more tricky shorter one. Moreover it allows for a simpler proof of monotonicity and continuity properties.

_Hiding:_ The agent $p \backslash a$ denotes the agent resulting from hiding the action $a$: explicit input of $a$ is rejected, but arbitrary numbers of hidden offers of $a$ are added:

$$S[p \backslash a](E)(s) = \{\text{hide}(d, q1, s, a):$$
$$q1 \in S[p](E)(\text{include}(d, s, a))$$
$$\land d \in \{0, 1\}^\infty \land \text{apos}(d, q1, s, a) \subseteq \text{true}\},$$

where hide, include, apos, and rpos are the least fixed points of the equations

include(0 & d, a & s, a) = include(d, s, a),

$$x \neq a \Rightarrow \text{include}(0 & d, x & s, a) = x \& \text{include}(d, s, a),$$

include(1 & d, s, a) = a & include(d, s, a),

$$\text{hide}(d, e, s, a) = e,$$

$$\text{hide}(0 & d, X & q, a & s, a) = R \& \text{hide}(d, X & q, s, a),$$

$$a \neq x \Rightarrow \text{hide}(0 & d, X & q, x & s, a) = X \& \text{hide}(d, q, s, a),$$

$$\text{hide}(1 & d, X & q, q, a) = \text{hide}(d, q, s, a),$$

$$\text{apos}(d, e, s, a) = \text{true},$$

$$\text{apos}(0 & d, X & q, a & s, a) = \text{false},$$

$$a \neq x \Rightarrow \text{apos}(0 & d, A & q, x & s, a) = \text{apos}(d, q, s, a),$$

$$a \neq x \Rightarrow \text{apos}(0 & d, R & q, x & s, a) = \text{false},$$

$$\text{apos}(1 & d, A & q, s, a) = \text{apos}(d, q, x & s, a),$$

$$\text{apos}(1 & d, R & q, x & s, a) = \text{apos}(d, q, x & s, a),$$

$$\text{rpos}(d, e, s, a) = \text{true},$$

$$\text{rpos}(0 & d, X & q, a & s, a) = \text{rpos}(d, X & q, s, a),$$

$$a \neq x \Rightarrow \text{rpos}(0 & d, A & q, x & s, a) = \text{apos}(d, q, s, a),$$

$$a \neq x \Rightarrow \text{rpos}(0 & d, R & q, x & s, a) = \text{apos}(d, q, s, a),$$

$$\text{rpos}(1 & d, A & q, x & s, a) = \text{apos}(d, q, x & s, a),$$

$$\text{rpos}(1 & d, R & q, x & s, a) = \text{false},$$

The definitions of apos and rpos simply say that an agent $p \backslash a$ may only reject some action $b$ ($\neq a$) if $p$ is also capable (possibly after accepting some hidden offers) to reject the hidden offer of the action $a$.

**Identifiers for Agents:** The meaning of an identifier is simply taken from the environment,

$$S[q](E) = E(q).$$
This way the behaviors of nonrecursively defined agents are fixed for all finite and infinite experiments. Note, that the auxiliary functions are all continuous. The definition of the semantic function $S$ can be mainly seen as an exercise in defining the auxiliary stream-processing functions, i.e., writing applicative programs.

4.3. Recursively Defined Agents

As it is well-known, the treatment of recursively defined agents as fixed points is the most complicated part of the denotational semantics for concurrent programs. The main reason is that classical fixed point theory based on exactly one domain with one partial ordering does not work.

4.3.1. Semantics of recursively defined agents. Given a recursively defined agent

$q::p$,

we may associate with it a functional for every environment,

$$T_{q::p}: \text{ENV} \rightarrow \text{PROCESS} \rightarrow \text{PROCESS}$$

declared by

$$T_{q::p}(E)[p1] = S[p](E[p1/q]).$$

Now following classical techniques we would like to define the semantics $S[q::p](E)$ of $q::p$ to be the “least” fixed point $f$ of the equation

$$f(s) = T_{q::p}(E)[f](s) \quad \text{for experiments } s. \quad (1)$$

Now (1) is an equation between sets. So we have to fix a partial ordering between sets for indicating in which sense the fixed point should be least. The classical orderings on powerdomains (see Appendix) do not work properly because they are only quasi-ordering for sets over nonflat domains such as the domain of streams. Inclusion ordering for the power set does not work since we do not want to consider agents with empty sets of behaviors and so we are missing a least element.

However, if we exclude the empty set, then the power set (without the empty set) ordered by set inclusion just forms a predomain. We can solve fixed point equations in predomains for monotonic and continuous functionals if we manage to replace the missing least element by some auxiliary construction. This auxiliary construction can be based on powerdomains.
We do not give the basic definitions of power domains here but just refer to the Appendix. Note that in the following "\~" is used to denote functional composition. Throughout this section we consider only environments which contain only processes $p$ with $\text{CONE}(p(s)) \subseteq PD(\text{STREAM}([A, R]))$ for all experiments $s$ and, where $\text{CONE} \cdot p$ is $\subseteq^*_{M}$-monotonic, $\subseteq^*_{E}$-monotonic, $\subseteq^*_{EM}$-monotonic, $\subseteq^*_{M}$-continuous, $\subseteq^*_{E}$-continuous, and $\subseteq^*_{EM}$-continuous. Such processes and, respectively, such environments are also called \textit{perfect}.

\textbf{Theorem.} For every CP-program $p$ and every perfect environment $E$, we have

1. the functional $T_{q;:p}(E)$ is $\subseteq$-monotonic and closely union continuous,
2. the functional $\text{PDOM} \cdot T_{q;:p}(E)$ is $\subseteq^*_{EM}$-monotonic and $\subseteq^*_{EM}$-continuous,
3. the functional $\text{PMDOM} \cdot T_{q;:p}(E)$ is $\subseteq^*_{M}$-monotonic and $\subseteq^*_{M}$-continuous,
4. the functional $\text{PEDOM} \cdot T_{q;:p}(E)$ is $\subseteq^*_{E}$-monotonic and $\subseteq^*_{E}$-continuous.

\textbf{Proof:} See Appendix.

Now we can define the meaning $S[q;:p]$ of the agent $q;:p$ as a fixed point of the functional $T_{q;:p}(E)$, i.e., we have

$$S[q;:p](E) = T_{q;:p}(E)[S[q;:p](E)].$$

This equation does not characterize $S[q;:p](E)$ uniquely, in general. In particular, now we have

1. to prove that the class of functions fulfilling these equations is nonempty;
2. uniquely define which function of the class is to be taken.

We define the function $S[q;:p](E)$ in three steps: At first we define approximations $SP$ and $\text{MIN} \cdot SP$ in the powerdomains of erratic and demonic nondeterminism, resp., such that we are sure that

$$\text{MIN} \cdot SP[q;:p](E) \subseteq^* S[q;:p](E) \subseteq^* SP[q;:p](E).$$

Since the set of closed sets without the empty set is a predomain if ordered by set inclusion, we can use $\text{MIN} \cdot SP[q;:p](E)$ in place of the missing least element. So based on these approximations we constructively can prove the existence of a fixpoint of (2) and define $S[q;:p](E)$ uniquely.
The approximation $SP[q::p](E)$ for $S[q::p](E)$ is defined by a fixed point equation in the powerdomain of erratic nondeterminism, such that intuitively for the required $S[q::p](E)$:

$$SP[q::p](E) = PDOM \cdot S[q::p](E).$$

should hold. Define

$$SP[q::p](E): STREAM(ACTION) \rightarrow PD(STREAM(\{A, R\}))$$

as the least fixed point $f$ (in the domain of erratic nondeterminism) of the equation:

$$f(s) = PDOM(T_{q::p}(E)[f](s)).$$

This fixed point is well defined since $T_{q::p}(E)$ is $\subseteq_{EM}$-monotonic and $\subseteq_{EM}$-continuous.

**Lemma.** $SP[q::p](E)$ is consistently defined, $\subseteq_{EM}$-monotonic, and $\subseteq_{EM}$-continuous.

Now we can also try to solve our fixed point equation in the powerdomain of demonic nondeterminism. This can be simply done by the functor $MIN$, that maps our solution in the powerdomain of erratic nondeterminism into the powerdomain of demonic nondeterminism.

$$MIN \cdot SP[q::p](E): STREAM(ACTION) \rightarrow PM(STREAM(\{A, R\})),$$

represents a solution $f$ of the equation

$$f(s) = PEDOM(T_{q::p}(E)[f](s)).$$

Now surely all minimal behaviours, i.e., the elements of $MIN \cdot SP[q::p](E)(s)$, are possible behaviours. Furthermore we want exactly that set of behaviours that includes these minimal ones and those that have definitely to be included for fulfilling the fixed point property. So we define $S[q::p](E)$ as the $\subseteq^*$-least function with

$$MIN \cdot SP[q::p](E) \subseteq^* S[q::p](E) \subseteq^* SP[q::p](E),$$

that is closed and that fulfills the specifying fixed point equation (1).

**Proof of the existence of $S[q::p](E)$ (constructive).** (1) Define a family of functions

$$S_l[q::p](E): STREAM(ACTION) \rightarrow P(STREAM(\{A, R\}))$$
So\[q::p\](E) = \text{MIN} \cdot SP[q::p](E),
S_{i+1}[q::p](E) = \text{CLOSE} \cdot T_{q::p}(E)[S_i[q::p](E)].

(2) We have
S_i[q::p](E) \subseteq* S_{i+1}[q::p](E),
because all language constructs are \subseteq*-monotonic in S[q::p](E) and

\begin{align*}
S_0[q::p](E) &= \text{MIN} \cdot SP[q::p](E) \\
&= \text{MIN} \cdot \text{PDOM} \cdot T_{q::p}(E)[S_0[q::p](E)] \\
&\subseteq* \text{CLOSE} \cdot T_{q::p}(E)[\text{MIN} \cdot \text{PDOM} \cdot S_0[q::p](E)] \\
&= \text{CLOSE} \cdot T_{q::p}(E)[S_0[q::p](E)] \\
&= S_1[p::q](E).
\end{align*}

(3) Define
S[q::p](E) = \bigcap^* \{z \in \text{PROCESS} : \\
\text{MIN} \cdot SP[q::p](E) \subseteq* z \land z \\
= \text{CLOSE} \cdot T_{q::p}(E)[z]\}.

since all language constructs are union-monotonic, according to Knaster and Tarski S[q::p](E) is well defined and a fixed point of the defining equations.

Note, that all processes that can be defined in our language are perfect, i.e., for perfect environments E and for all agents p we know that S[p](E) is perfect. Since T_{q::p}(E) is closely union continuous we obtain a corollary similar to Kleene's fixed point theorem. Let \( \Omega \) denote the process with \( \Omega(s) = \langle 1 \rangle \) for all experiments s.

**Corollary.**

S[q::p](E) = \text{CLOSE} \cdot (\bigcup^* T_{q::p}(E)[\text{MIN} \cdot T_{q::p}(E)[\Omega]])

In analogy to the classical fixed point theorem we can use this duality between the fixed point characterization and the lub-characterization for proving the consistency of operational and denotational semantics.

4.4. Consistency with Operational Semantics

Now we have two independent definitions of the meaning of CP, an operational one and a denotational one. It remains to prove that both
coincide w.r.t. the defined capabilities. For doing so we introduce the function

\[ \text{CAP: PROCESS} \rightarrow P(\text{STREAM(ACTION)}) \]

defined by

\[
\text{CAP}(P) = \{ s : \exists i \in \mathbb{N} : s \in \text{ACTION}^i \land A^i \subseteq P(s) \} \\
\cup \{ s : \exists i \in \mathbb{N} : s \in \text{ACTION}^i \times \{ \bot \} : \\
\forall a \in \text{ACTION} : A^i \circ \langle \bot \rangle \in P(s \circ \langle a \rangle) \\
\forall A^i \circ (R \& \langle \bot \rangle) \in P(s \circ \langle a \& \langle \bot \rangle \rangle) \}
\cup \{ s \in \text{ACTION}^\omega : A^\omega \subseteq P(s) \}.
\]

Now let the environment \( E_\bot \) be defined by

\[ E_\bot(q) = \Omega. \]

for every identifier \( q \).

The consistency of operational and mathematical semantics can be seen by the following theorem.

**Theorem.** For every CP-program \( p \) we have

\[ \text{cap}(p) = \text{CAP}(S[p](E_\bot)), \]

i.e., \( S[p](E_\bot) \) is extensionally equivalent to the operational semantics and hence \( S \) is consistent.

**Proof.** See Appendix.

Now we have proved that our denotational semantics is extensionally equivalent with the operational one, i.e., it is neither inconsistent nor too abstract. But our denotational semantics could carry too much information, i.e., it could be not abstract enough. This is true as it can be seen from the following example programs:

\[ p1 = (b \rightarrow e \rightarrow \text{skip}) \text{ or } (b \rightarrow \text{skip} | c \rightarrow \text{skip}) \text{ or } (b \rightarrow e \rightarrow \text{skip} | c \rightarrow \text{skip}), \]

\[ p2 = (b \rightarrow \text{skip}) \text{ or } (b \rightarrow \text{skip} | c \rightarrow \text{skip}) \text{ or } (b \rightarrow e \rightarrow \text{skip} | c \rightarrow \text{skip}). \]

We have

\[ R \& A \& A \& e \in S[p1](c \& b \& e \& c), \]
\[ -(R \& A \& A \& e \in S[p2](c \& b \& e \& e)), \]

but

\[ \text{cap}(C[p1]) = \text{cap}(C[p2]) \]
for any context $C$. Actually $S$ is not abstract enough. However, we can abstract from $S$ by introducing a slightly modified semantic function

$$S1: \langle \text{agent} \rangle \rightarrow \text{ENV} \rightarrow \text{PROCESS}$$

specified by

$$S1[p](E)(s) = \{\text{strip}(q) : q \in S[p](E)(s)\},$$

where \text{strip} and \text{forget} are the least fixed points of the equations:

\[
\begin{align*}
\text{strip}(e) &= e, \\
\text{strip}(A \& q) &= A \& \text{strip}(q) \\
\text{strip}(R \& q) &= R \& \text{forget}(q).
\end{align*}
\]

\[
\begin{align*}
\text{forget}(e) &= e, \\
\text{forget}(A \& q) &= \langle \bot \rangle, \\
\text{forget}(R \& q) &= R \& \text{forget}(q).
\end{align*}
\]

$S1$ is fully abstract as can be seen by the following theorem. Note, that the definition of CAP does not make any difference if we use $S1$ instead of $S$.

**Theorem.** The denotational semantics defined by $S1$ is fully abstract.

**Proof.** See Appendix.

So the denotational semantics defined by $S1$ is fully abstract, i.e., a semantic model which is as abstract as possible, but still consistent with extensional behavior defined by the operational semantics.

Note, that we can define $S1$ exactly the same way as $S$. The only thing we have to change is definition of $S$ in Section 4.2 inserting \text{strip} on all right-hand sides of the definitions. The fixed point theory then can be used exactly the same way. The equivalence with the operational semantics is also trivially guaranteed, since for CAP only sequences from $\{A\}^* \times \{R\}^*$ are considered. We can even define an experimental environment corresponding to $S1$, where an experiments ends as soon as the first time some offer has been accepted after a nonempty sequence of offers has been rejected. We have just preferred to work with $S$ since we found the experimental environment more suggestive.

$S1$ is fully abstract, since an agent may be influenced by the context in which decision to take at most for the next visible action to perform. If we would have language constructs (generalizing the joining of capabilities by $\mid$), where the nondeterministic choice of an agent can be influenced over a finite sequence of offered actions, then $S$ would become a fully abstract semantic model.
5. Properties of CP

In this section we are going to analyse a number of properties of CP. First we look at properties of the semantic functions. Then we investigate algebraic properties of CP.

5.1. Properties of Processes

By the denotational semantics every CP process is mapped (for every environment) onto a set-valued function. But not all functions can be obtained this way. The functions have particular properties. For instance they are monotonic and continuous in the ways defined in the previous sections. But there are further characteristic properties. For talking conveniently about these properties we introduce the length $|s|$ of a stream $s$ defined by

$$
\forall s \in \text{ATOM}^i \quad |s \circ \langle \bot \rangle| = |s| = i,
\forall s \in \text{ATOM}^\infty \quad |s| = \infty.
$$

**Lemma.** Let $f = S[p](E)$ for some CP-program $p$. Then for every experiment $s$ and every behavior $q$ we have

\begin{enumerate}
\item $q \in f(s) \Rightarrow |q| \leq |s|$, i.e., the length of the behaviors is bounded by the length of the experiments.
\item $q_1 \circ \langle R \rangle \circ q_2 \in f(s_1 \circ \langle a \rangle \circ s_2) \land |q_1| = |s_1| \Rightarrow q_1 \circ q_2 \in f(s_1 \circ s_2)$, i.e., a rejection does not change the “state” of a process.
\item $|s| = i \land |s_1| = |q_1| \Rightarrow -(q_1 \circ R^{i+1} \circ \langle A \rangle \circ q_2 \in f(s_1 \circ \langle a \rangle \circ s \circ \langle a \rangle \circ s_2))$, i.e., one rejected action cannot be accepted as long as all other actions offered in the meantime are rejected.
\item For all finite $q \in \{A, R\}^j$, $s \in \text{ACTION}^j$ with $j > i + 1$: $-(q \circ \langle R \rangle \circ \varepsilon \in f(s)) \land -(q \circ \langle R \rangle \circ \langle \bot \rangle \in f(s))$, i.e., if a process is able to reject, it has not terminated nor diverged so far.
\end{enumerate}

**Proof:** Structural induction on the definition of $S$.

This lemma clearly shows that rejected offers are input that does not affect the further output of an agent.

5.2. Algebraic Properties

An important concept for increasing our understanding of the semantic concepts of the language CP is to study its algebraic properties. Finding out, which agents are extensional equivalent (i.e., behave abstractly the same) and which are not, of course helps us basically to understand the semantic concepts. One even may define the meaning of CP this way (cf. Broy and Wirsing, 1982). However, since we have already fixed the semantics by
denotational/operational means we now can prove the following algebraic properties of CP: Here for agents $p_1$, $p_2$ we define

$$p_1 \sim p_2 \text{ iff } S_1[p_1] = S_1[p_2].$$

**Associativity laws:**

$$
(p_1 || p_2) || p_3 \sim p_1 || (p_2 || p_3),
(p_1; p_2); p_3 \sim p_1; (p_2; p_3),
(p_1 \text{ or } p_2) \text{ or } p_3 \sim p_1 \text{ or } (p_2 \text{ or } p_3),
(p_1 ||_e p_2) ||_e p_3 \sim p_1 ||_e (p_2 ||_e p_3),
$$

**Commutativity laws:**

$$
(p_1 || p_2) \sim (p_2 || p_1),
(p_1 \text{ or } p_2) \sim (p_2 \text{ or } p_1),
(p_1 ||_e p_2) \sim (p_2 ||_e p_1),
$$

**Distributivity laws for or:**

$$
p \text{ or } p \sim p,
(p_1 \text{ or } p_2) || p \sim (p_1 || p) \text{ or } (p_2 || p),
(p_1 \text{ or } p_2); p \sim (p_1; p) \text{ or } (p_2; p),
p; (p_1 \text{ or } p_2) \sim (p; p_1) \text{ or } (p; p_2),
(p_1 \text{ or } p_2) ||_e p \sim (p_1 ||_e p) \text{ or } (p_2 ||_e p),
x \rightarrow (p_1 \text{ or } p_2) \sim (x \rightarrow p_1) \text{ or } (x \rightarrow p_2),
$$

**Laws for $|$:**

$$
p || p \sim p,
p || \text{ skip } \sim p,
(x \rightarrow p_1) || (x \rightarrow p_2) \sim x \rightarrow (p_1 \text{ or } p_2),
((x_i \rightarrow p_i) || \cdots || (x_n \rightarrow p_n)); p \sim (x_i \rightarrow (p_i; p)) || \cdots || (x_n \rightarrow (p_n; p)),
$$

**Laws for skip and abort:**

$$
\text{abort}; p \sim \text{abort},
p; \text{ skip } \sim p \sim \text{ skip}; p,
\text{ abort } ||_e \text{ abort } \sim \text{ abort},
\text{ skip } ||_e \text{ skip } \sim \text{ skip},
(p ||_e \text{ abort }) \sim (p ||_e \text{ abort}); \text{ abort},
$$

**Laws for hiding:**

$$
\text{skip}\backslash a \sim \text{skip},
\text{abort}\backslash a \sim \text{abort},
(a \rightarrow p)\backslash a \sim p\backslash a,
x \neq a \rightarrow (x \rightarrow p)\backslash a \sim x \rightarrow (p\backslash a),
(p_1 \text{ or } p_2)\backslash a \sim (p_1\backslash a) \text{ or } (p_2\backslash a),
(q :: p)\backslash a = q :: (p\backslash a),
$$
Fixed point properties:
\[ q :: p \sim p[(q :: p)/q]. \]

All these algebraic properties can be easily proved by structural induction looking at the definition of the denotational semantics.

6. Comparison to Other Semantic Models

Giving a denotational semantics for concurrent processes, one has to face two essential problems: finding a semantic representation for agents and specifying the semantics of recursively defined agents as a particular solution for the fixed point equation for processes.

6.1. Semantic Representations

In the denotational semantics given for CP a CP-Program is mapped (for a given environment) onto a rather simple semantic representation: a process is a function mapping experiments into sets of behaviors. However there are many other ways to represent agents semantically.

Throughout this section we want to consider example program \( t \),

\[ q :: ((a \rightarrow q; b \rightarrow \text{skip}) \text{ or } (c \rightarrow \text{skip}) \text{ or } q). \]

6.1.1. CP-programs as trees. Our operational semantics gives a finitary tree for every program. For \( t \) we obtain the infinite finitary tree, tree(\( t \)) of Fig. 2a and for \( t|a \) the tree of Fig. 2b.

Now we want to abstract from the \( e \)-productions, i.e., we want to delete all \( e \)-arcs. But for every node we want to keep the information about infinite paths (nonterminating computations) labelled by \( e \) only starting from that node. So such nodes will be labelled by \( \perp \) now. We obtain for \( t \) the tree in Fig. 3a and for \( t|a \) the tree in Fig. 3b.

This shows that hiding leads to infinitary trees with nodes that have infinitely many branches. This is why hiding (even hiding the \( e \)-arcs) is noncontinuous, if we consider trees for the semantic representation of agents. Then the identity of the agent after having observed the action \( c \) is the union over all possible identities the agent actually (operationally) may have. Since after having observed the action \( c \) of the agent \( t|a \) the agent has only the capabilities to do a finite but arbitrary number of \( b \)'s and then to terminate, there is no way to represent this agent by a finitary tree with arcs only labelled by \( b \)'s and \( c \)'s, without having an infinite path labelled by \( b \)'s. This leads us to the problems of resumptions.

The semantic function \( S1 \) induces a congruence relation on \( e \)-free trees that is exactly the congruence (cf. Hennessy and Plotkin, 1980) that people
\[ \begin{align*} &\varepsilon / \varepsilon \quad \varepsilon \\
&\quad \quad \text{tree}(t) \\
&\quad c / a \\
&\quad \quad \epsilon / \varepsilon \\
&\quad \quad \quad \text{tree}(t) \\
&\quad c / \varepsilon \\
&\quad \quad \epsilon / \varepsilon \\
&\quad \quad \quad \text{tree}(t) \\
&\quad b / a \\
&\quad \quad \quad \epsilon / \varepsilon \\
&\quad \quad \quad \quad \text{tree}(t) \\
&\quad \quad \quad \quad c / \varepsilon \\
&\quad \quad \quad \quad \quad \epsilon / \varepsilon \\
&\quad \quad \quad \quad \quad \quad \text{tree}(t) \\
&\quad \quad \quad \quad \quad b / \varepsilon \\
&\quad \quad \quad \quad \quad \quad \epsilon / \varepsilon \\
&\quad \quad \quad \quad \quad \quad \quad \text{tree}(t) \\
&\quad \quad \quad \quad \quad \quad \quad c / \varepsilon \\
&\quad \quad \quad \quad \quad \quad \quad \quad \epsilon / \varepsilon \\
&\quad \quad \quad \quad \quad \quad \quad \quad \quad \text{tree}(t) \\
&\quad \quad \quad \quad \quad \quad \quad \quad b / \varepsilon \\
&\quad \quad \quad \quad \quad \quad \quad \quad \quad \epsilon / \varepsilon \\
&\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \text{tree}(t) \\
&\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad b / \\
\end{align*} \]

**Figure 2a**

\[ \begin{align*} &\varepsilon / \varepsilon \quad \varepsilon \\
&\quad \quad \text{tree}(t) \\
&\quad c / \varepsilon \\
&\quad \quad \epsilon / \varepsilon \\
&\quad \quad \quad \text{tree}(t) \\
&\quad \quad c / \varepsilon \\
&\quad \quad \quad \epsilon / \varepsilon \\
&\quad \quad \quad \quad \text{tree}(t) \\
&\quad \quad \quad \quad b / \varepsilon \\
&\quad \quad \quad \quad \quad \epsilon / \varepsilon \\
&\quad \quad \quad \quad \quad \quad \text{tree}(t) \\
&\quad \quad \quad \quad \quad \quad \quad c / \varepsilon \\
&\quad \quad \quad \quad \quad \quad \quad \quad \epsilon / \varepsilon \\
&\quad \quad \quad \quad \quad \quad \quad \quad \quad \text{tree}(t) \\
&\quad \quad \quad \quad \quad \quad \quad \quad \quad b / \\
\end{align*} \]

**Figure 2b**
tried to axiomatize in other work (but which seems impossible in first-order algebraic equations).

6.1.2. Resumption based semantics. In (Plotkin, 1976) there is already the suggestion to consider communicating processes as elements of the domain specified by the domain equation

$$\text{RES} \simeq [\text{ACTION} \rightarrow P(\text{ACTION}^+ \cup (\text{ACTION} \times \text{RES}))].$$

Such a way of representing processes seems to be a very natural one in the light of operational semantics. A process is an individual that performs some action and then becomes another process (or it may terminate or diverge). However, if a process passes a position of possible divergence successfully, there may be an infinite number of possible identities the process then may actually have. If we cannot observe which particular one is actually valid, the process semantically represents a “nondeterministic” individual consisting of an infinite set of processes. In particular it cannot be finitely approximated, i.e., that it is not a member of the powerdomain.
Thus apart from the domain problem, i.e., from the problem showing the existence of such a set and finding an appropriate ordering for it, the resumption of a process after having performed some action \( a \) may be an infinite set of processes (cf. the example above) that cannot be finitely approximated in the powerdomain: the function

\[
\text{resume}: \text{ACTION} \times \text{PROCESS} \to \text{PROCESS},
\]

specified by

\[
\text{resume}(a, p)(s) = \{ q : A & q \in p(a & s) \},
\]

is not continuous w.r.t. the powerdomain of erratic nondeterminism. More precisely it maps certain processes that are finitely approximable in the powerdomain of erratic nondeterminism onto processes that do not have this property. Thus it leads to unbounded nondeterminism, such that the \( \subseteq_{EM} \) continuity of the basic language constructs is no longer valid. This can be seen by the examples of the tree representations above (consider \( \text{resume}(c, S[t\alpha](E_\bot)) \); cf. also (Broy and Wirsing, 1982)).

In (de Bakker and Zucker, 1982) a denotational semantics for concurrent processes is given based on metric spaces. There, however, \( \varepsilon \)-productions are still contained. The semantic function is not abstract. Every behavior is total and thus maximal. Then the basic problems arising from power domains are avoided, but the resulting semantics is less abstract and "more operational." One could speak of a "denotational operational semantics." So any of the three power domains of angelic, demonic, or erratic nondeterminism works without problems and gives identical results.

6.1.3. Acceptor and refusal sets. Other papers present semantic definitions that work with acceptor sets (also called "ready sets") or refusal sets (cf. Hoare et al., 1981, Broy and Wirsing, 1982). As already pointed out, the identity of a process is not determined only by the set of all paths (runs) through its tree (this is only true for loosely coupled systems), since the operator \( \top \) and, similarly, the parallel combinator can be used to offer a set of actions simultaneously. The information about such sets is not contained in the set of all paths (runs). This is why sets of paths (runs, traces) are not sufficient (cf. Hennessy and Milner, 1980) as semantic representation.

Our approach is isomorphic to the notion of an acceptor set or a refusal set. Given a sequence of actions \( s \subseteq \text{ACTION}^I \) and a process \( f \) with \( A^I \circ \langle \bot \rangle \in f(s) \) then a set of actions \( c \subseteq \text{ACTION} \) is called an acceptor set after the observation of \( s \) if \( c = \{ c_1, \ldots, c_n \} \) and with \( sc = (c_1 \& (c_2 \& \cdots c_n \& c)) \),

\[
-(A^I \circ R^n \circ \langle \bot \rangle \in f(s \circ sc)).
\]
Sets that are not acceptor sets are called refusal sets (cf. Hoare et al., 1981). Since refused experiments do not change the state of our processes, our semantic representation and the one using representations from

\[ \text{ACTION}^* \times P(\text{ACTION}) \cup \text{ACTION}^{\infty} \]

are isomorphic. However, our presentation is different from (Hoare et al., 1981) in the use of powerdomains and fixed point theory.

6.2. Powerdomains Revisited

In (Hoare et al., 1981) a fixed point theory is used, that just works in the power domain of demonic nondeterminism with a particular representation (cf. the Smyth power domain). In particular a process \( p \) is represented by

\[ H[p](E)(s) = \{ q : \exists q_1 \in S[p](E)(s) : q_1 \subseteq q \}. \]

This means we work within the powerdomain of demonic nondeterminism using a \( \subseteq \)-maximal representation. So with this semantic representation we can simply use "reverse inclusion ordering" instead of \( \subseteq_M \). In particular we have

\[ S[p_1](E)(s) \subseteq_M S[p_2](E)(s) \quad \text{iff} \quad H[p_2](E)(s) \subseteq H[p_1](E)(s). \]

This explains the introduction of the particular process \texttt{chaos}. The semantics of \texttt{chaos} can easily be defined by

\[ S[\texttt{chaos}](E)(s) = \text{STREAM}([A, R]); \]

i.e., the agent \texttt{chaos} may produce any behavior for an experiment \( s \). In the powerdomain of demonic nondeterminism, \texttt{chaos} is equivalent (we denote this equivalence by \( \sim_n \)) to \texttt{abort}, because

\[
H[\texttt{abort}](E)(s) \\
= \{ q : \exists q_1 \in S[\texttt{abort}](E)(s) : q_1 \subseteq q \} \\
= \{ q : \exists q_1 \in (\bot) : q_1 \subseteq q \} \\
= \text{STREAM}([A, R]).
\]

And this explains some of the surprising algebraic laws that can be found in (Hoare \textit{at al.} 1981, p. 28), where, for instance,

\[ *\text{skip} \sim_n \text{chaos} \]

holds with

\[ *\text{skip} = q :: (\text{skip or (skip; q)}). \]
We have
\[ S[^\text{skip}](E)(s) = \{ e, \bot \} , \]
and thus
\[ H[^\text{skip}] = H[^\text{chaos}] . \]

Other models (cf. Olderog and Hoare, 1982) are based on the powerdomain of angelic nondeterminism. If we choose the representation
\[ O[p](E)(s) = \{ q : \exists q_1 \in S[p](E)(s) : q \subseteq q_1 \} , \]
we simply can use inclusion order instead of \( \sqsubseteq _E \). In particular,
\[ S[p_1](E)(s) \subseteq _E S[p_2](E)(s) \iff O[p_1](E)(s) \subseteq O[p_2](E)(s) . \]

It is a representation that is sufficient for considerations concerning partial correctness. But now we have other funny equations like
\[ p \mid \text{abort} \sim_0 p . \]

Note, that neither \( O \) nor \( H \) are consistent with our operational semantics. A semantic definition based on fixed point theory cannot work with just one power domain if our form of consistency with the operational definitions is required. However, the different congruence relations induced by the different powerdomains correspond nicely to different notions of correctness.

6.3. Notions of Correctness

Every kind of assertion calculi introduces some equivalence relation (in most cases a congruence relation) on the set of program terms. Two programs are considered to be equivalent, iff the same assertions can be deduced for them. For sequential deterministic processes the distinction between partial and total correctness is well known. There partial and total correctness induce precisely the same congruence relation on programs as the fully abstract denotational semantics. Thus the semantic models of both partial and total correctness are isomorphic to the fully abstract denotational model.

For nondeterministic sequential programs this does not hold: calculi of partial and total correctness (cf. wlp-calculus and wp-calculus in (Dijkstra, 1976)) induce distinct congruence relations on program terms. But taking the direct product (or the intersection) of both congruence relations induced by partial and total correctness we obtain a congruence relation which is identical to the one induced by the fully abstract denotational models.
In systems of nondeterministic communicating agents not even this holds. Partial correctness means: "only correct output will appear, but it is not guaranteed whether there is output at all." This clearly indicates that we have only to look at maximal elements (with respect to the ordering on streams) in our set of possible behaviors. So the powerdomain of angelic nondeterminism is the right one for discussing partial correctness.

In analogy to this, total correctness (in the restrictive sense of wp-calculus) means: "we are interested only in that output, that is guaranteed, i.e., where no divergence may occur." This very particular notion of total correctness has only to observe minimal elements in the set of possible behaviors. So it corresponds to the powerdomain of demonic nondeterminism.

The intersection of the two congruence relations yields the congruence relation induced by using the powerdomain of erratic nondeterminism, which does not coincide with the fully abstract denotational semantics. For a more thorough discussion of this issue see (Broy, 1983).

6.4. Other Semantic Concepts

For some of the constructs of CP the intended semantics is intuitively clear and there is not much choice. For others there basically is some freedom to define the semantics. This has to do with the strictness of operations. We call a binary composition σ of agents strict iff for all CP-programs

1. \( abort \sigma p \sim abort \)
2. \( po abort \sim abort \).

If (1) holds σ is called left-strict, if (2) holds then σ is called right-strict; σ is called semi-strict if for all p:

1. \( S[abort](E)(s) \subseteq S[abort \sigma p](E)(s) \)
2. \( S[abort](E)(s) \subseteq S[p o abort](E)(s) \).

and semi-left-strict or semi-right-strict if (1) or (2), resp., hold. Of course every strict function is semi-strict, too. If a construct is not semi-strict it is called nonstrict.

Now we look at our four basic binary constructs in CP:

1. sequential composition expressed by "\( ; \)"
2. parallel composition expressed by "\( ||_c \)"
3. free choice expressed by "\( or \)"
4. joining capabilities expressed by "\( [ \)"

That sequential composition should be left-strict and cannot be right-strict (not even semi-right-strict) is obvious. That or is semistrict is also clear. The definitions for "\( [ \)" and "\( ||_c \)" are less obvious. We have chosen "\( [ \)" to be
Semistrict. It can also be chosen strict or nonstrict (cf. the ambiguity operator in (Broy, 1982)). The same holds for the parallel construct. It can be chosen strict, semistrict, or nonstrict. We decided to choose it semistrict again. Note, that fairness, unbounded nondeterminism, and nonstrictness of nondeterministic language constructs (i.e., "angelicness") are closely related.

Note, that some of the confusions arising when trying to give a denotational style semantics to processes stem from the fact that in the powerdomain of demonic nondeterminism certain strict and semistrict functions are identified, while in the powerdomain of angelic nondeterminism certain semistrict and nonstrict functions are identified.

Since these distinct powerdomains correspond to distinct notions of partial and total correctness, we have in analogy: strict and semistrict functions cannot be distinguished in the restricted notion of total correctness, while semistrict and nonstrict cannot be distinguished using partial correctness.

6.5. Loosely Versus Tightly Coupled Systems

The language CP defines systems of tightly coupled (also called synchronous) agents corresponding to the rendezvous concept. Other language proposals for communicating systems correspond to loosely coupled (asynchronous) systems. The difference between these two concepts can now be made precise in our semantic models.

In a loosely coupled system all language constructs \( c \) correspond to additive functions in the following sense: All language constructs can be defined by

\[
S[c(p)](E) = \bigcup_{i=1}^{n} \{ f_i(x) : x \in S[p](E) \},
\]

where the \( f_i \) are continuous deterministic functions. As can be seen in the definitions above, the synchronized parallel composition or hiding cannot be defined that way.

7. Concluding Remarks

In the previous sections a comprehensive treatment of the semantics of a simple language for concurrent programming has been given. All the techniques used, however, carry over to "more realistic" languages not using just an alphabet of actions but a class of operations that change the environment and/or exchange information between processes.

Although closely related we did not touch the issue of event structures as treated by (Nielsen et al., 1981; Lauer and Shields, 1980; Winkowski, 1980) and many others. However, it seems clear to me that an event structure semantics of CP can also be tackled in the framework as defined above.
APPENDIX I:
SOME DEFINITIONS ON POWERSETS AND POWERS DOMAINS

In this section we give three powerdomain constructions based on the idea of ideal completions. We choose very particular representations for the elements of these powerdomains. A partially ordered set DOM is called countably algebraic domain if

1. every directed set $S \subseteq \text{DOM}$ has a least upper bound (lub),
2. the set of finite approximations of an element $x$ is directed and every element $x \in \text{DOM}$ is lub of the set of its finite approximations
$$x = \text{lub}\{ y \in \text{DOM}: y \subseteq x \text{ and } y \text{ finite} \},$$
3. the set of finite elements is countable.

Here a set $S \subseteq \text{DOM}$ is called directed if
$$\forall x, y \in S \exists z \in S: x \subseteq z \land y \subseteq z.$$  
An element $x \in \text{DOM}$ is called finite, if for all directed sets $S \subseteq \text{DOM}$ we have
$$x \subseteq \text{lub} S \Rightarrow \exists y \in S: x \subseteq y;$$

DOM is called consistently complete, if every set $S \subseteq \text{DOM}$ with an upper bound even, it has a least upper bound. Trivially then every set has a greatest lower bound (glb). Let DOM be a consistently complete, countably algebraic domain; for $S, S_1, S_2 \subseteq \text{DOM}$ we define

$$\text{MIN}(S) = \{ x \in S: \forall y \in S: y \subseteq x \Rightarrow x = y \},$$
$$\text{MAX}(S) = \{ x \in S: \forall y \in S: x \subseteq y \Rightarrow x = y \},$$
$$\text{CLOSE}(S) = \{ x \in \text{DOM}: \exists S_0 \subseteq S:\ ((\forall a, b \in S_0 \exists z \in S_0:\ a \subseteq z \land b \subseteq z) \land x = \text{lub} S_0) \lor ((\forall a, b \in S_0 \exists z \in S_0:\ z \subseteq a \land z \subseteq b) \land x = \text{glb} S_0) \},$$
$$\text{UPC}(S) = \{ x \in \text{DOM}: \exists y \in S: y \subseteq x \}$$
$$\text{DOC}(S) = \{ x \in \text{DOM}: \exists y \in S: x \subseteq y \}$$
$$\text{CONE}(S) = \{ x \in \text{DOM}: \exists y, z \in S: y \subseteq x \subseteq z \}.$$  

$S$ is called convex iff $\text{CONE}(S) = S$; $S$ is called closed iff $\text{CLOSE}(S) = S$. We have

$$\text{CONE}(S) = \text{UPC}(S) \cap \text{DOC}(S)$$
Trivially all these functions and notions on sets can be extended to set-valued functions and functionals over those functions by applying them elementwise.

The following three preorderings are used (cf. Plotkin, 1976; Smyth, 1978):

\[
\begin{align*}
S_1 \sqsubseteq_E S_2 & \iff \forall x \in S_1 \exists y \in S_2 : x \subseteq y, \\
S_1 \sqsubseteq_M S_2 & \iff \forall y \in S_2 \exists x \in S_1 : x \subseteq y, \\
S_1 \sqsubseteq_{EM} S_2 & \iff S_1 \sqsubseteq_E S_2 \land S_1 \sqsubseteq_M S_2.
\end{align*}
\]

Over nonflat (nondiscrete) domains these relations just define preorderings. What sets are identified if we try to make these relations into orderings can be seen from

**Lemma 1.** For closed sets \(S_1, S_2\) we have

\[
\begin{align*}
S_1 \sqsubseteq E S_2 & \iff \text{MAX}(S_1) \sqsubseteq E \text{MAX}(S_2), \\
S_1 \sqsubseteq M S_2 & \iff \text{MIN}(S_1) \sqsubseteq M \text{MIN}(S_2).
\end{align*}
\]

For arbitrary sets \(S_1, S_2\) we have

\[
\begin{align*}
S_1 \sqsubseteq M S_2 & \iff \text{UPC}(S_1) \sqsubseteq M \text{UPC}(S_2), \\
S_1 \sqsubseteq E S_2 & \iff \text{DOC}(S_1) \sqsubseteq E \text{DOC}(S_2), \\
S_1 \sqsubseteq_{EM} S_2 & \iff \text{CONE}(S_1) \sqsubseteq_{EM} \text{CONE}(S_2), \\
S_1 \sqsubseteq M S_2 & \iff \text{UPC}(S_2) \subseteq \text{UPC}(S_1), \\
S_1 \sqsubseteq E S_2 & \iff \text{DOC}(S_1) \subseteq \text{DOC}(S_2).
\end{align*}
\]

This lemma shows one pathological property of the powerdomains based on these "orderings": In a powerdomain particular distinct sets are considered as being equivalent, i.e., the powerdomain constructions actually consider classes of equivalent sets. But sets may not only be equivalent because they cannot be distinguished by the orderings above. Due to the principle of finite approximability and continuity two sets are considered to be equivalent in a countably algebraic powerdomain based on some ordering, iff the classes of finite sets of finite elements that approximate these sets in the sense of these orderings are identical.

Let \(\text{FDOM}\) denote the set of finite elements from \(\text{DOM}\). We take here a very concrete set-theoretic view of powerdomains. Their elements are just represented by subsets of \(\text{P}(\text{DOM})\), i.e., by particular elements of the powerset over \(\text{DOM}\). These representations are chosen in a very particular way which is most convenient for our semantic descriptions. The powerdomain \(\text{PD}(\text{DOM})\) of erratic nondeterminism (also called Plotkin power
domain or Egli–Milner power domain) is defined as follows. The set of finite elements is represented by the convex hull of finite sets of finite elements

\[ GD = \{ \text{CONE}(S) : S \subseteq FDOM \land |S| < \infty \}. \]

We immediately can prove

\[(GD, \sqsubseteq_{EM}) \text{ is po-set.}\]

PD(DOM) is defined as the ideal-completion of \((GD, \sqsubseteq_{EM})\). We choose as representation for PD(DOM) a subset of \(P(DOM)\), such that every ideal \(I \subseteq GD\) is represented by

\[ \{x \in DOM : \forall S1 \in I, y \in FDOM : \]
\[ y \sqsubseteq x \Rightarrow \exists S2 \in I, S1 \sqsubseteq_{EM} S2, z \in S2 : y \sqsubseteq z \sqsubseteq x \}. \]

By

\[ PDOM : P(DOM) \rightarrow PD(DOM) \]

we denote the function mapping every set \(S \subseteq DOM\) on its power domain representation. It is defined by

\[ PDOM(S) = \{x \in DOM : \forall S1 \in GD, y \in FDOM : \]
\[ y \sqsubseteq x \wedge S1 \sqsubseteq_{EM} S \]
\[ \Rightarrow \exists S2 \in GD, S1 \sqsubseteq_{EM} S2 \sqsubseteq_{EM} S, z \in S2 : y \sqsubseteq z \sqsubseteq x \}. \]

Note, that we have chosen a \(\sqsubseteq\)-maximal representation for \(PDOM(S)\), i.e., the \(\sqsubseteq\)-maximal set in the class of sets that are \(\sqsubseteq_{EM}\)-equivalent w.r.t. \(\sqsubseteq_{EM}\)-approximations by finite sets of finite elements. A proof is given in Lemma 2.

The power domain of demonic nondeterminism (also called Smyth power domain) is defined as

\[ GM = \{ \text{UPC}(S) : S \subseteq FDOM \land |S| < \infty \}. \]

One immediately can prove

\[(GM, \sqsubseteq_{M}) \text{ is po-set.}\]

PMO(DOM) is the ideal completion of \((GM, \sqsubseteq_{M})\). We choose as representation for PMO(DOM) a subset of DOM, such that every ideal \(I \subseteq GM\) is represented by

\[ \{x \in DOM : \forall S1 \in I, y \in FDOM : \]
\[ y \sqsubseteq x \Rightarrow \exists S2 \in I, S1 \sqsubseteq_{M} S2, z \in S2 : y \sqsubseteq z \sqsubseteq x \}. \]
By

\[ \text{PMDOMO: } P(\text{DOM}) \rightarrow \text{PMO(\text{DOM})} \]

we denote the function mapping every set \( S \subseteq \text{DOM} \) on its power domain representation. It is defined by

\[ \text{PMDOMO}(S) = \{ x \in \text{DOM} : \forall S_1 \in GM, y \in \text{FDOM}: y \subseteq x \land S_1 \subseteq_M S \Rightarrow \exists S_2 \in GM, S_1 \subseteq_M S_2 \subseteq_M S, z \in S_2 : y \subseteq z \subseteq x \}. \]

All sets in \( \text{PMO(\text{DOM})} \) are closed. Since we find it more convenient to work with a \( \subseteq \)-minimal representation we define

\[ \text{PM(DOM)} = \{ \text{MIN}(S) : S \subseteq \text{PMO(\text{DOM})} \} \]

and

\[ \text{PMDOM: } P(\text{DOM}) \rightarrow \text{PM(DOM)} \]

with

\[ \text{PMDOM}(S) = \text{MIN}(\text{PMDOMO}(S)). \]

Note, that we have chosen a \( \subseteq \)-minimal representation for \( \text{PMDOM}(S) \), i.e., the \( \subseteq \)-minimal set in the class of closed, finitely approximable sets that are \( \subseteq_M \)-equivalent w.r.t. \( \subseteq_M \)-approximations by finite sets of finite elements. A proof is given in Lemma 2.

The power domain of *angelic nondeterminism* (also called Hoare power domain) is defined as

\[ GE = \{ \text{DOC}(S) : S \subseteq \text{FDOM} \land |S| < \infty \}. \]

One immediately can prove

\[ (GE, \subseteq_E) \text{ is po-set.} \]

\( \text{PEO(DOM)} \) is the ideal-completion of \( (GE, \subseteq_E) \). We choose as representation for \( \text{PEO(DOM)} \) a subset of \( \text{DOM} \), such that every ideal \( I \subseteq GE \) is represented by

\[ \{ x \in \text{DOM} : \forall S_1 \in I, y \in \text{FDOM}: y \subseteq x \Rightarrow \exists S_2 \in I, S_1 \subseteq_E S_2, z \in S_2 : y \subseteq z \subseteq x \}. \]

By

\[ \text{PEDOMO: } P(\text{DOM}) \rightarrow \text{PEO(DOM)} \]
we denote the function mapping every set \( S \subseteq \text{DOM} \) on its power domain representation. It is defined by

\[
\text{PEDOMO}(S) = \{ x \in \text{DOM} : \forall S_1 \in \text{GE}, y \in \text{FDOM}: y \subseteq x \land S_1 \subseteq_E S \Rightarrow \exists S_2 \in \text{GE}, S_1 \subseteq_E S_2 \subseteq_E S, z \in S_2 : y \subseteq z \subseteq x \}. 
\]

The sets in \( \text{PEO} (\text{DOM}) \) are closed. We may represent them also by their maximal elements. This leads to a \( \subseteq \)-minimal representation for the power-domain of angelic nondeterminism. We define

\[
\text{PE}(\text{DOM}) = \{ \text{MAX}(S) : S \in \text{PEO}(\text{DOM}) \}
\]

and

\[
\text{PEDOM} : P(\text{DOM}) \to \text{PE}(\text{DOM})
\]

with

\[
\text{PEDOM} = \text{MAX}(\text{PEDOMO}(S))
\]

Note, that we have chosen a \( \subseteq \)-minimal representation for \( \text{PEDOM}(S) \), i.e., the \( \subseteq \)-minimal set in the class of closed, finitely approximable sets that are \( \subseteq_E \)-equivalent w.r.t. \( \subseteq_E \)-approximations by finite sets of finite elements. (For a proof see Lemma 2.)

Basically these powerdomains contain just those sets for which the respective relations form orderings and which can be approximated by finite sets of finite elements. They are isomorphic to (continuous) ideal completions of the representation class of finite sets of finite elements.

For sets \( S_1, S_2 \) we define equivalence-relations:

\[
S_1 \sim_{EM} S_2 \quad \text{iff} \quad \text{PDOM}(S_1) = \text{PDOM}(S_2)
\]

\[
S_1 \sim_M S_2 \quad \text{iff} \quad \text{PMDOM}(S_1) = \text{PMDOM}(S_2)
\]

\[
S_1 \sim_e S_2 \quad \text{iff} \quad \text{PEDOM}(S_1) = \text{PEDOM}(S_2).
\]

These equivalence relations can also be described in another way according to the following lemma.

\textbf{Lemma 2.} \textit{For} \( S_1, S_2 \subseteq \text{DOM} \) \textit{we have}

\begin{enumerate}
\item \( S_1 \sim_{EM} S_2 \) \text{iff} \( \forall S \subseteq \text{FDOM}, |S| < \infty : (S \subseteq_{EM} S_1 \leftrightarrow S \subseteq_{EM} S_2) \)
\item \( S_1 \sim_M S_2 \) \text{iff} \( \forall S \subseteq \text{FDOM}, |S| < \infty : (S \subseteq_M S_1 \leftrightarrow S \subseteq_M S_2) \)
\item \( S_1 \sim_e S_2 \) \text{iff} \( \forall S \subseteq \text{FDOM}, |S| < \infty : (S \subseteq_E S_1 \leftrightarrow S \subseteq_E S_2) \)
\item \( \text{PDOM}(S) = \bigcup \{ SO \subseteq \text{DOM} : S \sim_{EM} SO \} \)
\item \( \text{PMDOM}(S) = \text{MIN}(\bigcup \{ SO \subseteq \text{DOM} : S \sim_{M} SO \}) \)
\item \( \text{PEDOM}(S) = \text{MAX}(\bigcup \{ SO \subseteq \text{DOM} : S \sim_{E} SO \}) \).
\end{enumerate}
Proof. (1) + (2) + (3) are trivial in one direction, since in the definitions of PDOM, PMDOM, PEDOM it can simply be seen that PDOM(S) (and PMDOM(S), PEDOM(S), resp.) only depends on the set of finite sets SO of finite elements with SO ⊆_{EM} S. Now assume that S1 ⊆_{EM} S2 and there is some S ⊆ FDOM, |S| < ∞ with S ⊆_{EM} S1 but ¬(S ⊆_{EM} S2). Then either there exists exists some x ∈ S such that there does not exist y2 ∈ S2 with x ⊆ y2 and there exists some y1 ∈ S1 with x ⊆ y1; then y1 ∈ PDOM(S1) but ¬(y1 ∈ PDOM(S2)); or these exists some y2 ∈ S2 such that there does not exist x ∈ S with x ⊆ y2 and thus y2 ∈ PDOM(S2) but ¬(y2 ∈ PDOM(S1)). The proofs of (2) and (3) are analogous.

(4) If x ∈ SO for some SO ⊆_{EM} S, then for every pair of approximations y ⊆ x, S1 ⊆_{EM} S (and thus S1 ⊆_{EM} SO) with S1 ⊆ GD, y ∈ FDOM there exist approximations S2 ∈ GD with S1 ⊆_{EM} S2 ⊆_{EM} S (and thus S2 ⊆_{EM} SO) with z ∈ S2 such that y ⊆ z ⊆ x; hence x ∈ PDOM(S).

Now if x ∈ PDOM(S), then S ⊆_{EM} S ∪ {x} and thus x ∈ ∪ {S1 ⊆ DOM: S ⊆_{EM} S1}.

(5) In analogy to (4).

(6) In analogy to (5).

The concept of finite observability over algebraic domains simply means, that two objects are equal iff their classes of finite approximations are identical.

Lemma 3. Figure 4 commutes. In particular MIN and MAX are continuous functions.

Proof. Lemma 3 is a corollary of Lemma 2.

On function domains we use the classical ordering. Given a domain D ordered by ⊆, then the set of functions

\[ \{ f: D_2 \to D \} \]
with some given set (or domain) $D_2$ can be simply ordered by

\[ f_1 \sqsubseteq^* f_2 \iff \forall x \in D_2: f_1(x) \sqsubseteq f_2(x). \]

Analogously we write $\text{lub}^*$ for the lub on the function domain ordered by $\sqsubseteq^*$.

Unfortunately the simple powerset without the empty set ordered by inclusion ordering does not form a domain. For very obvious reasons we do not accept the empty set as element, since the set of possible computations of a nondeterministic program can never be empty. However $(P(DOM) \setminus \{\emptyset\}, \subseteq)$ forms a predomain, i.e., it has all properties of a domain besides the existence of a least element. We restrict ourselves to closed sets, i.e., to sets $S$ where with every directed set in $S$ its least upper bound is also in $S$. This is motivated by the concept of finite observability. Every object should be determined by its finite approximations.

Accordingly the **power predomain of closed sets** is defined as

\[ PC(DOM) = \{S \subseteq DOM: S = \text{CLOSE}(S)\}. \]

A function

\[ f: P(DOM) \to P(DOM) \]

is called closely union continuous iff

\[ f(\text{CLOSE}(\bigcup x_i)) = \text{CLOSE}(\bigcup f(x_i)) \]

for every $\subseteq$-chain $\{x_i\}_{i \in \mathbb{N}}$, $x_i \in P(DOM)$. Similarly a functional

\[ T: (DOM \to P(DOM)) \to (DOM \to P(DOM)) \]

is called closely union continuous iff

\[ T[\text{CLOSE} \cdot (\bigcup* f_i)] = \text{CLOSE} \cdot (\bigcup* T[f_i]) \]

for every $\subseteq$-chain $(f_i)_{i \in \mathbb{N}}$, $f_i: DOM \to P(DOM)$. Here $\cdot$ denotes the composition of functions and $\bigcup* f_i$ denotes the elementwise union of the set-valued function $f_i$, i.e.,

\[ (\bigcup* f_i)(x) =_{\text{def}} \bigcup f_i(x). \]
APPENDIX II: PROOFS

THEOREM. For every CP-program p and every perfect environment E, we have

1. the functional $T_{q: p}(E)$ is $\subseteq^k$-monotonic and closely union continuous,
2. the functional $\text{PDOM} \cdot T_{q: p}(E)$ is $\subseteq_{EM}^k$-monotonic and $\subseteq_{EM}^k$-continuous,
3. the functional $\text{PMDOM} \cdot T_{q: p}(E)$ is $\subseteq_k$-monotonic and $\subseteq_k$-continuous,
4. the functional $\text{PEDOM} \cdot T_{q: p}(E)$ is $\subseteq_{E}^k$-monotonic and $\subseteq_{E}^k$-continuous.

Proof. All language constructs are mapped onto functions that can be represented in a very specific form, i.e., for a given environment they are functionals

$$T: \text{PROCESS}^i \rightarrow \text{PROCESS},$$

with $i = 1, 2$, where (let now w.l.o.g. $i = 2$),

$$T[x, y](s) = \{h(q_1, q_2, d, s): q_1 \in x(g_1(s, d)) \land q_2 \in y(g_2(s, d)) \land d \in Z \land r(q_1, q_2, d, s) \subseteq \text{true}\},$$

where

$h: \text{STREAM}(\{A, R\})^2 \times \text{STREAM}(H) \times \text{STREAM}(\text{ACTION}) \rightarrow \text{STREAM}(\{A, R\}),$

$r: \text{STREAM}(\{A, R\})^2 \times \text{STREAM}(H) \times \text{STREAM}(\text{ACTION}) \rightarrow \text{BOOL}^\lambda,$

$g_1, g_2: \text{STREAM}(H) \times \text{STREAM}(\text{ACTION}) \rightarrow \text{STREAM}(\text{ACTION})$

are continuous functions, $Z \in \text{PDOM}(\text{STREAM}(H))$, $H$ is a finite set, and the predicate $r$ is defined in a way, such that for every $d \in Z$ and all $\subseteq_{EM}$ continuous processes $x_1, x_2, y_1, y_2$, with

$$x_1 \subseteq_{EM} x_2, \quad y_1 \subseteq_{EM} y_2$$

and all experiments $s_1, s_2$ with $s_1 \subseteq s_2$ for all

$q_1 \in x_1(g_1(s_1, d)), \quad q_2 \in y_1(g_2(s_1, d)),$

$q_3 \in x_2(g_1(s_2, d)), \quad q_4 \in y_2(g_2(s_2, d))$
with \( q_1 \subseteq q_3 \) and \( q_2 \subseteq q_4 \) there exists \( d' \in S \) with

\[
g_1(s_1, d) = g_1(s_1, d'), \quad g_2(s_1, d) = g_2(s_1, d')
\]

(thus \( q_1 \in x(g_1(s, d')) \), \( q_2 \in (g_2(s, d')) \)), and

\[
r(q_1, q_2, d', s_1) \sqsubseteq \text{true} \implies r(q_3, q_4, d', s_2) \sqsubseteq \text{true},
\]

Moreover \( T[x, y](s) \) is never empty.

Intuitively this property just says that taking a "more defined" process and continuing the experiment there exists at least one more (or equally) defined behavior.

\( \sqsubseteq^* \)-monotonicity. If

\[
x_1 \sqsubseteq^* x_2, y_1 \sqsubseteq^* y_2
\]

(i.e., for all \( s \): \( x_1(s) \subseteq x_2(s), y_1(s) \subseteq y_2(s) \)) then trivially

\[
T[x_1, y_1] \sqsubseteq^* T[x_2, y_2].
\]

Close union continuity. Now let \( \{x_i\}_{i \in N}, \{y_i\}_{i \in N} \) be \( \subseteq \)-chains with \( x_i, y_i \in \text{PROCESS} \), and

\[
x = \bigcup^* x_i, \quad y = \bigcup^* y_i.
\]

If \( q \in \text{CLOSE}(T[x, y](s)) \), then there exist \( q_i \in T[x_i, y_i](s) \), where

\[
\{q_i\}_{i \in N}
\]

forms a \( \subseteq \)-chain

with \( q = \text{lub} \{q_i\} \). Thus for every \( i \) there are \( j \) with

\[
q_i \in T[x_j, y_j](s);
\]

hence \( q_i \in \bigcup T[x_j, y_j](s) \) and so

\[
q \in \text{CLOSE}(\bigcup T[x_j, y_j](s)).
\]

\( \sqsubseteq_{EM}^* \)-monotonicity. Assume

\[
x_1 \sqsubseteq^*_{EM} x_2, \quad y_1 \sqsubseteq^*_{EM} y_2
\]

(i.e., for all \( s \): \( x_1(s) \subseteq_{EM} x_2(s), y_1(s) \subseteq_{EM} y_2(s) \)). For

\[
qx_1 \in x_1(g_1(s, d)), \quad qy_1 \in y_1(g_2(s, d)),
\]

there exist

\[
qx_2 \in x_2(g_1(s, d)), \quad qy_2 \in y_2(g_2(s, d))
\]
with
\[ qx_1 \subseteq qx_2, \quad qy_1 \subseteq qy_2. \]

According to the monotonicity of \( h \) we have
\[ h(qx_1, qy_1, d, s) \subseteq h(qx_2, qy_2, d, s) \]
and according to the properties of \( r \) we can choose \( d \) such that
\[ h(qx_1, qy_1, d, s) \in T[x_1, y_1](s) \]
\[ \Rightarrow h(qx_2, qy_2, d, s) \in T[x_2, y_2](s). \]

Thus \( T \) must be \( \subseteq_E \)-monotonic. If
\[ qx_2 \in x_2(g_1(s, d)), \quad qy_2 \in y_2(g_2(s, d)), \]
then there exist
\[ qx_1 \in x_1(g_1(s, d)), \quad qy_1 \in y_1(g_2(s, d)) \]
with
\[ qx_1 \subseteq qx_2, \quad qy_1 \subseteq qy_2. \]

According to the monotonicity of \( h \) we have
\[ h(qx_1, qy_1, d, s) \subseteq h(qx_2, qy_2, d, s) \]
and according to the properties of \( r \) we can choose \( d \) such that
\[ h(qx_2, qy_2, d, s) \in T[x_2, y_2](s) \]
\[ \Rightarrow h(qx_1, qy_1, d, s) \in T[x_1, y_1](s). \]

Thus \( T \) must be \( \subseteq_M \)-monotonic. From \( \subseteq_M \)-monotonicity and \( \subseteq_E \)-monotonicity together we obtain \( \subseteq_{EM} \)-monotonicity.

\( \subseteq_{EM} \) continuity. Now let \( \{x_i\}_{i \in \mathbb{N}}, \{y_i\}_{i \in \mathbb{N}} \) be \( \subseteq_{EM} \)-chains with \( x_i, y_i \in \text{PROCESS} \), and

\[ x(s) = \subseteq_{EM}\text{ lub } \{\text{PDOM}(x_i(s))\}, \quad y(s) = \subseteq_{EM}\text{ lub } \{\text{PDOM}(y_i(s))\}. \]

The \( \subseteq_{EM} \)-monotonicity immediately gives
\[ \subseteq_{EM}\text{ lub } \{\text{PDOM}(T[x, y](s))\} \subseteq_{EM} \text{PDOM}(T[x, y](s)). \]
If

\[ q \in \text{PDOM}(T[x, y](s)) \]

then there exist

\[ d \in \mathbb{Z}, qx \in \text{PDOM}(x(g_1(s, d))), qy \in \text{PDOM}(y(g_2(s, d))) \]

with

\[ q = h(qx, qy, d, s), \quad r(qx, qy, d, s) \subseteq \text{true}. \]

Then there exist

\[ qx_i \in \text{PDOM}(x_i(g_1(s, d))), \quad qy_i \in \text{PDOM}(y_i(g_2(s, d))) \]

with

\[ q_i = h(qx_i, qy_i, d, s), \quad r(qx_i, qy_i, d, s) \subseteq \text{true}. \]

and

\[ \text{lub}\{qx_i\} = qx, \quad \text{lub}\{qy_i\} = qy. \]

We have

\[ q_i \in T[x_i, y_i](s) \quad \text{with} \quad q = \text{lub}\{q_i\}. \]

Thus

\[ q \in \sqsubseteq_{\text{EM}} \text{lub} \text{PDOM}(T[x_i, y_i](s)). \]

If

\[ q \in \sqsubseteq_{\text{EM}} \text{lub} \text{PDOM}(T[x_i, y_i](s)), \]

then there exist

\[ q_i \in \text{PDOM}(T[x_i, y_i](s)) \]

with

\[ q = \text{lub}\{q_i\}. \]

Then according to the continuity of \( h, r, g_1, \) and \( g_2, \)

\[ q(s) = \text{lub}\{q_i(s)\} \in T[\sqsubseteq_{\text{EM}} \text{lub}^* \{\text{PDOM} \cdot x_i\}, \sqsubseteq_{\text{EM}} \text{lub}^* \{\text{PDOM} \cdot y_i\}](s) \]
and so
\[ T[x, y](s) \subseteq EM \subseteq EM^{lub}\{PDOM(T[x, y](s))\}. \]

Thus we obtain
\[ PDOM(T[x, y](s)) = EM^{lub}\{PDOM(T[x, y](s))\}. \]

Trivially then \( PMDOM \cdot T = \text{MIN} \cdot PDOM \cdot T \) and \( PDOM \cdot T = \text{MAX} \cdot PDOM \cdot T \) are \( \subseteq_{\text{mon}} \)-monotonic and \( \subseteq_{\text{cont}} \)-continuous and \( \subseteq_{\text{max}} \)-monotonic and \( \subseteq_{\text{cont}} \)-continuous, respectively, since \( \text{MIN} \) and \( \text{MAX} \) are monotonic and continuous functions. \]

**Theorem.** For every CP-program \( p \) we have
\[ \text{cap}(p) = \text{CAP}(S[p](E_{\bot})), \]
i.e., \( S[p](E_{\bot}) \) is extensionally equivalent to the operational semantics and \( S \) is consistent.

**Proof.** We prove the theorem by structural induction on the structure of the terms of the syntactic unit (agent). For the nonrecursive agents this can be done straightforwardly by checking the consistency of the definition of the relation defining the operational semantics with the definition of \( S \).

Let now \( q::p \) be an agent where no recursively defined agents occur in \( p \). Define
\[ q_0 = p[\text{abort}/q], \quad q_{i+1} = p[q_i/q]. \]

Let furthermore \( T_{q::p} \) be defined as in Section 4.3. According to the construction there we have
\[ \text{MIN}(\text{cap}(q::p)) = \text{MIN}(\text{CAP}(S[q::p](E_{\bot}))) = \text{CAP}(\text{MIN} \cdot S[q::p](E_{\bot})), \]
since for all experiments \( s \),
\[ \text{MIN}(S[p::q](E)(s)) = \text{MIN}(S[p::q](E)(s)) = \subseteq_{\text{mon}}^{lub}\{\text{MIN}(S[q_i](E)(s))\}, \]
and
\[ S[q_i](E)(s) = T_{q::p}^i(E_{\bot})[p](s), \]
and
\[ \text{MIN}(\text{cap}(q::p)) = \subseteq_{\text{mon}}^{lub}\{\text{MIN}(\text{cap}(q_i))\}. \]

Moreover the fixed point property of \( \text{cap}(q::p) \) immediately gives then
\[ \text{CAP}(S[q::p](E_{\bot})) \subseteq \text{cap}(q::p). \]
If \( s \in \text{cap}(q::p) \) is finite, then by induction on the length of \( s \) one easily proves \( s \in \text{CAP}(S[p](E_\bot)) \). If \( s \) is infinite, then \( s \) is maximal, i.e., \( s \in \text{CAP}(S[q::p](E_\bot)) \) iff \( s \in \text{PDOM}(\text{CAP}(S[q::p](E_\bot))) \). For reasons of \( \subseteq_{E_{EM}} \)-continuity we have

\[
\text{cap}(q::p) \in \text{PDOM}(\text{CAP}(S[q::p](E_\bot)));
\]

so \( s \in \text{CAP}(S[q::p](E_\bot)) \).

**Theorem.** The denotational semantics defined by \( S_1 \) is fully abstract.

**Proof.** Assume \( p_1, p_2 \) are agents with \( P_1 = S_1[p_1](E_\bot), P_2 = S_1[p_2](E_\bot) \), \( P_1, P_2 \in \text{PROCESS} \) with \( P_1 \neq P_2 \). Then there exists some experiment \( s \in \text{STREAM}(\text{ACTION}) \) with (w.l.o.g) \( q \in P_1(s) \) but \( \neg(q \in P_2(s)) \). Let (w.l.o.g) \( s \) be finite with

\[
s = s_0 \& \cdots \& s_n \& \varepsilon.
\]

Let furthermore \( c \) be the set of actions occurring in \( s \). If

\[
q = A^i,
\]

then

\[
s_0 \& \cdots \& s_i \& \varepsilon \in \text{cap}(p_1),
\]

but

\[
\neg(s_0 \& \cdots \& s_i \& \varepsilon \in \text{cap}(p_2)).
\]

If

\[
q = A^i \circ \langle \bot \rangle,
\]

then

\[
s_0 \& \cdots \& s_i \& \langle \bot \rangle \in \text{cap}(p_1),
\]

but

\[
\neg(s_0 \& \cdots \& s_i \& \langle \bot \rangle \in \text{cap}(p_2)).
\]

If

\[
q = A^i \circ R^i,
\]

then consider the agent \( C[p] \):

\[
p \parallel_c (s_0 \rightarrow \cdots \rightarrow s_i \rightarrow ((s_{i+1} \rightarrow \text{skip}) \| \cdots \| (s_{i+j} \rightarrow \text{skip})).
\]
Obviously

\[ s_0 \land \cdots \land s_l \land \langle \bot \rangle \in \text{cap}(C[p1]) \],

but

\[ \neg(s_0 \land \cdots \land s_l \land \langle \bot \rangle \in \text{cap}(C[p2])). \]

**ACKNOWLEDGMENTS**

It is a pleasure to acknowledge a number of stimulating discussions with Tony Hoare. The idea of using the concept of predomains for explaining the fixed point construction resulted from a discussion with John Reynolds. I am grateful to Frederike Nickl and Thomas Pinegger for reading drafts.

RECEIVED February 28, 1983; ACCEPTED August 1984

**REFERENCES**


