Theoretical Computer Science 60 (1988) 109-176
North-Holland

DESIGNING EQUIVALENT SEMANTIC MODELS FOR PROCESS CREATION*

Pierre AMERICA
Philips Research Laboratories, P.O. Box 80.000, 5600 JA Eindhoven, The Netherlands

Jaco DE BAKKER
Centre for Mathematics and Computer Science, P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

Communicated by M. Nivat
Received September 1987

Abstract. Operational and denotational semantic models are designed for languages with process creation, and the relationships between the two semantics are investigated. The presentation is organized in four sections dealing with a uniform and static, a uniform and dynamic, a nonuniform and static, and a nonuniform and dynamic language respectively. Here uniform/nonuniform refers to a language with uninterpreted/interpreted elementary actions, and static/dynamic to the distinction between languages with a fixed/growing number of parallel processes. The contrast between uniform and nonuniform is reflected in the use of linear time versus branching time models, the latter employing a version of Plotkin's resumptions. The operational semantics make use of Hennessy's and Plotkin's transition systems. All models are built on metric structures, and involve continuations in an essential way. The languages studied are abstractions of the parallel object-oriented language POOL for which we have designed separate operational and denotational semantics in earlier work. The paper provides a full analysis of the relationship between the two semantics for these abstractions. Technically, a key role is played by a new operator which is able to decide dynamically whether it should act as sequential or parallel composition.

Contents

* Most of this work has been carried out in the context of ESPRIT Project 415: *Parallel Architectures and Languages for Advanced Information Processing: A VLSI-directed Approach.*

1. Introduction

Process creation is an important programming concept which appears in a variety of forms in many contemporary programming styles. In imperative programming one finds it in languages such as Ada [1], NIL [43] and many others. In the context of functional or dataflow languages we refer to [22] for a semantic study dealing with process creation. For logic programming many recent references can be found in [42]. Object-oriented programming (see [5] for a general introduction from a theoretician's point of view) has the family of actor languages (see, e.g., [2, 23, 30]) as examples. The present study was inspired by the language POOL, an acronym for Parallel Object-Oriented Language, described in [3, 4].

In two previous investigations we have developed operational ($\mathcal{O}$) and denotational ($\mathcal{D}$) semantics for POOL [6, 7]. These two semantic models were designed independently of each other, and the investigation reported below constitutes the first step towards the goal of settling the relationship between the two models. For this purpose we concentrate on the programming notion of process creation together with a simple version of process communication, and leave a number of further key notions in POOL for later study. More specifically, we treat communication in the sense—approximately—as exemplified by CSP [31, 32] and do not treat message passing and method invocation—notions which should be situated at the same level as remote procedure call or Ada's rendez-vous. A similar combination of process creation with CSP-like communication was first described in [19], a paper which provides a proof-theoretic treatment of these concepts taken together.

Before going into the characteristics of the languages we shall deal with, let us say something about the terms "operational" and "denotational". *Operational* semantics gives a model of computation by constructing from a given program a kind of "abstract machine" having a set of "states" (which we shall call *configurations*), and describing the *transitions* this abstract machine can make from one state to another. *Denotational* semantics works by assigning a *meaning*, which is a mathematical entity, to each fragment of a program, in such a way that the meaning of a composite piece of program can be inferred by looking only at the meanings of its parts, not at their internal structure. We say that denotational semantics describes the meaning of programs in a *compositional* way. Fortunately, the technique we use for our operational semantics, transition systems in the style of Hennessy's and Plotkin's Structured Operational Semantics (SOS) [29, 38, 39] describes the abstract machine and its state transitions in a way that is directly related to the syntactic structure of the original program. Due to the explicit presence of this abstract machine, the transition systems employed have, we feel, a strong operational intuition.

The emphasis in our semantics design is very much on a systematic development of the tools for both the operational and denotational models. We have therefore structured the presentation in four sections, dealing with four languages of increasing complexity. Using some terminology which will be explained in a moment, we shall successively present operational and denotational semantics for

(1) a uniform and static language $\mathscr{L}_{\mathrm{us}}$;
(2) a uniform and dynamic language $\mathscr{L}_{\mathrm{ud}}$;
(3) a nonuniform and static language $\mathscr{L}_{\mathrm{nus}}$;
(4) a nonuniform and dynamic language $\mathscr{L}_{\mathrm{nud}}$.

These languages are conceptually ordered according to the following diagram:

$$\begin{tikzcd}
& \mathscr{L}_{\mathrm{nud}} & \\
\mathscr{L}_{\mathrm{ud}} \arrow[ur] & & \mathscr{L}_{\mathrm{nus}} \arrow[ul] \\
& \mathscr{L}_{\mathrm{us}} \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

In this classification, a *uniform* language is one which has uninterpreted elementary actions. In other words, the indivisible or atomic unit of such a language is just a symbol from some alphabet, and the meanings assigned to programs in a uniform language bear strong resemblance to formal languages (here with finite *and infinite* words). A nonuniform language has interpreted elementary actions, in our case assignments and communications. Thus, (individual) variables appear on the scene, and as a consequence we find in our semantics the notion of a *state*, i.e., of a mapping from variables to values. Programs now transform states, and we shall develop a mathematical structure with entities which combine the flavour of state-transforming functions with that of a record of the computational history. In Section 5, we shall provide evidence that the latter notion is necessary in view of the parallel execution operator.

The second distinction in the above diagram concerns that of static versus dynamic languages. In the former, we have a fixed number of parallel processes, in the latter a dynamically growing number of processes: each time a new process is created, the total number of active processes increases by one. (We shall not investigate in our paper any notion of process destruction, a concept not present in the language POOL.)

The simplest element in the partial order is $\mathscr{L}_{\text{us}}$, to be treated in Section 3. It is extended in two directions: one adds the notion of process creation ($\mathscr{L}_{\text{ud}}$), dealt with in Section 4, and the other adds the notion of interpreted elementary actions, described in Section 5. Finally, in Section 6, both extensions are brought together, and the full complexity of a nonuniform dynamic language is confronted.

In Sections 3 and 4, the languages are uniform and the semantic models are of the so-called "linear time" variety (see, e.g., [11] or [40]), i.e., they consist of sets of (finite or infinite) sequences over a certain alphabet. The operational semantics is a uniform version of the Structured Operational Semantics (SOS) of Hennessy and Plotkin [29, 38, 39]. The denotational semantics is built on *metric* foundations (apart from the above diagram, no partial order is employed in our paper); this remains true for later (nonuniform) sections. A *distance* between two sequences or sets of sequences is readily defined, and most of the tools of metric topology we use are quite standard. In particular, we shall make heavy use of Banach's fixed point theorem for contracting functions on a complete metric space. Accordingly, our (denotational) semantics will be defined, when dealing with recursive constructs, only when the recursion is *guarded.* In formal languages, one would say that the grammar concerned satisfies a Greibach condition. (In the nonuniform setting we shall take an approach where guardedness is automatically satisfied.)

In each of the Sections 3 to 6 we shall, after having presented the two semantic models, go on to investigate their *equivalence.* In Sections 3 and 4 we actually prove that the two semantics yield the same result, i.e., that for $t \in \mathscr{L}_{\text{us}}$ or $t \in \mathscr{L}_{\text{ud}}$ we have $\mathscr{O}[\![t]\!] = \mathscr{D}[\![t]\!]$. For $\mathscr{L}_{\text{us}}$, this is a result which was already obtained earlier (and presented in [16]). Below, we repeat certain parts of the proof as a first step towards the equivalence theorem for $\mathscr{L}_{\text{ud}}$, a result which we believe to be new. In the analysis of $\mathscr{L}_{\text{ud}}$ we make essential use of the notion of *continuation*, both of a syntactic and of a semantic kind. Since we develop the semantics of $\mathscr{L}_{\text{us}}$ as preparatory for $\mathscr{L}_{\text{ud}}$, we have adapted accordingly the treatment of [16], which does not employ continuations. The equivalence proofs for $\mathscr{L}_{\text{us}}$ and $\mathscr{L}_{\text{ud}}$ have strong similarities. On the other hand, there is also a fundamental difference having to do with the following consequence of process creation: in a statement with a syntactic sequential composition (";"), say $s_1;s_2$, we do not know whether to model the syntactic ";" by semantic concatenation ("$\cdot$") or by parallel execution ("$\|$"). To see this, contrast the statement $a;b$ yielding the singleton set $\{ab\}$ as its meaning, with the statement **new**$(a);b$. The intended meaning of the latter equals that of $a\|b$, which in turn equals the set $\{ab, ba\}$. To overcome this problem we introduce an auxiliary semantic operator ":" which is able, somewhat surprisingly, as it were dynamically to make the decision whether to opt for "$\cdot$" or "$\|$". We consider the introduction of this operator, together with the derivation of its basic technical properties (such as associativity) as a main contribution of our paper.

In Sections 5 and 6 we investigate the nonuniform case. $\mathcal{L}_{\text{nus}}$ has simple communication commands which are syntactic variations on CSP's $P_i?x$ and $P_j!e$ constructs. We stress that our mentioning CSP here is only to indicate the type of communication we have in our language. Partial, let alone full, modelling of CSP is not our aim here. The mathematical structures used to model $\mathcal{L}_{\text{nus}}$ and $\mathcal{L}_{\text{nud}}$ are Plotkin's *resumptions* [37], presented in a fully metric framework as first described in [17] and subsequently extended and put in a category-theoretic perspective in [8]. We use the terminology of *process domains P*, satisfying certain (reflexive) *domain equations* of the form

$$P \cong \mathcal{F}(P)$$

and we shall design the semantics of programs in $\mathcal{L}_{\text{nus}}$ and $\mathcal{L}_{\text{nud}}$ such that the meaning of a program is a *process* $p \in P$. Processes are objects which have a branching structure, and the models for $\mathcal{L}_{\text{nus}}$ and $\mathcal{L}_{\text{nud}}$ are called *branching time* [11, 40].

The operational models for $\mathcal{L}_{\text{nus}}$ and $\mathcal{L}_{\text{nud}}$ once more use SOS style transitions. An important new feature is that, in defining the operational meaning of a program, we collect the information from the induced transition steps into a process. In other words, we assemble the information in successive transition steps into a branching time object. Denotationally, we also use processes as meanings, obtained in the usual manner by a *compositional* system of defining equations. For the nonuniform languages, we do not have that $\mathcal{O}$ and $\mathcal{D}$ yield the same function: In order to allow a compositional definition of $\mathcal{D}$ for the communication constructs, we include in $\mathcal{D}[\![s]\!]$ more information than in $\mathcal{O}[\![s]\!]$ (here s is a nonuniform, static or dynamic, statement). We therefore introduce a natural extension $\mathcal{O}^*$ of $\mathcal{O}$, which preserves one-sided communication information, and then on the one hand establish that $\mathcal{O}^* = \mathcal{D}$, and on the other hand settle the relationship between $\mathcal{O}$ and $\mathcal{O}^*$ in terms of an *abstraction operator abs*, resulting in the equivalence $\mathcal{O} = abs \circ \mathcal{O}^*$.

In Section 6, we combine the techniques designed for $\mathcal{L}_{\text{ud}}$ and $\mathcal{L}_{\text{nus}}$ to deal with all of $\mathcal{L}_{\text{nud}}$. In this way, the reader may obtain a better understanding of this somewhat complicated case: The concepts of process creation and value communication have first been treated in isolation, and now a synthesis of the methods from Sections 4 and 5 is made. In $\mathcal{L}_{\text{nud}}$ we have *classes* (ultimately stemming from Simula [24]), and creation of a process amounts to the creation of a new instance of a class (in the world of object-oriented programming, this instance would be called a (new) object). Such an instance has a name which is (just) another value—in addition to values such as integers or truth-values—and which may be assigned to a variable. In $\mathcal{L}_{\text{nud}}$ we encounter for the first time *expressions* with nontrivial semantics. Consequently, the syntactic and semantic statement continuations used in previous sections are now extended with (syntactic and semantic) *expression continuations*. Operational and denotational semantics for $\mathcal{L}_{\text{nud}}$ are without major surprises once one has digested Sections 4 and 5. At various points, the definitions owe much to similar definitions in [6, 7], though a systematic redesign has been applied in order to allow the final equivalence proof. Again, techniques of Sections 4 and 5 are brought together, in particular leading to a nonuniform generalization of the ":" operator. Also, an additional argument is necessary to deal with the two forms of recursion now present, one in recursive procedures and the other in recursively defined classes.

This concludes our overview of the contents of the paper. We also mention that in Section 2 we collect some mathematical preliminaries. We list elementary definitions and some useful theorems in metric topology, and provide a brief sketch of the intuition and mathematical basis for (our way of) solving process domain equations.

Detailed semantic models of process creation are scarce in the literature. Semantic studies are reported in a few of the already cited papers [2, 23, 42, 43], but these are all focused on very different problems and techniques. Our work shares with [22] the central role played by continuations. However, that paper investigates process creation in a (deterministic) dataflow setting, and does not address semantic equivalence issues.

Our debt to Plotkin's seminal work in semantics should be clear from the above. To Nivat we are indebted for stimulating our interest in metric techniques going back to his lectures in [35]. Without the detailed semantic analysis of POOL described in [6, 7], the present paper would have been impossible. Many of our semantic definitions can be traced back to concepts and techniques first developed in these two papers.

2. Mathematical preliminaries

2.1. Notation

If X is a set, we denote with $\mathcal{P}(X)$ the power set of X, i.e., the collection of all subsets of X. $\mathcal{P}_\pi(X)$ denotes the collection of all subsets of X which have property π. A sequence $x_0, x_1, \ldots$ of elements of X is usually denoted by $(x_i)_{i=0}^{\infty}$ or, briefly, $(x_i)_i$. The notation $f: X \to Y$ expresses that f is a function with domain X and range Y. We use the notation $f\{y/x\}$, with $x \in X$ and $y \in Y$, for a *variant* of f, i.e., for the function which is defined by

$$f\{y/x\}(x') = \begin{cases} y & \text{if } x = x', \\ f(x') & \text{otherwise.} \end{cases}$$

If $f: X \to X$ and $f(x) = x$, we call x a *fixed point* of f.

2.2. Metric spaces

Metric spaces are the mathematical structures in which we carry out our semantic work. We give only the facts most needed in this paper. For more details, the reader is referred to [25, 26].

2.1. Definition. A metric space is a pair $\langle M, d\rangle$ where M is a nonempty set and d is a mapping $M \times M \to [0, 1]$ having the following properties:

(1) $\forall x, y \in M\, [d(x, y) = 0 \Leftrightarrow x = y]$,

(2) $\forall x, y \in M\, [d(x, y) = d(y, x)]$,

(3) $\forall x, y, z \in M\, [d(x, y) \leqslant d(x, z) + d(z, y)]$.

(d is called a *metric* or *distance.*)

Examples. (1) Let A be an arbitrary set. The *discrete metric* on A is defined as follows: Let $x, y \in A$.

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

(2) 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

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

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

2.2. Definition. Let $\langle M, d\rangle$ be a metric space and let $(x_i)_i$ be a sequence in M.

(1) We say that $(x_i)_i$ is a *Cauchy sequence* whenever we have

$$\forall \varepsilon > 0\, \exists N \in \mathbb{N}\, \forall n, m > N\, [d(x_n, x_m) < \varepsilon].$$

(2) 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

$$\forall \varepsilon > 0\, \exists N \in \mathbb{N}\, \forall n > N\, [d(x, x_n) < \varepsilon].$$

We call the sequence $(x_i)_i$ *convergent* and write $x = \lim_i x_i$.

(3) $\langle M, d\rangle$ is called *complete* whenever each Cauchy sequence in M converges to an element of M.

2.3. Definition. Let $\langle M_1, d_1\rangle$ and $\langle M_2, d_2\rangle$ be metric spaces.

(1) We say that $\langle M_1, d_1\rangle$ and $\langle M_2, d_2\rangle$ are *isometric* if there is a mapping $f: M_1 \to M_2$ such that

(a) f is a bijection,

(b) $\forall x, y \in M_1\, [d_2(f(x), f(y)) = d_1(x, y)]$.

We then write $M_1 \cong M_2$. If we have a function f satisfying only condition (1)(b), we call it an isometric embedding.

(2) Let $f: M_1 \to M_2$. We call f *continuous* whenever, for each sequence $(x_i)_i$ with limit x in M_1, we have that $\lim_i f(x_i) = f(x)$. We shall denote the set of all continuous functions from M_1 by $M_1 \to^{c} M_2$.

(3) We call a function $f: M_1 \to M_2$ *contracting* if there exists a real number c with $0 \le c < 1$ such that

$$\forall x, y \in M_1\ [d_2(f(x), f(y)) \le c\, d_1(x, y)].$$

(4) A function $f: M_1 \to M_2$ is called *non-distance-increasing* if

$$\forall x, y \in M_1\ [d_2(f(x), f(y)) \le d_1(x, y)].$$

We shall denote the set of all non-distance-increasing functions from M_1 to M_2 by $M_1 \to^{\mathrm{NDI}} M_2$.

2.4. Lemma. *Let $\langle M_1, d_1\rangle$ and $\langle M_2, d_2\rangle$ be metric spaces, and let $f: M_1 \to M_2$ be a contracting function. Then f is continuous. The same holds for non-distance-increasing functions.*

2.5. Theorem (Banach). *Let $\langle M, d\rangle$ be a complete metric space. Each contracting function $f: M \to M$ has a unique fixed point which equals $\lim_i f^i(x_0)$ for arbitrary $x_0 \in M$. (Here $f^0(x_0) = x_0$ and $f^{i+1}(x_0) = f(f^i(x_0))$.)*

Proof. Since f is contracting, the sequence $(f^i(x_0))_i$ is a Cauchy sequence. By the completeness of $\langle M, d\rangle$, the limit $x = \lim_i f^i(x_0)$ exists. By the continuity of f (Lemma 2.4), $f(x) = f(\lim_i f^i(x_0)) = \lim_i f^{i+1}(x_0) = x$. If, for some $y \in M$, $f(y) = y$ then, by the contractivity of f, $d(x, y) = d(f(x), f(y)) \le c\, d(x, y)$. Hence, since $c < 1$ we conclude that $d(x, y) = 0$, and $x = y$ follows. □

2.6. Definition. Let $\langle M, d\rangle$ be a metric space.

(1) A subset X of M is called *closed* whenever each converging sequence with elements in X has its limit in X.

(2) A subset X of M is called *compact* whenever each sequence in X has a subsequence which converges to an element of X.

Remarks. (1) The definition of compactness given here is in fact what is called *sequential compactness* in general topology. In a metric space this is equivalent to compactness.

(2) Taking, in Definition 2.6(2), X equal to M defines when the space $\langle M, d\rangle$ is called compact.

(3) In a metric space every compact set is closed.

2.7. Definition. Let $\langle M, d\rangle$, $\langle M_1, d_1\rangle$, and $\langle M_2, d_2\rangle$ be metric spaces.

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

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

(2) We define a metric d_P on the Cartesian product $M_1 \times M_2$ by

$$d_P(\langle x_1, y_1\rangle, \langle x_2, y_2\rangle) = \max_{i \in \{1,2\}} d_i(x_i, y_i).$$

(3) With $M_1 \sqcup M_2$ we denote the *disjoint union* of M_1 and M_2, which may be defined as $(\{1\} \times M_1) \cup (\{2\} \times M_2)$. We define a metric d_U on $M_1 \sqcup M_2$ as follows:

$$d_U(x, y) = \begin{cases} d_i(x, y) & \text{if } x, y \in \{i\} \times M_i \text{ for } i = 1 \text{ or } i = 2, \\ 1 & \text{otherwise.} \end{cases}$$

In the sequel we shall often write $M_1 \cup M_2$ instead of $M_1 \sqcup M_2$, implicitly assuming that M_1 and M_2 are already disjoint.

(4) Let $\mathcal{P}_{cl}(M) = \{X \mid X \subseteq M, X \text{ closed}\}$. We define a metric d_H on $\mathcal{P}_{cl}(M)$, called the *Hausdorff distance*, as follows:

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

where $d(x, Z) = \inf_{z \in Z} d(x, z)$ (here we use the convention that $\sup \emptyset = 0$ and $\inf \emptyset = 1$, so that the empty set will have distance 1 to every other set).

2.8. Theorem. *Let $\langle M, d\rangle$, $\langle M_1, d_1\rangle$, $\langle M_2, d_2\rangle$, d_F, d_P, d_U, and d_H be as in Definition* 2.7, *and suppose in addition that $\langle M, d\rangle$, $\langle M_1, d_1\rangle$, and $\langle M_2, d_2\rangle$ are* complete. *We have that*

(1) $\langle M_1 \to M_2, d_F\rangle$ *(together with $\langle M_1 \to^{c} M_2, d_F\rangle$ and $\langle M_1 \to^{\text{NDI}} M_2, d_F\rangle$)*,

(2) $\langle M_1 \times M_2, d_P\rangle$,

(3) $\langle M_1 \sqcup M_2, d_U\rangle$,

(4) $\langle \mathcal{P}_{cl}(M), d_H\rangle$

are complete metric spaces. (Strictly speaking, for the completeness of $M_1 \to M_2$, the completeness of M_1 is not required.)

In the sequel we shall often write $M_1 \to M_2$, $M_1 \times M_2$, $M_1 \sqcup M_2$, $\mathcal{P}_{cl}(M)$, etc., when we mean the metric spaces with the metrics just defined.

The proofs of parts (1), (2), and (3) of Theorem 2.8 are straightforward. Part (4) is more involved. It can be proved with the help of the following characterization of completeness of $\langle \mathcal{P}_{cl}(M), d_H\rangle$.

2.9. Theorem. *Let $\langle \mathcal{P}_{cl}(M), d_H\rangle$ be as in Definition* 2.7. *Let $(X_i)_i$ be a Cauchy sequence in $\mathcal{P}_{cl}(M)$. We have*

$$\lim_i X_i = \{\lim_i x_i \mid x_i \in X_i,\ (x_i)_i \text{ a Cauchy sequence in } M\}.$$

Theorem 2.9 is due to Hahn [28]. Proofs of Theorems 2.8 and 2.9 can be found, e.g., in [25] or [26]. The proofs are also repeated in [17].

2.10. Theorem (Metric completion). *Let M be an arbitrary metric space. Then there exists a metric space $\bar{M}$ (called the* completion *of M) together with an isometric embedding $i : M \to \bar{M}$ such that*

(1) *$\bar{M}$ is complete,*

(2) *for every complete metric space M' and isometric embedding $j : M \to M'$ there exists a unique isometric embedding $\bar{j} : \bar{M} \to M'$ such that $\bar{j} \circ i = j$.*

Proof. Standard topology. □

Finally, we have the following result from Rounds [41].

2.11. Theorem. *Let $f: M_1 \to M_2$ be an arbitrary function, where M_1 and M_2 are* compact *metric spaces, and define $\hat{f}: \mathcal{P}_{\mathrm{cl}}(M_1) \to \mathcal{P}(M_2)$ by $\hat{f}(X) = \{f(x) \mid x \in X\}$. Then the following statements are equivalent:*

(1) *f is continuous.*

(2) *For every $X \in \mathcal{P}_{\mathrm{cl}}(M_1)$ we have $\hat{f}(X) \in \mathcal{P}_{\mathrm{cl}}(M_2)$, and $\hat{f}$ is continuous with respect to the Hausdorff metrics.*

(3) *For every $X \in \mathcal{P}_{\mathrm{cl}}(M_1)$ we have $\hat{f}(X) \in \mathcal{P}_{\mathrm{cl}}(M_2)$, and, for each decreasing chain $(X_i)_i$ (i.e., $X_i \supseteq X_{i+1}$ for all i) of elements in $\mathcal{P}_{\mathrm{cl}}(M_1)$ we have*

$$\hat{f}\Big(\bigcap_i X_i\Big) = \bigcap_i \hat{f}(X_i).$$

2.3. Resumptions and domain equations

We begin with a brief intuitive introduction of the notion of *resumption* (due to Plotkin [37]). We use the terminology of *processes p, q,* which are elements of a *process domain P.* We emphasize that we are concerned here with semantics rather than with syntax: processes are elements of mathematical structures rather than (pieces of) program texts. Process domains are obtained as solutions of *domain equations.* In this informal introduction we let A and B stand for arbitrary (fixed) sets (where necessary provided with the discrete metric) and we shall denote by p_0 an arbitrary mathematical object which shall play the role of a *nil* process. A very simple equation is

$$P \cong \{p_0\} \cup (A \times P). \tag{2.1}$$

We can read this equation as follows: a process $p \in P$ is either p_0, which cannot take any action, or it is a pair $\langle a, q\rangle \in A \times P$, where a is the first action taken and q is the *resumption*, describing the rest of p's actions. Clearly, (2.1) has as a solution the set of all finite sequences $\langle a_1, a_2, \ldots, a_n, p_0\rangle$, with $n \geqslant 0$ and $a_i \in A$ for all i. The set of all these finite sequences plus all infinite sequences $\langle a_1, a_2, \ldots\rangle$ is another solution.

We next consider

$$P \cong \{p_0\} \cup (A \to (B \times P)). \tag{2.2}$$

This is already a much more interesting equation: each process p is either p_0 or a function which, when supplied with an argument a, yields a pair $p(a) = \langle b, p'\rangle$. We see that p maps a to b, at the same time turning itself into the resumption p'. We can say that p determines its first step b and the resumption p' on the basis of a.

The following equation we consider is

$$P \cong \{p_0\} \cup (A \to \mathcal{P}_{\mathrm{cl}}(B \times P)). \tag{2.3}$$

Now, if we feed a process $p \neq p_0$ with some $a \in A$, a whole set X of possible pairs $\langle b, q\rangle$ results, among which the process can choose freely. For reasons of cardinality, (2.3) has no solution when we take *all* subsets, rather than all closed subsets of $B \times P$. Moreover, we should be more precise about the metrics involved. We should have written (2.3) like this:

$$P \cong \{p_0\} \cup (A \to \mathcal{P}_{\mathrm{cl}}(B \times \mathrm{id}_{1/2}(P))) \tag{2.3$'$}$$

where, for any positive real number c, id_c maps a metric space $\langle M, d\rangle$ into $\langle M, d'\rangle$ with $d'(x, y) = c\, d(x, y)$. We shall adopt the convention that in domain equations like (2.1), (2.2) and (2.3) every occurrence of the defined space P on the right-hand side is implicitly surrounded by $\mathrm{id}_{1/2}$. (Note that (2.1) and (2.2) can be solved even without this convention, resulting in a set of sequences or trees respectively, with the discrete metric.)

It will turn out that (2.3) is the right type of domain equation for our purposes. We shall, in Sections 5 and 6, specialize A and B to certain sets which have the appropriate semantic connotations. As we shall see later, an important advantage of processes as in (2.3) is that they allow a natural definition of their *merge*, which combines interleaving and communication steps in a way which is quite familiar in concurrency semantics (for one example, see ACP [18]).

We next discuss how one may solve equations as exemplified by (2.1) to (2.3). These equations are special cases of domain equations as studied in depth in the domain theory initiated by Scott and developed further by many researchers (including Plotkin's [37], see, e.g., [27] for a comprehensive reference). We shall here briefly sketch an approach to the solution of such domain equations which is fully couched in the setting of (complete) metric spaces (first described in [17]) and, in this way, avoids any mention of order-theoretic structures. We thus obtain a unified mathematical foundation for our semantics since we exclusively base ourselves on metric techniques. We present a somewhat streamlined version of the results in [17]. There is an important class of domain equations not covered in that paper, viz. equations of the form

$$P \cong \cdots (P \to \cdots) \cdots \tag{2.4}$$

i.e., involving functional domains with the "unknown" domain on the left-hand side of "$\to$". Recently, a fuller treatment of the metric approach has been described by America and Rutten [8]. There, equations $P \cong \mathcal{F}(P)$ are solved in a category of metric spaces, also catering for situations as in (2.4). For the purpose of the present paper, the restricted case to be described below suffices, and we thus avoid the introduction of various category-theoretic notions which are not essential for the applications at hand.

We consider a domain equation

$$P \cong \mathcal{F}(P) \tag{2.5}$$

where $\mathcal{F}$ is a function (technically, a functor on the category of complete metric spaces, but we do not have to be aware of this) which is constructed according to the following syntax (where c is a real number, $0<c<1$, and M an arbitrary complete metric space with metric d_M):

$$\mathcal{F} ::= \mathcal{F}_M \mid \mathrm{id}_c \mid \mathcal{F}_1 \times \mathcal{F}_2 \mid \mathcal{F}_1 \sqcup \mathcal{F}_2 \mid \mathcal{P}_{\mathrm{cl}}(\mathcal{F}') \mid \mathcal{F}_M \to \mathcal{F}'. \tag{2.6}$$

The above definition of $\mathcal{F}$ should be understood as follows. For each complete metric space $\langle Q, d\rangle$ we define the complete metric space $\langle \mathcal{F}(Q), \mathcal{F}(d)\rangle$ to which $\mathcal{F}$ maps $\langle Q, d\rangle$:

(1) $\mathcal{F}_M(Q) = M$, $\mathcal{F}_M(d) = d_M$. Thus, $\mathcal{F}_M$ is the constant function, yielding $\langle M, d_M\rangle$ for every Q. In various applications, we just give some arbitrary set A and assume for A the discrete metric.

(2) $\mathrm{id}_c(Q) = Q$, $\mathrm{id}_c(d)(x, y) = c\,d(x, y)$.

(3) If $\mathcal{F} = \mathcal{F}_1 \times \mathcal{F}_2$, assume that $\mathcal{F}_i(Q) = Q_i$ and $\mathcal{F}_i(d) = d_i$ for $i = 1, 2$. Then we put $\mathcal{F}(Q) = Q_1 \times Q_2$ and $\mathcal{F}(d) = d_{\mathrm{P}}$ (see Definition 2.7).

(4) If $\mathcal{F} = \mathcal{F}_1 \sqcup F_2$, assume again that $\mathcal{F}_i(Q) = Q_i$ and $\mathcal{F}_i(d) = d_i$ for $i = 1, 2$. Then we put $\mathcal{F}(Q) = Q_1 \sqcup Q_2$ and $\mathcal{F}(d) = d_{\mathrm{U}}$ (see Definition 2.7).

(5) If $\mathcal{F} = \mathcal{P}_{\mathrm{cl}}(\mathcal{F}')$, assume that $\mathcal{F}'(Q) = Q'$ and $\mathcal{F}'(d) = d'$. Now we put $\mathcal{F}(Q) = \mathcal{P}_{\mathrm{cl}}(Q')$ and $\mathcal{F}(d) = (d')_{\mathrm{H}}$ (see Definition 2.7).

(6) If $\mathcal{F} = \mathcal{F}_M \to \mathcal{F}'$, we already know that $\mathcal{F}_M(Q) = M$ and $\mathcal{F}_M(d) = d_M$. Now assume that $\mathcal{F}'(Q) = Q'$ and $\mathcal{F}'(d) = d'$. We put $\mathcal{F}(Q) = M \to Q'$ and $\mathcal{F}(d) = (d')_{\mathrm{F}}$, where $(d')_{\mathrm{F}}$ is the function metric on $M \to Q'$ derived from d' (see Definition 2.7).

According to [17], for $\mathcal{F}$ as just given we can solve (2.5) by the following scheme: Define inductively

$$P_0 = \langle \{p_0\}, d_0\rangle \quad d_0 \text{ the discrete metric,}$$

$$P_{n+1} = \mathcal{F}(P_n).$$

Observe that—ignoring the obvious identification of P with $\{i\} \times P$ for $i = 1, 2$ in case $\mathcal{F}$ involves a disjoint union—we have for all n

$$P_n \subseteq P_{n+1}. \tag{2.7}$$

Now we put $\langle P_\omega, d_\omega\rangle = \langle \bigcup_n P_n, \bigcup_n d_n\rangle$ (with the obvious interpretation of $\bigcup_n d_n$) and we define $\langle \bar{P}, \bar{d}\rangle$ as the *completion* (see Theorem 2.10) of $\langle P\omega, d_\omega\rangle$. Then we have the following theorem.

2.12. Theorem. *For $\mathcal{F}$ and $\bar{P}$ as above, we have $\bar{P} \cong \mathcal{F}(\bar{P})$.*

Proof. A nonessential variation of the results of [17]. □

Remark. The scope of the techniques applied in the proof of Theorem 2.12 was not fully understood in [17], and substantial clarification was provided by [8]. In addition, [8] brings an essential generalization: The clause $\mathcal{F}_M \to \mathcal{F}'$ in (2.6) is replaced by $\mathcal{F}_1 \to \mathcal{F}_2$, thus dropping the restriction that only constants appear on the left-hand side of "$\to$". A precise analysis is provided of the ensuing situation, involving the notion of *contraction coefficient* $c \geqslant 0$ of a functor $\mathcal{F}$, and culminating in the result that, for $c < 1$, (2.5) has a unique solution (up to isometry). A key step in this analysis is a generalization of (2.7): in the presence of general functional domains we can no longer gloss over the need for a precise embedding of P_n into P_{n+1}, and a rigorous definition of an *arrow* $\iota : P_n \to P_{n+1}$ is needed. For arbitrary complete metric spaces $\langle M_1, d_1 \rangle$ and $\langle M_2, d_2 \rangle$, such an arrow $\iota : M_1 \to M_2$ is a pair $\langle i, j \rangle$ with $i : M_1 \to M_2$ an isometric embedding and $j : M_2 \to M_1$ a non-distance-increasing function such that $j \circ i$ is equal to the identity function on M_1.

3. A uniform and static language

We begin with a detailed study of $\mathscr{L}_{\text{us}}$, a uniform and static language. First we present its syntax, and its operational semantics in the style of Hennessy and Plotkin [29, 38, 39]. Next, we develop the metric framework to define the denotational semantics for $\mathscr{L}_{\text{us}}$. Finally, we discuss the relationship between the two semantics and outline an equivalence proof. Most of this section can already be found in [16, Section 2]; we repeat this material here to make the present paper self-contained and to prepare the way for the treatment of the dynamic case in the next section. There are a few new points in the development presented below as well, partly due to the fact that $\mathscr{L}_{\text{us}}$ has only one level of parallelism, partly caused by our wish to achieve a smooth transition to the definitions for $\mathscr{L}_{\text{ud}}$, the language with dynamic parallelism (a notion not treated in [16]). The latter aim has in particular motivated our use below of the technique of *continuations*.

3.1. Syntax and preliminary definitions

Let A be a finite alphabet of *elementary actions*, with typical elements a, b, c (by this we mean that the letters a, b, and c, possibly adorned with primes or subscripts, will be used to range over elements of A) and let *StmV* be an infinite set of *statement variables*, with typical elements x, y. Statement variables are used in the syntactic construct for *recursion*, as we shall see in a moment.

3.1. Definition (*Syntax for statements and programs*). (1) The set $\mathscr{S}_{\text{us}}$ of (uniform and static) *statements*, with typical element s, is defined by

$$s ::= a \mid x \mid s_1 ; s_2 \mid s_1 \cup s_2 \mid \mu x[s']$$

The prefix μx in the construct $\mu x[s']$ *binds* occurrences of x in s' in the usual way. We call a statement s *closed* if it contains no free occurrences of statement variables.

(2) The set $\mathscr{L}_{\text{us}}$ of (uniform and static) *programs*, with typical element t, is defined by

$$t ::= s_1 \| \cdots \| s_n \quad (n \geq 1).$$

Here we require that $s_1, \ldots, s_n$ are all closed (so that programs are always closed).

Examples. (1) Statements: $a;b$, $\mu x[(a;x)\cup b]$, $\mu x[(a;x)\cup(x;b)\cup c]$, $\mu x[(a_1;x;a_2)\cup\mu y[(y;b)\cup c]]$, $a;y;b$ (only the last example is not closed).

(2) Programs: Each of the closed statements listed under (1), and, in addition, $(a;b)\,\|\,\mu x[(a;x)\cup b]\,\|\,\mu x[(x;b)\cup c]$, $\mu x[a;x]\,\|\,\mu y[b;y]$.

A statement s is of one of the following forms:

- an elementary action a,
- the sequential composition $s_1;s_2$ of statements s_1 and s_2,
- the nondeterministic choice $s_1\cup s_2$ (also known as local or internal nondeterminism): $s_1\cup s_2$ is executed by executing either s_1 or s_2, where the choice is made nondeterministically.
- a statement variable x, which is (primarily) used in:
- the recursive construct $\mu x[s]$: its execution amounts to execution of s, where occurrences of x in s are executed by (recursively) executing $\mu x[s]$. For example, with the semantic definitions to be proposed presently, the intended meaning of $\mu x[(a;x)\cup b]$ is the set $a^*\cdot b\cup\{a^\omega\}$.

A program $t=s_1\|\cdots\|s_n$ consists of $n\geqslant 1$ statements which are to be executed in parallel. Since n remains fixed throughout the execution of t, we call the language $\mathscr{L}_{\mathrm{us}}$ *static* to distinguish it from the dynamic language $\mathscr{L}_{\mathrm{ud}}$ studied in Section 4.

$\mathscr{L}_{\mathrm{us}}$ has no synchronization or communication. The issues which arise when such notions are added to it are studied in detail in (later sections of) [16]. We do not want to complicate our treatment of $\mathscr{L}_{\mathrm{us}}$—which plays only a preliminary role in the present context—by including such ramifications.

Substitution of a statement for a statement variable is defined in the familiar way: $s[s'/x]$ denotes the result of substituting s' for all free occurrences of x in s, with the usual precaution of renaming bound variables when necessary to avoid clashes.

In both operational and denotational models we shall use the universe of *streams*, defined as follows.

3.2. Definition (*Streams*, cf. [20, 21]). We assume that $\perp\notin A$. The set A^{st} of all *streams* over A is defined by

$$A^{\mathrm{st}}=A^*\cup A^\omega\cup(A^*\times\{\perp\})$$

where A^* (A^ω) is the set of all finite (infinite) words over A.

We shall use u, v, w to range over A^{st} and use ε for the empty stream. Streams of the form $\langle u,\perp\rangle$ will be written as $u\cdot\perp$ or simply $u\perp$. We shall abbreviate $\langle\varepsilon,\perp\rangle$ to $\perp$. The use of $\perp$ is motivated, in an operational setting, by our wish to produce some visible result as the outcome of an infinite computation that does not produce an infinite sequence of elementary actions. For example, we shall organize the definitions such that both $\mu x[x]$ and $\mu x[(x;b)\cup c]$ deliver $\perp$ as an outcome (in the latter case together with cb^*).

We shall use a^ω for the infinite sequence of a's. *length*(u) yields the number of symbol occurrences (from $A \cup \{\perp\}$) in u. In particular, for $u \in A^\omega$, *length*$(u) = \infty$, and for $u = u'\perp$, $u' \in A^*$, we have *length*(u) = *length*$(u')+1$. we use "$\leqslant$" for the prefix ordering on A^{st}, i.e., we put $u \leqslant v$ whenever $u = v$ or $u \in A^*$ and, for some $w \in A^{\mathrm{st}}$, $u \cdot w = v$ (the reader who wants to see a precise definition of the concatenation "$\cdot$" of streams is referred to Definition 3.12). For example, we have $ab \leqslant abc$, $a^n \leqslant a^\omega$, $ab \leqslant ab\perp$, but $a\perp \not\leqslant ab\perp$. We recall that each $\leqslant$-chain $(u_i)_i$, with $u_i \leqslant u_{i+1}$, $i = 0, 1, \ldots$, has a least upper bound $u = \mathrm{lub}_i\, u_i$ in A^{st}, where $(u_i)_i$ is either infinitely often increasing ($u_i \neq u_{i+1}$ for infinitely many i) and then $u \in A^\omega$, or $(u_i)_i$ *stabilizes* in some u_{i_0} ($u_i = u_{i_0}$ for all $i \geqslant i_0$), and then $u = u_{i_0}$. We conclude this list of definitions with the notation $u(n)$, which denotes the $\leqslant$-prefix of u of length n in case this exists, and which equals u otherwise.

In both this and all subsequent sections we shall make extensive use of so-called *continuations*, both of syntactic and semantic variety. In defining the semantics of a statement, we shall use a continuation to indicate the "actions" which remain to be done *after* this statement. Syntactically, this is done by a piece of program text, a syntactic continuation, to be defined below. Semantic continuations will be introduced in Section 3.3. The use of continuations in the context of $\mathscr{L}_{\mathrm{us}}$ is not necessary or especially helpful, but it introduces the techniques which will be applied fruitfully in the following sections.

We shall denote the empty syntactic continuation by E (note that E is not itself a statement) and then define the following sets.

3.3. Definition (*Syntactic continuations*). (1) The set *SyCo* of *syntactic continuations*, with typical element r, is defined by

$$r \ ::= \ E \mid s; r'$$

Here we require that each statement s occurring in a syntactic continuation r is closed (so that syntactic continuations are always closed).

(2) We define the set *PSyCo* of *parallel syntactic continuations*, with typical element ρ, as follows:

$$\rho \ ::= \ r_1, \ldots, r_n \quad (n \geqslant 1).$$

3.2. Operational semantics

We now proceed with the operational semantics for $\mathscr{S}_{\mathrm{us}}$ and $\mathscr{L}_{\mathrm{us}}$. We apply the technique of *transition systems*, introduced by Hennessy and Plotkin [29, 38, 39], and proven to be quite fruitful in a variety of concurrency semantics. The particular version employed below is close to the style of definition in [9, 10], though these papers deal in fact with interpreted rather than with uninterpreted languages (cf., for example, the discussion in [12] of the distinction between uniform and nonuniform). In [16] we also discuss the relationships between our version of the transition formalism and other variants one may encounter in the literature.

A configuration is either a pair $\langle \rho, w\rangle$, with $w \in A^* \times \{\perp\}$, or simply a stream w, with $w \in A^*$. A *transition* is a pair of configurations of the form

$$\langle \rho, w\rangle \to \langle \rho', w'\rangle \quad \text{or} \quad \langle \rho, w\rangle \to w''$$

(where $w, w' \in A^* \times \{\perp\}$, $w'' \in A^*$). In order to understand such transitions, we first mention—anticipating later precise definitions—that a program $t = s_1 \| \cdots \| s_n$ will correspond to a parallel continuation $\rho = s_1;E, \ldots, s_n;E$. For each configuration $\langle \rho, w\rangle$, we view ρ as the program currently to be executed, and w as an (unfinished) stream of elementary actions collected so far. The "$\to$" relation as given above either reflects a one-step transition to a new such pair $\langle \rho', w'\rangle$, or a one-step transition to a (finished) stream w''. The *transition system* to be defined in a moment provides the information necessary to *deduce* transitions of the given form. More precisely, we shall define the relation "$\to$" between configurations as the *smallest* (with respect to set inclusion) relation which satisfies the axioms given in the following definition.

3.4. Definition (*Transition system for $\mathscr{L}_{\text{us}}$*). The system $\mathscr{T}_{\text{us}}$ for $\mathscr{L}_{\text{us}}$ consists of the following five *axioms* (in a self-explanatory notation):

$$\langle \ldots, a;r, \ldots, w\perp\rangle \to \langle \ldots, r, \ldots, wa\perp\rangle, \qquad \textbf{Elem}$$

$$\langle \ldots, (s_1;s_2);r, \ldots, w\rangle \to \langle \ldots, s_1;(s_2;r), \ldots, w\rangle, \qquad \textbf{SeqComp}$$

$$\langle \ldots, (s_1 \cup s_2);\, r, \ldots, w\rangle \to \langle \ldots, s_1;r, \ldots, w\rangle \,|\, \langle \ldots, s_2;r, \ldots, w\rangle \qquad \textbf{Choice}$$

(here $X \to Y | Z$ is short for $X \to Y$ and $X \to Z$),

$$\langle \ldots, \mu x[s];r, \ldots, w\rangle \to \langle \ldots, s[\mu x[s]/x];r, \ldots, w\rangle, \qquad \textbf{Rec}$$

$$\langle E, \ldots, E, w\perp\rangle \to w. \qquad \textbf{Term}$$

(Note that, by our conventions, in the first and fifth axiom $w \in A^*$, and in the remaining ones $w \in A^* \times \{\perp\}$.)

Our next step is the definition of a semantic function $\mathcal{O}[\![\cdot]\!]$, yielding, when applied to some ρ, a subset of A^{st}.

3.5. Definition. We define the function

$$\mathcal{O}[\![\cdot]\!]: PSyCo \to \mathscr{P}(A^{\text{st}})$$

as follows. Let $\rho \in PSyCo$. We put a stream w into $\mathcal{O}[\![\rho]\!]$ whenever one of the following conditions is satisfied:

(1) There is a finite sequence of configurations $(\langle \rho_i, w_i\rangle)_{i=0}^{n}$ such that $\langle \rho_i, w_i\rangle \to \langle \rho_{i+1}, w_{i+1}\rangle$ for $i = 0, \ldots, n-1$, $\rho_0 = \rho$, $w_0 = \perp$, and $\langle \rho_n, w_n\rangle \to w$.

(2) There is an infinite sequence of configurations $(\langle \rho_i, w_i\rangle)_{i=0}^{\infty}$, such that $\langle \rho_i, w_i\rangle \to \langle \rho_{i+1}, w_{i+1}\rangle$ for $i = 0, 1, \ldots$, $\rho_0 = \rho$, $w_0 = \perp$, $w_i = w_i'\perp$, and $w = (\text{lub}_i\, w_i')\perp$.

Remark. In clause (2) we use the obvious fact that if $\langle \rho, w\bot\rangle \to \langle \rho', w'\bot\rangle$, then $w \leq w'$. Note that, for $(w_i')_i$ infinitely often increasing, $w' =^{\mathrm{def}} \mathrm{lub}_i\, w_i'$ belongs to A^ω, so from the definition $w = w'\bot$ we infer that $w = w'$ (by Definition 3.12, concatenating any stream to the right of some infinite stream has no effect). For $(w_i')_i$ stabilizing in w_{i_0}', we obtain $w = w_{i_0}'\bot$.

Examples. (1) $\mathcal{O}[\![\mu x[(a;x)\cup b];E]\!] = \{a^\omega\} \cup a^*b$, $\mathcal{O}[\![\mu x[(x;a)\cup b];E]\!] = \{\bot\} \cup ba^*$.
(2) $\mathcal{O}[\![(c\cup(a;b));E,d;E]\!] = \{cd, dc, dab, adb, abd\}$.

We conclude the operational semantics definitions with the definition of $\mathcal{O}[\![t]\!]$ for $t \in \mathcal{L}_{\mathrm{us}}$:

3.6. Definition. The mapping $\mathcal{O}[\![\cdot]\!] : \mathcal{L}_{\mathrm{us}} \to \mathcal{P}(A^{\mathrm{st}})$ is defined as follows. Let $t = s_1\| \cdots \|s_n \in \mathcal{L}_{\mathrm{us}}$. Then

$$\mathcal{O}[\![t]\!] = \mathcal{O}[\![s_1;E, \ldots, s_n;E]\!].$$

Remark. There is a natural connection between the notions discussed above when restricted to programs without parallelism ($t = s_1$) and the languages with finite or infinite words produced by context-free grammars in the sense of, e.g., Nivat [35]. For example, the grammar $X \to aXb|c$ produces $\{a^\omega\} \cup \{a^n cb^n \mid n \geq 1\}$, and so does $\mathcal{O}[\![\mu x[(a;x;b)\cup c]]\!]$. A difference arises in the presence of unguarded recursion (cf. Definition 3.14 below); for example, $\mathcal{O}[\![\mu x[(x;b)\cup c]]\!]$ equals $\{\bot\} \cup cb^*$, whereas $X \to Xb|c$ would, by Nivat's definitions, produce only cb^*. Briefly, the role of $\bot$ in our style(s) of semantics has no counterpart in traditional formal language theory. Fixed point considerations for infinitary languages generated by grammars which may be left recursive (in other words, which do not satisfy the Greibach condition) are discussed for instance by Niwinski [36].

A number of elementary properties of $\mathcal{O}[\![\cdot]\!]$ are collected in the following lemma.

3.7. Lemma. (1) $\mathcal{O}[\![E]\!] = \{\varepsilon\}$.
(2) $\mathcal{O}[\![a;r]\!] = a\cdot\mathcal{O}[\![r]\!]$.
(3) $\mathcal{O}[\![(s_1;s_2);r]\!] = \mathcal{O}[\![s_1;(s_2;r)]\!]$.
(4) $\mathcal{O}[\![(s_1\cup s_2);r]\!] = \mathcal{O}[\![s_1;r]\!] \cup O[\![s_2;r]\!]$.
(5) $\mathcal{O}[\![\mu x[s];r]\!] = \mathcal{O}[\![s[\mu x[s]/x];r]\!]$.

Remark. This lemma presupposes the formal definition of operations on (sets of) streams to be given in Definition 3.12.

Proof of Lemma 3.7. Obvious from the definitions. $\square$

3.3. Denotational semantics

By way of preparation for the denotational semantics for $\mathcal{L}_{\mathrm{us}}$, we present some basic definitions which introduce the metric setting we apply for this purpose.

3.8. Definition. We define the distance $d : A^{\mathrm{st}} \times A^{\mathrm{st}} \to [0, 1]$ by

$$d(u, v) = 2^{-\sup\{n \mid u(n) = v(n)\}},$$

where $2^{-\infty} = 0$.

Examples. $d(a_1a_2a_3, a_1a_2a_4) = 2^{-2}$; $d(a^n, a^\omega) = 2^{-n}$; $d(\varepsilon, \perp) = 1$.

3.9. Lemma. (1) *(A^{st}, d) is a complete metric space.*

(2) *For finite A, (A^{st}, d) is compact.*

Proof. See, e.g., [35]. □

Let $\mathcal{P}_{\mathrm{nc}}(A^{\mathrm{st}})$ denote the collection of all nonempty closed subsets of A^{st}. We usually abbreviate $\mathcal{P}_{\mathrm{nc}}(A^{\mathrm{st}})$ to S_{nc}. Let X, Y range over S_{nc}. We put $X(n) = \{u(n) \mid u \in X\}$. Now we also define a distance $\hat{d}$ on S_{nc}.

3.10. Definition. The distance $\hat{d} : S_{\mathrm{nc}} \times S_{\mathrm{nc}} \to [0, 1]$ is defined by

$$\hat{d}(X, Y) = 2^{-\sup\{n \mid X(n) = Y(n)\}},$$

where, again, $2^{-\infty} = 0$.

We have the following important theorem.

3.11. Theorem. (1) *$(S_{\mathrm{nc}}, \hat{d})$ is a complete metric space, and if A is finite, this space is compact.*

(2) *$\hat{d}$ coincides with the Hausdorff distance (cf. Definition 2.7) induced on S_{nc} by the distance d on streams.*

Proof. Part (2) is easy from the definitions, and part (1) then follows from Theorem 2.8 (together with a theorem that says that compactness also carries over from any M to $\mathcal{P}_{\mathrm{cl}}(M)$, see [25, 26]). The omission of the empty subset, which has distance 1 to every other subset does not disturb closedness or compactness. □

Remark. As a consequence of part (1) of Theorem 3.11, each Cauchy sequence $(X_n)_n$ in $(S_{\mathrm{nc}}, \hat{d})$ has a limit $\lim_n X_n$ in $(S_{\mathrm{nc}}, \hat{d})$, a fact we shall employ several times below.

Next we introduce three semantic operators "$\cdot$", "$\cup$", and "$\|$", which are counterparts of the syntactic operators of sequential composition, choice and parallel execution. The first two are well-known; the $\|$-operator (when applied to two sets) consists of the *shuffle* of all streams in the two operands. As remarked before, no operations involving synchronization or communication are considered for this language. The precise definition of the semantic operators proceeds in stages.

3.12. Definition (*Semantic operators*). (1) We assume as known the operation "$\cdot$" of prefixing an element $a \in A$ to a finite stream $u \in A^*$, yielding as a result $a \cdot u$ (also written as au). Moreover, we put $a \cdot \langle u, \bot \rangle = \langle au, \bot \rangle$ for $u \in A^*$.

(2) Assume $X, Y \subseteq A^* \cup (A^* \times \{\bot\})$. We define

(a) $a \cdot X = \{au \mid u \in X\}$;

(b) for $u \in A^* \cup (A^* \times \{\bot\})$, we define $u \cdot X$ by induction on the length of u, as follows: $\varepsilon \cdot X = X$, $\bot \cdot X = \{\bot\}$, $(au) \cdot X = a \cdot (u \cdot X)$;

(c) $X \cdot Y = \bigcup \{u \cdot Y \mid u \in X\}$;

(d) $X \cup Y$ is (indeed) the set-theoretic union of X and Y;

(e) $u \mathbin{\|\!\lfloor} W$ (which will be used in (2)(f) is defined by induction on the length of u, as follows: $\varepsilon \mathbin{\|\!\lfloor} X = X$, $\bot \mathbin{\|\!\lfloor} X = \{\bot\}$, $(au) \mathbin{\|\!\lfloor} X = a \cdot (\{u\} \parallel X)$;

(f) $X \parallel Y = (X \mathbin{\|\!\lfloor} Y) \cup (Y \mathbin{\|\!\lfloor} X)$, where $X \mathbin{\|\!\lfloor} Y = \bigcup \{u \mathbin{\|\!\lfloor} X \mid u \in X\}$.

(3) Assume that X and Y are arbitrary elements of $\boldsymbol{S}_{\mathrm{nc}}$, and let $\mathbf{op} \in \{\cdot, \cup, \parallel\}$. Then we put

$$X \mathbf{\ op\ } Y = \lim_n (X(n) \mathbf{\ op\ } Y(n)).$$

3.13. Lemma. (1) *The operators* $\mathbf{op}$ *from* $\{\cdot, \cup, \parallel\}$ *are well-defined. In particular, for each* $X, Y \in \boldsymbol{S}_{\mathrm{nc}}$, $(X(n) \mathbf{\ op\ } Y(n))_n$ *is a Cauchy sequence.*

(2) *Each* $\mathbf{op}$ *is a continuous mapping*: $\boldsymbol{S}_{\mathrm{nc}} \times \boldsymbol{S}_{\mathrm{nc}} \to \boldsymbol{S}_{\mathrm{nc}}$.

Proof. Either by combining results from [11] with Rounds's theorem (Theorem 2.11), or by appropriately modifying the proof as given in [17, Appendix B]. □

We need one last step before we can give the definition for the denotational semantic function $\mathcal{D}[\![\cdot]\!]$. We shall restrict the definition of $\mathcal{D}[\![\cdot]\!]$ to statements involving only *guarded* recursion defined as follows.

3.14. Definition. (1) A statement variable x may occur *exposed* in a statement s. This notion is inductively defined as follows:

(a) x occurs exposed in x;

(b) if x occurs exposed in s, then x occurs exposed in $s;s'$, $s \cup s'$, $s' \cup s$, and $\mu y[s]$ for $y \neq x$.

(2) A statement s is called *guarded* when for each of its recursive substatements of the form $\mu x[s']$ we have that x does not occur exposed in s'. A program $t = s_1 \parallel \cdots \parallel s_n$ is called guarded if all its constituents s_i are guarded.

Examples. The statements $\mu x[a;x]$ and $\mu x[\mu y[b;y];x]$ are guarded, whereas $\mu x[(x;b) \cup c]$ and $\mu y[\mu x[y];b]$ are unguarded.

Let $\mathcal{S}^{\mathrm{g}}_{\mathrm{us}}$ denote the sets of guarded statements and $\mathcal{L}^{\mathrm{g}}_{\mathrm{us}}$ the set of guarded programs. We shall now define the mappings $\mathcal{D}$:

$$\mathcal{D}[\![\cdot]\!] : \mathcal{S}^{\mathrm{g}}_{\mathrm{us}} \to (\Gamma \to (SeCo \xrightarrow{\mathrm{NDI}} \boldsymbol{S}_{\mathrm{nc}}))$$

and

$$\mathcal{D}[\![\cdot]\!] : \mathcal{L}^{\mathrm{g}}_{\mathrm{us}} \to \boldsymbol{S}_{\mathrm{nc}}$$

where Γ is the set of *environments* and *SeCo* the set of *semantic continuations*, both to be defined below. (Recall from Definition 2.3 that $\rightarrow^{\mathrm{NDI}}$ stands for the set of all non-distance-increasing functions.) We take γ to range over Γ and φ to range over $SeCo \rightarrow^{\mathrm{NDI}} \mathcal{S}_{\mathrm{nc}}$. The type of especially the first $\mathcal{D}$ might require some explanation; it means that we apply the function $\mathcal{D}$ to a guarded statement, an environment, and a continuation in order to get an element from $\mathcal{S}_{\mathrm{nc}}$, i.e., a nonempty, closed set of streams.

The definition of the set *SeCo* of semantic continuations is simple: We just take

$$SeCo = \mathcal{S}_{\mathrm{nc}},$$

and use X, Y to range over *SeCo* as well. A semantic continuation denotes the semantics of the statements to be executed after the one to which $\mathcal{D}[\![\cdot]\!]$ is applied. To be more precise, when $\mathcal{D}$ is applied to a (guarded) statement s and an environment γ, we get a function $\varphi : SeCo \rightarrow^{\mathrm{NDI}} \mathcal{S}_{\mathrm{nc}}$. The interpretation of this function is as follows: if $X \in SeCo = \mathcal{S}_{\mathrm{nc}}$ is the semantics of a statement, say s', to be executed after s, then the semantics of s and s' together is given by $\varphi(X)$ (this is illustrated very well by part (1)(b) of Definition 3.15 below). At this point continuations may seem a complicated way of doing a simple thing (concatenating sequences), but in later sections we shall see that the technique of continuations enables denotational semantics to do in a simple way things that otherwise require quite an effort.

There are two reasons to require the function φ to be non-distance-increasing: The technical reason is that we want Lemma 3.16 below to hold. The intuitive reason has to do with the fact that such a function φ will not have the opportunity to analyse its argument in detail and make decisive choices based on that analysis, but it will just concatenate the argument to the end of some set of streams, possibly (in later sections) interleaving it with yet another set of streams. This kind of operation will "shift" the argument "to the future", and due to the nature of the metric on $\mathcal{S}_{\mathrm{nc}}$, this means that the distance between $\varphi(X)$ and $\varphi(Y)$ will possibly be smaller than the distance between X and Y, but definitely not greater.

For the set of environments we use

$$\Gamma = StmV \rightarrow (SeCo \xrightarrow{\mathrm{NDI}} \mathcal{S}_{\mathrm{nc}}).$$

An environment gives a meaning to each statement variable. In more conventional languages, which use procedure declarations where we use the μ-construct, the meaning of such a set of declarations would be recorded in an environment γ, which is subsequently used to interpret the procedure calls in the statements after the declarations. Our recursive construct effectively combines a declaration and a call of a "procedure", named with a statement variable. Therefore the statement s within the recursive construct $\mu x[s]$ will be interpreted with respect to an environment different from the one used in interpreting the recursive construct, where the difference lies in the meaning assigned to the statement variable x (see equation (3.1) below).

We are now sufficiently prepared for the following definition.

3.15. Definition (*Denotational semantics for $\mathscr{S}_{\text{us}}$ and $\mathscr{L}_{\text{us}}$*). (1) Assume that $s \in \mathscr{S}_{\text{us}}$ is guarded. We define $\mathscr{D}[\![s]\!]$ by structural induction on s:

(a) $\mathscr{D}[\![a]\!](\gamma)(X) = a \cdot X$,

(b) $\mathscr{D}[s_1;s_2]\!](\gamma)(X) = \mathscr{D}[\![s_1]\!](\gamma)(\mathscr{D}[\![s_2]\!](\gamma)(X))$,

(c) $\mathscr{D}[\![s_1 \cup s_2]\!](\gamma)(X) = \mathscr{D}[\![s_1]\!](\gamma)(X) \cup \mathscr{D}[\![s_2]\!](\gamma)(X)$,

(d) $\mathscr{D}[\![x]\!](\gamma)(X) = \gamma(x)(X)$,

(e) $\mathscr{D}[\![\mu x[s]]\!](\gamma)(X) = \varphi_\infty(X)$ where φ_∞ is the unique fixed point of the operator $\Phi : (SeCo \to^{\text{NDI}} S_{\text{nc}}) \to (SeCo \to^{\text{NDI}} S_{\text{nc}})$ given by $\Phi(\varphi) = \mathscr{D}[\![s]\!](\gamma\{\varphi/x\})$. (We use the variant notation $\gamma\{\varphi/x\}$ introduced in Section 2.1.)

(2) For $t = s_1 \| \cdots \| s_n$, t guarded, we put

$$\mathscr{D}[\![t]\!] = \mathscr{D}[\![s_1]\!](\gamma)(\{\varepsilon\}) \| \cdots \| \mathscr{D}[\![s_n]\!](\gamma)(\{\varepsilon\})$$

where γ is arbitrary (and we assume the obvious associativity of "$\|$").

The definition in clause (1)(e) is justified by the following lemma.

3.16. Lemma. *If s is guarded and x does not occur exposed in s, then we have that the operator Φ defined by $\Phi = \lambda\varphi.\mathscr{D}[\![s]\!](\gamma\{\varphi/x\})$ is* contracting.

Proof. Induction on the complexity of s, using the condition on x. □

By Banach's theorem (Theorem 2.5), the operator Φ in Definition 3.15(1)(e) indeed has a unique fixed point φ_∞. In particular, for the meaning of $\mu x[s]$ we have the familiar fixed point relation (for each γ):

$$\varphi_\infty = \mathscr{D}[\![\mu x[s]]\!](\gamma) = \mathscr{D}[\![s]\!](\gamma\{\varphi_\infty/x\}). \tag{3.1}$$

Note furthermore that $\varphi_\infty = \lim_i \varphi_i$, where φ_0 can be chosen arbitrarily and the rest of the sequence is given by $\varphi_{i+1} = \mathscr{D}[\![s]\!](\gamma\{\varphi_i/x\})$.

3.4. Equivalence of operational and denotational semantics

After having defined both $\mathscr{O}$ and $\mathscr{D}$ for (guarded elements of) $\mathscr{S}_{\text{us}}$ and $\mathscr{L}_{\text{us}}$, we next discuss the relationship between the two semantics. We shall in fact establish that, for t guarded,

$$\mathscr{O}[\![t]\!] = \mathscr{D}[\![t]\!]. \tag{3.2}$$

We need some technical properties of $\mathscr{O}$ which will play a role in the inductive argument to prove (3.2). A very detailed treatment of variants of these results can be found in [16] (variants stemming from the fact that the latter deals with nested parallelism as well). Therefore, we state the results here without proof.

3.17. Lemma. (1) $\mathscr{O}[\![s;r]\!] = \mathscr{O}[\![s;E]\!] \cdot \mathscr{O}[\![r]\!]$.
(2) $\mathscr{O}[\![r_1, r_2]\!] = \mathscr{O}[\![r_1]\!] \| \mathscr{O}[\![r_2]\!]$.

For the statement of the next theorem we need some further notation: Consider a recursive construct $\mu x[s]$. Let Ω be a new elementary action, i.e., $\Omega \notin A$. (This is the only place where we find it convenient to distinguish a syntactic elementary action (Ω) from the corresponding semantic one ($\perp$).) Ω will play a role only in connection with Theorem 3.18 below. We first introduce a corresponding axiom (extending the list of transition axioms in Definition 3.4):

$$\langle \ldots, \Omega; r, \ldots, w\rangle \to w. \qquad \textbf{Undef}$$

(Recall that $w \in A^* \times \{\perp\}$. Thus, **Undef** is an axiom which terminates the computation with an unfinished stream.) Moreover, for each $n \geqslant 0$, s, and x, we introduce the notation $s_x^{(n)}$ given by

$$s_x^0 = \Omega, \qquad s_x^{(n+1)} = s[s_x^{(n)}/x].$$

The following theorem is proved in [16].

3.18. Theorem. *Assume that $\mu x[s]$ is closed and guarded. Then we have*

$$\mathcal{O}[\![\mu x[s];r]\!] = \lim_n \mathcal{O}[\![s_x^{(n)};r]\!].$$

Proof. See the argument in [16], which involves an elaborate development of auxiliary tools. □

Theorem 3.18 is in fact crucial for the proof of (3.2). We shall prove (3.2) in a way that anticipates the strategy followed in the next section where we deal with $\mathcal{L}_{\mathrm{ud}}$. Our reason for doing this is our wish to pinpoint the places where the proof of the dynamic case is essentially more involved than that of the static case.

In order to prove (3.2), we first prove a more general result, and then obtain (3.2) as a direct corollary.

3.19. Theorem. *Let s be guarded but not necessarily closed, and let the set of free statement variables of s be contained in $\{x_1, \ldots, x_m\}$, $m \geqslant 0$. Let $s_1, \ldots, s_m$ be closed and guarded statements, let $\tilde{s} = s[s_i/x_i]_{i=1}^m$, and let, for any r, $\mathcal{O}[\![r]\!]$ be short for $\lambda X.(\mathcal{O}[\![r]\!] \cdot X)$. Let furthermore*

$$\varphi_i = \mathcal{O}[\![s_i;E]\!]$$

for $i = 1, \ldots, m$, and let $\tilde{\gamma} = \gamma\{\varphi_i/x_i\}_{i=1}^m$. Then we have

$$\mathcal{O}[\![\tilde{s};E]\!] = \mathcal{D}[\![s]\!](\tilde{\gamma}).$$

Proof. Induction on the complexity of s. We treat three representative cases:

Case 1: $s = x_i$. Then $\mathcal{O}[\![\tilde{s};E]\!] = \mathcal{O}[\![s_i;E]\!] = \varphi_i = \mathcal{D}[\![x_i]\!](\tilde{\gamma})$.

Case 2: $s = s';s''$. Now the free statement variables of s' and s'' are also among $\{x_1, \ldots, x_m\}$. We can write $\tilde{s}' = s'[s_i/x_i]_{i=1}^m$ and similarly for s''. Then we get

$$\begin{aligned} \mathcal{O}[\![\tilde{s};E]\!] &= \mathcal{O}[\![(\tilde{s}';\tilde{s}'');E]\!] \\ &= \mathcal{O}[\![\tilde{s}';(\tilde{s}'';E)]\!] \qquad \text{(Lemma 3.7)} \end{aligned}$$

$$= \lambda X.\mathcal{O}[\![\tilde{s}';(\tilde{s}'';E)]\!] \cdot X$$
$$= \lambda X.(\mathcal{O}[\![s';E]\!] \cdot (\mathcal{O}[\![\tilde{s}'';E]\!] \cdot X)) \quad \text{(Lemma 3.17 and associativity of “·”)}$$
$$= \lambda X.(\mathcal{O}[\![\tilde{s}';E]\!](\mathcal{O}[\![\tilde{s}'';E]\!](X)))$$
$$= \lambda X.(\mathcal{D}[\![s']\!](\tilde{\gamma})(\mathcal{D}[\![s'']\!](\tilde{\gamma})(X))) \quad \text{(twice the induction hypothesis)}$$
$$= \mathcal{D}[\![s';s'']\!](\tilde{\gamma}).$$

Case 3: $s = \mu y[s']$. Let us first remark that from the conditions on s and $s_1, \ldots, s_m$ it follows that $\tilde{s}$ is guarded. We define $\tilde{s}' = s'[s_i/x_i]_{i=1}^m$ (note that y may still be free in $\tilde{s}'$). Now we have on the one hand

$$\mathcal{O}[\![\tilde{s};E]\!] = \lambda X.(\mathcal{O}[\![\tilde{s};E]\!] \cdot X)$$
$$= \lambda X.\lim_n (\mathcal{O}[\![\tilde{s}_y'^{(n)};E]\!] \cdot X) \quad \text{(Theorem 3.18 and continuity of “·”)}$$
$$= \lim_n (\mathcal{O}[\![\tilde{s}_y'^{(n)};E]\!]).$$

On the other hand, we have $\mathcal{D}[\![s]\!](\tilde{\gamma}) = \lim_n \psi_n$, where ψ_0 can be chosen freely and $\psi_{n+1} = \mathcal{D}[\![s']\!](\tilde{\gamma}\{\psi_n/y\})$. Our choice for ψ_0 will be $\psi_0 = \lambda X.\{\perp\}$. We prove, by induction on n, that

$$\mathcal{O}[\![\tilde{s}_y'^{(n)};E]\!] = \psi_n. \tag{3.3}$$

The case $n = 0$ is clear. Now assume (3.3) as induction hypothesis. Then

$$\mathcal{O}[\![\tilde{s}_y'^{(n+1)};E]\!] = \mathcal{O}[\![s'[s_i/x_i]_{i=1}^m[\tilde{s}_y'^{(n)}/y];E]\!]$$
$$= \mathcal{D}[\![s']\!](\gamma\{\varphi_i/x_i\}_{i=1}^m\{\psi_n/y\}) = \mathcal{D}[\![s']\!](\tilde{\gamma}\{\psi_n/y\}) = \psi_{n+1}.$$

Here we have used the main induction hypothesis with s' replacing s, $m+1$ replacing m, and $s_1, \ldots, s_m, \tilde{s}_y'^{(n)}$ replacing $s_1, \ldots, s_m$. In order for the main induction hypothesis to apply we have to establish that $\mathcal{O}[\![\tilde{s}_y'^{(n)};E]\!] = \psi_n$, which is nothing but our nested induction hypothesis (3.3).

Now that we have proved (3.3) for all n, it is evident that $\mathcal{O}[\![\tilde{s};E]\!] = \mathcal{D}[\![s]\!](\tilde{\gamma})$, which proves the most difficult part of the theorem. □

3.20. Corollary. *For guarded t we have $\mathcal{O}[\![t]\!] = \mathcal{D}[\![t]\!]$.*

Proof. For any closed and guarded s, and any γ, we have, by the previous theorem, that $\mathcal{O}[\![s;E]\!] = \mathcal{D}[\![s]\!](\gamma)$. Hence, $\mathcal{O}[\![s;E]\!] = \mathcal{O}[\![s;E]\!](\{\varepsilon\}) = \mathcal{D}[\![s]\!](\gamma)(\{\varepsilon\})$. If $t = s_1 \| \cdots \| s_n$, we therefore obtain

$$\mathcal{O}[\![t]\!] = \mathcal{O}[\![s_1;E, \ldots, s_n;E]\!] = \mathcal{O}[\![s_1;E]\!] \| \cdots \| \mathcal{O}[\![s_n;E]\!]$$
$$= \mathcal{D}[\![s_1]\!](\gamma)(\{\varepsilon\}) \| \cdots \| \mathcal{D}[\![s_n]\!](\gamma)(\{\varepsilon\}) = \mathcal{D}[\![t]\!]. \quad \square$$

We conclude this section with a remark on possible other models for $\mathcal{L}_{\text{us}}$. Besides the operational and metric denotational (linear time) models for $\mathcal{L}_{\text{us}}$, we have also developed several other models which have been described elsewhere:

(1) A denotational semantics based on a cpo structure on (certain) sets of streams equipped with the Smyth order [12, 14, 33, 34].

(2) A denotational semantics based on a cpo structure on (certain) sets of so-called finite observations equipped with the order of reverse set inclusion [12, 14].

(3) A branching time denotational semantics based on a process domain of the kind described in Section 2.3 [11].

The equivalence of the models in (1) and (2) has been established in [14], the equivalence of the model in (1) and the denotational metric model is proved in [13], and the relationship between the branching time model and (any of) the linear time models is settled in [11].

4. A uniform and dynamic language

We now turn our attention to a language with *process creation*. In this section we study the uniform version of this phenomenon as couched in the language $\mathscr{L}_{\text{ud}}$. In Section 5 we shall investigate a nonuniform generalization.

A substantial part of the semantic theory for $\mathscr{L}_{\text{us}}$ can be carried over to the present case. Thus, we can be much shorter in our definitions. The main equivalence result also closely follows the approach from Section 3, but for one important new problem which requires nontrivial additional analysis.

4.1. Syntax and intuitive explanation

We start with the following definition.

4.1. Definition (*Syntax for statements and programs*). (1) Let s range over the set $\mathscr{S}_{\text{ud}}$ of (uniform and dynamic) *statements*:

$$s ::= a \mid x \mid s_1;s_2 \mid s_1 \cup s_2 \mid \mu x[s'] \mid \mathbf{new}(s').$$

(2) Let t range over the set $\mathscr{L}_{\text{ud}}$ of (uniform and dynamic) *programs*:

$$t ::= s$$

Here we require again that s is closed. Thus, a program in $\mathscr{L}_{\text{ud}}$ is simply a closed statement from $\mathscr{S}_{\text{ud}}$.

The intuitive operational semantics for t or s may be described in terms of a dynamically growing number of processes which execute statements in parallel in the following manner:

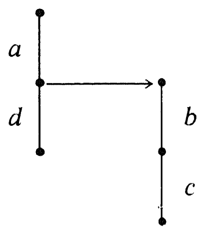
(1) Set an auxiliary variable i to 1, and set s_1 to s, the program to be executed. A process, numbered 1, is created to execute this s_1.

(2) Processes 1 to i are executed in parallel. Process j executes s_j $(1 \leqslant j \leqslant i)$ in the usual way (see Section 3) if s_j begins with an elementary action, sequential composition, choice, or a recursive construct. For example, if s_j begins with an elementary action a, then this a is appended to the output word, and s_j is set to its (syntactic) continuation (the part after this atomic action).

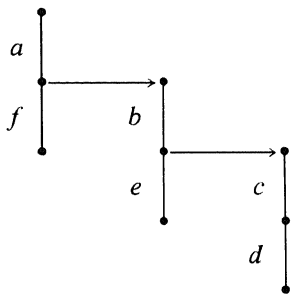
(3) If some process j $(1 \leqslant j \leqslant i)$ has to execute a statement of the form **new**(s'), then the following happens: The variable i is set to $i+1$, then s_i is set to s', and a new process, with number i, is created to execute s_i. Process j will continue to execute the part after the **new**-statement (s_j is set to its continuation). Go back to step (2).

(4) Execution terminates when there is no process left with a nonempty continuation.

Examples. (1) The statement a;**new**$(b;c);d$ determines the execution as suggested by the following picture (where the arrow denotes creation of a new process):



(2) The statement a;**new**$(b$;**new**$(c;d);e);f$ determines the execution as suggested by the diagram:



4.2. Operational and denotational semantics

The above intuitive explanation would clearly benefit from a more formal description, and this will be the main content of the present section.

We first develop the operational semantics for $\mathcal{L}_{\mathrm{ud}}$. We profit from the preparatory work in Section 3, and assume the general framework as described there. Also, configurations $\langle \rho, w \rangle$ or simply w' (with $w \in A^* \times \{\perp\}$, $w' \in A^*$) are as before, except that the statements s in such a parallel syntactic continuation ρ (see Definition 3.3) should now belong to $\mathcal{S}_{\mathrm{ud}}$ instead of $\mathcal{S}_{\mathrm{us}}$. The transition relation "$\rightarrow$" is now defined as the smallest relation satisfying the axioms in the following definition.

4.2. Definition (*Transition system for* $\mathcal{L}_{\mathrm{ud}}$). The transition system $\mathcal{T}_{\mathrm{ud}}$ for $\mathcal{L}_{\mathrm{ud}}$ consist of all the axioms of Definition 3.4 (i.e., of all of $\mathcal{T}_{\text{us}}$), and in addition the axiom

$$\langle \ldots, \mathbf{new}(s);r, \ldots, w\rangle \to \langle \ldots, r, \ldots, s;E, w\rangle. \qquad \textbf{New}$$

Here on the left-hand side we have a parallel syntactic continuation ρ with, say, $n \geqslant 1$ components and $\mathbf{new}(s);r$ as the ith component (for some i, $1 \leqslant i \leqslant n$). On the right-hand side we have the parallel syntactic continuation ρ' with $n+1$ components, r as the ith component and $s;E$ as the $(n+1)$st component (and no changes with respect to ρ in the remaining components).

The definition of $\mathcal{O}[\![\rho]\!]$ is as before, but now with respect to transition system $\mathcal{T}_{\text{ud}}$. Also, since each $t \in \mathcal{L}_{\text{ud}}$ equals some $s \in \mathcal{S}_{\text{ud}}$, we simply put, for $t = s$, $\mathcal{O}[\![t]\!] = \mathcal{O}[\![s;E]\!]$.

Example. Take $t = a;\mathbf{new}(b;\mathbf{new}(c);e);f$. Then $\mathcal{O}[\![t]\!] = \{afbce, abfce, abcfe, abcef, afbec, abfec, abefc, abecf\}$.

The elementary properties of $\mathcal{O}$ listed in Lemma 3.7 remain valid. In addition, we have the following lemma.

4.3. Lemma. $\mathcal{O}[\![\mathbf{new}(s);r]\!] = \mathcal{O}[\![r, s;E]\!]$.

Proof. Clear from the definitions. □

We proceed with the definitions for the denotational semantics for $\mathcal{S}_{\text{ud}}$ and $\mathcal{L}_{\text{ud}}$. A complication which arises is that the notion of a statement being guarded has to be refined. A typical case concerns a recursive construct such as $\mu x[\mathbf{new}(a);x]$, where the elementary action a does not fulfil the duties of a guard: this construct may choose to start execution with the recursive call x. The precise definition of guardedness requires an amended definition of "x is exposed in s", and this involves, in turn, a notion of generalized **new**-statement.

4.4. Definition. (1) A *generalized* ***new*** *statement* g is defined by

$$g \ ::= \ \mathbf{new}(s) \mid g_1;g_2 \mid g' \cup s \mid s \cup g' \mid \mu x[g']$$

(2) When a statement variable x occurs *exposed* in a statement $s \in \mathcal{S}_{\text{ud}}$ is defined inductively as follows:

(a) x occurs exposed in x;

(b) if x occurs exposed in s, then x occurs exposed in $s;s'$, $s \cup s'$, $s' \cup s$, $\mu y[s]$ (if $y \neq x$), $\mathbf{new}(s)$, and in $g;s$.

(3) A statement $s \in \mathcal{S}_{\text{ud}}$ is called *guarded* if, for all its recursive substatements of the form $\mu x[s']$, s' contains no exposed occurrences of x.

We shall now give a denotational semantics for $\mathcal{L}_{\text{ud}}$ by defining

$$\mathcal{D}[\![\cdot]\!] : \mathcal{S}^{\text{g}}_{\text{ud}} \to (\Gamma \to (SeCo \xrightarrow{\text{NDI}} \boldsymbol{S}_{\text{nc}})) \quad \text{and} \quad \mathcal{D}[\![\cdot]\!] : \mathcal{L}^{\text{g}}_{\text{ud}} \to \boldsymbol{S}_{\text{nc}},$$

where we use Γ, *SeCo*, and S_{nc} as in Section 3.3. (Analogously to Section 3.3, $\mathcal{S}^{\mathrm{g}}_{\mathrm{ud}}$ denotes the set of guarded statements, and $\mathcal{L}^{\mathrm{g}}_{\mathrm{ud}}$ the set of guarded programs.)

4.5. Definition. (1) For guarded $s \in \mathcal{S}_{\mathrm{ud}}$, s not of the form $\mathbf{new}(s')$, we take over the clauses from Definition 3.15.

(2) For guarded s of the form $\mathbf{new}(s')$ we put

$$\mathcal{D}[\![\mathbf{new}(s')]\!](\gamma)(X) = \mathcal{D}[\![s']\!](\gamma)(\{\varepsilon\}) \parallel X.$$

(3) For guarded $t \in \mathcal{L}_{\mathrm{ud}}$, $t = s$, we put $\mathcal{D}[\![t]\!] = \mathcal{D}[\![s]\!](\gamma)(\{\varepsilon\})$, where γ is arbitrary.

We see that the meaning of a new-construct $\mathbf{new}(s')$ in a situation that X remains to be done (i.e., with a semantic continuation X) is given by the result of putting X in parallel with the meaning of s' where nothing remains to be done after it (continuation $\{\varepsilon\}$).

Remark. It has been proved that the expressive power of $\mathcal{L}_{\mathrm{ud}}$ is essentially greater than that of $\mathcal{L}_{\mathrm{us}}$, in the sense that for each $t \in \mathcal{L}_{\mathrm{us}}$ there is a $t' \in \mathcal{L}_{\mathrm{ud}}$ such that $\mathcal{O}[\![t]\!] = \mathcal{O}[\![t']\!]$ (indeed, take $t' = t$), but not the other way around. (IJ.J. Aalbersberg and P. America, personal communication.)

4.3. Equivalence of operational and denotational semantics

We now address the question as to whether, for guarded t, $\mathcal{O}[\![t]\!] = \mathcal{D}[\![t]\!]$. We follow the line of reasoning as in Section 3. First, we again have this lemma.

4.6. Lemma. (1) *For all* $r_1, r_2 \in SyCo$ *we have* $\mathcal{O}[\![r_1, r_2]\!] = \mathcal{O}[\![r_1]\!] \parallel \mathcal{O}[\![r_2]\!]$.

(2) *If* $\mu x[s]$ *is closed and guarded, then* $\mathcal{O}[\![\mu x[s];r]\!] = \lim_n \mathcal{O}[\![s_x^{(n)};r]\!]$.

Proof. See the sources given with Lemma 3.17 and Theorem 3.18. $\square$

The next step in the argument concerns the analogue of Lemma 3.17(1) (and, somewhat more hidden, the way in which $\mathcal{O}[\![\cdot]\!]$ is defined, cf. Theorem 3.19). Let us see whether we may expect that $\mathcal{O}[\![s;r]\!] = \mathcal{O}[\![s;E]\!] \cdot \mathcal{O}[\![r]\!]$. It is easy to see that this is not the case by taking, for example, $s = \mathbf{new}(a)$ and $r = b;E$. Then the left-hand side equals $\{ab, ba\}$ and the right-hand side equals $\{ab\}$. On the other hand, taking $s = a$, $r = b;E$, we see that neither is it true in general that $\mathcal{O}[\![s;r]\!] = \mathcal{O}[\![s;E]\!] \parallel \mathcal{O}[\![r]\!]$. What we need here (and in the definition of $\mathcal{O}[\![\cdot]\!]$) is an operator which, as it were, is able to decide dynamically whether the operation at hand is of a sequential or of a parallel character.

Having pinpointed the problem which distinguishes the situation in the current section from that in Section 3, we develop some additional tools and associated lemmas in such a way that eventually we shall be able to adopt the same style of argument for the main equivalence result as used in Section 3.

We shall introduce the *semantic operator* ":", which should clearly be distinguished from both "·" and "$\|$". The definition of ":" requires the introduction of an auxiliary elementary action, not belonging to $A \cup \{\perp\}$, and denoted by $\surd$. Its intuitive function is to mark the termination of a local process and (thus) to indicate where a continuation should start. We shall put $A' = A \cup \{\surd\}$, and introduce the extended stream set A^{est} as

$$A^{\mathrm{est}} = A^{\mathrm{st}} \cup \{w_1 \surd w_2 \mid w_1 \in A^*,\ w_2 \in A^{\mathrm{st}}\}.$$

We now define the operator ":" as follows.

4.7. Definition. We shall put $S'_{\mathrm{nc}} = \mathcal{P}_{\mathrm{nc}}(A^{\mathrm{est}})$ (recall that $S_{\mathrm{nc}} = \mathcal{P}_{\mathrm{nc}}(A^{\mathrm{st}})$).

(1) The operator ":": $A^{\mathrm{est}} \times A^{\mathrm{est}} \to S'_{\mathrm{nc}}$ is given by

$$w\!:\!w' = \begin{cases} w_1 \cdot (w_2 \,\|\, w') & \text{if } w = w_1 \surd w_2, \\ \{w\} & \text{otherwise.} \end{cases}$$

(Note that w' could again contain an occurrence of $\surd$, which will behave as an ordinary elementary action with respect to "$\|$".)

(2) For X, $Y \in S'_{\mathrm{nc}}$, X and Y with finite streams only, we put

$$X\!:Y = \bigcup\{u\!:\!v \mid u \in X,\ v \in Y\}.$$

(3) For arbitrary X, $Y \in S'_{\mathrm{nc}}$, we put

$$X\!:Y = \lim_n (X(n)\!:Y(n)).$$

An important technical lemma concerning the operator ":" is the following one.

4.8. Lemma. (1) ":" *is continuous as a mapping* $A^{\mathrm{est}} \times A^{\mathrm{est}} \to S'_{\mathrm{nc}}$ *and as a mapping* $S'_{\mathrm{nc}} \times S'_{\mathrm{nc}} \to S'_{\mathrm{nc}}$.

(2) *Restricting the domain of* ":" *to* $S'_{\mathrm{nc}} \times S_{\mathrm{nc}}$ *will restrict its range to* S_{nc}, *or in other words,* ":": $S'_{\mathrm{nc}} \times S_{\mathrm{nc}} \to S_{\mathrm{nc}}$.

(3) $(X\!:Y)\!:\!Z = X\!:(Y\!:\!Z)$, *for* $X, Y, Z \in S'_{\mathrm{nc}}$.

(4) $\{w\surd\}\!:\!X = wX$, *for* $w \in A^{\mathrm{st}}$, $X \in S'_{\mathrm{nc}}$.

(5) $(X \cup Y)\!:\!Z = (X\!:\!Z) \cup (Y\!:\!Z)$, *for* $X, Y, Z \in S'_{\mathrm{nc}}$.

(6) $(X \,\|\, Y)\!:\!Z = X \,\|\, (Y\!:\!Z)$, *for* $X \in S_{\mathrm{nc}}$, $Y, Z \in S'_{\mathrm{nc}}$.

Proof. We only prove part (3). Below, we shall prove that $(u\!:\!v)\!:\!w = u\!:\!(v\!:\!w)$ for $u, v, w \in A^{\mathrm{est}}$. Then we obtain, for X, Y, Z with finite streams only,

$$(X\!:Y)\!:\!Z = \bigcup_{u \in X:Y} \bigcup_{w \in Z} (u\!:\!w) = \bigcup_{u_1 \in X} \bigcup_{u_2 \in Y} \bigcup_{w \in Z} ((u_1\!:\!u_2)\!:\!w)$$

$$= \bigcup_{u_1 \in X} \bigcup_{u_2 \in Y} \bigcup_{w \in Z} (u_1\!:\!(u_2\!:\!w)) = X\!:\!(Y\!:\!Z).$$

For general X, Y, Z, we take the limit of $X(n)\!:Y(n)\!:\!Z(n)$.

We now prove that $(u{:}v){:}w = u{:}(v{:}w)$. If $u \in A^{\mathrm{st}}$ (so that u has no occurrence of $\surd$) then $(u{:}v){:}w = \{u\} = u{:}(v{:}w)$, and if $v \in A^{\mathrm{st}}$ then $(u{:}v){:}w = u{:}v = u{:}(v{:}w)$. Now suppose that $u = u_1 \surd u_2$ and $v = v_1 \surd v_2$. We prove two inclusions:

(1) $(u{:}v){:}w \subseteq u{:}(v{:}w)$. We have $u{:}v = u_1 \cdot (u_2 \| v)$, so $(u{:}v){:}w = \bigcup_{w' \in (u_2 \| v)} (u_1 w'){:}w$. Let $w' \in u_2 \| v$. We distinguish two subcases:

(a) $w' \in A^{\mathrm{st}}$. This is only possible (since $v = v_1 \surd v_2$) if $u_2 \in A^{\omega} \cup (A^* \times \{\perp\})$. Then $w' \in u_2 \| v_1$, so $w' \in u_2 \| (v_1 \cdot (v_2 \| w)) = u_2 \| (v{:}w)$, and therefore $(u_1 w'){:}w = \{u_1 w'\} \subseteq u{:}(v{:}w)$.

(b) $w' = w'_1 \surd w'_2$. Now there are u_{21}, u_{22} such that $u_2 = u_{21} u_{22}$, $w'_1 \in u_{21} \| v_1$, $w'_2 \in u_{22} \| v_2$. We obtain

$$\begin{aligned}(u_1 w'){:}w &= u_1 w'_1 (w'_2 \| w) \subseteq u_1 (u_{21} \| v_1)(u_{22} \| v_2 \| w)\\ &\subseteq u_1 (u_2 \| (v_1 (v_2 \| w))) = u{:}(v;w).\end{aligned}$$

(2) $u{:}(v{:}w) \subseteq (u{:}v){:}w$. We have $u{:}(v{:}w) = u_1 \cdot (u_2 \| (v{:}w)) = \bigcup_{u' \in v:w} u_1 \cdot (u_2 \| u') = \bigcup_{v' \in v_2 \| w} u_1 \cdot (u_2 \| (v_1 v'))$. Now let $v' \in v_2 \| w$ and $w' \in u_2 \| (v_1 v')$. There are u_{21}, u_{22}, w'_1, and w'_2 such that $w' = w'_1 w'_2$, $w'_1 \in u_{21} \| v_1$, $w'_2 \in u_{22} \| v'$. We have that

$$u_1 w'_1 \surd (u_{22} \| v_2) \subseteq u_1 \cdot (u_2 \| v_1 \surd v_2) = u{:}v. \tag{4.1}$$

(The inclusion holds since $u_2 \| v_1 \surd v_2$ contains the set $(u_{21} \| v_1) \surd (u_{22} \| v_2)$, which in turn contains $w'_1 \surd (u_{22} \| v_2)$.) We conclude that

$$u_1 w' = u_1 w'_1 w'_2 \in u_1 w'_1 (u_{22} \| v_2 \| w) \subseteq (u{:}v){:}w,$$

where the last inclusion follows from (4.1) by postfixing both sides with "$:w$". □

We next show how the new operator ":" solves the problems described after Lemma 4.6. First we extend—for the remainder of this section—the definition of *SyCo* (cf. Definition 3.3), and now put

$$r \ ::= \ E \mid \surd \mid s;r'$$

We emphasize that the "elementary action" $\surd$ occurs only in syntactic continuations; the syntax for statements $s \in \mathcal{S}_{\mathrm{ud}}$ is not modified. Before we can state and prove the equivalent of Lemma 3.17(1), we discuss the induced amendment of the transition system $\mathcal{T}_{\mathrm{ud}}$. Firstly, all axioms of $\mathcal{T}_{\mathrm{ud}}$ now refer to r (and ρ) which may involve $\surd$. Secondly, we extend $\mathcal{T}_{\mathrm{ud}}$ with an axiom catering for $\surd$. In the present context, we need this axiom only in a restricted version:

$$\langle \ldots, \surd, \ldots, w\perp \rangle \to \langle \ldots, E, \ldots, w\surd\perp \rangle \qquad \textbf{Elem}'$$

where $w \in A^*$ and none of the continuations appearing at the dots (...) involves $\surd$. In other words, we restrict attention to parallel syntactic continuations ρ which involve at most one constituent syntactic continuation r ending in $\surd$. This is no real restriction since that property applies to all configurations in transition sequences which interest us: It holds trivially for ρ containing only one component, and it is preserved by applications of the axiom **New**, which creates new components.

We can now state the following lemma, which applies the technique of induction loading to prove Corollary 4.10.

4.9. Lemma. *Let $s \in \mathcal{S}_{\mathrm{ud}}$ (not necessarily closed) and suppose that all the free variables in s are in $\{x_1, \ldots, x_k\}$. Now let $s_1, \ldots, s_k$ be closed and guarded and define $\tilde{s} = s[s_i/x_i]_{i=1}^{k}$. Suppose further that for $i = 1, \ldots, k$ and for any r we have*

$$\mathcal{O}[\![s_i; r]\!] = \mathcal{O}[\![s_i; \surd]\!] : \mathcal{O}[\![r]\!]$$

and that $\tilde{s}$ is guarded. Then we have for any r

$$\mathcal{O}[\![\tilde{s}; r]\!] = \mathcal{O}[\![\tilde{s}; \surd]\!] : \mathcal{O}[\![r]\!].$$

Proof. Induction on the complexity of s. We give full details of the proof, in order to exhibit its dependence on Lemma 4.8.

(1) If $s = a$, then $\tilde{s} = a$, so we get

$$\begin{aligned} \mathcal{O}[\![\tilde{s}; r]\!] &= \mathcal{O}[\![a; r]\!] = a \cdot \mathcal{O}[\![r]\!] && \text{(Lemma 3.7)} \\ &= \{a\surd\} : \mathcal{O}[\![r]\!] && \text{(Lemma 4.8(4))} \\ &= \mathcal{O}[\![a; \surd]\!] : \mathcal{O}[\![r]\!] = \mathcal{O}[\![\tilde{s}; \surd]\!] : \mathcal{O}[\![r]\!]. \end{aligned}$$

(2) If $s = x_i$, then $\tilde{s} = s_i$ and the property follows from the assumption about s_i.

(3) If $s = s'; s''$, then we get in an obvious way $\tilde{s} = \tilde{s}'; \tilde{s}''$, so

$$\begin{aligned} \mathcal{O}[\![\tilde{s}; r]\!] &= \mathcal{O}[\![(\tilde{s}'; \tilde{s}''); r]\!] \\ &= \mathcal{O}[\![\tilde{s}'; (\tilde{s}''; r)]\!] && \text{(Lemma 3.7)} \\ &= \mathcal{O}[\![\tilde{s}'; \surd]\!] : \mathcal{O}[\![\tilde{s}''; r]\!] && \text{(ind. hyp. for } s'\text{)} \\ &= \mathcal{O}[\![\tilde{s}'; \surd]\!] : (\mathcal{O}[\![\tilde{s}''; \surd]\!] : \mathcal{O}[\![r]\!]) && \text{(ind. hyp. for } s''\text{)} \\ &= (\mathcal{O}[\![\tilde{s}'; \surd]\!] : \mathcal{O}[\![\tilde{s}''; \surd]\!]) : \mathcal{O}[\![r]\!] && \text{(Lemma 4.8(3))} \\ &= \mathcal{O}[\![\tilde{s}'; (\tilde{s}''; \surd)]\!] : \mathcal{O}[\![r]\!] && \text{(ind. hyp. for } s'\text{)} \\ &= \mathcal{O}[\![(\tilde{s}'; \tilde{s}''); \surd]\!] : \mathcal{O}[\![r]\!] \\ &= \mathcal{O}[\![\tilde{s}; \surd]\!] : \mathcal{O}[\![r]\!]. \end{aligned}$$

(4) If $s = s' \cup s''$, then, again, $\tilde{s} = \tilde{s}' \cup \tilde{s}''$ and we get

$$\begin{aligned} \mathcal{O}[\![\tilde{s}; r]\!] &= \mathcal{O}[\![(\tilde{s}' \cup \tilde{s}''); r]\!] \\ &= \mathcal{O}[\![\tilde{s}'; r]\!] \cup \mathcal{O}[\![\tilde{s}''; r]\!] && \text{(Lemma 3.7)} \\ &= (\mathcal{O}[\![\tilde{s}'; \surd]\!] : \mathcal{O}[\![r]\!]) \cup (\mathcal{O}[\![\tilde{s}''; \surd]\!] : \mathcal{O}[\![r]\!]) && \text{(ind. hyp. for } s', s''\text{)} \\ &= (\mathcal{O}[\![\tilde{s}'; \surd]\!] \cup \mathcal{O}[\![\tilde{s}''; \surd]\!]) : \mathcal{O}[\![r]\!] && \text{(Lemma 4.8(5)} \end{aligned}$$

$$= \mathcal{O}[\![(\tilde{s}' \cup \tilde{s}'');\surd]\!]:\mathcal{O}[\![r]\!]$$
$$= \mathcal{O}[\![\tilde{s};\surd]\!]:\mathcal{O}[\![r]\!].$$

(5) If $s = \mathbf{new}(s')$, we get $\tilde{s} = \mathbf{new}(\tilde{s}')$ and then

$$\begin{aligned}
\mathcal{O}[\![\tilde{s};\, r]\!] &= \mathcal{O}[\![\mathbf{new}(\tilde{s}');r]\!] \\
&= \mathcal{O}[\![\tilde{s}';E]\!]\|\mathcal{O}[\![r]\!] && \text{(Lemmas 4.3 and 4.6(1))} \\
&= (\mathcal{O}[\![\tilde{s}';E]\!]\|\{\surd\}):\mathcal{O}[\![r]\!] && (*) \\
&= (\mathcal{O}[\![\tilde{s}';E]\!]\|\mathcal{O}[\![\surd]\!]):\mathcal{O}[\![r]\!] \\
&= \mathcal{O}[\![\mathbf{new}(\tilde{s}');\surd]\!]:\mathcal{O}[\![r]\!] \\
&= \mathcal{O}[\![\tilde{s};\surd]\!]:\mathcal{O}[\![r]\!].
\end{aligned}$$

Here, at the place marked (*), we have used $(X\|\{\surd\})$: $Z = X\|Z$ if $X \in \boldsymbol{S}_{\text{nc}}$, $Z \in \boldsymbol{S}'_{\text{nc}}$; this is a special case of Lemma 4.8(6) together with Lemma 4.8(4).

(6) Let $s = \mu x[s']$. Suppose (without loss of generality) that $x \notin \{x_1, \ldots, x_k\}$. Put $\tilde{s}' = s'[s_i/x_i]_{i=1}^{k}$, so that $\tilde{s} = \mu x[\tilde{s}']$. Then we have by Lemma 4.6(2)

$$\mathcal{O}[\![\tilde{s};r]\!] = \mathcal{O}[\![\mu x[\tilde{s}'];r]\!] = \lim_n \mathcal{O}[\![\tilde{s}_x'^{(n)};\, r]\!].$$

Now we shall prove in a minute that

$$\mathcal{O}[\![\tilde{s}_x'^{(n)};\, r']\!] = \mathcal{O}[\![\tilde{s}_x'^{(n)};\surd]\!]:\mathcal{O}[\![r']\!] \tag{4.2}$$

for all n and for all r'. Once we have proved this, we can calculate

$$\begin{aligned}
\mathcal{O}[\![\tilde{s};r]\!] &= \lim_n \mathcal{O}[\![\tilde{s}_x'^{(n)};r]\!] && \text{(Lemma 4.6(2))} \\
&= \lim_n(\mathcal{O}[\![\tilde{s}_x'^{(n)};\surd]\!]:\mathcal{O}[\![r]\!]) && \text{(property (4.2))} \\
&= (\lim_n \mathcal{O}[\![\tilde{s}_x'^{(n)};\surd]\!]):\mathcal{O}[\![r]\!] && \text{(continuity of ":")} \\
&= \mathcal{O}[\![\tilde{s};\surd]\!]:\mathcal{O}[\![r]\!] && \text{(Lemma 4.6(2)),}
\end{aligned}$$

which is what we wanted.

We still have to do the proof of property (4.2), which runs by an induction on n (nested within our original induction on the complexity of s). For the case $n = 0$, we have $\tilde{s}_x'^{(0)} = \Omega$, so $\mathcal{O}[\![\tilde{s}_x'^{(0)};r']\!] = \bot = \bot:\mathcal{O}[\![r']\!] = \mathcal{O}[\![\tilde{s}_x'^{(0)};\surd]\!]:\mathcal{O}[\![r']\!]$.

For the induction step we assume that property (4.2) holds for a certain value of n. Then we can apply the main induction hypothesis for $k+1$ to s' with $x_1, \ldots, x_{k+1} = x_1, \ldots, x_k, x$ and $s_1, \ldots, s_{k+1} = s_1, \ldots, s_k, \tilde{s}_x'^{(n)}$ in order to get

$$\begin{aligned}
\mathcal{O}[\![\tilde{s}_x'^{(n+1)};r']\!] &= \mathcal{O}[\![\tilde{s}'[\tilde{s}_x'^{(n)}/x];r']\!] \\
&= \mathcal{O}[\![s'[s_i/x_i]_{i=1}^{k+1};r']\!] \\
&= \mathcal{O}[\![s'[s_i/x_i]_{i=1}^{k+1};\surd]\!]:\mathcal{O}[\![r']\!] \\
&= \mathcal{O}[\![\tilde{s}_x'^{(n+1)};\surd]\!]:\mathcal{O}[\![r']\!]. \qquad \Box
\end{aligned}$$

4.10. Corollary. *For closed and guarded* s, $\mathcal{O}[\![s;r]\!] = \mathcal{O}[\![s;\surd]\!]:\mathcal{O}[\![r]\!]$.

We are, at last, sufficiently prepared for the main theorem of this section.

4.11. Theorem. *Let* $s \in \mathcal{S}_{\mathrm{ud}}$, *not necessarily closed, and let the set of free statement variables of* s *be contained in* $\{x_1, \ldots, x_m\}$, $m \geq 0$. *Let* $s_1, \ldots, s_m$ *be closed and guarded statements, let* $\tilde{s} = s[s_i/x_i]_{i=1}^m$, *and define* $\mathcal{O}[\![r]\!]$ *by*

$$\mathcal{O}[\![E]\!] = \mathcal{O}[\![\surd]\!] = \lambda X.X, \qquad \mathcal{O}[\![s';r]\!] = \lambda X.(\mathcal{O}[\![s';\surd]\!]:\mathcal{O}[\![r]\!](X)).$$

Let furthermore $\varphi_i = \mathcal{O}[\![s_i;E]\!]$ *for* $i = 1, \ldots, m$, *and let* $\tilde{\gamma} = \gamma\{\varphi_i/x_i\}_{i=1}^m$. *Now if* $\tilde{s}$ *is also guarded, we have*

$$\mathcal{O}[\![\tilde{s};E]\!] = \mathcal{D}[\![s]\!](\tilde{\gamma}).$$

Proof. Very similar to that of Theorem 3.19. We shall prove two cases of old statements plus the case of the new statement.

Case 1: $s = s';s''$

$$\begin{aligned}
&\mathcal{O}[\![\tilde{s};E]\!] \\
&\quad = \mathcal{O}[\![(\tilde{s}';\tilde{s}'');E]\!] \\
&\quad = \lambda X.(\mathcal{O}[\![(\tilde{s}';\tilde{s}'');\surd]\!]:\mathcal{O}[\![E]\!](X)) \\
&\quad = \lambda X.(\mathcal{O}[\![\tilde{s}';\,(\tilde{s}'';\surd)]\!]:X) && \text{(Lemma 3.7)} \\
&\quad = \lambda X.(\mathcal{O}[\![\tilde{s}';\surd]\!]:(\mathcal{O}[\![\tilde{s}'';\surd]\!]:X)) && \text{(Corollary 4.10 and Lemma 4.8(3))} \\
&\quad = \lambda X.\mathcal{O}[\![\tilde{s}';E]\!](\mathcal{O}[\![\tilde{s}'';E]\!](X)) \\
&\quad = \lambda X.\mathcal{D}[\![s']\!](\tilde{\gamma})(\mathcal{D}[\![s'']\!](\tilde{\gamma})(X)) && \text{(ind. hyp. for } s' \text{ and } s'') \\
&\quad = \mathcal{D}[\![s';s'']\!](\tilde{\gamma}) = \mathcal{D}[\![s]\!](\tilde{\gamma}).
\end{aligned}$$

Case 2: $s = \mu y[s']$. As in Theorem 3.19, let us define $\tilde{s}' = s'[s_i/x_i]_{i=1}^m$ and calculate

$$\mathcal{O}[\![\tilde{s};E]\!] = \lambda X.(\mathcal{O}[\![\tilde{s};\surd]\!]:X) = \lambda X.\lim_n(\mathcal{O}[\![\tilde{s}'^{(n)}_y;\surd]\!]:X) = \lim_n \mathcal{O}[\![\tilde{s}'^{(n)}_y;E]\!].$$

Here we have used Lemma 4.6(2) and the continuity of ":". From this point on the argument follows exactly the same lines as in Theorem 3.19.

Case 3: $s = \mathbf{new}(s')$.

$$\begin{aligned}
&\mathcal{O}[\![\widetilde{\mathbf{new}(s')};E]\!] \\
&\quad = \lambda X.(\mathcal{O}[\![\surd,\tilde{s}';E]\!]:X) \\
&\quad = \lambda X.((\{\surd\}\|\mathcal{O}[\![\tilde{s}';E]\!]):X) \\
&\quad = \lambda X.(\mathcal{O}[\![\tilde{s}';E]\!]\|X) && \text{(Lemma 4.8, parts (6) and (4))} \\
&\quad = \lambda X.((\mathcal{O}[\![\tilde{s}';\surd]\!]:\mathcal{O}[\![E]\!])\|X) && \text{(Corollary 4.10)} \\
&\quad = \lambda X.(\mathcal{O}[\![\tilde{s}';E]\!](\{\varepsilon\})\|X) && \text{(Lemma 3.7 and def. of } \mathcal{O}) \\
&\quad = \lambda X.(\mathcal{D}[\![s']\!](\tilde{\gamma})(\{\varepsilon\})\|X) && \text{(induction hypothesis)} \\
&\quad = \mathcal{D}[\![\mathbf{new}(s')]\!](\tilde{\gamma}) && \text{(Definition 4.5).} \quad \square
\end{aligned}$$

4.12. Corollary. *For guarded $t \in \mathscr{L}_{\mathrm{ud}}$ we have $\mathcal{O}[\![t]\!] = \mathcal{D}[\![t]\!]$.*

Proof. Clear from Theorem 4.11. $\square$

We have thus completed the semantic analysis of $\mathscr{L}_{\mathrm{ud}}$, and are now ready for the generalization to the nonuniform case.

5. A nonuniform and static language

This section is devoted to the semantic definitions for a nonuniform and static language. The elementary actions are now interpreted, viz. as assignments and communication actions. However, for the moment we return to a static framework, and leave the treatment of the dynamic case to the next section.

5.1. Syntax

The nonuniform framework involves the introduction of three new syntactic classes:

- The set *IndV* of individual variables, with typical elements x, y. For *IndV* we take an infinite alphabet of variable names.
- The set *Exp* of expressions, with typical element e.
- The set *Test* of conditions, with typical element b.

We shall return to the syntax for expressions and conditions in a moment. Note that we have changed the notation with respect to Sections 3 and 4 in that we now use x, y for individual rather than statement variables. For the latter purpose we here use variables v ranging over *StmV*. (The nonuniform framework has no streams, so we can freely use the letters u, v, w.)

In the static case, a program will again be composed of n components $s_1, \ldots, s_n$. Contrary to the uniform case, we are also interested in the *identity* of, in general, the ith statement (or *process*, in a terminology used, e.g., in CSP [31, 32]), and we introduce for this purpose the set $I = \{1, 2, \ldots\}$ of indices, with i, j, k, l ranging over I. Typically, indices i, j will be used in communication statements of the form $i?x$ or $j!e$, denoting communication of two sorts: The first occurs, in general, in some process k and requires a value for the variable x from process i. The second occurs, say, in a process l and sends the current value α of the expression e to process j. In the case that $k = j$ and $l = i$ and, moreover, the communications *synchronize* in the usual sense, then the "handshake" communication can indeed take place, and the variable x takes the value α. Once more, this informal description requires formal definition, to be elaborated in the sequel.

The last syntactic set we need to introduce is that of (individual) *constants*. We shall not bother to make a distinction between syntactic constants and semantic (basic) values, and use the set V, with typical elements α, β, for both purposes.

We now define the syntax for $\mathscr{S}_{\mathrm{nus}}$ and $\mathscr{L}_{\mathrm{nus}}$ (and for *Exp*).

5.1. Definition. (1) Let e range over the set *Exp* of expressions:

$$e ::= x \mid \alpha \mid e_1 \mathbf{op}\, e_2 \mid \mathbf{op}\, e'$$

(Here **op** stands for an arbitrary binary or unary operator. We prefer not to take the trouble to introduce general n-ary function symbols into our language.)

(2) We do not specify a syntax for the elements b of *Test*. We only require that their evaluation terminates and takes place without complications such as side-effects.

(3) Let s range over the set $\mathcal{S}_{\text{nus}}$ of nonuniform and static statements:

$$s ::= x := e \mid s_1 ; s_2 \mid v \mid \mu v[s'] \mid \textbf{if } b \textbf{ then } s_1 \textbf{ else } s_2 \textbf{ fi} \mid i?x \mid i!e$$

(4) Let t range over the set $\mathcal{L}_{\text{nus}}$ of nonuniform and static programs:

$$t ::= s_1 \| \cdots \| s_n \quad (n \geq 1).$$

We require that the statements $s_1, \ldots, s_n$ are closed and furthermore that every index i occurring in t actually corresponds to a component statement, i.e., $i \leq n$.

We see $\mathcal{L}_{\text{nus}}$ is similar to (classical) CSP (as in [31]). There are also important differences: the absence (in $\mathcal{L}_{\text{nus}}$) of guarded commands with communication in guards or features such as the distributed termination convention. On the other hand, $\mathcal{L}_{\text{nus}}$ has full recursion rather than only iteration. Compared with $\mathcal{L}_{\text{us}}$, we have simplified $\mathcal{L}_{\text{nus}}$ by dropping the "$\cup$" operator. Extension of the treatment below to cover "$\cup$" is not difficult and we leave it to the reader.

5.2. Operational semantics

We proceed with the development of the framework for the operational semantics for $\mathcal{L}_{\text{nus}}$. Syntactic continuations r are, as before, defined by

$$r ::= E \mid s ; r'$$

where s is closed. Instead of parallel syntactic continuations ρ in the form of n-tuples $r_1, \ldots, r_n$, we now let ρ range over sets of the form

$$\{\langle i_1, r_1\rangle, \ldots, \langle i_n, r_n\rangle\} \quad (n \geq 1)$$

where all the indices $i_1, \ldots, i_n$ must be different. Thus, in the pair $\langle i, r\rangle$, we make explicit the identity of the component r. We shall not require that every index i occurring in a communication statement $i!e$ or $i?x$ within ρ also occurs as the first component of a pair $\langle i, r\rangle \in \rho$.

We shall often use the notation $\rho \cup \{\langle i, r_i\rangle\}$, with the convention that ρ is supposed not to contain an element of the form $\langle i, r'\rangle$. Such a condition also applies to the notation $\rho \cup \{\langle i, r_i\rangle, \langle j, r_j\rangle\}$: here we suppose that $i \neq j$ and that ρ does not contain an element whose index is i or j.

The next step in the development of the semantic model is the introduction of *states*, and of the meaning or evaluation function for expressions (and conditions).

5.2. Definition. (1) The set of states Σ, with typical element σ, is defined by

$$\Sigma = I \to (IndV \to V).$$

(2) We define the meaning function for expressions,

$$[\![\cdot]\!] : Exp \to (I \to (\Sigma \to V)),$$

as follows:

$$[\![x]\!](i)(\sigma) = \sigma(i)(x), \qquad [\![\alpha]\!](i)(\sigma) = \alpha,$$
$$[\![e_1 \ \mathbf{op} \ e_2]\!](i)(\sigma) = ([\![e_1]\!](i)(\sigma)) \ \mathbf{op}_{\text{sem}}([\![e_2]\!](i)(\sigma)),$$
$$[\![\mathbf{op} \ e]\!](i)(\sigma) = \mathbf{op}_{\text{sem}}([\![e]\!](i)(\sigma)).$$

Here we use $\mathbf{op}_{\text{sem}}$ for the semantic operator corresponding to **op**.

(3) We do not give a detailed definition of $[\![b]\!](i)(\sigma)$, which yields an element of the set of truth values $\{\mathbf{t}, \mathbf{f}\}$.

The operational semantics for $\mathcal{S}_{\text{nus}}$ and $\mathcal{L}_{\text{nus}}$ is again given through a transition system. This time, configurations are of the form $\langle \rho, \sigma\rangle$. Transitions are pairs of configurations written in the form

$$\langle \rho, \sigma\rangle \to \langle \rho', \sigma'\rangle.$$

There is no special role here for (an equivalent of) the $\perp$-action.

Nonuniform transitions involve states rather than streams as the intermediate and final results. Since states are entities which are not naturally amenable to the operation of merging, we shall encounter below the necessity to resort to additional means to formulate results which are counterparts of uniform facts such as $\mathcal{O}[\![r_1, r_2]\!] = \mathcal{O}[\![r_1]\!] \parallel \mathcal{O}[\![r_2]\!]$.

We first give the transition system $\mathcal{T}_{\text{nus}}$ for $\mathcal{L}_{\text{nus}}$. Extending the formalism of the uniform case, we also employ *rules*, written in the format

$$\frac{1 \to 2}{3 \to 4}.$$

The meaning of such a rule is the following: In case a transition $1 \to 2$ is an element of $\mathcal{T}_{\text{nus}}$, then the rule allows us to infer that $3 \to 4$ is a valid transition of $\mathcal{T}_{\text{nus}}$ as well.

Remark. Our framework for the operational semantics gives us quite some freedom, so that we can choose whether to use a rule or an axiom to express the semantics of a certain construct. The intuitive meaning remains the same, but technically an axiom needs a transition to perform a certain transformation, while a rule does not. We could, in fact, formulate the operational semantics for $\mathcal{L}_{\text{nus}}$ in terms of axioms only, but we prefer the version as adopted below. The reason for this is our wish to stay as close as possible to the denotational semantics to be developed subsequently. The denotational framework does not provide so much freedom, mainly because of the necessity to arrive at contracting operators having unique fixed points. We have chosen the denotational semantics with the least possible number of computation steps, and tuned the operational semantics to match it.

5.3. Definition. The transition system $\mathcal{T}_{\text{nus}}$ specifies the relation "$\rightarrow$" between configurations of the form $\langle\rho, \sigma\rangle$ as the *smallest* relation which satisfies the following axioms and rules:

$$\langle\rho \cup \{\langle i, (x := e);r\rangle\}, \sigma\rangle \rightarrow \langle\rho \cup \{\langle i, r\rangle\}, \sigma'\rangle \qquad \textbf{Ass}$$

where $\sigma' = \sigma\{\sigma(i)\{\beta/x\}/i\}$ and $\beta = [\![e]\!](i)(\sigma)$.

$$\frac{\langle\rho \cup \{\langle i, s_1;(s_2;r)\rangle\}, \sigma\rangle \rightarrow \langle\rho', \sigma'\rangle}{\langle\rho \cup \{\langle i, (s_1;s_2);r\rangle\}, \sigma\rangle \rightarrow \langle\rho', \sigma'\rangle}, \qquad \textbf{SeqComp}$$

$$\langle\rho \cup \{\langle i, \mu v[s];r\rangle\}, \sigma\rangle \rightarrow \langle\rho \cup \{\langle i, s[\mu v[s]/v];r\rangle\}, \sigma\rangle, \qquad \textbf{Rec}$$

$$\langle\rho \cup \{\langle i, \textsf{if } b \textsf{ then } s_1 \textsf{ else } s_2 \textsf{ fi};r\rangle\}, \sigma\rangle \rightarrow \langle\rho \cup \{\langle i, s_1;r\rangle\}, \sigma\rangle \qquad \textbf{Cond}$$

in case $[\![b]\!](i)(\sigma) = \mathbf{t}$, and an analogous axiom for the case $[\![b]\!](i)(\sigma) = \mathbf{f}$.

$$\langle\rho \cup \{\langle i, (j?x);r_1\rangle, \langle j, (i!e);r_2\rangle\}, \sigma\rangle \rightarrow \langle\rho \cup \{\langle i, r_1\rangle, \langle j, r_2\rangle\}, \sigma'\rangle \qquad \textbf{Comm}$$

where $\sigma' = \sigma\{\sigma(i)\{\beta/x\}/i\}$, and $\beta = [\![e]\!](j)(\sigma)$.

Remarks. (1) Observe that no transition is defined for a configuration $\langle\rho \cup \{\langle i, (j?x);r\rangle\}, \sigma\rangle$ in the case that ρ does not contain the matching pair $\langle j, (i!e);r'\rangle$ (and a symmetric observation).

(2) The difference in treatment between **SeqComp** and **Rec**—the first as a rule the second as an axiom—is motivated by the corresponding definition in the denotational semantics (which will be given in Definition 5.8). In operational terms replacing $(s_1;s_2);r$ by $s_1;(s_2;r)$ does not take a time step, whereas the replacement of $\mu v[s]$ by $s[\mu v[s]/v]$ *does* take a (silent) time step, (i.e., a step that does not change the state). In a uniform setting, the same effect would be obtained by transforming each recursive construct $\mu x[s]$ into $\mu x[\textsf{skip};s]$ where **skip** is a special elementary action denoting the silent step. Accordingly, the automatic introduction of silent steps obviates the need for the guardedness restriction.

(3) In the axioms **Ass**, **Cond**, and **Comm** we see how the evaluation of an expression e or condition b is parameterized by the index of the statement which contains the occurrence of the expression or condition involved. Effectively, this means that different components are treated as if they had disjoint sets of variables.

The transition system $\mathcal{T}_{\text{nus}}$ is a natural generalization of the corresponding system $\mathcal{T}_{\text{us}}$ and $\mathcal{T}_{\text{ud}}$. What is more difficult is the definition of $\mathcal{O}[\![\rho]\!]$ and $\mathcal{O}[\![t]\!]$: a formulation which is a straightforward extension of the uniform approach is not feasible assuming that we want to express results which are variations on relationships such as

$$\mathcal{O}[\![\rho_1 \cup \rho_2]\!] = \mathcal{O}[\![\rho_1]\!] \parallel \mathcal{O}[\![\rho_2]\!]. \qquad (5.1)$$

Two problems arise when we consider (5.1). The first concerns the basic question as to well-formedness of (5.1): we have as yet no outcome for $\mathcal{O}[\![\rho]\!]$ which allow the operation of merging to be applied to two instances of it. The second may be considered as a more "practical" one: In a situation where ρ_1 involves a send and ρ_2 a matching receive communication, $\rho_1 \cup \rho_2$ will allow a matching transition by the **Comm** axiom, whereas the components ρ_1 and ρ_2 separately do not allow the corresponding send and receive actions to proceed. Thus, we expect that neither $\mathcal{O}[\![\rho_1]\!]$ nor $\mathcal{O}[\![\rho_2]\!]$ will contain the necessary information enabling the communication to take place through the semantic operator "$\|$" (in whatever way the latter will be defined).

In order to solve the principal problem, we apply a new method, which might be considered somewhat drastic in an operational context: we choose to deliver a *process*, now taken in the technical sense of Section 2.3, as the outcome of $\mathcal{O}[\![\rho]\!]$. Thus, the outcome of $\mathcal{O}[\![\rho]\!]$ is an element of a certain *process domain* P obtained as the solution of an appropriate recursive domain equation $P \cong \mathcal{F}(P)$, where the form of $\mathcal{F}$ is to be determined in a moment. We intend to show that, by adopting this approach, we achieve two goals: Firstly, we shall be in a position to define "$\|$" as an operation on processes and to apply it to $\mathcal{O}[\![\rho_1]\!]$ and $\mathcal{O}[\![\rho_2]\!]$ above. Secondly, since we shall employ processes as well in our denotational model, we have a much smaller distance to bridge between the operational and denotational definitions.

The domain equation we use to determine the appropriate process domain P exploited below is described in the following definition.

5.4. Definition. (1) Let the set *Comm* of communications, with typical element τ, be given by

$$Comm = I \times (I?IndV \cup I!V).$$

(The delimiters "?" and "!" are used here to underline the connection with statements of the form $i!x$ and $i!e$. Properly speaking, they are cosmetic variants of the Cartesian product operator "$\times$".)

(2) Let the set *Step* of steps, with typical element η, be given by

$$Step = \Sigma \cup Comm.$$

(3) Let the function $\mathcal{F}$ be given by

$$\mathcal{F}(P) = \{p_0\} \cup (\Sigma \to \mathcal{P}_{\text{cl}}(Step \times P)).$$

(4) Let P be the process domain solving the equation $P \cong \mathcal{F}(P)$. We shall use p, q to range over P.

(5) Let $P_0 = \{p_0\}$, $P_{n+1} = \mathcal{F}(P_n)$. By the general theory (Section 2.3) we know that each $p \in P$ is either an element of some P_n, in which case we shall call p *finite*, or else p is called *infinite* and there is a Cauchy sequence $(p_n)_n$ with $p_n \in P_n$ such that $p = \lim_n p_n$. For finite p, we call the smallest n such that $p \in P_n$ its *degree*.

(6) We shall use X, Y to range over $\mathcal{P}_{\text{cl}}(Step \times P)$ and π to range over $Step \times P$.

Example. We have $\langle\langle i, j?x\rangle, p\rangle \in Step \times P$. Below, we shall always adopt for this the simpler notation $\langle i, j?x, p\rangle$.

We proceed with the semantic definitions for the familiar operators "$\cdot$" and "$\|$", this time defined as mappings $P \times P \to P$. We shall in fact propose two definitions. The first one is probably simpler, and is based on an induction on the degree for finite processes. The second one involves Banach's theorem and is given here to familiarize the reader with its subsequent use in definitions where the simpler inductive definition is less convenient.

5.5. Definition. Let $p, q \in P$. We define $p \cdot q$ and $p \| q$ as follows:

(1) (Definition by induction on the degree of p and q.) We first consider the case that both p and q are finite. We put $p_0 \cdot p = p_0 \| p = p \| p_0 = p$. If p is (or if p and q are) different from p_0, we put

$$p \cdot q = \lambda\sigma.(p(\sigma) \cdot q),$$
$$p \| q = \lambda\sigma.((p(\sigma) \| q) \cup (q(\sigma) \| p) \cup (p(\sigma)|_\sigma\, q(\sigma)))$$

where $X \cdot q = \{\pi \cdot q \mid \pi \in X\}$, $X \| q = \{\pi \| q \mid \pi \in X\}$, $\langle \eta, p'\rangle \cdot q = \langle \eta, p' \cdot q\rangle$, and $\langle \eta, p'\rangle \| q = \langle \eta, p' \| q\rangle$ (note that, here, the degree of p' is less than the degree of p, or the maximum of the degrees of p and q). Moreover,

$$X|_\sigma\, Y = \bigcup \{\pi_1|_\sigma\, \pi_2 \mid \pi_1 \in X, \pi_2 \in Y\},$$

where $\pi_1|_\sigma\, \pi_2$ is defined by

$$\langle i, j?x, p_1\rangle|_\sigma \langle j, i!\alpha, p_2\rangle = \{\langle \sigma', p_1 \| p_2\rangle\}$$

with $\sigma' = \sigma\{\sigma(i)\{\alpha/x\}/i\}$, together with a symmetric clause, and $\pi_1|_\sigma\, \pi_2 = \emptyset$ for π_1, π_2 not of the above form.

Finally, for p or q infinite, so that we have $p = \lim_n p_n$ and $q = \lim_n q_n$ with $p_n, q_n \in P_n$, we put $p \cdot q = \lim_n (p_n \cdot q_n)$ and $p \| q = \lim_n (p_n \| q_n)$.

(2) (Definition with Banach's theorem.) We define "$\cdot$" and "$\|$" as the unique fixed points of the contracting (higher-order) functions $\Phi, \Psi : (P \times P \to P) \to (P \times P \to P)$ given in the following manner: Let $\varphi, \psi \in P \times P \to P$ be arbitrary. We now define $\Phi(\varphi)$ and $\Psi(\psi)$. Let us abbreviate $\Phi(\varphi)(p, q)$ to $p \tilde{\varphi} q$ and $\Psi(\psi)(p, q)$ to $p \tilde{\psi} q$. Then we put

$$p \tilde{\varphi} q = \begin{cases} q & \text{if } p = p_0, \\ \lambda\sigma.(p(\sigma)\, \hat{\varphi}\, q) & \text{if } p \neq p_0; \end{cases}$$

$$p \tilde{\psi} q = \begin{cases} q & \text{if } p = p_0, \\ p & \text{if } q = p_0, \\ \lambda\sigma.((p(\sigma)\, \hat{\psi}\, q) \cup (q(\sigma)\, \hat{\psi}\, p) \cup (p(\sigma)|_{\sigma,\psi}\, q(\sigma))) & \text{otherwise;} \end{cases}$$

where $X \hat{\varphi} q = \{\pi \check{\varphi} q \mid \pi \in X\}$, $X \hat{\psi} q = \{\pi \check{\psi} q \mid \pi \in X\}$, $\langle \eta, p'\rangle \check{\varphi} q = \langle \eta, p' \varphi q\rangle$, $\langle \eta, p'\rangle \check{\psi} q = \langle \eta, p' \psi q\rangle$, and where

$$X|_{\sigma,\psi}\, Y = \bigcup \{\pi_1|_{\sigma,\psi}\, \pi_2 \mid \pi_1 \in X, \pi_2 \in Y\}.$$

Here $\pi_1|_{\sigma,\psi}\,\pi_2$ is given by

$$\langle i, j?x, p_1\rangle|_{\sigma,\psi}\,\langle j, i!\alpha, p_2\rangle = \{\langle \sigma', p_1 \psi p_2\rangle\}$$

with $\sigma' = \sigma\{\sigma(i)\{\alpha/x\}/i\}$, together with a symmetric clause, and $\pi_1|_{\sigma,\psi}\,\pi_2 = \emptyset$ for π_1, π_2 not of the above form.

Now we define "$\cdot$" to be the unique fixed point of Φ and "$\|$" as the unique fixed point of Ψ.

It should be clear from these definitions that they are variations on one theme: in the second an appeal to Banach's theorem replaces the inductive argument of the first. We omit the proof that the above definitions are justified (and that they define the same operators). Details of a very similar proof are given in [7].

We are now ready for definition of the operational semantics of $\mathcal{L}_{\text{nus}}$.

5.6. Definition. (1) We define $\mathcal{O}[\![\cdot]\!]: PSyCo \to P$ as follows: Let $\rho \in PSyCo$. If $\rho \subseteq \{\langle 1, E\rangle, \ldots, \langle n, E\rangle\}$, we put $\mathcal{O}[\![\rho]\!] = p_0$. Otherwise,

$$\mathcal{O}[\![\rho]\!] = \lambda\sigma.\{\langle \sigma', \mathcal{O}[\![\rho']\!]\rangle \mid \langle \rho, \sigma\rangle \to \langle \rho', \sigma'\rangle\}$$

where, of course, the transition relation "$\to$" is the one given by $\mathcal{T}_{\text{nus}}$.

(2) The function $\mathcal{O}[\![\cdot]\!]: \mathcal{L}_{\text{nus}} \to P$ is defined as follows. Let $t = s_1 \| \cdots \| s_n$. Then

$$\mathcal{O}[\![t]\!] = \mathcal{O}[\![\{\langle 1, s_1; E\rangle, \ldots, \langle n, s_n; E\rangle\}]\!].$$

It is not difficult to verify that $\mathcal{O}$ as given in part (1) of this definition is well-defined. Once more, we deduce this by the following reasoning: Let the (higher-order) mapping $F: (PSyCo \to P) \to (PSyCo \to P)$ be defined in the following manner:

$$F(\mathcal{M})(\rho) = \begin{cases} p_0 & \text{if } \rho \subseteq \{\langle 1, E\rangle, \ldots, \langle n, E\rangle\}, \\ \lambda\sigma.\{\langle \sigma', \mathcal{M}(\rho')\rangle \mid \langle \rho, \sigma\rangle \to \langle \rho', \sigma'\rangle\} & \text{otherwise.} \end{cases}$$

Then F is a contracting mapping, and $\mathcal{O}$ as given in Definition 5.6(1) is the unique fixed point of F.

Remarks. (1) It is not difficult to establish that, for each $\langle \rho, \sigma\rangle$, there are only finitely many $\langle \rho', \sigma'\rangle$ such that $\langle \rho, \sigma\rangle \to \langle \rho', \sigma'\rangle$. Hence, the set occurring in the $\lambda\sigma.\{\ldots\}$ clause in Definition 5.6(1) is finite and therefore closed.

(2) Note that $\mathcal{O}[\![\rho]\!] = \lambda\sigma.\emptyset$ may well occur. For example, $\mathcal{O}[\![\{\langle 1, (2?x); E\rangle\}]\!] = \lambda\sigma.\emptyset$ since there are no transitions $\langle\{\langle 1, (2?x); E\rangle\}, \sigma\rangle \to \cdots$ defined in $\mathcal{T}_{\text{nus}}$. In general, $\mathcal{O}$ does not preserve information on one-sided attempts at communication.

(3) Processes p which equal $\mathcal{O}[\![\rho]\!]$ for some ρ are in fact elements of a process domain P' which satisfies

$$P' \cong \{p_0\} \cup (\Sigma \to \mathcal{P}_{\text{cl}}(\Sigma \times P')).$$

This is the case since no steps in $Comm \times P$ are delivered by the transition relation "$\to$". The more involved process domain P is exploited in full only in the definitions of $\mathcal{O}^*$ and of the denotational semantics $\mathcal{D}$, both of which we shall discuss presently.

Now that we have given a process interpretation for $\mathcal{O}[\![\rho]\!]$, yielding results in a domain for which "$\|$" is well-defined, we have a well-formed question to ask: is it true that $\mathcal{O}[\![\rho_1 \cup \rho]\!] = \mathcal{O}[\![\rho_1]\!] \| \mathcal{O}[\![\rho_2]\!]$? The answer is negative—for the same reason as already explained earlier. However, a not too far-fetched variation on this property, which does indeed hold, will be presented soon. Rather than immediately getting to this, we first develop the denotational semantics for $\mathcal{L}_{\text{nus}}$. In this way, the reader may acquire some additional appreciation for the way we utilize the process notion in our framework. In fact, a combination of ideas involving:

- the tools of environments and semantic continuations as employed in Section 3,
- the operational semantics of $\mathcal{L}_{\text{nus}}$, and
- the definition(s) of "$\|$"

will altogether provide most of the background to understand the denotational definition.

5.3. Denotational semantics

We introduce semantic continuations and environments in the following definition.

5.7. Definition. (1) The set of semantic continuations is given by $SeCo =^{\text{def}} P$.

(2) We define the set of environments by $\Gamma =^{\text{def}} StmV \to (I \to (SeCo \to^{\text{NDI}} P))$. We shall use p, q to range over $SeCo$ and γ to range over Γ.

The definition of $\mathcal{D}$ will be given for all $s \in \mathcal{S}_{\text{nus}}$ and all $t \in \mathcal{L}_{\text{nus}}$. Thus, the restriction to statements with only guarded recursion is lifted. As remarked earlier, this is explained by our definition of recursion which involves a treatment of recursive calls such that always at least one initial "silent" step is made upon "procedure entrance". That is, (the equivalent of) a transition is made which does not affect the state but which does take (what may be seen as) one unit of time. For example, execution of $\mu v[v]$ will result in an infinite sequence of such silent steps (rather than in just $\perp$ as in the uniform case). All this is a matter of taste rather than of principle. One may disagree with our feeling that silent steps are more natural in a nonuniform than in a uniform setting.

We now give the definitions of $\mathcal{D}[\![s]\!]$ and of $\mathcal{D}[\![t]\!]$. We shall often suppress parentheses around arguments of functions for easier readability.

5.8. Definition. (1) We define the function

$$\mathcal{D}[\![\cdot]\!] : \mathcal{S}_{\text{nus}} \to (\Gamma \to (I \to (SeCo \xrightarrow{\text{NDI}} P)))$$

as follows:

(a) $\mathcal{D}[\![x := e]\!]\gamma i p = \lambda\sigma.\{\langle\sigma', p\rangle\}$, where $\sigma' = \sigma\{\sigma(i)\{\alpha/x\}/i\}$ and $\alpha = [\![e]\!]i\sigma$;

(b) $\mathcal{D}[\![s_1; s_2]\!]\gamma i p = \mathcal{D}[\![s_1]\!]\gamma i(\mathcal{D}[\![s_2]\!]\gamma i p)$;

(c) $\mathcal{D}[\![\textbf{if } b \textbf{ then } s_1 \textbf{ else } s_2 \textbf{ fi}]\!]\gamma i p$
$= \lambda\sigma.\{\langle\sigma, \textbf{if } [\![b]\!]i\sigma = \textbf{t} \textbf{ then } \mathcal{D}[\![s_1]\!]\gamma i p \textbf{ else } \mathcal{D}[\![s_2]\!]\gamma i p \textbf{ fi}\rangle\}$;

(d) $\mathscr{D}[\![v]\!]\gamma ip = \gamma(v)ip$;

(e) $\mathscr{D}[\![\mu v[s]]\!]\gamma ip = \varphi_\infty(i)(p)$ where φ_∞ is the unique fixed point of the operator Φ, which maps the space $I \to (SeCo \to^{\mathrm{NDI}} P)$ to itself, and is given by

$$\Phi(\varphi) = \lambda i.\lambda p.\lambda\sigma.\{\langle\sigma, \mathscr{D}[\![s]\!]\gamma\{\varphi/v\}ip\rangle\};$$

(f) $\mathscr{D}[\![j?x]\!]\gamma ip = \lambda\sigma.\{\langle i, j?x, p\rangle\}$;
$\mathscr{D}[\![j!e]\!]\gamma ip = \lambda\sigma.\{\langle i, j!\alpha, p\rangle\}$, where $\alpha = [\![e]\!]i\sigma$.

(2) We define the function $\mathscr{D}[\![\cdot]\!] : \mathscr{L}_{\text{nus}} \to P$ as follows: Let $t = s_1 \| \cdots \| s_n$ and let γ be arbitrary. Then

$$\mathscr{D}[\![t]\!] = \mathscr{D}[\![s_1]\!]\gamma 1 p_0 \| \cdots \| \mathscr{D}[\![s_n]\!]\gamma n p_0.$$

Remark. The definition in clause (1)(e) above is justified by the fact that the function Φ is contracting. Note that its unique fixed point can again be obtained as $\varphi_\infty = \lim_k \varphi_k$, where φ_0 is arbitrary and

$$\varphi_{k+1} = \Phi(\varphi_k) = \lambda i.\lambda p.\lambda\sigma.\{\langle\sigma, \mathscr{D}[\![s]\!]\gamma\{\varphi_k/v\}ip\rangle\}.$$

Examples. (1) $\mathscr{D}[\![\mu v[v]]\!]\gamma ip = \lambda\sigma.\{\langle\sigma, \lambda\sigma.\{\langle\sigma, \ldots\rangle\}\rangle\}$.

(2) We have

$$\begin{aligned}\mathscr{D}[\![(2?x)\|(1!3)]\!] &= \mathscr{D}[\![2?x]\!]\gamma 1 p_0 \| \mathscr{D}[\![1!3]\!]\gamma 2 p_0\\ &= \lambda\sigma.\{\langle 1, 2?x, p_0\rangle\} \| \lambda\sigma.\{\langle 2, 1!3, p_0\rangle\} \overset{\text{def}}{=} q_1 \| q_2\\ &= \lambda\sigma.\{\langle 1, 2?x, q_2\rangle, \langle 2, 1!3, q_1\rangle, \langle\sigma\{\sigma(1)\{3/x\}/1\}, p_0\rangle\}.\end{aligned}$$

The resulting process, say q, contains two steps resulting from one-sided (failing) communication: $\langle 1, \ldots\rangle$ and $\langle 2, \ldots\rangle$. Moreover, there is one step resulting from successful communication: $\langle\sigma\{\ldots\}, p_0\rangle$, where 3 is assigned to x. We recall that the latter step ultimately results from the definition of $\pi_1|_\sigma \pi_2$ (or $\pi_1|_{\sigma,\psi} \pi_2$) given in Definition 5.5. The operation of abstraction, to be introduced in a moment, will simplify the result q to just $\lambda\sigma.\{\langle\sigma\{\ldots\}, p_0\rangle\}$, throwing away the unsuccessful parts $\langle 1, \ldots\rangle$ and $\langle 2, \ldots\rangle$.

5.4. Equivalence of operational and denotational semantics

We return to the question concerning the (non)compositionality of $\mathscr{O}$. We shall introduce an extension of $\mathscr{T}_{\text{nus}}$ to $\mathscr{T}^*_{\text{nus}}$, which induces an associated operational semantics $\mathscr{O}^*$, and we then settle the relationship between $\mathscr{O}$, $\mathscr{O}^*$, and $\mathscr{D}$.

5.9. Definition. (1) We expand the notion of configuration such that it includes pairs of the form $\langle\rho, \eta\rangle$ (recall that η ranges over $Step = \Sigma \cup Comm$). Therefore, in addition to configurations of the form $\langle\rho, \sigma\rangle$, we also consider configurations of the form $\langle\rho, \tau\rangle$. (Actually, the latter ones will only occur on the right-hand side of a transition.)

(2) The transition system $\mathcal{T}^*_{\text{nus}}$ extends the system $\mathcal{T}_{\text{nus}}$ of Definition 5.3 by adding to it the axioms

$$\langle \rho \cup \{\langle i, (j?x);r\rangle, \sigma\rangle \to \langle \rho \cup \{\langle i, r\rangle\}, \langle i, j?x\rangle\rangle, \qquad \textbf{IndComm1}$$

$$\langle \rho \cup \{\langle i, (j!e);r\rangle\}, \sigma\rangle \to \langle \rho \cup \{\langle i, r\rangle\}, \langle i, j!\alpha\rangle\rangle \qquad \textbf{IndComm2}$$

where $\alpha = [\![e]\!]i\sigma$. Moreover, the rule **SeqComp** of $\mathcal{T}_{\text{nus}}$:

$$\frac{\langle \rho \cup \{\langle i, s_1;(s_2;r)\rangle\}, \sigma\rangle \to \langle \rho', \sigma'\rangle}{\langle \rho \cup \{\langle i, (s_1;s_2);r\rangle\}, \sigma\rangle \to \langle \rho', \sigma'\rangle}$$

is replaced by

$$\frac{\langle \rho \cup \{\langle i, s_1;(s_2;r)\rangle\}, \sigma\rangle \to \langle \rho', \eta'\rangle}{\langle \rho \cup \{\langle i, (s_1;s_2);r\rangle\}, \sigma\rangle \to \langle \rho', \eta'\rangle}.$$

(3) The operational meaning $\mathcal{O}^*: PSyCo \to P$ is defined by

$$\mathcal{O}^*[\![\rho]\!] = \begin{cases} p_0 & \text{if } \rho \subseteq \{\langle 1, E\rangle, \ldots, \langle n, E\rangle\}, \\ \lambda\sigma.\{\langle \eta', \mathcal{O}^*[\![\rho']\!]\rangle \mid \langle \rho, \sigma\rangle \to \langle \rho', \eta'\rangle\} & \text{otherwise.} \end{cases}$$

(Here we take "$\to$" as determined by $\mathcal{T}^*_{\text{nus}}$.)

(4) The operational meaning $\mathcal{O}^*: \mathcal{L}_{\text{nus}} \to P$ is defined as follows: Let $t = s_1 \| \cdots \| s_n$. Then

$$\mathcal{O}^*[\![t]\!] = \mathcal{O}^*[\![\{\langle 1, s_1;E\rangle, \ldots, \langle n, s_n;E\rangle\}]\!].$$

Following the detailed analysis as in [16], it is not difficult to prove the following theorem.

5.10. Theorem. $\mathcal{O}^*[\![\rho_1 \cup \rho_2]\!] = \mathcal{O}^*[\![\rho_1]\!] \| \mathcal{O}^*[\![\rho_s]\!]$.

For example,

$$\begin{aligned} &\mathcal{O}^*[\![\{\langle 1, (2?x);E\rangle, \langle 2, (1!3);E\rangle\}]\!] \\ &\quad = \lambda\sigma.\{\langle 1, 2?x, p_1\rangle, \langle 2, 1!3, p_2\rangle, \langle \sigma\{\sigma(1)\{3/x\}/1\}, p_0\rangle\} \end{aligned}$$

where $p_1 = \lambda\sigma.\{\langle 2, 1!3, p_0\rangle\}$ and $p_2 = \lambda\sigma.\{\langle 1, 2?x, p_0\rangle\}$. Thus,

$$\begin{aligned} \mathcal{O}^*[\![\{\langle 1, (2?x);E\rangle, \langle 2, (1!3);E\rangle\}]\!] &= \lambda\sigma.\{\langle 1, 2?x, p_0\rangle\} \| \lambda\sigma.\{\langle 2, 1!3, p_0\rangle\} \\ &= \mathcal{O}^*[\![\{\langle 1, (2?x);E\rangle\}]\!] \| \mathcal{O}^*[\![\{\langle 2, (1!3);E\rangle\}]\!]. \end{aligned}$$

The relationship between $\mathcal{O}$ and $\mathcal{O}^*$ is settled by the introduction of an *abstraction operator* $abs: P \to P'$ (with P' as given in remark (3) after Definition 5.6). When applied to some $p \in P$, $abs(p)$ deletes from p all pairs $\langle \tau, p'\rangle$ which occur anywhere "inside" p: all unsuccessful attempts at communication disappear, and only the results of successful communications remain, together with the "normal" steps caused by, e.g., assignments. Again (as was the case with any p), $abs(p)$ may have (inner) branches of the form $\lambda\sigma.\emptyset$—a phenomenon which is often called *deadlock*.

The abstraction operator is defined as follows.

5.11. Definition. For finite p we put $abs(p_0) = p_0$, $abs(\lambda\sigma.X) = \lambda\sigma.abs(X)$, and

$$abs(X) = \{\langle\sigma', abs(p')\rangle \mid \langle\sigma', p'\rangle \in X\}.$$

(Note that a pair $\langle\tau, p'\rangle \in X$ will not contribute to $abs(X)$.) For infinite p, with $p = \lim_n p_n$ and $p_n \in P_n$, we take $abs(p) = \lim_n abs(p_n)$.

Again relying on the general results in [16], we have the following theorem.

5.12. Theorem. $\mathcal{O} = abs \circ \mathcal{O}^*$.

The final part of this section is devoted to the proof of the equality of $\mathcal{O}^*$ and $\mathcal{D}$.

5.13. Theorem. *For all* $t \in \mathcal{L}_{\text{nus}}$, $\mathcal{O}^*[\![t]\!] = \mathcal{D}[\![t]\!]$.

The proof closely follows the strategy applied for the uniform version of this result described in Section 3. We first state a simple lemma on $\mathcal{O}^*$ which we need below.

5.14. Lemma. (1) $\mathcal{O}^*[\![\{\langle i, (x := e);r\rangle\}]\!] = \lambda\sigma.\{\langle\sigma', \mathcal{O}^*[\![\{\langle i, r\rangle\}]\!]\rangle\}$, *with* σ' *as usual.*

(2) $\mathcal{O}^*[\![\{\langle i, (s_1;s_2);r\rangle\}]\!] = \mathcal{O}^*[\![\{\langle i, s_1;(s_2;r)\rangle\}]\!]$.

(3) $\mathcal{O}^*[\![\{\langle i, \textbf{if } b \textbf{ then } s_1 \textbf{ else } s_2 \textbf{ fi}; r\rangle\}]\!]$
$= \lambda\sigma.\{\langle\sigma, \textbf{if } [\![b]\!]i\sigma \textbf{ then } \mathcal{O}^*[\![\{\langle i, s_1;r\rangle\}]\!] \textbf{ else } \mathcal{O}^*[\![\{\langle i, s_2;r\rangle\}]\!] \textbf{fi}\rangle\}$.

(4) $\mathcal{O}^*[\![\{\langle i, (j?x);r\rangle\}]\!] = \lambda\sigma.\{\langle i, j?x, \mathcal{O}^*[\![\{\langle i, r\rangle\}]\!]\rangle\}$.

(5) $\mathcal{O}^*[\![\{\langle i, (j!e);r\rangle\}]\!] = \lambda\sigma.\{\langle i, j!\alpha, \mathcal{O}^*[\![\{\langle i, r\rangle\}]\!]\rangle\}$ *where* $\alpha = [\![e]\!]i\sigma$.

(6) $\mathcal{O}^*[\![\{\langle i, (j?x);r_1\rangle, \langle j, (i!e);r_2\rangle\}]\!] = \lambda\sigma.\{\langle i, j?x, \mathcal{O}^*[\![\{\langle i, r_1\rangle, \langle j, (i!e);r_2\rangle\}]\!]\rangle$,
$\langle j, i!\alpha, \mathcal{O}^*[\![\{\langle i, (j?x); r_1\rangle, \langle j, r_2\rangle\}]\!]\rangle, \langle\sigma', \mathcal{O}^*[\![\{\langle i, r_1\rangle, \langle j, r_2\rangle\}]\!]\rangle\}$ *with* $\alpha = [\![e]\!]i\sigma$ *and* σ' *as usual.*

Proof. Easy from the definitions of $\mathcal{T}^*_{\text{nus}}$ and $\mathcal{O}^*$. $\square$

Remark. Note that part (2) of this lemma would not hold in the form as given if $\mathcal{T}_{\text{nus}}$ contained an axiom for **SeqComp**, rather than a rule. Conversely, part (3) would not hold if we had a rule for **Cond**, instead of an axiom.

The next lemma applies some notation which is a slight variant of the one introduced preceding Theorem 3.18. Let us, temporarily, add the statement **skip** to our language, with an associated transition

$$\langle\rho \cup \{\langle i, \textbf{skip};r\rangle\}, \sigma\rangle \to \langle\rho \cup \{\langle i, r\rangle\}, \sigma\rangle \qquad \textbf{Skip}$$

(note that we could take **skip** as another name for $x := x$). Let, for given s and v, $s_v^{(n)}$ be defined by $s_v^{(0)} = \textbf{skip}$ and $s_v^{(n+1)} = \textbf{skip};s[s_v^{(n)}/v]$. We can then prove the following lemma, once more using the framework of [16].

5.15. Lemma. *For closed s:*

$$\mathcal{O}^*[\![\{\langle i, \mu v[s]; r\rangle\}]\!] = \lim_n \mathcal{O}^*[\![\{\langle i, s_v^{(n)}; r\rangle\}]\!].$$

We are now ready for the statement of the main step in the proof of Theorem 5.13.

5.16. Lemma. *Let $s \in \mathcal{S}_{\text{nus}}$ be arbitrary (not necessarily closed) and let the set of free statement variables in s be contained in $\{v_1, \ldots, v_k\}$, $k \geq 0$. Let $s_1, \ldots, s_k$ be closed statements, and let $\tilde{s} = s[s_h/v_h]_{h=1}^k$. Let, for any ρ, $\mathcal{O}[\![\rho]\!]$ be short for $\lambda p.(\mathcal{O}^*[\![\rho]\!] \cdot p)$. Let, furthermore, for $h = 1, \ldots, k$,*

$$\varphi_h = \lambda i. \mathcal{O}[\![\{\langle i, s_h; E\rangle\}]\!]$$

and let $\tilde{\gamma} = \gamma\{\varphi_h/v_h\}_{h=1}^k$. We then have, for any i,

$$\mathcal{O}[\![\{\langle i, \tilde{s}; E\rangle\}]\!] = \mathcal{D}[\![s]\!](\tilde{\gamma})(i).$$

Proof. Induction on the complexity of s, following the argument as given in the proof of Theorem 3.19, but for the addition of an extra parameter i, and replacement of X by p (and using Lemmas 5.14 and 5.15 to deal with the individual cases). □

5.17. Corollary. *For closed s:*

$$\mathcal{O}[\![\{\langle i, s; E\rangle\}]\!] = \mathcal{D}[\![s]\!](\gamma)(i).$$

Now it is easy to prove Theorem 5.13.

Proof of Theorem 5.13. Take any $t = s_1 \| \cdots \| s_n$. Then

$$\begin{aligned}\mathcal{O}^*[\![t]\!] &= \mathcal{O}^*[\![\{\langle 1, s_1; E\rangle, \ldots, \langle n, s_n; E\rangle\}]\!]\\ &= \mathcal{O}^*[\![\{\langle 1, s_1; E\rangle\}]\!] \,\|\, \cdots \,\|\, \mathcal{O}^*[\![\{\langle n, s_n; E\rangle\}]\!].\end{aligned}$$

By Corollary 5.17, we have for each i that

$$\mathcal{O}^*[\![\{\langle i, s_i; E\rangle\}]\!] = \mathcal{O}^*[\![\{\langle i, s_i; E\rangle\}]\!] \cdot p_0 = \mathcal{O}[\![\{\langle i, s_i; E\rangle\}]\!](p_0) = \mathcal{D}[\![s_i]\!](\gamma)(i)(p_0).$$

Thus,

$$\begin{aligned}\mathcal{O}^*[\![t]\!] &= \mathcal{O}^*[\![\{\langle 1, s_1; E\rangle\}]\!] \,\|\, \cdots \,\|\, \mathcal{O}^*[\![\{\langle n, s_n; E\rangle\}]\!]\\ &= \mathcal{D}[\![s_1]\!](\gamma)(1)(p_0) \,\|\, \cdots \,\|\, \mathcal{D}[\![s_n]\!](\gamma)(n)(p_0) = \mathcal{D}[\![t]\!]. \qquad \square\end{aligned}$$

Remark. Contrary to the situation for the uniform case, we have at present investigated only metric (operational and denotational) models for $\mathcal{L}_{\text{nus}}$. Therefore we have no information on the feasibility of order-theoretic models for this purpose.

6. A nonuniform and dynamic language

We have, at last, arrived at the presentation of the semantic models of a nonuniform and dynamic language. Not surprisingly, it brings a synthesis of the ideas of Sections 4 and 5; for the reader who has understood these sections, the present section contains few surprises. Still, some technical difficulties which are not straightforward from previous considerations remain to be overcome.

6.1. Informal introduction and syntax

As usual, we begin with the syntax. Statements are almost as before, but for the fact that communications $i?x$ or $i!e$ (with static i, $1 \leqslant i \leqslant n$) are now replaced by communications $e?x$ or $e!e'$, in which the value of the expression e is (the name of) a dynamically created process. The expression itself can be, for example, a variable, in which this process name is stored. The syntax of expressions also contains an essential new clause, viz. "**new**(c)". This expresses that a new process (of class c) is to be created. Each program consists of a set of *class declarations* $\langle c_k \Leftarrow s_k \rangle_{k=1}^{n}$, and, assuming that c above equals c_k for some k, the (side-)effect of **new**(c) is the creation of a new process which will execute the statement $s = s_k$. Here we have the counterpart of the construct **new**(s) in Section 4. In addition, this new process is referred to by a (new) name, say α, and the value of the expression e will be this name α. Therefore, in the (common) case that **new**(c) occurs in an assignment $x := \mathbf{new}(c)$, the name α of the newly created process is assigned to x. In this way, upon subsequent occurrences of x in, e.g., $x!e$, it is known that the value of e has to be sent to process α.

We now give the formal syntactic definitions. Let *CNam* be the collection of *class names*, with typical element c. Let *IndV* and *StmV* be as before, and let α and β range over the set *Obj* of objects to be defined presently.

6.1. Definition. (1) The set *Exp* of expressions, with typical element e, is defined by

$$e \ ::= \ x \mid \alpha \mid e_1 \mathbf{op}\, e_2 \mid \mathbf{op}\, e' \mid \mathbf{new}(c)$$

(Here, again, **op** stands for an arbitrary binary or unary operator.)

(2) We do not give a detailed syntactic definition for the set *Test* of conditions (with typical element b) but we assume, for simplicity, that conditions (unlike expressions) can be evaluated without side-effects.

(3) We define the set $\mathscr{S}_{\text{nud}}$ of statements, with typical element s, by

$$s \ ::= \ x := e \mid s_1 ; s_2 \mid v \mid \mu v[s'] \mid \mathbf{if}\ b\ \mathbf{then}\ s_1\ \mathbf{else}\ s_2\ \mathbf{fi} \mid e?x \mid e!e' \mid ?x \mid !e$$

(4) The set $\mathscr{L}_{\text{nud}}$ of programs, with typical element t is defined by

$$t \ ::= \ \langle c_1 \Leftarrow s_1, \ldots, c_n \Leftarrow s_n \rangle \quad (n \geqslant 1).$$

Here we require that all the s_i are closed, that all the c_i are different, and that any class name c occurring in any s_i (in the context **new**(c)) is one of $c_1, \ldots, c_n$.

Remarks. (1) In $\mathscr{S}_{\text{nud}}$ we allow communications of the form $?x$ or $!e$ which do not name a corresponding process (they are, in fact, willing to communicate with *any* other process). However, we shall require, in order that a match be established between a pair of send and receive statements, that at least one of the two explicitly identifies the process in which the other occurs. (Hence, no communication takes place between $?x$ and $!e$.)

(2) By convention, executing a program $t=\langle c_k \Leftarrow s_k\rangle_{k=1}^{n}$ is initiated by executing the statement $x:=\mathbf{new}(c_1)$, for some fresh x (i.e., some individual variable not occurring in t). In other words, a process of class c_1 is created implicitly. (Its name is stored nowhere, so this process cannot be addressed explicitly by other processes.)

(3) Note that we now have two forms of recursion, one in constructs of the form $\mu v[s]$ and the other in case of a declaration such as $c \Leftarrow \cdots c \cdots$.

The set *Obj* of objects replaces the set of values v which we encountered in Section 5. It consists firstly of the so-called *standard objects SObj*. Here one may think of the union of the set of values V and the truth-values $\{\mathbf{t}, \mathbf{f}\}$ as employed in Section 5. Moreover, we now also have the set of so-called *active objects AObj*, which consists of the names of processes as mentioned in the introductory paragraph of this section. In fact, we may see *AObj* as the generalization of the set I of Section 5. We define *AObj* as

$$AObj = CNam \times \mathbb{N}$$

where $\mathbb{N}$ is the set of nonnegative integers. At each moment an active object $\langle c, l\rangle$ is the name of the lth process of class c, i.e., the process created by the lth execution of a $\mathbf{new}(c)$ construct.

From now on we shall use the term "object" in the above sense, i.e., for an element of *AObj*, not to confuse it with the technical term "process" in the sense of Section 2.3, the precise meaning of which we shall give in Definition 6.5.

6.2. Operational semantics

We proceed with the preparations for the operational semantics for $\mathscr{L}_{\text{nud}}$. Firstly, we refine the class of syntactic continuations, by distinguishing between statement continuations and expression continuations.

6.2. Definition. (1) The class of syntactic statement continuations *SyStCo*, with typical element r, is defined by

$$r ::= E \mid s;r' \mid e:g$$

where s is closed. (The colon ":" used here should not be confused with the semantic operator ":" as introduced in Definition 4.7. Here it is simply a syntactic symbol, comparable with ";".)

(2) The class of syntactic expression continuations *SyExCo*, with typical element g, is defined by

$$g ::= \lambda z.r$$

where $z \in \mathit{IndV}$. Here z may not occur as the left-hand side of an assignment in r.

(3) The class of parallel syntactic (statement) continuations *PSyCo*, with typical element ρ, is defined as the collection of sets of the form

$$\{\langle \alpha_1, r_1\rangle, \ldots, \langle \alpha_n, r_n\rangle\} \quad (n \geqslant 0)$$

where the α_i are different elements of *AObj*.

The intuitive meaning of a syntactic expression continuation $g = \lambda z.r$ is to describe a computation which depends on some value. The variable z serves as a placeholder for this value in r. When g is given a value, i.e., an object $\alpha \in \mathit{Obj}$, then it delivers a syntactic statement continuation $r[\alpha/z]$ (where the value α is put in the place of z). A syntactic statement continuation r of the form $e{:}g$ is executed by first evaluating the expression e (which may or may not take some time steps or have some side-effect) and then feeding its value into g in the way described above. This yields a syntactic statement continuation which is executed subsequently.

We also extend the class of states by introducing a second component, as follows.

6.3. Definition. We define the set of states by $\Sigma = \Sigma_1 \times \Sigma_2$, with typical element $\sigma = \langle \sigma_{(1)}, \sigma_{(2)}\rangle$. We put $\Sigma_1 = \mathit{AObj} \to (\mathit{IndV} \to \mathit{Obj})$ and $\Sigma_2 = \mathit{CNam} \to \mathbb{N}$.

A state σ has the following function:

- The first component $\sigma_{(1)}$ is as σ in Section 5, but for the replacement of I by *AObj* and of V by *Obj*. Thus, for any object α and individual variable x, $\sigma_{(1)}(\alpha)(x)$ is the value of α's x-variable.
- The second component $\sigma_{(2)}$ records for each class name c the number $l = \sigma_{(2)}(c)$ of objects of that class that have been created up to this point.

We shall usually suppress indices and simply write σ, also in cases where $\sigma_{(1)}$ or $\sigma_{(2)}$ is meant.

In the transition system to be presented in a moment, we shall take into account the fact that evaluation of expressions may now be more involved since they may contain **new**-constructs. For reasons of simplicity, we shall not include a similar extension in our treatment of conditions. We shall, just as in Section 5, assume that evaluation of a condition b—expressed by the notation $[\![b]\!](\alpha)(\sigma)$—is simple and has no side-effects. (Of course, it is a minor exercise to adapt the treatment below to cover the case of conditions which may include **new**-constructs.)

The operational semantics for $\mathcal{L}_{\mathrm{nud}}$ is given in terms of a transition system $\mathcal{T}_{\mathrm{nud}}$ of axioms and rules for configurations $\langle \rho, \sigma\rangle$. Throughout, $\mathcal{T}_{\mathrm{nud}}$ assumes one fixed program $t = \langle c_k \Leftarrow s_k\rangle_{k=1}^{n}$, and we shall also assume that all class names occurring in any statement are declared in this program t. (We might carry the information contained in t along as an extra component of the configuration, but we find this too cumbersome.)

6.4. Definition. The transition system $\mathcal{T}_{\text{nud}}$ is given by the following axioms and rules:

$$\langle \rho \cup \{\langle \alpha, (x := \beta); r \rangle\}, \sigma \rangle \rightarrow \langle \rho \cup \{\langle \alpha, r \rangle\}, \sigma' \rangle \qquad \textbf{Ass1}$$

where $\sigma' = \sigma\{\sigma(\alpha)\{\beta/x\}/\alpha\}$.

$$\frac{\langle \rho \cup \{\langle \alpha, e : \lambda z.((x := z); r) \rangle\}, \sigma \rangle \rightarrow \langle \rho', \sigma' \rangle}{\langle \rho \cup \{\langle \alpha, (x := e); r \rangle\}, \sigma \rangle \rightarrow \langle \rho', \sigma' \rangle} \qquad \textbf{Ass2}$$

where z is a fresh variable, i.e., an individual variable not occurring in ρ, e, or r (actually, it is sufficient to require that z does not occur in r). Note that this rule is only useful if e is not itself a constant β.

SeqComp, **Rec**, and **Cond** are as in Definition 5.3 (with α replacing i).

$$\frac{\langle \rho \cup \{\langle \alpha, e : \lambda z.((z?x); r) \rangle\}, \sigma \rangle \rightarrow \langle \rho', \sigma' \rangle}{\langle \rho \cup \{\langle \alpha, (e?x); r \rangle\}, \sigma \rangle \rightarrow \langle \rho', \sigma' \rangle} \qquad \textbf{Receive1}$$

with z fresh.

$$\frac{\langle \rho \cup \{\langle \alpha, e : \lambda z.(e' : \lambda z'.((z!z'); r)) \rangle\}, \sigma \rangle \rightarrow \langle \rho', \sigma' \rangle}{\langle \rho \cup \{\langle \alpha, (e!e'); r \rangle\}, \sigma \rangle \rightarrow \langle \rho', \sigma' \rangle} \qquad \textbf{Send1}$$

with z and z' fresh.

$$\frac{\langle \rho \cup \{\langle \alpha, e : \lambda z.((!z); r) \rangle\}, \sigma \rangle \rightarrow \langle \rho', \sigma' \rangle}{\langle \rho \cup \{\langle \alpha, (!e); r \rangle\}, \sigma \rangle \rightarrow \langle \rho', \sigma' \rangle} \qquad \textbf{Send2}$$

with z fresh.

$$\langle \rho \cup \{\langle \alpha, (\beta?x); r_1 \rangle, \langle \beta, (\alpha!\alpha'); r_2 \rangle\}, \sigma \rangle \rightarrow \langle \rho \cup \{\langle \alpha, r_1 \rangle, \langle \beta, r_2 \rangle\}, \sigma' \rangle \qquad \textbf{Comm1}$$

where $\sigma' = \sigma\{\sigma(\alpha)\{\alpha'/x\}/\alpha\}$.

$$\langle \rho \cup \{\langle \alpha, (\beta?x);\, r_1 \rangle, \langle \beta, (!\alpha'); r_2 \rangle\}, \sigma \rangle \rightarrow \langle \rho \cup \{\langle \alpha, r_1 \rangle, \langle \beta, r_2 \rangle\}, \sigma' \rangle \qquad \textbf{Comm2}$$

with σ' as above.

$$\langle \rho \cup \{\langle \alpha, (?x); r_1 \rangle, \langle \beta, (\alpha!\alpha'); r_2 \rangle\}, \sigma \rangle \rightarrow \langle \rho \cup \{\langle \alpha, r_1 \rangle, \langle \beta, r_2 \rangle\}, \sigma' \rangle \qquad \textbf{Comm3}$$

with σ' as above.

$$\langle \rho \cup \{\langle \alpha, x : g \rangle\}, \sigma \rangle \rightarrow \langle \rho \cup \{\langle \alpha, \sigma(\alpha)(x) : g \rangle\}, \sigma \rangle. \qquad \textbf{IndV}$$

$$\frac{\langle \rho \cup \{\langle \alpha, r[\beta/z] \rangle\}, \sigma \rangle \rightarrow \langle \rho', \sigma' \rangle}{\langle \rho \cup \{\langle \alpha, \beta : \lambda z.r \rangle\}, \sigma \rangle \rightarrow \langle \rho', \sigma' \rangle}. \qquad \textbf{Obj}$$

$$\frac{\langle \rho \cup \{\langle \alpha, (\beta_1 \, \mathbf{op}_{\text{sem}} \, \beta_2) : g \rangle\}, \sigma \rangle \rightarrow \langle \rho', \sigma' \rangle}{\langle \rho \cup \{\langle \alpha, (\beta_1 \, \mathbf{op} \, \beta_2) : g \rangle\}, \sigma \rangle \rightarrow \langle \rho', \sigma' \rangle}. \qquad \textbf{Binop1}$$

Here, β_1 $\mathbf{op}_{\text{sem}}$ β_2 stands for the object β that results if we apply the semantic operator $\mathbf{op}_{\text{sem}}$ corresponding to **op** to the objects β_1 and β_2.

$$\frac{\langle \rho \cup \{\langle \alpha, e_1 : \lambda z_1.(e_2 : \lambda z_2.((z_1 \;\mathbf{op}\; z_2) : g))\rangle\}, \sigma\rangle \to \langle \rho', \sigma'\rangle}{\langle \rho \cup \{\langle \alpha, (e_1 \;\mathbf{op}\; e_2) : g\rangle\}, \sigma\rangle \to \langle \rho', \sigma'\rangle} \qquad \textbf{Binop2}$$

with z_1 and z_2 fresh.

$$\frac{\langle \rho \cup \{\langle \alpha, (\mathbf{op}_{\text{sem}}\, \beta) : g\rangle\}, \sigma\rangle \to \langle \rho', \sigma'\rangle}{\langle \rho \cup \{\langle \alpha, (\mathbf{op}\; \beta) : g\rangle\}, \sigma\rangle \to \langle \rho', \sigma'\rangle}. \qquad \textbf{Unop1}$$

Again, $\mathbf{op}_{\text{sem}}$ β stands for the object β' that results if we apply the semantic operator $\mathbf{op}_{\text{sem}}$ corresponding to **op** to the object β.

$$\frac{\langle \rho \cup \{\langle \alpha, e : \lambda z.((\mathbf{op}\; z) : g)\rangle\}, \sigma\rangle \to \langle \rho', \sigma'\rangle}{\langle \rho \cup \{\langle \alpha, (\mathbf{op}\; e) : g\rangle\}, \sigma\rangle \to \langle \rho', \sigma'\rangle} \qquad \textbf{Unop2}$$

with z fresh.

$$\langle \rho \cup \{\langle \alpha, \mathbf{new}(c) : g\rangle\}, \sigma\rangle \to \langle \rho \cup \{\langle \alpha, \beta : g\rangle, \langle \beta, s; E\rangle\}, \sigma'\rangle \qquad \textbf{New}$$

where $c \Leftarrow s$ occurs in t, $\beta = \langle c, \sigma(c) + 1\rangle$ and $\sigma' = \sigma\{\sigma(c) + 1/c\}$.

Remarks. (1) In the **New** axiom, dealing with the case $e = \mathbf{new}(c)$, a new object executing the statement s is created, and the name $\beta = \langle c, \sigma(c) + 1\rangle$ is delivered as the resulting value for e. As we already saw, $\langle c, l\rangle$ is the name of the lth object of class c, and, for each c, $\sigma(c)$ stores the currently highest object number. This also explains the update σ' of σ upon object creation.

(2) The general scheme to deal with expression evaluation is the following. If the expression e occurs in a certain context, for example $x := e; r$, then an application of a rule (in our example, **Ass2**) transforms the context to one of the form $e{:}g$ (in our case, $e : \lambda z.(x := z; r)$), indicating that first e is to be evaluated, after which its value can be used. Because a rule is applied and not an axiom, this does not take any time steps. Now the axioms **IndV** or **New** (which do take a time step) or rules like **Binop1** and **Unop1** (which do not take time) will take care of the evaluation of the expression. If necessary, the rules **Binop2** or **Unop2** will break the expression further apart (again without taking time). After the expression has been evaluated, the rule **Obj** will put the resulting object β back into the original context, and further axioms or rules (in our example, **Ass1**) will deal with this result β in an appropriate way.

The step from $\mathcal{T}_{\text{nud}}$ to the corresponding $\mathcal{O}$ is very similar to the one described in Section 5. We first introduce the relevant process domain.

6.5. Definition. (1) The set *Comm* of communications (with typical element τ) is defined by

$$Comm = AObj \times (AObj?IndV \cup ?IndV \cup AObj!Obj \cup !Obj).$$

(2) We define the set *Step* of steps (with typical element η) by

$$Step = \Sigma \cup Comm.$$

(3) The process domain P (typical elements p and q) is the solution of the following domain equation:

$$P \cong \{p_0\} \cup (\Sigma \to \mathcal{P}_{\text{cl}}(Step \times P)).$$

6.6. Definition. (1) $\mathcal{O}[\![\cdot]\!]: PSyCo \to P$ is defined by

$$\mathcal{O}[\![\rho]\!] = \begin{cases} p_0 & \text{if } \rho = \{\langle\alpha_1, E\rangle, \ldots, \langle\alpha_n, E\rangle\}, \\ \lambda\sigma.\{\langle\sigma', \mathcal{O}[\![\rho']\!]\rangle \mid \langle\rho, \sigma\rangle \to \langle\rho', \sigma'\rangle\} & \text{otherwise.} \end{cases}$$

(2) $\mathcal{O}[\![\cdot]\!]: \mathcal{L}_{\text{nud}} \to P$ is defined as follows. Let $t = \langle c_k \Leftarrow s_k \rangle_{k=1}^n$. Then

$$\mathcal{O}[\![t]\!] = \mathcal{O}[\![\{\langle\langle c_1, 1\rangle, s_1; E\rangle\}]\!].$$

Remark. Although not specified here, the process $p = \mathcal{O}[\![t]\!]$ will of course be started in a state σ_0, which satisfies $\sigma_0(c_1) = 1$ and $\sigma_0(c) = 0$ for $c \neq c_1$. The choice of this σ_0 and ρ above amounts to starting the computation with the first object of class c_1, while objects of other classes do not yet exist.

Anticipating the definition of $p \| q$, to be given in Definition 6.7, we again remark that it is not the case that $\mathcal{O}[\![\rho_1 \cup \rho_2]\!] = \mathcal{O}[\![\rho_1]\!] \| \mathcal{O}[\![\rho_2]\!]$. As before, we shall remedy this by extending $\mathcal{T}_{\text{nud}}$ to $\mathcal{T}^*_{\text{nud}}$, and then introducing a corresponding extension of $\mathcal{O}$ to $\mathcal{O}^*$.

6.3. *Denotational semantics*

We proceed with the denotational semantic definitions. We first fill in the details of the definition of the merge operator "$\|$" (in this section, we do not use the operator "$\cdot$").

6.7. Definition. Let Ψ, ψ, $\tilde{\psi}$, $\hat{\psi}$, $\check{\psi}$, X, Y, and π be as in Definition 5.5(2), but with P as in Definition 6.5. The only new element in the definition of "$\|$" with respect to Definition 5.5 concerns $\pi_1 |_{\sigma,\psi} \pi_2$, which is here given by

$$\langle\alpha, \beta ?x, p_1\rangle |_{\sigma,\psi} \langle\beta, \alpha !\alpha', p_2\rangle = \{\langle\sigma', p_1 \psi p_2\rangle\},$$

$$\langle\alpha, ?x, p_1\rangle |_{\sigma,\psi} \langle\beta, \alpha !\alpha', p_2\rangle = \{\langle\sigma', p_1 \psi p_2\rangle\},$$

$$\langle\alpha, \beta ?x, p_1\rangle |_{\sigma,\psi} \langle\beta, !\alpha', p_2\rangle = \{\langle\sigma', p_1 \psi p_2\rangle\}$$

with $\sigma' = \sigma\{\sigma(\alpha)\{\alpha'/x\}/\alpha\}$, together with three symmetric clauses, and $\pi_1 |_{\sigma,\psi} \pi_2 = \emptyset$ for π_1, π_2 not of the above form.

Corresponding to the distinction, for syntactic continuations, between statement continuations r and expression continuations g, we have a similar distinction at the semantic level: We have, besides the set of semantic statement continuations $SeStCo =^{\mathrm{def}} P$ (with typical element p), also a set of semantic expression continuations $SeExCo =^{\mathrm{def}} Obj \to P$, with typical element f.

Furthermore, corresponding to the two types of recursion, we accordingly have two components of an environment, defined as follows.

6.8. Definition. The set of environments is defined by $\Gamma = \Gamma_1 \times \Gamma_2$, with typical element $\gamma = \langle \gamma_{(1)}, \gamma_{(2)} \rangle$, where

$$\Gamma_1 = StmV \to (AObj \to (SeStCo \xrightarrow{\mathrm{NDI}} P)) \quad \text{and} \quad \Gamma_2 = CNam \to (AObj \to P).$$

In an environment $\gamma = \langle \gamma_{(1)}, \gamma_{(2)} \rangle$, the first component $\gamma_{(1)}$ assigns an interpretation to each statement variable, which gives a process after being told which object is to execute the statement and which process is to be activated after this statement variable. This first component corresponds to the environments as used in Section 5.

The second component $\gamma_{(2)}$ is important for the creation of new objects. When given the class c and the name α of the object to be created, $\gamma_{(2)}(c)(\alpha)$ is the process to be activated for it.

Again, we shall often omit the indices in dealing with environments.

We shall define two semantic evaluation functions $\mathcal{D}$ and $\mathcal{E}$, the first for statements and programs, and the second for expressions. Since expressions are now more involved than in Section 5, we consequently need a more complicated definition of their meanings. The relevant types are

$$\mathcal{D}[\![\cdot]\!] : \mathcal{S}_{\mathrm{nud}} \to (\Gamma \to (AObj \to (SeStCo \xrightarrow{\mathrm{NDI}} P))),$$

$$\mathcal{E}[\![\cdot]\!] : Exp \to (\Gamma \to (AObj \to (SeExCo \xrightarrow{\mathrm{NDI}} P)))$$

and, in addition, $\mathcal{D}[\![\cdot]\!] : \mathcal{L}_{\mathrm{nud}} \to P$. We draw attention to the fact that $\mathcal{E}[\![e]\!]$, when supplied with some γ, α, and f, delivers a process $p \in P$ instead of some value $\beta \in Obj$. Values (i.e., objects) which result from evaluating an expression are always passed on to some expression continuation rather than being delivered explicitly by the semantic function.

6.9. Definition. (1) The function $\mathcal{E}$ is defined by

(a) $\mathcal{E}[\![x]\!]\gamma\alpha f = \lambda\sigma.\{\langle \sigma, f(\sigma(\alpha)(x))\rangle\}$;

(b) $\mathcal{E}[\![\beta]\!]\gamma\alpha f = f(\beta)$;

(c) $\mathcal{E}[\![e_1 \ \mathbf{op} \ e_2]\!]\gamma\alpha f = \mathcal{E}[\![e_1]\!]\gamma\alpha(\lambda\beta_1.\mathcal{E}[\![e_2]\!]\gamma\alpha(\lambda\beta_2.f(\beta_1 \ \mathbf{op}_{\mathrm{sem}} \ \beta_2)))$;

(d) $\mathcal{E}[\![\mathbf{op} \ e]\!]\gamma\alpha f = \mathcal{E}[\![e]\!]\gamma\alpha(\lambda\beta.f(\mathbf{op}_{\mathrm{sem}} \ \beta))$;

(e) $\mathcal{E}[\![\mathsf{new}(c)]\!]\gamma\alpha f = \lambda\sigma.\{\langle \sigma', \gamma(c)(\beta) \| f(\beta)\rangle\}$ where $\beta = \langle c, \sigma(c)+1\rangle$ and $\sigma' = \sigma\{\sigma(c)+1/c\}$.

(2) We define the function $\mathscr{D}$ for statements as follows:

(a) $\mathscr{D}[\![x := e]\!]\gamma\alpha p = \mathscr{E}[\![e]\!]\gamma\alpha(\lambda\beta.\lambda\sigma.\{\langle\sigma', p\rangle\})$ where $\sigma' = \sigma\{\sigma(\alpha)\{\beta/x\}/\alpha\}$;

(b) $\mathscr{D}[\![s_1;s_2]\!]\gamma\alpha p = \mathscr{D}[\![s_1]\!]\gamma\alpha(\mathscr{D}[\![s_2]\!]\gamma\alpha p)$;

(c) $\mathscr{D}[\![\textbf{if } b \textbf{ then } s_1 \textbf{ else } s_2 \textbf{ fi}]\!]\gamma\alpha p$
$= \lambda\sigma.\{\langle\sigma, \textbf{if } [\![b]\!]\alpha\sigma = \textbf{t then } \mathscr{D}[\![s_1]\!]\gamma\alpha p \textbf{ else } \mathscr{D}[\![s_2]\!]\gamma\alpha p \textbf{ fi}\rangle\}$;

(d) $\mathscr{D}[\![v]\!]\gamma\alpha p = \gamma_{(1)}(v)\alpha p$;

(e) $\mathscr{D}[\![\mu v[s]]\!]\gamma\alpha p = \varphi_\infty(\alpha)(p)$, where φ_∞ is the unique fixed point of the function Φ, from the space $AObj \to (SeStCo \to^{\mathrm{NDI}} P)$ to itself, which is given by

$$\Phi(\varphi) = \lambda\alpha.\lambda p.\lambda\sigma.\{\langle\sigma, \mathscr{D}[\![s]\!]\gamma\{\varphi/v\}\alpha p\rangle\};$$

(f) $\mathscr{D}[\![e?x]\!]\gamma\alpha p = \mathscr{E}[\![e]\!]\gamma\alpha(\lambda\beta.\lambda\sigma.\{\langle\alpha, \beta?x, p\rangle\})$;

(g) $\mathscr{D}[\![?x]\!]\gamma\alpha p = \lambda\sigma.\{\langle\alpha, ?x, p\rangle\}$;

(h) $\mathscr{D}[\![e!e']\!]\gamma\alpha p = \mathscr{E}[\![e]\!]\gamma\alpha(\lambda\beta.\mathscr{E}[\![e']\!]\gamma\alpha(\lambda\beta'.\lambda\sigma.\{\langle\alpha, \beta!\beta', p\rangle\}))$;

(i) $\mathscr{D}[\![!e]\!]\gamma\alpha p = \mathscr{E}[\![e]\!]\gamma\alpha(\lambda\beta.\lambda\sigma.\{\langle\alpha, !\beta, p\rangle\})$.

(3) Let, for a program t, the mapping $\Psi_t: \Gamma_2 \to \Gamma_2$ be given as follows:

$$\Psi_t(\gamma_2)(c) = \lambda\alpha.\mathscr{D}[\![s]\!](\langle\gamma_1, \gamma_2\rangle)(\alpha)(p_0)$$

where $c \Leftarrow s$ occurs in t, and $\gamma_1 \in \Gamma_1$ is arbitrary (since t is closed, the choice of γ_1 is really immaterial). If c is not declared in t, we can put $\Psi_t(\gamma_2)(c) = \lambda\alpha.p_0$, for example.

Let γ_{2t} be the unique fixed point of Ψ_t (see the remark below). We put $\gamma_t =^{\mathrm{def}} \langle\gamma_1, \gamma_{2t}\rangle$, for arbitrary $\gamma_1 \in \Gamma_1$.

(4) Now we can define the denotational semantics of programs as follows. Let $t = \langle c_1 \Leftarrow s_1, \ldots, c_n \Leftarrow s_n\rangle$. Then

$$\mathscr{D}[\![t]\!] = \mathscr{D}[\![s_1]\!](\gamma_t)(\langle c_1, 1\rangle)(p_0).$$

Remarks. (1) The clause for $\mathscr{E}[\![\textbf{new}(c)]\!]$ uses essentially the same idea as in Section 4 of putting the newly created process $\gamma(c)(\beta)$ in parallel with the (expression) continuation f (supplied with the new name β which is the value of the expression **new**(c)). Here $\gamma(c)(\beta)$—or $\gamma_{(2)}(c)(\beta)$, to be precise—will, in the context of a program $t = \langle c_k \Leftarrow s_k\rangle_{k=1}^n$, contain the relevant information on the class c as a result of the definition of γ_t (to be precise, γ_{2t}) in clause (3). We also observe that due to our requirement that all class names used in a program t must be also be declared in it, the result of γ_t for undeclared classes does not matter (actually, new objects of such classes would execute the process p_0).

(2) Clause (2)(e) is justified by the fact that the mapping Φ is contracting. Again we can obtain its unique fixed point by $\varphi_\infty = \lim_i \varphi_i$, where φ_0 is arbitrary and

$$\varphi_{i+1} = \lambda\alpha.\lambda p.\lambda\sigma.\{\langle\sigma, \mathscr{D}[\![s]\!]\gamma\{\varphi_i/v\}\alpha p\rangle\}.$$

(3) The mapping Ψ_t in clause (3) is contracting since recursive occurrences of c in any s are always constituents of statements which take time steps (specifically in evaluating **new**(c)) before we apply γ to such a recursive occurrence of c.

6.4. Equivalence of operational and denotational semantics

We start this section with the promised extension of $\mathcal{T}_{\text{nus}}$ and $\mathcal{O}$.

6.10. Definition. (1) The notion of configuration is expanded so as to include pairs of the form $\langle \rho, \eta\rangle$ (note that η ranges over $\mathit{Step} = \Sigma \cup \mathit{Comm}$).

(2) We obtain the transition system $\mathcal{T}^*_{\text{nud}}$ from $\mathcal{T}_{\text{nud}}$ by adding the axioms

$$\langle \rho \cup \{\langle \alpha, (\beta ?x);r\rangle\}, \sigma\rangle \to \langle \rho \cup \{\langle \alpha, r\rangle\}, \langle \alpha, \beta ?x\rangle\rangle, \qquad \textbf{Receive2}$$

$$\langle \rho \cup \{\langle \alpha, (?x);r\rangle\}, \sigma\rangle \to \langle \rho \cup \{\langle \alpha, r\rangle\}, \langle \alpha, ?x\rangle\rangle, \qquad \textbf{Receive3}$$

$$\langle \rho \cup \{\langle \alpha, (\beta !\beta');r\rangle\}, \sigma\rangle \to \langle \rho \cup \{\langle \alpha, r\rangle\}, \langle \alpha, \beta !\beta'\rangle\rangle, \qquad \textbf{Send3}$$

$$\langle \rho \cup \{\langle \alpha, (!\beta);r\rangle\}, \sigma\rangle \to \langle \rho \cup \{\langle \alpha, r\rangle\}, \langle \alpha, !\beta\rangle\rangle \qquad \textbf{Send4}$$

and by replacing, in all rules,

$$\frac{\langle \rho_1, \sigma\rangle \to \langle \rho', \sigma'\rangle}{\langle \rho_2, \sigma\rangle \to \langle \rho', \sigma'\rangle}$$

by

$$\frac{\langle \rho_1, \sigma\rangle \to \langle \rho', \eta'\rangle}{\langle \rho_2, \sigma\rangle \to \langle \rho', \eta'\rangle}.$$

(3) Now we define $\mathcal{O}^*[\![\cdot]\!] : \mathit{PSyCo} \to P$ by

$$\mathcal{O}^*[\![\rho]\!] = \begin{cases} p_0 & \text{if } \rho = \{\langle \alpha_1, E\rangle, \ldots, \langle \alpha_n, E\rangle\}, \\ \lambda\sigma.\{\langle \eta', \mathcal{O}^*[\![\rho']\!]\rangle \mid \langle \rho, \sigma\rangle \to \langle \rho', \eta'\rangle\} & \text{otherwise.} \end{cases}$$

(4) $\mathcal{O}^*[\![\cdot]\!] : \mathcal{L}_{\text{nud}} \to P$ is defined as follows. Let $t = \langle c_k \Leftarrow s_k\rangle_{k=1}^n$. Then

$$\mathcal{O}^*[\![t]\!] = \mathcal{O}^*[\![\{\langle\langle c_1, 1\rangle, s_1; E\rangle\}]\!].$$

As in Section 5, we have the following lemma.

6.11. Lemma. $\mathcal{O}^*[\![\rho_1 \cup \rho_2]\!] = \mathcal{O}^*[\![\rho_1]\!] \parallel \mathcal{O}^*[\![\rho_2]\!]$.

The abstraction operator *abs* can be defined as in Definition 5.11 (but now applied to P as in Definition 6.5). Again, we have

6.12. Lemma. $\mathcal{O} = \mathit{abs} \circ \mathcal{O}^*$.

We can now discuss the relationship between $\mathcal{O}^*$ and $\mathcal{D}$. The treatment combines ideas of Sections 4 and 5. We first present a lemma listing various properties of $\mathcal{O}^*$

which are either direct from its definition, or follow as in Section 5 (in turn relying on [16]).

6.13. Lemma. (1) $\mathcal{O}^*[\![\{\langle\alpha, (x := \beta);r\rangle\}]\!] = \lambda\sigma.\{\langle\sigma', \mathcal{O}^*[\![\{\langle\alpha, r\rangle\}]\!]\rangle\}$ *with* σ' *as usual.*

(2) $\mathcal{O}^*[\![\{\langle\alpha, (x := e);r\rangle\}]\!] = \mathcal{O}^*[\![\{\langle\alpha, e : \lambda z.((x := z);r)\rangle\}]\!]$ *where* z *is fresh.*

(3) $\mathcal{O}^*[\![\{\langle\alpha, (s_1;s_2);r\rangle\}]\!] = \mathcal{O}^*[\![\{\langle\alpha, s_1;(s_2;r)\rangle\}]\!]$.

(4) $\mathcal{O}^*[\![\{\langle\alpha, \textbf{if } b \textbf{ then } s_1 \textbf{ else } s_1 \textbf{ fi};r\rangle\}]\!]$
$= \lambda\sigma.\{\langle\sigma, \textbf{if } [\![b]\!]\alpha\sigma \textbf{ then } \mathcal{O}^*[\![\{\langle\alpha, s_1;r\rangle\}]\!] \textbf{ else } \mathcal{O}^*[\![\{\langle\alpha, s_2;r\rangle\}]\!]\textbf{fi}\rangle\}$.

(5) $\mathcal{O}^*[\![\{\langle\alpha, \mu v[s];r\rangle\}]\!] = \lim_n \mathcal{O}^*[\![\{\langle\alpha, s_v^{(n)};r\rangle\}]\!]$ *where* $s_v^{(0)} = \textbf{skip}$ *and* $s_v^{(n+1)} = \textbf{skip};s[s_v^{(n)}/v]$. *Note that here we cannot use* $x := x$ *for* **skip** *any more because* $x := x$ *now costs* two *steps.*

(6) $\mathcal{O}^*[\![\{\langle\alpha, (e?x);r\rangle\}]\!] = \mathcal{O}^*[\![\{\langle\alpha, e : \lambda z.((z?x);r))\rangle\}]\!]$ *with* z *fresh, and similar equations for* $e!e'$ *and* $!e$.

(7) $\mathcal{O}^*[\![\{\langle\alpha, (\beta?x);r\rangle\}]\!] = \lambda\sigma.\{\langle\alpha, \beta?x, \mathcal{O}^*[\![\{\langle\alpha, r\rangle\}]\!]\rangle\}$ *and similar equations for* $?x$, $\beta!\beta'$, *and* $!\beta$.

(8) $\mathcal{O}^*[\![\{\langle\alpha, (\beta?x);r_1\rangle, \langle\beta, (\alpha!\alpha'); r_2\rangle\}]\!] = \lambda\sigma.\{\langle\alpha, \beta?x, \mathcal{O}^*[\![\{\langle\alpha, r_1\rangle, \langle\beta, (\alpha!\alpha');r_2\rangle\}]\!]\rangle$, $\langle\beta, \alpha!\alpha', \mathcal{O}^*[\![\{\langle\alpha, (\beta?x);r_1\rangle, \langle\beta, r_2\rangle\}]\!]\rangle, \langle\sigma', \mathcal{O}^*[\![\{\langle\alpha, r_1\rangle, \langle\beta, r_2\rangle\}]\!]\rangle\}$ *where* σ' *is as usual, and similar equations for* $?x$ *with* $\alpha!\alpha'$ *and for* $\beta?x$ *with* $!\alpha'$.

(9) $\mathcal{O}^*[\![\{\langle\alpha, x : g\rangle\}]\!] = \lambda\sigma.\{\langle\sigma, \mathcal{O}^*[\![\{\langle\alpha, \sigma(\alpha)(x) : g\rangle\}]\!]\rangle\}$.

(10) $\mathcal{O}^*[\![\{\langle\alpha, \beta : \lambda z.r\rangle\}]\!] = \mathcal{O}^*[\![\{\langle\alpha, r[\beta/z]\rangle\}]\!]$.

(11) $\mathcal{O}^*[\![\{\langle\alpha, (\beta_1 \textbf{ op } \beta_2) : g\rangle\}]\!] = \mathcal{O}^*[\![\{\langle\alpha, (\beta_1 \textbf{ op}_{\text{sem}} \beta_2) : g\rangle\}]\!]$ *and a similar equation for unary operators.*

(12) $\mathcal{O}^*[\![\{\langle\alpha, (e_1 \textbf{ op } e_2) : g\rangle\}]\!] = \mathcal{O}^*[\![\{\langle\alpha, e_1 : \lambda z_1.(e_2 : \lambda z_2.((z_1 \textbf{ op } z_2) : g)))\rangle\}]\!]$ *and a similar equation for unary operators.*

(13) $\mathcal{O}^*[\![\{\alpha, \textbf{new}(c) : g\rangle\}]\!] = \lambda\sigma.\{\langle\sigma', \mathcal{O}^*[\![\{\langle\alpha, \beta : g\rangle, \langle\beta, s;E\rangle\}]\!]\rangle\}$ *where* $c \Leftarrow s$ *occurs in* t *and with* $\sigma' = \sigma\{\sigma(c) + 1/c\}$ *and* $\beta = \langle c, \sigma(c) + 1\rangle$.

We continue with the analysis which links $\mathcal{O}^*$ with $\mathcal{D}$ and $\mathcal{E}$. Our aim is the proof of the following theorem.

6.14. Theorem. *For a given program* $t = \langle c_k \Leftarrow s_k\rangle_{k=1}^n$, *for closed* s, *arbitrary* r, e *and* g, *and for* γ_t *as in Definition* 6.9(3), *we have*

(1) $\mathcal{O}^*[\![\{\langle\alpha, e : g\rangle\}]\!] = \mathcal{E}[\![e]\!](\gamma_t)(\alpha)(\lambda\beta.\mathcal{O}^*[\![\{\langle\alpha, \beta : g\rangle\}]\!])$,

(2) $\mathcal{O}^*[\![\{\langle\alpha, s;r\rangle\}]\!] = \mathcal{D}[\![s]\!](\gamma_t)(\alpha)(\mathcal{O}^*[\![\{\langle\alpha, r\rangle\}]\!])$.

In order to prove this theorem, we apply a nonuniform version of the strategy used at the end of Section 4. Since we are concerned with both statements and expressions, we need the nonuniform argument in two forms. Firstly, we introduce the branching time analogues of the constructs $u\surd v$ from Section 4. One form also mentions the $\surd$, the other one is parameterized by objects β from *Obj*, each of which plays a role similar to the one played by $\surd$. For the remainder of this section we introduce *three* domains P, Q, and R with typical elements p, q, and r respectively (the last not to be confused with $r \in \mathit{SyStCo}$).

6.15. Definition. (1) Recall from Definition 6.5 that P is the solution of

$$P \cong \{p_0\} \cup (\Sigma \to \mathcal{P}_{\mathrm{cl}}(Step \times P)).$$

As before, we shall use X to range over $\mathcal{P}_{\mathrm{cl}}(Step \times P)$ and π to range over $Step \times P$.

(2) The domain Q is the solution of the following domain equation

$$Q \cong \{p_0\} \cup (\{\surd\} \times P) \cup (\Sigma \to \mathcal{P}_{\mathrm{cl}}(Step \times Q)).$$

We shall use Y to range over $\mathcal{P}_{\mathrm{cl}}(Step \times Q)$ and ξ to range over $Step \times Q$.

(3) The domain R is defined as the solution of

$$R \cong (Obj \times P) \cup (\Sigma \to P_{\mathrm{cl}}(Step \times R)).$$

We shall use Z to range over $\mathcal{P}_{\mathrm{cl}}(Step \times R)$ and ζ to range over $Step \times R$.

The intuitive interpretation of Q and R is as follows. An element of Q is a process executing a specific *statement* (the "local" one), possibly in parallel with some other processes. Termination of the local statement is explicitly indicated by $\surd$. The idea is that a continuation can start at that point (see the definition of the operator ":" below). More specifically, if $q \in Q$ is of the form $\langle \surd, p \rangle$ this means that the local process terminates immediately, and that the parallel processes continue with p. If in q the local process does not terminate immediately, an ordinary step is possible, after which we come in the same situation again. Because we have also included p_0 in Q, P can be embedded in Q in a canonical way. We shall therefore assume that actually $P \subseteq Q$.

An element of R is evaluating an *expression*, again possibly in parallel with other processes. It will be composed with elements of $Obj \to Q$ or $Obj \to R$ by the operator ":". If the evaluation of the expression terminates, it delivers a value β being the result of this expression, together with an ordinary process p representing the ongoing computation of the other processes (which is to be executed in parallel with the semantic expression continuation).

We shall define four forms of the operator ":" which will take care of the composition of elements of Q and R with appropriate continuations (notice the analogy with Definition 4.7):

6.16. Definition. (1) We define ":": $Q \times Q \to Q$ by the following clauses (which can be completed to a full definition along the lines of Definition 5.5):

(a) $p_0 : q = p_0$;

(b) $\langle \surd, p \rangle : q = p \| q$ (see Definition 6.17 below);

(c) $(\lambda\sigma. Y) : q = \lambda\sigma.(Y : q)$, where $Y : q = \{\xi : q \mid \xi \in Y\}$ and $\langle \eta, q' \rangle : q = \langle \eta, q' : q \rangle$.

(2) We define ":": $Q \times R \to R$ as follows:

(a) $p_0 : r = p_0$;

(b) $\langle \surd, p \rangle : r = p \| r$ (see Definition 6.17);

(c) $(\lambda\sigma. Y) : r = \lambda\sigma.(Y : r)$, where $Y : r = \{\xi : r \mid \xi \in Y\}$ and $\langle \eta, q' \rangle : r = \langle \eta, q' : r \rangle$.

(3) The operator ":": $R \times (Obj \to Q) \to Q$ is given by the following clauses:

(a) $\langle \beta, p\rangle : f = p \| f(\beta)$;

(b) $(\lambda\sigma.Z) : f = \lambda\sigma.(Z : f)$, where $Z : f = \{\zeta : f \mid \zeta \in Z\}$ and $\langle \eta, r\rangle : f = \langle \eta, r : f\rangle$.

(4) Finally, we define the operator ":": $R \times (Obj \to R) \to R$ by the following clauses (we shall use h to range over $Obj \to R$):

(a) $\langle \beta, p\rangle : h = p \| h(\beta)$;

(b) $(\lambda\sigma.Z) : h = \lambda\sigma.(Z : h)$, where $Z : h = \{\zeta : h \mid \zeta \in Z\}$ and $\langle \eta, r\rangle : h = \langle \eta, r : h\rangle$.

Note that if $q \in P$, then $p : q \in P$, so that we also have ":": $Q \times P \to P$. Analogously, if $f \in Obj \to P$, then we get $r : f \in P$, so that we can state ":": $R \times (Obj \to P) \to P$.

We also need the definitions of $p \| q$ and $p \| r$:

6.17. Definition. (1) We define the operator "$\|$": $P \times Q \to Q$ by the following clauses:

(a) $p_0 \| q = q$, $p \| p_0 = p$, $p \| \langle \surd, p'\rangle = \langle \surd, p \| p'\rangle$;

(b) for $p \neq p_0$ and $q \notin \{p_0\} \cup (\{\surd\} \times P)$ we define

$$p \| q = \lambda\sigma.((p(\sigma) \| q) \cup (p \| q(\sigma)) \cup (p(\sigma) |_\sigma q(\sigma)));$$

(c) for $X \in \mathcal{P}_{\text{cl}}(Step \times P)$ we put $X \| q = \{\pi \| q \mid \pi \in X\}$, where $\langle \eta, p'\rangle \| q = \langle \eta, p' \| q\rangle$;

(d) for $Y \in \mathcal{P}_{\text{cl}}(Step \times Q)$ we put $p \| Y = \{p \| \xi \mid \xi \in Y\}$, where $p \| \langle \eta, q'\rangle = \langle \eta, p \| q'\rangle$;

(e) for X and Y as above, we define

$$X |_\sigma Y = \bigcup \{\pi |_\sigma \xi \mid \pi \in X, \xi \in Y\}$$

where $\langle \eta_1, p'\rangle |_\sigma \langle \eta_2, q'\rangle = \{\langle \sigma', p' \| q'\rangle\}$ with σ' as usual if η_1 and η_2 are matching communications, and $\pi |_\sigma \xi = \emptyset$ otherwise.

Note that restricted to $P \times P$ this coincides with the old operator "$\|$" (see Definition 6.7).

(2) We define the operator "$\|$": $P \times R \to R$ by the following clauses:

(a) $p_0 \| r = r$, $p \| \langle \beta, p'\rangle = \langle \beta, p \| p'\rangle$;

(b) for $p \neq p_0$ and $r \notin Obj \times P$ we define

$$p \| r = \lambda\sigma.((p(\sigma) \| r) \cup (p \| r(\sigma)) \cup (p(\sigma) |_\sigma r(\sigma)));$$

(c) for $X \in \mathcal{P}_{\text{cl}}(Step \times P)$ we put $X \| r = \{\pi \| r \mid \pi \in X\}$, where $\langle \eta, p'\rangle \| r = \langle \eta, p' \| r\rangle$;

(d) for $Z \in \mathcal{P}_{\text{cl}}(Step \times R)$ we put $p \| Z = \{p \| \zeta \mid \zeta \in Z\}$, where $p \| \langle \eta, r'\rangle = \langle \eta, p \| r'\rangle$;

(e) for X and Z as above, we define

$$X |_\sigma Z = \bigcup \{\pi |_\sigma \zeta \mid \pi \in X, \zeta \in Z\}$$

where $\langle \eta_1, p'\rangle |_\sigma \langle \eta_2, r'\rangle = \{\langle \sigma', p' \| r'\rangle\}$ with σ' as usual if η_1 and η_2 are matching communications, and $\pi |_\sigma \zeta = \emptyset$ otherwise.

Analogous to Lemma 4.8 we have the following important lemma.

6.18. Lemma. (1) *All forms of the mappings ":" and "$\|$" are continuous.*

(2) *The operators "$\|$" are associative*:

(a) $(p_1 \| p_2) \| q = p_1 \| (p_2 \| q)$,

(b) $(p_1 \| p_2) \| r = p_1 \| (p_2 \| r)$.

(3) *The operators ":" with the first argument from Q are associative*:

(a) $(q_1:q_2):q_3 = q_1:(q_2:q_3)$,

(b) $(q_1:q_2):r = q_1:(q_2:r)$.

(4) *The operators ":" with the first argument from R have an analogous property (let us call it λ-associativity)*:

(a) $(r:f):q = r:\lambda\beta.(f(\beta):q)$,

(b) $(r:f):r' = r:\lambda\beta.(f(\beta):r')$,

(c) $(r:h):f = r:\lambda\beta.(h(\beta):f)$,

(d) $(r:h):h' = r:\lambda\beta.(h(\beta):h')$.

(5) *Finally, we have a kind of distributivity*:

(a) $(p\|q):q' = p\|(q:q')$,

(b) $(p\|q):r = p\|(q:r)$,

(c) $(p\|r):f = p\|(r:f)$,

(d) $(p\|r):h = p\|(r:h)$.

Proof. Part (1) can be proved by observing that each version of ":" or "‖" is the unique fixed point of an appropriate higher-order function that maps continuous operators into continuous operators. Therefore, ":" and "‖" are themselves continuous.

For the other parts, one first proves that $p:q = p$ and $p:r = p$ for all $p \in P$, $q \in Q$, and $r \in R$. The rest of the properties are then proved in the order (2)-(5)-(3)-(4), by a metric argument. We illustrate this technique by giving the proof of part (3)(a). (We assume that part (5) has already been proved.) Consider the operators Φ and Ψ, given by $\Phi(q_1, q_2, q_3) = (q_1:q_2):q_3$ and $\Psi(q_1, q_2, q_3) = q_1:(q_2:q_3)$. Both can be seen as elements of the metric space $Q \times Q \times Q \to Q$. We shall show that $\Phi = \Psi$ by proving $d(\Phi, \Psi) = 0$. Let us therefore denote $d(\Phi, \Psi)$ by ε, or in other words,

$$\varepsilon = \sup_{q_1, q_2, q_3 \in Q} d_Q((q_1:q_2):q_3, q_1:(q_2:q_3)). \tag{6.1}$$

Now let $q_1, q_2, q_3 \in Q$ be arbitrary. We show

$$d_Q((q_1:q_2):q_3, q_1:(q_2:q_3)) \leq \tfrac{1}{2}\varepsilon. \tag{6.2}$$

Distinguish the following cases:

(1) $q_1 = p_0$. Then $(q_1:q_2):q_3 = p_0:q_3 = p_0 = q_1:(q_2:q_3)$.

(2) $q_1 = \langle\surd, p\rangle$. Then $(q_1:q_2):q_3 = (p\|q_2):q_3 =$ (by part (5)(a)) $p\|(q_2:q_3) = q_1:(q_2:q_3)$.

(3) $q_1 \in \Sigma \to \mathcal{P}_{\text{cl}}(Step \times Q)$. Now by Definition 6.16 we have that $q_1:q_2$, $(q_1:q_2):q_3$, and $q_1:(q_2:q_3)$ are also elements of $\Sigma \to \mathcal{P}_{\text{cl}}(Step \times Q)$. Let $\sigma \in \Sigma$ be arbitrary and set $Y = q_1(\sigma)$. Then we get, by Definition 6.16,

$$(q_1:q_2)(\sigma) = Y:q_2 = \{\xi:q_2 \mid \xi \in Y\} = \{\langle\eta, q':q_2\rangle \mid \langle\eta, q'\rangle \in Y\},$$

$$((q_1:q_2):q_3)(\sigma) = (Y:q_2):q_3 = \{\langle\eta, (q':q_2):q_3\rangle \mid \langle\eta, q'\rangle \in Y\},$$

and

$$(q_1:(q_2:q_3))(\sigma) = Y:(q_2:q_3) = \{\langle\eta, q':(q_2:q_3)\rangle \mid \langle\eta, q'\rangle \in Y\}.$$

Now the time has come to remember our convention from Section 2.3 that, implicitly, every occurrence in the right-hand side of the domain being defined is surrounded by $\mathrm{id}_{1/2}$ (cf. equation (2.3′)). Of course, this also holds for the defining equation for Q in Definition 6.15. From (6.1) it follows that

$$d_Q((q'\!:\!q_2)\!:\!q_3,\, q'\!:\!(q_2\!:\!q_3)) \leq \varepsilon.$$

Therefore

$$d_{\mathrm{id}_{1/2}(Q)}((q'\!:\!q_2)\!:\!q_3,\, q'\!:\!(q_2\!:\!q_3)) \leq \tfrac{1}{2}\varepsilon.$$

By applying the clauses of Definition 2.7 (and remembering that σ was arbitrary) we can conclude that

$$d_Q((q_1\!:\!q_2)\!:\!q_3,\, q_1\!:\!(q_2\!:\!q_3)) \leq \tfrac{1}{2}\varepsilon.$$

Because q_1, q_2, and q_3 were arbitrary in (6.2), we can conclude from (6.1) that $\varepsilon \leq \frac{1}{2}\varepsilon$, so that $d(\Phi, \Psi) = \varepsilon = 0$ and $\Phi = \Psi$. □

Next, we state the analogues of Lemma 4.9 and Corollary 4.10. By way of preparation we need some extensions to the definitions of *PSyCo* and $\mathcal{O}^*$.

6.19. Definition. (1) We define the set *PSyCo′*, with typical element $\dot{\rho}$, to be the same as *PSyCo*, except that *at most one* of the components has an $\dot{r} \in \mathit{SyStCo}'$, defined (together with $\dot{g} \in \mathit{SyExCo}'$) by

$$\dot{r} ::= \surd \mid s;\dot{r}' \mid e\!:\!\dot{g} \qquad \dot{g} ::= \lambda z.\dot{r}$$

with s closed.

(2) The set *PSyCo″*, with typical element $\ddot{\rho}$, is the same as *PsyCo* except that *exactly one* component has an $\ddot{r} \in \mathit{SyStCo}''$, which is defined together with $\ddot{g} \in \mathit{SyExCo}''$ by

$$\ddot{r} ::= s;\ddot{r}' \mid e\!:\!\ddot{g} \qquad \ddot{g} ::= \lambda z.\ddot{r} \mid \surd$$

with s closed.

(3) We define the function $\dot{\mathcal{O}}[\![\cdot]\!]: \mathit{PSyCo}' \to Q$ as follows

$$\dot{\mathcal{O}}[\![\dot{\rho}]\!] = \begin{cases} p_0 & \text{if } \dot{\rho} = \{\langle \alpha_1, E\rangle, \ldots, \langle \alpha_k, E\rangle\}, \\ \langle \surd, \mathcal{O}^*[\![\rho']\!]\rangle & \text{if } \dot{\rho} = \{\langle \alpha, \surd\rangle\} \cup \rho', \\ \lambda\sigma.\{\langle \sigma', \dot{\mathcal{O}}[\![\dot{\rho}']\!]\rangle \mid \langle \sigma, \dot{\rho}\rangle \to \langle \sigma', \dot{\rho}'\rangle\} & \text{otherwise.} \end{cases}$$

Here we interpret the transition relation "→" with respect to $\mathcal{T}^*_{\mathrm{nud}}$ (only extended in so far that we declare the existing axioms and rules also applicable to our new parallel syntactic continuations).

(4) We define the function $\ddot{\mathcal{O}}[\![\cdot]\!]: \mathit{PSyCo}'' \to R$ as follows

$$\ddot{\mathcal{O}}[\![\ddot{\rho}]\!] = \begin{cases} \langle \beta, \mathcal{O}^*[\![\rho']\!]\rangle & \text{if } \ddot{\rho} = \{\langle \alpha, \beta\!:\!\surd\rangle\} \cup \rho', \\ \lambda\sigma.\{\langle \sigma', \ddot{\mathcal{O}}[\![\ddot{\rho}']\!]\rangle \mid \langle \sigma, \ddot{\rho}\rangle \to \langle \sigma', \ddot{\rho}'\rangle\} & \text{otherwise.} \end{cases}$$

Note that $\mathit{PsyCo} \subseteq \mathit{PSyCo}'$, and that $\dot{\mathcal{O}}$ restricted to *PsyCo* is equal to $\mathcal{O}^*$. Furthermore, Lemma 6.13 also holds for $\dot{\mathcal{O}}$ and $\ddot{\mathcal{O}}$, and we can restate Lemma 6.11 as follows.

6.20. Lemma. (1) $\dot{\mathcal{O}}[\![\rho \cup \dot{\rho}]\!] = \mathcal{O}^*[\![\rho]\!] \| \dot{\mathcal{O}}[\![\dot{\rho}]\!]$.
(2) $\ddot{\mathcal{O}}[\![\rho \cup \ddot{\rho}]\!] = \mathcal{O}^*[\![\rho]\!] \| \ddot{\mathcal{O}}[\![\ddot{\rho}]\!]$.

Now we can state the next lemma.

6.21. Lemma. (1) *For any* $e \in Exp$, $\alpha \in AObj$, *and* $g \in SyExCo$ *we have*

$$\mathcal{O}^*[\![\{\langle\alpha, e:g\rangle\}]\!] = \ddot{\mathcal{O}}[\![\{\langle\alpha, e:\surd\rangle\}]\!]:(\lambda\beta.\mathcal{O}^*[\![\{\langle\alpha, \beta:g\rangle\}]\!])$$

and the same for any $\dot{g}$ *with* $\mathcal{O}^*$ *replaced by* $\dot{\mathcal{O}}$ *and for any* $\ddot{g}$ *with* $\mathcal{O}^*$ *replaced by* $\ddot{\mathcal{O}}$.
(2) *Let* $s \in \mathcal{S}_{\text{nud}}$ (*not necessarily closed*) *and let all free statement variables of s be contained in* $\{v_1, \ldots, v_k\}$. *Now let* $s_1, \ldots, s_k$ *be closed statements such that, for any* α *and r,*

$$\mathcal{O}^*[\![\{\langle\alpha, s_i;r\rangle\}]\!] = \dot{\mathcal{O}}[\![\{\langle\alpha, s_i;\surd\rangle\}]\!]:\mathcal{O}^*[\![\{\langle\alpha, r\rangle\}]\!]$$

and for any $\dot{r}$ *the same with* $\mathcal{O}^*$ *replaced by* $\dot{\mathcal{O}}$ *and for any* $\ddot{r}$ *the same with* $\mathcal{O}^*$ *replaced by* $\ddot{\mathcal{O}}$. *If we define* $\tilde{s} = s[s_i/v_i]_{i=1}^k$, *then we have, for any* α *and r,*

$$\mathcal{O}^*[\![\{\langle\alpha, \tilde{s};r\rangle\}]\!] = \dot{\mathcal{O}}[\![\{\langle\alpha, \tilde{s};\surd\rangle\}]\!]:\mathcal{O}^*[\![\{\langle\alpha, r\rangle\}]\!]$$

and analogously for any $\dot{r}$ *and for any* $\ddot{r}$.

Proof. Part (1) is proved by induction on the complexity of e. We give some typical cases:

Case 1: $e = \beta$.

$$\begin{aligned}
&\ddot{\mathcal{O}}[\![\{\langle\alpha, \beta:\surd\rangle\}]\!]:(\lambda\beta'.\mathcal{O}^*[\![\{\langle\alpha, \beta':g\rangle\}]\!])\\
&\quad = \langle\beta, p_0\rangle:(\lambda\beta'.\mathcal{O}^*[\![\{\langle\alpha, \beta':g\rangle\}]\!]) && \text{(Definition 6.19)}\\
&\quad = p_0 \| \mathcal{O}^*[\![\{\langle\alpha, \beta:g\rangle\}]\!] && \text{(Definition 6.16)}\\
&\quad = \mathcal{O}^*[\![\{\langle\alpha, \beta:g\rangle\}]\!] && \text{(Definition 6.7).}
\end{aligned}$$

Exactly the same proof works for $\dot{g}$ with $\dot{\mathcal{O}}$ and for $\ddot{g}$ with $\ddot{\mathcal{O}}$.

Case 2: $e = \mathbf{op}\ e'$.

$$\begin{aligned}
&\mathcal{O}^*[\![\{\langle\alpha, (\mathbf{op}\ e'):g\rangle\}]\!]\\
&\quad = \mathcal{O}^*[\![\{\langle\alpha, e':\lambda z.(\mathbf{op}\ z:g)\rangle\}]\!] && \text{(Lemma 6.13(12))}\\
&\quad = \ddot{\mathcal{O}}[\![\{\langle\alpha, e':\surd\rangle\}]\!]:(\lambda\beta'.\mathcal{O}^*[\![\{\langle\alpha, \beta':\lambda z.(\mathbf{op}\ z:g)\rangle\}]\!]) && \text{(ind. hyp.)}\\
&\quad = \ddot{\mathcal{O}}[\![\{\langle\alpha, e':\surd\rangle\}]\!]:(\lambda\beta'.\mathcal{O}^*[\![\{\langle\alpha, \mathbf{op}\ \beta':g\rangle\}]\!]) && \text{(Lemma 6.13(10))}\\
&\quad = \ddot{\mathcal{O}}[\![\{\langle\alpha, e':\surd\rangle\}]\!]:(\lambda\beta'.\mathcal{O}^*[\![\{\langle\alpha, \mathbf{op}_{\text{sem}}\ \beta':g\rangle\}]\!]) && \text{(Lemma 6.13(11))}\\
&\quad = \ddot{\mathcal{O}}[\![\{\langle\alpha, e':\surd\rangle\}]\!]:(\lambda\beta'.\ddot{\mathcal{O}}[\![\{\langle\alpha, \mathbf{op}_{\text{sem}}\ \beta':\surd\rangle\}]\!]:(\lambda\beta.\mathcal{O}^*[\![\{\langle\alpha, \beta:g\rangle\}]\!]))\\
& && \text{(Case 1)}\\
&\quad = (\ddot{\mathcal{O}}[\![\{\langle\alpha, e':\surd\rangle\}]\!]:(\lambda\beta'.\ddot{\mathcal{O}}[\![\{\langle\alpha, \mathbf{op}_{\text{sem}}\ \beta':\surd\rangle\}]\!])):(\lambda\beta.\mathcal{O}^*[\![\{\langle\alpha, \beta:g\rangle\}]\!])\\
& && \text{(Lemma 6.18(4))}
\end{aligned}$$

$$= (\ddot{\mathcal{O}}[\![\{\langle\alpha, e'{:}\surd\rangle\}]\!] : (\lambda\beta'.\ddot{\mathcal{O}}[\![\{\langle\alpha, \beta'{:}\lambda z.(\mathbf{op}\ z : \surd)\rangle\}]\!])) : (\lambda\beta.\mathcal{O}^*[\![\{\langle\alpha, \beta{:}g\rangle\}]\!])$$
$$\text{(Lemma 6.13(11, 10))}$$

$$= \ddot{\mathcal{O}}[\![\{\langle\alpha, e'{:}\lambda z.(\mathbf{op}\ z : \surd)\rangle\}]\!] : (\lambda\beta.\mathcal{O}^*[\![\{\langle\alpha, \beta{:}g\rangle\}]\!]) \qquad \text{(ind. hyp.)}$$

$$= \ddot{\mathcal{O}}[\![\{\langle\alpha, \mathbf{op}\ e' : \surd\rangle\}]\!] : (\lambda\beta.\mathcal{O}^*[\![\{\langle\alpha, \beta{:}g\rangle\}]\!]). \qquad \text{(Lemma 6.13(12))}$$

Again, the proof is also valid for $\dot{g}$ and $\ddot{g}$.

Case 3: $e = \mathbf{new}(c)$.

$$\mathcal{O}^*[\![\{\langle\alpha, \mathbf{new}(c){:}g\rangle\}]\!]$$

$$= \lambda\sigma.\{\langle\sigma', \mathcal{O}^*[\![\{\langle\alpha, \beta{:}g\rangle, \langle\beta, s;E\rangle\}]\!]\rangle\}$$
$$\text{(Lemma 6.13(13), with } s, \sigma', \text{ and } \beta \text{ as usual)}$$

$$= \lambda\sigma.\{\langle\sigma', \mathcal{O}^*[\![\{\langle\beta, s;E\rangle\}]\!] \parallel \mathcal{O}^*[\![\{\langle\alpha, \beta{:}g\rangle\}]\!]\rangle\} \qquad \text{(Lemma 6.11)}$$

$$= \lambda\sigma.\{\langle\sigma', \mathcal{O}^*[\![\{\langle\beta, s;E\rangle\}]\!] \parallel (\ddot{\mathcal{O}}[\![\{\langle\alpha, \beta{:}\surd\rangle\}]\!] : (\lambda\beta'.\mathcal{O}^*[\![\{\langle\alpha, \beta'{:}g\rangle\}]\!]))\rangle\}$$
$$\text{(Case 1)}$$

$$= \lambda\sigma.\{\langle\sigma', (\mathcal{O}^*[\![\{\langle\beta, s;E\rangle\}]\!] \parallel \ddot{\mathcal{O}}[\![\{\langle\alpha, \beta{:}\surd\rangle\}]\!]) : (\lambda\beta'.\mathcal{O}^*[\![\{\langle\alpha, \beta'{:}g\rangle\}]\!])\rangle\}$$
$$\text{(Lemma 6.18(5))}$$

$$= \lambda\sigma.\{\langle\sigma', \mathcal{O}^*[\![\{\langle\beta, s;E\rangle\}]\!] \parallel \ddot{\mathcal{O}}[\![\{\langle\alpha, \beta{:}\surd\rangle\}]\!]\rangle\} : (\lambda\beta'.\mathcal{O}^*[\![\{\langle\alpha, \beta'{:}g\rangle\}]\!])$$
$$\text{(Definition 6.16)}$$

$$= \lambda\sigma.\{\langle\sigma', \ddot{\mathcal{O}}[\![\{\langle\beta, s;E\rangle, \langle\alpha, \beta{:}\surd\rangle\}]\!]\rangle\} : (\lambda\beta'.\mathcal{O}^*[\![\{\langle\alpha, \beta'{:}g\rangle\}]\!])$$
$$\text{(Lemma 6.20)}$$

$$= \ddot{\mathcal{O}}[\![\{\langle\alpha, \mathbf{new}(c){:}\surd\rangle\}]\!] : (\lambda\beta'.\mathcal{O}^*[\![\{\langle\alpha, \beta'{:}g\rangle\}]\!]). \qquad \text{(Lemma 6.13(13))}$$

Once again, the proof is also valid for $\dot{g}$ and $\ddot{g}$.

Now we can prove part (2) by induction on the complexity of s. Again some typical cases:

Case 4: $s = x := e$ (so $\tilde{s} = s$).

$$\mathcal{O}^*[\![\{\langle\alpha, x := e;r\rangle\}]\!]$$

$$= \mathcal{O}^*[\![\{\langle\alpha, e{:}\lambda z.(x := z;r)\rangle\}]\!] \qquad \text{(Lemma 6.13(2))}$$

$$= \ddot{\mathcal{O}}[\![\{\langle\alpha, e{:}\surd\rangle\}]\!] : (\lambda\beta.\mathcal{O}^*[\![\{\langle\alpha, \beta{:}\lambda z.(x := z;r)\rangle\}]\!]) \qquad \text{(part (1))}$$

$$= \ddot{\mathcal{O}}[\![\{\langle\alpha, e{:}\surd\rangle\}]\!] : (\lambda\beta.\mathcal{O}^*[\![\{\langle\alpha, x := \beta;r\rangle\}]\!]) \qquad \text{(Lemma 6.13(10))}$$

$$= \ddot{\mathcal{O}}[\![\{\langle\alpha, e{:}\surd\rangle\}]\!] : (\lambda\beta.\lambda\sigma.\{\langle\sigma', \mathcal{O}^*[\![\{\langle\alpha, r\rangle\}]\!]\rangle\}) \qquad \text{(Lemma 6.13(1), } \sigma' \text{ as usual)}$$

$$= \ddot{\mathcal{O}}[\![\{\langle\alpha, e{:}\surd\rangle\}]\!] : (\lambda\beta.\lambda\sigma.\{\langle\sigma', \dot{\mathcal{O}}[\![\{\langle\alpha, \surd\rangle\}]\!] : \mathcal{O}^*[\![\{\langle\alpha, r\rangle\}]\!]\rangle\})$$
$$\text{(because } \dot{\mathcal{O}}[\![\{\langle\alpha, \surd\rangle\}]\!] = \langle\surd, p_0\rangle \text{ and } \langle\surd, p_0\rangle : q = p_0 \| q = q)$$

$$= \ddot{\mathcal{O}}[\![\{\langle\alpha, e{:}\surd\rangle\}]\!] : (\lambda\beta.\lambda\sigma.\{\langle\sigma', \dot{\mathcal{O}}[\![\{\langle\alpha, \surd\rangle\}]\!]\rangle\} : \mathcal{O}^*[\![\{\langle\alpha, r\rangle\}]\!])$$
$$\text{(Definition 6.16)}$$

$$= \ddot{\mathcal{O}}\llbracket\{\langle\alpha, e{:}\surd\rangle\}\rrbracket : (\lambda\beta.\dot{\mathcal{O}}\llbracket\{\langle\alpha, x := \beta;\surd\rangle\}\rrbracket : \mathcal{O}^*\llbracket\{\langle\alpha, r\rangle\}\rrbracket) \qquad \text{(Lemma 6.13(1))}$$

$$= \ddot{\mathcal{O}}\llbracket\{\langle\alpha, e{:}\surd\rangle\}\rrbracket : (\lambda\beta.\dot{\mathcal{O}}\llbracket\{\langle\alpha, \beta{:}\lambda z.(x := z;\surd)\rangle\}\rrbracket : \mathcal{O}^*\llbracket\{\langle\alpha, r\rangle\}\rrbracket) \qquad \text{(Lemma 6.13(10))}$$

$$= (\ddot{\mathcal{O}}\llbracket\{\langle\alpha, e{:}\surd\rangle\}\rrbracket : (\lambda\beta.\dot{\mathcal{O}}\llbracket\{\langle\alpha, \beta{:}\lambda z.(x := z;\surd)\rangle\}\rrbracket)) : \mathcal{O}^*\llbracket\{\langle\alpha, r\rangle\}\rrbracket \qquad \text{(Lemma 6.18(4))}$$

$$= \dot{\mathcal{O}}\llbracket\{\langle\alpha, e{:}\lambda z.(x := z;\surd)\rangle\}\rrbracket : \mathcal{O}^*\llbracket\{\langle\alpha, r\rangle\}\rrbracket \qquad \text{(part (1))}$$

$$= \dot{\mathcal{O}}\llbracket\{\langle\alpha, x := e;\surd\rangle\}\rrbracket : \mathcal{O}^*\llbracket\{\langle\alpha, r\rangle\}\rrbracket. \qquad \text{(Lemma 6.13(2))}$$

For $\dot{r}$ or $\ddot{r}$ instead of r the proof runs exactly the same.

Case 5: $s = e?x$ (so $\tilde{s} = s$).

$$\mathcal{O}^*\llbracket\{\langle\alpha, e?x;r\rangle\}\rrbracket$$

$$= \mathcal{O}^*\llbracket\{\langle\alpha, e{:}\lambda z.(z?x;r)\rangle\}\rrbracket \qquad \text{(Lemma 6.13(6))}$$

$$= \ddot{\mathcal{O}}\llbracket\{\langle\alpha, e{:}\surd\rangle\}\rrbracket : (\lambda\beta.\mathcal{O}^*\llbracket\{\langle\alpha, \beta{:}\lambda z.(z?x;r)\rangle\}\rrbracket) \qquad \text{(part (1))}$$

$$= \ddot{\mathcal{O}}\llbracket\{\langle\alpha, e{:}\surd\rangle\}\rrbracket : (\lambda\beta.\mathcal{O}^*\llbracket\{\langle\alpha, \beta?x;r\rangle\}\rrbracket) \qquad \text{(Lemma 6.13(10))}$$

$$= \ddot{\mathcal{O}}\llbracket\{\langle\alpha, e{:}\surd\rangle\}\rrbracket : (\lambda\beta.\lambda\sigma.\{\langle\alpha, \beta?x, \mathcal{O}^*\llbracket\{\langle\alpha, r\rangle\}\rrbracket\rangle\}) \qquad \text{(Lemma 6.13(7))}$$

$$= \ddot{\mathcal{O}}\llbracket\{\langle\alpha, e{:}\surd\rangle\}\rrbracket : (\lambda\beta.\lambda\sigma.\{\langle\alpha, \beta?x, \dot{\mathcal{O}}\llbracket\{\langle\alpha, \surd\rangle\}\rrbracket : \mathcal{O}^*\llbracket\{\langle\alpha, r\rangle\}\rrbracket\rangle\}) \qquad \text{(see above)}$$

$$= \ddot{\mathcal{O}}\llbracket\{\langle\alpha, e{:}\surd\rangle\}\rrbracket : (\lambda\beta.\lambda\sigma.\{\langle\alpha, \beta?x, \dot{\mathcal{O}}\llbracket\{\langle\alpha, \surd\rangle\}\rrbracket\rangle\} : \mathcal{O}^*\llbracket\{\langle\alpha, r\rangle\}\rrbracket) \qquad \text{(Definition 6.16)}$$

$$= \ddot{\mathcal{O}}\llbracket\{\langle\alpha, e{:}\surd\rangle\}\rrbracket : (\lambda\beta.\dot{\mathcal{O}}\llbracket\{\langle\alpha, \beta{:}\lambda z.(z?x;\surd)\rangle\}\rrbracket : \mathcal{O}^*\llbracket\{\langle\alpha, r\rangle\}\rrbracket) \qquad \text{(Lemma 6.13(7, 10))}$$

$$= (\ddot{\mathcal{O}}\llbracket\{\langle\alpha, e{:}\surd\rangle\}\rrbracket : (\lambda\beta.\dot{\mathcal{O}}\llbracket\{\langle\alpha, \beta{:}\lambda z.(z?x;\surd)\rangle\}\rrbracket)) : \mathcal{O}^*\llbracket\{\langle\alpha, r\rangle\}\rrbracket \qquad \text{(Lemma 6.18(4))}$$

$$= \dot{\mathcal{O}}\llbracket\{\langle\alpha, e{:}\lambda z.(z?x;\surd)\rangle\}\rrbracket : \mathcal{O}^*\llbracket\{\langle\alpha, r\rangle\}\rrbracket \qquad \text{(part (1))}$$

$$= \dot{\mathcal{O}}\llbracket\{\langle\alpha, e?x;\surd\rangle\}\rrbracket : \mathcal{O}^*\llbracket\{\langle\alpha, r\rangle\}\rrbracket. \qquad \text{(Lemma 6.13(6))}$$

Case 6: $s = \mu v[s']$. Without loss of generality we can assume that $v \notin \{v_1, \ldots, v_k\}$. If we define $\tilde{s}' = s'[s_i/v_i]_{i=1}^{k}$, then we have $\tilde{s} = \mu v[\tilde{s}']$. Now we first prove, by induction on n, that for any α and r (and also for $\dot{r}$),

$$\mathcal{O}^*\llbracket\{\langle\alpha, \tilde{s}_v'^{(n)};r\rangle\}\rrbracket = \dot{\mathcal{O}}\llbracket\{\langle\alpha, \tilde{s}_v'^{(n)};\surd\rangle\}\rrbracket : \mathcal{O}^*\llbracket\{\langle\alpha, r\rangle\}\rrbracket. \qquad (6.3)$$

For $n = 0$, we get $\tilde{s}_v'^{(0)} = \textbf{skip}$ and

$$\mathcal{O}^*\llbracket\{\langle\alpha, \textbf{skip};r\rangle\}\rrbracket$$

$$
\begin{aligned}
&= \lambda\sigma.\{\langle\sigma, \mathcal{O}^*\llbracket\{\langle\alpha, r\rangle\}\rrbracket\rangle\} && \text{(definition of } \textbf{skip}\text{)}\\
&= \lambda\sigma.\{\langle\sigma, \dot{\mathcal{O}}\llbracket\{\langle\alpha, \surd\rangle\}\rrbracket : \mathcal{O}^*\llbracket\{\langle\alpha, r\rangle\}\rrbracket\rangle\} && \text{(see above)}\\
&= \lambda\sigma.\{\langle\sigma, \dot{\mathcal{O}}\llbracket\{\langle\alpha, \surd\rangle\}\rrbracket\rangle\} : \mathcal{O}^*\llbracket\{\langle\alpha, r\rangle\}\rrbracket && \text{(Definition 6.16)}\\
&= \dot{\mathcal{O}}\llbracket\{\langle\alpha, \textbf{skip};\surd\rangle\}\rrbracket : \mathcal{O}^*\llbracket\{\langle\alpha, r\rangle\}\rrbracket && \text{(definition of } \textbf{skip}\text{).}
\end{aligned}
$$

Now let us assume (6.3) for certain n; then we can apply the outer induction hypothesis for s', with $v_{k+1} = v$ and $s_{k+1} = \tilde{s}_v'^{(n)}$. If we define $\hat{s}_v'^{(n)} = \tilde{s}'[\tilde{s}_v'^{(n)}/v] = s'[s_i/v_i]_{i=1}^{k+1}$, this gives us

$$
\mathcal{O}^*\llbracket\{\langle\alpha, \hat{s}_v'^{(n)};r\rangle\}\rrbracket = \dot{\mathcal{O}}\llbracket\{\langle\alpha, \hat{s}_v'^{(n)};\surd\rangle\}\rrbracket : \mathcal{O}^*\llbracket\{\langle\alpha, r\rangle\}\rrbracket. \tag{6.4}
$$

Now we can calculate

$$
\begin{aligned}
&\mathcal{O}^*\llbracket\{\langle\alpha, \tilde{s}_v'^{(n+1)};r\rangle\}\rrbracket\\
&\quad= \mathcal{O}^*\llbracket\{\langle\alpha, (\textbf{skip};\hat{s}_v'^{(n)});r\rangle\}\rrbracket\\
&\quad= \mathcal{O}^*\llbracket\{\langle\alpha, \textbf{skip};(\hat{s}_v'^{(n)};r)\rangle\}\rrbracket && \text{(Lemma 6.13(3))}\\
&\quad= \lambda\sigma.\{\langle\sigma, \mathcal{O}^*\llbracket\{\langle\alpha, \hat{s}_v'^{(n)};r\rangle\}\rrbracket\rangle\} && \text{(definition of } \textbf{skip}\text{)}\\
&\quad= \lambda\sigma.\{\langle\sigma, \dot{\mathcal{O}}\llbracket\{\langle\alpha, \hat{s}_v'^{(n)};\surd\rangle\}\rrbracket : \mathcal{O}^*\llbracket\{\langle\alpha, r\rangle\}\rrbracket\rangle\} && \text{(by (6.4))}\\
&\quad= \lambda\sigma.\{\langle\sigma, \dot{\mathcal{O}}\llbracket\{\langle\alpha, \hat{s}_v'^{(n)};\surd\rangle\}\rrbracket\rangle\} : \mathcal{O}^*\llbracket\{\langle\alpha, r\rangle\}\rrbracket && \text{(Definition 6.16)}\\
&\quad= \dot{\mathcal{O}}\llbracket\{\langle\alpha, \textbf{skip};(\hat{s}_v'^{(n)};\surd)\rangle\}\rrbracket : \mathcal{O}^*\llbracket\{\langle\alpha, r\rangle\}\rrbracket && \text{(definition of } \textbf{skip}\text{)}\\
&\quad= \dot{\mathcal{O}}\llbracket\{\langle\alpha, (\textbf{skip};\, \hat{s}_v'^{(n)});\surd\rangle\}\rrbracket : \mathcal{O}^*\llbracket\{\langle\alpha, r\rangle\}\rrbracket && \text{(Lemma 6.13(3))}\\
&\quad= \dot{\mathcal{O}}\llbracket\{\langle\alpha, \tilde{s}_v'^{(n+1)};\surd\rangle\}\rrbracket : \mathcal{O}^*\llbracket\{\langle\alpha, r\rangle\rrbracket
\end{aligned}
$$

which gives us (6.3) for $n+1$.

Finally, we can compute as follows:

$$
\begin{aligned}
&\mathcal{O}^*\llbracket\{\langle\alpha, \mu v[\tilde{s}'];r\rangle\}\rrbracket\\
&\quad= \lim_n \mathcal{O}^*\llbracket\{\langle\alpha, \tilde{s}_v'^{(n)};r\rangle\}\rrbracket && \text{(Lemma 6.13(5))}\\
&\quad= \lim_n(\dot{\mathcal{O}}\llbracket\{\langle\alpha, \tilde{s}_v'^{(n)};\surd\rangle\}\rrbracket : \mathcal{O}^*\llbracket\{\langle\alpha, r\rangle\}\rrbracket) && \text{(by (6.3))}\\
&\quad= (\lim_n \dot{\mathcal{O}}\llbracket\{\langle\alpha, \tilde{s}_v'^{(n)};\surd\rangle\}\rrbracket) : \mathcal{O}^*\llbracket\{\langle\alpha, r\rangle\}\rrbracket && \text{(Lemma 6.18(1))}\\
&\quad= \dot{\mathcal{O}}\llbracket\{\langle\alpha, \mu v[\tilde{s}'];\surd\rangle\}\rrbracket : \mathcal{O}^*\llbracket\{\langle\alpha, r\rangle\}\rrbracket && \text{(Lemma 6.13(5)).} \quad \square
\end{aligned}
$$

In order to prove Theorem 6.14, in addition to the reasoning encountered earlier, there is one extra step necessary to deal with the possible recursion in declarations such as $c \Leftarrow \ldots$ **new**(c) .... This step involves the second component $\gamma_{(2)}$ of an environment γ. For simplicity's sake we again drop the indices.

6.22. Lemma. *Let t be a fixed program. If $\gamma \in \Gamma$ satisfies*

$$\gamma(c) = \lambda\alpha.\mathcal{O}^*[\![\{\langle\alpha, s; E\rangle\}]\!] \tag{6.5}$$

for $c \Leftarrow s$ in t, then we have the following:

(1) *For any $e \in Exp$, $\gamma \in \Gamma$, $\alpha \in AObj$, and $f \in Obj \to P$ we have*

$$\mathscr{E}[\![e]\!]\gamma\alpha f = \ddot{\mathcal{O}}[\![\{\langle\alpha, e{:}\surd\rangle\}]\!]{:}f$$

(2) *Let $s \in \mathscr{S}_{\text{nud}}$ (not necessarily closed) and assume that the free statement variables in s are all in $\{v_1, \ldots, v_k\}$ and let $s_1, \ldots, s_k$ be closed. Put $\tilde{s} = s[s_i/v_i]_{i=1}^k$ and define*

$$\varphi_i = \lambda\alpha.\lambda p.(\dot{\mathcal{O}}[\![\{\langle\alpha, s_i; \surd\rangle\}]\!]{:}p) \quad (i = 1, \ldots, k)$$

and let $\tilde{\gamma} = \gamma\{\varphi_i/v_i\}_{i=1}^k$. Then we have, for any α and p,

$$\mathscr{D}[\![s]\!]\tilde{\gamma}\alpha p = \dot{\mathcal{O}}[\![\{\langle\alpha, \tilde{s}; \surd\rangle\}]\!]{:}p.$$

Proof. The proof follows the same line of argument as in Sections 4 and 5. It runs by induction on the complexity of e and s. We make use of Lemmas 6.13, 6.18, 6.20, and 6.21 and we need the assumption (6.5) to deal with the case $e = \mathsf{new}(c)$.

We shall deal with some typical cases here, starting with part (1).

Case 1: $e = \beta$.

$$\begin{aligned}
\mathscr{E}[\![\beta]\!]\gamma\alpha f &= f(\beta) && \text{(Definition 6.9)}\\
&= p_0 \| f(\beta) && \text{(Definition 6.17)}\\
&= \langle\beta, p_0\rangle{:}f && \text{(Definition 6.16)}\\
&= \ddot{\mathcal{O}}[\![\{\langle\alpha, \beta{:}\surd\rangle\}]\!]{:}f && \text{(Definition 6.19).}
\end{aligned}$$

Case 2: $e = \mathbf{op}\ e'$.

$$\begin{aligned}
&\mathscr{E}[\![\mathbf{op}\ e']\!]\gamma\alpha f\\
&\quad= \mathscr{E}[\![e']\!]\gamma\alpha(\lambda\beta.f(\mathbf{op}_{\text{sem}}\,\beta)) && \text{(Definition 6.9)}\\
&\quad= \mathscr{E}[\![e']\!]\gamma\alpha(\lambda\beta.\ddot{\mathcal{O}}[\![\{\langle\alpha, \mathbf{op}_{\text{sem}}\,\beta{:}\surd\rangle\}]\!]{:}f) && \text{(Case 1 for } \mathbf{op}_{\text{sem}}\,\beta)\\
&\quad= \mathscr{E}[\![e']\!]\gamma\alpha(\lambda\beta.\ddot{\mathcal{O}}[\![\{\langle\alpha, \beta{:}\lambda z.(\mathbf{op}\ z{:}\surd)\rangle\}]\!]{:}f) && \text{(Lemma 6.13(11, 10))}\\
&\quad= \ddot{\mathcal{O}}[\![\{\langle\alpha, e'{:}\surd\rangle\}]\!]{:}(\lambda\beta.\ddot{\mathcal{O}}[\![\{\langle\alpha, \beta{:}\lambda z.(\mathbf{op}\ z{:}\surd)\rangle\}]\!]{:}f) && \text{(ind. hyp.)}\\
&\quad= (\ddot{\mathcal{O}}[\![\{\langle\alpha, e'{:}\surd\rangle\}]\!]{:}(\lambda\beta.\ddot{\mathcal{O}}[\![\{\langle\alpha, \beta{:}\lambda z.(\mathbf{op}\ z{:}\surd)\rangle\}]\!])){:}f && \text{(Lemma 6.18(4))}\\
&\quad= \ddot{\mathcal{O}}[\![\{\langle\alpha, e'{:}\lambda z.(\mathbf{op}\ z{:}\surd)\rangle\}]\!]{:}f && \text{(Lemma 6.21)}\\
&\quad= \ddot{\mathcal{O}}[\![\{\langle\alpha, \mathbf{op}\ e'{:}\surd\rangle\}]\!]{:}f && \text{(Lemma 6.13(12))}
\end{aligned}$$

Case 3: $e = \mathsf{new}(c)$.

$$
\begin{aligned}
&\mathscr{E}[\![\mathbf{new}(c)]\!]\gamma\alpha f \\
&\quad= \lambda\sigma.\{\langle\sigma', \gamma(c)(\beta)\|f(\beta)\rangle\} && \text{(Definition 6.9, with } \sigma' \text{ and } \beta \text{ as usual)}\\
&\quad= \lambda\sigma.\{\langle\sigma', \gamma(c)(\beta)\,\|\,(\ddot{\mathcal{O}}[\![\{\langle\alpha, \beta{:}\surd\rangle\}]\!]{:}f)\rangle\} && \text{(see Case 1)}\\
&\quad= \lambda\sigma.\{\langle\sigma', (\gamma(c)(\beta)\,\|\,\ddot{\mathcal{O}}[\![\{\langle\alpha, \beta{:}\surd\rangle\}]\!]){:}f\rangle\} && \text{(Lemma 6.18(5))}\\
&\quad= \lambda\sigma.\{\langle\sigma', (\gamma(c)(\beta)\,\|\,\ddot{\mathcal{O}}[\![\{\langle\alpha, \beta{:}\surd\rangle\}]\!])\rangle\}{:}f && \text{(Definition 6.16)}\\
&\quad= \lambda\sigma.\{\langle\sigma', (\mathcal{O}^*[\![\{\langle\beta, s;E\rangle\}]\!]\,\|\,\ddot{\mathcal{O}}[\![\{\langle\alpha, \beta{:}\surd\rangle\}]\!])\rangle\}{:}f && \text{(by (6.5))}\\
&\quad= \lambda\sigma.\{\langle\sigma', \ddot{\mathcal{O}}[\![\{\langle\beta, s;E\rangle, \langle\alpha, \beta{:}\surd\rangle\}]\!]\rangle\}{:}f && \text{(Lemma 6.21)}\\
&\quad= \ddot{\mathcal{O}}[\![\{\langle\alpha, \mathbf{new}(c){:}\surd\rangle\}]\!]{:}f. && \text{(Lemma 6.13(13))}
\end{aligned}
$$

And now part (2). Again we deal with a few typical cases.
Case 4: $s = x := e$, so $\tilde{s} = s$.

$$
\begin{aligned}
&\mathscr{D}[\![x := e]\!]\tilde{\gamma}\alpha p \\
&\quad= \mathscr{E}[\![e]\!]\tilde{\gamma}\alpha(\lambda\beta.\lambda\sigma.\{\langle\sigma', p\rangle\}) && \text{(Definition 6.9, with } \sigma' \text{ as usual)}\\
&\quad= \mathscr{E}[\![e]\!]\tilde{\gamma}\alpha(\lambda\beta.\dot{\mathcal{O}}[\![\{\langle\alpha, \beta{:}\lambda z.(x := z;\surd)\rangle\}]\!]{:}p) \\
& && \text{(see proof of Lemma 6.21, Case 4)}\\
&\quad= \ddot{\mathcal{O}}[\![\{\langle\alpha, e{:}\surd\rangle\}]\!]:(\lambda\beta.\dot{\mathcal{O}}[\![\{\langle\alpha, \beta{:}\lambda z.(x := z;\surd)\rangle\}]\!]{:}p) && \text{(part (1))}\\
&\quad= (\ddot{\mathcal{O}}[\![\{\langle\alpha, e{:}\surd\rangle\}]\!]:(\lambda\beta.\dot{\mathcal{O}}[\![\{\langle\alpha, \beta{:}\lambda z.(x := z;\surd)\rangle\}]\!])){:}p && \text{(Lemma 6.18(4))}\\
&\quad= \dot{\mathcal{O}}[\![\{\langle\alpha, e{:}\lambda z.(x := z;\surd)\rangle\}]\!]{:}p && \text{(Lemma 6.21)}\\
&\quad= \dot{\mathcal{O}}[\![\{\langle\alpha, x := e;\surd\rangle\}]\!]{:}p. && \text{(Lemma 6.13(2))}
\end{aligned}
$$

Case 5: $s = \mu v[s']$. Let us assume again that $v \notin \{v_1, \ldots, v_k\}$, so that, if we define $\tilde{s}' = s'[s_i/v_i]_{i=1}^{k}$, then we have $\tilde{s} = \mu v[\tilde{s}']$. Now, on the one hand, we have, by Lemma 6.13(5) and Lemma 6.18(1), that

$$\dot{\mathcal{O}}[\![\{\langle\alpha, \tilde{s};\surd\rangle\}]\!]{:}p = \lim_n(\dot{\mathcal{O}}[\![\{\langle\alpha, \tilde{s}_v'^{(n)};\surd\rangle\}]\!]{:}p). \tag{6.6}$$

On the other hand, Definition 6.9 says that

$$\mathscr{D}[\![s]\!]\tilde{\gamma}\alpha p = \lim_n \psi_n(\alpha)(p) \tag{6.7}$$

where ψ_0 can be chosen arbitrarily, and

$$\psi_{n+1} = \lambda\alpha.\lambda p.\lambda\sigma.\{\langle\sigma, \mathscr{D}[\![s']\!]\tilde{\gamma}\{\psi_n/v\}\alpha p\rangle\}.$$

Now we make a definite choice for ψ_0, namely

$$\psi_0 = \lambda\alpha.\lambda p.(\dot{\mathcal{O}}[\![\{\langle\alpha, \tilde{s}_v'^{(0)};\surd\rangle\}]\!]{:}p)$$

and we prove, by induction on n, that

$$\psi_n = \lambda\alpha.\lambda p.(\dot{\mathcal{O}}[\![\{\langle\alpha, \tilde{s}_v'^{(n)};\surd\rangle\}]\!]{:}p). \tag{6.8}$$

For $n=0$ this is obvious, so assume (6.8) for some n; then we can apply the outer induction hypothesis to s' with $v_{k+1}=v$ and $s_{k+1}=\tilde{s}'^{(n)}_v$, so our inner induction hypothesis (6.8) says that $\varphi_{k+1}=\psi_n$. We then get (because $s'[s_i/v_i]_{i=1}^{k+1}=\tilde{s}'[\tilde{s}'^{(n)}_v/v]$)

$$\mathcal{D}\llbracket s'\rrbracket\tilde{\gamma}\{\psi_n/v\}\alpha p=\dot{\mathcal{O}}\llbracket\{\langle\alpha,\tilde{s}'[\tilde{s}'^{(n)}_v/v];\surd\rangle\}\rrbracket{:}p \tag{6.9}$$

and we calculate

$$\begin{aligned}
\psi_{n+1}(\alpha)(p)&=\lambda\sigma.\{\langle\sigma,\mathcal{D}\llbracket s'\rrbracket\tilde{\gamma}\{\psi_n/v\}\alpha p\rangle\} && \text{(definition of } \psi_{n+1})\\
&=\lambda\sigma.\{\langle\sigma,\dot{\mathcal{O}}\llbracket\{\langle\alpha,\tilde{s}'[\tilde{s}'^{(n)}_v/v];\surd\rangle\}\rrbracket{:}p\rangle\} && \text{(by (6.9))}\\
&=\lambda\sigma.\{\langle\sigma,\dot{\mathcal{O}}\llbracket\{\langle\alpha,\tilde{s}'[\tilde{s}'^{(n)}_v/v];\surd\rangle\}\rrbracket\rangle\}{:}p && \text{(Definition 6.16)}\\
&=\dot{\mathcal{O}}\llbracket\{\langle\alpha,\mathbf{skip};(\tilde{s}'[\tilde{s}'^{(n)}_v/v];\surd)\rangle\}\rrbracket{:}p && \text{(definition of } \mathbf{skip})\\
&=\dot{\mathcal{O}}\llbracket\{\langle\alpha,(\mathbf{skip};\tilde{s}'[\tilde{s}'^{(n)}_v/v]);\surd\rangle\}\rrbracket{:}p && \text{(Lemma 6.13(3))}\\
&=\dot{\mathcal{O}}\llbracket\{\langle\alpha,\tilde{s}'^{(n+1)}_v;\surd\rangle\}\rrbracket{:}p && \text{(definition of } \tilde{s}'^{(n+1)}_v).
\end{aligned}$$

Finally, (6.8) tells us that in (6.6) and (6.7) we are taking the limit of the same sequence, so their respective left-hand sides are equal. □

One more step is necessary before we reach the desired conclusion.

6.23. Lemma. *Let γ_t be as in Definition 6.9(3). Then we have that γ_t satisfies (6.5).*

Proof. Choose any γ satisfying (6.5). Then, by the definition of Ψ_t (in Definition 6.9(3)), we have, for $c\Leftarrow s$ in t,

$$\begin{aligned}
\Psi_t(\gamma)(c)&=\lambda\alpha.\mathcal{D}\llbracket s\rrbracket(\gamma)(\alpha)(p_0)\\
&=\lambda\alpha.(\dot{\mathcal{O}}\llbracket\{\langle\alpha,s;\surd\rangle\}\rrbracket{:}p_0) && \text{(Lemma 6.22)}\\
&=\lambda\alpha.(\dot{\mathcal{O}}\llbracket\{\langle\alpha,s;\surd\rangle\}\rrbracket{:}\mathcal{O}\llbracket\{\langle\alpha,E\rangle\}\rrbracket) && \text{(Definition 6.19)}\\
&=\lambda\alpha.\mathcal{O}^*\llbracket\{\langle\alpha,s{:}E\rangle\}\rrbracket && \text{(Lemma 6.21)}\\
&=\gamma && \text{(by (6.5)).}
\end{aligned}$$

If we have furthermore that $\gamma(c)=\lambda\alpha.p_0$ for c not declared in t, then we have that γ is a fixed point of Ψ_t, so that $\gamma=\gamma_t$. □

Now we can prove Theorem 6.14:

Proof of Theorem 6.14. For part (1), we calculate as follows:

$$\mathcal{O}^*\llbracket\{\langle\alpha,e{:}g\rangle\}\rrbracket=\dot{\mathcal{O}}\llbracket\{\langle\alpha,e{:}\surd\rangle\}\rrbracket{:}(\lambda\beta.\mathcal{O}^*\llbracket\{\langle\alpha,\beta{:}g\rangle\}\rrbracket)\quad\text{(Lemma 6.21)}$$

$$= \mathscr{E}[\![e]\!]\gamma_t\alpha(\lambda\beta.\mathcal{O}^*[\![\{\langle\alpha, \beta:g\rangle\}]\!]) \qquad \text{(Lemma 6.22)}$$

where the application of Lemma 6.22 is allowed by Lemma 6.23.

Now for part (2), we have

$$\mathcal{O}^*[\![\{\langle\alpha, s;r\rangle\}]\!] = \dot{\mathcal{O}}[\![\{\langle\alpha, s;\surd\rangle\}]\!] : \mathcal{O}^*[\![\{\langle\alpha, r\rangle\}]\!] \quad \text{(Lemma 6.21)}$$
$$= \mathscr{D}[\![s]\!]\gamma_t\alpha(\mathcal{O}^*[\![\{\langle\alpha, r\rangle\}]\!]) \qquad \text{(Lemma 6.22)}$$

where $\tilde{s} = s$ and $\tilde{\gamma}_t = \gamma_t$ because s is closed. Here, again, Lemma 6.23 justifies the application of Lemma 6.22. □

6.24. Corollary. *For any* $t \in \mathscr{L}_{\text{nud}}$, $\mathcal{O}^*[\![t]\!] = \mathscr{D}[\![t]\!]$.

Proof. Let $t = \langle c_i \Leftarrow s_i\rangle_{i=1}^k$; then we have

$$\begin{aligned}
\mathcal{O}^*[\![t]\!] &= \mathcal{O}^*[\![\{\langle\langle c_1, 1\rangle, s_1; E\rangle\}]\!] && \text{(Definition 6.10(4))}\\
&= \mathscr{D}[\![s_1]\!](\gamma_t)(\langle c_1, 1\rangle)(\mathcal{O}^*[\![\{\langle\langle c_1, 1\rangle, E\rangle\}]\!]) && \text{(Theorem 6.14(2))}\\
&= \mathscr{D}[\![s_1]\!](\gamma_t)(\langle c_1, 1\rangle)(p_0) && \text{(Definition 6.10(3))}\\
&= \mathscr{D}[\![t]\!] && \text{(Definition 6.9(4)).} \quad \square
\end{aligned}$$

With Corollary 6.24, we have obtained the ultimate goal of our paper: to establish the equivalence of an operational and a denotational semantics for a nonuniform language with process creation.

Acknowledgment

Discusssions with Jeffery Zucker led to a considerably improved way of incorporating syntactic continuations for the uniform case. Moreover, Definition 4.4 is due to him. We are also indebted to Dr. Zucker for pointing out several minor and some major flaws in a draft of this paper which we (hope to) have corrected in the present version.

The contributions of Joost Kok and Jan Rutten to the design of the POOL semantics as reported in [6, 7] were absolutely essential for the present investigation. The idea of assembling transition sequence information into a process is due to Joost Kok. We acknowledge fruitful discussions on our work in the Amsterdam concurrency group, including Frank de Boer, Joost Kok, John-Jules Meyer, Jan Rutten and Erik de Vink.

Finally, we express our thanks to Marisa Venturini Zilli for the opportunity extended to the second author to lecture on the material of this paper in the Advanced

School on Mathematical Models for the Semantics of Parallelism, Rome, September 1986.

References

[1] *The Programming Language Ada Reference Manual*, American National Standards Institute, ANSI/MIL-STD-1815A-1983 (also published as: Lecture Notes in Computer Science **155** (Springer, Berlin, 1983)).

[2] G. Agha, Semantic considerations in the Actor paradigm of concurrent computations, in: S.D. Brookes, A.W. Roscoe and G. Winskel, eds., *Proc. Seminar on Concurrency*, Carnegie-Mellon University, Pittsburgh, PA, July 9-11, 1984, Lecture Notes in Computer Science **197** (Springer, Berlin, 1984) 151-179.

[3] P. America, Definition of the programming language POOL-T, ESPRIT Project 415, Doc. No. 91, Philips Research Laboratories, Eindhoven, The Netherlands, September 1985.

[4] P. America, Rationale for the design of POOL, ESPRIT Project 415, Doc. No. 53, Philips Research Laboratories, Eindhoven, The Netherlands, January 1986.

[5] P. America, Objected-oriented programming: a theoretician's introduction, *EATCS Bull. 29* (June 1986) 69-84.

[6] P. America, J.W. De Bakker, J.N. Kok and J.J.M.M. Rutten, Operational semantics of a parallel object-oriented language, in: *Conf. Rec. 13th Symp. on Principles of Programming Languages*, St. Petersburg, FL (January 13-15, 1986) 194-208.

[7] P. America, J.W. De Bakker, J.N. Kok and J.J.M.M. Rutten, A denotational semantics of a parallel object-oriented language, Report CS-R8626, Centre for Mathematics and Computer Science, Amsterdam, The Netherlands, August 1986.

[8] P. America and J.J.M.M. Rutten, Solving reflexive domain equations in a category of complete metric spaces, in: *Proc. Third Workshop on Mathematical Foundations of Programming Language Semantics*, New Orleans, LA, April 8-10, 1987, Lecture Notes in Computer Science **298** (Springer, Berlin, 1988) 254-288.

[9] K.R. Apt, Recursive assertions and parallel programs, *Acta Inform.* **15** (1981) 219-232.

[10] K.R. Apt, Formal justification of a proof system for communicating sequential processes, *J. ACM* **30**(1) (1983) 197-216.

[11] J.W. De Bakker, J.A. Bergstra, J.W. Klop and J.-J.Ch. Meyer, Linear time and branching time semantics for recursion with merge, *Theoret. Comput. Sci.* **34** (1984) 135-156.

[12] J.W. De Bakker, J.N. Kok, J.-J.Ch. Meyer, E.-R. Olderog and J.I. Zucker, Contrasting themes in the semantics of imperative concurrency, in: J.W. De Bakker, W.-P. De Roever and G. Rozenberg, eds., *Current Trends in Concurrency: Overviews and Tutorials*, Lecture Notes in Computer Science **224** (Springer, Berlin, 1986) 51-121.

[13] J.W. De Bakker and J.-J.Ch. Meyer, Order and metric in the stream semantics of elemental concurrency, *Acta Inform.* **24** (1987) 491-511.

[14] J.W. De Bakker, J.-J.Ch. Meyer and E.-R. Olderog, Infinite streams and finite observations in the semantics of uniform concurrency, *Theoret. Comput. Sci.* **49**(2, 3) (1987) 87-112.

[15] J.W. De Bakker, J.-J.Ch. Meyer, E.-R. Olderog and J.I. Zucker, Transition systems, infinitary languages and the semantics of uniform concurrency, in: *Proc. 17th ACM Symp. on the Theory of Computing*, Providence, RI (1985) 252-262.

[16] J.W. De Bakker, J.-J.Ch. Meyer, E.-R. Olderog and J.I. Zucker, Transition systems, metric spaces and ready sets in the semantics of uniform concurrency (full version of [15]), *J. Comput. System Sci.* **36** (1988) 158-224..

[17] J.W. De Bakker and J.I. Zucker, Processes and the denotational semantics of concurrency, *Inform. and Control* **54** (1982) 70-120.

[18] J.A. Bergstra and J.W. Klop, Process algebra for synchronous communication, *Inform. and Control* **60** (1984) 109-137.

[19] F.S. De Boer, A proof rule for process creation, in: M. Wirsing, ed., *Formal Description of Programming Concepts III, Proc. Third IFIP WG 2.2 Working Conf.*, Gl. Avernæs, Ebberup, Denmark, August 25-28, 1986 (North-Holland, Amsterdam, 1987) 23-50.

[20] M. Broy, Fixed point theory for communication and concurrency, in: D. Bjørner, ed., *Formal Description of Programming Concepts II* (North-Holland, Amsterdam, 1983) 125-146.

[21] M. Broy, Applicative real-time programming, in: R.E.A. Mason, ed., *Information Processing '83: Proc. IFIP Conference* (North-Holland, Amsterdam, 1983) 259-264.

[22] A. De Bruin and A.P.W. Böhm, The denotational semantics of dynamic networks of processes, *ACM Trans. Programming Languages and Systems* 7(4) (1985) 656-679.

[23] W.D. Clinger, Foundations of actor semantics, Technical Report No. 633, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, May 1981.

[24] O.-J. Dahl, B. Myhrhaug and K. Nygaard, SIMULA 67, Common Base Language, Norwegian Computing Center, Forskningsvn. lb., Oslo, Norway, 1967.

[25] J. Dugundji, *Topology* (Allyn & Bacon, Newton, MA, 1966).

[26] R. Engelking, *General Topology* (Polish Scientific Publishers, Warsaw, 1977).

[27] G. Gierz, K.H. Hofmann, K. Keimel, J.D. Lawson, M. Mislove and D.S. Scott, *A Compendium of Continuous Lattices* (Springer, Berlin, 1980).

[28] H. Hahn, *Reelle Funktionen* (Chelsea, New York, 1948).

[29] M. Hennessy and G.D. Plotkin, Full abstraction for a simple parallel programming language, in: J. Bečvář, ed., *Proc. 8th Symp. on Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science **74** (Springer, Berlin, 1979) 108-120.

[30] C. Hewitt, Viewing control structures as patterns of passing messages, *Artificial Intelligence* **8** (1977) 323-364.

[31] C.A.R. Hoare, Communicating sequential processes, *Comm. ACM* **21**(8) (1978) 666-677.

[32] C.A.R. Hoare, *Communicating Sequential Processes* (Prentice-Hall, Englewood Cliffs, NJ, 1985).

[33] J.-J.Ch. Meyer, Merging regular processes by means of fixed point theory, *Theoret. Comput. Sci.* **45** (1986) 193-260.

[34] J.-J.Ch. Meyer and E.P. de Vink, Applications of compactness in the Smyth powerdomain of streams, in: *Proc. TAPSOFT '87, Vol. 1*, Pisa, Italy, March 23-27, 1987, Lecture Notes in Computer Science **249** (Springer, Berlin, 1987) 241-255.

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

[36] D. Niwinski, Fixed point semantics for algebraic (tree) grammars, in: M. Nielsen and E.M. Schmidt, eds., *Proc. 9th Internat. Coll. on Automata, Languages and Programming*, Lecture Notes in Computer Science **140** (Springer, Berlin, 1982) 384-396.

[37] G.D. Plotkin, A powerdomain construction, *SIAM J. Comput.* **5**(3) (1976) 452-487.

[38] G.D. Plotkin, A structural approach to operational semantics, Report DAIMI FN-19, Computer Science Department, Aarhus University, September 1981.

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

[40] A. Pnueli, Linear and branching structures in the semantics and logics of reactive systems, in: W. Brauer, ed., *Proc. 12th Internat. Coll. on Automata, Languages and Programming*, Nafplion, Greece, July 15-19, 1985, Lecture Notes in Computer Science **194** (Springer, Berlin, 1985) 15-32.

[41] W.C. Rounds, On the relationship between Scott domains, synchronization trees and metric spaces, Report CRL-TR-25-83, University of Michigan, 1983.

[42] V.A. Saraswat, The concurrent logic programming language CP: definition and operational semantics, in: *Conf. Rec. 14th Symp. on Principles of Programming Languages*, München, Fed. Rep. Germany (January 21-23, 1987) 49-62.

[43] S.A. Smolka and R.E. Strom, A CCS semantics for NIL, in: M. Wirsing, ed., *Formal Description of Programming Concepts III, Proc. Third IFIP WG 2.2 Working Conf.*, Gl. Avernæs, Ebberup, Denmark, August 25-28, 1986 (North-Holland, Amsterdam, 1987) 347-373.