Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

J.J.M.M. Rutten

Semantic correctness for a parallel object-oriented language

Computer Science/Department of Software Technology Report CS-R8843 October

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 Pure Research (Z.W.O.).

Semantic Correctness for a Parallel Object-Oriented Language

J.J.M.M. Rutten
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

Different semantic models are studied for a language called POOL: a parallel object-oriented language. It is a simplified version of POOL-T, a language that is actually used to write programs for a parallel machine. The most important aspect of this language is that it describes a system as a collection of communicating objects that all have internal activities which are executed in parallel. For POOL, an operational and a denotational semantics have been developed previously. The former semantics aims at the intuitive operational meaning of the language, whereas the main characteristic of the latter is compositionality. In this paper, we relate both models, which are quite different, and prove the semantic correctness of the denotational semantics with respect to the operational semantics. Our semantic investigations take place in the mathematical framework of complete metric spaces. For the operational semantics we use a simple space of functions from states to compact sets of streams (which are sequences of states); for the denotational semantics, a domain of processes is used, which is the solution of a reflexive domain equation over a category of complete metric spaces. The main mathematical tool we use is Banach's theorem, which states that contractions on complete metric spaces have unique fixed points. Both the operational and the denotational semantics are reformulated and are presented, as well as many operators on the semantic domains, as the fixed point of a suitably defined contraction. In this way, we are able to establish a formal equivalence between both models. For this purpose, we introduce an intermediate domain, which first is compared to the operational model by means of an abstraction operator. This function takes processes, which are tree-like structures, as arguments and yields sets of streams as results. Next, it is shown that both the intermediate and the denotational model are fixed points of the same contraction, from which their equality follows. From both facts, the main result of our study follows: The operational meaning of a POOL program is equal to the denotational meaning to which the abstraction operator is applied. In this manner, the correctness of the denotational semantics with respect to the operational semantics is established.

1980 Mathematical Subject Classification: 68B10, 68C01.
1986 Computing Reviews Categories: D.3.1, F.3.2, F.3.3.
Key words and phrases: operational semantics, denotational semantics, process creation, object-oriented programming, semantic correctness, complete metric spaces, contractions.

1. Introduction

We study different semantic models for a language called POOL: parallel object-oriented language. Although the theoretical foundations of object-oriented programming in general, and of *parallel* object-oriented programming in particular, have not been paid much attention to, this language has been extensively studied in a formal semantic context: In [ABKR86(a)] and [ABKR86(b)], an operational and a denotational semantics of POOL have been developed. The main goal of this paper is to compare the two models, which are quite different, by proving some formal relation between them, which at the same time will establish the correctness of the denotational semantics with respect to the operational semantics. Before we explain in some detail the language POOL and the contents of this paper, we first give a short explanation of the notion of semantic correctness and the way it can be proved.

A semantics for a programming language $\mathfrak{L}$ is a mapping $\mathfrak{M}:\mathfrak{L}\rightarrow D$, where D is some mathematical domain (a set, a complete partial ordering, a complete metric space), which we call the semantic *universe* of $\mathfrak{M}$. Sometimes $\mathfrak{M}$ is called a model for $\mathfrak{L}$. Traditionally, two main types of semantics are distinguished: *operational* semantics and *denotational* semantics. Without wanting to get involved in a discussion about the precise definitions, we state that in our view the main characteristic of the former is that its definition is based on a *transition relation* ([HP79], [Pl81], [Pl83]); a denotational semantics

(*) This work was carried out in the context of ESPRIT project 415: Parallel Architectures and Languages for AIP — a VLSI-directed approach.

Report CS-R8843
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

is characterised by the fact that it is defined in a *compositional* manner: the denotational semantics of a composite statement is given in terms of the denotational semantics of its components. (As a second distinctive property one often considers the way in which recursion is treated: The usual view is that an operational semantics treats recursion by means of so-called *syntactic environments* (or body replacement) whereas a denotational semantic uses *semantic environments*, in combination with some fixed-point argument.)

Now consider an operational semantics $\mathcal{O}:\mathcal{L}\to D$ and a denotational semantics $\mathcal{D}:\mathcal{L}\to D'$. A natural question is whether $\mathcal{D}$ is *correct* with respect to $\mathcal{O}$, that is, whether $\mathcal{D}$ makes *at least* the same distinctions as $\mathcal{O}$ does. (Often, $\mathcal{D}$ makes more; see [KR88] for a simple example.) If we define for a semantics $\mathcal{M}:\mathcal{L}\to D''$ an equivalence relation $\equiv_{\mathcal{M}}$ by

$$s \equiv_{\mathcal{M}} t \Leftrightarrow \mathcal{M}[\![s]\!] = \mathcal{M}[\![t]\!],$$

for all $s, t \in \mathcal{L}$, then the correctness of $\mathcal{D}$ with respect to $\mathcal{O}$ can be formally expressed by the condition:

$$\equiv_{\mathcal{D}} \subset \equiv_{\mathcal{O}}.$$

One way to prove the correctness of $\mathcal{D}$ is to introduce a so-called *abstraction* operator $\alpha:D'\to D$, which (is in general not injective and) relates the denotational semantic universe with the operational one. If one can prove that

$$\mathcal{O} = \alpha \circ \mathcal{D}$$

then a precise relation between $\mathcal{O}$ and $\mathcal{D}$ has been established, which moreover implies the correctness of $\mathcal{D}$ with respect to $\mathcal{O}$.

As a mathematical framework for our semantic descriptions we have chosen *complete metric spaces*. (For the basic definitions of topology see [Du66] or [En77].) In this we follow and generalize [BZ82]. (For other applications of this type of semantic framework see [BKMOZ86].) We follow [KR88] in using contractions on complete metric spaces as our main mathematical tool, exploring the fact that contractions have *unique* fixed points (Banach's theorem). We shall define both operators on our semantic universes and the semantic models themselves as fixed points of suitably defined contractions. In this way, we are able to use a general method for proving semantic correctness: Suppose we have defined $\mathcal{O}$ as the fixed point of a contraction

$$\Phi: (\mathcal{L}\to D)\to(\mathcal{L}\to D).$$

If we next show that also $\alpha\circ\mathcal{D}$ is a fixed point of Φ then Banach's theorem implies that $\mathcal{O}=\alpha\circ\mathcal{D}$.

It is the approach sketched above that will be applied to the language POOL. Before doing so, we start in section 2 with a toy language that is extremely simple but has with POOL in common a construct for process creation. This section can be seen as a prolongation of the introduction and tries to give the reader some feeling for the techniques used. Since no definitions or results of this section are used in the other sections it can be skipped without any problem.

The language POOL is described in detail in section 3. It is a simplified version of the language POOL-T, which is defined in [Am85] and for which [Am86] and [Am87] give an account of the design considerations. POOL-T was designed in subproject A of ESPRIT project 415 with the purpose of programming a highly parallel machine which is also being developed in this project (see [Od87] for an overview). The language provides all the facilities needed to program reasonably large parallel systems and several large applications and many small ones have been written in it.

In POOL, a system is viewed as a collection of *objects*. These are dynamic entities containing *data* (stored in *variables*) and *methods* (a kind of procedures). Objects can be created dynamically during the execution of a program and each of them has an internal activity (its *body*) in which it can execute expressions and statements. While inside an object everything proceeds sequentially, the concurrent execution of the bodies of all the objects can give rise to a large amount of parallelism. Objects can interact by sending *messages* to each other. Acceptance of a message gives rise to a rendez-vous between sender and receiver, during which an appropriate method is executed.

In section 4, we follow [ABKR86(a)] in defining an operational semantics for POOL. It is based on a transition relation and is given, and here we differ from [ABKR86(a)], as the fixed point of a contraction. The semantic domain used is a complete metric space of (functions from states to) compact sets of streams, which are sequences of states.

In section 5, we present a denotational semantics for POOL, very similar to the model given in [ABKR86(b)]. We define a mapping from the set of POOL programs (called *units*) to some reflexive domain of processes $\overline{P}$ (cf. [Pl76]), which is a complete metric space with tree-like structures for its elements. It satisfies a reflexive domain equation, which is solved by deriving from it a functor on a category of complete metric spaces and then taking the fixed point of this functor. The mathematical techniques to do so are sketched in section 2 of [ABKR86(b)] and presented in detail in [AR88]. Before we assign a semantic value to the unit as a whole, we first define the semantics of expressions and statements, which will be given by functions of the following type:

$$\mathcal{D}_E\colon L_E \to AObj \to Cont_E \to \overline{P}, \quad \text{and} \quad \mathcal{D}_S\colon L_S \to AObj \to Cont_S \to \overline{P},$$

where L_E and L_S are the sets of expressions and statements and

$$Cont_E = Obj \to \overline{P}, \quad Cont_S = \overline{P}.$$

The semantic domain $AObj$ stands for the set of (active) object names. Its appearance in the semantics of expressions and statements reflects the fact that in POOL each expression or statement is evaluated *by a certain object.* Further, a *continuation* will be given as an argument to the semantic functions. This describes what will happen *after* the execution of the current expression or statement. As the continuation of an expression generally depends upon the result of this expression (an object name), its type is $Obj \to \overline{P}$, whereas the type of continuations of statements is simply $\overline{P}$. The use of continuations makes it possible to define the semantics, especially of object creation, in a convenient and concise way. (For more examples of the use of continuations in semantics, see [Br86] and [Go79].)

After having defined an operational and a denotational semantics for POOL, we come to the main subject of our paper: The comparison of both models. This constitutes a non-trivial problem, mainly because, first, the respective semantic domains are very different and, secondly, because the denotational semantics is defined in terms of continuations, whereas the operational semantics is direct, that is, does not use continuations. Moreover, the communication mechanism of POOL (consisting of message passing with method invocation) is dealt with quite differently by the two models. The solution that we propose consists of the introduction of an intermediate semantic model, in section 6, which has in common with the operational semantics that it is direct (without continuations) and that it is based on the same transition relation, but which has for its range the same reflexive domain of processes as the denotational model has. Then, in section 7, this intermediate model is related to the operational semantics by means of an abstraction operator which takes processes as arguments and yields sets of streams. Next, it is connected with (an extended version of) the denotational semantics by the observation that both models are fixed point of the same contraction. As a result, it follows that the operational semantics of a unit equals its denotational meaning to which the abstraction operator is applied.

Section 8, which contains the references, is followed by three appendices. Appendix I gives the mathematical definitions we use; in appendix II, the abstraction operator that is used in the proof of the semantic correctness for POOL is defined in all formal detail. Finally, appendix III shows how the language POOL can be extended with so-called *standard* objects and how the definitions and proofs can be adapted in order to obtain a similar correctness result for the extended language.

Semantic treatments of parallel object-oriented languages in general are scarce; we only know [Cl81], which gives a detailed mathematical model for an actor language. This is done by defining a set of so-called augmented actor event diagrams, each of which is a fairly complicated structure representing (the beginning of) a single computation. In order to deal with nondeterminism, a novel power domain construction is used. As to the comparison of operational and denotational semantics

for languages with process creation, we only know of [AB88], where some simplified versions of POOL are studied. None of these languages, however, contains the original POOL-T constructs for communication (for message passing with method invocation), the treatment of which, in the correctness proof, we consider to be an essential part of this paper.

ACKNOWLEDGEMENTS: We wish to thank Pierre America for his detailed and constructive comments on preliminary versions of this paper. Discussions with Jaco de Bakker are gratefully acknowledged, as well as the contributions of the Amsterdam Concurrency Group: Jaco de Bakker, Frank de Boer, Arie de Bruin, Joost Kok, John-Jules Meyer and Erik de Vink. We thank Mini Middelberg for the expert typing of this document.

2. A VERY SIMPLE LANGUAGE WITH PROCESS CREATION

Before we tackle the main problem of this paper, we would like to start with a much simpler case: We introduce a very small "toy" language L_T and present an operational and a denotational semantics for it. Next, we shall compare these two models. All this can be regarded as a little exercise, a "warming up" so to speak, aiming at a better understanding of what follows in the next section: It turns out that for both the languages L_T and POOL (to be introduced in the next section) the operational and denotational semantics can be compared in very much the same way.

For the definition of L_T we need a set $(a,b \in)A$ of *elementary actions*. (Throughout this paper, we shall use the notation $(x, y \in)X$ for the introduction of a set X with typical elements x and y.) For A we take an arbitrary, possibly infinite, set. It will contain a subset $(c \in)C \subseteq A$ of so-called *communications*. Similarly to CCS ([Mil80]), we define a bijection $^- : C \to C$ with $^- \circ ^- = id_C$. It yields for every $c \in C$ a matching communication $\overline{c}$. In $A \setminus C$ we have a special element τ denoting successful communication.

DEFINITION 2.1 (Syntax for L_T)
The set of statements $(s,t \in)L_T$ is given by

$$s ::= a \mid s_1;s_2 \mid \mathbf{new}(s).$$

Note that $a \in A \supseteq C$. To L_T we add a special element E, denoting the *empty* statement. Note that syntactic constructs like $s; E$ and $\mathbf{new}(E)$ are *not* in L_T.

A statement is of one of the following forms: First, it can be an elementary action a. Here elementary means that it is an uninterpreted action. Examples of possible interpretations are assignments, or read and write actions. Secondly, a statement s can be the sequential composition $s_1; s_2$ of statements s_1 and s_2. Finally, it may be a new-statement $\mathbf{new}(s)$, the execution of which amounts to the creation of a new process which executes s. A more detailed explanation will follow below.

The operational semantics will be formulated using the notion of *parallel statements*. A parallel statement is a finite sequence of statements which are to be executed in parallel.

DEFINITION 2.2 (Parallel statements)
Let $(\rho,\pi \in)Par$ be given by $Par = (L_T)^*$, the set of finite sequences of statements. Typical elements will also be indicated by $<s_1, \ldots, s_n>$, for $n \geq 1$. For $\rho = <s_1, \ldots, s_n>$ and $\pi = <t_1, \ldots, t_m>$ we define $\rho^\wedge\pi = <s_1, \ldots, s_n, t_1, \ldots, t_m>$.

Next we define the operational semantics of parallel statements. It is based on the well known notion of a *transition relation* (in the style of Hennessy and Plotkin ([HP79, Pl81, Pl83])).

DEFINITION 2.3 (Transition relation for *Par*)
Let $\to \subseteq Par \times A \times Par$ be the smallest relation (writing $\rho - a \to \rho'$ for $(\rho, a, \rho') \in \to$) satisfying:

(1) $<a>-a\rightarrow<E>$, $<a;s>-a\rightarrow<s>$

(2) if $<s>-a\rightarrow\rho$, then $<\mathbf{new}(s)>-a\rightarrow\rho$

(3) if $<s,t>-a\rightarrow\rho$, then $<\mathbf{new}(s);t>-a\rightarrow\rho$

(4) if $<s_1;(s_2;s_3)>-a\rightarrow\rho$, then $<(s_1;s_2);s_3>-a\rightarrow\rho$

(5) if $\rho-a\rightarrow\rho'$, then $\rho^\wedge\pi-a\rightarrow\rho'^\wedge\pi$ and $\pi^\wedge\rho-a\rightarrow\pi^\wedge\rho'$

(6) if $\rho-c\rightarrow\rho'$ and $\pi-\overline{c}\rightarrow\pi'$, then $\rho^\wedge\pi-\tau\rightarrow\rho'^\wedge\pi'$,

for $a\in A$, $c\in C$, $s,t,s_1,s_2,s_3\in L_T$, and $\rho,\rho',\pi,\pi'\in Par$.

Intuitively, $\rho-a\rightarrow\rho'$ tells us that starting in the parallel statement ρ the elementary action a can be performed, resulting in the parallel statement ρ'. Interesting in the definition above are (3), (5) and (6). According to (3), the parallel statements $<s,t>$ and $<\mathbf{new}(s);t>$ can perform the same elementary actions. In other words, evaluating $<\mathbf{new}(s);t>$ results in a parallel statement $<s,t>$. Thus we see that the length of a parallel statement increases when $\mathbf{new}(s)$ is evaluated. Operationally, this can be viewed as the creation of a process that starts evaluating s, while statement t is being executed in parallel. According to (5), a composite parallel statement $\rho^\wedge\pi$ can perform all the elementary actions that can be performed by either ρ or π. In (6) it is expressed that if ρ can perform a communication action c and π can perform a matching communication action $\overline{c}$, then $\rho^\wedge\pi$, the parallel statement composed of ρ and π, can perform a τ action, denoting a successful communication.

EXAMPLE: $<\mathbf{new}(c);a;\mathbf{new}(\overline{c});b> -a\rightarrow <c,\ \mathbf{new}(\overline{c});b> -b\rightarrow <c,\ \overline{c},\ E> -\tau\rightarrow <E,\ E,\ E>$.

Before we give the definition of the operational semantics of parallel statements, we introduce its semantic universe P.

DEFINITION 2.4 (Semantic universe P)
Let A^* denote the set of finite sequences or *words* of elements of A; let ϵ denote the empty word. We extend this set by allowing as the last element of a finite sequence a special element ∂, which denotes *deadlock*:

$$(w\in)A_\partial=A^*\cup A^*\cdot\{\partial\}.$$

Now we define $(p,q\in)P=\mathcal{P}_{nf}(A_\partial^*)$, the set of all non-empty, finite subsets of A_∂^*. Let $d_{A_\partial^*}$ denote the usual metric on A_∂^* (see the definition in A.1.1). We take $d_P=(d_{A_\partial^*})_H$, the Hausdorff metric induced by $d_{A_\partial^*}$, as a metric on P. According to proposition A.7, we have that (P,d_P) is a complete metric space.

DEFINITION 2.5 (Operational semantics $\mathcal{O}$)
Let $\mathcal{O}=$ Fixed Point (Φ), where $\Phi:(Par\rightarrow P)\rightarrow(Par\rightarrow P)$ is given, for $F\in Par\rightarrow P$, and $\rho\in Par$, by

$$\Phi(F)(\rho)=\begin{cases}\{\epsilon\} & \text{if } \rho=<E,\ldots,E>\\ \{\partial\} & \text{if } \forall a\forall\rho'\,[\rho-a\rightarrow\rho'\Rightarrow a\in C]\wedge\rho\neq<E,\ldots,E>\\ \bigcup\{a\cdot F(\rho'):\ \rho-a\rightarrow\rho'\wedge a\notin C\} & \text{otherwise.}\end{cases}$$

It is straightforward to show that Φ is a contraction and thus has a unique fixed point.

Since our language does not contain any constructs for recursion, we need not be able to describe infinite behavior. Therefore, it is not really necessary to define $\mathcal{O}$ using a contraction on a complete metric space. It would have been sufficient to take P as an ordinary set without any metric, and define $\mathcal{O}$ with an easy induction on the structure of statements. Our motivation for nevertheless exploiting metric structures here is given by the fact that in the next section we *will* deal with recursion and

infinite behavior. There the use of some mathematical structure which can handle these, such as complete metric spaces, is obligatory. Our use of complete metric spaces at this stage can be seen as part of the introductory function of this section.

The operational semantics $\mathcal{O}$ can be best explained by giving a few

EXAMPLES:

$$\mathcal{O}[\![<a>]\!] = a\cdot\mathcal{O}[\![<E>]\!] = a\cdot\{\epsilon\} = \{a\}$$

$$\mathcal{O}[\![<\textbf{new}\ (a)>]\!] = \{a\}$$

$$\mathcal{O}[\![<c>]\!] = \{\partial\}$$

$$\mathcal{O}[\![<c,\bar{c}>]\!] = \{\tau\}$$

$$\mathcal{O}[\![<a;b>]\!] = a\cdot\mathcal{O}[\![<b>]\!] = \{ab\}$$

$$\mathcal{O}[\![<\textbf{new}\ (a);b>]\!] = \{a\cdot\mathcal{O}[\![<E,b>]\!],\ b\cdot\{\mathcal{O}[\![<a,E>]\!]\} = \{ab, ba\}$$

Note that a single communication $<c>$, without a matching communication $\bar{c}$ in parallel, creates a deadlock.

Such an operational semantics is nice, because it is intuitively very clear. However, it is not *compositional* with respect to the binary syntactic operator ;, that is, there is no semantic operator $\tilde{;}: P\times P\to P$, corresponding to ;, such that for all s and t:

$$\mathcal{O}[\![<s;t>]\!] = \mathcal{O}[\![<s>]\!]\ \tilde{;}\ \mathcal{O}[\![<t>]\!].$$

This can be easily seen by the following argument. Suppose there *is* such an operator $\tilde{;}$. Then:

$$\begin{aligned}\mathcal{O}[\![<\textbf{new}\ (a);b>]\!] &= \mathcal{O}[\![<\textbf{new}\ (a)>]\!]\ \tilde{;}\ \mathcal{O}[\![<b>]\!]\\ &= [\text{since } \mathcal{O}[\![<\textbf{new}\ (a)>]\!] = \mathcal{O}[\![<a>]\!]]\\ &\quad\ \mathcal{O}[\![<a>]\!]\ \tilde{;}\ \mathcal{O}[\![<b>]\!]\\ &= \mathcal{O}[\![<a;b>]\!],\end{aligned}$$

which yields a contradiction, as can be seen from the examples above.

The denotational semantics to be defined in a moment has the property that it is compositional with respect to the syntactic operators in L_T.

First, we define a suitable semantic universe.

DEFINITION 2.6 (Semantic universe $\bar{P}$)
We define a complete metric space $(p,q\in)\bar{P}$ by $\bar{P} = \mathcal{P}_{nf}(A^*)$, the set of non-empty finite subsets of A^*. Let $d_A\cdot$ be the usual metric on A^*; we define $d_P = (d_A\cdot)_H$.

The only difference between P and $\bar{P}$ is that the latter does not contain finite sequences ending in ∂.

DEFINITION 2.7 (Denotational semantics $\mathcal{D}$)
Let $\mathcal{D}: L_T\to \text{Cont}\to\bar{P}$, where $\text{Cont}=\bar{P}$ denotes the set of *continuations*, be given by

$$\mathcal{D}[\![a]\!](p) = a\cdot p,\ \mathcal{D}[\![E]\!](p) = p$$

$$\mathcal{D}[\![\textbf{new}(s)]\!](p) = p\,\|\mathcal{D}[\![s]\!](\{\epsilon\})$$

$$\mathcal{D}[\![s;t]\!](p) = \mathcal{D}[\![s]\!](\mathcal{D}[\![t]\!](p)),$$

with $\|: P\times P\to P$ as defined below.

A continuation $p \in \mathrm{Cont}$ denotes the semantics of the statement to be executed after the one to which $\mathcal{D}$ is applied. The meaning of a new-construct **new** (s) with continuation p is determined as follows: The meaning of s is computed with the empty continuation $\{\epsilon\}$, which indicates that after s nothing remains to be done. Since s is to be executed in parallel with everything that follows, the result is composed in parallel with p, which indicates the remainder of the program after s.

DEFINITION 2.8 (Parallel composition $\|$)
Let $\|$: $P \times P \to P$ be such that it satisfies, for $p,q \in P$,

$$p \| q = p \lfloor\!\lfloor q \cup q \lfloor\!\lfloor p \cup p \mid q,$$

where

$$p \lfloor\!\lfloor q = \bigcup \{a \cdot (p_a \| q) \colon p_a \neq \emptyset\} \cup \{q \colon \epsilon \in p\},$$

$$p \mid q = \bigcup \{\tau \cdot (p_c \| q_{\bar{c}}) \colon p_c \neq \emptyset \neq q_{\bar{c}}\},$$

with $p_a = \{w \colon a \cdot w \in p\}$, the set containing all the postfixes of a in p.

The above definition is self-referential and needs some justification. Formally, we can define $\|$ as the fixed point of a contraction Ψ: $(\bar{P} \times \bar{P} \to \bar{P}) \to (\bar{P} \times \bar{P} \to \bar{P})$ given, for $f \in \bar{P} \times \bar{P} \to \bar{P}$, by

$$\Psi(f)(p,q) = p \lfloor\!\lfloor_f q \cup q \lfloor\!\lfloor_f p \cup p \mid_f q,$$

where

$$p \lfloor\!\lfloor_f q = \bigcup \{a \cdot f(p_a, q) \colon p_a \neq \emptyset\} \cup \{q \colon \epsilon \in p\},$$

$$p \mid_f q = \bigcup \{\tau \cdot (f(p_c, q_{\bar{c}})) \colon p_c \neq \emptyset \neq q_{\bar{c}}\}.$$

Note that $\mathcal{D}$ is compositional with respect to ";". The corresponding semantic operator $\tilde{;}$: $((\bar{P} \to \bar{P}) \times (\bar{P} \to \bar{P})) \to (\bar{P} \to \bar{P})$ is not expressed explicitly in the definition of $\mathcal{D}$. For completeness sake, we give its definition. We have, for $f,g \in \bar{P} \to \bar{P}$:

$$f \tilde{;} g = \lambda p \cdot f(g(p)).$$

Semantic equivalence of $\mathcal{O}$ and $\mathcal{D}$
After having defined $\mathcal{O}$ and $\mathcal{D}$ for *Par* and L_T, we next discuss the relationship between the two semantics. We shall compare $\mathcal{O}$ and $\mathcal{D}$ by relating both to an intermediate semantics $\mathcal{O}'$: $Par \to P$, given in

DEFINITION 2.9 (Intermediate semantics $\mathcal{O}'$)
Let $\mathcal{O}' =$ Fixed Point (Φ'), where Φ': $(Par \to \bar{P}) \to (Par \to \bar{P})$ is given, for $F \in Par \to \bar{P}$ and $\rho \in Par$, by

$$\Phi'(F)(\rho) = \begin{cases} \{\epsilon\} & \text{if } \rho = <E, \ldots, E> \\ \bigcup \{a \cdot F(\rho') \colon \rho - a \to \rho'\} & \text{otherwise.} \end{cases}$$

Note that in Φ', as opposed to Φ, single-sided communication steps $a \in C$ are allowed. The difference between $\mathcal{O}$ and $\mathcal{O}'$ can be illustrated by giving a few examples:

$$\mathcal{O}[\![<c>]\!] = \{\partial\}, \quad \mathcal{O}[\![<c,\bar{c}>]\!] = \{\tau\},$$

$$\mathcal{O}'[\![<c>]\!] = \{c\}, \quad \mathcal{O}'[\![<c,\bar{c}>]\!] = \{c\bar{c}, \bar{c}c, \tau\}.$$

The relationship between $\mathcal{O}$ and $\mathcal{O}'$ will be expressed using the following abstraction operation.

DEFINITION 2.10 (Abstraction operator α)
We define an abstraction operator α: $\bar{P} \to P$ by

$$\alpha(p)=\begin{cases}\{\partial\} & \text{if } \forall a[p_a\neq\varnothing \Rightarrow a\in C]\\ \bigcup\{a\cdot(\alpha(p_a)):\ a\notin C\wedge p_a\neq\varnothing\}\ \cup\ \{\epsilon:\epsilon\in p\} & \text{otherwise,}\end{cases}$$

with p_a as in definition 2.8. (For a justification of this self-referential definition see the remark following definition 2.8.)

The definition of α can be understood as follows: If all the words $w\in p$ begin with a communication action $a\in C$, we have operationally a deadlock, since no single communication action is allowed. Therefore, we then have: $\alpha(p)=\{\partial\}$. In the last case, $\alpha(p)$ contains all the words in p that begin with a non-communication action $a\in A\setminus C$, with α recursively applied to p_a, the set of postfixes of a; additionally, $\alpha(p)$ contains ϵ if $\epsilon\in p$.

The following theorem can be proved straightforwardly.

THEOREM 2.11: $\forall F\in Par\to\bar{P}\,[\Phi(\alpha\circ F)=\alpha\circ\Phi'(F)]$

Since Φ and Φ' are contractions and thus have unique fixed points, it follows that

COROLLARY 2.12: $\vartheta=\alpha\circ\vartheta'$

PROOF
We have: $\alpha\circ\vartheta'=\alpha\circ\Phi'(\vartheta')=\Phi(\alpha\circ\vartheta')$. Thus both $\alpha\circ\vartheta'$ and ϑ are fixed points of Φ which implies that they are equal.

The relationship between ϑ' and $\mathfrak{D}$ can be elegantly expressed using the following mapping.

DEFINITION 2.13
We define $\sim$: $(L_T\to Cont\to\bar{P})\to(Par\to\bar{P})$ as follows. We denote, for $F\in L_T\to Cont\to\bar{P}$, $\sim(F)$ by $\tilde{F}$ and put

$$\tilde{F}=\lambda\rho\in Par\cdot(F(s_1)(\{\epsilon\})\|\cdots\|F(s_n)(\{\epsilon\})),$$

with $\rho=\langle s_1,\ldots,s_n\rangle$.

A simple consequence, using the associativity of $\|$, of this definition is: $\tilde{F}(\rho^\wedge\tau)=\tilde{F}(\rho)\|\tilde{F}(\tau)$. If the function $\tilde{F}$ takes a parallel statement $\langle s_1,\ldots,s_n\rangle$ as an argument, then the F values of all the substatements s_i supplied with the empty continuation $\{\epsilon\}$ are computed and next composed in parallel.
Now we can prove that $\vartheta'=\tilde{\mathfrak{D}}$. It is a corollary of the following

THEOREM 2.14: $\Phi'(\tilde{\mathfrak{D}})=\tilde{\mathfrak{D}}$

PROOF
The proof uses induction on the structure of parallel statements. We treat one typical case, leaving the other ones to the reader. Consider $\rho^\wedge\pi\in Par$ and suppose $\rho\neq\langle E,\ldots,E\rangle$ and $\pi\neq\langle E,\ldots,E\rangle$. Suppose we already know that $\Phi'(\tilde{\mathfrak{D}})(\rho)=\tilde{\mathfrak{D}}(\rho)$ and $\Phi'(\tilde{\mathfrak{D}})(\pi)=\tilde{\mathfrak{D}}(\pi)$. We show: $\Phi'(\tilde{\mathfrak{D}})(\rho^\wedge\pi)=\tilde{\mathfrak{D}}(\rho^\wedge\pi)$.

$$\begin{aligned}\Phi'(\tilde{\mathfrak{D}})(\rho^\wedge\pi) &= \bigcup\{a\cdot\tilde{\mathfrak{D}}(\rho'):\ \rho^\wedge\pi-a\to\rho'\}\\ &= \text{[definition of }\to\text{ (2.3 (5) and (6))]}\\ &\quad \bigcup\{a\cdot\tilde{\mathfrak{D}}(\rho'^\wedge\pi):\ \rho-a\to\rho'\}\ \cup\ \bigcup\{a\cdot\tilde{\mathfrak{D}}(\rho^\wedge\pi'):\ \pi-a\to\pi'\}\ \cup\\ &\quad \bigcup\{\tau\cdot\tilde{\mathfrak{D}}(\rho'^\wedge\pi'):\ \rho-c\to\rho'\wedge\pi-\bar{c}\to\pi'\}\end{aligned}$$

$= $ [definition $\sim$]

$\bigcup \{a\cdot(\tilde{\mathfrak{D}}(\rho')\|\tilde{\mathfrak{D}}(\pi)) \colon \rho - a \to \rho'\} \cup \bigcup \{a\cdot(\tilde{\mathfrak{D}}(\rho)\|\tilde{\mathfrak{D}}(\pi')) \colon \pi - a \to \pi'\} \cup$

$\bigcup \{\tau\cdot(\tilde{\mathfrak{D}}(\rho')\|\tilde{\mathfrak{D}}(\pi')) \colon \rho - c \to \rho' \wedge \pi - \overline{c} \to \pi'\}$

$= $ [definitions $\lfloor\!\lfloor$ and $|\ ]$

$(\bigcup \{a\cdot\tilde{\mathfrak{D}}(\rho') \colon \rho - a \to \rho'\} \lfloor\!\lfloor \tilde{\mathfrak{D}}(\pi)) \cup (\bigcup \{a\cdot\tilde{\mathfrak{D}}(\pi') \colon \pi - a \to \pi'\} \lfloor\!\lfloor \tilde{\mathfrak{D}}(\rho)) \cup$

$(\bigcup \{c\cdot\tilde{\mathfrak{D}}(\rho') \colon \rho - c \to \rho'\} | \bigcup \{\overline{c} \cdot \tilde{\mathfrak{D}}(\pi') \colon \pi - \overline{c} \to \pi'\})$

$= (\Phi'(\tilde{\mathfrak{D}})(\rho) \lfloor\!\lfloor \tilde{\mathfrak{D}}(\pi)) \cup (\Phi'(\tilde{\mathfrak{D}})(\pi) \lfloor\!\lfloor \tilde{\mathfrak{D}}(\rho)) \cup (\Phi'(\tilde{\mathfrak{D}})(\rho) | \Phi'(\tilde{\mathfrak{D}})(\pi))$

$= $ [induction]

$(\tilde{\mathfrak{D}}(\rho) \lfloor\!\lfloor \tilde{\mathfrak{D}}(\pi)) \cup (\tilde{\mathfrak{D}}(\pi) \lfloor\!\lfloor \tilde{\mathfrak{D}}(\rho) \cup (\tilde{\mathfrak{D}}(\rho) | \tilde{\mathfrak{D}}(\pi))$

$= \tilde{\mathfrak{D}}(\rho)\|\tilde{\mathfrak{D}}(\pi)$

$= \tilde{\mathfrak{D}}(\rho^\wedge\pi)$ □

COROLLARY 2.15: $\mathfrak{O}' = \tilde{\mathfrak{D}}$

Combining Corollaries 2.12 and 2.15 now yields the main theorem of this section.

MAIN THEOREM 2.16: $\mathfrak{O} = \alpha \circ \tilde{\mathfrak{D}}$

COROLLARY 2.17: $\forall s \in L_T\, [\mathfrak{O}[\![<s>]\!] = \alpha(\mathfrak{D}[\![s]\!](\{\epsilon\}))]$.

3. THE LANGUAGE POOL

In this paper, we compare different semantic models of a language that we call POOL: Parallel Object-Oriented Language. It is a simplified version of a language called POOL-T, which is defined in [Am85]. (For an account of the design considerations for POOL-T see [Am86] and [Am87].) The simplification is two-fold. First, we omitted certain language constructs from POOL-T (such as the select statement and the method call) as well as some of the protection mechanisms offered by the definition of classes (such as different classes having different (instances of) variables and method definitions). We have done this in order to make life somewhat easier: the semantic definitions are shorter and so are the proofs of the theorems. We feel justified in doing so, since it is straightforward to extend the approach of this paper to the full language. Secondly, we give an abstract syntactic description of POOL which is a simplified version of the formal description of POOL-T.

A POOL program describes the behavior of a whole system in terms of its constituents, *objects*. Objects contain some internal data, and some procedures that act on these data (these are called *methods* in the object-oriented jargon). Objects are entities of a dynamic nature: they can be created dynamically, their internal data can be modified, and they have an internal activity of their own. At the same time they are units of protection: the internal data of one object are not directly accessible for other objects.

An object uses *variables* (more specifically: instance variables) to store its internal data. Each variable can contain the *name* of an object (another object, or, possibly, the object under consideration itself). An assignment to a variable can make it refer to an object different from the object referred to before. The variables of one object cannot be accessed directly by other objects. They can only be read and changed by the object itself.

Objects can interact by sending *messages* to each other. A message is a request for the receiver to

execute a certain method. Messages are sent and received explicitly. In sending a message, the sender mentions the destination object, the method to be executed, and possibly a parameter (which is again an object name) to be passed to this method. After this, its activity is suspended. The receiver can specify the set of methods that will be accepted, but it can place no restrictions on the identity of the sender or on the parameters of messages. If a message arrives specifying an appropriate method, the method is executed with the parameters contained in the message. Upon termination, this method delivers a result (an object name), which is returned to the sender of the message. The latter then resumes its own execution. Note that this form of communication strongly resembles the rendez-vous mechanism of Ada ([ANSI83]).

A method can access the variables of the object by which it is executed (the receiver of a message). Furthermore, it has a formal parameter, which is initialized to the actual parameter specified in the message.

When an object is created, a local activity is started: the object's *body*. When several objects have been created, their bodies execute in parallel. This is the way parallelism is introduced into the language. Synchronization and communication takes places by sending messages, as described above.

Objects are grouped into *classes*. All objects in one class (the *instances* of that class) execute the same body. In creating an object, only its desired class must be specified. In this way a class serves as a blueprint for the creation of its instances.

At this point, it might be useful to emphasize the distinction between an object and its name. Objects are intuitive entities as described above. In this paper, there will appear no mathematical construction that directly models a single object with all its dynamic properties (although it would be interesting to see a semantics which does this). Object names, on the other hand, are modeled explicitly as elements of some abstract set *Obj*. Object names are only *references* to objects. On its own, an object name gives little information about the object it refers to. In fact, object names are just sufficient to distinguish the individual objects from each other. Note that variables and parameters contain object names, and that expressions result in object names, not objects. If in the sequel we speak, for example, of "the object α", we hope the reader will understand that the object with name α is meant.

Now we describe the (abstract) syntax of the language POOL. We assume that the following sets of syntactic elements are given:

$(x \in) IVar$ (instance variables),

$(u \in) TVar$ (temporary variables),

$(C \in) CName$ (class names),

$(m \in) MName$ (method names).

DEFINITION 3.1 (Expressions, statements, units)
We define the set of expressions $(e \in) L_E$ and the set of statements $(s \in) L_S$ by:

$$e ::= x \mid u \mid e_1!m(e_2) \mid \textbf{new}(C) \mid s;e \mid \textbf{self}$$

$$s ::= x \leftarrow e \mid u \leftarrow e \mid \textbf{answer}\ m \mid s_1;s_2 \mid \textbf{if}\ e\ \textbf{then}\ s_1\ \textbf{else}\ s_2\ \textbf{fi} \mid \textbf{do}\ e\ \textbf{then}\ s\ \textbf{od}.$$

The set $(U \in) Unit$ of units is defined by

$$U ::= < (C_1 \Leftarrow s_1, \ldots, C_n \Leftarrow s_n), (m_1 \Leftarrow < u_1, e_1 >, \ldots, m_k \Leftarrow < u_m, e_k >) >.$$

We write $C \Leftarrow s \in U$ if there exists an i such that $C_i = C$ and $s_i = s$. Similarly, we write $m \Leftarrow < u, e > \in U$.

An instance variable or a temporary variable used as an expression will yield as its value the object name that is currently stored in that variable.

The next kind of expression is a send expression. Here, e_1 is the destination object, to which the

message will be sent, m is the method to be invoked, and e_2 is the parameter. When a send expression is evaluated, the destination expression and the parameter expression are evaluated successively. Next, the message is sent to the destination object. When this object answers the message, the corresponding method is executed, that is, the formal parameter is initialized to the name of the object in the message, and the expression in the method definition is evaluated. The value which results from this evaluation is sent back to the sender of the message and this will be the value of the send expression.

A new-expression indicates that a new object is to be created, an instance of the indicated class. Its body starts executing in parallel with all other objects in the system. The result of the new-expression is (the name of) this newly created object.

An expression may also be preceded by a statement. In this case the statement is executed before the expression is evaluated.

The expression **self** always results in the name of the object that is executing this expression.

The first two kinds of statements are assignments, to an instance variable and to a temporary variable, respectively. An assignment is executed by first evaluating the expression on the right, and then making the variable on the left refer to the resulting object.

An answer statement indicates that a message is to be answered. The object executing the answer statement waits until a message arrives with a method name that is specified by the answer statement. Then it executes the method (after initializing the formal parameter). The result of the method is sent back to the sender of the message, and the answer statement terminates.

Sequential composition, conditionals and loops have the usual meaning.

Units are the programs of POOL. A unit consists of a number of definitions of class bodies and methods. If a unit is to be executed, a single new instance of the *last* class defined in the unit is created and execution of its body is started. This object has the task to start the whole system, by creating new objects and putting them to work.

The relationship between POOL and POOL-T is the following: POOL is obtained from POOL-T via two successive simplifications. First, certain language constructs from POOL-T are omitted (like the select statement) as well as some of the protection mechanisms in POOL-T, which are offered by the definition of classes (such as different classes having different variables and method definitions). Secondly, some syntactical simplifications are performed and some context information is omitted (POOL-T is a statically typed language whereas POOL is not). The reason for making the first simplification is simply lack of space, to which should be added the consideration that it would be straightforward to extend our results to the full language. The sole reason for making the second simplification is that POOL-T is a practical programming language, for which readability, among others, is more important than syntactic simplicity. Therefore, it is convenient to take a simplified language, POOL, as the semantic core of POOL-T.

If one compares the version of POOL described in this paper with the one given in [ABKR86(a)] and [ABKR86(b)], some minor differences can be observed. (For example, in the send expression of definition 3.1 above only one parameter can be specified whereas in the definitions of the papers mentioned an arbitrary number of parameters is allowed.) However, it can easily be seen that it is straightforward to adapt the definitions and proofs given in this paper such that they apply to the version of POOL occurring in [ABKR86(a)] and [ABKR86(b)].

4. An operational semantics for POOL

In this section we give the definition of an operational semantics for POOL, which is a modified version of the one given in [ABKR86(a)]. (At the end of this section, we shall compare both models in some detail.) It is based on a *transition relation* and will be defined as the fixed point of a suitable contraction. For this purpose, we introduce a number of syntactic and semantic notions.

First of all, we introduce the set of objects.

DEFINITION 4.1 (Objects)
We assume given a set $AObj$ of names for *active* objects together with a function

$$\nu: \mathcal{P}_{fin}(AObj) \rightarrow AObj$$

such that $\nu(X) \notin X$, for every finite $X \subseteq AObj$. Given a set X of object names, the function ν yields a new name not in X.
Further we define

$$Obj = AObj \cup SObj,$$

where $SObj$ is the set of so-called *standard* objects, to be introduced in Appendix III.

A possible example of such a set $AObj$ and function ν could be obtained by setting:

$$AObj = \mathbb{N},$$

$$\nu(X) = \max\{n : n \in X\} + 1.$$

In POOL, a few standard classes, the instances of which are called standard objects, are predefined; examples are the classes of booleans and integers. The semantic treatment of these standard objects is somewhat different from the way the active objects (which are created during the execution of a POOL program) are treated. Because we want to formulate our semantic models as concisely as possible in order to focus on the correctness proof, the standard objects are treated in an appendix (III).

Next, it is convenient to extend the sets L_E of expressions and L_S of statements by adding some auxiliary syntactic constructs.

DEFINITION 4.2 ($L_{E'}$, $L_{S'}$)
Let $(e \in) L_{E'}$ and $(s \in) L_{S'}$ be defined by

$$e ::= x \mid u \mid e_1!m(e_2) \mid \textbf{new}(C) \mid s;e \mid \textbf{self} \mid \alpha \mid (e, \phi)$$

$$s ::= x \leftarrow e \mid u \leftarrow e \mid \textbf{answer}\ m \mid s_1;s_2 \mid \textbf{if}\ e\ \textbf{then}\ s_1\ \textbf{else}\ s_2\ \textbf{fi} \mid \textbf{do}\ e\ \textbf{then}\ s\ \textbf{od} \mid$$
$$\textbf{release}(\beta, s) \mid (e, \psi)$$

with $\alpha, \beta \in AObj$, $\phi \in L_{PE}$ and $\psi \in L_{PS}$. Here the sets of *parameterized expressions* $(\phi \in) L_{PE}$ and *parameterized statements* $(\psi \in) L_{PS}$ are given by

$$\phi ::= \lambda u \cdot e$$

$$\psi ::= \lambda u \cdot s,$$

with the restriction that u does not occur at the left-hand side of an assignment in e or s. For $\alpha \in Obj$, $\phi = \lambda u \cdot e$ and $\psi = \lambda u \cdot s$, we shall use $\phi(\alpha)$ and $\psi(\alpha)$ to denote the expression and the statement obtained by syntactically substituting α for all free occurrences of u in ϕ and ψ, respectively. The restriction just mentioned ensures that the result of this substitution again is a well-formed expression or statement.

Let us explain the new syntactic constructs. In addition to what we already had in L_E, an expression $e \in L_{E'}$ can be an *active* object α or a pair (e, ϕ) of an expression e and a parameterized expression ϕ. The latter will be executed as follows: First the expression e is evaluated, then the result β is substituted in ϕ and $\phi(\beta)$ is executed. As new statements we have release statements **release**(β, s) and parameterized statements (e, ϕ). If the statement **release**(β, s) is executed, the active object β will start executing the statement s (in parallel to the objects that are already executing). The release statement will be used in the description of the communication between two objects (see definition 4.8 below). The interpretation of (e, ψ) is similar to that of (e, ϕ).

DEFINITION 4.3 (Empty statement)
The set $L_{S'}$, as given in the definition above, is extended with a special element E, denoting the *empty statement*. This extended set is again called $L_{S'}$. Note that we do *not* have elements like $s;E$ or **do** e **then** E **od** in $L_{S'}$. (There is, however, one exception: We *do* allow E in **if** e **then** s **else** E **fi**, which is needed in definition 4.8(A8) below.)

DEFINITION 4.4 (States)
The set of states $(\sigma \in)\Sigma$ is defined by

$$\Sigma = (AObj \rightarrow IVar \rightarrow Obj) \times (AObj \rightarrow TVar \rightarrow Obj) \times \mathcal{P}_{fin}(AObj).$$

The three components of σ are denoted by $<\sigma_1, \sigma_2, \sigma_3>$. The first and the second component of a state store the values of the instance variables and the temporary variables of each active object. The third component contains the object names currently in use. We need it in order to give unique names to newly created objects.

We shall use the following variant notation. By $\sigma\{\beta/\alpha, x\}$ (with $x \in IVar$) we shall denote the state σ' that is as σ but for the value of $\sigma_1'(\alpha)(x)$, which is β. Similarly, we denote by $\sigma\{\beta/\alpha, u\}$ (with $u \in TVar$) the state σ' that is as σ but for the value of $\sigma_1'(\alpha)(u)$, which is β.

DEFINITION 4.5 (Labelled statements)
The set of *labelled statements* $((\alpha, s) \in) LStat$ is given by $LStat = AObj \times L_{S'}$.

A labelled statement (α, s) should be interpreted as a statement s which is going to be executed by the active object α.

Sometimes, we also need labelled parameterized statements. Therefore, we extend *LStat*:

$$LStat' = LStat \cup (AObj \times L_{PS}).$$

A pair (α,ψ) indicates that the active object α will execute the statement ψ as soon as it receives a value which it can supply to ψ as an argument.

Before we can give the definition of a transition relation for POOL, we first have to explain which *configurations* and *transition labels* we are going to use.

DEFINITION 4.6 (Configurations)
The set of configurations $(\rho \in) Conf$ is given by

$$Conf = \mathcal{P}_{fin}(LStat) \times \Sigma.$$

We also introduce:

$$Conf' = \mathcal{P}_{fin}(LStat') \times \Sigma.$$

Typical elements of *Conf* and *Conf'* will also be indicated by $<X,\sigma>$ and $<Y,\sigma>$.

We shall consider only configurations $<X,\sigma>$ that are *consistent* in the following sense: For $X = \{(\alpha_1, s_1), \ldots, (\alpha_k, s_k)\}$, we call $<X,\sigma>$ consistent if the following conditions are satisfied:

$$\forall i,j \in \{1, \ldots, k\}\ [i \neq j \Rightarrow \alpha_i \neq \alpha_j], \text{ and}$$

$$\{\alpha_1, \ldots, \alpha_k\} \subseteq \sigma_3.$$

Whenever we introduce a configuration $<X, \sigma>$, it will be tacitly assumed that it is consistent.

A configuration $<X, \sigma>$, consisting of a finite set X of labelled statements and a state σ, represents a "snap shot" of the execution of a POOL program. It shows what objects are active and

what statements they are executing; furthermore, it contains a state σ, in which the values of the variables of the active objects as well as the set of object names currently in use are stored.

DEFINITION 4.7 (Transition labels)
The set of *transition labels* $(\lambda\in)\Lambda$ is given by

$$\Lambda = \{\tau\} \cup \{(\alpha, \beta_1!m(\beta_2))\colon \alpha,\beta_1 \in AObj,\ \beta_2 \in Obj\} \cup \{(\beta?m)\colon \beta \in AObj\}.$$

These labels will be used in the definition of the transition relation below and are to be interpreted as follows. The label τ indicates a so-called *computation* step. Next, $(\alpha, \beta_1!m(\beta_2))$ indicates that object α sends a message to object β_1 requesting the execution of the method m with parameter β_2. Finally, $(\beta?m)$ indicates that the object β is willing to answer a message specifying the method m.

Now we are ready to define a transition relation for POOL.

DEFINITION 4.8 (Transition relation)
Let $U \in Unit$. We define a *labelled transition relation*

$$-U\rightarrow \subseteq \text{Conf} \times \Lambda \times \text{Conf}'.$$

Triples $<\rho_1, \lambda, \rho_2> \in -U\rightarrow$ will be called *transitions* and are denoted by

$$\rho_1\ -U, \lambda\rightarrow\ \rho_2.$$

Such a transition reflects a possible execution step of type λ of the configuration ρ_1, yielding a new configuration ρ_2. The relation $-U\rightarrow$ is defined as the smallest relation satisfying the following properties:

Axioms

(A1) $<\{(\alpha, (x, \psi))\}, \sigma> -U, \tau\rightarrow <\{(\alpha, (\sigma_1(\alpha)(x), \psi))\}, \sigma>$

(A2) $<\{(\alpha, (u, \psi))\}, \sigma> -U, \tau\rightarrow <\{(\alpha, (\sigma_2(\alpha)(u), \psi))\}, \sigma>$

(A3) $<\{(\alpha, (\beta_1!m(\beta_2), \psi)\}, \sigma> -U, (\alpha, (\beta_1!m(\beta_2)))\rightarrow <\{(\alpha, \psi)\}, \sigma>$

(A4) $<\{(\alpha, (\textbf{new}\ (C), \psi))\}, \sigma> -U, \tau\rightarrow <\{(\alpha, (\beta, \psi)), (\beta, s_C)\}, \sigma'>$, where:
$C \Leftarrow s_C \in U,\ \beta = \nu(\sigma_3),\ \sigma' = <\sigma_1, \sigma_2, \sigma_3 \cup \{\beta\}>$.

(A5) $<\{(\alpha, z\leftarrow\beta)\}, \sigma> -U, \tau\rightarrow <\{(\alpha, E)\}, \sigma\{\beta/\alpha, z\}>$, for $z \in IVar \cup TVar$.

(A6) $<\{(\alpha, \textbf{answer}\ m)\}, \sigma> -U, (\alpha?m)\rightarrow <\{(\alpha, E)\}, \sigma>$

(A7) $<\{(\alpha, \textbf{do}\ e\ \textbf{then}\ s\ \textbf{od}\ )\}, \sigma> -U, \tau\rightarrow$
$<\{(\alpha, \textbf{if}\ e\ \textbf{then}\ (s;\ \textbf{do}\ e\ \textbf{then}\ s\ \textbf{od}\ )\ \textbf{else}\ E\ \textbf{fi}\ )\}, \sigma>$

Rules

(R1) If $<\{(\alpha, (e, \lambda u \cdot z\leftarrow u))\}, \sigma> -U, \lambda\rightarrow \rho$,
then $<\{(\alpha, z\leftarrow e)\}, \sigma> -U, \lambda\rightarrow \rho$, for $z \in IVar \cup TVar$.

(R2) If $<\{(\alpha, s)\}, \sigma> -U, \lambda\rightarrow <\{(\alpha, s')\} \cup X, \sigma'>$,
then $<\{(\alpha, s;t)\}, \sigma> -U, \lambda\rightarrow <\{(\alpha, s';t)\} \cup X, \sigma'>$
(read t instead of $s';t$ if $s' = E$).
If $<\{(\alpha, s)\}, \sigma> -U, \lambda\rightarrow <\{(\alpha, \psi)\} \cup X, \sigma'>$,

then $<\{(\alpha, s;t)\},\sigma> -U, \lambda\rightarrow <\{(\alpha, \lambda u\cdot(\psi(u);t))\}\cup X,\sigma'>$.

(R3) If $<\{(\alpha, s_i)\},\sigma> -U, \lambda\rightarrow \rho$, then $<\{(\alpha,$ **if** β **then** s_1 **else** s_2 **fi** $)\},\sigma> -U, \lambda\rightarrow \rho$,

where $s_i=\begin{cases} s_1 & \text{if } \beta=tt \\ s_2 & \text{if } \beta=ff. \end{cases}$

(R4) If $<\{(\alpha, t),(\beta,s)\},\sigma> -U, \lambda\rightarrow \rho$, then $<\{(\alpha,$ **release** $(\beta,s);t)\},\sigma> -U, \lambda\rightarrow \rho$

(read **release**(β,s) instead of **release**$(\beta,s);t$ if $t=E$).

(R5) If $<\{(\alpha, (e,\lambda u\cdot$**if** u **then** s_1 **else** s_2**fi** $))\},\sigma> -U, \lambda\rightarrow \rho$,

then $<\{(\alpha,$ **if** e **then** s_1 **else** s_2 **fi** $)\},\sigma> -U, \lambda\rightarrow \rho$.

(Here s_2 is allowed to be E.)

(R6) If $<\{(\alpha, ((e_1,\lambda u_1\cdot(e_2,\lambda u_2\cdot u_1!m(u_2))),\psi))\},\sigma> -U, \lambda\rightarrow \rho$,

then $<\{(\alpha, (e_1!m(e_2),\psi))\},\sigma> -U, \lambda\rightarrow \rho$.

(R7) If $<\{(\alpha, s;(e,\psi))\},\sigma> -U, \lambda\rightarrow \rho$, then $<\{(\alpha, (s;e,\psi))\},\sigma> -U, \lambda\rightarrow \rho$.

(R8) If $<\{(\alpha, (e,\lambda u\cdot(\phi(u),\psi)))\},\sigma> -U, \lambda\rightarrow \rho$, then $<\{(\alpha, ((e,\phi),\psi))\},\sigma> -U, \lambda\rightarrow \rho$.

(R9) If $<\{(\alpha, \psi(\beta))\},\sigma> -U, \lambda\rightarrow \rho$, then $<\{(\alpha, (\beta,\psi))\},\sigma> -U, \lambda\rightarrow \rho$, for $\beta\in Obj$.

If $<\{(\alpha, \psi(\alpha))\},\sigma> -U, \lambda\rightarrow \rho$, then $<\{(\alpha,$ **(self**$,\psi))\},\sigma> -U, \lambda\rightarrow \rho$.

(R10) If $<X,\sigma> -U, \lambda\rightarrow <X',\sigma'>$, then $<X\cup Y,\sigma> -U, \lambda\rightarrow <X'\cup Y,\sigma'>$.

(R11) If $<X,\sigma> -U, (\alpha,\beta_1!m(\beta_2))\rightarrow <\{(\alpha, \psi)\}\cup X',\sigma>$ and

$<Y,\sigma> -U, \beta_1?m\rightarrow <\{(\beta_1,s)\}\cup Y',\sigma>$,

then $<X\cup Y,\sigma> -U, \tau\rightarrow$

$<\{(\beta_1,(e_m,\lambda u\cdot(u_m\leftarrow\sigma_2(\beta_1)(u_m);$ **release**$(\alpha, \psi(u));s))))\}\cup X'\cup Y',\sigma'>$,

where $\sigma'=\sigma\{\beta_2/\beta_1,u_m\}$, and $m\Leftarrow<u_m,e_m>\in U$.

(End of definition.)

The general scheme for the evaluation of an expression is very similar to the approach taken in [AB88]. Expressions always occur in the context of a (possibly parameterized) statement, such as $x\leftarrow e$. A statement containing e as a subexpression is transformed into a pair (e,ψ) of the expression e and a parameterized statement ψ by application of one of the rules. (In our example, $x\leftarrow e$ becomes $(x, \lambda u\cdot x\leftarrow u)$ by an application of (R1).) Then e is evaluated, using the axioms and rules, and results in some value $\beta'\in Obj$. (Applying (A1) transforms $(x, \lambda u\cdot x\leftarrow u)$ of our example into $(\beta', \lambda u\cdot x\leftarrow u)$, for some $\beta'\in Obj$.) Next, an application of (R9) will put the resulting object β' back into the original context ψ (yielding $x\leftarrow\beta'$ in our example). Finally, the statement $\psi(\beta')$ is further evaluated by using the axioms and the rules. (The evaluation of $x\leftarrow\beta'$ results, by using (A6), in a transformation of the state.)

Let us briefly explain some of the axioms and rules above.

In (A4) a new object is created. Its name β is obtained by applying the function ν to the set σ_3 of the active object names currently in use and is delivered as the result of the evaluation of **new**(C). The body s_C of class C, defined in the unit U, is going to be evaluated by β. Note that the state σ is changed by extending σ_3 with β.

In (R8), the evaluation of an expression pair (e,ϕ), where ϕ is a parameterized expression, in the context of a parameterized statement ψ is reduced to the evaluation of the expression e in the context of the adapted parameterized statement $\lambda u\cdot(\phi(u),\psi)$.

(R11) describes the communication rendez-vous of POOL. If the object α is sending a message to object β_1, requesting the execution of the method m and if the object β_1 is willing to answer such a message, then the following happens: The object β_1 starts executing the expression e_m, which corresponds to the definition of the method m in U, while its state $\sigma_2(\beta_1)$ is changed by setting u_m, the formal parameter belonging to m, to β_2, the parameter sent by the object α to β_1. After the execution of e_m, the object β_1 continues by executing $u_m \leftarrow \sigma_2(\beta_1)(u_m)$, which restores the old value of u_m, followed by the statement **release**$(\alpha,\psi(u));s$. The execution of **release**$(\alpha,\psi(u))$ will reactivate the object α, which starts executing $\psi(u)$, the statement obtained by substituting the result u of the execution of e_m into ψ. Note that during the execution of e_m the object α is non-active, as can be seen from the fact that α does not occur as the name of any labelled statement in the configuration resulting from this transition. Finally, the object β_1 proceeds with the execution of the statement s which is the remainder of its body.

(Note that we have not incorporated any transitions for the standard objects; this is done in Appendix III.)

Now we are ready for the definition of the operational semantics of POOL. It will use the following semantic universe.

DEFINITION 4.9 (Semantic universe P)
Let $(w \in)\Sigma_\partial^\infty = \Sigma^* \cup \Sigma^\omega \cup \Sigma^* \cdot \{\partial\}$, the set of *streams*. We define

$$(p,q \in)P = \Sigma \to \mathcal{P}_{ncompact}(\Sigma_\partial^\infty),$$

where $\mathcal{P}_{ncompact}(\Sigma_\partial^\infty)$ is the set of all non-empty compact subsets of Σ_∂^∞, and the symbol ∂ denotes deadlock. The set P is a complete metric space when supplied with the usual metric (see definition A.6).

The elements of P will be used to represent the operational meanings of statements and units. For a given state $\sigma \in \Sigma$, the set $p(\sigma)$ contains streams $w \in \Sigma_\partial^\infty$, which are sequences of states representing possible computations. They can be of one of three forms: If $w \in \Sigma^*$, it stands for a finite normally terminating computation. If $w \in \Sigma^\omega$, it represents an infinite computation. Finally, if $w \in \Sigma^* \cdot \{\partial\}$, it reflects a finite abnormally terminating computation, which is indicated by the symbol ∂ for deadlock.

DEFINITION 4.10 (Operational semantics for POOL)
We define the operational semantics of finite subsets of labelled statements. Let, for a unit $U \in Unit$, the function

$$\Phi_U: (\mathcal{P}_{fin}(LStat) \to P) \to (\mathcal{P}_{fin}(LStat) \to P)$$

be given, for $F \in \mathcal{P}_{fin}(LStat) \to P$ and $X \in \mathcal{P}_{fin}(LStat)$, by:

$$\Phi_U(F)(X) = \lambda\sigma \cdot \begin{cases} \{\epsilon\} & \text{if } \forall\alpha\forall s\ [(\alpha,s) \in X \Rightarrow s = E] \\ \{\partial\} & \text{if } \neg <X,\sigma> -U,\tau\to \text{ and } \exists\alpha\exists s\ [s \neq E \wedge (\alpha,s) \in X] \\ \bigcup \{\sigma' \cdot F(X')(\sigma') : <X,\sigma> -U,\tau\to <X',\sigma'>\} & \text{otherwise,} \end{cases}$$

where

$$<X,\sigma> -U,\tau\to = \exists X'\exists\sigma'\ [<X,\sigma> -U,\tau\to <X',\sigma'>].$$

Now the operational semantics $\mathcal{O}_U: \mathcal{P}_{fin}(LStat) \to P$ is given as

$$\mathcal{O}_U = \text{Fixed Point } (\Phi_U).$$

It is straightforward to prove that Φ_U is a contraction and thus has a unique fixed point.
The definition of Φ_U is very similar to the definition of Φ in the previous section (definition 2.5). If,

for a given $X \in \mathcal{P}_{fin}(LStat)$ and $\sigma \in \Sigma$, we have that $\neg <X,\sigma> \ -U, \tau \rightarrow$, then no computation steps, which are indicated by τ, are possible from $<X,\sigma>$. The transitions that *are* possible are of the form

$$<X,\sigma> \ -U, (\alpha,\beta_1!m(\beta_2)) \rightarrow \rho \ , \quad \text{or} \ <X,\sigma> \ -U, (\alpha?m) \rightarrow \rho' \ ,$$

denoting attempts of a single object α to perform a communication action without any matching object being present. This is an instance of deadlock and therefore we here have: $\mathcal{O}_U[\![X]\!](\sigma)=\{\partial\}$. On the other hand, for every transition

$$<X,\sigma> \ -U, \tau \rightarrow <X',\sigma'>$$

the set $\mathcal{O}_U[\![X]\!](\sigma)$ includes the set $\sigma' \cdot \mathcal{O}_U[\![X']\!](\sigma')$, in which the transformed state σ' is concatenated with the operational meaning of X' in state σ'.

Finally, we can give the operational semantics of a unit.

DEFINITION 4.11 (Operational semantics of a unit)
Let $[\![\ \cdots \ ]\!]_{\mathcal{O}}$: $Unit \rightarrow P$ be given, for a unit $U=<(\ldots, C_n \Leftarrow s_n), \ldots>$, by

$$[\![U]\!]_{\mathcal{O}}=\mathcal{O}_U[\![\{(\nu(\varnothing),s_n)\}]\!].$$

The execution of a unit $U=<(\ldots, C_n \Leftarrow s_n), \ldots>$ consists of the creation of an object of class C_n and the execution of its body. Its name is given by $\nu(\varnothing)$, the name of the first object.

Comparison with [ABKR86(a)]

In [ABKR86(a)], an operational semantics for POOL is defined which differs from $\mathcal{O}_U$ in a number of respects: There, a transition relation without labels is used whereas we have a labelled transition relation here; further, in [ABKR86(a)] communication is modeled by means of a so-called *wait* statement as opposed to the release statement we use here; also our use of parameterized expressions and statements is new. All these differences can be seen as minor variations of the semantic definitions and are motivated by the main goal of this paper, which is to relate the operational semantics with the denotational one. There is one major difference, however, which we shall treat in some detail: In definition 4.10 of this paper, $\mathcal{O}_U$ is given as the fixed point of a contraction, whereas in [ABKR86(a)] the operational semantics is defined in terms of finite and infinite sequences of transitions. In order to show the equivalence of both approaches, we now define an operational semantics $\mathcal{O}_U^*$ in the style of [ABKR86(a)], for which we next shall prove that it equals $\mathcal{O}_U$.

DEFINITION 4.12 (Alternative operational semantics)
Let, for a $U \in Unit$, the function

$$\mathcal{O}_U^*: \mathcal{P}_{fin}(LStat) \rightarrow P$$

be given as follows. Let $X \in \mathcal{P}_{fin}(LStat)$ and $\sigma \in \Sigma$. We put for a word $w \in \Sigma_\partial^\infty$:

$$w \in \mathcal{O}_U^*[\![X]\!](\sigma)$$

if and only if one of the following conditions is satisfied:

(1) $w=\sigma_1 \cdots \sigma_n$ and there exist $X_1, \ldots, X_n$ such that

$$<X,\sigma> \ -U,\tau \rightarrow \ <X_1,\sigma_1> \ -U,\tau \rightarrow \cdots -U,\tau \rightarrow \ <X_n,\sigma_n> \ \text{and} \ \forall (\alpha,s) \in X_n \, [s=E]$$

(2) $w=\sigma_1\sigma_2 \cdots$ and there exist $X_1,X_2, \ldots$ such that

$$<X,\sigma> \ -U,\tau \rightarrow \ <X_1,\sigma_1> \ -U,\tau \rightarrow \ <X_2,\sigma_2> \ -U,\tau \rightarrow \cdots$$

(3) $w=\sigma_1 \cdots \sigma_n \cdot \partial$ and there exist $X_1, \ldots, X_n$ such that

$$<X,\sigma> \ -U,\tau \rightarrow \ <X_1,\sigma_1> \ -U,\tau \rightarrow \cdots -U,\tau \rightarrow \ <X_n,\sigma_n>$$

and $\exists(\alpha,s)\in X_n\ [s\neq E]$ and $\neg<X_n,\sigma_n>-U,\tau\rightarrow$

It is not straightforward that the sets $\Theta^*_U[\![X]\!](\sigma)$ are in P, that is, that they are compact; we prove this fact in the following

LEMMA 4.13 *(Compactness of Θ^*_U): For every $X\in\mathcal{P}_{fin}(LStat)$ and $\sigma\in\Sigma$: $\Theta^*_U[\![X]\!](\sigma)$ is compact.*

PROOF
Let $(w_i)_i$ be a sequence of words in $\Theta^*_U[\![X]\!](\sigma)$ ($\subseteq\Sigma^\infty_\partial$), say

$$w_i = \sigma_i^1\sigma_i^2\sigma_i^3\cdots .$$

We show that $(w_i)_i$ has a converging subsequence with its limit in $\Theta^*_U[\![X]\!](\sigma)$. Assume for simplicity that all words w_i are infinite. Since $w_i\in\Theta^*_U[\![X]\!](\sigma)$, for every i, there exist infinite transition sequences such that

$$<X,\sigma>\rightarrow<X_i^1,\sigma_i^1>\rightarrow<X_i^2,\sigma_i^2>\rightarrow\cdots$$

(omitting the labels U,τ). From the definition of $\rightarrow$ it follows that the set

$$\{<X',\sigma'>:\ <X,\sigma>\rightarrow<X',\sigma'>\}$$

is finite. Thus there exists a pair $<X_1,\sigma_1>$ such that for infinitely many i's:

$$<X_i^1,\sigma_i^1> = <X_1,\sigma_1>.$$

Let $f_1:\mathbb{N}\rightarrow\mathbb{N}$ be a monotonic function with, for all i,

$$<X^1_{f(i)},\ \sigma^1_{f(i)}> = <X_1,\sigma_1>.$$

Next we proceed with the subsequence $(w_{f_1(i)})_i$ of $(w_i)_i$ and repeat the above argument, now with respect to the set

$$\{<X',\sigma'>:\ <X_1,\sigma_1>\rightarrow<X',\sigma'>\}.$$

Continuing in this way, we find a sequence of monotonic functions $(f_k)_k$, defining a sequence of subsequences of $(w_i)_i$, and a sequence of configurations $(<X_k,\sigma_k>)_k$ such that

$$\forall k\ \forall j\ \forall i\leqslant k\ [\,\sigma^i_{f_k(j)} = \sigma_i\,]$$

and $<X,\sigma>\rightarrow<X_1,\sigma_1>\rightarrow<X_2,\sigma_2>\rightarrow\cdots$

and moreover such that the sequence $(w_{f_{k+1}(i)})_i$ is a subsequence of the sequence of $(w_{f_k(i)})_i$. Now we define

$$g(i) = f_i(i).$$

Then we have

$$\lim_{i\rightarrow\infty} w_{g(i)} = \sigma_1\sigma_2\sigma_3\cdots.$$

Thus we have constructed a converging subsequence of $(w_i)_i$ with its limit in $\Theta^*_U[\![X]\!](\sigma)$. (In case the words w_i are not all infinite a similar argument can be given.)

It is not difficult to show that $\Theta_U=\Theta^*_U$:

THEOREM 4.14: $\Theta_U=\Theta^*_U$

PROOF

We prove that Θ^*_U is also a fixed point of Φ_U, from which the equality follows. Let $X \in \mathcal{P}_{fin}(LStat)$ such that $\exists (\alpha,s) \in X\, [s \neq E]$, let $\sigma \in \Sigma$ and let $w \in \Sigma^\infty_\delta$. If $w = \delta$ then

$$w \in \Phi_U(\Theta^*_U)(X)(\sigma) \Leftrightarrow w \in \Theta^*_U[\![X]\!](\sigma).$$

Now suppose $w \neq \delta$. We have

$$w \in \Theta^*_U[\![X]\!](\sigma) \Leftrightarrow \exists \sigma' \in \Sigma\, \exists X' \in \mathcal{P}_{fin}(LStat)\, \exists w' \in \Sigma^\infty_\delta$$
$$[<X,\sigma> \rightarrow <X',\sigma'> \wedge\, w = \sigma' \cdot w' \wedge w' \in \Theta^*_U[\![X']\!](\sigma)]$$
$$\Leftrightarrow \text{[definition } \Phi_U\text{]}$$
$$w \in \Phi_U(\Theta^*_U)(X)(\sigma).$$

So we see: $\Theta^*_U = \Phi_U(\Theta^*_U)$.

5. A DENOTATIONAL SEMANTICS FOR POOL

The denotational semantics that is defined in this section was already presented (in a slightly different form) in [ABKR86(b)]. (For a comparison of the two models we refer the reader to the end of this section.)

Our denotational model has a so-called *domain* (a solution of a reflexive domain equation) for its semantic universe. In [BZ82] it was first described how to solve these equations in a metric setting. Then, in [AR88], this approach was generalized in order to deal with equations of the form: $P \cong \cdots P \rightarrow \cdots$, a case that was not covered by [BZ82]. For a quick overview of the main results of [AR88], the reader might want to read section 2 of [ABKR86(b)].

Further, our model is based on the use of *continuations*. For an extensive treatment of continuations and expression continuations, which we shall use as well, we refer to [Go79].

We start with the definition of a domain P, the elements of which we shall call *processes* from now on.

DEFINITION 5.1 (Semantic process domain P)
Let $(\bar{P}, q \in) P$ be a complete ultra-metric space satisfying the following reflexive domain equation:

$$P \cong \{p_0\} \cup id_{\frac{1}{2}}(\Sigma \rightarrow \mathcal{P}_{compact}(Step_P)),$$

where $(\pi, \rho \in) Step_P$ is

$$Step_P = Comp_P \cup Send_P \cup Answer_P,$$

with

$$Comp_P = \Sigma \times P,$$
$$Send_P = Obj \times MName \times Obj \times (Obj \rightarrow P) \times P,$$
$$Answer_P = Obj \times MName \times (Obj \rightarrow (Obj \rightarrow P) \rightarrow P).$$

(The sets $\{p_0\}$, Σ, *Obj*, and *MName* become complete ultra-metric spaces by supplying them with the discrete metric.)

In [AR88], it is described how to find for such an equation a solution which is unique up to isomorphy. Let us try to explain intuitively the intended interpretation of the domain P. First, we observe that in the equation above the subexpression $id_{\frac{1}{2}}$ is necessary only to guarantee that the equation is solvable by defining a contracting functor on $\mathcal{C}$, the category of complete metric spaces (see Appendix I). For a, say, more operational understanding of the equation it does not matter.

A process $p \in P$ is either p_0 or a function from Σ to $\mathcal{P}_{compact}(Step_P)$, the set of all *compact* subsets of

$Step_{\bar{P}}$. The process p_0 is the terminated process. For $p \neq p_0$, the process p has the choice, depending on the current state σ, among the *steps* in the set $p(\sigma)$. If $p(\sigma) = \emptyset$, then no further action is possible, which is interpreted as abnormal termination. For $p(\sigma) \neq \emptyset$, each step $\pi \in p(\sigma)$ consists of some action (for instance, a change of the state σ or an attempt at communication) and a *resumption* of this action, that is to say, the remaining actions to be taken after this action. There are three different types of steps $\pi \in Step_{\bar{P}}$.

First, a step may be an element of $\Sigma \times \bar{P}$, say

$$\pi = <\sigma', p'>.$$

The only action is a change of state: σ' is the new state. Here the process p' is the resumption, indicating the remaining actions process p can do. (When $p' = p_0$ no steps can be taken after this step π.)

Secondly, π might be a *send step*, $\pi \in Send_{\bar{P}}$. In this case we have, say

$$\pi = <\alpha, m, \beta, f, p>,$$

with $\alpha \in Obj, m \in MName, \beta \in Obj$, $f \in (Obj \rightarrow \bar{P})$, and $p \in \bar{P}$. The action involved here consists of an attempt at communication, in which a message is sent to the object α, specifying the method m, together with the parameter β. This is the interpretation of the first three components α, m, and β. The fourth component f, called the *dependent* resumption of this send step, indicates the steps that will be taken after the sender has received the result of the message. These actions will depend on the result, which is modeled by f being a function that yields a process when it is applied to an object name (the result of the message). The last component p, called the *independent* resumption of this send step, represents the steps to be taken after this send step that need *not* wait for the result of the method execution.

Finally, π might be an element of $Answer_{\bar{P}}$, say

$$\pi = <\alpha, m, g>$$

with $\alpha \in Obj$, $m \in MName$, and $g \in (Obj \rightarrow (Obj \rightarrow \bar{P}) \rightarrow {}^1\bar{P})$. It is then called an *answer step*. The first two components of π express that the object α is willing to accept a message that specifies the method m. The last component g, the resumption of this answer step, specifies what should happen when an appropriate message actually arrives. The function g is then applied to the parameter in this message and to the dependent resumption of the sender (specified in its corresponding send step). It then delivers a process which is the resumption of the sender and the receiver *together*, which is to be composed in parallel with the independent resumption of the send step.

We now define a semantic operator for the *parallel composition* (or *merge*) of two processes, for which we shall use the symbol $\|$. It is *auxiliary* in the sense that it does not correspond to a syntactic operator in the language POOL.

DEFINITION 5.2 (Parallel composition)
Let $\| : \bar{P} \times \bar{P} \rightarrow \bar{P}$ be such that it satisfies the following equation:

$$p \| q = \lambda\sigma \cdot ((p(\sigma) \mathbin{\hat{\|}} q) \cup (q(\sigma) \mathbin{\hat{\|}} p) \cup (p(\sigma) \mid_o q(\sigma))),$$

for all $p, q \in \bar{P} \setminus \{p_0\}$, and such that $p_0 \| q = q \| p_0 = p_0$. Here, $X \mathbin{\hat{\|}} q$ and $X \mid_o Y$ are defined by:

$$X \mathbin{\hat{\|}} q = \{\pi \mathbin{\hat{\|}} q : \pi \in X\},$$

$$X \mid_o Y = \bigcup \{\pi \mid_o \rho : \pi \in X, \rho \in Y\},$$

where $\pi \mathbin{\hat{\|}} q$ is given by

$$<\sigma', p'> \mathbin{\hat{\|}} q = <\sigma', p' \| q>,$$

$$<\alpha, m, \beta, f, p> \mathbin{\hat{\|}} q = <\alpha, m, \beta, f, p \| q>, \text{ and}$$

$$<\alpha, m, g> \mathbin{\hat{\|}} q = <\alpha, m, \lambda\beta \cdot \lambda h \cdot (g(\beta)(h) \| q)>,$$

and $\pi|_\sigma\rho$ by

$$\pi|_\sigma\rho = \begin{cases} \{<\sigma,\ g(\beta)(f)\|p>\} & \text{if } \pi=<\alpha,m,\beta,f,p> \text{ and } \rho=<\alpha,m,g> \\ & \text{or } \rho=<\alpha,m,\beta,f,p> \text{ and } \pi=<\alpha,m,g> \\ \varnothing & \text{otherwise.} \end{cases}$$

We observe that this definition is self-referential, since the merge operator occurs at the right hand side of the definition. For a formal justification of this definition see the appendix of [ABKR86(b)], where the merge operator is given as the unique fixed point of a contraction on $\overline{P}\times\overline{P}\to^1\overline{P}$.

Since we intend to model parallel composition by interleaving, the merge of two processes p and q consists of three parts. The first part contains all possible first steps of p followed by the parallel composition of their respective resumptions with q. The second part contains similarly the first steps of q. The last part contains the communication steps that result from two matching communication steps taken simultaneously by process p and q. For $\pi\in Step_{\overline{P}}$ the definition of $\pi|q$ is straightforward. The definition of $\pi|_\sigma\rho$ is more involved. It is the empty set if π and ρ do not match. Now suppose they do match, say $\pi=<\alpha,m,\beta,f,p>$ and $\rho=<\alpha,m,g>$. Then π is a *send* step, denoting a request to object α to execute the method m, and ρ is an *answer* step, denoting that the object α is willing to accept a message that requests the execution of the method m. In $\pi|_\sigma\rho$, the state σ remains unaltered. Since g, the third component of ρ, represents the meaning of the execution of the method m, it needs the parameter β that is specified by α. Moreover, g depends on the dependent resumption f of the send step π. This explains why both β and f are supplied as arguments to the function g. Now it can be seen that $g(\beta)(f)\|p$ represents the resumption of the sender and the receiver together. (In order to get more insight in this definition it is advisable to return to it after having seen the definition of the semantics of an answer statement.)

The merge operator is associative, which can easily be proved as follows. Define

$$\epsilon = \sup_{p,q,r\in\overline{P}}\{d_{\overline{P}}((p\|q)\|r, p\|(q\|r))\}$$

Then, using the fact that the operator $\|$ satisfies the equation above, one can show that $\epsilon\leqslant \frac{1}{2}\cdot\epsilon$. Therefore $\epsilon=0$, and $\|$ is associative.

Now we come to the definition of the semantics of expressions and statements. We specify a pair of functions $<\mathfrak{D}_E,\mathfrak{D}_S>$ of the following type:

$$\mathfrak{D}_E\colon L_E\to AObj\to Cont_E\to^1\overline{P},$$

$$\mathfrak{D}_S\colon L_S\to AObj\to Cont_S\to^1\overline{P}$$

where

$$Cont_E = Obj\to\overline{P} \text{ and } Cont_S=\overline{P}.$$

Let $s\in L_S$, $\alpha\in AObj$, and $p\in\overline{P}$. The semantic value of the statement s is given by

$$\mathfrak{D}_S[\![s]\!](\alpha)(p).$$

The object name α represents the object that executes s. Secondly, the semantic value of s depends on its so-called *continuation* p: the semantic value of everything that will happen after the execution of s. The main advantage of the use of continuations is that it enables us to describe the semantics of expressions in a concise and elegant way.

The semantic value of an expression $e\in L_E$, for an object α and an expression continuation $f\in Cont_E$, is given by

$$\mathfrak{D}_E[\![e]\!](\alpha)(f).$$

The evaluation of an expression e always results in a value (an element of *Obj*), upon which the

continuation of such an expression generally depends. The function f, when applied to the result β of e, will yield a process $f(\beta) \in \bar{P}$ that is to be executed after the evaluation of e.

Please note the difference between the notions of *resumption* and *continuation*. A resumption is a part of a semantic step $\pi \in Step_{\bar{P}}$, indicating the remaining steps to be taken after the current one. A continuation, on the other hand, is an argument to a semantic function. It may appear as a resumption in the result. A good example of this is the definition of $\hat{F}_S(x \leftarrow e)$ (in definition 5.3(S1)) below.

DEFINITION 5.3 (Semantics of expressions and statements)
Let

$$Q_E = L_E \rightarrow AObj \rightarrow Cont_E \rightarrow {}^1\bar{P}$$

$$Q_S = L_S \rightarrow AObj \rightarrow Cont_S \rightarrow {}^1\bar{P}.$$

For every unit $U \in Unit$ we define a pair of functions $\mathfrak{D}_U = <\mathfrak{D}_E, \mathfrak{D}_S>$ by

$$\mathfrak{D}_U = \text{Fixed Point } (\Psi_U),$$

where

$$\Psi_U: (Q_E \times Q_S) \rightarrow (Q_E \times Q_S)$$

is defined by induction on the structure of L_E and L_S by the following clauses. For $F = <F_E, F_S>$ we denote $\Psi_U(F)$ by $\hat{F} = <\hat{F}_E, \hat{F}_S>$. Let $p \in Cont_S = \bar{P}$, $f \in Cont_E = Obj \rightarrow \bar{P}$ and $\alpha \in AObj$. Then:

EXPRESSIONS

(E1, instance variable)

$$\hat{F}_E(x)(\alpha)(f) = \lambda\sigma\cdot\{<\sigma, f(\sigma_1(\alpha)(x))>\}.$$

The value of the instance variable x is looked up in the first component of the state σ supplied with the name α of the object that is evaluating the expression. The continuation f is then applied to the resulting value.

(E2, temporary variable)

$$\hat{F}_E(u)(\alpha)(f) = \lambda\sigma\cdot\{<\sigma, f(\sigma_2(\alpha)(u))>\}$$

(E3, send expression)

$$\hat{F}_E(e_1!m(e_2))(\alpha)(f) = \hat{F}_E(e_1)(\alpha)(\lambda\beta_1\cdot\hat{F}_E(e_2)(\alpha)(\lambda\beta_2\cdot\lambda\sigma\cdot\{<\beta_1, m, \beta_2, f, p_0>\})).$$

The expressions e_1 and e_2 are evaluated successively. Their results correspond to the formal parameters β_1 and β_2 of their respective continuations. Finally, a send step is performed. The object name β_1 refers to the object to which the message is sent; β_2 represents the parameter for the execution of the method m. Besides these values and the method name m, the final step $<\beta_1, m, \beta_2, f, p_0>$ also contains the expression continuation f of the send expression as the dependent resumption. If the attempt at communication succeeds, this continuation will be supplied with the result of the method execution. The independent resumption of this send step is initialized at p_0.

(E4, new-expression)

$$\hat{F}_E(\text{ new } (C))(\alpha)(f) = \lambda\sigma\cdot\{<\sigma', f(\beta) \| F_S(s_C)(\beta)(p_0)>\},$$

where

$$\beta = \nu(\sigma_3),$$

$$\sigma' = <\sigma_1, \sigma_2, \sigma_3 \cup \{\beta\}>, \quad C \Leftarrow s_C \in U.$$

A new object of class C is created. It is called $\nu(\sigma_3)$: the function ν supplied with the set of all object

names currently in use yields a name that is not yet being used. The state σ is changed by expanding the set σ_3 with the new name β. The process $F_S(s_C)(\beta)(p_0)$ is the meaning of the body of the new object β with p_0 as a nil continuation. It is composed in parallel with $f(\beta)$, the process resulting from the application of the continuation f to β, the result of the evaluation of this new-expression. We are able to perform this parallel composition because we know from f what should happen after the evaluation of this new-expression, so here the use of continuations is essential.

(E5, sequential composition)

$$\hat{F}_E(s;e)(\alpha)(f)=\hat{F}_S(s)(\alpha)(\hat{F}_E(e)(\alpha)(f)).$$

The continuation of s is the execution of e followed by f. Note that a semantic operator for sequential composition is absent: the use of continuations has made it superfluous.

(E6, **self**)

$$\hat{F}_E(\textbf{ self })(\alpha)(f)=f(\alpha).$$

The continuation of f is supplied with the value of the expression **self**, that is, the name of the object executing this expression. We use $f(\alpha)$ instead of $\lambda\beta\cdot\{<\sigma,f(\alpha)>\}$ in this definition wishing to express that the value of self is immediately present: it does not take a step to evaluate it.

STATEMENTS

(S1, assignment to an instance variable)

$$\hat{F}_S(x\leftarrow e)(\alpha)(p)=\hat{F}_E(e)(\alpha)(\lambda\beta\cdot\lambda\sigma\cdot\{<\sigma',p>\}),$$

where $\sigma'=\sigma\{\beta/\alpha,x\}$. The expression e is evaluated and the result β is assigned to x.

(S2, assignment to a temporary variable)

$$\hat{F}_S(u\leftarrow e)(\alpha)(p)=\hat{F}_E(e)(\alpha)(\lambda\beta\cdot\lambda\sigma\cdot\{<\sigma',p>\}),$$

where $\sigma'=\sigma\{\beta/\alpha,u\}$.

(S3, answer statement)

$$\hat{F}_S(\textbf{ answer } m)(\alpha)(p)=\lambda\sigma\cdot\{<\alpha,m,g_m>\},$$

where

$$g_m=\lambda\beta\cdot\lambda f\cdot\lambda\sigma\cdot\{<\sigma',F_E(e_m)(\alpha)(\lambda\beta'\cdot\lambda\bar{\sigma}\cdot\{<\bar{\sigma}',f(\beta')\|p>\})>\},$$

with

$$\sigma'=\sigma\{\beta/\alpha,u_m\},$$

$$\bar{\sigma}'=\bar{\sigma}\{\sigma_2(\alpha)(u_m)/\alpha,u_m\},$$

$$m\Leftarrow<u_m,e_m>\in U.$$

The function g_m represents the execution of the method m followed by its continuation. This function g_m expects a parameter β and an expression continuation f, both to be received from an object sending a message specifying the method m. The execution of the method m consists of the evaluation of the expression e_m, which is used in the definition of m, preceded by a state transformation in which the temporary variable u_m is initialized at the value β. After the execution of e, this temporary variable is set back to its old value again. Next, both the continuation of the sending object, supplied with the result β' of the execution of the method m, and the given continuation p are to be executed in parallel. This explains the last resumption: $f(\beta')\|p$.

Now that we have defined the semantics of send expressions and answer statements let us briefly return to the definition of $\pi|_\sigma\rho$ (definition 5.2). Let $\pi=<\alpha,m,\beta,f,q>$ (the result from the elaboration of a send expression) and $\rho=<\alpha,m,g>$ (resulting from an answer statement). Then $\pi|_\sigma\rho$ is

defined as

$$\pi|_{\sigma}\rho=\{<\sigma,g(\beta)(f)\|q>\}.$$

We see that the execution of the method m proceeds in parallel with the independent resumption q of the sender. Now that we know how g is defined we have

$$g(\beta)(f)=\lambda\sigma\cdot\{<\sigma',F_E(e_m)(\alpha)(\lambda\beta'\cdot\lambda\bar{\sigma}\cdot\{<\bar{\sigma}',f(\beta')\|p>\})>\}.$$

The continuation of the execution of m is given by $\lambda\beta'\cdot\lambda\bar{\sigma}\cdot\{<\bar{\sigma}',f(\beta')\|p>\}$, which consists of a state transformation followed by the parallel composition of the continuations f and p. This represents the fact that after the rendez-vous, during which the method is executed, the sender and the receiver of the message can proceed in parallel again. (Of course, the independent resumption q may still be executing at this point.) Moreover, the result β' of the method execution is passed on to the continuation f of the send expression.

(S4, sequential composition)

$$\hat{F}_S(s_1;s_2)(\alpha)(p)=\hat{F}_S(s_1)(\alpha)(\hat{F}_S(s_2)(\alpha)(p)).$$

(S5, conditional)

$$\begin{aligned}&\hat{F}_S(\textbf{ if } e \textbf{ then } s_1 \textbf{ else } s_2 \textbf{ fi })(\alpha)(p)=\\&\quad\hat{F}_E(e)(\alpha)(\lambda\beta\cdot \text{ if } \beta=tt\\&\qquad\qquad\text{then } \hat{F}_S(s_1)(\alpha)(p)\\&\qquad\qquad\text{else } \hat{F}_S(s_2)(\alpha)(p)\\&\qquad\qquad\text{fi }).\end{aligned}$$

(S6, loop statement)

$$\begin{aligned}&\hat{F}_S(\textbf{ do } e \textbf{ then } s \textbf{ od })(\alpha)(p)=\\&\quad\lambda\sigma\cdot\{<\sigma,\ \hat{F}_E(e)(\alpha)(\lambda\beta)\cdot \text{ if } \beta=tt\\&\qquad\qquad\text{then } \hat{F}_S(s)(\alpha)(F_S(\textbf{ do } e \textbf{ then } s \textbf{ od })(\alpha)(p))\\&\qquad\qquad\text{else } p\\&\qquad\qquad\text{fi })>\}.\end{aligned}$$

(End of definition 5.3.)

It is not difficult to prove that Ψ_U is a contraction and hence has a unique fixed point $\mathfrak{D}_U$. As a matter of fact, we have defined Ψ_U such that it satisfies this property. Note that the original functions F_E and F_S have been used in only three places: in the definition of the semantics of a new-expression, of an answer statement, and of a loop statement. Here the syntactic complexity of the defining part is not necessarily less than that of what is being defined. At those places, we have ensured that the definition is "guarded" by some step $\lambda\sigma\cdot\{<\sigma',\ldots>\}$. It is easily verified that in this manner the contractiveness of Ψ_U is indeed implied.

DEFINITION 5.4 (Denotational semantics of a unit)
We define $[\![\cdots]\!]_{\mathfrak{D}}$: $Unit\to\bar{P}$. For a unit $U\in Unit$, with $U=<(\ldots, C_n\Leftarrow s_n),\ldots>$, we set

$$[\![U]\!]_{\mathfrak{D}}=\mathfrak{D}_S[\![s_n]\!](\nu(\varnothing))(p_0).$$

The execution of a unit always starts with the creation of an object of class C_n and the execution of

its body. Therefore, the meaning of a unit U is given by the denotational meaning of s_C, the body of class C_n, supplied with $\nu(\varnothing)$, denoting the name of the first active object, and with p_0, the empty continuation.

Comparison with [ABKR86(b)]

There are some differences between the denotational semantics $<\mathcal{D}_E,\mathcal{D}_S>$ presented here and the denotational semantics given in [ABKR86(b)]: The former model is given as the fixed point of a contraction Ψ_U and does not use so-called *environments* to deal with process creation (**new**(*C*)) and the meaning of the execution of a method (**answer** *m*); the latter model is defined without the use of a contraction and *does* use environments. In [ABKR86(b)], the semantics of a unit U is given with the help of a special environment γ_U, which contains information about the class and method definitions in U and is obtained as the fixed point of a suitably defined contraction. Another difference is the way the loop statement is treated: In this paper, the definition of its semantics fits smoothly in the definition of $<\mathcal{D}_E,\mathcal{D}_S>$ as a fixed point. In [ABKR86(b)], a contraction is defined especially for this case.

Another way to express these differences is that the three constructs for recursion present in POOL (i.e., the new expression, the answer and the loop statement) are treated here by means of one fixed point definition, whereas in [ABKR86(b)], environments are used for the first two forms of recursion and a specially defined contraction for the last one. However, we state (without proof) that the two definitions are equivalent: it is straightforward how to translate the one approach into the other.

An additional difference between the denotational semantics of a unit given here and the one presented in [ABKR86(b)] is the presence of a semantic representation of the standard objects in the latter, whereas these are not treated in this section. As mentioned before, we do not treat standard objects now because we want to concentrate on the correctness proof. In order to show, however, that our proof (to be presented in section 7) can also deal with standard objects, we shall extend, in Appendix III, both our operational and our denotational semantics with a semantic representation of standard objects, and prove that the correctness result still holds for these extended models.

6. An intermediate semantics

After having defined an intermediate semantics $\mathcal{O}_U$ for $\mathcal{P}_{fin}$ (*LStat*) and a denotational semantics $\mathcal{D}_U$ for L_E and L_S we shall, in the next section, discuss the relationship between the two. As we did in section 2, we shall compare $\mathcal{O}_U$ and $\mathcal{D}_U$ by relating both to an intermediate semantics $\mathcal{O}_U'$: $\mathcal{P}_{fin}$ (*LStat*)$\to\overline{P}$, the definition of which is the subject of this section.

Definition. 6.1 (Intermediate semantics $\mathcal{O}_U'$)
Let $U\in$ *Unit*. Let $\mathcal{O}_U'$: $\mathcal{P}_{fin}$ (*LStat*)$\to\overline{P}$ be given by

$$\mathcal{O}_U' = \textit{Fixed Point}\ (\Phi_U'),$$

where

$$\Phi_U'\colon (\mathcal{P}_{fin}\ (\textit{LStat})\to\overline{P})\to(\mathcal{P}_{fin}\ (\textit{LStat})\to\overline{P})$$

is defined, for $F\in\mathcal{P}_{fin}(\textit{LStat})\to\overline{P}$ and $X\in\mathcal{P}_{fin}$ (*LStat*), as follows.
If $\forall\alpha\forall s\ [(\alpha,s)\in X\Rightarrow s=E]$, then $\Phi_U'(F)(X)=p_0$. Otherwise we have

$$\Phi_U'(F)(X)=\lambda\sigma\cdot(C_F\cup S_F\cup A_F)$$

where

$$C_F = \{<\sigma',F(X')>\colon <X,\sigma> -U,\tau\to <X',\sigma'>\},$$

$$S_F = \{<\beta_1, m, \beta_2, \lambda\beta\cdot F(\{(\alpha,\psi(\beta))\}),\ F(X')>:$$
$$<X,\sigma> -U,\ (\alpha,\beta_1!m(\beta_2))\rightarrow <\{(\alpha,\psi)\}\cup X',\sigma>\},$$
$$A_F = \{<\alpha, m, g_m>:\ <X,\sigma> -U,\ (\alpha?m)\rightarrow <\{(\alpha,s)\}\cup X',\sigma>\}$$

with

$$g_m = \lambda\beta\cdot\lambda f\cdot(\lambda\overline{\sigma}\cdot\{<\overline{\sigma}', \mathcal{D}_E[\![e_m]\!](\alpha)(\lambda\beta'\cdot\lambda\hat{\sigma}\cdot\{<\hat{\sigma}', f(\beta')\| F(\{(\alpha,s)\})>\})>\} \parallel F(X')),$$

and

$$\overline{\sigma}' = \overline{\sigma}\{\beta/\alpha, u_m\},$$
$$\hat{\sigma}' = \hat{\sigma}\{\overline{\sigma}_2(\alpha)(u_m)/\alpha, u_m\},$$
$$m \Leftarrow <u_m, e_m> \in U.$$

(It is straightforward to show that Φ_U' is a contraction.)

The function $\mathcal{O}_U'$ differs from the operational semantics $\mathcal{O}_U$ in two ways. First, its range is the semantic universe $\overline{P}$, which is used for the denotational semantics $\mathcal{D}_U$, instead of $\underline{P}$, the semantic universe of $\mathcal{O}_U$: For every set $X \in \mathcal{P}_{fin}(LStat)$ the function $\mathcal{O}_U'$ yields a *process* $\mathcal{O}_U'(X) \in \overline{P}$, rather than a function from states to sets of streams of states. Secondly, in addition to the computation steps (indicated by the set C_F above) single-sided communication steps are present in $\mathcal{O}_U'(X)$ (indicated by S_F and A_F, for send and answer steps), whereas $\mathcal{O}_U(X)$ contains only computation steps. On the other hand, the similarity between the definitions of $\mathcal{O}_U$ and $\mathcal{O}_U'$ is obvious: both are based on the transition relation $-U\rightarrow$ for $\mathcal{P}_{fin}(LStat)$.

At first sight, two facts regarding the relation between $\mathcal{O}'_U$ and $\mathcal{D}_U$ can be mentioned. First, they have the same range, that is, the semantic universe $\overline{P}$ of processes, in which single-sided communication actions are visible. Secondly, $\mathcal{D}_U$ is defined compositionally with the use of semantic operators (like the merge $\|$), whereas the definition of $\mathcal{O}_U'$ is based, as was mentioned above, on the transition relation $-U\rightarrow$.

In the next section the relationship between $\mathcal{O}_U$, $\mathcal{O}_U'$ and $\mathcal{D}_U$ will be formally expressed. Let us, for the time being, try to elucidate the definition of $\mathcal{O}_U'$ above by explaining what communication steps are present in $\mathcal{O}_U'(X)$.

Corresponding with every send transition of the form

$$<X,\sigma> -U,\ (\alpha,\beta_1!m(\beta_2))\rightarrow <\{(\alpha,\psi)\})\cup X',\sigma>$$

the set $\mathcal{O}_U'(X)(\sigma)$, for a state $\sigma \in \Sigma$, contains a send step of the form

$$<\beta_1,\ m,\ \beta_2,\ \lambda\beta\cdot\mathcal{O}_U'(\{(\alpha,\psi(\beta))\}),\ \mathcal{O}_U'(X')>.$$

Here β_1, m and β_2 indicate that a message specifying the method m with parameter β_2 is sent to the object β_1. The dependent resumption of this send step is $\lambda\beta\cdot\mathcal{O}_U'(\{(\alpha,\psi(\beta))\})$: the meaning of the statement that will be executed by α as soon as it receives the result β of the message. The last component of this send step, the independent resumption, consists of $\mathcal{O}_U'(X')$, which is the meaning of all the statements executed by objects other than α. Thus it is reflected that these objects need not wait till the message is answered; they may proceed in parallel.

Next, $\mathcal{O}_U'(X)(\sigma)$ can contain some answer steps. For every answer transition

$$<X,\sigma> -U,\ (\alpha?m)\rightarrow <\{(\alpha,s)\}\cup X',\sigma>$$

the set $\mathcal{O}_U'(X)(\sigma)$ includes an answer step

$$<\alpha, m, g_m>,$$

with g_m as in the definition above. It indicates that the object α is willing to answer a message

specifying the method m, while the resumption g_m indicates what should happen when an appropriate message arrives. This function g_m, when supplied with a parameter β and a dependent resumption f (both to be received from the sending object), consists of the parallel composition of the process $\Theta_U'(X')$ together with the process

$$\lambda\bar{\sigma}\cdot\{<\bar{\sigma}',\ \mathcal{D}_E[\![e_m]\!](\alpha)(\lambda\beta'\cdot\lambda\hat{\sigma}\cdot\{<\hat{\sigma}',\ f(\beta')\|\Theta_U'(\{(\alpha,s)\})>\})>\}\ .$$

(Note that we have used the function $\mathcal{D}_E$ here; the definition of Θ_U' therefore depends on its definition.) The process $\Theta_U'(X')$ stands for the meaning of all the statements executed by objects other than the object α: these objects may proceed in parallel with the execution of the method m, the meaning of which is indicated by the second process. Its interpretation is the same as in the definition of $\mathcal{D}_S[\![$ **answer** $m\ ]\!](\alpha)(p)$ in the previous section but for the fact that here the last resumption of this process consists of $f(\beta')\|\Theta_U'(\{(\alpha,s)\})$: the parallel composition of the dependent resumption of the sender (supplied with the result β' of the method m) and the meaning of the statement s, with which the object α will continue after it has answered the message.

7. Semantic correctness

We are now ready to establish the main result of this paper. We shall relate the operational semantics Θ_U and the denotational semantics $\mathcal{D}_U$ by first comparing Θ_U and Θ_U', the intermediate semantics defined in the previous section, and next comparing Θ_U' and $\mathcal{D}_U$. These relationships will be formally expressed by means of suitably defined abstraction operations. From this we shall deduce the fact that

$$[\![U]\!]_\Theta = abstr([\![U]\!]_\mathcal{D}),$$

where abstr: $\bar{P}\to P$ is such an abstraction operation.

Part 1: Comparing Θ_U and Θ_U'

We start with the definition of abstr: $\bar{P}\to P$, which relates the semantic universes P and $\bar{P}$ of Θ_U and Θ_U'.

Definition 7.1 (Abstraction operation *abstr*)
Let *abstr*: $\bar{P}\to P$ be defined as follows. We set $abstr(p_0)=\{\epsilon\}$. If $p\in\bar{P}\setminus\{p_0\}$, then

$$abstr(p)=\lambda\sigma\cdot\begin{cases}\{\partial\} & \text{if } p(\sigma)\cap Comp_{\bar{P}}=\varnothing\\ \bigcup\{\sigma'\cdot abstr(p')(\sigma'):\ <\sigma',p'>\in p(\sigma)\} & \text{otherwise,}\end{cases}$$

where $Comp_{\bar{P}}=\Sigma\times\bar{P}$. (Formally, we can define this operation correctly by giving it as the fixed point of a suitably defined contraction on $\bar{P}\to P$: See Appendix II for an extensive formal treatment of the function *abstr*.)

The function *abstr* transforms a process $p\in\bar{P}$ into a function $abstr(p)\in P=\Sigma\to\mathcal{P}_{nc}(\Sigma_\partial^\infty)$, which yields for every $\sigma\in\Sigma$ a set $abstr(p)(\sigma)$ of streams. (If one regards the process p as a tree-like structure, these streams can be considered the branches of p.) If $p(\sigma)\cap Comp_{\bar{P}}=\varnothing$, that is, if $p(\sigma)$ is empty or contains only single-sided communication steps, then we have a case of deadlock because, operationally, single-sided communication is not possible. Therefore we then have: $abstr(p)(\sigma)=\{\partial\}$. If, however, $p(\sigma)$ does contain a computation step $<\sigma',p'>$, then we have: $\sigma'\cdot abstr\ (p')(\sigma')\subseteq abstr\ (p)(\sigma)$. The changed state σ' is concatenated with $abstr(p')(\sigma')$, in which σ' is passed through to *abstr* applied to p', the resumption of $<\sigma',p'>$. Thus the effect of different state transformations occurring subsequently in p is accumulated.

Next, we use the operation *abstr* to relate Φ_U and Φ_U'.

Theorem 7.2 *(Relating Φ_U and Φ_U'):* $\forall F\in\mathcal{P}_{fm}(LStat)\to\bar{P}\ [\Phi_U(abstr\circ F)=abstr\circ(\Phi_U'(F))]$

PROOF
Let $F \in \mathcal{P}_{fin}(LStat) \to \bar{P}$, $X \in \mathcal{P}_{fin}(LStat)$ and $\sigma \in \Sigma$. Suppose $\neg\forall\alpha\forall s\,[(\alpha,s) \in X \Rightarrow s = E]$. If $\neg\langle X,\sigma\rangle - U, \tau \to$, then

$$\Phi_U(abstr \circ F)(X)(\sigma) = \{\partial\}$$
$$= abstr(\Phi_U'(F)(X))(\sigma)$$

since $\Phi_U'(F)(X)(\sigma) \cap Comp_{\bar{P}} = \varnothing$. (Recall that $Comp_{\bar{P}} = \Sigma \times \bar{P}$.) If $\langle X,\sigma\rangle - U, \tau \to$ we have

$$\Phi_U(abstr \circ F)(X)(\sigma) = \bigcup \{\sigma' \cdot (abstr \circ F)(X')(\sigma') : \langle X,\sigma\rangle - U, \tau \to \langle X',\sigma'\rangle\}$$
$$= \bigcup \{\sigma' \cdot (abstr(F)(X'))(\sigma') : \langle X,\sigma\rangle - U, \tau \to \langle X',\sigma'\rangle\}$$
$$= [\text{see definition 6.1}]$$
$$abstr(\lambda\sigma \cdot C_F)(\sigma)$$
$$= abstr(\lambda\sigma \cdot (C_F \cup S_F \cup A_F))(\sigma)$$
$$= abstr(\Phi_U'(F)(X))(\sigma)$$
$$= (abstr \circ \Phi_U'(F))(X)(\sigma).$$

Since Φ_U and Φ_U' are contractions and thus have unique fixed points, the following corollary is straightforward:

COROLLARY 7.3: $\Theta_U = abstr \circ \Theta_U'$

Part 2. Comparing Θ_U' and $\mathcal{D}_U$.
In order to compare Θ_U': $\mathcal{P}_{fin}(LStat) \to \bar{P}$ and $\mathcal{D}_U \in Q_E \times Q_S$ we define an extension of $\mathcal{D}_U$ $(= \langle \mathcal{D}_E, \mathcal{D}_S \rangle)$ in two steps. First, we define $\mathcal{D}_U'$ $(= \langle \mathcal{D}_E', \mathcal{D}_S' \rangle) \in Q_E' \times Q_S'$, with

$$Q_E' = L_E' \to AObj \to Cont_E \to^1 \bar{P},$$
$$Q_S' = L_S' \to AObj \to Cont_S \to^1 \bar{P},$$

which is as $\mathcal{D}_U$ but with the extended sets of expressions and statements, L_E' and L_S', for its domain. (Recall that L_S' is used in the definition of $LStat = AObj \times L_S'$.) Next, we extend $\mathcal{D}_U'$ to $\mathcal{D}_U^*$: $\mathcal{P}_{fin}(LStat) \to \bar{P}$, which takes sets of labelled statements for its arguments.

DEFINITION 7.4 ($\mathcal{D}_U'$)
Let Ψ'_U: $(Q_E' \times Q_S') \to (Q_E' \times Q_S')$ be defined as follows. For $F = \langle F_E, F_S \rangle$, we denote $\Psi'_U(F)$ by $\bar{F} = \langle \bar{F}_E, \bar{F}_S \rangle$. Let $\alpha \in AObj$, $p \in Cont_S = \bar{P}$ and $f \in Cont_E = Obj \to \bar{P}$. Now $\bar{F}$ is defined similarly to $\Psi_U(F)$ (definition 5.3) but with the following clauses added:

$$\bar{F}_E(\beta)(\alpha)(f) = f(\beta), \quad \text{for } \beta \in Obj \supseteq AObj,$$
$$\bar{F}_E((e,\varphi))(\alpha)(f) = \bar{F}_E(e)(\alpha)(\lambda\beta \cdot \bar{F}_E(\varphi(\beta))(\alpha)(f))$$
$$\bar{F}(E)(\alpha)(p) = p$$
$$\bar{F}_S\ (\mathbf{release}(\beta,s))(\alpha)(p) = p \,||\, \bar{F}_S(s)(\beta)(p_0)$$
$$\bar{F}_S((e,\psi))(\alpha)(p) = \bar{F}_E(e)(\alpha)(\lambda\beta \cdot \bar{F}_S(\psi(\beta))(\alpha)(p)).$$

Finally, we set

$$\mathcal{D}_U' = \langle \mathcal{D}_E', \mathcal{D}_S' \rangle$$
$$= \text{Fixed Point } (\Psi'_U).$$

The meaning of (e,φ) is obtained by first evaluating the expression e, then substituting the result β into the parameterized expression φ and finally evaluating the expression $\varphi(\beta)$. The interpretation of $\mathcal{D}_S'[\![(e,\psi)]\!]$ is similar. In $\mathcal{D}_S'[\![\text{release}(\beta,s)]\!](\alpha)(p)$, the meaning of the statement s (when executed by the object β and with the empty continuation p_0) is computed and composed in parallel with the process p, the continuation of the release statement.

DEFINITION 7.5 ($\mathcal{D}_U^*$)
Let $\mathcal{D}_U^*$: $\mathcal{P}_{fin}(LStat)\to\overline{P}$ be given by

$$\mathcal{D}_U^* = (\tilde{\mathcal{D}_U'}),$$

where $\sim$: $(Q_E\times Q_S)\to(\mathcal{P}_{fin}(LStat)\to\overline{P})$ is defined as follows: If $F=<F_E, F_S>$, then $\sim(F)$, here being denoted by F is given by

$$\tilde{F}(\{(\alpha_1,s_1),\ldots,(\alpha_k,s_k)\}) = F_S(s_1)(\alpha_1)(p_0)\| \cdots \| F_S(s_k)(\alpha_k)(p_0).$$

(We put $\tilde{F}(\varnothing)=p_0$.)

Note that we have: $\tilde{F}(X\cup Y)=\tilde{F}(X)\|\tilde{F}(Y)$.

The omission of parentheses in the parallel composition above is justified by the fact that $\|$ is associative.

Given a finite set X of labelled statements (α_i,s_i), the value of $\mathcal{D}_U^*(X)$ is obtained by first computing the semantics of every labelled statement $(\alpha_i,s_i)\in X$. This is given by $\mathcal{D}_S[\![s_i]\!](\alpha_i)(p_0)$, where the label α_i indicates the name of the object executing the statement and where p_0 indicates that after s_i nothing remains to be done. Next, all the resulting processes are composed in parallel.

Now that we have extended the domain of $\mathcal{D}_U$ to $\mathcal{P}_{fin}(LStat)$ we are ready to prove the fact that $\mathcal{D}_U^*=\mathcal{O}_U'$. It is a straightforward corollary of theorem 7.7 below. The proof of this theorem makes use of the following

LEMMA 7.6
For all $\alpha\in AObj$ and $\psi\in L_{PS}$ we have:

$$\forall\beta\,[\Phi_U'(\mathcal{D}_U^*)(\{(\alpha,\psi(\beta))\}) = \mathcal{D}_U^*(\{(\alpha,\psi(\beta))\})] \Rightarrow$$
$$\forall e\in L_E'\,[\Phi_U'(\mathcal{D}_U^*)(\{(\alpha,(e,\psi))\}) = \mathcal{D}_U^*(\{(\alpha,(e,\psi))\})]$$

PROOF
The proof uses induction on the complexity of expressions. We treat two simple basic cases, being (lazy and) confident that these will show the reader how to proceed in the other cases. So let $\alpha\in AObj$ and $\psi\in L_{PS}$ and suppose

$$\forall\beta\,[\Phi_U'(\mathcal{D}_U^*)(\{(\alpha,\psi(\beta))\}) = \mathcal{D}_U^*(\{(\alpha,\psi(\beta))\})].$$

For $e=\beta$ we have

$$\begin{aligned}
\Phi_U'(\mathcal{D}_U^*)(\{(\alpha,(\beta,\psi))\}) &= \Phi_U'(\mathcal{D}_U^*)(\{(\alpha,\psi(\beta))\}) \\
&= [\text{ hypothesis }] \\
&\quad \mathcal{D}_U^*(\{(\alpha,\psi(\beta))\}) \\
&= \mathcal{D}_S'[\![\psi(\beta)]\!](\alpha)(p_0) \\
&= \mathcal{D}_S'[\![(\beta,\psi)]\!](\alpha)(p_0) \\
&= \mathcal{D}_U^*(\{(\alpha,(\beta,\psi))\});
\end{aligned}$$

if $e=\beta_1!m(\beta_2)$ then

$$\begin{aligned}
\Phi_U'(\mathcal{D}_U^*)(\{(\alpha,(\beta_1!m(\beta_2),\psi))\}) &= \lambda\sigma\cdot\{<\beta_1,m,\beta_2,\lambda\beta\cdot\mathcal{D}_U^*(\{(\alpha,\psi(\beta))\}),\ p_0>\}\\
&= \lambda\sigma\cdot\{<\beta_1,m,\beta_2,\lambda\beta\cdot\mathcal{D}_S'[\![\psi(\beta)]\!](\alpha)(p_0),\ p_0>\}\\
&= \mathcal{D}_E'[\![\beta_1]\!](\alpha)(\lambda\beta'_1\cdot\mathcal{D}_E'[\![\beta_2]\!](\alpha)(\lambda\beta'_1\cdot\\
&\quad \lambda\sigma\cdot\{<\beta'_1,m,\beta'_2,\ \lambda\beta\cdot\mathcal{D}_S'[\![\psi(\beta)]\!](\alpha)(p_0),\ p_0>\}))\\
&= \mathcal{D}_E'[\![\beta_1!m(\beta_2)]\!](\alpha)(\lambda\beta\cdot\mathcal{D}_S'[\![\psi(\beta)]\!](\alpha)(p_0))\\
&= \mathcal{D}_S'[\![(\beta_1!m(\beta_2),\ \psi)]\!](\alpha)(p_0)\\
&= \mathcal{D}_U^*(\{(\alpha,(\beta_1!m(\beta_2),\ \psi))\}).
\end{aligned}$$

THEOREM 7.7: $\Phi_U'(\mathcal{D}_U^*)=\mathcal{D}_U^*$

PROOF
We show: $\forall X\in\mathcal{P}_{fin}(LStat)\ [\Phi_U'(\mathcal{D}_U^*)(X)=\mathcal{D}_U^*(X)]$, using induction on the number of elements in X.

Case 1: $X=\{(\alpha,s)\}$, with $\alpha\in AObj$, $s\in L_S'$.
The proof uses induction on the complexity of the statement s. We treat some typical cases.

(i) **answer** m:

$$\Phi_U'(\mathcal{D}_U^*)(\{(\alpha,\ \textbf{answer}\ m)\})= \lambda\sigma\cdot\{<\alpha,m,g_m>\}$$

with

$$\begin{aligned}
g_m &= \lambda\beta\cdot\lambda f\cdot\lambda\bar{\sigma}\cdot\{<\bar{\sigma}',\mathcal{D}_E[\![e_m]\!](\alpha)(\lambda\beta'\cdot\lambda\hat{\sigma}\cdot\{<\hat{\sigma}',f(\beta')\|\mathcal{D}_U^*(\{(\alpha,E)\})>\})>\}\\
&= \lambda\beta\cdot\lambda f\cdot\lambda\bar{\sigma}\cdot\{<\bar{\sigma}',\mathcal{D}_E[\![e_m]\!](\alpha)(\lambda\beta'\cdot\lambda\hat{\sigma}\cdot\{<\hat{\sigma}',f(\beta')>\})>\}
\end{aligned}$$

(and $\bar{\sigma}'$ and $\hat{\sigma}'$ as in definition 6.1). If we compare this to the definition of $\mathcal{D}_S[\![\textbf{answer}\ m]\!]$ (definition 5.3(S3)) we see

$$\begin{aligned}
\lambda\sigma\cdot\{<\alpha,m,g_m>\} &= \mathcal{D}_S[\![\textbf{answer}\ m]\!](\alpha)(p_0)\\
&= \mathcal{D}_U^*(\{(\alpha,\ \textbf{answer}\ m)\}).
\end{aligned}$$

(ii) $x\leftarrow e$: we distinguish two subcases. First , if $e=\beta$, then

$$\begin{aligned}
\Phi_U'(\mathcal{D}_U^*)(\{(\alpha,x\leftarrow\beta)\}) &= \lambda\sigma\cdot\{<\sigma\{\beta/\alpha,x\},p_0>\}\\
&= \mathcal{D}_E'[\![\beta]\!](\alpha)(\lambda\beta\cdot\lambda\sigma\cdot\{<\sigma\{\beta/\alpha,x\},\ p_0>\})\\
&= \mathcal{D}_S'[\![x\leftarrow\beta]\!](\alpha)(p_0)\\
&= \mathcal{D}_U^*(\{(\alpha,x\leftarrow\beta)\}).
\end{aligned}$$

If $e\notin Obj$, then:

$$\begin{aligned}
\Phi_U'(\mathcal{D}_U^*)(\{(\alpha,x\leftarrow e)\}) &= [\ \text{definition}\ -U\rightarrow\ ]\\
&\quad \Phi_U'(\mathcal{D}_U^*)(\{(\alpha,(e,\lambda u\cdot x\leftarrow u))\})\\
&= [\ \text{see (v) below}\ ]\\
&\quad \mathcal{D}_U^*(\{(\alpha,(e,\lambda u\cdot x\leftarrow u))\})\\
&= \mathcal{D}_E'[\![e]\!](\alpha)(\lambda\beta\cdot\mathcal{D}_S'[\![x\leftarrow\beta]\!](\alpha)(p_0))\\
&= \mathcal{D}_S'[\![x\leftarrow e]\!](\alpha)(p_0)\\
&= \mathcal{D}_U^*(\{(\alpha,x\leftarrow e)\}).
\end{aligned}$$

(iii) $s_1;s_2$: case analysis for s_1.
(iv) **do** e **then** s **od**:

$$\begin{aligned}
&\Phi_U'(\mathcal{D}_U^*)(\{(\alpha,\ \textbf{do}\ e\ \textbf{then}\ s\ \textbf{od}\ )\})\\
&= \lambda\sigma\cdot\{<\sigma,\mathcal{D}_U^*(\{(\alpha,\ \textbf{if}\ e\ \textbf{then}\ s;(\textbf{do}\ e\ \textbf{then}\ s\ \textbf{od})\ \textbf{else}\ E\ \textbf{fi}\ )\})>\}\\
&= \lambda\sigma\cdot\{<\sigma,\mathcal{D}_E'[\![e]\!](\alpha)(\lambda\beta\cdot\ \text{if}\ \beta=tt\ \text{then}\\
&\quad\ \mathcal{D}_S'[\![s]\!](\alpha)(\mathcal{D}_S'[\![\textbf{do}\ e\ \textbf{then}\ s\ \textbf{od}]\!](\alpha)(p_0))\ \text{else}\ p_0\ \text{fi}\ )>\}\\
&= \mathcal{D}_S'[\![\textbf{do}\ e\ \textbf{then}\ s\ \textbf{od}]\!](\alpha)(p_0)\\
&= \mathcal{D}_U^*(\{(\alpha,\ \textbf{do}\ e\ \textbf{then}\ s\ \textbf{od}\ )\}).
\end{aligned}$$

(v) (e,ψ): by induction we have that the theorem holds for $(\alpha,\psi(\beta))$, for every $\beta\in Obj$. Now we can apply lemma 7.6.

Case 2: $X\in\mathcal{P}_{fin}$ $(LStat)$ and X has at least two elements.
Suppose we have two disjoint sets X_1 and X_2 in $\mathcal{P}_{fin}$ $(LStat)$ with $X=X_1\cup X_2$ such that

$$\Phi_U'(\mathcal{D}_U^*)(X_i)=\mathcal{D}_U^*(X_i)$$

for $i=1,2$. Assume $X_1,\ X_2\neq\{<\alpha_1,E>,\ \ldots,\ <\alpha_n,E>\}$. We shall show that from this induction hypothesis it follows that

$$\Phi_U'(\mathcal{D}_U^*)(X_1\cup X_2)=\ \mathcal{D}_U^*(X_1\cup X_2).$$

(This is proved in very much the same way as the fact that $\Phi'(\tilde{\mathcal{D}})(\rho)=\tilde{\mathcal{D}}(\rho)$ and $\Phi'(\tilde{\mathcal{D}})(\pi)=\tilde{\mathcal{D}}(\pi)$ implies $\Phi'(\tilde{\mathcal{D}})(\rho^\wedge\pi)=\tilde{\mathcal{D}}(\rho^\wedge\pi)$, which occurs in theorem 2.14 of section 2.)

From the definition of $-U\rightarrow$ (definition 4.8, rules 10 and 11) it follows that

$$\Phi'_U(\mathcal{D}_U^*)(X_1\cup X_2)=\ \lambda\sigma\cdot(X_1^q\cup X_2^q\cup Z).$$

Here

$$\begin{aligned}
X_1^q\ =\ &\{<\sigma',\ \mathcal{D}_U^*(X'_1\cup X_2)>:\ <X_1,\ \sigma>\ -U,\ \tau\rightarrow\ <X'_1,\ \sigma'>\}\\
&\cup\{<\beta_1,\ m,\ \beta_2,\ \lambda\beta\cdot\mathcal{D}_U^*(\{(\alpha,\ \psi(\beta))\}),\ \mathcal{D}_U^*(X_1'\cup X_2)>:\\
&\qquad <X_1,\sigma>\ -U,\ (\alpha,\beta_1!m(\beta_2))\rightarrow\ <X_1'\cup\{(\alpha,\psi)\},\sigma>\}\\
&\cup\{<\alpha,m,g_m>:\ <X_1,\sigma>\ -U,\ (\alpha?m)\rightarrow\ <X_1'\cup\{(\alpha,s)\},\ \sigma>\}
\end{aligned}$$

with

$$\begin{aligned}
g_m\ =\ &\lambda\beta\cdot\lambda f\cdot(\lambda\sigma\cdot\{<\overline{\sigma}',\ \mathcal{D}_E[\![e_m]\!](\alpha)(\lambda\beta'\cdot\lambda\hat{\sigma}\cdot\{<\hat{\sigma}',\ f(\beta')\|\mathcal{D}_U^*(\{(\alpha,s)\})>\})>\}\\
&\|\ \mathcal{D}_U^*(X_1'\cup X_2))
\end{aligned}$$

and e_m, $\overline{\sigma}'$ and $\hat{\sigma}'$ as in definition 6.1. The set X_2^q is like X_1^q but with the roles of X_1 and X_2 interchanged. Finally,

$$\begin{aligned}
Z\ =\ &\{<\sigma',\mathcal{D}_U^*(\{(\beta_1,\ (e_m,\ \lambda u\cdot(u_m\leftarrow\sigma_2(\beta_1)(u_m);\ \textbf{release}\ (\alpha,\psi(u));\ s)))\}\bigcup X'_1\cup X_2')>:\\
&<X_i,\sigma>\ -U,\ (\alpha,\beta_1!m(\beta_2))\rightarrow\ <\{(\alpha,\psi)\}\cup X_i',\ \sigma>\ \text{and}\\
&<X_j,\sigma>\ -U,\ (\beta_1?m)\rightarrow\ <\{(\beta_1,s)\}\cup X_j',\ \sigma>,\ \ \text{for}\ i=1,\ j=2\ \text{or}\ i=2,\ j=1\}
\end{aligned}$$

(and $\sigma'=\sigma\{\beta_2/\beta_1,\ u_m\}$, $m\Leftarrow<u_m,\ e_m>\in U$). The steps in X_i^q correspond to the transition steps that can be made from $X_1\cup X_2$ as a result of a transition step from X_i (by an application of rule 10 in the definition of $-U\rightarrow$), for $i=1,2$.

The set Z contains those steps that correspond with a communication transition from $X_1\cup X_2$

which results from a send transition from X_i and an answer transition from X_j (for $i=1$, $j=2$ or $i=2$, $j=1$) by an application of rule 11.

Now we have

$$X_1^q = \Phi_U'(\mathcal{D}_U^*)(X_1)(\sigma) \mathbin{\mathbb{L}} \mathcal{D}_U^*(X_2),$$

$$X_2^q = \Phi_U'(\mathcal{D}_U^*)(X_2)(\sigma) \mathbin{\mathbb{L}} \mathcal{D}_U^*(X_1),$$

$$Z = \Phi_U'(\mathcal{D}_U^*)(X_1)(\sigma) \mid_\sigma \Phi_U'(\mathcal{D}_U^*)(X_2)(\sigma).$$

The proofs of these facts are not difficult (but tiresome and therefore omitted). It follows that

$$\begin{aligned}
\Phi_U'(\mathcal{D}_U^*)(X_1 \cup X_2) &= \lambda\sigma\cdot(X_1^q \cup X_2^q \cup Z) \\
&= \lambda\sigma\cdot(\Phi_U'(\mathcal{D}_U^*)(X_1)(\sigma) \mathbin{\mathbb{L}} \mathcal{D}_U^*(X_2) \cup \\
&\qquad \Phi_U'(\mathcal{D}_U^*)(X_2)(\sigma) \mathbin{\mathbb{L}} \mathcal{D}_U^*(X_1) \cup \\
&\qquad \Phi_U'(\mathcal{D}_U^*)(X_2)(\sigma) \mid_\sigma \Phi_U'(\mathcal{D}_U^*)(X_2)(\sigma)) \\
&= [\text{ induction hypothesis }] \\
&\quad \lambda\sigma\cdot(\mathcal{D}_U^*(X_1)(\sigma) \mathbin{\mathbb{L}} \mathcal{D}_U^*(X_2) \cup \\
&\qquad \mathcal{D}_U^*(X_2)(\sigma) \mathbin{\mathbb{L}} \mathcal{D}_U^*(X_1) \cup \\
&\qquad \mathcal{D}_U^*(X_1)(\sigma) \mid_\sigma \mathcal{D}_U^*(X_2)(\sigma)) \\
&= [\text{ definition } \| \,] \\
&\quad \mathcal{D}_U^*(X_1) \| \mathcal{D}_U^*(X_2) \\
&= \mathcal{D}_U^*(X_1 \cup X_2).
\end{aligned}$$

This concludes the proof of theorem 7.7. □

Since Θ_U' and $\mathcal{D}_U^*$ are both fixed points of the same contraction Φ_U', they must be equal:

COROLLARY 7.8: $\Theta_U' = \mathcal{D}_U^*$

Part 3. Collecting the results

We have proved that $\Theta_U = \mathit{abstr} \circ \Theta_U'$ and that $\Theta_U' = \mathcal{D}_U^*$. Thus

THEOREM 7.9: $\Theta_U = \mathit{abstr} \circ \mathcal{D}_U^*$

From this theorem we deduce the main theorem of this paper:

THEOREM 7.10: $[\![U]\!]_\Theta = \mathit{abstr}([\![U]\!]_\mathcal{D})$

PROOF

Let $U = <(\ldots, C_n \Leftarrow s_n), \ldots>$. Then

$$\begin{aligned}
[\![U]\!]_\Theta &= \Theta_U[\![\{(\nu(\varnothing), s_n)\}]\!] \\
&= \mathit{abstr}(\mathcal{D}_U^*(\{(\nu(\varnothing), s_n)\})) \\
&= \mathit{abstr}(\mathcal{D}_S'[\![s_n]\!](\nu(\varnothing))(p_0)) \\
&= \mathit{abstr}(\mathcal{D}_S[\![s_n]\!](\nu(\varnothing))(p_0)) \\
&= \mathit{abstr}([\![U]\!]_\mathcal{D}).
\end{aligned}$$

8. REFERENCES

[Am85] P. AMERICA, *Definition of the programming language* POOL-T, ESPRIT project 415, Doc. No. 0091, Philips Research Laboratories, Eindhoven, 1985.

[Am86] P. AMERICA, *Rationale for the design of* POOL, ESPRIT project 415, Doc. No. 0053, Philips Research Laboratories, Eindhoven, 1986.

[Am87] P. AMERICA, POOL-T — *A parallel object-oriented language,* in: "Object-Oriented Concurrent Systems" (A. Yonezawa and M. Tokoro, Eds.), MIT Press, 1987.

[AB88] P. AMERICA, J.W. DE BAKKER, *Designing equivalent semantic models for process creation,* in: Theoretical Computer Science 60, 1988, pp. 109-176.

[ABKR86(a)] P. AMERICA, J.W. DE BAKKER, J.N. KOK, J.J.M.M. RUTTEN, *Operational semantics of a parallel object-oriented language,* in: "Conference Record of the 13th Symposium on Principles of Programming Languages, St. Petersburg, Florida," 1986, pp. 194-208.

[ABKR86(b)] P. AMERICA, J.W. DE BAKKER, J.N. KOK, J.J.M.M. RUTTEN, *A denotational semantics of a parallel object-oriented language,* Technical Report (CS-R8626), Centre for Mathematics and Computer Science, Amsterdam, 1986. (To appear in: Information and Computation.)

[ANSI83] ANSI, *Reference manual for the Ada programming language,* ANSI / MIL-STD 1815 A, United States Department of Defense, Washington D. C., 1983.

[AR88] P. AMERICA, J.J.M.M. RUTTEN, *Solving reflexive domain equations in a category of complete metric spaces,* in: Proceedings 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 298, Springer-Verlag, 1988, pp. 254-288. (To appear in the Journal of Computer and System Sciences.)

[Br86] A. DE BRUIN, *Experiments with continuation semantics: Jumps, backtracking, dynamic networks,* Ph. D. thesis, Free University of Amsterdam, 1986.

[BBKM84] J.W. DE BAKKER, J.A. BERGSTRA, J.W. KLOP, J.-J.CH. MEYER, *Linear time and branching time semantics for recursion with merge,* Theoretical Computer Science 34, 1984, pp. 135-156.

[BKMOZ86] 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 (J.W. de Bakker, W.P. de Roever, G. Rozenberg, Eds.), Lecture Notes in Computer Science 224, Springer-Verlag, 1986, pp. 51-121.

[BZ82] J.W. DE BAKKER, J.I. ZUCKER, *Processes and the denotational semantics of concurrency,* Information and Control 54, 1982, pp. 70-120.

[Cl81] W.D. CLINGER, *Foundations of actor semantics,* Ph. D. thesis, Massachusetts Institute of Technology (AI-TR-633), 1981.

[Du66] J. DUGUNDJI, *Topology,* Allyn and Bacon, inc., Boston, 1966.

[En77] E. ENGELKING, *General topology,* Polish Scientific Publishers, 1977.

[Go79] M.J.C. GORDON, *The denotational description of programming languages,* Springer-Verlag, 1979.

[HP79] M. HENNESSY, G.D. PLOTKIN, *Full abstraction for a simple parallel programming language,* in: Proceedings 8th MFCS (J. Bečvář ed.), Lecture Notes in Computer Science 74, Springer-Verlag, 1979, pp. 108-120.

[KR88] J.N. KOK, J.J.M.M. RUTTEN, *Contractions in comparing concurrency semantics,* in: Proceedings 15th ICALP, Tampere, Lecture Notes in Computer Science 317, Springer-Verlag, 1988, pp. 317-332.

[Mi51] E. MICHAEL, *Topologies on spaces of subsets,* Trans. AMS 71, 1951, pp.152-182.

[Mil80] R. MILNER, *A calculus of communicating systems,* Lecture Notes in Computer Science 92, Springer-Verlag, 1980.

[Od87] E.A.M. ODIJK, *The* DOOM *system and its applications: a survey of ESPRIT 415 subproject A,* in: "Parallel Architectures and Languages Europe, Volume I" (J.W. de

Bakker, A.J. Nijman, and P.C. Treleaven, Eds.), Lecture Notes in Computer Science 258, Springer-Verlag, 1987, pp. 461-479.

[Pl76] G.D. PLOTKIN, *A powerdomain construction,* SIAM Journal of Computing, Vol. 5, no. 3, 1976, pp. 452-487.

[Pl81] G.D. PLOTKIN, *A structural approach to operational semantics,* Report DAIMI FN-19, Comp. Sci. Dept., Aarhus Univ. 1981.

[Pl83] G.D. PLOTKIN, *An operational semantics for CSP,* in: Formal Description of Programming Concepts II (D. Bjørner ed.), North-Holland, Amsterdam, 1983, pp. 199-223.

APPENDIX I: MATHEMATICAL DEFINITIONS

DEFINITION A.1 (Metric space)
A *metric space* is a pair (M,d) with M a non-empty set and d a mapping $d:M\times M\to[0,1]$ (a *metric* or *distance*) that satisfies the following properties:
(a) $\forall x,y\in M\,[d(x,y)=0 \Leftrightarrow x=y]$
(b) $\forall x,y\in M\,[d(x,y)=d(y,x)]$
(c) $\forall x,y,z\in M\,[d(x,y)\leqslant d(x,z)+d(z,y)]$.
We call (M,d) an *ultra-metric space* if the following stronger version of property (c) is satisfied:
(c') $\forall x,y,z\in M\,[d(x,y)\leqslant \max\{d(x,z),d(z,y)\}]$.
Please note that we consider only metric spaces with bounded diameter: the distance between two points never exceeds 1.

EXAMPLES A.1.1
(a) Let A be an arbitrary set. The *discrete* metric d_A on A is defined as follows. Let $x,y\in A$, then

$$d_A(x,y)=\begin{cases}0 & \text{if } x=y\\ 1 & \text{if } x\neq y.\end{cases}$$

(b) Let A be an alphabet, and let $A^\infty=A^*\cup A^\omega$ denote the set of all finite and infinite words over A. Let, for $x\in A^\infty$, $x[n]$ denote the prefix of x of length n, in case $length(x)\geqslant n$, and x otherwise. We put

$$d(x,y)=2^{-sup\{n:x[n]=y[n]\}},$$

with the convention that $2^{-\infty}=0$. Then (A^∞,d) is a metric space.

DEFINITION A.2
Let (M,d) be a metric space, let $(x_i)_i$ be a sequence in M.
(a) We say that $(x_i)_i$ is a *Cauchy sequence* whenever we have:
$\forall\epsilon>0\ \exists N\in\mathbb{N}\ \forall n,m>N\ [d(x_n,x_m)<\epsilon]$.
(b) Let $x\in M$. We say that $(x_i)_i$ *converges to* x and call x the *limit* of $(x_i)_i$ whenever we have:
$\forall\epsilon>0\ \exists N\in\mathbb{N}\ \forall n>N\ [d(x,x_n)<\epsilon]$.
Such a sequence we call *convergent.* Notation: $\lim_{i\to\infty}x_i=x$.
(c) The metric space (M,d) is called *complete* whenever each Cauchy sequence converges to an element of M.

DEFINITION A.3
Let $(M_1,d_1),(M_2,d_2)$ be metric spaces.
(a) We say that (M_1,d_1) and (M_2,d_2) are *isometric* if there exists a bijection $f:M_1\to M_2$ such that:
$\forall x,y\in M_1\ [d_2(f(x),f(y))=d_1(x,y)]$. We then write $M_1\cong M_2$. When f is not a bijection (but only an injection), we call it an *isometric embedding.*

(b) Let $f:M_1 \to M_2$ be a function. We call f *continuous* whenever for each sequence $(x_i)_i$ with limit x in M_1 we have that $\lim_{i\to\infty} f(x_i)=f(x)$.

(c) Let $A \geqslant 0$. With $M_1 \to^A M_2$ we denote the set of functions f from M_1 to M_2 that satisfy the following property:
$\forall x,y \in M_1\ [d_2(f(x),f(y)) \leqslant A \cdot d_1(x,y)]$.
Functions f in $M_1 \to^1 M_2$ we call *non-expansive,* functions f in $M_1 \to^\epsilon M_2$ with $0 \leqslant \epsilon < 1$ we call *contracting*.
(For every $A \geqslant 0$ and $f \in M_1 \to^A M_2$ we have: f is continuous.)

PROPOSITION A.4 (Banach's fixed-point theorem)
Let (M,d) be a complete metric space and $f:M \to M$ a contracting function. Then there exists an $x \in M$ such that the following holds:
(1) $f(x)=x$ *(x is a fixed point of f),*
(2) $\forall y \in M\ [f(y)=y \Rightarrow y=x]$ *(x is unique),*
(3) $\forall x_0 \in M\ [\lim_{n\to\infty} f^{(n)}(x_0)=x]$, *where* $f^{(n+1)}(x_0)=f(f^{(n)}(x_0))$ and $f^{(0)}(x_0)=x_0$.

DEFINITION A.5 (Closed and compact subsets)
A subset X of a complete metric space (M,d) is called *closed* whenever each Cauchy sequence in X has a limit in X and is called *compact* whenever each sequence in X has a subsequence that converges to an element of X.

DEFINITION A.6
Let $(M,d),(M_1,d_1),\ldots,(M_n,d_n)$ be metric spaces.

(a) With $M_1 \to M_2$ we denote the set of all continuous functions from M_1 to M_2. We define a metric d_F on $M_1 \to M_2$ as follows. For every $f_1,f_2 \in M_1 \to M_2$

$$d_F(f_1,f_2)=\sup_{x\in M_1}\{d_2(f_1(x),f_2(x))\}.$$

For $A \geqslant 0$ the set $M_1 \to^A M_2$ is a subset of $M_1 \to M_2$, and a metric on $M_1 \to^A M_2$ can be obtained by taking the restriction of the corresponding d_F.

(b) With $M_1 \cup \cdots \cup M_n$ we denote the *disjoint union* of $M_1,\ldots,M_n$, which can be defined as $\{1\}\times M_1 \cup \cdots \cup \{n\}\times M_n$. We define a metric d_U on $M_1 \cup \cdots \cup M_n$ as follows. For every $x,y \in M_1 \cup \cdots \cup M_n$

$$d_U(x,y) = \begin{cases} d_j(x,y) & \text{if } x,y \in \{j\}\times M_j,\ 1 \leqslant j \leqslant n \\ 1 & \text{otherwise.} \end{cases}$$

(c) We define a metric d_P on $M_1 \times \cdots \times M_n$ by the following clause.
For every $(x_1,\ldots,x_n),(y_1,\ldots,y_n) \in M_1 \times \cdots \times M_n$

$$d_P((x_1,\ldots,x_n),(y_1,\ldots,y_n))=\max_i\{d_i(x_i,y_i)\}.$$

(d) Let $\mathcal{P}_{closed}(M)=\{X: X \subseteq M \wedge X \text{ is closed}\}$. We define a metric d_H on $\mathcal{P}_{closed}(M)$, called the *Hausdorff distance,* as follows. For every $X,Y \in \mathcal{P}_{closed}(M)$ with $X,Y \neq \emptyset$

$$d_H(X,Y)=\max\{\sup_{x\in X}\{d(x,Y)\},\sup_{y\in Y}\{d(y,X)\}\},$$

where $d(x,Z)=^{def}\inf_{z\in Z}\{d(x,z)\}$ for every $Z \subseteq M$, $x \in M$. For $X \neq \emptyset$ we put

$$d_H(\emptyset,X)=d_H(X,\emptyset)=1.$$

The following spaces

$$\mathcal{P}_{compact}(M) = \{X: X \subseteq M \wedge X \text{ is compact}\}$$

$$\mathcal{P}_{ncompact}(M) = \{X: X \subseteq M \wedge X \text{ is nonempty and compact}\}$$

are supplied with a metric by taking the respective restrictions of d_H.

(e) Let $c \in [0,1]$. We define: $id_c(M,d) = (M, c \cdot d)$.

PROPOSITION A.7

Let (M,d), $(M_1,d_1), \ldots, (M_n,d_n)$, d_F, d_U, d_P and d_H be as in definition A.6 and suppose that (M,d), $(M_1,d_1), \ldots, (M_n,d_n)$ are complete. We have that

(a) $(M_1 \underline{\to} M_2, d_F)$, $(M_1 \to^A M_2, d_F)$,

(b) $(M_1 \cup \cdots \cup M_n, d_U)$,

(c) $(M_1 \times \cdots \times M_n, d_P)$,

(d) $(\mathcal{P}_{closed}(M), d_H)$, $(\mathcal{P}_{compact}(M), d_H)$ and $(\mathcal{P}_{ncompact}(M), d_H)$

are complete metric spaces. If (M,d) and (M_i,d_i) are all ultra-metric spaces these composed spaces are again ultra-metric. (Strictly spoken, for the completeness of $M_1 \to M_2$ and $M_1 \to^A M_2$ we do not need the completeness of M_1. The same holds for the ultra-metric property.)

The proofs of proposition A.7 (a), (b) and (c) are straightforward. Part (d) is more involved. It can be proved with the help of the following characterization of the completeness of the Hausdorff metric.

PROPOSITION A.8

Let $(\mathcal{P}_{closed}(M), d_H)$ be as in definition A.6. Let $(X_i)_i$ be a Cauchy sequence in $\mathcal{P}_{closed}(M)$. We have:

$$\lim_{i\to\infty} X_i = \{\lim_{i\to\infty} x_i \mid x_i \in X_i, (x_i)_i \text{ a Cauchy sequence in } M\}.$$

The proof of proposition A.8 can be found in [Du66] and [En77]. The completeness of the Hausdorff space containing compact sets is proved in [Mi51].

APPENDIX II: THE FUNCTION *abstr*

The definition of $abstr: \overline{P} \to P$ can be viewed as a fixed point characterization of a somewhat differently and more intuitively defined operation

$$abstr^*: \overline{P} \to P,$$

which we introduce below. Next, we show that $abstr = abstr^*$.

DEFINITION II.1 ($abstr^*$)

Let $p \in \overline{P}$ and $\sigma \in \Sigma$, and let $w \in \Sigma_\partial^\infty$.

(1) We call w a *finite stream* in $p(\sigma)$ if there exist $<\sigma_1,p_1>, \ldots, <\sigma_n,p_n>$ such that

$$w = \sigma_1 \cdots \sigma_n \wedge \forall 1 \leqslant i < n\ [<\sigma_{i+1},p_{i+1}> \in p_i(\sigma_i)] \wedge <\sigma_1,p_1> \in p(\sigma) \wedge p_n = p_0.$$

(2) We call w an *infinite stream* in $p(\sigma)$ if there exist $<\sigma_1,p_1>, <\sigma_2,p_2>, \ldots$ such that

$$w = \sigma_1\sigma_2 \cdots \wedge \forall 1 \leqslant i\ [<\sigma_{i+1},p_{i+1}> \in p_i(\sigma)] \wedge <\sigma_1,p_1> \in p(\sigma).$$

(3) We call w a *deadlocking stream* in $p(\sigma)$ if there exist $<\sigma_1,p_1>, \ldots, <\sigma_n,p_n>$ such that

$$w = \sigma_1 \cdots \sigma_n \cdot \partial \wedge \forall 1 \leqslant i < n\ [<\sigma_{i+1},p_{i+1}> \in p_i(\sigma_i)] \wedge$$

$$<\sigma_1,p_1> \in p(\sigma) \wedge p_n \neq p_0 \wedge p_n(\sigma_n) \cap (\Sigma \times \overline{P}) = \varnothing.$$

Now we define a function $abstr^*: \overline{P} \to P$ by

$$abstr^*(p) = \lambda\sigma \cdot \{w: w \text{ is a stream in } p(\sigma)\}.$$

We have to verify that for every $p \in \overline{P}$ and $\sigma \in \Sigma$ the set $abstr^*(p)(\sigma)$ is compact. This is not trivial and is proved in theorem II.3 below (which is a slightly generalised form of lemma AII.4 in [BBKM84]). The fact that we use in the definition of $\overline{P}$ *compact* subsets rather than closed ones is essential for the proof. (For a process domain defined with *closed* subsets, [BBKM84] provides a counterexample of the theorem.)

In the proof of theorem II.3 below, we need the following lemma:

LEMMA II.2
Let $q = \lim_{n\to\infty} q_n$, *for* $q, q_n \in \overline{P}$: *assume (without loss of generality) that for all* $n \geqslant 0$

$$d(q,q_n) \leqslant 2^{-(n+1)}.$$

Let $\sigma \in \Sigma$ *and let* $(w_i)_i$ *be a sequence in* Σ_δ^∞ *with* $w_i \in abstr^*(q_i)(\sigma)$, *for every* $i \geqslant 0$. *Then*

$$\forall n\, \exists u\, [\, w_n[n] \cdot u \in abstr^*(q)(\sigma)\,].$$

PROOF
Let $w_n[n] = \sigma_1 \cdots \sigma_n$. (In the case of termination or deadlock the rest of the proof is analogous to this case.) Now there must be $q^1, \ldots, q^n$ with

$$<\sigma_1, q^1> \in q_n(\sigma) \text{ and } <\sigma_{i+1}, q^{i+1}> \in q^i(\sigma_i)$$

for $1 \leqslant i \leqslant n$. We shall show that there are $\overline{q}^1, \ldots, \overline{q}^n$ with $<\sigma_1, \overline{q}^1> \in q(\sigma)$ and $<\sigma_{i+1}, \overline{q}^{i+1}> \in \overline{q}^i(\sigma_i)$ for $1 \leqslant i \leqslant n$. We do this inductively: For $i = 1$ we observe that $d(q,q_n) \leqslant 2^{-(n+1)}$, so $d(q(\sigma), q_n(\sigma)) \leqslant 2^{-n} \leqslant \frac{1}{2}$. Because $<\sigma_1, q^1> \in q_n(\sigma)$, there must be a $\overline{q}^1$ with

$$<\sigma_1, \overline{q}^1> \in q(\sigma) \text{ and } d(q^1, \overline{q}^1) \leqslant 2^{-n}.$$

For the inductive step, let $1 \leqslant i \leqslant n$ and let $\overline{q}^i$ be such that $d(q^i, \overline{q}^i) \leqslant 2^{-(n+1)+i}$. Then

$$d(q^i(\sigma_i), \overline{q}^i(\sigma_i)) \leqslant 2^{-n+i} \leqslant \tfrac{1}{2}.$$

Because $<\sigma_{i+1}, q^{i+1}> \in q^i(\sigma_i)$ there must be a $\overline{q}^{i+1}$ with

$$<\sigma_{i+1}, \overline{q}^{i+1}> \in \overline{q}^i(\sigma_i) \text{ and } d(q^{i+1}, \overline{q}^{i+1}) \leqslant 2^{-n+i}.$$

With $\overline{q}^1, \ldots, \overline{q}^n$ suitably chosen, we can take $u \in abstr^*(\overline{q}^n)(\sigma_n)$ arbitrary, and then $w_n[n] \cdot u$ will be in $abstr^*(q)(\sigma)$.

THEOREM II.3: *For every* $p \in \overline{P}$ *and* $\sigma \in \Sigma$ *the set* $abstr^*(p)(\sigma)$ *is compact.*

PROOF
Let $(w_i)_i$ be a sequence in $abstr^*(p)(\sigma)$. We shall show that there exists a subsequence of $(w_i)_i$ that has its limit in $abstr^*(p)(\sigma)$. First we introduce some notation: For an arbitrary word $w \in \Sigma_\delta^\infty$, $w<k>$ indicates the word that is obtained from w by omitting the first k elements. We call $p_0 = p$, $\sigma_0 = \sigma$ and $f_0 = id_{\mathbb{N}}$, the identity function on the set of natural numbers. We shall inductively construct for every $n \geqslant 0$ a function $f_n : \mathbb{N} \to \mathbb{N}$, a process $p_n \in \overline{P}$, and a state σ_n such that:

1. $\forall i \geqslant 0\, [\, w_{f_n(i)}[n] = \sigma_1 \cdots \sigma_n\,]$
2. $\forall i,\ 0 \leqslant i < n\, [\, <\sigma_{i+1}, p_{i+1}> \in p_i(\sigma_i)\,]$
3. $\exists (v_i)_i$ in $abstr^*(p_n)(\sigma_n)\ \forall i \geqslant 1\, [\, v_i[i] = w_{f_n(i)}<n>[i]\,]$
4. f_n is monotonic and there exists a monotonic h with $f_n = f_{n-1} \circ h$.

Once we have constructed such sequences $(f_n)_n$, $(p_n)_n$ and $(\sigma_n)_n$, we are done: We can define

$$g(i) = f_i(i).$$

This function is monotonic and we have

$$\lim_{i\to\infty} w_{g(i)} = \sigma_1{\cdot}\sigma_2 \cdots$$

Since $\sigma_1{\cdot}\sigma_2 \cdots \in \mathit{abstr}^*(p)(\sigma)$ we thus have found a subsequence $(w_{g(i)})_i$ of $(w_i)_i$, which has its limit in $\mathit{abstr}^*(p)(\sigma)$.

The construction is as follows: Suppose we are at stage $n \geqslant 0$. Let $(v_i)_i$ be a sequence in $\mathit{abstr}^*(p_n)(\sigma_n)$ satisfying property 3. above. Let for every $i \geqslant 1$

$$v_i = \tau_1^i{\cdot}\tau_2^i \cdots.$$

Then there are $q_1^i, q_2^i, \ldots \in \overline{P}$ with

$$<\tau_1^i, q_1^i> \in p_n(\sigma_n), \text{ and}$$

$$\forall j \geqslant 1\ [<\tau_{j+1}^i, q_{j+1}^i> \in q_j^i(\tau_j^i)].$$

Since the set $p_n(\sigma_n)$ is compact, the sequence $(<\tau_1^i, q_1^i>)_i$ has a converging subsequence, which is given by, say, the monotonic function h and which has a limit, say $<\tau, q>$ in $p_n(\sigma_n)$. We may assume

$$\forall j \geqslant 1\ [\tau_1^{h(j)} = \tau \wedge d(q_1^{h(j)}, q) \leqslant 2^{-(j+1)}].$$

Now we take

$$p_{n+1} = q,\ \ \sigma_{n+1} = \tau,\ \ f_{n+1} = f_n \circ h.$$

In order to show that this construction works, we have to verify that p_{n+1}, σ_{n+1}, and f_{n+1} again satisfy properties 1. through 4. above.

1. We have for every $i \geqslant 1$:

$$\begin{aligned} w_{f_{n+1}(i)}[n+1] &= w_{f_{n+1}(i)}[n]{\cdot}w_{f_{n+1}(i)}(n+1) \\ &= \sigma_1 \cdots \sigma_n{\cdot}w_{f_{n+1}(i)}<n>(1) \\ &= \sigma_1 \cdots \sigma_n{\cdot}v_{h(i)}(1) \\ &= \sigma_1 \cdots \sigma_n{\cdot}\sigma_{n+1}. \end{aligned}$$

2. We have $<\sigma_{n+1}, p_{n+1}> = <\tau, q> \in p_n(\sigma_n)$.
3. In order to prove this property, we are going to apply the following version of lemma II.2: For all $q, q_1, q_2, \ldots \in \overline{P}$, and for all $x_1, x_2, \ldots \in \Sigma_\partial^\infty$,

$$\forall i \geqslant 1\ [d(q, q_i) \leqslant 2^{-(i+1)} \wedge x_i \in \mathit{abstr}^*(q_i)(\sigma)] \Rightarrow$$

$$\exists (y_i)_i \text{ in } \mathit{abstr}^*(q)(\sigma)\ \forall i \geqslant 1\ [y_i[i] = x_i[i]].$$

This we now use: Since

$$\forall i \geqslant 1\ [d(p_{n+1}, q_1^{h(i)}) \leqslant 2^{-(i+1)} \wedge v_{h(i)}<1> \in \mathit{abstr}^*(q_1^{h(i)})(\sigma_{n+1})]$$

there must exist a sequence $(v_i')_i$ in $\mathit{abstr}^*(p_{n+1})(\sigma_{n+1})$ with

$$\forall i \geqslant 1\ [v_i'[i] = v_{h(i)}<1>[i]].$$

Now

$$\begin{aligned} v_{h(i)}<1>[i] &= v_{h(i)}[h(i)]<1>[i] \\ &= w_{f_{n+1}(i)}<n>[h(i)]<1>[i] \\ &= w_{f_{n+1}(i)}<n><1>[i] \\ &= w_{f_{n+1}(i)}<n+1>[i]. \end{aligned}$$

(Here we have used twice the fact that $h(i)>i$, for all $i \geqslant 1$.)

4. By definition.

This concludes the proof of theorem II.3.

Next we show that the function $abstr: \bar{P} \to P$, given in definition 7.1, can be defined as the fixed point of a contraction.

DEFINITION II.4 (Formal definition *abstr*)

We define $\Xi: (\bar{P} \to^1 P) \to (\bar{P} \to^1 P)$; let $F \in \bar{P} \to^1 P$, $P \in \bar{P}$ and $\sigma \in \Sigma$. We put

$$\Xi(F)(p_0)(\sigma) = \{\epsilon\},$$

$$\Xi(F)(p)(\sigma) = \{\partial\}, \quad \text{if } p(\sigma) \cap Comp_{\bar{P}} = \emptyset.$$

Otherwise, we set

$$\Xi(F)(p)(\sigma) = \bigcup \{\sigma' \cdot F(p')(\sigma') : <\sigma', p'> \in p(\sigma)\}.$$

Finally, we define

$$abstr = \text{Fixed Point}(\Xi)$$

It is straightforward to show Ξ is contracting. The fact that for every $p \in \bar{P}$ and $\sigma \in \Sigma$ the set $\Xi(F)(p)(\sigma)$ is compact needs some explanation. In order to prove this, it is convenient to adapt the definition of Ξ a little. Recalling that $P = \Sigma \to \mathcal{P}_{ncomp}(\Sigma_\partial^\infty)$ we define

$$\Xi': ((\bar{P} \times \Sigma) \to^1 \mathcal{P}_{ncomp}(\Sigma_\partial^\infty)) \to ((\bar{P} \times \Sigma) \to^1 \mathcal{P}_{ncomp}(\Sigma_\partial^\infty)),$$

where the superscript 1 above the arrow indicates that we consider only non-expansive (and hence continuous) functions, by

$$\Xi'(F)(<p,\sigma>) = \bigcup \{\sigma' \cdot F(<p',\sigma'>) : <\sigma', p'> \in p(\sigma)\}.$$

Now

$$\begin{aligned} \Xi'(F)(<p,\sigma>) &= \bigcup_{<\sigma',p'> \in p(\sigma)} \{\sigma' \cdot F(<p',\sigma'>)\} \\ &= \bigcup_{\sigma'} (\sigma' \cdot \{F(<p',\sigma'>) : <\sigma', p'> \in p(\sigma)\}) \\ &= \bigcup_{\sigma'} (\sigma' \cdot F(\{<p',\sigma'> : <\sigma', p'> \in p(\sigma)\})) \end{aligned}$$

This union can be seen to be compact by first observing that from the compactness of p it follows that the union is finite: the set

$$\{\sigma' : \exists p' \in \bar{P}[<\sigma', p'> \in p(\sigma)]\}$$

is finite. The compactness of $p(\sigma)$ further implies the compactness of the isomorphic set

$$\{<p',\sigma'> : <\sigma', p'> \in p(\sigma)\},$$

for every $\sigma' \in \Sigma$, which is preserved under the continuous mapping F and the concatenation with σ'. So we have a finite union of compact sets, which is again compact. Now the compactness of $\Xi(F)(p)(\sigma)$ follows straightforwardly from the compactness of $\Xi'(F')(<p',\sigma'>)$, for arbitrary F', p' and σ'. The fact that $\Xi(F)$ is again non-expansive is also easily verified.

We conclude this appendix by showing that *abstr* and $abstr^*$ are equal:

THEOREM II.5: $abstr = abstr^*$

PROOF: Consider $p \in \overline{P} - \{p_0\}$ and $\sigma \in \Sigma$ such that $p(\sigma) \cap (\Sigma \times \overline{P}) \neq \varnothing$. Then:

$$
\begin{aligned}
w \in abstr^*(p)(\sigma) &\Leftrightarrow \text{[definition } abstr^*\text{]} \\
&\quad \exists \sigma' \in \Sigma \, \exists w' \in \Sigma_\partial^\infty \, \exists p' \in \overline{P} \; [w = \sigma' \cdot w' \wedge w' \in abstr^*(p')(\sigma')] \\
&\Leftrightarrow \text{[definition } \Xi\text{]} \\
&\quad w \in \Xi(abstr)(p)(\sigma).
\end{aligned}
$$

The other cases are easy. We see: $abstr^* = \Xi(abstr^*)$. Because Ξ is a contraction the theorem follows. (Note the similarity of this proof and the one of theorem 4.14.)

APPENDIX III: STANDARD OBJECTS

We want to extend the language under consideration with a few standard classes of so-called *standard* objects, namely the classes Boolean and Integer. On these objects the usual operations can be performed, but they must be formulated by sending messages. For example, the addition 23+11 is indicated by the send expression 23! add (11), sending a message with method name add and parameter 11 to the standard object 23. The set of expressions L_E, given in definition 3.1, is extended with these standard objects:

$$e ::= x \mid u \mid e_1!m(e_2) \mid \mathbf{new}(C) \mid s;e \mid \mathbf{self} \mid \alpha,$$

where $\alpha \in SObj$, with

$$SObj = \mathbb{Z} \cup \{tt, ff\}.$$

Recall that we already defined (in definition 4.1):

$$Obj = AObj \cup SObj$$
$$(= AObj \cup \mathbb{Z} \cup \{tt, ff\}).$$

Intuitively, the evaluation of the expression α, with $\alpha \in SObj$, results in that object itself. For instance, the value of the expression 29 will be the integer 29.

Below, we shall first extend the definition of the operational semantics, next we adapt the definition of the denotational semantics (following [ABKR86(b)]), and finally we shall prove that the equivalence result of section 7 still holds.

III.1 *Standard objects in the operational semantics*

We extend the set $L_{E'}$, given in definition 4.2, with the standard objects:

$$e ::= x \mid u \mid e_1!m(e_2) \mid \mathbf{new}(C) \mid s;e \mid \mathbf{self} \mid \alpha \mid (e, \phi),$$

where now $\alpha \in Obj = AObj \cup SObj$.

Next we add to the set of labeled statements (definition 4.5) an abstract element S_t that represents all standard objects and for which transitions will be specified in a moment:

$$LStat^* = LStat \cup \{S_t\}.$$

The following transitions are possible from S_t:

$$<\{S_t\}, \sigma> - n?\mathsf{add} \rightarrow <\{S_t\}, \sigma>$$
$$<\{S_t\}, \sigma> - n?\mathsf{sub} \rightarrow <\{S_t\}, \sigma>$$
$$<\{S_t\}, \sigma> - b?\mathsf{and} \rightarrow <\{S_t\}, \sigma>$$
$$<\{S_t\}, \sigma> - b?\mathsf{or} \rightarrow <\{S_t\}, \sigma>$$

$$<\{S_t\},\ \sigma> - b?\text{not} \rightarrow <\{S_t\},\ \sigma>$$

for every $n \in \mathbb{Z}$ and $b \in \{tt, ff\}$. (This list can be extended with transitions for other operations.) Communication with a standard object is now modeled by the following transitions:

$$\text{If } <\{(\alpha,s)\},\ \sigma> - (\alpha, n!\text{add}\ (m)) \rightarrow <\{(\alpha,\psi)\},\ \sigma>$$

$$\text{then } <\{(\alpha,s),\ S_t\},\ \sigma> - \gamma \rightarrow <\{(\alpha,\ \psi(n+m)),\ S_t\}.$$

$$\text{If } <\{(\alpha,s)\},\ \sigma> - (\alpha,\ b_1!\text{and}\ (b_2)) \rightarrow <\{(\alpha,\psi)\},\ \sigma>$$

$$\text{then } <\{(\alpha,s),\ S_t\},\ \sigma> - \gamma \rightarrow <\{(\alpha,\psi(b_1 \wedge b_2)),\ S_t\},\ \sigma>,$$

and by similar transitions for the other operations. The result of, for example, an addition of the integers n and m is computed and passed through to the parameterized statement of the object requesting the execution of the method add.

Finally, the operational semantics of a unit (definition 4.11) is changed by taking into account the standard objects; we put

$$[\![U]\!]_\theta = \mathcal{O}_U[\![\{(\nu(\varnothing), s_n),\ S_t\}]\!].$$

(In the operational semantics defined in [ABKR86(a)], the standard objects are treated somewhat differently. There no special rules are given for the communication with a standard object; instead, some axioms are added that replace in one step a send expression that addresses a standard object by the corresponding value of the result.)

III.2 *Standard objects in the denotational semantics*

The denotational meaning of a standard object $\alpha \in L_E$ is given by

$$\mathcal{D}_E[\![\alpha]\!](\beta)(f) = f(\alpha),$$

where $\beta \in AObj$, and $f \in Cont_E$.

We follow [ABKR86(b)] in introducing a process $p_{St} \in \overline{P}$ that represents the denotational meaning of the standard objects. For this we have to adapt our semantic process domain $\overline{P}$. In definition 5.1 the domain $\overline{P}$ is given by

$$\overline{P} \cong \{p_0\} \cup id_{1/2}(\Sigma \rightarrow \mathcal{P}_{compact}(Step_{\overline{P}})).$$

In order to let the standard process p_{St}, to be defined below, fit into our semantic domain nicely, we are forced to use closed subsets of steps rather than compact ones. Let us indicate the process domain given in definition 5.1 by $\overline{P}_{co}$. We introduce here $\overline{P}_{cl}$, which satisfies:

$$\overline{P}_{cl} \cong \{p_0\} \cup id_{1/2}(\Sigma \rightarrow \mathcal{P}_{closed}(Step_{\overline{P}_{cl}})).$$

We have, via an obvious embedding, that $\overline{P}_{co} \subseteq \overline{P}_{cl}$.

Next we introduce $p_{St} \in \overline{P}_{cl}$, which represents the meaning of all standard objects. It satisfies the following equation:

$$\begin{aligned} p_{St} = \lambda\sigma\cdot\ & (\{<n,\ \text{add},\ g_n^+>:\ n \in \mathbb{Z}\} \cup \\ & \{<n, \text{sub},\ g_n^->:\ n \in \mathbb{Z}\} \cup \\ & \{<b,\ \text{and},\ g_b^\wedge>:\ b \in \{tt, ff\}\} \cup \\ & \{<b,\ \text{or},\ g_b^\vee>:\ b \in \{tt, ff\}\} \cup \\ & \{<b,\ \text{not},\ g_{\bar{b}}>:\ b \in \{tt, ff\}\}), \end{aligned}$$

where

$$g_n^+ = \lambda\overline{\beta} \in Obj^* \cdot \lambda f \in Obj \rightarrow \overline{P} \cdot (\text{if } \overline{\beta} \in \mathbb{Z} \text{ then } f(n + \overline{\beta}) \| p_{St} \text{ else } p_{St} \text{ fi}),$$

$$g_n^- = \lambda\bar{\beta}\in Obj^*\cdot\lambda f\in Obj\to\bar{P}\cdot \text{ (if } \bar{\beta}\in\mathbb{Z} \text{ then } f(n-\bar{\beta})\|p_{St} \text{ else } p_{St} \text{ fi)},$$

$$g_b^\wedge = \lambda\bar{\beta}\in Obj^*\cdot\lambda f\in Obj\to\bar{P}\cdot \text{ (if } \bar{\beta}\in\{tt,ff\} \text{ then } f(b\wedge\bar{\beta})\|p_{St} \text{ else } p_{St} \text{ fi)}$$

$$g_b^\vee = \lambda\bar{\beta}\in Obj^*\cdot\lambda f\in Obj\to\bar{P}\cdot \text{ (if } \bar{\beta}\in\{tt,ff\} \text{ then } f(b\vee\bar{\beta})\|p_{St} \text{ else } p_{St} \text{ fi)}$$

$$g_b^\neg = \lambda\bar{\beta}\in Obj^*\cdot\lambda f\in Obj\to\bar{P}\cdot f(\neg b)\|p_{St}.$$

This definition is self-referential since p_{St} occurs at the righthand side of the definition. Formally, p_{St} can be given as the fixed point of a suitably defined contraction on $\bar{P}_{cl}$.

We observe that p_{St} is an infinitely branching process, which is an element of $\bar{P}_{cl}$ but not of $\bar{P}_{co}$. This explains the introduction of $\bar{P}_{cl}$.

The operational intuition behind the definition of p_{St} is the following: For every $n\in\mathbb{Z}$ the set $p_{St}(\sigma)$ contains, among others, two elements, namely $<n,\text{add},g_n^+>$ and $<n,\text{sub},g_n^->$. These steps indicate that the integer object n is willing to execute its methods add and sub. If, for example by evaluating n !add(n'), a certain active object sends a request to integer object n to execute the method add with parameter n', then g_n^+, supplied with n' and the continuation f of the active object, is executed. We have that $g_n^+(n')(f)$ is, by definition, the parallel composition of f supplied with the immediate result of the execution of the method add, namely $n+n'$, and the process p_{St}, which remains unaltered: $g_n^+(n')(f)=f(n+n')\|p_{St}$. (A similar explanation applies to the presence in $p_{St}(\sigma)$ of the triples representing the booleans.)

The standard objects are assumed to be present at the execution of every unit U. Therefore we adapt the denotational semantics of a unit (definition 5.4) as follows:

$$[\![U]\!]_{\mathfrak{D}} = \mathfrak{D}_S[\![s_n]\!](\nu(\varnothing))(p_0)\|p_{St}.$$

III.3 *Semantic equivalence*

Finally, we extend the arguments presented in section 7 in order to show that for the modified versions of $[\![U]\!]_\Theta$ and $[\![U]\!]_{\mathfrak{D}}$, as presented above, we still have:

$$[\![U]\!]_\Theta = abstr([\![U]\!]_{\mathfrak{D}}).$$

We begin by adapting the intermediate semantics Θ_U' (definition 6.1), which will now be of type

$$\Theta_U'\colon \mathcal{P}_{fin}(LStat^*)\to\bar{P}_{cl}.$$

We put:

$$\Theta_U'(\{S_t\}) = p_{St}$$

and for $X\subseteq LStat^*-\{S_t\}$ $(=LStat)$:

$$\Theta_U'(X\cup\{S_t\}) = \Theta_U'(X)\|\Theta_U'(\{S_t\}),$$

with $\Theta_U'(X)$ as defined according to definition 6.1.

Next we extend the definition of *abstr* to an operation:

$$abstr^*:\bar{P}_{cl}\to(\Sigma\to\mathcal{P}(\Sigma_\partial^\infty)),$$

where *abstr** is defined as in definition II.1. Please note, however, that for processes $p\in\bar{P}_{cl}$ it is in general *not* the case that *abstr*$^*(p)(\sigma)$ is a *closed* subset of Σ_∂^∞. Fortunately we can prove the following, which turns out to be all we need:

THEOREM III.1: *For every* $p\in\bar{P}_{co}$ *and* $\sigma\in\Sigma$: *abstr*$^*(p\|p_{St})(\sigma)$ *is compact.*

PROOF

The proof is analogous the one for theorem II.3, given the additional observation that for every $p\in\bar{P}_{co}$ the set

$$(p \| p_{St})(\sigma) \cap (\Sigma \times \overline{P}_{cl})$$

is compact, which we prove now.

According to the definition of $\|$ we have

$$(p \| p_{St})(\sigma) = p(\sigma) \| \_ p_{St} \cup p_{St}(\sigma) \| \_ p \cup p(\sigma) |_{\sigma} p_{St}(\sigma).$$

From the continuity of $\|$ and the compactness of $p(\sigma)$ it follows that

$$(p(\sigma) \| \_ p_{St}) \cap (\Sigma \times \overline{P}_{cl}) = \{ <\sigma', p' \| p_{St}> : <\sigma', p'> \in p(\sigma) \}$$

is compact. Secondly, the set

$$(p_{St}(\sigma) \| \_ p) \cap (\Sigma \times \overline{P}_{cl})$$

is empty. Finally, we show that

$$(p(\sigma) |_{\sigma} p_{St}(\sigma)) \cap (\Sigma \times \overline{P}_{cl})$$

is compact. Consider a sequence $(<\sigma, q_i>)_i$ in this intersection. We show that it has a converging subsequence $(<\sigma, q_{k(i)}>)_i$. According to the definition of $|_{\sigma}$ there exist sequences $(<\alpha_i, m_i, \beta_i, f_i, p_i>)_i$ in $p(\sigma)$ and $(<\alpha_i, m_i, g_i>)_i$ in $p_{St}(\sigma)$ such that

$$q_i = g_i(\beta_i)(f_i) \| p_i.$$

Because $p(\sigma)$ is compact there exists a monotonic function $k: \mathbb{N} \to \mathbb{N}$ such that

$$(<\alpha_{k(i)}, m_{k(i)}, \beta_{k(i)}, f_{k(i)}, p_{k(i)}>)_i$$

is convergent. From the definition of the metric on $\overline{P}_{cl}$ it follows that we may assume that there exist α, m and β such that for all i

$$\alpha_{k(i)} = \alpha, \ m_{k(i)} = m, \ \text{and} \ \beta_{k(i)} = \beta.$$

The definition of p_{St} implies that for every $<\alpha, m, g>$ in $p_{St}(\sigma)$ the function g is entirely determined by α and m. Thus

$$(<\alpha_{k(i)}, m_{k(i)}, g_{k(i)}>)_i = (<\alpha, m, g_{k(i)}>)_i = (<\alpha, m, g>)_i,$$

for some g. Suppose we have

$$f = \lim_{i \to \infty} f_{k(i)} \wedge p = \lim_{i \to \infty} p_{k(i)};$$

then $<\alpha, m, \beta, f, p> \in p(\sigma)$ and

$$\lim_{i \to \infty} <\sigma, q_i> = <\sigma, g(\beta)(f) \| p> \in (p(\sigma) |_{\sigma} p_{St}(\sigma)) \cap (\Sigma \times \overline{P}_{cl}). \qquad \square$$

COROLLARY III.2: $abstr^* \circ \Theta_U' \in \mathcal{P}_{fin}(LStat^*) \to P$

(Recall that $P = \Sigma \to \mathcal{P}_{ncompact}(\Sigma_\partial^\infty)$.)

THEOREM III.3: $\Theta_U = abstr^* \circ \Theta_U'$

This theorem can be proved by showing that in addition to Θ_U also $abstr^* \circ \Theta_U'$ is a fixed point of Φ_U. This can be done analogously to the proof of theorem 7.2. From this observation and the fact that Φ_U is a contraction the theorem follows.

The definition of $\mathcal{D}_U^*$, which is given in definition 7.5, is also changed. It will be a function of type

$$\mathcal{D}_U^*: \mathcal{P}_{fin}(LStat^*) \to \overline{P}_{cl}$$

that is like the original $\mathcal{D}_U^*$ but for the clause that

$$\mathfrak{D}^*_U(\{S_t\}) = p_{St}.$$

A last step towards the goal of this third appendix, which is to prove the semantic equivalence of the denotational and operational semantics with standard objects present, consists of the observation that theorem 7.6, stating that

$$\Phi_U'(\mathfrak{D}^*_U) = \mathfrak{D}^*_U,$$

can be proved for the new version of $\mathfrak{D}^*_U$ as well. The extended proof involves some new case analysis (within Case 2), concerning the communications with standard objects. This being the last appendix, this step being the last step towards our goal, and the author being only human, we omit the details and state without proof:

THEOREM III.4: *(Extended version of 7.6):* $\Phi_U'(\mathfrak{D}^*_U)=\mathfrak{D}^*_U$

COROLLARY III.5: *(Extended version of 7.7):* $\mathfrak{O}'=\mathfrak{D}^*_U$

Finally we are ready to prove the extended version of the main theorem (7.9) of our paper:

THEOREM III.6: $[\![U]\!]_{\mathfrak{O}} = abstr^*([\![U]\!]_{\mathfrak{D}})$

PROOF

$$\begin{aligned}
[\![U]\!]_{\mathfrak{O}} &= \mathfrak{O}_U[\![\{(\nu(\varnothing), s_n), S_t\}]\!] \\
&= \text{[theorem III.3]} \\
&\quad abstr^*(\mathfrak{O}_U'(\{(\nu(\varnothing), s_n), S_t\})) \\
&= \text{[corollary III.5]} \\
&\quad abstr^*(\mathfrak{D}^*_U(\{(\nu(\varnothing), s_n), S_t\})) \\
&= abstr^*(\mathfrak{D}_S[\![s_n]\!](\nu(\varnothing))(p_0)\|p_{St}) \\
&= abstr^*([\![U]\!]_{\mathfrak{D}}).
\end{aligned}$$

□