Comparative Metric Semantics for Modern Second Order Communication Abstractions

MIRCEA IVAN AND ENEIA NICOLAE TODORAN

Abstract: We study the semantics of a language $L^2_J$ that provides second order communication and synchronization on multiple channels in the style introduced in Join calculus. We employ the mathematical methodology of metric semantics in designing and relating a denotational and an operational semantics for $L^2_J$. The semantic models are designed with continuations.

Key Words: Metric semantics, continuations for concurrency, second order communication, Join synchronization

Received: October 20, 2010
1 Introduction

In a recent paper [18] we presented a semantic study of an imperative language $L_J$ that extends Hoare’s CSP model [9] with communication on multiple channels and synchronization based on join patterns in the style introduced in Join calculus [8]. In this paper we consider a language $L^2_J$ that extends $L_J$ with second order communication: sending and receiving of statements rather than values. $L^2_J$ generalizes the traditional model of second order communication - studied, e.g., in [2, 3] and also in our previous work [19] - with a mechanism of synchronization inspired by the Join calculus.

$L^2_J$ provides two primitives for concurrent interaction on multiple channels: $c!s$ and $c_1?x_1\&\cdots\&c_n?x_n$; we call the latter a join pattern. A communication in $L^2_J$ is an interaction between $n + 1$ processes, one executing a join pattern $c_1?x_1\&\cdots\&c_n?x_n$, and $n$ other processes executing corresponding (send) actions $c_1!s_1, \cdots, c_n!s_n$. The statements $s_1, \cdots, s_n$ are transmitted concurrently along the channels $c_1, \cdots, c_n$ (each $s_i$ is transmitted along the corresponding channel $c_i$) from the processes that execute the actions $c_1!s_1, \cdots, c_n!s_n$ to the process that holds the join pattern $c_1?x_1\&\cdots\&c_n?x_n$. The latter stores the $n$ received statements in the variables $x_1, \cdots, x_n$ (each $s_i$ is assigned to the corresponding statement variable $x_i$). The communication only occurs when the $n + 1$ processes are all ready for the interaction. The stored statements can be called by $\text{call}(x_i)$ statements. When $n = 1$ the whole interaction behaves like a (classic second order) point-to-point communication.

In this paper we employ the mathematical methodology of metric semantics [4, 1] in designing and relating an operational and a denotational semantics for $L^2_J$. The semantic models are designed with the “continuation semantics for concurrency” (CSC) technique introduced by us in [22]. CSC is a general tool for designing denotational and operational models of concurrency in interleaving semantics. The central characteristic of the CSC technique is the modeling of continuations as structured configurations of computations (denotations) rather than the straightforward functions to some answer type that are used in the classic technique of continuations [16]. The structure of an CSC continuation is representative for the control concepts of the (concurrent) language under study.

The focus of this study is on the communication primitives of $L^2_J$. We show that CSC continuations can easily synchronize multiple communication attempts and allow us to combine in a simple and flexible manner traditional concurrent control concepts with second order communication primitives and advanced synchronization mechanisms. In [17] we presented a denotational semantics for a language similar to $L^2_J$. In this paper we offer both a denotational semantics and an operational semantics for $L^2_J$. Next, we establish the precise mathematical relation between the two semantic models, following the general methodology advocated in [4]. The first paper that uses metric spaces and continuations for concurrency in designing and relating semantic models of second order communication is [19]. [19] studies traditional communication between two processes that synchronize on a single (second-order) communication channel. The present paper offers a generalization of [19] to second-order communication on multiple channels, in the style introduced in Join calculus.

The Join calculus has recently inspired the design of Join Java [10], Polyphonic C# [5] (concurrent extensions of the mainstream languages Java and C#, respectively) and JoCaml [11] (a concurrent extension of the functional programming language ML). The relevance of our work is given by the presence of higher-order features in JoCaml. Also, second order communication provides a simple form of code mobility. Our work may be seen as a first step toward a formal treatment of object mobility in languages with Join methods like Join Java.
and Polyphonic C#.

Contribution

As far as we know this is the first paper that presents a comparative semantics study for (a form of) higher-order communication with synchronization on multiple channels in the style introduced in the Join calculus [8].

The CSC technique seems to simplify the semantic treatment of second order communication. We obtain a relatively simple relation between the operational and the denotational semantics of \( L^2_J \) by using only basic techniques of metric semantics. In [2, 3] a comparative metric semantics study of second order communication is presented. The relation obtained there between the denotational and the operational models is more complex than the one reported by us in section 7 of this paper. Also, in [2, 3] more advanced techniques, like processes as terms [15] and metric labelled transition systems [6] are needed to establish that relation.

Overview

The rest of the paper is organized as follows. The formal syntax of \( L^2_J \) is introduced in section 3. The operational and the denotational semantics are presented in sections 5 and 6 and are related in section 7.

2 Notation and theoretical preliminaries

The notation \( (x \in)X \) introduces the set \( X \) with typical element \( x \) ranging over \( X \). Let \( f \in X \rightarrow Y \) be a function. The function \( (f \mid x \mapsto y) : X \rightarrow Y \), 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 \( (f \mid x_1 \mapsto y_1 | \cdots | x_n \mapsto y_n) \) as an abbreviation for \( (\cdots (f \mid x_1 \mapsto y_1) \cdots | x_n \mapsto y_n) \). If \( f : X \rightarrow X \) and \( f(x) = x \) we call \( x \) a fixed point of \( f \). When this fixed point is unique (see theorem 2.1) we write \( x = \text{fix}(f) \).

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 \( '\cong' \)), complete metric space, and compact set. We recall that if \( (X, d_X), (Y, d_Y) \) are metric spaces, a function \( f : X \rightarrow Y \) 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 nonexpansive functions from \( X \) to \( Y \) by \( X \rightarrow^1 Y \). The following theorem is at the core of metric semantics.

Theorem 2.1 (Banach) Let \( (X, d_X) \) be a complete metric space. Each contracting function \( f : X \rightarrow X \) has a unique fixed point.

Definition 2.2 Let \( (X, d_X), (Y, d_Y) \) be (ultra) metric spaces. On \( (x \in)X, (f \in)X \rightarrow Y \) (the function space), \( ((x, y) \in)X \times Y \) (the cartesian product), \( (u, v) \in)X + Y \) (the disjoint union of \( X \) and \( Y \)) and \( (U, V \in)\mathcal{P}(X) \) (the power set of \( X \)), one can define the following metrics:
(a) \( d_{\frac{1}{2}, X} : X \times X \rightarrow [0, 1], \quad d_{\frac{1}{2}, X}(x_1, x_2) = \frac{1}{2} \cdot d_X(x_1, x_2) \)

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

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

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

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

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

(e) \( d_H : \mathcal{P}(X) \times \mathcal{P}(X) \rightarrow [0, 1], \quad d_H(U, V) = \max\{\sup_{u \in U} d(u, V), \sup_{v \in V} d(u, U)\} \)

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 use the abbreviations \( \mathcal{P}_{\text{nco}}(\cdot) \) and \( \mathcal{P}_{\text{finite}}(\cdot) \) to denote the power sets of 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 \( (X, d_{\frac{1}{2}, X}) \).

**Remark 2.3** Let \((X, d_X), (Y, d_Y), d_{\frac{1}{2}, X}, d_{X \rightarrow Y}, d_{X \times Y}, d_{X + Y} \) and \( d_H \) be as in definition 2.2. In case \( d_X, d_Y \) are ultrametrics, so are \( d_{\frac{1}{2}, X}, d_{X \rightarrow Y}, d_{X \times Y}, d_{X + Y} \) and \( d_H \). If in addition \((X, d_X), (Y, d_Y)\) are complete then \( \frac{1}{2} \cdot X, X \rightarrow Y, X \rightarrow Y, X \times Y, X + Y, \) and \( \mathcal{P}_{\text{nco}}(X) \) (with the metrics defined above) are also complete metric spaces.

## 3 Syntax of \( L^2_J \)

The syntax of \( L^2_J \) is given in BNF in 5.1. The basic components are a set \((v \in)\text{Var}\) of variables, a set \((e \in)\text{Exp}\) of expressions, a set \((c \in)\text{Ch}\) of (second order) communication channels and a set \((x \in)\text{Svar}\) of statement variables.

**Definition 3.1** (Syntax of \( L^2_J \))

(a) **(Join patterns)** \( \langle j \in J \rangle ::= c?x \mid j \& j \)

For an \( L^2_J \) program to be valid the channels \( c_1, \ldots, c_n \) and the statement variables \( x_1, \ldots, x_n \) in a join pattern \( j = (c_1?x_1 & \cdots & c_n?x_n) \) must be pairwise distinct.

(b) **(Statements)** \( s \in \text{Stat} ::= \text{skip} \mid v := e \mid e!s \mid j \mid \text{call}(x) \mid s; s + s \mid s || s \)

The language \( L^2_J \) provides assignment \((v := e)\), recursion, sequential composition \((s; s)\), non-deterministic choice \((s + s)\), parallel composition \((s || s)\) and the communication mechanism (based on the interaction between a join pattern \( c_1?x_1 & \cdots & c_n?x_n \) and \( n \) corresponding send statements \( c_1!s_1, \ldots, c_n!s_n \)) that was explained informally in the introduction. We assume that the meaning of expressions is given by a valuation \( \mathcal{E}[: \text{Exp} \rightarrow \Sigma \rightarrow \text{Val}, \text{where}\ Val \text{ is a set of values and } (\sigma \in)\Sigma = \text{Var} \rightarrow \text{Val} \text{ is a set of states.} \)
4 Continuation structure for $L^2_J$

In the definition of the denotational semantics we use a complete ultrametric space. The operational semantics is defined by means of a transition relation embedded in a deductive system defined in the style of structured operational semantics [14]. Following [4], we use the term *resumption* as an operational counterpart of the term *continuation*. The distinctive characteristic of the CSC technique is the representation of continuations as structured configurations of computations (denotations of statements). Similarly, resumptions are structured configurations of statements. The structure of a CSC continuation (resumption) is representative of the language under study.

As shown in [22] in order to handle the general combination of sequential and parallel composition in $L^2_J$ we need a tree-like structure with active elements at the leaves. The major issue that gives rise to a tree-like structure is the presence of statements such as $(s_1 \parallel s_2); s_3$. In such a statement the execution of $s_3$ can only begin after the completion of the parallel execution of both $s_1$ and $s_2$. The basic idea is that we place $s_3$ as an inner node and $s_1$ and $s_2$ as leaves of such a tree and we give priority to the leaves. Following [22] we define the domain of continuations and the set of resumptions with the aid of an auxiliary set $(\alpha \in Id)$ of identifiers endowed with a partial ordering relation.

**Definition 4.1**

(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 $\text{max}: \mathcal{P}(Id) \rightarrow \mathcal{P}(Id)$ by:

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

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

$Id$ is the set of all finite (possibly empty) sequences over $\{1, 2\}$. $\alpha \leq \alpha'$ iff $\alpha$ is a prefix of $\alpha'$. In this paper we use the symbol '∙' as a concatenation operator over sequences and we use the symbol 'ϵ' to represent the empty sequence. We can represent tree-like structures as suggested below:

```
          α ∙ 1 ∙ 1
         /       \
      α ∙ 1         α ∙ 2 ∙ 1
     /   \        /   \
 α ∙ 2 ∙ 1      α ∙ 2 ∙ 2
```

Let $A = \{\alpha, \alpha ∙ 1, \alpha ∙ 2, \alpha ∙ 1 ∙ 1, \alpha ∙ 1 ∙ 2, \alpha ∙ 2 ∙ 1, \alpha ∙ 2 ∙ 2\}$. The maximal elements of $(A, \leq_A)$ are exactly the leaves of the tree: $\text{max}(A) = \{\alpha ∙ 1 ∙ 1, \alpha ∙ 1 ∙ 2, \alpha ∙ 2 ∙ 1, \alpha ∙ 2 ∙ 2\}$.

Let $(\pi \in \Pi = \mathcal{P}_{finite}(Id)$ and let $(x \in X)$ be a metric domain. We use the following notation:

$$\|X\| \overset{\text{def}}{=} \Pi \times (Id \rightarrow X)$$

Let $\alpha \in Id, (\pi, \varpi) \in \|X\|$ (with $\pi \in \Pi, \varpi \in Id \rightarrow X$). We define $id: \|X\| \rightarrow \Pi, id(\pi, \varpi) = \pi$.

We also use the following abbreviations:
We can obtain a domain by endowing the sets $\text{Id}$ and $\Pi$ with the discrete metric (which is an ultrametric).\footnote{We avoid using the notion of a partially ordered multiset which usually denotes a more elaborated mathematical structure; see, e.g., ch. 16 of [4].} $id(\pi,\varpi)$ returns the collection of identifiers for the computations contained in the bag $(\pi,\varpi)$, $(\pi,\varpi)(\alpha)$ returns the computation with identifier $\alpha$, $(\pi,\varpi) \setminus \pi'$ removes the computations with identifiers in $\pi$, and $((\pi,\varpi) \mid \alpha \mapsto x)$ replaces the computation with identifier $\alpha$.

By a slight abuse we will use the same notations when $(\pi,\varpi)$ replaces the computation with identifier $\alpha$.

Let $(\pi,\varpi) \in \llbracket X \rrbracket$. We treat $(\pi,\varpi)$ as a 'function' with finite graph $\{(\alpha,\varpi(\alpha)) \mid \alpha \in \pi \}$, thus ignoring the behavior of $\varpi$ for any $\alpha \notin \pi$ ($\pi$ is the 'domain' of $(\pi,\varpi)$). We use this mathematical structure to represent finite partially ordered bags (or multisets)\footnote{$d(x,y) = 0$ if $x = y$. If $x \neq y$ then $d(x,y) = 1$.} of computations. The set $\text{Id}$ is used to distinguish between multiple occurrences of a computation in such a bag. We can obtain a domain by endowing the sets $\text{Id}$ and $\Pi$ with the discrete metric (which is an ultrametric).\footnote{We avoid using the notion of a partially ordered multiset which usually denotes a more elaborated mathematical structure; see, e.g., ch. 16 of [4].} $id(\pi,\varpi)$ returns the collection of identifiers for the computations contained in the bag $(\pi,\varpi)$.

We introduce three auxiliary mappings that produce values of the type $\text{Sched} = \mathcal{P}_{\text{finite}}(\text{Id}^+)$, which are finite collections of schedules. A schedule $(\in \text{Id}^+)$ is a finite and nonempty sequence of identifiers. We define the mappings $\text{sched}, \text{scheda}, \text{scheds}$ by $\text{Res} \rightarrow \text{Sched}$ as follows:
We use the convention: 

\[sched(r) = scheda(r) \cup scheds(r)\]

\[scheda(r) = \{(\alpha) | \alpha \in \max(id(r)), r(\alpha) \in Stat\}\]

\[scheds(r) = \{(\alpha, \alpha_1, \ldots, \alpha_n) | \alpha, \alpha_1, \ldots, \alpha_n \in \max(id(r)), r(\alpha) \in J^\emptyset, r(\alpha_1), \ldots, r(\alpha_n) \in Ch \times Stat, r(\alpha) = (c_1?x_1 & \cdots & c_n?x_n), r(\alpha_1) = c_1!s_1, \ldots, r(\alpha_n) = c_n!s_n\}\]

\(scheda\) produces schedules of length 1, that are used to activate individual computations (statements). \(scheds\) produces schedules of length > 1 that are used to model the pattern matching synchronization that is characteristic for languages based on the Join calculus [8]. Obviously, \(scheda(r) \cap scheds(r) = \emptyset\), for all \(r \in Res\).

The operational semantics of \(L_2^2\) is based on a transition relation \(\rightarrow \subseteq Conf' \times (Res \times \Theta \times \Sigma)\), with elements \((t, t')\) written in the notation \(t \rightarrow t'\). It is easy to check, by using the rules of \(T_2^2\) (definition 5.2), that if \(t \in Conf'\) and \(t \rightarrow t'\) then \(t' \in (Res \times \Theta \times \Sigma)\), first for all \(t = (s, r^\alpha, \theta, \sigma) \in (Stat \times Res^\alpha \times \Theta \times \Sigma)\) by structural induction on \(s\), and next for all configuration of the form \((r, \theta, \sigma) \in (Res \times \Theta \times \Sigma)\), by using the fact that \(id(r)\) is finite.

We use the convention:

\[t_1 \not\rightarrow t_2 \quad \text{is an abbreviation for:} \quad \frac{t_2 \rightarrow t'}{t_1 \not\rightarrow t'}\]

**Definition 5.2** *(Transition system for \(L_2^2\): \(T_2^2\))* The transition relation for \(L_2^2\) is the smallest subset of \(Conf' \times (Res \times \Theta \times \Sigma)\) satisfying the rules below. In \((A2)\) \(\sigma = (\sigma | v \mapsto E[v](\sigma))\).

\((A1)\) \(\text{Skip}, r^\alpha, \theta, \sigma) \rightarrow (r, \theta, \sigma)\)

\((A2)\) \(v := e, r^\alpha, \theta, \sigma) \rightarrow (r, \theta, \sigma)\)

\((A3)\) \(c!s, r^\alpha, \theta, \sigma) \rightarrow ((r | \alpha \mapsto c!s), \theta, \sigma)\)

\((A4)\) \(j, r^\alpha, \theta, \sigma) \rightarrow ((r | \alpha \mapsto \langle j \rangle), \theta, \sigma)\)

\((A5)\) \(\text{Call}(x), r^\alpha, \theta, \sigma) \rightarrow (\theta(x), r^\alpha, \theta, \sigma)\)

\((R6)\) \((s_1 + s_2, r^\alpha, \theta, \sigma) \not\rightarrow (s_1, r^\alpha, \theta, \sigma)\)

\((R7)\) \((s_1 + s_2, r^\alpha, \theta, \sigma) \not\rightarrow (s_2, r^\alpha, \theta, \sigma)\)

\((R8)\) \((s_1; s_2, r^\alpha, \theta, \sigma) \not\rightarrow (s_1, (r | \alpha \mapsto s_2)^{\alpha-1}, \theta, \sigma)\)

\((R9)\) \((s_1 || s_2, r^\alpha, \theta, \sigma) \not\rightarrow (s_1, (r | \alpha \cdot 2 \mapsto s_2)^{\alpha-1}, \theta, \sigma)\)

\((R10)\) \((s_1 || s_2, r^\alpha, \theta, \sigma) \not\rightarrow (s_2, (r | \alpha \cdot 1 \mapsto s_1)^{\alpha-2}, \theta, \sigma)\)

\((A11)\) \((r, \theta, \sigma) \rightarrow (r \setminus \{\alpha, \alpha_1, \ldots, \alpha_n\}, \theta, \sigma) \quad \forall (\alpha, \alpha_1, \ldots, \alpha_n) \in scheds(r)\)

where \(r(\alpha) = (c_1?x_1 \& \cdots \& c_n?x_n), r(\alpha_1) = c_1!s_1, \ldots, r(\alpha_n) = c_n!s_n, \bar{\delta} = (\theta | x_1 \mapsto s_1 | \cdots | x_n \mapsto s_n)\)
The CSC technique is a semantic formalization of a process scheduler [22]. In a configuration of the form \((s, r^α, θ, σ)\) the computation (statement) \(s\) is active. The other computations are contained in the resumption \(r\) and wait for their turn to be activated (scheduled for evaluation). The identifier \(α\) points to the conceptual position of the active computation \(s\) with respect to the computations contained in \(r\). Each computation remains active only until it performs an elementary step (axioms A1-A5). Subsequently, another computation taken from the resumption is activated (rule (R12)). In this way it can be obtained the desired interleaving behavior for parallel composition.

The axioms (A3) and (A4) create communication attempts that are combined by using rule (A11). An \(L^2\) communication is an interaction in which several statements are transmitted concurrently on different channels. In (A11) the process that executes the join pattern \(c_1 x_1 & \cdots & c_n x_n\) receives the statements \(s_1, \ldots, s_n\) and assigns them to corresponding statement variables \(x_1, \ldots, x_n\). Rule (A5) specifies that the execution of procedure \(x\) in the current syntactic store \(θ\) amounts to the execution of \(θ(x)\). Rules (A11) and (R12) give priority to the computations at the leaves of the tree that represents the resumption. In the case of a sequential composition \(s_1; s_2\) (rule (R8)) the computations \(s_1\) and \(s_2\) are given the identifiers \(α\cdot 1\) and \(α\), respectively. As \(α\cdot 1 > α\), the evaluation of \(s_2\) will begin only after the completion of the evaluation of \(s_1\). Rules (R9) and (R10) define the semantics of a parallel composition \(s_1 ∥ s_2\). The computations \(s_1\) and \(s_2\) are given the identifiers \(α\cdot 1\) and \(α\cdot 2\), respectively; \(α\cdot 1\) and \(α\cdot 2\) are incomparable (with respect to \(≤\)) therefore the statements \(s_1\) and \(s_2\) are evaluated in an interleaved manner.

**Definition 5.3** (Normal termination and deadlock) We define \(\text{terminates}, \text{blocks} : \text{Res} → \text{Bool}:

\[
\text{terminates}(r) = (\text{id}(r) = ∅)
\]

\[
\text{blocks}(r) = (\text{id}(r) \neq ∅) ∧ (\text{sched}(r) = ∅)
\]

Let \(t(∈ \text{Conf'})\). We say that \(t\) terminates if \(t = (r, θ, σ) ∈ (\text{Res} × Θ × Σ)\) and \(\text{terminates}(r)\). Also, we say that \(t\) blocks if \(t = (r, θ, σ) ∈ (\text{Res} × Θ × Σ)\) and \(\text{blocks}(r)\).

Let \(t ∈ \text{Conf'}\). We write \(t →\) to express the fact that \(t\) has transitions, i.e. \(∃t' : t → t'\). Also, we write \(t \nrightarrow\) to express the fact that \(t\) has no transitions, i.e. \(¬(∃t' : t → t')\).

**Lemma 5.4**

(a) \(t \nrightarrow ⇔ t \text{ terminates or } t \text{ blocks}\)

(b) If \(t = (s, r^α, θ, σ) ∈ (\text{Stat} × \text{Res}^α × Θ × Σ)\) then \(t →\).

Lemma 5.4 follows easily from the rules of \(T^2\) (5.4(b) follows by structural induction on \(s\)).

**Definition 5.5** Let \((x ∈ X)\) be a nonempty complete space. The space \(X^\infty\) is defined by the domain equation \(X^\infty \cong \{ε\} + \{δ\} + (X × \frac{1}{2} \cdot X^\infty)\). \(ε\) models the empty sequence. \(δ\) is used to model deadlock. The elements of \(X^\infty\) are finite sequences (possibly followed by \(δ\) or infinite sequences over \(X\). We usually write \(x_1 x_2 \cdots x_n\) and \(x_1 x_2 \cdots x_n δ\) instead of \((x_1, (x_2, \cdots (x_n, ε) \cdots ))\) and \((x_1, (x_2, \cdots (x_n, δ) \cdots ))\). Also, instead of \((x_1, (x_2, \cdots ))\) we write \(x_1 x_2 \cdots\). If \(X\) is endowed with the discrete metric we obtain a Baire-like distance on \(X^\infty\) [4].
The first 3 steps correspond to applications of rules (A3) and (A4) of \( \omega, \sigma \) t
ational semantics each trace is a sequence of pairs \((\theta, \sigma) \in \Theta \times \Sigma\). We endow the space \( \Theta \times \Sigma \) with the discrete metric and define: \( \mathbf{P}_O = \mathcal{P}_{nco}(\Theta \times \Sigma) \).

(b) Let \((S \in )\mathcal{S}_O = \text{Conf}' \to \mathbf{P}_O \) and let \( \Psi : \mathcal{S}_O \to \mathcal{S}_O \) be given by:

\[
\Psi(S)(t) =
\begin{cases}
\{ \epsilon \} & \text{if } t \text{ terminates} \\
\{ \theta \} & \text{if } t \text{ blocks} \\
\cup \{(\theta, \sigma) \cdot S(r, \theta, \sigma) \mid t \to (r, \theta, \sigma)\} & \text{otherwise}
\end{cases}
\]

(c) Let \( r_0(\in \text{Res}) = (\emptyset, \lambda \alpha. \text{skip} \) ). We put \( O = \text{fix}(\Psi) \) and define \( O[] : \text{Stat} \to (\Theta \times \Sigma) \to \mathbf{P}_O \):

\[
O[s](\theta, \sigma) = O(s, r_0^\alpha, \theta, \sigma)
\]

It is easy to check that \((s, r_0^\alpha, \theta, \sigma)\) is a derivable configuration, and if \( t \) is derivable and \( t \to t' \) or \( t \not\to t' \) then \( t' \) is also derivable. Also, one can check that \( T_J^2 \) is finitely branching (for all \( t \in \text{Conf}' \) the set \( \{ t' \mid t \to t' \} \) is finite). The proof can proceed in two steps: first for all the derivable configurations of the form \((s, r^\alpha, \theta, \sigma)\) by structural induction on \( s \) and next for all configurations of the form \((r, \theta, \sigma)\) by using the fact that the set \( \text{id}(r) \) is finite. This implies that \( T_J^2 \) induces a compact operational semantics \([4]\). The mapping \( \Psi \) is a contraction (and thus it has a unique fixed point) due to the "\((\theta, \sigma)"\)-step in its definition.

Example 5.7 Let \( s_1 = (c_1!(v := 1)) \parallel (c_2! \text{skip}) \) and \( s_2 = (c_1?x_1 & c_2?x_2); (\text{call}(x_1) \parallel \text{call}(x_2)). \) Let also \( \sigma \in \Sigma, \theta \in \Theta \) and \( \overline{\sigma} = (\sigma \mid v \mapsto 1), \overline{\theta} = (\theta \mid x_1 \mapsto (v := 1)) \). One can check that:

\[
O[s_1 \parallel s_2](\theta, \sigma) = ((\theta, \sigma)(\theta, \sigma)(\overline{\theta}, \sigma)(\overline{\theta}, \sigma)(\overline{\theta}, \sigma)(\overline{\theta}, \sigma)(\overline{\theta}, \sigma)(\overline{\theta}, \sigma)(\overline{\theta}, \sigma),
(\theta, \sigma)(\theta, \sigma)(\theta, \sigma)(\overline{\theta}, \sigma)(\overline{\theta}, \sigma)(\overline{\theta}, \sigma)(\overline{\theta}, \sigma)(\overline{\theta}, \sigma)(\overline{\theta}, \sigma),
(\theta, \sigma)(\theta, \sigma)(\theta, \sigma)(\overline{\theta}, \sigma)(\overline{\theta}, \sigma)(\overline{\theta}, \sigma)(\overline{\theta}, \sigma)(\overline{\theta}, \sigma)(\overline{\theta}, \sigma))
\]

The first 3 steps correspond to applications of rules (A3) and (A4) of \( T_J^2 \). The 4th step results from an application of (A11). The 5th step is a silent step produced by using (A5). Depending on the moment when the assignment \( v := 1 \) is executed we obtain 3 different execution traces.

6 Denotational semantics (D)

In this section we employ the CSC technique in designing a continuation-based denotational (compositional) semantics for \( L_J^3 \). The final yield of the denotational semantics is also an element of a linear-time domain \( \mathbf{P}_D \) (a collection of execution traces). In the case of denotational semantics each trace is a sequence of pairs \((\omega, \sigma)\), where \( \omega(\epsilon) \) is a semantic store. The type of the denotational semantics \( D \) for \( L_J^3 \) is \( \text{Sem}_D = \text{Stat} \to \mathbf{D} \):

\[
(\phi \in )D \cong (\text{Cont}^\alpha \times \Omega) \to \Sigma \to \mathbf{P}_D
\]
A continuation is a configuration of computations \((\in \mathbf{Comp})\). In [21] the elements of \(\mathbf{Cont}\) are called \textit{closed continuations}, and the elements of \(\mathbf{Cont}^a\) are called \textit{open continuations}. Both open and closed continuations are representations of the \textit{rest of the computation} [16]. An open continuation is also an \textit{evaluation context} [7, 21] for the active computation (denotation). In an expression \(D(s)(\gamma^a,\omega)(\sigma)\) the identifier \(\alpha\) points to the conceptual position of the active computation \(D(s)\) with respect to the other computations contained in the continuation \(\gamma\). We denote typical elements \((\gamma,\alpha)\) of \(\mathbf{Cont}^a\) by \(\gamma^a\). Also, for easier readability we denote typical elements \((c,\phi)\) of \(Ch \times D\) by \(c\phi\). In the equations above the sets \(Id\) (and \(\Pi\)), \(\Sigma, J^\circ\) and \(Ch\) are endowed with the discrete metric, which is an ultrametric. The above system of domain equations has a unique solution (up to isometry) [1]. The solutions for \(D\) and the other domains are obtained as complete ultrametric spaces.

We define the mappings \(Sched, Scheda, Scheds : \mathbf{Cont} \rightarrow Sched\) that compute schedules and the predicates \(Terminates, Blocks : \mathbf{Cont} \rightarrow \mathbb{Bool}\) that we need to detect normal termination and deadlock. Obviously, \(Scheda(\gamma) \cap Scheds(\gamma) = \emptyset\), for all \(\gamma \in \mathbf{Cont}\).

\[
\text{Sched}(\gamma) = Scheda(\gamma) \cup Scheds(\gamma)
\]

\[
\text{Scheda}(\gamma) = \{(\alpha) \mid \alpha \in \max(id(\gamma)), \gamma(\alpha) \in D\}
\]

\[
\text{Scheds}(\gamma) = \{(\alpha, \alpha_1, \cdots, \alpha_n) \mid \alpha, \alpha_1, \cdots, \alpha_n \in \max(id(\gamma)), \\
\gamma(\alpha) \in J^\circ, \gamma(\alpha_1), \cdots, \gamma(\alpha_n) \in Ch \times D, \\
\gamma(\alpha) = (c_1?x_1&\cdots&c_n?x_n), \gamma(\alpha_1) = c_1?\phi_1, \cdots, \gamma(\alpha_n) = c_n?\phi_n\}
\]

\[
\text{Terminates}(\gamma) = (id(\gamma) = \emptyset)
\]

\[
\text{Blocks}(\gamma) = (id(\gamma) \neq \emptyset) \land (Sched(\gamma) = \emptyset)
\]

**Definition 6.1** (Denotational semantics \((D)\) for \(L^2\))

\(a\) We define \(C : (\mathbf{Cont} \times \Omega) \rightarrow \Sigma \rightarrow \mathcal{P}_D\) by:

\[
C(\gamma,\omega)(\sigma) =
\]

\[
\begin{align*}
\text{if} & \quad \text{Terminates}(\gamma) \text{ then } \{\epsilon\} \\
\text{else} & \quad \text{if} \quad \text{Blocks}(\gamma) \text{ then } \{\delta\} \\
\text{else} & \quad (\bigcup\{\gamma(\alpha)((\gamma \setminus \{\alpha\})^a,\omega)(\sigma) \mid (\alpha) \in Scheda(\gamma)\}) \quad \bigcup \\
& \quad (\bigcup\{\overline{\omega}, \sigma \cdot C(\gamma \setminus \{\alpha, \alpha_1, \cdots, \alpha_n\},\overline{\omega})(\sigma) \\
& \quad \quad \text{ where } \gamma(\alpha) = (c_1?x_1&\cdots&c_n?x_n), \gamma(\alpha_1) = c_1?\phi_1, \cdots, \gamma(\alpha_n) = c_n?\phi_n, \\
& \quad \quad \quad \overline{\omega} = (\omega \mid x_1 \mapsto \phi_1 \mid \cdots \mid x_n \mapsto \phi_n)\}
\end{align*}
\]
structural induction on \( s \)

**Proof.**

**Lemma 6.2**

Definition 6.1 is justified by lemma 6.2.

\[
\begin{align*}
\Phi & : \text{Sem}_D \rightarrow \text{Sem}_D \text{ for } (S \in \text{Sem}_D) \text{ by:} \\
\Phi(S)(\text{skip})(\gamma^\alpha, \omega)(\sigma) &= (\omega, \sigma) \cdot C(\gamma, \omega)(\sigma) \\
\Phi(S)(v := e)(\gamma^\alpha, \omega)(\sigma) &= (\omega, \sigma) \cdot C(\gamma, \omega)(\sigma) \quad \text{where } \sigma = (\sigma | v \mapsto \mathcal{E}[e](\sigma)) \\
\Phi(S)(c!(s))(\gamma^\alpha, \omega)(\sigma) &= (\omega, \sigma) \cdot C((\gamma | \alpha \mapsto c!S(s)), \omega)(\sigma) \\
\Phi(S)(\text{call}(x))(\gamma^\alpha, \omega)(\sigma) &= (\omega, \sigma) \cdot C((\gamma | \alpha \mapsto \langle x \rangle), \omega)(\sigma) \\
\Phi(S)(s_1 + s_2)(\gamma^\alpha, \omega)(\sigma) &= \Phi(S)(s_1)(\gamma^\alpha, \omega)(\sigma) \cup \Phi(S)(s_2)(\gamma^\alpha, \omega)(\sigma) \\
\Phi(S)(s_1; s_2)(\gamma^\alpha, \omega)(\sigma) &= \Phi(S)(s_1)((\gamma | \alpha \mapsto S_s)_{\alpha}, \omega)(\sigma) \\
\Phi(S)(s_1 || s_2)(\gamma^\alpha, \omega)(\sigma) &= \Phi(S)(s_1)((\gamma | \alpha \cdot 2 \mapsto S_s)_{\alpha}, \omega)(\sigma) \cup \Phi(S)(s_2)((\gamma | \alpha \cdot 1 \mapsto S_s)_{\alpha}, \omega)(\sigma)
\end{align*}
\]

We put \( \mathcal{D} = \text{fix}(\Phi) \). Let \( \gamma_0 = (\emptyset, \lambda \alpha. \mathcal{D}(\text{skip})) \). We define \( \mathcal{D}[\cdot] : \text{Stat} \rightarrow (\Omega \times \Sigma) \rightarrow \text{P}_D \) by:

\[
\mathcal{D}[s](\omega, \sigma) = \mathcal{D}(s)(\gamma_0', \omega)(\sigma)
\]

Definition 6.1 is justified by lemma 6.2.

**Lemma 6.2**

(a) The mapping \( C \) (introduced in 6.1) is well-defined.

(b) For all \( \gamma_1, \gamma_2 \in \text{Cont}, \omega_1, \omega_2 \in \Omega : d(C(\gamma_1, \omega_1), C(\gamma_2, \omega_2)) \leq 2 \cdot d((\gamma_1, \omega_1), (\gamma_2, \omega_2)). \)

Also, for all \( S \in \text{Sem}_D, s \in \text{Stat}, \gamma^\alpha \in \text{Cont}^\alpha, \omega \in \Omega, \sigma \in \Sigma:

(c) \( \Phi(S)(s)(\gamma^\alpha, \omega)(\sigma) \in \text{P}_D \) (it is well defined),

(d) \( \Phi(S)(s) \) is nonexpansive (in \( \gamma^\alpha, \omega \)), and

(e) \( \Phi \) is \( \frac{1}{2} \)-contractive (in \( S \)).

**Proof.** Similar lemmas are given in [18, 22]. Here we only prove 6.2(e). We proceed by structural induction on \( s \). We only consider the subcase \( s \equiv s_1 || s_2 \).

\[
\begin{align*}
&\quad d(\Phi(S_1)(s_1 || s_2)(\gamma^\alpha, \omega)(\sigma), \Phi(S_2)(s_1 || s_2)(\gamma^\alpha, \omega)(\sigma)) \\
&= d(\Phi(S_1)(s_1)((\gamma | \alpha \cdot 2 \mapsto S_s)_{\alpha}^1, \omega)(\sigma) \cup \Phi(S_1)(s_2)((\gamma | \alpha \cdot 1 \mapsto S_s)_{\alpha}^2, \omega)(\sigma), \\
&\quad \Phi(S_2)(s_1)((\gamma | \alpha \cdot 2 \mapsto S_s)_{\alpha}^1, \omega)(\sigma) \cup \Phi(S_2)(s_2)((\gamma | \alpha \cdot 1 \mapsto S_s)_{\alpha}^2, \omega)(\sigma)) \\
&\leq [\cup \text{ is nonexpansive } ] \\
&\quad \max\{d(\Phi(S_1)(s_1)((\gamma | \alpha \cdot 2 \mapsto S_s)_{\alpha}^1, \omega)(\sigma), \\
&\quad \Phi(S_2)(s_1)(\gamma)(\gamma | \alpha \cdot 2 \mapsto S_s)_{\alpha}^1, \omega)(\sigma))(\\star) \}
\end{align*}
\]
\[\begin{align*}
&d(\Phi(S_1)(s_2)|(\gamma | \alpha \cdot 1 \mapsto S_1(s_1))^{\alpha^2}, \omega)(\sigma), \\
&\Phi(S_2)(s_2)|(\gamma | \alpha \cdot 1 \mapsto S_2(s_1))^{\alpha^2}, \omega)(\sigma)\}^{**}
\end{align*}\]

We only treat \((*)\); \((***)\) can be handled similarly.

\((*) \leq [\ d \text{ is an ultrametric }]
\]

\[
\max\{d(\Phi(S_1)(s_1)|(\gamma | \alpha \cdot 2 \mapsto S_1(s_2))^{\alpha^1}, \omega)(\sigma), \Phi(S_1)(s_1)|(\gamma | \alpha \cdot 2 \mapsto S_2(s_2))^{\alpha^1}, \omega)(\sigma)\}
\]

\[
\leq [6.2(d), \text{ind. hyp. }]
\]

\[
\max\{d((\gamma | \alpha \cdot 2 \mapsto S_1(s_2)), (\gamma | \alpha \cdot 2 \mapsto S_2(s_2))), \frac{1}{2} \cdot d(S_1, S_2)\}
\]

\[
= \max\{\frac{1}{2} \cdot d(S_1(s_2), S_2(s_2)), \frac{1}{2} \cdot d(S_1, S_2)\} \leq \frac{1}{2} \cdot d(S_1, S_2)
\]

\[\blacksquare\]

7 Relating \(\mathcal{O}\) and \(\mathcal{D}\)

We prove that \(\forall s \in \text{Stat}, \theta \in \Theta, \sigma \in \Sigma : SEM(\mathcal{O}[s]((\theta, \sigma)) = \mathcal{D}[s](\text{sem}(\theta), \sigma), \) where:

Definition 7.1

(a) \(\text{sem} : \Theta \rightarrow \Omega \text{ is defined by: } \text{sem}(\theta) = \lambda x.\mathcal{D}(\theta(x)).\)

(b) We let \(w \) range over \((w \in)(\Theta \times \Sigma)^{\infty}\) and define \(\text{Sem} : (\Theta \times \Sigma)^{\infty} \rightarrow (\Omega \times \Sigma)^{\infty}\) as the unique mapping satisfying: \(\text{Sem}(\epsilon) = \epsilon, \text{Sem}(\delta) = \delta\) and \(\text{Sem}((\theta, \sigma) \cdot w) = (\text{sem}(\theta), \sigma) \cdot \text{Sem}(w).\)

(c) Recall that \(\mathcal{P}_\mathcal{O} = \mathcal{P}_nco((\Theta \times \Sigma)^{\infty})\) and \(\mathcal{P}_\mathcal{D} = \mathcal{P}_nco((\Omega \times \Sigma)^{\infty}).\) We let \(W \) range over \((W \in)\mathcal{P}_\mathcal{O}\) and define \(\text{SEM} : \mathcal{P}_\mathcal{O} \rightarrow \mathcal{P}_\mathcal{D}\) by: \(\text{SEM}(W) = \{\text{Sem}(w) | w \in W\} \).

Definition 7.2 Let \(\varsigma : \text{Comp} \rightarrow \text{Comp}, \varsigma((j)) = (j), \varsigma(c!s) = c!\mathcal{D}(s)\) and \(\varsigma(s) = \mathcal{D}(s).\) We define \(\Gamma : \text{Res} \rightarrow \text{Cont}\) by \(\Gamma(r) = (id(r), \lambda \alpha. \varsigma(r(\alpha))).\) By using the abbreviation introduced in section 4 we can write \(\Gamma(r)(\alpha) = \varsigma(r(\alpha)).\) Also, we define \(\mathcal{R} : \text{Conf}' \rightarrow \mathcal{P}_\mathcal{D} :\)

\[
\mathcal{R}(r, \theta, \sigma) = C(\Gamma(r), \text{sem}(\theta))(\sigma)
\]

\[
\mathcal{R}(s, r^\alpha, \theta, \sigma) = \mathcal{D}(s)(\Gamma(r)^\alpha, \text{sem}(\theta))(\sigma)
\]

As \(\mathcal{D}[s](\text{sem}(\theta), \sigma) = \mathcal{D}(s)(\gamma^\alpha, \text{sem}(\theta))(\sigma) = \mathcal{D}(s)(\Gamma(r_0)^\alpha, \text{sem}(\theta))(\sigma) = \mathcal{R}(s, r_0^\alpha, \theta, \sigma),\) and \(\mathcal{O}[s](\theta, \sigma) = \mathcal{O}(s, r_0^\alpha, \theta, \sigma),\) in order to prove that \(\text{SEM}(\mathcal{O}[s]((\theta, \sigma)) = \mathcal{D}[s](\text{sem}(\theta), \sigma)\) it suffices to show that \(\text{SEM} \circ \mathcal{O} = \mathcal{R}.\) In 7.4 and 7.6 we prove that both \(\text{SEM} \circ \mathcal{O}\) and \(\mathcal{R}\) are fixed points of the same (higher-order) contraction \(\Phi_R,\) defined in 7.3.
Definition 7.3  Let \((S \in) \text{Sem}_R = \text{Conf}' \to \text{PD}\). We define \(\Phi_R : \text{Sem}_R \to \text{Sem}_R\) by:
\[
\Phi_R(S)(t) = \begin{cases} 
\{\epsilon\} & \text{if } t \text{ terminates} \\
\{\delta\} & \text{if } t \text{ blocks} \\
\cup \{(\text{sem}(\theta), \sigma) \cdot S(r, \theta, \sigma) \mid t \to (r, \theta, \sigma)\} & \text{otherwise}
\end{cases}
\]

The transitions \(t \to (r, \theta, \sigma)\) are with respect to \(T^2_J\) (5.2).

Lemma 7.4  \(\text{SEM} \circ \mathcal{O} = \text{fix}(\Phi_R)\)

Proof. We show that for all \(t \in \text{Conf}' : (\text{SEM} \circ \mathcal{O})(t) = \Phi_R(\text{SEM} \circ \mathcal{O})(t)\). If \(t\) terminates \(\Phi_R(\text{SEM} \circ \mathcal{O})(t) = \{\epsilon\} = (\text{SEM} \circ \mathcal{O})(t)\). Also, if \(t\) blocks \(\Phi_R(\text{SEM} \circ \mathcal{O})(t) = \{\delta\} = (\text{SEM} \circ \mathcal{O})(t)\). It is easy to check that \(\text{SEM}((\theta, \sigma) \cdot \mathcal{W}) = (\text{sem}(\theta), \sigma) \cdot \text{SEM}(\mathcal{W})\) and \(\text{SEM}(\mathcal{W}_1 \cup \mathcal{W}_2) = \text{SEM}(\mathcal{W}_1) \cup \text{SEM}(\mathcal{W}_2)\). So, if \(t\) has indeed transitions we have:
\[
\Phi_R(\text{SEM} \circ \mathcal{O})(t) = \cup \{(\text{sem}(\theta), \sigma) \cdot (\text{SEM} \circ \mathcal{O})(r, \theta, \sigma) \mid t \to (r, \theta, \sigma)\}
\]
\[
= \cup \{(\text{SEM}((\theta, \sigma) \cdot \mathcal{O}(r, \theta, \sigma)) \mid t \to (r, \theta, \sigma)\} \quad [T^2_J \text{ is finitely branching}]
\]
\[
= \text{SEM}(\cup \{(\theta, \sigma) \cdot \mathcal{O}(r, \theta, \sigma) \mid t \to (r, \theta, \sigma)\})
\]
\[
= \text{SEM}(\mathcal{O}(t)) = (\text{SEM} \circ \mathcal{O})(t)
\]

Lemma 7.5 collects some properties that are used in the proof of 7.6. The proof of 7.5 is simple enough and left to the reader.

Lemma 7.5

(a) If \(t\) terminates then \(\mathcal{R}(t) = \{\epsilon\}\).
(b) Also, if \(t\) blocks then \(\mathcal{R}(t) = \{\delta\}\).
(c) For all \(r \in \text{Res} : \text{scheda}(r) = \text{Scheda}(\Gamma(r))\).
(d) For all \(r \in \text{Res} : \text{scheds}(r) = \text{Scheds}(\Gamma(r))\).
(e) For all \(r \in \text{Res}, \pi \in \Pi : \Gamma(r \setminus \pi) = \Gamma(r) \setminus \pi\)
(f) For all \(r \in \text{Res}, \alpha \in \text{Id}, \rho \in \text{Comp} : (\Gamma(r) \mid \alpha \mapsto \rho) = (\Gamma(r) \mid \alpha \mapsto \varsigma(\rho))\)
(g) For all \(\theta \in \Theta, x_1, \cdots, x_n \in S\text{var}, s_1, \cdots, s_n \in \text{Stat}:
\[
\text{sem}(\theta \mid x_1 \mapsto s_1 \mid \cdots \mid x_n \mapsto s_n) = (\text{sem}(\theta) \mid x_1 \mapsto D(s_1) \mid \cdots \mid x_n \mapsto D(s_n))
\]

Lemma 7.6  \(\mathcal{R} = \text{fix}(\Phi_R)\)

The proof of lemma 7.6 is relegated to the appendix. The main result of the paper is obtained in theorem 7.7. The proof of 7.7 combines 7.4, 7.6 and Banach’s theorem 2.1.
Theorem 7.7 $SEM(O[s](\theta, \sigma)) = D[s](sem(\theta), \sigma), \forall s \in Stat, \theta \in \Theta, \sigma \in \Sigma.$

Proof.

$SEM(O[s](\theta, \sigma)) = (SEM \circ O)(s, r_0^e, \theta, \sigma)$ [7.4,7.6,2.1]

$= R(s, r_0^e, \theta, \sigma) = D(s)(\Gamma(r_0^e), sem(\theta))(\sigma)$

$= D[s](sem(\theta), \sigma)$

8 Concluding remarks and future research

In this paper we extended the semantic models given in [18] with second order communication: sending and receiving of statements rather than simple data values. As far as we know this is the first paper that reports a denotational semantics for (a form of) higher-order communication combined with synchronization on multiple channels in the style introduced in the Join calculus [8]. The semantic models given in this paper and in [18] are designed with CSC continuations, a technique introduced by us in previous work [22]. In a denotational model designed with the CSC technique all (concurrent) control concepts are represented as operations manipulating continuations. The use of CSC continuations proved to be fruitful. We obtained a relatively simple relation between the denotational semantics and the operational semantics by using only basic techniques of metric semantics [4].

In [20] we presented a semantic interpreter designed with the CSC technique for a concurrent object-oriented language with Join methods, inspired by Join Java [10] and Polyphonic C# [5], which in turn are based on the Join calculus [8]. The communication of statements provides a simple form of code mobility. The research reported in this paper shows that CSC continuations can provide a framework for combining code mobility with other fundamental concurrent control concepts in a uniform manner. In the near future we intend to extend the object-oriented system with Join methods introduced in [20] with primitives for object mobility.

Second order communication is a particular form of higher order interaction. Another objective of future research is the study of $\omega$-order communication (parameterized processes of arbitrary high order) in the presence of Join synchronization. In recent unpublished work we developed in Haskell [13] a semantic interpreter that implements a dynamic denotational semantics designed with CSC continuations for a language with $\omega$-order communication and classic point-to-point communication. It should be possible to extend that interpreter with Join synchronization by using the technique presented in this paper. The complete mathematical formalization is again subject of future research that would represent an important step toward a formal denotational semantics for JoCaml [11].

9 Acknowledgements

This paper is based on work partially funded by ANCS, CNMP-PC project no. 11-052/2007, project title: “GlobalComp: Models, semantics logics and technologies for global computing” (2007-2010).
References


A Appendix

Proof. Lemma 7.6 We show that $\forall t \in Conf': \Phi_R(\mathcal{R})(t) = \mathcal{R}(t)$. This follows immediately from lemma 7.5(a) and 7.5(b) if $t$ terminates or $t$ blocks (i.e. if $t \not\rightarrow$). We handle the case when $t$ has indeed transitions in two steps. First, we prove the desired property for all $t$ of the form $t = (s, r^\alpha, \theta, \sigma)(\in Stat \times Res^\alpha \times \Theta \times \Sigma)$ by structural induction on $s$.

- Case $s \equiv j$.

  $\Phi_R(\mathcal{R})(j, r^\alpha, \theta, \sigma) = (sem(\theta), \sigma) \cdot \mathcal{R}((r | \alpha \rightarrow (j)), \theta, \sigma)$

  $= (sem(\theta), \sigma) \cdot C(\Gamma(r | \alpha \rightarrow (j)), sem(\theta))(\sigma)$ \hspace{1cm} [7.5(f)]

  $= D(j)(\Gamma(r)^\alpha, sem(\theta))(\sigma) = \mathcal{R}(j, r^\alpha, \theta, \sigma)$

- Case $s \equiv s_1 || s_2$.

  $\Phi_R(\mathcal{R})(s_1 || s_2, r^\alpha, \theta, \sigma)$ \hspace{1cm} [def. $T^2$]

  $= (\bigcup \{(sem(\theta'), \sigma') \cdot \mathcal{R}(r', \theta', \sigma') \mid (s_1, (r | \alpha \cdot 2 \rightarrow s_2)^{\alpha-1}, \theta, \sigma) \rightarrow (r', \theta', \sigma')\}) \cup$

  $\bigcup \{(sem(\theta'), \sigma') \cdot \mathcal{R}(r', \theta', \sigma') \mid (s_2, (r | \alpha \cdot 1 \rightarrow s_1)^{\alpha-2}, \theta, \sigma) \rightarrow (r', \theta', \sigma')\})$

  $= \Phi_R(\mathcal{R})(s_1, (r | \alpha \cdot 2 \rightarrow s_2)^{\alpha-1}, \theta, \sigma) \cup \Phi_R(\mathcal{R})(s_2, (r | \alpha \cdot 1 \rightarrow s_1)^{\alpha-2}, \theta, \sigma)$ \hspace{1cm} [ind. hyp.]

  $= \mathcal{R}(s_1, (r | \alpha \cdot 2 \rightarrow s_2)^{\alpha-1}, \theta, \sigma) \cup \mathcal{R}(s_2, (r | \alpha \cdot 1 \rightarrow s_1)^{\alpha-2}, \theta, \sigma)$ \hspace{1cm} [def. $\mathcal{R}$, 7.5(f)]
By using 7.5(d) and 7.5(g), for

\[ D(s_1) = D(s_1) \cup \{ \alpha \cdot 2 \mapsto D(s_2) \} \]

we have:

\[ D(s_1) \cup D(s_2) = D(s_1) \cup D(s_2) \]

In the sequel we obtain:

\[ D(s_1) \cup D(s_2) = D(s_1) \cup D(s_2) \]

In the second step we prove that \( \Phi(\mathcal{R}(r, \theta, \sigma)) = \mathcal{R}(r, \theta, \sigma) \) when \((r, \theta, \sigma) \rightarrow\). By 5.6 and 5.2 (rules (A11) and (R12)):

\[
\Phi(\mathcal{R}(r, \theta, \sigma)) = (\bigcup \{(\text{sem}(\theta'), \theta') \in \mathcal{R}(r') | (a) \in \text{scheda}(r), (r(a), (r(a) \setminus \{a\}) \mapsto (r', \theta', \sigma'))\}) \cup \]

Where \( r(a) = \langle c_1 ? x_1 \& \cdots \& c_n ? x_n \rangle, r(a_1) = c_1 ! s_1, \cdots, r(a_n) = c_n ! s_n, \)

\[ \bar{\theta} = (\theta | x_1 \mapsto s_1 | \cdots | x_n \mapsto s_n) \]

In the sequel we obtain:

\[(\ast) = \bigcup \{(\Phi(\mathcal{R}(r, \theta, \sigma)), (r \setminus \{a\}) \mapsto (r(a), (r(a) \setminus \{a\}) \mapsto (r', \theta', \sigma')) \} \in \text{scheda}(r) \} \quad \text{[first step of the proof]}\]

\[ = \bigcup \{\Phi(\mathcal{R}(r, \theta, \sigma)) | (a) \in \text{scheda}(r) \} \quad \text{[7.5(c), 7.5(e)]}\]

\[ = \bigcup \{(\Gamma(r)(a), (\Gamma(r) \setminus \{a\}) \mapsto (\text{sem}(\theta))(\sigma) | (a) \in \text{Scheda}(\Gamma(r)) \} \]

By using 7.5(d) and 7.5(g), for \( (\ast) \) we have:

\[(\ast\ast) = \bigcup \{(\bar{\omega}, \sigma) \cdot C(\Gamma(r) \setminus \{a, a_1, \cdots, a_n\}, \bar{\omega})(\sigma) \}

Where \( \Gamma(r)(a) = \langle c_1 ? x_1 \& \cdots \& c_n ? x_n \rangle, \)

\[ \Gamma(r)(a_1) = c_1 ! \phi_1, \cdots, \Gamma(r)(a_n) = c_n ! \phi_n, \]

\[ \bar{\omega} = (\text{sem}(\theta) | x_1 \mapsto \phi_1 | \cdots | x_n \mapsto \phi_n) \]

\[ | (a, a_1, \cdots, a_n) \in \text{Scheda}(r) \}

In \( (\ast\ast) \) \( \phi_i = D(s_i), 1 \leq i \leq n \). We obtain:

\[ \Phi(\mathcal{R}(r, \theta, \sigma)) = (\ast) \cup (\ast\ast) \]

\[ = C(\Gamma(r), \text{sem}(\theta))(\sigma) \]

\[ = \mathcal{R}(r, \theta, \sigma) \]

\[ \blacksquare \]