A Truly Concurrent Process Calculus over Recyclable Resources

Dan TEODOSIU
ADESCO Services, Strada Polona 43
010493 BUCHAREST, Romania, Europe
dan@teodosiu.net

Abstract. In this paper we present a truly concurrent semantics for deterministic concurrent recursive processes accessing quantified recyclable resources. The process semantics is built upon the new coherently complete and prime algebraic domain of continued multi-pomsets, which is a quantitative version of the domain of complex resource pomsets. The CSP-like process language that we study contains several deterministic quantitative process operators, namely hiding, restriction, sequential, serial and parallel, as well as an observable recursion. A deterministic structural operational machine is displayed that allows extracting a linear and a continued operational semantics. The denotational semantics is naturally defined on continued multi-pomsets for all finitary operators and lifted to a functional domain over environments modelling recursion. The robustness of the present semantical work is demonstrated by proving that the denotational semantics is fully abstract with respect to the linear and the continued operational semantics and by relating it via observation to bisimilarity.

1 Introduction

Hennessy & Plotkin [6] have shown in their seminal work how power domains can be employed in order to build semantic models of parallelism, by reducing it to interleaving and choice. In this paper we present a genuine approach to parallelism, that avoids using choice, thereby relying on true concurrency. The latter has motivated numerous domain-theoretic approaches based on variants of the event structures introduced by Winskel [14], of which the pomsets advocated by Pratt [9] are a particular but versatile case.

Along this line of research, an advanced truly concurrent approach, which has developed an operational and a matching denotational programming semantics modelling the customary process combinators excepting choice, was elaborated in Diekert & Gastin [2], Gastin & Teodosiu [5] and Gastin & Mislove [4]. This work grounds upon the appealing interaction between processes and resources in any environment, a paradigm also driving the work in applied computer science and management. One should also note the related work of Pym & Tofts [10] presenting a truly concurrent algebra and logic of processes and resources for a number of process combinators including choice.
A new stream in automata theory, as emphasized by the recent monograph on weighted automata [3], consists in attaching weights to transitions that measure their cost in terms of some available resources and extending these notions to the recognized words and languages. Classical automata theory can in particular be recovered by considering unit weights, which is why the weighted view is more flexible, opening the way to new applications in engineering and economy.

A first attempt to develop a process language that combines the above truly concurrent approach on the denotational side with a weighted automata view on the operational side was undertaken in Teodosiu [11,12] that operated an enrichment of the underlying domains and operators aimed at modelling processes accessing consumable resources. In that setting, resources reserved by a process are never returned to the environment upon termination, hence consumed. Therefore, the central idea there was to base the denotation on a resource semantics that adds the resources of sequentially composed processes. Most notably, hiding is a plus-action, hence non-idempotent, in this non-classical approach.

In this paper we deal with processes accessing recyclable resources. In this view, resources are allocated (reserved) by a process during execution and automatically freed (returned) to the environment upon termination. Therefore, the essential idea here is to base the denotation on a resource semantics that joins the resources of sequentially composed processes. We note that hiding is a join-action, hence idempotent, as in the classical approaches to process algebra.

We display a deterministic quantitative structural operational machine which engenders a linear operational semantics, that records only the strings of executed actions, and a continued operational semantics, that records the pomset of executed actions as well as the subsequent allocation of a process.

The denotational semantics is based on the domain of continued multi-pomsets. These are composed of an observation, which is a multi-pomset of presently executed actions, and of a continuation, which is a multi-set of subsequently allocated resources. The labeling of the denotational models and of the operational transition rules is quantified with the amount of resources being allocated. Quantification requires to replace the previously employed boolean algebra of sets of resources, with a real algebra of multi-sets of resources, while taking care that relevant algebraic properties remain valid in this new setting. Furthermore, it leads to defining operators whose quantitative semantics enriches that of previously considered operators.

The paper is organized as follows. Section 2 recalls basic facts about partial orders. Section 3 introduces the domain of continued multi-pomsets. Section 4 presents the process language and defines the allocation semantics. Section 5 displays the deterministic structural operational machine, which engenders the linear and the continued operational semantics. Section 6 is devoted to the ample compositional denotational semantics. Section 7 presents the main results of full abstraction of the denotational semantics with respect to the linear and the continued operational semantics and relates it via observation to bisimilarity.

A complete version of our work containing all results with full proofs may be consulted online at Teodosiu [13].
2 Partial Orders

For a detailed exposition we refer to the presentation of Abramsky & Jung [1].

A partial order (PO) is a pair \((X, \leq)\) where \(\leq\) is a reflexive, antisymmetric and transitive binary relation on \(X\). A subset \(Y \subseteq X\) is directed (resp. coherent) iff for all \(x, y \in Y\) there exists \(z \in Y\) (resp. \(z \in X\)) such that \(x \leq z\) and \(y \leq z\). \((X, \leq)\) is a directed complete PO (DCPO) (resp. coherently complete PO (CCPO)) iff every directed (resp. coherent) subset has a least upper bound. It follows directly that any CCPO is a DCPO.

An element \(x \in X\) is prime (resp. compact) iff for all (resp. all directed) subsets \(Y \subseteq X\) having a least upper bound, \(x \leq \bigvee Y\) implies \(x \leq y\) for some \(y \in Y\). The set of all prime (resp. compact) elements of \(X\) below \(x \in X\) is denoted by \(\text{Prm}(x)\) (resp. \(\text{Kmp}(x)\)). A partial order \((X, \leq)\) is \(p\)-algebraic iff \(x = \bigvee \text{Prm}(x)\) for all \(x \in X\). It is \(k\)-algebraic iff \(\text{Kmp}(x)\) is directed and \(x = \bigvee \text{Kmp}(x)\) for all \(x \in X\). It is easy to show that any \(p\)-algebraic CCPO is \(k\)-algebraic.

A \(k\)-algebraic DCPO is called a \((Scott-)domain\). A mapping \(F : (X, \leq) \to (X', \leq')\) is \((Scott-)continuous\) iff for all directed sets \(Y \subseteq X\), such that \(\bigvee Y\) exists, \(\bigvee F(Y)\) exists, and \(F(\bigvee Y) = \bigvee F(Y)\).

The set of finite ordinals (i.e. the least infinite ordinal) is denoted by \(\omega\).

3 Multi-Pomsets and Continued Multi-Pomsets

We start with some algebraic preliminaries about vectors of finite and infinite positive reals, also called multi-sets.

We denote by \(\Re_+ = [0, \infty]\) the set of positive extended reals. We fix in this section a countable set of resources \(\mathcal{R}\). The set of amounts is the set of multi-sets of resources \(\mathbf{A} = \mathcal{R} \to \Re_+\). We define \(0, \infty \in \mathbf{A}\) by \(0(\alpha) = 0\) and \(\infty(\alpha) = \infty\) for all \(\alpha \in \mathcal{R}\). The support of \(a \in \mathbf{A}\) is \(\text{supp}(a) = \{\alpha \in \mathcal{R} \mid a(\alpha) \neq 0\}\). We also fix in this section a set of actions \(\mathbf{A}_\mathcal{R} \subseteq \{a \in \mathbf{A} \mid a \neq 0\}\).

On \(\mathbf{A}\) we define componentwise the order \(\leq\) \(\mathbf{A} \times \mathbf{A} \to \mathbf{A}\), the complement \(^- : \mathbf{A} \to \mathbf{A}\), the sum \(+ : \mathbf{A} \times \mathbf{A} \to \mathbf{A}\), the infimum \(\wedge : \mathbf{A} \times \mathbf{A} \to \mathbf{A}\), the supremum \(\vee : \mathbf{A} \times \mathbf{A} \to \mathbf{A}\) and the skew difference \(\setminus : \mathbf{A} \times \mathbf{A} \to \mathbf{A}\), whereby for all \(n, m \in \Re_+\) we set \(\pi = \infty\) if \(n = 0\), \(\pi = 0\) if \(n \neq 0\), \(n \setminus m = \infty\) if \(n = \infty\), \(n \setminus m = 0\) if \(n \neq \infty\) and \(m = \infty\), \(n \setminus m = (n - m) \vee 0\) if \(n, m \neq \infty\). For \(a, b \in \mathbf{A}\) such that \(b \leq a\) let \(a - b = a \setminus b\). For \(a, b \in \mathbf{A}\) we define the independence \(a \perp b\) iff \(a \wedge b = 0\).

We canonically identify each set \(a \subseteq \mathcal{R}\) with its characteristic multi-set \(a \in \mathbf{A} \to \Re_+\) defined by \(a(\alpha) = 1\) for \(\alpha \in a\) and \(a(\alpha) = 0\) for \(\alpha \notin a\). This identification allows us to set \(\mathcal{P}(\mathcal{R}) \subseteq \mathbf{A} \to \Re_+ = \mathbf{A}\).

The multiplicity attached by an action to a resource measures the reserved amount or allocation. If, for example, \(\mathcal{R} = \{\alpha, \beta, \gamma\}\) then an action allocated respectively 3, 5 and 7 units of \(\alpha, \beta\) and \(\gamma\) is denoted by the multi-set \(3\alpha + 5\beta + 7\gamma \in \mathbf{A}\). Immediate examples come from computer science (an action allocated 5 processors, 10 channels and 2 memories) or workflow management (an action allocated 100 men, 5 tools and 10 objects). Engineering and economy, where the notion of resource is ubiquitous, should offer further examples of applications that employ fractional or even continuous amounts of resources.
3.1 The Domain of Multi-Pomsets $\mathcal{P}$

The following sketch of multi-pomsets, which appeared in slightly modified form in [12], is included for the sake of completion. Further definitions and results with full proofs pertaining to multi-pomsets have been worked out in [11].

A multi-labelled partial order is a triple $(E, \preceq, \rho)$, where

1. the synchronization relation $\preceq \subseteq E \times E$ is a partial order on the set of events $E$ satisfying the past finiteness condition that $\{ f \in E \mid f \preceq e \}$ is finite for all $e \in E$.
2. the event-labelling $\rho : E \to A_R$ satisfies the over-synchronization condition that $\rho(e) \land \rho(f) \neq 0 \implies e \preceq f$ or $f \preceq e$ for all $e, f \in E$.

A multi-pomset is the isomorphism class $[E, \preceq, \rho]$ of a real multi-labelled partial order $(E, \preceq, \rho)$. The set of multi-pomsets is denoted by $\mathcal{P}$. A finite multi-pomset is a multi-pomset whose event set is finite. The set of finite multi-pomsets is denoted by $\mathcal{F}$. The empty multi-pomset is $0 = [\emptyset, \emptyset, \emptyset] \in \mathcal{P}$. For all $a \in A_R$ we define the action multi-pomset $a = [(\{\emptyset\}, \{(\emptyset, \emptyset)\}, \{(\emptyset, a)\})] \in \mathcal{P}$.

The synchronization relation reflects the temporal (or causal) order between the events of the multi-pomset. The past finiteness condition is a technical assumption that formalizes the computational intuition that all events have finite causes but can be relaxed if one wishes to deal with transfinite multi-pomsets.

The over-synchronization condition is equivalent to the fact that for each resource the events allocating it are sequentialized (totally ordered). In particular it implies that the multi-pomset has no auto-concurrency, that is, for all $a \in A_R$ the set $\rho^{-1}(a) = \{ e \in E \mid \rho(e) = a \}$ is totally ordered by $\preceq$. The number of occurrences $|_a : \mathcal{P} \to \omega + 1$ of a $a \in A_R$ is defined for all $x \in \mathcal{P}$ by $|x|_a = \text{ord}(\rho^{-1}(a), \preceq \cap \rho^{-1}(a) \times \rho^{-1}(a)) \leq \omega$, that is the ordinal associated with the well-order induced by $\preceq$ on $\rho^{-1}(a)$.

In order to avoid cumbersome isomorphism proofs we define the standard representative of $x = (E, \preceq, \rho)$ as the unique isomorphic multi-labelled partial order $\hat{x} = (E_x, \preceq_x, \rho_x)$, such that

1. $E_x = \phi_x(E) = \{ (a, n) \mid a \in A_R \text{ and } n < |x|_a \} \subseteq A_R \times \omega = \mathbb{E}$,
2. $(a, n) \preceq_x (a, m)$ for all $a \in A_R$ and $n \leq m < |x|_a$,
3. $\rho_x(a, n) = a$ for all $a \in A_R$ and $n < |x|_a$.

The past in $x \in \mathcal{P}$ of $F \subseteq \mathbb{E}$ is $\downarrow_x F = \{ e \in E_x \mid \exists f : e \preceq_x f \in F \}$. The restriction of $x \in \mathcal{P}$ to $F \subseteq \mathbb{E}$ is $x/F = [E_x \cap F, \preceq_x \cap F \times F, \rho_x \cap F \times A_R]$. The prefix is defined for all $x, y \in \mathcal{P}$ by $x \preceq y$ if $E_x = \downarrow_y E_x$ and $x = y/E_x$. In this case we define the residue $x^{-1}y = y/(E_y \setminus E_x)$.

The next theorem shows that $(\mathcal{P}, \preceq)$ is an interesting semantic domain.

**Theorem 1.** $(\mathcal{P}, \preceq)$ is a p-algebraic CCPO, hence, it is a (Scott-)domain.

The alphabet $\text{alph} : \mathcal{P} \to \mathcal{P}(A_R)$ is defined for all $x \in \mathcal{P}$ by $\text{alph}(x) = \rho_x(E_x)$. The allocation $\text{alloc} : \mathcal{P} \to A$ is defined for all $x \in \mathcal{P}$ by $\text{alloc}(x) = \bigvee_{e \in E_x} \rho_x(e)$. It can be shown that $\text{alloc} : (\mathcal{P}, \preceq) \to (A, \preceq)$ is continuous. For all $x, y \in \mathcal{P}$
we define the independence \( x \perp y \) iff \( \text{alloc}(x) \perp \text{alloc}(y) \). The continuation \( \text{cont} : \mathbb{P} \to \mathbb{A} \) is defined for all \( x \in \mathbb{P} \) by \( \text{cont}(x) = \bigwedge \{ \text{alloc}(k^{-1}x) \mid k \in \mathbb{F}, k \leq x \} \). It measures the eventually persistent allocation.

We need later on the following definitions. For \( a \in \mathbb{A}_R \) and \( x \in \mathbb{P} \) satisfying the action prefix \( a \leq x \) let the action residue be \( a^{-1}x = x/(E_x \setminus \{(a,0)\}) \). Let \( \emptyset \) denote the empty string in \( \mathbb{A}_R^* \). For \( u \in \mathbb{A}_R^* \) and \( x \in \mathbb{P} \) we define the linear prefix \( u \preceq x \) and the linear residue \( u^{-1}x \) inductively on \( u \) by:

- Let \( 0 \preceq x \) and \( 0^{-1}x = x \).
- For \( a \in \mathbb{A}_R \) let \( ua \preceq x \) iff \( u \preceq x \), \( a \leq u^{-1}x \), and let \( (ua)^{-1}x = a^{-1}(u^{-1}x) \).

The set of linearizations of \( r \) is \( \text{Lin}(x) = \{ u \in \mathbb{A}_R^* \mid u \preceq x, u^{-1}x = 0 \} \).

We note that for any \( a \in \mathbb{A}_R \subseteq \mathbb{P} \) and \( x \in \mathbb{P} \) we have \( a \preceq x \) iff \( a \leq x \) iff \( x \) contains a minimal event labelled with \( a \), so in this case the prefix and linear prefix relation coincide.

The main deficiency of \( (\mathbb{P}, \preceq) \) is the fact that it is unsuitable to define continuous denotations for the operators of our process language, since for example the sequential composition is not monotone (hence, not continuous). We surmount this obstacle by introducing next the domain of continued multi-pomsets.

### 3.2 The Domain of Continued Multi-Pomsets \( \mathbb{C} \)

A continued multi-pomset is a pair \( x = (r, R) \), where \( r \in \mathbb{P} \) is a multi-pomset, and \( R \in \mathbb{A} \) is a multi-set of resources, such that \( \text{cont}(r) \leq R \). The set of continued multi-pomsets is denoted by \( \mathbb{C} \). The multi-pomset \( r \) is denoted by \( \text{obs}(x) \) and called the observation of \( x \). The multi-set \( R \) is denoted by \( \text{cont}(x) \) and called the continuation of \( x \). If the continuation \( \text{cont}(x) \) is zero, \( x \) is called terminated. The allocation of \( x \in \mathbb{C} \) is \( \text{alloc}(x) = \text{alloc}(\text{obs}(x)) \cup \text{cont}(x) \).

The first component of a continued multi-pomset is a multi-pomset describing the already observed part of the process, while the second component is a multi-set of resources representing the quota allocated to the process for its continuation.

The approximation order on \( \mathbb{C} \) is defined for all \( (r, R), (s, S) \in \mathbb{C} \) by \( (r, R) \preceq (s, S) \iff r \leq s \) and \( \text{alloc}(r^{-1}s) \cup S \leq R \). The underlying idea here is that we increase the information about a process by letting grow its observation \( r \leq s \) according to the present continuation quota \( \text{alloc}(r^{-1}s) \leq R \), while reducing the subsequent continuation quota \( S \leq R \).

For \( x \in \mathbb{C} \) let \( K(x) = \{ (k, \text{alloc}(k^{-1}\text{obs}(x)) \cup \text{cont}(x)) \mid k \in \mathbb{F}, k \leq \text{obs}(x) \} \).

We use the fact that \( K(x) \) is directed, \( K(x) \subseteq \text{Kmp}(x) \) and \( \bigsqcup K(x) = x \).

Relying on Theorem 1, the next theorem shows that \( (\mathbb{C}, \preceq) \) is a suitable semantic domain.

**Theorem 2.** \( (\mathbb{C}, \preceq) \) is a \( p \)-algebraic CCPO, hence, it is a (Scott-)domain.

The main virtue of \( (\mathbb{C}, \preceq) \) is the fact that it allows defining internal and continuous denotations for all process operators of our language.

We need later on the following definitions. For \( u \in \mathbb{A}_R^* \) and \( x \in \mathbb{C} \) we extend the linear prefix relation by \( u \preceq x \) iff \( u \preceq \text{obs}(x) \) and the linear residue by \( u^{-1}x = (u^{-1}\text{obs}(x), \text{cont}(x)) \).
4 The Process Language

We fix in the following disjoint countable sets of constants $\mathcal{C}$ and variables $\mathcal{V}$ and let $\mathcal{R} = \mathcal{C} \cup \mathcal{V}$. We fix $\mathcal{A}_C \subseteq \{a : \mathcal{C} \rightarrow \mathbb{R}_+ \mid \text{supp}(a) \text{ is finite and } a \neq 0\}$ satisfying the technical assumption $\bigwedge \{a(\alpha) \mid a \in \mathcal{A}_C \text{ and } a(\alpha) \neq 0\} > 0$ for all $\alpha \in \mathcal{R}$, we define $\mathcal{A}_V = \{x \mid x \in \mathcal{V}\}$, and let $\mathcal{A}_R = \mathcal{A}_C \cup \mathcal{A}_V$.

The transition rules of our deterministic structural operational machine, whereby the allocation of sequential compositions does not add but join those of the parts, thus generalizing [4].

Most crucially, contrary to the consumption of [11,12], the allocation of sequen-

The language of closed terms $\mathcal{L}_c$ is the set of terms without free variables. The allocation $\text{Alloc}(p) \in \mathcal{A}$ of a term $p \in \mathcal{L}$ is inductively defined by

\[
\begin{align*}
\text{Alloc}(\text{SKIP}) &= 0 & \text{Alloc}(p \cdot q) &= \text{Alloc}(p) \lor \text{Alloc}(q) \\
\text{Alloc}(c) &= c & \text{Alloc}(p \parallel q) &= \text{Alloc}(p) \lor \text{Alloc}(q) \\
\text{Alloc}(p \odot T) &= \text{Alloc}(p) \setminus T & \text{Alloc}(x) &= 0 \\
\text{Alloc}(p \odot T) &= \text{Alloc}(p) \land T & \text{Alloc}(\text{rec } x.p) &= \{x\} \lor \text{Alloc}(p) \\
\text{Alloc}(p ; q) &= \text{Alloc}(p) \lor \text{Alloc}(q)
\end{align*}
\]

The above definition determines a compositional semantics $\text{Alloc} : \mathcal{L} \rightarrow \mathcal{A}$. Most crucially, contrary to the consumption of [11,12], the allocation of sequential compositions does not add but join those of the parts, thus generalizing [4].

5 The Operational Semantics

The transition rules of our deterministic structural operational machine, whereby $c \in \mathcal{A}_C$, $T \in \mathcal{A}$, $C \subseteq \mathcal{A}_R$, $x \in \mathcal{V}$, $p, p', q, q' \in \mathcal{L}$, $a \in \mathcal{A}_R \cup \{0\}$, are

\[
\begin{align*}
\text{[ACT]} & \quad \frac{c \rightarrow \text{SKIP}}{c \in \mathcal{C}, \quad p \rightarrow p'} \\
\text{[HID]} & \quad \frac{p \odot T \rightarrow p' \odot T}{p \rightarrow p', \quad a \leq T} \\
\text{[RES]} & \quad \frac{p \rightarrow p'}{p ; q \rightarrow p' ; q} \\
\text{[SEQ1]} & \quad \frac{\text{Alloc}(p) = 0, \ q \rightarrow q'}{p ; q \rightarrow p ; q'} \\
\text{[SEQ2]} & \quad \frac{\text{Alloc}(p) = 0}{q \rightarrow p} \\
\text{[SER1]} & \quad \frac{p \rightarrow p'}{\text{Alloc}(p) \land a, \ q \rightarrow q'} \\
\text{[SER2]} & \quad \frac{p \rightarrow p'}{\text{Alloc}(p) \parallel a, \ q \rightarrow q'} \\
\text{[PAR0]} & \quad \frac{a \in C, \ p \rightarrow p', \ q \rightarrow q'}{p \parallel q \rightarrow p' \parallel q'} \\
\text{[PAR1]} & \quad \frac{C, \text{Alloc}(p) \land a, \ p \rightarrow p'}{p \parallel q \rightarrow p' \parallel q} \\
\text{[PAR2]} & \quad \frac{C, \text{Alloc}(p) \parallel a, \ q \rightarrow q'}{p \parallel q \rightarrow p \parallel q'} \\
\text{[REC]} & \quad \frac{\text{rec } x.p = q}{q \rightarrow p[q/x]}
\end{align*}
\]
As usual, \( p[q/x] \) denotes the term that is obtained from \( p \) after substituting all occurrences of the variable \( x \) by \( q \). Note that recursion is modeled in an observable way, each unwinding producing as observation the variable being recursed, which may subsequently be hidden.

For \( u \in \mathbb{A}^*_R \) and \( p, p' \in \mathcal{L}, r \in \mathbb{A}^V \), let the linear transition \( p \xrightarrow{u} p' \) be inductively defined on the length of \( u \) by

- Let \( p \xrightarrow{0} p' \) iff \( p = p' \).
- For \( a \in \mathbb{A}_R \) let \( p \xrightarrow{u,a} p' \) iff there exists \( q \in \mathcal{L} \) such that \( p \xrightarrow{u} q \) and \( q \xrightarrow{a} p' \).

Using the linear transition we next define a linear and a continued operational semantics of closed process terms as follows.

The linear behaviour of \( p \in \mathcal{L}_c \) is \( \mathbb{A}^*_R(p) = \{ u \in \mathbb{A}^*_R \mid p \xrightarrow{u} \} \).

For any fixed \( E \subseteq \mathbb{E} \) the intersection \( \bigcap P \) of a set \( P \subseteq \mathbb{P} \) such that for all \( r \in P \) we have \( E_r = E \) is defined by \( E \bigcap P = E \) and \( \leq \bigcap P = \bigcap \{ \leq_r \mid r \in P \} \).

The restriction of \( p \in \mathcal{L}_c \) to \( E \) is \( p/E = \bigcap \{ u \in \mathbb{A}^*_R \mid p \xrightarrow{u} \} \) and \( E_u = E \).

The continued behaviour of a process term, that observes continued multi-pomsets of actions as we shall later on see, is defined by extracting an observation which is a restriction of the linear behaviours of that process term and a continuation which records the future allocation of that process term.

The continued behaviour of \( p \in \mathcal{L}_c \) is \( \mathbb{C}(p) = \{ (p/E_u, \text{Alloc}(p')) \mid p \xrightarrow{u} p' \} \).

### 6 The Denotational Semantics

We next construct a denotational semantics for our process language using a functional domain over environments of continued multi-pomsets.

We endow the set of environments \( \mathbb{C}^V \) with the product order \( \sqsubseteq \), that is, we set \( (\mathbb{C}^V, \sqsubseteq) = (\mathbb{C}, \sqsubseteq)^V \). \( \text{alloc}^V : \mathbb{C}^V \rightarrow \mathbb{A}^V \) is defined for all environments \( \sigma \in \mathbb{C}^V \) and \( x \in \mathcal{V} \) by \( \text{alloc}^V(\sigma)(x) = \text{alloc}(\sigma(x)) \).

We define as usual the overridden environment \( \sigma[x \mapsto v] \in \mathbb{C}^V \) to be identical to the environment \( \sigma \in \mathbb{C}^V \) on all arguments except \( x \in \mathcal{V} \), which is assigned the value \( v \in \mathbb{C} \).

The functional domain \( \mathbb{D} \) is the set of continuous mappings \( f : (\mathbb{C}, \sqsubseteq)^V \rightarrow (\mathbb{C}, \sqsubseteq) \) satisfying for all \( \sigma \in \mathbb{C}^V \) and \( y \in \mathcal{V} \) the functional condition

\[
\text{alloc}(f(\sigma[y \mapsto (0,0)])) \leq \text{alloc}(f(\sigma)) \leq \text{alloc}(f(\sigma[y \mapsto (0,0)])) \vee \text{alloc}(\sigma(y)) \quad(*)
\]

We pointwise lift the ordering \( \sqsubseteq \) from \( \mathbb{C} \) to \( \mathbb{D} \), that is, for \( f, g \in \mathbb{D} \) we define \( f \sqsubseteq g \) iff \( f(\sigma) \sqsubseteq g(\sigma) \) for all \( \sigma \in \mathbb{C}^V \).

Using Theorem 2, we can show that \( (\mathbb{D}, \sqsubseteq) \) is a DCPO.

**Theorem 3.** \( (\mathbb{D}, \sqsubseteq) \) is a DCPO.

We now proceed with the denotation of the operators of our language. We mention in advance that the semantics of the restriction operator extends that of [5] otherwise than [11,12], the semantics of the hiding and recursion operators enriches that of [4] differently than [11,12], while the semantics of the sequential, serial and parallel compositions is adapted from [5,4,11,12].
6.1 The Empty Process SKIP and the Constant Action \( c \)

The empty process \( \text{SKIP} : \mathbb{C}^0 \rightarrow \mathbb{C} \) is defined by \( \text{SKIP}() = (0, 0) \). For all \( c \in \mathbb{A}_C \), the constant action \( c : \mathbb{C}^0 \rightarrow \mathbb{C} \) is defined by \( c() = (c, 0) \).

6.2 The Hiding Operator \( x \otimes T \)

The following hiding operator is an enrichment of the one treated in [4], which acts differently from the one defined in [11,12]. The hiding operator allows to persistently internalize some given quota of resources and prevents other processes from synchronizing on events that employ them. As usually, this allows to model local, as opposed to global, computation and communication.

For every \( T \in \mathbb{A} \), the hiding operator \( \otimes T : \mathbb{P} \rightarrow \mathbb{P} \) is defined for all \( [E, \preceq, \rho] \in \mathbb{P} \) by \( [E, \preceq, \rho] \otimes T = [E', \preceq', \rho'] \) where \( \rho(e) = \rho(e) \setminus T \), \( E' = \{ e \in E \mid \rho'(e) \neq 0 \} \), \( \preceq' = \preceq \cap E' \times E' \).

The hiding \( x \otimes T \) erases a given allocation \( T \) out of each event of \( x \), thereby rendering it unobservable.

For every \( T \in \mathbb{A} \), the hiding operator \( \otimes T : \mathbb{C} \rightarrow \mathbb{C} \) is defined for all \( x \in \mathbb{C} \) by \( x \otimes T = (\text{obs}(x) \otimes T, \text{cont}(x) \setminus T) \).

As can be expected, \( \otimes T \) is a \( \lor \)-action on \( \mathbb{C} \).

6.3 The Restriction Operator \( x / T \)

The following restriction operator is an extension of the one treated in [5], which is different from the one in [11,12]. The restriction operator persistently blocks a process on all but some given quota of resources. This may be employed to assure confinement of that process to a given safe environment, rendering this construct interesting for security protocols.

For every \( T \in \mathbb{A} \), the prefix restriction operator \( / T : \mathbb{P} \rightarrow \mathbb{P} \) and the corresponding suffix restriction operator \( T^{-1} : \mathbb{P} \rightarrow \mathbb{A} \) are defined by \( x / T = \bigvee \{ y \in \mathbb{P} \mid y \leq x, \text{alloc}(y) \leq T \} \) and \( T^{-1} x = \text{alloc}(x / T^{-1} x) \).

One can easily see that \( x / T \) is the restriction of \( x \) to the set of events having a past of events that allocate resources below \( T \), that is \( x / T = x / E'_x \), where \( E'_x = \{ e \in E_x \mid \rho_x(f) \leq T \text{ for all } f \leq e \} \), while \( T^{-1} x \) are the resources used by the events of \( x \) belonging to the remaining part, that is and \( T^{-1} x = \text{alloc}(x / E''_x) \) where \( E''_x = E_x \setminus E'_x \).

For every \( T \in \mathbb{A} \), the restriction operator \( \otimes T : \mathbb{C} \rightarrow \mathbb{C} \) is defined for all \( (r, R) \in \mathbb{C} \) by \( (r, R) \otimes T = (r / T, (T^{-1} r \lor R) \land T) \).

As can be expected, \( \otimes T \) is an \( \land \)-action on \( \mathbb{C} \).

6.4 The Sequential Composition \( x ; y \)

The following sequential composition is similar to the one treated in [4,11,12]. The sequential composition enforces a complete synchronization between the first and the second concurrent process. The compound processes are scheduled such that the entire first process occurs before the entire second process, hence
they are temporally ordered, even if independent. We shall later on essentially use the sequential composition in order to define the denotational semantics of recursion.

The sequential composition \( \cdot \) : \( \mathbb{P}^2 \rightarrow \mathbb{P} \) is defined for all \( x_1 = [E_1, \preceq_1, \rho_1] \in \mathbb{P} \) and \( x_2 = [E_2, \preceq_2, \rho_2] \in \mathbb{P} \) by \( x_1 \cdot x_2 = [E_1 \cup E_2, (\preceq_1 \cup E_1 \times E_2 \cup \preceq_2)^*, \rho_1 \cup \rho_2] \).

The sequential composition \( ; \) : \( \mathbb{C}^2 \rightarrow \mathbb{C} \) is defined for all \( x, y \in \mathbb{C} \) by
\[
x ; y = \begin{cases} 
(\text{obs}(x), \text{cont}(x) \lor \text{alloc}(y)) & \text{if cont}(x) \neq 0, \\
(\text{obs}(x) ; \text{obs}(y), \text{cont}(y)) & \text{if cont}(x) = 0.
\end{cases}
\]

One can easily verify that \( \langle \mathbb{C}, ;, (0,0) \rangle \) is a monoid.

### 6.5 The Serial Composition \( x \cdot y \)

The following serial composition is a generalization of the one treated in [5], which is similar to the one in [11,12]. The serial composition enforces synchronizations between the first and the second concurrent process in order to prevent races for resources. Events at the end of the first and events at the beginning of the second process are not temporally ordered if they are independent and can thus occur concurrently. Automatic parallelizers and transactional systems make extensive use of this kind of construct.

The serial composition \( \cdot \) : \( \mathbb{P}^2 \rightarrow \mathbb{P} \) is defined for all \( x_1 = [E_1, \preceq_1, \rho_1] \in \mathbb{P} \) and \( x_2 = [E_2, \preceq_2, \rho_2] \in \mathbb{P} \) if \( \text{cont}(x_1) \land \text{alloc}(x_2) = 0 \) by \( x_1 \cdot x_2 = [E, \preceq, \rho] \), where \( E = E_1 \cup E_2, \preceq = (\preceq_1 \cup \{ (e_1, e_2) \in E_1 \times E_2 \mid \rho_1(e_1) \land \rho_2(e_2) \neq 0 \} \cup \preceq_2)^*, \rho = \rho_1 \cup \rho_2. \)

The serial composition \( \cdot \) : \( \mathbb{C} \times \mathbb{C} \rightarrow \mathbb{C} \) is defined for all \( (r, R), (s, S) \in \mathbb{C} \) by
\[
(r, R) \cdot (s, S) = (r \cdot (s / \tilde{R}), R \lor \tilde{R}^{-1} s \lor S).
\]

One can easily verify that \( \langle \mathbb{C}, \cdot, (0,0) \rangle \) is a monoid.

It can be shown that using the serial composition together with hidings enables us to denote all compact continued multi-pomsets by closed terms of the process language, which means that the denotational semantics is optimal. It also implies that our language is able to specify all (multi-)pomsets not merely series-parallel pomsets as do simpler languages.

### 6.6 The Parallel Composition \( x \parallel y \)

The following parallel composition is an enrichment of the one treated in [4], which is analogous to the one in [11,12]. The parallel composition is indexed by a set of channels that two concurrent processes may employ in order to synchronize. Events labeled by channels are commonly executed by both processes, while each process separately executes events which are independent of the channels and of the other process. This construct may be used for example to model data-parallel programming, where processes perform a task in common by working independently in parallel.

Let \( x_1 = [E_1, \preceq_1, \rho_1], x_2 = [E_2, \preceq_2, \rho_2] \in \mathbb{P} \) be in standard representation. We define their parallel composition by \( x_1 \parallel x_2 = [E_1 \cup E_2, (\preceq_1 \cup \preceq_2)^*, \rho_1 \cup \rho_2]. \)
Note that \( x_1 \parallel x_2 \) may fail to be a multi-pomset for two different reasons. First, \( \preceq \) may fail to be antisymmetric. This is the case for instance if \( x_1 = a \cdot b \) and \( x_2 = b \cdot a \). Second, \( \preceq \) may fail to be over-synchronized. This is the case for instance if \( x_1 = a \) and \( x_2 = b \) with \( \neg (a \parallel b) \).

Let \( x_1, x_2 \in C \) and \( C \subseteq A_R \). We define \( (r_1, r_2) \in P_C(x_1, x_2) \subseteq P^2 \) iff

1. for all \( i \in \{1, 2\} \) we have \( r_i \leq \text{obs}(x_i) \),
2. \( |r_1|_a = |r_2|_a \) for all \( a \in C \),
3. for all \( \{i, j\} = \{1, 2\} \) and \( a \in \text{alph}(r_i) \) we have \( a \in C \) or \((a \parallel C\) and \( a \parallel x_j \) ),
4. \( r_1 \| r_2 \in P \).

For all \( C \subseteq A_R \) the parallel composition \( \parallel_C : C \times C \rightarrow C \) is defined for all \( (s_1, S_1), (s_2, S_2) \in C \) by \( (s_1, S_1) \parallel_C (s_2, S_2) = (r_1 \| r_2, \text{alloc}(r_1^{-1}s_1) \lor S_1 \lor \text{alloc}(r_2^{-1}s_2) \lor S_2) \), where \( (r_1, r_2) = \bigvee P_C(x_1, x_2) \).

### 6.7 The Recursion Operator \( \text{rec} \cdot f \)

The following recursion operator is an enrichment of the one treated in [4], which is computationally simpler than the one defined in [11,12], due to the functional condition (*) imposed on the functional domain \( D \). The recursion operator renders the process language substantially more expressive, ensuring that recursive processes unfold to complex infinite multi-pomsets thereby accounting for the subtle semantical problems that arise.

The denotational semantics of the recursion operator essentially differs from the least fixed point semantics. Indeed, for any fixed \( x \in V \), we define the recursion operator \( \text{rec} \cdot x : D \rightarrow D, f \mapsto \text{rec} \cdot x \cdot f \) by computing \( (\text{rec} \cdot x \cdot f)(\sigma) \) for all \( f \in D \) and \( \sigma \in C^V \) as the least fixed point of the recursion \( v = \{x \} : f(\sigma[x \mapsto v]) \) above the minimal initial allocation that corresponds to \( x = (0, 0) \).

Therefore, we define the recursion mapping \( C_x : D \times C^V \times C \rightarrow C \) by \( C_x(f, \sigma, v) = \{x\} : f(\sigma[x \mapsto v]) \), and the initial mapping \( A_x : D \times C^V \rightarrow A \) by \( A_x(f, \sigma) = \{x\} \lor f(\text{alloc}(\sigma)[x \mapsto (0, 0)]) = \text{alloc}(C_x(f, \sigma, (0, 0))) \).

In order to compute \((\text{rec} \cdot x \cdot f)(\sigma)\), we start with \( x_0(f, \sigma) = (0, A_x(f, \sigma)) \in C \) and iterate the mapping \( C_x(f, \sigma) : (C, \sqsubseteq) \rightarrow (C, \sqsubseteq) \), by defining \( x_{n+1}(f, \sigma) = C_x(f, \sigma, x_n(f, \sigma)) \) for all \( n < \omega \). Since it can easily be shown that \( C_x(f, \sigma) : (C, \sqsubseteq) \rightarrow (C, \sqsubseteq) \) is continuous and \( x_0(f, \sigma) \) is a prefixed point, it follows that the sequence \( x_n(f, \sigma) \) is increasing in the DCPo \((C, \sqsubseteq)\), hence, the least fixed point \((\text{rec} \cdot x \cdot f)(\sigma) = \bigcup_{n < \omega} x_n(f, \sigma) \) above \( x_0(f, \sigma) \) indeed exists and satisfies \((\text{rec} \cdot x \cdot f)(\sigma) = C_x(f, \sigma, (\text{rec} \cdot x \cdot f)(\sigma))\).

Finally, we show that \( \text{rec} \cdot x : D \rightarrow D \) is well-defined, that is, \( \text{rec} \cdot x \cdot f \in D \) for all \( f \in D \), and moreover, relying on Theorem 3, that \( \text{rec} \cdot x : (D, \sqsubseteq) \rightarrow (D, \sqsubseteq) \) is continuous.

### 6.8 The Denotational Construction

The denotational semantics \( [\cdot] : L \rightarrow D \) inductively defined below is uniquely determined by the denotation which has been defined in the previous subsections
for each finitary operator symbol as an operation on \(C\) having the same arity and satisfying the defining conditions of \(D\) (that is, we indeed have \([\text{SKIP}]\), \([c]\), \([p \odot T]\), \([p \mathbin{\boxplus} T]\), \([p : q]\), \([p \cdot q]\), \([p \parallel q]\) \(\in D\) for \(p, q \in \mathcal{L}\)) and the denotation which has been defined for the infinitary recursion operator symbol as an operation on \(D\) (that is, we indeed have \([\text{rec}.x.p]\) \(\in D\) for \(p \in \mathcal{L}\)). Hereby, the functional condition (*) is straightforward to check whereas continuity is increasingly difficult to prove and demands ingenious elaborations. From the above we may thus state the following

**Theorem 4.** The denotational semantics \([\ ] : \mathcal{L} \to D\) inductively defined by

\[
\begin{align*}
[\text{SKIP}](\sigma) &= \text{SKIP}() \\
[x](\sigma) &= c() \\
[p \odot T](\sigma) &= [p](\sigma) \odot T \\
[p \mathbin{\boxplus} T](\sigma) &= [p](\sigma) \mathbin{\boxplus} T \\
[p : q](\sigma) &= [p](\sigma) : [q](\sigma) \\
[\text{rec}.x.p](\sigma) &= (\text{rec} x.[p])(\sigma)
\end{align*}
\]

is well-defined, that is \([p] \in D\) for all \(p \in \mathcal{L}\).

### 6.9 Central Denotational Properties

First, we are able to inductively check the following easy but crucial identity that relates the allocation mapping to the allocation semantics.

**Proposition 1.** If \(p \in \mathcal{L}\), then \(\text{alloc}([p]) = \text{Alloc}(p)\).

The next hard but insightful technical result concerns the interaction of the operators with the action residue. Note the close resemblance of these denotational residuation rules to the operational transition rules previously defined. For all \(u \in \mathcal{A}_R, x, x' \in C\) if \(u^{-1}x = x'\) then we suppose in particular that \(u \subseteq x\).

**Proposition 2.** For any \(a \in \mathcal{A}_R \cup \{0\}\), \(x, x', y, y' \in C\), \(c \in \mathcal{A}_C\), \(T \in A\), \(C \subseteq \mathcal{A}_R\), \(f \in D\), \(\sigma \in C^V\) we have the following properties:

\[
\begin{align*}
\text{[ACT]} & \quad c^{-1}c() = 0 \\
\text{[HID]} & \quad \frac{a^{-1}x = x'}{(a \mathbin{\boxplus} T)^{-1}(x \odot T) = x' \odot T} \\
\text{[RES]} & \quad \frac{a^{-1}x = x'}{a^{-1}(x \odot T) = x' \odot T} \\
\text{[SEQ1]} & \quad \frac{\text{alloc}(x) = 0}{a^{-1}y = y'} \\
\text{[SEQ2]} & \quad \frac{a^{-1}y = y'}{a^{-1}(x ; y) = x ; y'} \\
\text{[SER1]} & \quad \frac{a^{-1}x = x'}{a^{-1}(x \cdot y) = x' \cdot y} \\
\text{[SER2]} & \quad \frac{\text{alloc}(x) \perp a}{a^{-1}(x \cdot y) = x \cdot y'} \\
\text{[SER3]} & \quad \frac{a \in C, a^{-1}x = x', a^{-1}y = y'}{a^{-1}(x \parallel y) = x' \parallel y'} \\
\text{[PAR0]} & \quad \frac{\text{alloc}(y) \perp a}{a^{-1}(x \parallel y) = x' \parallel y'} \\
\text{[PAR1]} & \quad \frac{C, \text{alloc}(x) \perp a}{a^{-1}(x \parallel y) = x' \parallel y'} \\
\text{[PAR2]} & \quad \frac{C, \text{alloc}(x) \perp a}{a^{-1}(x \parallel y) = x \parallel y'} \\
\text{[REC]} & \quad \frac{\text{[rec}.x.f(\sigma)\]}{\{x\}^{-1}y = f(\sigma[x \mapsto y])}
\end{align*}
\]
The next tedious but essential technical result concerns the interaction of the operators with the action prefix.

**Proposition 3.** For any \( a \in \mathbb{A}_R \), \( x, y \in \mathbb{C} \), \( c \in \mathbb{A}_C \), \( T \in \mathbb{A} \), \( C \subseteq \mathbb{A}_R \), \( f \in \mathbb{D} \), \( \sigma \in \mathbb{C}^V \) we have the following properties:

\[
\begin{align*}
[\text{ACT}] \quad & a \sqsubseteq c \implies a = c. \\
[\text{HID}] \quad & a \sqsubseteq x \otimes T \implies u \sqsubseteq ub \sqsubseteq x \text{ for some } u \in \mathbb{A}_R^* \text{ and } b \in \mathbb{A}_R \\
\quad & \text{such that } u \otimes T = 0 \text{ and } (ub) \otimes T = a. \\
[\text{RES}] \quad & a \sqsubseteq x \otimes T \implies a \sqsubseteq x \text{ and } a \leq T. \\
[\text{SEQ}] \quad & a \sqsubseteq x ; y \implies (a \sqsubseteq x) \text{ or } (\text{alloc}(x) = 0 \text{ and } a \sqsubseteq y). \\
[\text{SER}] \quad & a \sqsubseteq x \cdot y \implies (a \sqsubseteq x) \text{ or } (\text{alloc}(x) \perp a \text{ and } a \sqsubseteq y). \\
[\text{PAR}] \quad & a \sqsubseteq x \parallel C y \implies (a \in C \text{ and } a \sqsubseteq x \text{ and } a \sqsubseteq y) \text{ or } \\
\quad & (C, \text{alloc}(y) \perp a \text{ and } a \sqsubseteq x) \text{ or } \\
\quad & (C, \text{alloc}(x) \perp a \text{ and } a \sqsubseteq y). \\
[\text{REC}] \quad & a \sqsubseteq (\text{rec} \cdot f)(\sigma) \implies a = \{x\}.
\end{align*}
\]

### 7 Full Abstraction and Bisimilarity

We finally arrive at the main results of this work relating the denotational to the operational semantics by full abstraction and to bisimilarity via observation. They follow arguments and constructions analogous to the ones previously employed in [4,11,12], thereby heavily relying on the numerous properties proved in the preceding section.

We shall state in this section two results of full abstraction, a linear and a continued one. To this purpose we first exhibit linear translations to and fro between linear transition on the operational side and linear prefix and residue on the denotational side.

The next proposition, which is easy to prove relying on Propositions 1 and 2, allows us to translate linear transition to linear prefix and residue.

**Proposition 4.** If \( u \in \mathbb{A}_R^* \), \( p \in \mathbb{L}_c \), \( p \xrightarrow{u} p' \), then \( p' \in \mathbb{L}_c \), \( u^{-1}[p] = [p'] \).

The next proposition, which is difficult to prove relying on Propositions 1 and 3, allows us to translate linear prefix to linear transition. We only obtain the translation for the large subclass of terms called nice, which have no hiding subterms that hide variables of open subterms. This condition is rather sensible to assume for any practical purposes as argued in [4], wherefrom this notion is inherited in enriched form, as it is in [11,12].

Let \( \preceq \) denote the subterm ordering in \( \mathbb{L} \). We define the language of nice terms \( \mathbb{L}_n = \{ p \in \mathbb{L} \mid \text{if } p_1 \otimes T \preceq p \text{ then } T \wedge V = 0 \text{ or } p_1 \in \mathbb{L}_c \} \) and the language of closed and nice terms \( \mathbb{L}_{c,n} = \mathbb{L}_c \cap \mathbb{L}_n \).

**Proposition 5.** If \( u \in \mathbb{A}_R^* \), \( p \in \mathbb{L}_{c,n} \), \( u \sqsubseteq [p] \) then \( p \xrightarrow{u} . \)
Using the linear translations of Propositions 4 and 5, we are able to state a simple denotational characterization of the linear operational behaviour, that observes strings of actions.

**Theorem 5. (Linear Congruence)** For all \( p \in \mathcal{L}_{c,n} \) we have

\[
\Lambda^*_R(p) = \{ u \in \Lambda^*_R \mid u \subseteq [p] \}
\]

As a major purpose, we next infer a full abstraction result that characterizes the denotational equivalence of processes in terms of equal linear operational observations. The proof is accomplished by probing process terms in suitable hiding contexts. A context \( C(\_\_\_) \) is a term \( C \in \mathcal{L} \) with one distinguished variable denoted by \( \_\_\_ \). A context \( C(\_\_\_) \) is nice-preserving, n.p. for short, iff \( C(p) \in \mathcal{L}_n \) whenever \( p \in \mathcal{L}_{c,n} \).

**Theorem 6. (Linear Full Abstraction)** For all \( p, q \in \mathcal{L}_{c,n} \) we have

\[
[p] = [q] \iff \text{ for all n.p. contexts } C(\_\_\_) \text{ we have } \Lambda^*_R(C(p)) = \Lambda^*_R(C(q))
\]

Using the fact that each multi-pomset \( x \in \mathcal{P} \) is the intersection of its set of linearizations \( \text{Lin}(x) \subseteq \Lambda^*_R \), we derive a notable denotational characterization of the continued operational behaviour, that proves to observe continued multi-pomsets of actions, thus a posteriori justifying the designation.

**Theorem 7. (Continued Congruence)** For all \( p \in \mathcal{L}_{c,n} \) we have

\[
C(p) = K([p]) \quad \text{and} \quad [p] = \bigsqcup C(p)
\]

This allows us to finally state a significant full abstraction result which uses the continued behaviour in order to distinguish between processes.

**Theorem 8. (Continued Full Abstraction)** For all \( p, q \in \mathcal{L}_{c,n} \) we have

\[
[p] = [q] \iff \text{ for all n.p. contexts } C(\_\_\_) \text{ we have } C(C(p)) = C(C(q))
\]

We conclude the presentation with the insightful characterization of bisimilarity as the kernel equivalence of denotational observation, thereby essentially relying on Propositions 4 and 5. We first recall the basic definitions.

A relation \( S \subseteq \mathcal{L}_{c,n} \times \mathcal{L}_{c,n} \) is a simulation iff for all \( p, q, q' \in \mathcal{L}_{c,n} \) and \( u \in \Lambda^*_R \) we have that \( p \xrightarrow{u} q \) and \( q \xrightarrow{u} q' \) implies \( p \xrightarrow{u} p' \) and \( p' \xrightarrow{u} q' \) for some \( p' \in \mathcal{L}_{c,n} \). \( S \) is a bisimulation iff \( S \) and \( S^t \) are simulations.

The coarsest bisimulation \( \approx = \bigcup \{ S \mid S \text{ bisimulation} \} \), called bisimilarity, is known to be an equivalence relation (see for example [8]).

**Theorem 9. (Bisimilarity)** For all \( p, q \in \mathcal{L}_{c,n} \) we have

\[
p \approx q \iff \text{obs}([p]) = \text{obs}([q])
\]
8 Conclusion

This paper set out to model concurrent processes accessing quantities of recyclable resources which are allocated (reserved) during execution and automatically freed (returned) to the environment upon termination. This purpose was achieved by elaborating a truly concurrent denotational and operational semantics for a process language of deterministic concurrent recursive processes.

The process language contains a complete set of deterministic quantitative process operators. The starting point literally for the semantics is a resource semantics that most notably does not add (as in [11,12]) but join (as in [4]) those of sequentially composed parts, which is the very reason why the term allocation is employed instead of consumption.

Next, we displayed a simple deterministic structural operational machine that allows extracting a linear and a continued behaviour. We then defined a rich compositional and continuous denotational semantics over continued multi-pomsets for all finitary operators of our language. Finally, the denotational semantics for the remaining recursion operator was constructed using a functional domain over environments of continued multi-pomsets to which the whole semantics could subsequently be lifted.

The paper concluded by proving that the denotational semantics is fully abstract with respect to the linear and the continued operational semantics, meanwhile closely related to bisimilarity, thus establishing the robustness of the presented semantical work.

Nondeterministic choice is the only operator usually present in classical process languages [7,8] which has been intentionally left out due to the fact that the primary concern of this work was to model deterministic processes. However, a possibility to include the choice operator could consist in enriching multi-pomsets with a further relation on events expressing conflict of choices, thus adopting a modelling view that is closer to event structures [14].

In the present setting the allocation of resources has been modelled using the customary algebra on vectors of extended positive reals. A natural generalization would consist in formalizing the allocation by vectors of abstract weights of an ordered algebra satisfying a set of necessary algebraic laws. Such an endeavour would allow combining more abstractly the domain-theoretic denotational approach with a weighted operational view, thus covering and unifying further resource models.

The developed theoretical approach could be followed by a prospective work on expected applications. For the case of integral resources immediate applications could come from computer science and workflow management. More involved applications may refer to resources that are present in fractional or even continuous amounts corresponding to measures in engineering and economy, where the notion of resource is ubiquitous.

Finally, the presented language can be also employed to abstractly specify and handle labelled partial orders which are considerably more expressive and complex than the series-parallel constructions usually considered in the specialized literature.
References


