Automation
Computers
Applied Mathematics
ISSN 1221–437X
Vol. 21 (2012) no. 1
pp. 3–17

A Study on the Relationship Between Direct Semantics and Continuation Semantics for Concurrency

Eneia Nicolae Todoran

Eneia Nicolae Todoran: Technical University of Cluj-Napoca
Faculty of Automation and Computer Science
Baritiu Street 28, 400027, Cluj-Napoca, Romania

Eneia.Todoran@cs.utcluj.ro

Abstract: We consider an abstract concurrent language $\mathcal{L}$ embodying a mechanism of asynchronous communication. We specify the behavior of programs in the style of Plotkin's structural operational semantics. We consider two different transition system specifications for $\mathcal{L}$. One is designed following the direct approach to concurrency semantics. The other one is designed with continuation semantics for concurrency (CSC). For the language $\mathcal{L}$, we investigate the relationship between direct semantics and CSC. We define a bisimulation relation between the two transition systems and we use some basic techniques of metric semantics to establish the formal relationship between the two semantic models.

Key Words: Operational semantics, metric spaces, continuations for concurrency.

Received: October 1, 2012

1 Introduction

Continuation semantics for concurrency (CSC) [12] is a general tool for designing operational and denotational models of concurrent languages and systems. In denotational models designed with CSC, a continuation is a structured configuration of computations (denotations of program statements) rather than just a function to some answer type as in the classic technique of continuations [11]. Following [1, 12] in this paper we use the term *resumption* as an operational counterpart of the term *continuation*. In an operational model designed with CSC a resumption is a structured configuration of program statements. A general idea in continuation semantics is that a program is decomposed into a current statement and the remainder of the program. A continuation is a representation of such a remainder of the program. The evaluation mechanism of CSC is explained in [12, 13].

Most works employ the direct (rather than the continuation-based) style in designing semantic models of concurrent languages and systems. The reader may consult, e.g., the monograph [1], where the direct style is employed in designing the semantics of parallel composition for various concurrent programming languages. Alternatively, the semantic designer could employ CSC for the same purpose. In this paper we investigate the relationship between an operational semantics designed in direct semantics and an operational semantics designed with CSC. For this purpose we use bisimulation semantics [5, 7, 6] and basic techniques of metric semantics [1].

We consider an abstract concurrent language $\mathcal{L}$ incorporating a general mechanism of asynchronous communication. Based on results in process algebra [4], asynchronous interaction is primitive and synchronous communication could be expressed in terms of asynchronous primitives. The relation between synchronous and asynchronous interaction has received considerable attention in recent years [8, 14]. Asynchronous interaction represents a basic mechanism in various modern distributed systems, Internet and Web applications. The language $\mathcal{L}$ that we consider in this paper embodies the paradigm of asynchronous communication studied in [3]; instances of this paradigm include asynchronous CSP, dataflow and concurrent constraint programming [10]. $\mathcal{L}$ extends the paradigmatic language studied in [3] with recursion.

We specify the behavior of $\mathcal{L}$ programs in an operational manner in the style of Plotkin's structural operational semantics [9]. We consider two different transition system specifications for $\mathcal{L}$. One is designed following the direct approach to concurrency semantics as, e.g., in [1]. The other one is designed with CSC. We establish the formal relationship between direct semantics and continuation semantics for concurrency by defining a bisimulation relation between the two transition systems. By using this bisimulation relation and an argument of the kind '$\epsilon \leq \frac{1}{2} \cdot \epsilon \Rightarrow \epsilon = 0$', which is standard in metric semantics, we prove that the two operational semantics assign the same meaning to any $\mathcal{L}$ program. As an easy consequence, any property or concurrency law that holds in a semantic model designed in direct semantics also holds in a semantic model designed with CSC. For example, parallel composition is associative and commutative in the semantic model designed in direct semantics (see, e.g., [1, 6]). This means that parallel composition is also associative and commutative in the semantic model designed with CSC.

2 Theoretical preliminaries

The notation $(x \in)X$ introduces the set X with typical element x ranging over X. For any set X, we denote by $|X|$ the *cardinal number* of X. $|X| = 0$ when the set X is empty. For

any set X we denote by $\mathcal{P}_\pi(X)$ the collection of all subsets of X which have property π. Let $f \in X \to Y$ be a function. The function $(f \mid x \mapsto y) : X \to Y$, is defined (for $x, x' \in X, y \in Y$) by:

$$(f \mid x \mapsto y)(x') = \begin{cases} y & \text{if } x' = x \\ f(x') & \text{if } x' \neq x \end{cases}$$

We write $(f \mid x_1 \mapsto y_1 \mid \cdots \mid x_n \mapsto y_n)$ as an abbreviation for $(\cdots (f \mid x_1 \mapsto y_1) \cdots \mid x_n \mapsto y_n)$. If $f : X \to X$ and $f(x) = x$ we call x a *fixed point* of f. When this fixed point is unique (see Theorem 2.2) we write $x = fix(f)$. We assume known the notion of a *partially ordered set*. We recall that, given a partially ordered set $(X, \leq_X)$, an element $x \in X$ is said to be *maximal* if there are no elements strictly greter than x in X, that is if $x \leq_X y$ then $y \leq_X x$ in which case $x = y$.

2.1 Metric spaces

Following [1], the study presented in this paper takes place in the mathematical framework of *1-bounded complete metric spaces*. We assume known the following notions: *metric* (and *ultrametric*) space, *isometry* (distance preserving bijection between metric spaces; we denote it by '$\cong$') and *completeness* of a metric space.

Example 2.1 *(a) Let $(a, b \in)A$ be a set. The so-called* discrete metric *d_A on A is defined as follows. $d_A(a, b) =$ if $(a = b)$ then 0 else 1. For any set A, (A, d_A) is a complete metric space.*

(b) Let $(a \in)A$ be a nonempty set, and let $(x, y \in)A^\infty = A^* \cup A^\omega$, where $A^*(A^\omega)$ is the set of all finite (infinite) sequences over A. One can define a metric over A^∞, by $d(x, y) = 2^{-sup\{n \mid x(n)=y(n)\}}$, where $x(n)$ denotes the prefix of x of length n, in case $length(x) \geq n$, and x otherwise. Also, by convention $2^{-\infty} = 0$. d is a Baire-like metric. (A^∞, d) is a complete ultrametric space.

We recall that if (X, d_X), (Y, d_Y) are metric spaces, a function $f{:}X \to Y$ is a *contraction* if $\exists k \in \mathbb{R}$, $0 \leq k < 1 \forall x_1, x_2 \in X : d_Y(f(x_1), f(x_2)) \leq k \cdot d_X(x_1, x_2)$. When $k = 1$ the function f is called *non-expansive*. In the sequel we denote the set of all k-contracting (nonexpansive) functions from X to Y by $X \xrightarrow{k} Y$ ($X \xrightarrow{1} Y$).

Theorem 2.2 *(Banach) Let (X, d_X) be a complete metric space. Each contracting function $f : X \to X$ has a* unique *fixed point.*

Definition 2.3 *Let $(X, d_X), (Y, d_Y)$ be (ultra) metric spaces. On $(x \in)X$, $(f \in)X \to Y$ (the function space), $((x, y) \in)X \times Y$ (the cartesian product), $(u, v \in)X + Y$ (the disjoint union of X and Y, which can be defined by $X + Y = (\{1\} \times X) \cup (\{2\} \times Y)$) and $(U, V \in)\mathcal{P}(X)$ (the power set of X), one can define the following metrics:*

(a) $d_{\frac{1}{2} \cdot X} : X \times X \to [0, 1]$, $\quad d_{\frac{1}{2} \cdot X}(x_1, x_2) = \frac{1}{2} \cdot d_X(x_1, x_2)$

(b) $d_{X \to Y} : (X \to Y) \times (X \to Y) \to [0, 1]$, $\quad d_{X \to Y}(f_1, f_2) = sup_{x \in X} d_Y(f_1(x), f_2(x))$

(c) $d_{X \times Y} : (X \times Y) \times (X \times Y) \to [0, 1]$

$$d_{X\times Y}((x_1,y_1),(x_2,y_2)) = max\{d_X(x_1,x_2), d_Y(y_1,y_2)\}$$

(d) $d_{X+Y} : (X+Y)\times(X+Y) \to [0,1]$

$$d_{X+Y}(u,v) = \text{ if } (u,v \in X) \text{ then } d_X(u,v) \text{ else if } (u,v \in Y) \text{ then } d_Y(u,v) \text{ else } 1$$

(e) $d_H : \mathcal{P}(X)\times\mathcal{P}(X) \to [0,1]$, $d_H(U,V) = max\{sup_{u\in U} d(u,V), sup_{v\in V} d(v,U)\}$ *where* $d(u,W)=inf_{w\in W} d(u,w)$ *and by convention* $sup\emptyset=0$, $inf\emptyset=1$ *(*d_H *is the* Hausdorff *distance).*

We recall that, given a metric space (X, d_X), a subset A of X is called *compact* whenever each sequence in A has a convergent subsequence with limit in A. We use the abbreviations $\mathcal{P}_{co}(\cdot)$, $\mathcal{P}_{nco}(\cdot)$ and $\mathcal{P}_{finite}(\cdot)$ to denote the power sets of *compact*, *non-empty and compact* and *finite* subsets of '$\cdot$', respectively. Also, we often suppress the metrics part in domain definitions, and write, e.g., $\frac{1}{2}\cdot X$ instead of $(X, d_{\frac{1}{2}\cdot X})$.

Remark 2.4 *Let* $(X,d_X),(Y,d_Y), d_{\frac{1}{2}\cdot X}$, $d_{X\to Y}$, $d_{X\times Y}$, d_{X+Y} *and* d_H *be as in definition 2.3. In case* d_X, d_Y *are ultrametrics, so are* $d_{\frac{1}{2}\cdot X}, d_{X\to Y}, d_{X\times Y}, d_{X+Y}$ *and* d_H*. If in addition* $(X,d_X),(Y,d_Y)$ *are complete then* $\frac{1}{2}\cdot X$, $X\to Y$, $X \xrightarrow{1} Y$, $X\times Y, X+Y$, *and* $\mathcal{P}_{\mathrm{nco}}(X)$ *(with the metrics defined above) are also complete metric spaces.*

2.2 Bisimulation

A labelled transition system T is a triple $(Conf, Obs, \to)$, where $(c \in)Conf$ is a set of *configurations*, $(a \in)Obs$ is a set of *observations* and $\to$ is a subset of $Conf \times Obs \times Conf$, i.e. $\to \subseteq Conf \times Obs \times Conf$. Usually, a triple (c,a,c') is written in the notation $c \xrightarrow{a} c'$.

Let $T = (Conf, Obs, \to)$ be a transition system. The concept of a *bisimulation* was introduced by Milner [5, 6] and Park [7]. A (strong) bisimulation on T is a relation $R \subseteq Conf \times Conf$, satisfying for all $(c_1,c_2) \in R$:

(a) For all a, c_1' if $c_1 \xrightarrow{a} c_1'$ then there exists c_2' such that $c_2 \xrightarrow{a} c_2'$ and $(c_1', c_2') \in R$.

(b) For all a, c_2' if $c_2 \xrightarrow{a} c_2'$ then there exists c_1' such that $c_1 \xrightarrow{a} c_1'$ and $(c_1', c_2') \in R$.

An unlabelled state transition system is a tuple $(Conf, \to)$ where $(c \in)Conf$ is a set (of configurations or states) and $\to \subseteq Conf \times Conf$ is a binary relation over $Conf$ (of transitions). If $c, c' \in Conf$ one usually expresses the fact that $(c,c') \in \to$ as $c \longrightarrow c'$.

A labelled transition system where the set of labels consists of only one element is equivalent to an unlabelled transition system. In this paper we only deal with unlabelled transition systems.

3 Syntax of $\mathcal{L}$ and notation conventions

We assume given a set $(a \in)Act$ of *atomic actions* and a set $(x \in)Pvar$ of *procedure variables.*

Definition 3.1 *(Syntax of* $\mathcal{L}$*)*

(a) (Statements) $s(\in Stat) ::= a \mid x \mid s+s \mid s;s \mid s \between s \mid s \parallel s$

(b) (Guarded statements) $g(\in GStat) ::= a \mid g + g \mid g; s \mid g \between s \mid g \parallel g$

(c) (Declarations) $(D \in)Decl = PVar \to GStat$

(d) (Programs) $(\pi \in)Prog = Decl \times Stat$

$;, +$ and $\parallel$ are operators for sequential, alternative (nondeterministic) and parallel composition, respectively. $\parallel$ is also called a *merge* operator, and $\between$ is the *left merge* operator. There is a special symbol $\delta \in Act$, whose behavior is explained below. $(g \in)G$ is the class of *guarded statements*. Recursion is defined based on a set $(D \in)Decl = Y \to G$ of *declarations*.

The meaning of atomic actions is defined by an interpretation function $I : Act \to \Sigma \to (\{\uparrow\} \cup \Sigma)$, where $(\sigma \in)\Sigma$ is a set of *states*. If $I(a)(\sigma) = \uparrow$ the action a cannot proceed in state σ; its execution is *suspended*. When all processes are suspended *deadlock* occurs. Notice that $I(\delta)(\sigma) = \uparrow, \forall \sigma \in \Sigma$, i.e. the action δ suspends in all states. $\mathcal{L}$ incorporates the mechanism of *asynchronous communication* studied in [3]. As explained in [3], this form of asynchronous communication can be encountered in concurrent constraint programming, and also in other languages like dataflow or asynchronous CSP.

The semantics of $\mathcal{L}$ will be defined by means of two transition systems T_D and T_C. In each case, a transition relation of a type $C \times C$ will be induced, where $(c \in)C$ is a class of *configurations*. The elements (c, c') of $C \times C$ will be written in the form $c \longrightarrow c'$ in the case of T_D. Similarly, for T_C we will use the notation $c \Longrightarrow c'$. In the definitions of T_D and T_C we use the following conventions:

$$c_1 \longrightarrow_0 c_2 \quad \text{is an abbreviation for} \quad \frac{c_2 \longrightarrow c'}{c_1 \longrightarrow c'}$$

and

$$c_1 \Longrightarrow_0 c_2 \quad \text{is an abbreviation for} \quad \frac{c_2 \Longrightarrow c'}{c_1 \Longrightarrow c'}$$

For the sake of brevity (and without loss of generality) in the sequel we assume given some fixed declaration $D(\in Decl)$ and all considerations in any given argument refer to this fixed D.

4 Direct semantics for $\mathcal{L}$

The operational semantics given in this this section is designed following the traditional direct approach to concurrency semantics (as, e.g, in [1]).

Definition 4.1 *Let* $u(\in U) ::= E \mid s$. *The set of* configurations *for direct semantics is* $Conf_D = U \times \Sigma$.

By convention, in the sequel we identity $E; s$, $E \parallel s$ and $s \parallel E$ with s, i.e., for each $s \in Stat$ we have $E; s \equiv E \parallel s \equiv s \parallel E \equiv s$, where '$\equiv$' denotes syntactic identity. Also, we identity $E \parallel E$ with E, i.e. $E \parallel E \equiv E$.

Definition 4.2 *(T_D) The transition relation for $\mathcal{L}$ is the smallest subset of $Conf_D \times Conf_D$ satisfying the following axioms and rules*

(A1) $(a, \sigma) \longrightarrow (E, \sigma')$ $\qquad$ if $I(a)(\sigma) = \sigma' \quad (\neq \uparrow)$

(R2) $(x,\sigma) \longrightarrow_0 (D(x),\sigma)$

(R3) $(s_1 + s_2,\sigma) \longrightarrow_0 (s_1,\sigma)$

(R4) $(s_1 + s_2,\sigma) \longrightarrow_0 (s_2,\sigma)$

(R5) $(s_1 \parallel s_2,\sigma) \longrightarrow_0 (s_1 \between s_2,\sigma)$

(R6) $(s_1 \parallel s_2,\sigma) \longrightarrow_0 (s_2 \between s_1,\sigma)$

(R7) $\dfrac{(s,\sigma)\longrightarrow(u,\sigma')}{(s;s',\sigma)\longrightarrow(u;s',\sigma')}$

(R8) $\dfrac{(s,\sigma)\longrightarrow(u,\sigma')}{(s \between s',\sigma)\longrightarrow(u \parallel s',\sigma')}$

According to axiom (A1) an atomic action $a \in Act$ can only be executed in those states σ where $I(a)(\sigma) \neq \uparrow$; it's execution is suspended until a state $\overline{\sigma}$ is reached where $I(a)(\overline{\sigma}) \neq \uparrow$. We use the notation $(u,\sigma) \not\longrightarrow$ to express the fact that (u,σ) has no transitions. For all $(u,\sigma) \in U \times \Sigma$ it can be decided whether $(u,\sigma) \not\longrightarrow$, by induction on the complexity measure c_u given in Definition 4.5.

Definition 4.3 *(Semantic universe)* $\mathbf{P} = \mathcal{P}_{nco}(\Sigma_\delta^\infty)$*, where* $\Sigma_\delta^\infty = \Sigma^* \cup \Sigma^*\{\delta\} \cup \Sigma^\omega$*, i.e.,* Σ_δ^∞ *is the collection of all finite sequences of elements of type* Σ *(states), possibly terminated with the symbol* δ*, together with the collection of all infinite sequences of elements of type* Σ*. As a metric on* $\mathbf{P}$ *we take* $(d_B)_H$*, i.e. the Hausdorff distance induced by the Baire metric on* Σ_δ^∞*.*

In this paper we use the symbol '$\cdot$' as a concatenation operator over sequences. In particular, if $\sigma \in \Sigma$ and $q \in \Sigma_\delta^\infty$ then $\sigma \cdot q \in Act_\delta^\infty$ is the sequence obtained by prefixing the state σ to the sequence q. Also, we use the notation $\sigma \cdot p = \{\sigma \cdot q \mid q \in p\}$, for any $\sigma \in \Sigma$, $p \in \mathbf{P}$.

Definition 4.4 *(Operational semantics* $\vec{O}$*) for* $\mathcal{L}$*)*

(a) Let $(S \in) Sem_O = U \to \mathbf{P}$ *and let* $\Phi : Sem_O \to Sem_O$ *be given by:*

$$\Phi(S)(E,\sigma) = \{\epsilon\}$$
$$\Phi(S)(s,\sigma) = \{\delta\} \qquad \text{if } (s,\sigma) \not\longrightarrow$$
$$\Phi(S)(s,\sigma) = \bigcup\{\sigma \cdot S(u,\sigma') \mid (s,\sigma)\longrightarrow(u,\sigma')\}$$

(b) We put $\vec{O} = fix(\Phi)$ *and define* $\vec{\mathcal{O}}[\![\cdot]\!] : Stat \to \Sigma \to \mathbf{P}$ *by* $\vec{\mathcal{O}}[\![s]\!]\sigma = \vec{O}(s,\sigma)$*.*

One can prove that T_D is *finitely branching*[1] (and thus it induces a compact operational semantics; see, e.g., [1]), by induction on the complexity measure defined in Definition 4.5. Also, a mapping like Φ is contracting in particular due to the "$\sigma \cdot \ldots$"-step in its definition. We say that the configuration (E,σ) *terminates*. Also, when $(s,\sigma) \not\longrightarrow$ we say that (s,σ) *blocks*.

For inductive reasonings we use the complexity measure given in Definition 4.5, as in [1].

[1] i.e., the set $\{(u',\sigma') \mid (u,\sigma)\longrightarrow(u',\sigma')\}$ is finite for any $(u,\sigma) \in Conf_D$.

Definition 4.5 *(Complexity measure) Let* $c_s : Stat \to \mathbb{N}$ *be given by*

$$\begin{array}{rcl} c_s(a) & = & 1 \\ c_s(x) & = & 1 + c_s(D(x)) \\ c_s(s_1 \ \mathsf{op} \ s_2) & = & 1 + c_s(s_1) \quad \mathsf{op} \in \{;, \between\} \\ c_s(s_1 \ \mathsf{op} \ s_2) & = & 1 + max\{c_s(s_1), c_s(s_2)\} \quad \mathsf{op} \in \{+, \|\} \end{array}$$

Also, we define $c_u : U \to \mathbb{N}$ *as follows:* $c_u(E) = 0$ *and* $c_u(s) = c_s(s)$.

5 Continuation semantics for $\mathcal{L}$

In the CSC approach [12] continuations are structured configurations of computations. The structure of continuations is specific of the (concurrent) language under investigation. In the case of a language like $\mathcal{L}$, which combines parallel composition with a general operator for sequential composition, a continuation is a tree with active computations (statements) at the leaves [12, 13]. Following [1] we use the term *resumption* as an operational counterpart of the term *continuation*.

5.1 Structure of resumptions

In order to define such trees of computations we employ a partially ordered set of *identifiers* $(Id, \leq)$. $(\alpha, \beta \in)Id$ is the set of all finite, possibly empty (ϵ), sequences over $\{1, 2\}$ and $\alpha \leq \alpha'$ iff α is a prefix of α'.

Definition 5.1

(a) Let $(\alpha \in)Id = \{1, 2\}^*$ *be a set of* identifiers*, equipped with the following partial ordering:* $\alpha \leq \alpha$*' iff* $\alpha' = \alpha \cdot i_1 \cdots i_n$ *for* $i_1, \cdots, i_n \in \{1, 2\}, n \geq 0$.

(b) We define a function $max : \mathcal{P}(Id) \to \mathcal{P}(Id)$ *by:*

$$max(A) = \{\alpha \mid \alpha \text{ is a maximal element of } (A, \leq_A)\}$$

$A \in \mathcal{P}(Id)$ *and* $\leq_A$ *is the restriction of* $\leq$ *to the* A.

The construct $(Id, \leq)$ can be used to represent tree-like structures. For example, let $A = \{\alpha,\ \alpha \cdot 1,$
$\alpha \cdot 2, \alpha \cdot 1 \cdot 1, \alpha \cdot 1 \cdot 2, \alpha \cdot 2 \cdot 1, \alpha \cdot 2 \cdot 2\}$. The maximal elements of $(A, \leq_A)$ are the *leaves* of the tree: $max(A) = \{\alpha \cdot 1 \cdot 1, \alpha \cdot 1 \cdot 2, \alpha \cdot 2 \cdot 1, \alpha \cdot 2 \cdot 2\}$.

Let $(x \in)X$ be a set. We use the following notation:

$$\{\!|X|\!\}^{Id} \stackrel{not.}{=} \mathcal{P}_{finite}(Id) \times (Id \to X)$$

Let $\alpha \in Id, (\pi, \varphi) \in \{\!|X|\!\}^{Id}$ with $\pi \in \mathcal{P}_{finite}(Id)$, $\varphi \in Id \to X$. We define $id : \{\!|X|\!\}^{Id} \to \mathcal{P}_{finite}(Id)$, $id(\pi, \varphi) = \pi$. We also use the following abbreviations:

$$\begin{array}{rcll} (\pi, \varphi)(\alpha) & \stackrel{not.}{=} & \varphi(\alpha) & (\in X) \\ (\pi, \varphi) \setminus \alpha & \stackrel{not.}{=} & (\pi \setminus \{\alpha\}, \varphi) & (\in \{\!|X|\!\}^{Id}) \\ (\pi, \varphi) \cap \pi' & \stackrel{not.}{=} & (\pi \cap \pi', \varphi) & (\in \{\!|X|\!\}^{Id}) \\ ((\pi, \varphi) \mid \alpha \mapsto x) & \stackrel{not.}{=} & (\pi \cup \{\alpha\}, (\varphi \mid \alpha \mapsto x)) & (\in \{\!|X|\!\}^{Id}) \end{array}$$

The basic idea is that we treat (π, φ) as a 'function' with finite graph $\{(\alpha, \varphi(\alpha)) \mid \alpha \in \pi\}$, thus ignoring the behaviour of φ for any $\alpha \notin \pi$ (π is the 'domain' of (π, φ)). We use this mathematical structure to represent finite partially ordered *bags* (or multisets)[2] of computations. The set Id is used to distinguish between multiple occurrences of a computation in such a bag. The operators behave as follows. $id(\pi, \varphi)$ returns the collection of identifiers for the valid computations contained in the bag (π, φ), $(\pi, \varphi)(\alpha)$ returns the computation with identifier α, $(\pi, \varphi) \setminus \alpha$ removes the computation with identifier α, $(\pi, \varphi) \cap \pi'$ removes all computations with identifiers $\notin \pi'$, and $((\pi, \varphi) \mid \alpha \mapsto x)$ replaces the computation with identifier α.

5.2 Operational semantics

In this subsection we present a transition system T_C designed with continuations for $\mathcal{L}$.

Definition 5.2

(a) (Resumptions) We define the set of closed resumptions *by* $(r \in)Res = \{\!|Stat|\!\}^{Id}$. *Also, we define the set of* open resumptions *by* $ORes = \{(\alpha, r) \mid (\alpha, r) \in Id \times Res \text{ and } \nu(\alpha, r)\}$, *where* $\nu : Id \times Res \to Bool$ *is given by* $\nu(\alpha, r) = (\alpha \notin id(r))$ and $(\alpha \in max(\{\alpha\} \cup id(r)))$. *Stat is the class of* $\mathcal{L}$ *statements given in Definition 3.1.*

(b) (Configurations) The class of configurations *for continuation semantics is given by* $(t \in)Conf_C = (Res \cup (Stat \times ORes)) \times \Sigma$. *We also let* w *range over* $(w \in)W = Res \cup (Stat \times ORes)$. *A configuration is either a pair* (r, σ) *with* $r \in Res$ *and* $\sigma \in \Sigma$, *or a stucture* $((s, (\alpha, r)), \sigma) \in (Stat \times ORes)) \times \Sigma$, *with* $s \in Stat$. $(\alpha, r) \in ORes$ *and* $\sigma \in \Sigma$. *We define the predicate* $empty : W \to Bool$, $empty(r) = (id(r) = \emptyset)$, *and* $empty(s, (\alpha, r)) = (id(r) = \emptyset)$.

Definition 5.3 *(T_C) The transition relation for $\mathcal{L}$ is the smallest subset of $Conf_C \times Conf_C$ satisfying the following axioms and rules:*

(A1) $((a, (\alpha, r)), \sigma) \Longrightarrow (r, \sigma')$ if $I(a)(\sigma) = \sigma'$ $(\neq\uparrow)$

(R2) $((x, (\alpha, r)), \sigma) \Longrightarrow_0 ((D(x), (\alpha, r)), \sigma)$

(R3) $((s_1 + s_2, (\alpha, r)), \sigma) \Longrightarrow_0 ((s_1, (\alpha, r)), \sigma)$

(R4) $((s_1 + s_2, (\alpha, r)), \sigma) \Longrightarrow_0 ((s_2, (\alpha, r)), \sigma)$

(R5) $((s_1; s_2, (\alpha, r)), \sigma) \Longrightarrow_0 ((s_1, (\alpha \cdot 1, (r \mid \alpha \mapsto s_2))), \sigma)$

(R6) $((s_1 \parallel s_2, (\alpha, r)), \sigma) \Longrightarrow_0 ((s_1, (\alpha \cdot 1, (r \mid \alpha \cdot 2 \mapsto s_2))), \sigma)$

(R7) $((s_1 \parallel s_2, (\alpha, r)), \sigma) \Longrightarrow_0 ((s_2, (\alpha \cdot 2, (r \mid \alpha \cdot 1 \mapsto s_1))), \sigma)$

(R8) $(r, \sigma) \Longrightarrow_0 ((r(\alpha), (\alpha, r \setminus \alpha)), \sigma)$ $\forall \alpha \in max(id(r))$ if $id(r) \neq \emptyset$

[2] We avoid using the notion of a *partially ordered multiset* which is a more refined structure – see [2], or ch. 16 of [1].

One can prove that T_C is finitely branching, and thus it induces a compact operational semantics (see, e.g., [1]), by induction on the complexity measure given in Definition 5.4. As in section 4, we use the notation $t \not\rightarrow$ to express the fact that t has no transitions. Also, one can prove that for all $t \in Conf_C$ it can be decided whether $t \not\rightarrow$. The both proofs can proceed by induction on the complexity measure c_w given in Definition 5.4. If $t = (r, \sigma)$ and $id(r) = \emptyset$ we say that t *terminates*. If $t \neq (r, \sigma)$ when $id(r) = \emptyset$ and $t \not\rightarrow$ we say that t *blocks*.

Definition 5.4 *(Complexity measure) Let $c_s : Stat \to \mathbb{N}$ be as in Definition 4.5. We define $c_w : W \to \mathbb{N}$ as follows: $c_w((s, (\alpha, r))) = c_s(s)$, $c_w(r) = 0$ if $id(r) = \emptyset$ and $c_w(r) = 1 + max\{c_s(r(\alpha)) \mid \alpha \in max(id(r)))\}$ if $id(r) \neq \emptyset$.*

Lemma 5.5 *If $t \Longrightarrow t'$ then $t' \in Res \times \Sigma$, i.e. the transition relation induced by T_C is a subset of $Conf_C \times (Res \times \Sigma)$. This is why in the sequel we will only deal with transitions of the form $t \Longrightarrow (r, \sigma)$, with $r \in Res$.*

Proof The proof can proceed by induction on $c_w(w)$ in two steps: first for all $w \in (Stat \times ORes)$ and next for all $w \in Res$. □

Definition 5.6 *(Operational semantics ($\overrightarrow{\overrightarrow{O}}$) for $\mathcal{L}$) Let $\mathbf{P} = \mathcal{P}_{nco}(\Sigma_\delta^\infty)$ be as in Definition 4.3.*

(a) Let $(S \in) Conf_C \to \mathbf{P}$ and let $\Phi : (Conf_C \to \mathbf{P}) \to (Conf_C \to \mathbf{P})$ be given by:

$$\Phi(S)(t) = \{\epsilon\} \qquad \text{if } t \text{ terminates}$$
$$\Phi(S)(t) = \{\delta\} \qquad \text{if } t \text{ blocks}$$
$$\Phi(S)(t) = \bigcup\{\sigma \cdot S(r, \sigma') \mid t \Longrightarrow (r, \sigma')\} \qquad \text{otherwise}$$

(b) We put $\overrightarrow{\overrightarrow{O}} = fix(\Phi)$ and define $\overrightarrow{\overrightarrow{\mathcal{O}}}[\![\cdot]\!] : Stat \to \Sigma \to \mathbf{P}$ by $\overrightarrow{\overrightarrow{\mathcal{O}}}[\![s]\!]\sigma = \overrightarrow{\overrightarrow{O}}((s, (\epsilon, (\emptyset, \lambda\alpha\,.\,\delta))), \sigma)$.

6 Relation between direct semantics and continuation semantics

In this section we investigate the relation between $\overrightarrow{O}$ and $\overrightarrow{\overrightarrow{O}}$. We show that $\overrightarrow{O}(stat(w), \sigma) = \overrightarrow{\overrightarrow{O}}(w, \sigma)$, and, as a consequence $\overrightarrow{\mathcal{O}}[\![s]\!] = \overrightarrow{\overrightarrow{\mathcal{O}}}[\![s]\!], \forall s \in Stat$, where the mapping *stat* is defined in Definition 6.1.

Definition 6.1

(a) For any $r \in Res$ we define $r_{=\alpha}, r_{\geq\alpha}, r_{>\alpha} \in Res$ as follows:[3]

$$r_{=\alpha} = r \cap \{\alpha\}$$
$$r_{\geq\alpha} = r \cap \{\alpha' \mid \alpha' \in id(r), \alpha' \geq \alpha\}$$
$$r_{>\alpha} = r \cap \{\alpha' \mid \alpha' \in id(r), \alpha' > \alpha\}$$

If $w = (s, (\alpha, r)) \in W$ we define:

$$w_{\geq\alpha_0} = (s, (\alpha, r_{\geq\alpha_0}))$$

[3] *The notation $r \cap \pi$ was introduced in subsection 5.1.*

$$w_{>\alpha_0} = (s, (\alpha, r_{>\alpha_0}))$$

Note that $r = r_{\geq\epsilon}$ *and* $w = w_{\geq\epsilon}$.

(b) We define $\uplus : Res \times Res \to Res$ *as follows (we use the infix notation for* $\uplus$*):*

$$(r_1 \uplus r_2)(\alpha) = \text{ if } \alpha \in id(r_1) \text{ then } r_1(\alpha) \text{ else } r_2(\alpha)$$

Also, we use the notations

$$(s, (\alpha, r_1)) \uplus r_2 \stackrel{not.}{=} (s, (\alpha, r_1 \uplus r_2))$$
$$r_1 \uplus (s, (\alpha, r_2)) \stackrel{not.}{=} (s, (\alpha, r_1 \uplus r_2))$$

(c) We define $stat : W \to U$*, for any* $\varphi \in (Id \to Stat), \alpha \in Id, s \in Stat$ *as follows:*

$$stat(\emptyset, \varphi) = E$$
$$stat((\emptyset, \varphi) \mid \alpha \mapsto s) = s$$
$$stat(r_{>\alpha} \uplus r_{=\alpha}) = stat(r_{>\alpha}); stat(r_{=\alpha})$$
$$stat(r_{\geq\alpha\cdot 1} \uplus r_{\geq\alpha\cdot 2}) = stat(r_{\geq\alpha\cdot 1}) \| stat(r_{\geq\alpha\cdot 2})$$
$$stat(s, (\alpha, (\emptyset, \varphi))) = s$$
$$stat(s, (\alpha, r_{>\alpha_0} \uplus r_{=\alpha_0})) = stat(s, (\alpha, r_{\geq\alpha_0})); stat(r_{=\alpha_0}) \quad \text{if } \alpha_0 \in id(r) \text{ and } \alpha > \alpha_0$$
$$stat(s, (\alpha, r_{\geq\alpha_0\cdot 1} \uplus r_{=\alpha_0\cdot 2})) = stat(s, (\alpha, r_{\geq\alpha_0\cdot 1})) \between stat(r_{\geq\alpha_0\cdot 2})$$
$$\text{if } id(r_{\geq\alpha_0\cdot 2}) \neq \emptyset \text{ and } \alpha \geq \alpha_0 \cdot 1$$
$$stat(s, (\alpha, r_{\geq\alpha_0\cdot 2} \uplus r_{=\alpha_0\cdot 2})) = stat(s, (\alpha, r_{\geq\alpha_0\cdot 2})) \between stat(r_{\geq\alpha_0\cdot 1})$$
$$\text{if } id(r_{\geq\alpha_0\cdot 1}) \neq \emptyset \text{ and } \alpha \geq \alpha_0 \cdot 2$$

Definition 6.2

(a) We define $\simeq_d \subseteq Conf_D \times Conf_D$ *as follows:* $(u_1, \sigma) \simeq_d (u_2, \sigma)$ iff $(u_1, \sigma) \longrightarrow (u', \sigma') \Leftrightarrow (u_2, \sigma) \longrightarrow (u', \sigma')$.

(b) We define $\simeq_c \subseteq Conf_C \times Conf_C$ *as follows:* $(w_1, \sigma) \simeq_c (w_2, \sigma)$ iff $(w_1, \sigma) \Longrightarrow (r', \sigma') \Leftrightarrow (w_2, \sigma) \Longrightarrow (r', \sigma')$.

The main results of the paper are given in Theorem 6.4. Lemma 6.3 is needed in the proof of Theorem 6.4. Also, the Appendix contains two technical Lemmas, A.1 and A.2, that are needed in the proof of Lemma 6.3.

Lemma 6.3 $S \subseteq (Conf_D \cup Conf_C) \times (Conf_D \cup Conf_C)$ *defined by* $S = \{((w, \sigma), (stat(w), \sigma)) \mid w \in W, \sigma \in \Sigma\}$ *is a (strong) bisimulation with respect to* $\Longrightarrow \cup \longrightarrow$.

Proof We have to prove that $\forall((w, \sigma), (stat(w), \sigma)) \in S$:

(i) Whenever $(w, \sigma) \Longrightarrow (r, \sigma')$ then, for some u, σ', $(stat(w), \sigma) \longrightarrow (u, \sigma')$ and $u = stat(r)$ (i.e. $((r, \sigma'), (u, \sigma')) \in S$).

(ii) Whenever $(stat(w), \sigma) \longrightarrow (u, \sigma')$ then, for some r, σ', $(w, \sigma) \Longrightarrow (r, \sigma')$ and $u = stat(r)$ (i.e. $((r, \sigma'), (u, \sigma')) \in S$).

We proceed by induction on $c_u(stat(w))$ for both (i) and (ii). We only treat (i) and only consider the subcase when $stat(w) = s_1 \parallel s_2$, for some $s_1, s_2 \in Stat$. This happens in either of the following subcases:

$w = ((\emptyset, \varphi) \mid \alpha \mapsto (s_1 \parallel s_2))$, for some $\varphi \in Id \to Stat, \alpha \in Id$,

$w = (s_1 \parallel s_2, (\alpha, (\emptyset, \varphi)))$, for some $\varphi \in Id \to Stat, \alpha \in Id$,

$w = r_{\geq \alpha \cdot 1} \uplus r_{\geq \alpha \cdot 2}$, for some $r \in Res$ when $\neg(empty(r_{\geq \alpha \cdot 1}))$ and $\neg(empty(r_{\geq \alpha \cdot 2}))$.

$((\emptyset, \varphi) \mid \alpha \mapsto (s_1 \parallel s_2)) \simeq_c (s_1 \parallel s_2, (\alpha, (\emptyset, \varphi))) \simeq_c ((\emptyset, \varphi) \mid \alpha \cdot 1 \mapsto s_1 \mid \alpha \cdot 2 \mapsto s_2)$, by Lemma A.1(c) and A.1(f). Also, $((\emptyset, \varphi) \mid \alpha \cdot 1 \mapsto s_1 \mid \alpha \cdot 2 \mapsto s_2) = ((\emptyset, \varphi) \mid \alpha \cdot 1 \mapsto s_1) \uplus ((\emptyset, \varphi) \mid \alpha \cdot 2 \mapsto s_2)$. Hence, it suffices to study the behavior of a configuration (w, σ) where $w = r_{\geq \alpha \cdot 1} \uplus r_{\geq \alpha \cdot 2}$, for some $r \in Res$ when $\neg(empty(r_{\geq \alpha \cdot 1}))$ and $\neg(empty(r_{\geq \alpha \cdot 2}))$. By A.2(j)

$$(r_{\geq \alpha \cdot 1} \uplus r_{\geq \alpha \cdot 2}, \sigma) \Longrightarrow (r', \sigma') \Leftrightarrow$$
$$[[((r_{\geq \alpha \cdot 1}, \sigma) \Longrightarrow (r'_{\geq \alpha \cdot 1}, \sigma')) \wedge (r' = r'_{\geq \alpha \cdot 1} \uplus r_{\geq \alpha \cdot 2})] \vee$$
$$[((r_{\geq \alpha \cdot 2}, \sigma) \Longrightarrow (r'_{\geq \alpha \cdot 2}, \sigma')) \wedge (r' = r_{\geq \alpha \cdot 1} \uplus r'_{\geq \alpha \cdot 2})]]$$

If $r' = r'_{\geq \alpha \cdot 1} \uplus r_{\geq \alpha \cdot 2}$ and $(r_{\geq \alpha \cdot 1}, \sigma) \Longrightarrow (r'_{\geq \alpha \cdot 1}, \sigma')$ then, as $c_u(stat(r_{\geq \alpha \cdot 1} \uplus r_{\geq \alpha \cdot 2})) > c_u(r_{\geq \alpha \cdot 1})$ we can apply the induction hypothesis and infer that whenever $(r_{\geq \alpha \cdot 1}, \sigma) \Longrightarrow (r'_{\geq \alpha \cdot 1}, \sigma')$ then, for some u_1, $(stat(r_{\geq \alpha \cdot 1}), \sigma) \longrightarrow (u_1, \sigma')$ and $u_1 = stat(r'_{\geq \alpha \cdot 1})$. By Lemma A.2(k), $(stat(r_{\geq \alpha \cdot 1}), \sigma) \longrightarrow (u_1, \sigma')$ implies that we also have $(stat(r_{\geq \alpha \cdot 1}) \parallel stat(r_{\geq \alpha \cdot 2}), \sigma) \longrightarrow (u_1 \parallel stat(r_{\geq \alpha \cdot 2}), \sigma')$. Also, notice that $stat(r'_{\geq \alpha \cdot 1} \uplus r_{\geq \alpha \cdot 2}) = stat(r'_{\geq \alpha \cdot 1}) \parallel stat(r_{\geq \alpha \cdot 2}) = u_1 \parallel stat(r_{\geq \alpha \cdot 2})$. The case when $r' = r_{\geq \alpha \cdot 1} \uplus r'_{\geq \alpha \cdot 2}$ and $(r_{\geq \alpha \cdot 2}, \sigma) \Longrightarrow (r'_{\geq \alpha \cdot 2}, \sigma')$ can be handled similarly. We conclude that, whenever $(r_{\geq \alpha \cdot 1} \uplus r_{\geq \alpha \cdot 2}, \sigma) \Longrightarrow (r', \sigma')$ then, for some u', $(stat(r_{\geq \alpha \cdot 1} \uplus r_{\geq \alpha \cdot 2}), \sigma) \longrightarrow (u', \sigma')$ and $stat(r') = u'$, i.e. $(r', u') \in S$, which implies that (i) holds. □

Theorem 6.4

(a) For all $w \in W, \sigma \in \Sigma$: $\overrightarrow{O}(stat(w), \sigma) = \overrightarrow{\overrightarrow{O}}(w, \sigma)$.

(b) For all $s \in Stat$: $\overrightarrow{\mathcal{O}}[\![s]\!] = \overrightarrow{\overrightarrow{\mathcal{O}}}[\![s]\!]$.

Proof For part (a) let

$$\varepsilon = \sup\{d(\overrightarrow{O}(stat(w), \sigma), \overrightarrow{\overrightarrow{O}}(w, \sigma)) \mid w \in W, \sigma \in \Sigma\}$$

We show that $\varepsilon \leq \frac{1}{2} \cdot \varepsilon$, which means that $\varepsilon = 0$. More precisely, we show that

$$\forall w \in W, \sigma \in \Sigma [d(\overrightarrow{O}(stat(w), \sigma), \overrightarrow{\overrightarrow{O}}(w, \sigma)) \leq \tfrac{1}{2} \cdot \varepsilon]$$

When $w = (\emptyset, \varphi) \in Res$, for some $\varphi \in Id \to Stat$ (which implies $empty(w)$), we have:

$$d(\overrightarrow{O}(stat(w), \sigma), \overrightarrow{\overrightarrow{O}}(w, \sigma)) = d(\overrightarrow{O}(E, \sigma), \overrightarrow{\overrightarrow{O}}((\emptyset, \varphi), \sigma)) = d(\{\epsilon\}, \{\epsilon\}) = 0 \leq \tfrac{1}{2} \cdot \varepsilon$$

Otherwise, we have (recall that T_D is finitely branching, hence below we have a finite union for all $w \in W$, and $stat(w) \in Stat$)

$$\overrightarrow{O}(stat(w), \sigma) = \bigcup_{(stat(w), \sigma) \longrightarrow (u, \sigma')} \sigma \cdot \overrightarrow{O}(u, \sigma')$$

and (T_C is also finitely branching, hence below we also have a finite union)

$$\overrightarrow{\overrightarrow{O}}(w,\sigma) = \bigcup_{(w,\sigma)\Longrightarrow(r,\sigma')} \sigma\cdot \overrightarrow{\overrightarrow{O}}(r,\sigma')$$

Now, by Lemma 6.3 whenever $(w,\sigma)\Longrightarrow(r,\sigma)$ then, for some u, $(stat(w),\sigma)\longrightarrow(u,\sigma')$ and $u = stat(r)$. Also, whenever $(stat(w),\sigma)\longrightarrow(u,\sigma')$ then, for some r, $(w,\sigma)\Longrightarrow(r,\sigma)$ and $u = stat(w)$. Therefore $\{\sigma\cdot \overrightarrow{O}(u,\sigma') \mid (stat(w),\sigma)\longrightarrow(u,\sigma')\} = \{\sigma\cdot \overrightarrow{O}(stat(r),\sigma') \mid (w,\sigma)\Longrightarrow(r,\sigma')\}$, and thus we have:

$$\bigcup_{(stat(w),\sigma)\longrightarrow(u,\sigma')} \sigma\cdot \overrightarrow{O}(u,\sigma') = \bigcup_{(w,\sigma)\Longrightarrow(r,\sigma')} \sigma\cdot \overrightarrow{O}(stat(r),\sigma')$$

Hence, we have:

$$\begin{aligned}
&d(\overrightarrow{O}(stat(w),\sigma), \overrightarrow{\overrightarrow{O}}(w,\sigma))\\
&= d(\bigcup_{(stat(w),\sigma)\longrightarrow(u,\sigma')} \sigma\cdot \overrightarrow{O}(u,\sigma'), \bigcup_{(w,\sigma)\Longrightarrow(r,\sigma')} \sigma\cdot \overrightarrow{\overrightarrow{O}}(r,\sigma'))\\
&= d(\bigcup_{(w,\sigma)\Longrightarrow(r,\sigma')} \sigma\cdot \overrightarrow{O}(stat(r),\sigma'), \bigcup_{(w,\sigma)\Longrightarrow(r,\sigma')} \sigma\cdot \overrightarrow{\overrightarrow{O}}(r,\sigma'))\\
&\text{['}\textstyle\bigcup\text{' is nonexpansive]}\\
&\leq max_{(w,\sigma)\Longrightarrow(r,\sigma')} d(\sigma\cdot \overrightarrow{O}(stat(r),\sigma'), \sigma\cdot \overrightarrow{\overrightarrow{O}}(r,\sigma'))\\
&\leq \tfrac{1}{2}\cdot max_{(w,\sigma)\Longrightarrow(r,\sigma')} d(\overrightarrow{O}(stat(r),\sigma'), \overrightarrow{\overrightarrow{O}}(r,\sigma')) \leq \tfrac{1}{2}\cdot\varepsilon
\end{aligned}$$

We have thus shown that, for all $w \in W, \sigma \in \Sigma : d(\overrightarrow{O}(stat(w),\sigma), \overrightarrow{\overrightarrow{O}}(w,\sigma)) \leq \frac{1}{2}\cdot\varepsilon$, from which $\sup\{d(\overrightarrow{O}(stat(w),\sigma), \overrightarrow{\overrightarrow{O}}(w,\sigma)) \mid w \in W, \sigma \in \Sigma\} = \varepsilon \leq \frac{1}{2}\cdot\varepsilon$, i.e. $\varepsilon = 0$, follows. The implication is that for all $w \in W, \sigma \in \Sigma$: $\overrightarrow{O}(stat(w),\sigma) = \overrightarrow{\overrightarrow{O}}(w,\sigma)$, which proves part (a).

Part (b) follows easily by using part (a). For any $s \in Stat$ we put $w(\in W) = (s,(\epsilon,(\emptyset,\lambda\alpha\,.\,\delta)))$. We have:

$$\begin{aligned}
&\overrightarrow{\overrightarrow{\mathcal{O}}}[\![s]\!]\\
&= \lambda\sigma\,.\,\overrightarrow{\overrightarrow{\mathcal{O}}}[\![s]\!]\sigma \qquad \text{[Def. } \overrightarrow{\overrightarrow{\mathcal{O}}}[\![\cdot]\!]]\\
&= \lambda\sigma\,.\,\overrightarrow{\overrightarrow{O}}(w,\sigma) \qquad \text{[part (a)]}\\
&= \lambda\sigma\,.\,\overrightarrow{O}(stat(w),\sigma)\\
&= \lambda\sigma\,.\,\overrightarrow{O}(s,\sigma) \qquad \text{[Def. } \overrightarrow{\mathcal{O}}[\![\cdot]\!]]\\
&= \lambda\sigma\,.\,\overrightarrow{\mathcal{O}}[\![s]\!]\sigma\\
&= \overrightarrow{\mathcal{O}}[\![s]\!]
\end{aligned}$$

□

Theorem 6.4 implies that all properties, or concurrency laws, that hold in a semantic model designed in direct semantics also hold in a semantic model designed with continuation semantics for concurrency. Corolarry 6.5 is an easy consequence of Theorem 6.4 and well-known properties of the transition system T_D and operational semantics $\overrightarrow{\mathcal{O}}[\![\cdot]\!]$, which are designed in classic direct semantics. For example, the reader may consult the monograph [1] where all properties given in Corolarry 6.5 are proved in direct semantics.

Corollary 6.5 *For all* $s, s_1, s_2, s_3 \in Stat$*:*

(a) $\vec{\mathcal{O}}[\![s_1 + s_2]\!] = \vec{\mathcal{O}}[\![s_2 + s_1]\!]$

(b) $\vec{\mathcal{O}}[\![(s_1 + s_2) + s_3]\!] = \vec{\mathcal{O}}[\![s_1 + (s_2 + s_3)]\!]$

(c) $\vec{\mathcal{O}}[\![s + s]\!] = \vec{\mathcal{O}}[\![s]\!]$

(d) $\vec{\mathcal{O}}[\![(s_1 + s_2); s_3]\!] = \vec{\mathcal{O}}[\![s_1; s_3 + s_2; s_3]\!]$

(e) $\vec{\mathcal{O}}[\![(s_1; s_2); s_3]\!] = \vec{\mathcal{O}}[\![s_1; (s_2; s_3)]\!]$

(f) $\vec{\mathcal{O}}[\![s + \delta]\!] = \vec{\mathcal{O}}[\![s]\!]$

(g) $\vec{\mathcal{O}}[\![\delta; s]\!] = \vec{\mathcal{O}}[\![\delta]\!]$

(h) $\vec{\mathcal{O}}[\![s_1 \| s_2]\!] = \vec{\mathcal{O}}[\![s_1 \between s_2 + s_2 \between s_1]\!]$

(i) $\vec{\mathcal{O}}[\![a \between s]\!] = \vec{\mathcal{O}}[\![a; s]\!]$

(j) $\vec{\mathcal{O}}[\![(a; s_1) \parallel s_2]\!] = \vec{\mathcal{O}}[\![a; (s_2 \| s_1)]\!]$

(k) $\vec{\mathcal{O}}[\![(s_1 + s_2) \between s_3]\!] = \vec{\mathcal{O}}[\![s_1 \between s_3 + s_2 \between s_3]\!]$

7 Conclusion

For a simple abstract concurrent language embodying a general mechanism of asymchronous communication we designed and compared two different operational semantics, both constructed in the style of structural operational semantics [9]. The first operational model is based on a transition system designed in direct-style semantics. The second one is based on a transition system designed with continuation semantics for concurrency [12]. By using bisimulation semantics [6] and some techniques of metric semantics [1] we proved that the two operational semantics assign the same meaning to any program in the asynchronous language under investigation, and thus, any property that holds in direct semantics also holds in continuation semantics.

8 Acknowledgements

The research reported in this paper was partially supported by the Romanian Ministry of Education, Research, Youth and Sports, National Authority for Scientific Research, Capabilities / Module III, Bilateral Collaboration between Romania and Greece, project no. 582/16.07.2012, project title: "SemNat: Semantic Models and Technologies for Natural Computing" (2012-2014).

References

[1] J.W.de Bakker and E.P. de Vink. *Control flow semantics.* MIT Press, 1996.

[2] J.W. De Bakker and J.H.A. Warmerdam, Metric pomset semantics for a concurrent language with recursion, *LNCS* 469:21–49, Springer, 1990.

[3] F. De Boer, J.N. Kok, C. Palamidessi, J.J.M.M. Rutten. A paradigm for asynchronous communication and its application to concurrent constraint programming. In Apt, K.R., De Bakker, J.W. and Rutten, J.J.M.M, eds., *Logic Programming Languages: Constraints, Functions and Objects*, 82–114, MIT Press, 1993.

[4] K. Honda, M. Tokoro. An object calculus for asynchronous communication. *LNCS* 512:133147, Springer, 1991.

[5] R. Milner. A calculus of communicationg systems. *LNCS* 92, Springer, 1980.

[6] R. Milner. *Communicating and mobile systems: the π-calculus.* Cambridge University Press, 1999.

[7] D.M.R. Park. Concurrency and automata on infinite sequences. *LNCS* 104:167–183, Springer, 1981.

[8] C. Palamidessi. Comparing the expressive power of the synchronous and the asynchronous π-calculus. *Math. Structures in Computer Science*, 13(5): 685-719, 2003.

[9] G.D. Plotkin. A structural approach to operational semantics. Report DAIMI FN-19, Aarhus University, 1981.

[10] V. Saraswat. *Concurrent constraint programming.* MIT Press, 1993.

[11] C. Stratchey, and C. Wadsworth. Continuations: a mathematical semantics for handling full jumps. *Journal of Higher-Order and Symbolic Computation*, 13(1):135–152, 2000 (Reprint of the Technical Monograph PRG-11, Oxford University, Computing Laboratory, 1974).

[12] E.N. Todoran. Metric semantics for synchronous and asynchronous communication: a continuation-based approach. *Electronic Notes in Theoretical Computer Science*, 28:119–146, Elsevier, 2000.

[13] E.N. Todoran, and N. Papaspyrou. Continuations for prototyping concurrent languages. Technical Report CSD-SW-TR-1-06, National Technical University of Athens, Software Engineering Laboratory, 2006.

[14] Synchronous and Asynchronous Interaction in Distributed Systems (SAS). Project funded by DFG (German Research Foundation), 2010-2012, `http://concurrency-theory.service.tu-berlin.de/joomla/projects/projects`.

A Appendix

The following Lemmas are needed in the proof of Lemma 6.3. We omit here the proofs, which are not difficult but are laborius. The proofs can proceed by induction on $c_u(u)$ (given in Definition 4.5) and $c_w(w)$ (given in Definition 5.4), respectively.

Lemma A.1

(a) $\simeq_d, \simeq_c$ *are equivalence relations (they are symmetric, reflexive and transitive).*

(b) $(x,\sigma) \simeq_d (D(x),\sigma)$.

(c) $(((\emptyset,\varphi) \mid \alpha \mapsto s),\sigma) \simeq_c ((s,(\alpha,(\emptyset,\varphi))),\sigma)$

(d) $((x,(\alpha,r)),\sigma) \simeq_c ((D(x),(\alpha,r)),\sigma)$

(e) $((s_1;s_2,(\alpha,r)),\sigma) \simeq_c ((s_1,(\alpha\cdot 1,(r \mid \alpha \mapsto s_2))),\sigma)$

(f) $((s_1 \| s_2,(\alpha,(\emptyset,\varphi))),\sigma) \simeq_c (((\emptyset,\varphi) \mid \alpha\cdot 1 \mapsto s_1 \mid \alpha\cdot 2 \mapsto s_2),\sigma)$

Lemma A.2

(a) $(w_{>\alpha},\sigma) \Longrightarrow (r,\sigma') \Rightarrow r = r_{>\alpha}$.

(b) $(w_{\geq\alpha},\sigma) \Longrightarrow (r,\sigma') \Rightarrow r = r_{\geq\alpha}$.

(c) $(w_{>\alpha},\sigma) \Longrightarrow_0 (w',\sigma') \Rightarrow w' = w_{>\alpha}$.

(d) $(w_{\geq\alpha},\sigma) \Longrightarrow_0 (w',\sigma) \Rightarrow w' = w_{\geq\alpha}$.

(e) If $\neg(empty(w_{>\alpha}))$, $\neg(empty(r_{=\alpha}))$ *and* $\sigma \in \Sigma$ *then*

$$(w_{>\alpha} \uplus r_{=\alpha},\sigma) \Longrightarrow (r'_{\geq\alpha},\sigma') \Leftrightarrow [((w_{>\alpha},\sigma) \Longrightarrow (r'_{>\alpha},\sigma')) \wedge (r'_{\geq\alpha} = r'_{>\alpha} \uplus r_{=\alpha})].$$

(f) For all $u \in U, \sigma \in \Sigma$: $(s_1;s_2,\sigma) \longrightarrow (u,\sigma') \Leftrightarrow \exists u_1,\sigma_1[((s_1,\sigma) \longrightarrow (u_1,\sigma_1)) \wedge (u \equiv u_1;s_2)]$.

(g) If $\neg(empty(r_{\geq\alpha\cdot 2}))$ *and* $\beta \geq \alpha\cdot 1$ *then*

$$((s,(\beta,r_{\geq\alpha\cdot 1})) \uplus r_{\geq\alpha\cdot 2},\sigma) \Longrightarrow (r',\sigma') \Leftrightarrow$$
$$[(((s,(\beta,r_{\geq\alpha\cdot 1})),\sigma) \Longrightarrow (r'_{\geq\alpha\cdot 1},\sigma')) \wedge (r' = r'_{\geq\alpha\cdot 1} \uplus r_{\geq\alpha\cdot 2})].$$

(h) If $\neg(empty(r_{\geq\alpha\cdot 1}))$ *and* $\beta \geq \alpha\cdot 2$ *then*

$$((s,(\beta,r_{\geq\alpha\cdot 2})) \uplus r_{\geq\alpha\cdot 1},\sigma) \Longrightarrow (r',\sigma') \Leftrightarrow$$
$$[(((s,(\beta,r_{\geq\alpha\cdot 2})),\sigma) \Longrightarrow (r'_{\geq\alpha\cdot 2},\sigma')) \wedge (r' = r_{\geq\alpha\cdot 1} \uplus r'_{\geq\alpha\cdot 2})].$$

(i) For all $u \in U, \sigma \in \Sigma$: $(s_1 \between s_2,\sigma) \longrightarrow (u,\sigma') \Leftrightarrow \exists u_1,\sigma_1[((s_1,\sigma) \longrightarrow (u_1,\sigma_1)) \wedge (u \equiv u_1 \| s_2)]$.

(j) If $\neg(empty(r_{\geq\alpha\cdot 1}))$, $\neg(empty(r_{\geq\alpha\cdot 2}))$ *and* $\sigma \in \Sigma$ *then*

$$(r_{\geq\alpha\cdot 1} \uplus r_{\geq\alpha\cdot 2},\sigma) \Longrightarrow (r',\sigma') \Leftrightarrow$$
$$[[((r_{\geq\alpha\cdot 1},\sigma) \Longrightarrow (r'_{\geq\alpha\cdot 1},\sigma')) \wedge (r' = r'_{\geq\alpha\cdot 1} \uplus r_{\geq\alpha\cdot 2})] \vee$$
$$[((r_{\geq\alpha\cdot 2},\sigma) \Longrightarrow (r'_{\geq\alpha\cdot 2},\sigma')) \wedge (r' = r_{\geq\alpha\cdot 1} \uplus r'_{\geq\alpha\cdot 2})]]$$

(k) $(s_1 \| s_2,\sigma) \longrightarrow (u,\sigma') \Leftrightarrow$

$$\exists u_1[((s_1,\sigma) \longrightarrow (u_1,\sigma')) \wedge (u \equiv u_1 \| s_2)] \vee$$
$$\exists u_2[((s_2,\sigma) \longrightarrow (u_2,\sigma')) \wedge (u \equiv s_1 \| u_2)]$$