A CPS-semantics for a typed lambda-calculus of exception handling with fixed-point

Catherine Pilière

Chapter 1

A CPS-semantics for a typed λ-calculus of exception handling with fix-point

Catherine Pilière

Abstract. We propose to add a fixed-point combinator to a λ-calculus of exceptions handling whose type system corresponds to classical logic through the Curry-Howard isomorphism. To this end, we here give a CPS-semantics to the calculus and show that for non-exceptional terms, this semantics possesses the property of computational adequacy.

1 Introduction

In [de Groote,1995] Philippe de Groote proposed a computational interpretation of classical logic through a simply typed λ-calculus which features an exceptions handling mechanism inspired by the ML language.

This calculus has several interesting properties among others: strong normalisation, confluence and subject reduction, the greater part of them being given by its typing system which ensures that every raised exception is eventually handled whenever the whole term is correctly typed.

Our goal consists in seeing if this interpretation can provide a realistic system of exceptions handling, with emphasis on the study of its behaviour in the presence of a general fixed-point combinator.

As such an operator implies the loss of strong-normalisation which guaranteed most of the properties of the calculus, it is first necessary to give a mathematical model (*i.e.*, a denotational semantics) to the calculus in order to retain the indispensable framework of reasoning.

Nevertheless, as we are interested in exceptions handling, it appears clearly that the direct denotational semantics is inadequate because it does not provide a way of expressing what is happening when an exception is raised and handled. So it is necessary to use a "continuation passing style semantics" ("CPS semantics " for short), because it is the only way of taking account of context changes which could appear during the evaluation of the terms.[Reynolds,1974]

On the other hand, known direct denotational semantics (where we focus on the expressions of the λ-calculus which do not contain exceptional subterms), possess the property of "computational adequacy" [Winskel,1993] [Gunter,1992], which allows us to reason about the operational behaviour of the terms of the language.

By bringing to the fore a relation between the two semantics, from which we know that a term denotes the undefined element by the direct semantics *if and only if* it denotes the undefined element by the continuation semantics, we establish that the continuation semantics possesses this property too.

We first present the λ_{exn}-calculus with fixed-point and its operational semantics. Part two describes its CPS-interpretation, part three being concerned with the direct semantics of the terms of the calculus which do not contain exceptional values (λ_v) and the property of computational adequacy. The last part is devoted to the equivalence between these two models for λ_v.

2 The λ_{exn}-calculus with fixpoint

Definition 1 *The types of the calculus are given by the following grammar:*

$$\mathcal{T} \ ::= \ \iota \mid exn \mid \mathcal{T} \rightarrow \mathcal{T}$$

where ι and exn stand for the base types, exn being the type of exceptional values.

λ_{exn} features an exception handling mechanism by means of exception variables y which act as datatype constructors: these exceptional variables are of functional type, say $\tau \rightarrow exn$ and then, when applied to a term of type τ, return exceptions. An exception acts like all the terms of base type but may also be raised under the form of packets, which are then propagated and possibly handled.

The packet $(\mathcal{R}aiseM)$ will be represented by the term $(\mathcal{R}M)$, the exception declaration $let\ exception\ y\ : \alpha \rightarrow exn\ in\ M\ handle\ (yx) \Rightarrow N end$ being represented by the term $\langle y \cdot M | x \cdot N \rangle$.

Multiple declarations such as $\langle y_1 \cdot \langle y_2 \cdot \ldots \langle y_n \cdot M | x \cdot N_n \rangle \ldots \rangle | x \cdot N_1 \rangle$ will be abbreviated by $\langle \overrightarrow{y} \cdot M | x \cdot \overrightarrow{N} \rangle$.

The term $\mu f \cdot \lambda x \cdot M$ represents the recursive function solution of the equation $f \ = \ \lambda x \cdot M(f, x)$.

As our interest lies in the termination property, we will consider the set of constants as reduced to the single element $*$. Nevertheless, the proofs would be applicable with a more realistic set.

Definition 2 *The syntax of the expressions of the calculus is the following:*

$$E \ ::= \ * \mid x \mid y \mid \lambda x \cdot E \mid (EE) \mid (\mathcal{R}E) \mid \langle y \cdot E | x \cdot E \rangle \mid \mu f \cdot \lambda x \cdot E.$$

The set $FV(T)$ of free variables of a term T is defined as usual. In particular, the free occurences of y in M and x in N are bound in $\langle y \cdot M | x \cdot N \rangle$ and similarly, the free occurences of f in $\lambda x \cdot M$ are bound in $\mu f \cdot \lambda x \cdot M$.

Definition 3 *Define a typing environment to be a function that assigns a type to every variable. Let* Γ *stand for such an environment. The expressions of the language are typed in the following manner:*

$$\Gamma \ \vdash \ * : \iota$$

$$\Gamma \ \vdash \ x : \Gamma(x)$$

$$\frac{\Gamma, x : \alpha \ \vdash \ M : \beta}{\Gamma \ \vdash \ \lambda x \cdot M : \alpha \to \beta}$$

$$\frac{\Gamma \ \vdash \ M : \alpha \to \beta \quad \Gamma \ \vdash \ N : \alpha}{\Gamma \ \vdash \ MN : \beta}$$

$$\frac{\Gamma \ \vdash \ M : exn}{\Gamma \ \vdash \ (\mathcal{R}M) : \alpha}$$

$$\frac{\Gamma, y : \alpha \to exn \ \vdash \ M : \beta \quad \Gamma, x : \alpha \ \vdash \ N : \beta}{\Gamma \ \vdash \ \langle y \cdot M | x \cdot N \rangle : \beta}$$

$$\frac{\Gamma, x : \alpha, f : \alpha \to \beta \ \vdash \ M : \beta}{\Gamma \ \vdash \ \mu f \cdot \lambda x \cdot M : \alpha \to \beta}.$$

If exn is seen as the absurdity type $false$, then the type system above, provided that we forget the rule for fixed-point, corresponds to classical logic through the Curry-Howard isomorphism. Now, a natural question arises: what is the meaning of the last rule ? Another way this rule can be expressed is the following:

$$\frac{\Gamma, f : \alpha \to \beta \ \vdash \ \lambda x \cdot M : \alpha \to \beta}{\Gamma \ \vdash \ \mu f \cdot \lambda x \cdot M : \alpha \to \beta},$$

which appears clearly as a very unexpected logical paradox: $(A \Rightarrow A) \ \vdash A$ ...

However, we can also see it as a rough approximation for the Noetherian induction:

$$\frac{\forall k \ ([\forall i < k \ A(i)] \Rightarrow \ A(k))}{\forall k \ A(k)},$$

where "$<$" denotes a *well-founded order.*

Our interest lies in a call-by-value calculus, this means a calculus where β-reduction is performed only if the argument belongs to a particular set which will be called "set of values" and noted by *Val*.

Definition 4 *The set Val of values is defined as follows:*

$$Val \ ::= \ * \mid x \mid y \mid \lambda x \cdot M \mid \ (yVal).$$

In the following, V (with possible subscript) will stand for a member of Val.

Val	$: V \rightarrow_v V \quad \forall V \in Val$
$\beta_{\mathbf{v}}$	$: (\lambda x \cdot M)V \rightarrow_v M[V/x]$
$\mathbf{Raise_{left}}$	$: V_1(\mathcal{R}\,V) \rightarrow_v (\mathcal{R}\,V)$
$\mathbf{Raise_{right}}$	$: (\mathcal{R}\,V)M \rightarrow_v (\mathcal{R}\,V)$
$\mathbf{Raise_{idem}}$	$: (\mathcal{R}(\mathcal{R}\,V)) \rightarrow_v (\mathcal{R}\,V)$
$\mathbf{Handle_{simp}}$	$: \langle y \cdot M \mid x \cdot N \rangle \rightarrow_v M \quad if \ y \notin FV(M)$
$\mathbf{Handle_{raise}}$	$: \langle \overrightarrow{y} \cdot (\mathcal{R} y_i \, V) \mid x \cdot \overrightarrow{N} \rangle \rightarrow_v \langle \overrightarrow{y} \cdot N_i[V/x] \mid x \cdot \overrightarrow{N} \rangle$
$\mathbf{Handle_{left}}$	$: V\langle y \cdot M \mid x \cdot N \rangle \rightarrow_v \langle y \cdot V\,M \mid x \cdot V\,N \rangle$
$\mathbf{Handle_{right}}$	$: \langle y \cdot M \mid x \cdot N \rangle O \rightarrow_v \langle y \cdot MO \mid x \cdot NO \rangle$
$\mathbf{Raise_{handle}}$	$: (\mathcal{R}\langle y \cdot M \mid x \cdot N \rangle) \rightarrow_v \langle y \cdot (\mathcal{R}M) \mid x \cdot (\mathcal{R}N) \rangle$
Fix	$: \mu f \cdot \lambda x \cdot M \rightarrow_v \lambda x \cdot M[\mu f \cdot \lambda x \cdot M/f]$

Table 1.1: *Evaluation rules of λ_{exn}*

We will use the notations $\stackrel{\star}{\rightarrow}_v$ for the reflexive and transitive closure of $\rightarrow_v$ and $M\uparrow_v$ to express the fact that the evaluation process loops for the term M.

3 CPS-semantics

From a functional point of view, programs can be represented by closed λ-terms of ground type. Because λ_{exn} deals with exceptions handling and recursion, the evaluation of a program may give a value, in an ideal situation, or end with a raised but uncaught exception, which is in fact not too bad as it is an observational result too, or else the evaluation can never stop, this last case being of course very unexpected.

At the start of the process, the information needed for evaluation is entirely contained in the term, as it does not depend of any free variable. Later

on, things become more complicated: the handled objects, which are subterms of initial ones, are not always closed terms, but maybe some closures (*i.e.*, terms bound to environments) and their evaluations possibly interact. As long as this interaction stays purely applicative, it is possible to mirror the operational behaviour of the terms by way of a direct denotational semantics, even for call-by-value [Reynolds,1974]. But here, exception handling makes this no more adequate because the propagation of packets (*i.e.*, raised exceptions) is done to the detriment of applicative contexts.

In order to model this situation, let us denote the set of final outputs of programs by O as a reference to the observational character of its members. We may distinguish two ways terms would be given an interpretation. A term can be considered without taking account of the applicative context it belongs to, but only for its value in a given environment. Such a value is said to be *explicit*; typically, a closure is an explicit value.

However, when we consider the evaluation process, it appears clearly that, at every moment, the final output of the program is not only determined by the explicit value of the term being evaluated, but by the current applicative context too. In a sense, a term is then potentially dependent on an evaluation context. This observation leads to a second level of term interpretation which consists in giving them an *implicit* value. Intuitively, an *implicit* value is a function whose argument is an applicative context (*i.e.*, a continuation) to be applied to an *explicit* value.

Definition 5 *Let O be a CPO of observational values; let I be the CPO* $\{\bot, *\}$ *ordered by* $\bot \sqsubseteq *$*; let* $A \to B$ *stand for the continuous function space between the CPO's A and B. The* explicit *types interpretation is given by:*

$$\begin{aligned} [\![\iota]\!]^-_{\mathrm{cps}} &= \mathrm{I} \\ [\![false]\!]^-_{\mathrm{cps}} &= \mathrm{O} \\ [\![\sigma \to \tau]\!]^-_{\mathrm{cps}} &= ([\![\sigma]\!]^-_{\mathrm{cps}} \to [\![\tau]\!]^+_{\mathrm{cps}}) \end{aligned}$$

the implicit *interpretation of a type* τ *being given by:*

$$[\![\tau]\!]^+_{\mathrm{cps}} = (([\![\tau]\!]^-_{\mathrm{cps}} \to \mathrm{O}) \to \mathrm{O}).$$

Let c stand for a continuation and ρ stand for an environment function that assigns to non exceptional variables $x : \tau$ an explicit value $\rho(x) \in [\![\tau]\!]^-_{\mathrm{cps}}$ and let η stand for an environment function that assigns to exceptional variables $y : \tau \to exn$ an explicit value $\eta(y) \in [\![\tau \to false]\!]^-_{\mathrm{cps}}$. We can now give the CPS-semantics for λ_{exn} by the following structural induction:

$$[\![*]\!]_{\mathrm{cps}}\, \rho\, \eta\, c = c\,(*),$$

$$[\![x]\!]_{\mathrm{cps}}\, \rho\, \eta\, c = c\,(\rho\,(x)),$$

$[\![y]\!]_{\mathrm{cps}}\,\rho\,\eta\,c = c\,(\eta\,(y))$,

$[\![\lambda x \cdot M]\!]_{\mathrm{cps}}\,\rho\,\eta\,c = c\,(\lambda\!\!\lambda d.[\![M]\!]_{\mathrm{cps}}\,\rho[d/x]\,\eta)$,

$[\![MN]\!]_{\mathrm{cps}}\,\rho\,\eta\,c = [\![M]\!]_{\mathrm{cps}}\,\rho\,\eta\,(\lambda\!\!\lambda m.[\![N]\!]_{\mathrm{cps}}\,\rho\,\eta\,(\lambda\!\!\lambda n.m\,(n)\,(c)))$,

$[\![(\mathcal{R}M)]\!]_{\mathrm{cps}}\,\rho\,\eta\,c = [\![M]\!]_{\mathrm{cps}}\,\rho\,\eta\,(\lambda\!\!\lambda x.x)$,

$[\![\langle y \cdot M | x \cdot N\rangle]\!]_{\mathrm{cps}}\,\rho\,\eta\,c = [\![M]\!]_{\mathrm{cps}}\,\rho\,\eta[(\lambda\!\!\lambda d.\lambda\!\!\lambda k.k([\![N]\!]_{\mathrm{cps}}\,\rho[d/x]\,\eta\,c))/y]\,c$,

$[\![\lambda\!\!\lambda f \cdot \lambda x \cdot M]\!]_{\mathrm{cps}}\,\rho\,\eta\,c = c\,(\bigsqcup_{i\in\omega} F_i)$, where:

$$\begin{cases} F_0 & = \bot, \\ F_{i+1} & = \lambda\!\!\lambda d.[\![M]\!]_{\mathrm{cps}}\,\rho[d/x, F_i/f]\,\eta. \end{cases}$$

4 Direct semantics

The non-exceptional terms of the calculus can be interpreted in the frame of lifted CPO's and the strict continuous functions. (For more details, see [Winskel,1993] or [Paulson,1987].)

Definition 6 *Let A be a CPO, with the order $\sqsubseteq_A$. The* lifting *of A consists in adding a new element, say $\bot$, to a copy of A and in defining for this new set denoted A' a new partial order $\sqsubseteq_{A'}$ in the following manner:*

$$\forall x, y \in A' x \sqsubseteq_{A'} y \iff (x = \bot) \text{ or } (x, y \in A \text{ and } x \sqsubseteq_A y).$$

Definition 7 *Let f be a continuous function from a CPO A to a CPO B. Let $\bot_A$ and $\bot_B$ be respectively the smallest elements of A and B. The function f is said to be* strict *if and only if*

$$\forall x \in A,\ x = \bot_A \Longrightarrow f(x) = \bot_B.$$

We will denote by "$A_\bot$" the CPO result of the lifting operation applied to the CPO A and by "$A \circ\!\!\to B$" the strict continuous function space from the CPO A to the CPO B.

Under the previous conventions, we can now describe how type expressions are interpreted.

Definition 8 *Let I stand for the two elements CPO $\{\bot, *\}$ ordered by $\bot \sqsubseteq *$.*

$$\begin{aligned} [\![\iota]\!]_{\mathrm{dir}} &= \mathrm{I} \\ [\![\sigma \to \tau]\!]_{\mathrm{dir}} &= ([\![\sigma]\!]_{\mathrm{dir}} \circ\!\!\to [\![\tau]\!]_{\mathrm{dir}})_\bot \end{aligned}$$

We use strict functions in order to model call-by-value evaluation: if the argument loops (*i.e.*, if it denotes $\perp$), then the result of the application of the function to this argument loops too and thus its denotation must be undefined too.

$$[\![*]\!]_{\mathrm{dir}}\,\rho = *,$$

$$[\![x]\!]_{\mathrm{dir}}\,\rho = \rho(x),$$

$$[\![\lambda x \cdot M]\!]_{\mathrm{dir}}\,\rho = up\,(strict\,(\lambda\!\!\lambda d.[\![M]\!]_{\mathrm{dir}}\,\rho[d/x])),$$

$$[\![MN]\!]_{\mathrm{dir}}\,\rho = down\,([\![M]\!]_{\mathrm{dir}}\,\rho)\,([\![N]\!]_{\mathrm{dir}}\,\rho),$$

$$[\![\mu f \cdot \lambda x \cdot M]\!]_{\mathrm{dir}}\,\rho = up\,(strict\,(\bigsqcup_{i\in\omega} F_i)), \text{ where:}$$

$$\begin{cases} F_0 & = \perp, \\ F_{i+1} & = \lambda\!\!\lambda d.[\![M]\!]_{\mathrm{dir}}\,\rho[d/x, up(strict(F_i))/f]. \end{cases}$$

Such a mathematical interpretation of a call-by-value λ-calculus has been shown to agree with given evaluation rules. Two different proofs can be found in [Winskel,1993] and [Gunter,1992]. For λ_v, the agreement of denotational and operational semantics is expressed by:

Theorem 1 *[Winskel,1993][ch.11, p.190-200]* $M \xrightarrow{\star}_v * \iff [\![M]\!]_{\mathrm{dir}} = *$

It follows that, provided it does not deal with exceptions, a closed term of ground type, that is, a program, loops *if and only if* it is denoted by $\perp$:

Corollary 1 *Let* $M : \iota$ *be such that* $FV(M) = \emptyset$.
Then: $M \uparrow_v \iff [\![M]\!]_{\mathrm{dir}} = \perp$.

Proof. Let M be a closed term whose evaluation loops. Then, it is not true that $M \xrightarrow{\star}_v *$. It follows from theorem 1 that $[\![M]\!]_{\mathrm{dir}} \neq *$, that is: $[\![M]\!]_{\mathrm{dir}} = \perp$.
Suppose now that $[\![M]\!]_{\mathrm{dir}} = \perp$. By theorem 1 we cannot have $M \xrightarrow{\star}_v *$, hence $M \uparrow_v$ □.

5 Equivalence between the two semantics

In order to establish the computational adequacy of the CPS-semantics we propose for λ_{exn}, we define an equivalence notion between direct semantics and CPS-semantics by adapting Reynolds [Reynolds,1974] and then prove that for non-exceptional terms, the two models yield equivalent results with regard to termination.

Definition 9 *Let $\alpha \in \mathcal{T}$, $d \in [\![\alpha]\!]_{\mathrm{dir}}$, and $d' \in [\![\alpha]\!]^{+}_{\mathrm{cps}}$. The relation $d \sim d'$ is defined as follows:*

1. $\alpha = \iota$. $d \sim d'$ *if and only if* $(d = \bot_{[\![\alpha]\!]_{\mathrm{dir}}}$ *and* $d' = \bot_{[\![\alpha]\!]^{+}_{\mathrm{cps}}})$, *or else* $(d = *$ *and* $d' = \mathbb{k}c.c\,(*))$.

2. $\alpha = \sigma \to \tau$. $d \sim d'$ *if and only if* $(d = \bot_{[\![\alpha]\!]_{\mathrm{dir}}}$ *and* $d' = \bot_{[\![\alpha]\!]^{+}_{\mathrm{cps}}})$, *or else there exists* $(f \in ([\![\sigma]\!]_{\mathrm{dir}} \to [\![\tau]\!]_{\mathrm{dir}})$ *and* $f' \in [\![\sigma \to \tau]\!]^{-}_{\mathrm{cps}})$ *such that:* $\begin{cases} d = up\,(strict\,(f)) \\ d' = \mathbb{k}c.c\,(f') \\ \forall e \in [\![\sigma]\!]_{\mathrm{dir}},\ e' \in [\![\sigma]\!]^{-}_{\mathrm{cps}}, e \sim \mathbb{k}c.c\,(e') \text{ implies } f\,(e) \sim f'\,(e'). \end{cases}$

Lemma 1 *Let $\alpha \in \mathcal{T}$. Then, under the following assumptions:*

1. $\forall i\ a_i \in [\![\alpha]\!]_{\mathrm{dir}},\ b_i \in [\![\alpha]\!]^{+}_{\mathrm{cps}}$
2. $\forall i, j\ i < j \Longrightarrow a_i \sqsubseteq a_j$ *and* $b_i \sqsubseteq b_j$,
3. $\forall i\ a_i \sim b_i$

we have: $\bigsqcup_i (a_i) \sim \bigsqcup_i (b_i)$

Proof. (See Appendix A)

Proposition 1 *Let ρ, ρ' be such that $\forall x\ \rho(x) \sim \mathbb{k}c \cdot c(\rho'(x))$. Then, we have for all $e \in E$: $[\![e]\!]_{\mathrm{dir}}\,\rho \sim [\![e]\!]_{\mathrm{cps}}\,\rho'$.*

Proof. (By structural induction on the terms)

- $\boldsymbol{e \equiv *}$: One has: $[\![*]\!]_{\mathrm{dir}}\,\rho \stackrel{def}{=} *$; $[\![*]\!]_{\mathrm{cps}}\,\rho' \stackrel{def}{=} \mathbb{k}c \cdot c(*)$. By definition of $\sim$, one gets trivially: $[\![*]\!]_{\mathrm{dir}}\,\rho \sim [\![*]\!]_{\mathrm{cps}}\,\rho'$.
- $\boldsymbol{e \equiv x}$: Obvious from the semantics.
- $\boldsymbol{e \equiv \lambda x \cdot M}$: We have: $\begin{cases} [\![\lambda x \cdot M]\!]_{\mathrm{dir}}\rho = up(strict(\mathbb{k}d \cdot [\![M]\!]_{\mathrm{dir}}\,\rho[d/x])) \\ [\![\lambda x \cdot M]\!]_{\mathrm{cps}}\rho' = \mathbb{k}c \cdot c(\mathbb{k}d \cdot [\![M]\!]_{\mathrm{cps}}\,\rho'[d/x]). \end{cases}$

  Let e and e' be such that $e \sim \mathbb{k}c \cdot c(e')$. Then, by induction hypothesis on M, we get:

$$(\mathbb{k}d \cdot [\![M]\!]_{\mathrm{dir}}\,\rho[d/x])(e) = [\![M]\!]_{\mathrm{dir}}\,\rho[e/x] \sim (\mathbb{k}d \cdot [\![M]\!]_{\mathrm{cps}}\,\rho'[d/x])(e') = [\![M]\!]_{\mathrm{cps}}\,\rho'[e'/x]$$

  Thus, by definition of $\sim$, we finally obtain:

$$\forall \rho, \rho' :\ \{\forall x :\ \rho(x) \sim \mathbb{k}c \cdot c(\rho'(x))\} \Longrightarrow \{[\![\lambda x \cdot M]\!]_{\mathrm{dir}}\rho \sim [\![\lambda x \cdot M]\!]_{\mathrm{cps}}\rho'\}.$$

$e \equiv MN$: From the semantics, we know:

$$\begin{cases} [\![MN]\!]_{\mathrm{dir}}\rho = down([\![M]\!]_{\mathrm{dir}}\rho)([\![N]\!]_{\mathrm{dir}}\rho) \\ [\![MN]\!]_{\mathrm{cps}}\rho' = \lambda\!\!\lambda c \cdot [\![M]\!]_{\mathrm{cps}}\rho'(\lambda\!\!\lambda m \cdot [\![N]\!]_{\mathrm{cps}}\rho'(\lambda\!\!\lambda n \cdot mnc)). \end{cases}$$

Suppose: $\forall x:\ \rho(x) \sim \lambda\!\!\lambda c \cdot c(\rho'(x))$.

Then, by induction hypothesis: $\begin{cases} [\![M]\!]_{\mathrm{dir}}\rho \sim [\![M]\!]_{\mathrm{cps}}\rho' \\ [\![N]\!]_{\mathrm{dir}}\rho \sim [\![N]\!]_{\mathrm{cps}}\rho', \end{cases}$

with $\begin{cases} [\![M]\!]_{\mathrm{dir}}\rho \in [\![\alpha \to \beta]\!]_{\mathrm{dir}},\ [\![M]\!]_{\mathrm{cps}}\rho' \in [\![\alpha \to \beta]\!]^+_{\mathrm{cps}} \\ [\![N]\!]_{\mathrm{dir}}\rho \in [\![\alpha]\!]_{\mathrm{dir}},\ [\![N]\!]_{\mathrm{cps}}\rho' \in [\![\alpha]\!]^+_{\mathrm{cps}} \end{cases}$

first case: $[\![M]\!]_{\mathrm{dir}}\rho = \bot_{[\![\alpha\to\beta]\!]_{\mathrm{dir}}},\ [\![M]\!]_{\mathrm{cps}}\rho' = \bot_{[\![\alpha\to\beta]\!]^+_{\mathrm{cps}}}$.

Then $[\![MN]\!]_{\mathrm{dir}}\rho = \bot_{[\![\beta]\!]_{\mathrm{dir}}}$ and $[\![MN]\!]_{\mathrm{cps}}\rho' = \lambda\!\!\lambda c \cdot \bot_O$

$= \bot_{([\![\beta]\!]^-_{\mathrm{cps}}\to O)\to O} = \bot_{[\![\beta]\!]^+_{\mathrm{cps}}}$ are related by definition of $\sim$.

second case: $[\![M]\!]_{\mathrm{dir}}\rho \neq \bot_{[\![\alpha\to\beta]\!]_{\mathrm{dir}}},\ [\![M]\!]_{\mathrm{cps}}\rho' \neq \bot_{[\![\alpha\to\beta]\!]^+_{\mathrm{cps}}}$.

By the way, we know that: $\begin{cases} [\![M]\!]_{\mathrm{dir}}\rho = up(strict(f_M)) \\ [\![M]\!]_{\mathrm{cps}}\rho' = \lambda\!\!\lambda c \cdot c(f'_M), \end{cases}$

where $\begin{cases} f_M \in [\![\alpha]\!]_{\mathrm{dir}} \to [\![\beta]\!]_{\mathrm{dir}} \\ f'_M \in [\![\alpha \to \beta]\!]^-_{\mathrm{cps}} = ([\![\alpha]\!]^-_{\mathrm{cps}} \to [\![\beta]\!]^+_{\mathrm{cps}}) \end{cases}$

and $\forall e \in [\![\alpha]\!]_{\mathrm{dir}}, e' \in [\![\alpha]\!]^-_{\mathrm{cps}}:\ e \sim \lambda\!\!\lambda c \cdot c(e') \Rightarrow f_M(e) \sim f'_M(e')$.

Hence we have: $\begin{cases} [\![MN]\!]_{\mathrm{dir}}\rho = strict(f_M)([\![N]\!]_{\mathrm{dir}}\rho) \\ [\![MN]\!]_{\mathrm{cps}}\rho' = \lambda\!\!\lambda c \cdot [\![N]\!]_{\mathrm{cps}}\rho'(\lambda\!\!\lambda n \cdot f'_M(n)c). \end{cases}$

Two cases are now possible:

1. $[\![N]\!]_{\mathrm{dir}}\rho = \bot_{[\![\alpha]\!]_{\mathrm{dir}}},\ [\![N]\!]_{\mathrm{cps}}\rho' = \bot_{[\![\alpha]\!]^+_{\mathrm{cps}}}$.
   Then $[\![MN]\!]_{\mathrm{dir}}\rho = \bot_{[\![\beta]\!]_{\mathrm{dir}}}$ and $[\![MN]\!]_{\mathrm{cps}}\rho' = \bot_{[\![\beta]\!]^+_{\mathrm{cps}}}$.
   Hence, $[\![MN]\!]_{\mathrm{dir}}\rho \sim [\![MN]\!]_{\mathrm{cps}}\rho'$.
2. $\exists n \in [\![\alpha]\!]_{\mathrm{dir}}$ such that $[\![N]\!]_{\mathrm{dir}}\rho = n$.
   Then, by definition of $\sim$ and induction hypothesis, we know that $\exists n' \in [\![\alpha]\!]^-_{\mathrm{cps}}$ such that $[\![N]\!]_{\mathrm{cps}}\rho' = \lambda\!\!\lambda c \cdot c(n')$ and we have $n \sim \lambda\!\!\lambda c \cdot c(n')$.
   Thus we obtain: $\begin{cases} [\![MN]\!]_{\mathrm{dir}}\rho = f_M(n) \\ [\![MN]\!]_{\mathrm{cps}}\rho' = \lambda\!\!\lambda c \cdot f'_M(n')c, \end{cases}$ which are related by induction hypothesis, as $f'_M(n')$ and $\lambda\!\!\lambda c \cdot f'_M(n')c$ represent the same element in the functional space.

$e \equiv \mu f \cdot \lambda x \cdot M$: with the assumption: $\forall x\ \rho(x) \sim \lambda\!\!\lambda c \cdot c(\rho'(x))$,
we prove that: $\forall i\ up\,(strict\,(F_i)) \sim \lambda\!\!\lambda c \cdot c(F'_i))$,
that is: $\forall i\ \forall e \in [\![\sigma]\!]_{\mathrm{dir}},\ e' \in [\![\sigma]\!]^-_{\mathrm{cps}}, e \sim \lambda\!\!\lambda c.c\,(e')\ implies\ F_i\,(e) \sim F'_i\,(e')$,
where: $\begin{cases} F_0 = \bot_{[\![\alpha]\!]_{\mathrm{dir}}\to[\![\beta]\!]_{\mathrm{dir}}}, \\ F_{i+1} = \lambda\!\!\lambda d \cdot [\![M]\!]_{\mathrm{dir}}\rho[d/x, up(strict(F_i))/f] \end{cases}$

and $\begin{cases} F'_0 & = \bot_{[\![\alpha\to\beta]\!]^-_{\mathrm{cps}}}, \\ F'_{i+1} & = \lambda\!\!\lambda d \cdot [\![M]\!]_{\mathrm{cps}}\, \rho[d/x, F'_i/f]. \end{cases}$

Then, by the lemma and the fact that all the functions we use are continuous, we will be in a position to conclude that the meanings of $\mu f \cdot \lambda x \cdot M$ in the two semantics are related by $\sim$.

The proof will be carried out by an auxiliary induction on i.

base case: one has, by the semantics:

$$\begin{cases} up\,(strict\,(F_0)) & = up\,(strict\,(\bot_{[\![\alpha]\!]_{\mathrm{dir}}\to[\![\beta]\!]_{\mathrm{dir}}}) \\ \lambda\!\!\lambda c \cdot c(F'_0) & = \lambda\!\!\lambda c \cdot c(\bot_{[\![\alpha\to\beta]\!]^-_{\mathrm{cps}}}). \end{cases}$$

As $\forall e \in [\![\alpha]\!]_{\mathrm{dir}}$, $\forall e' \in [\![\alpha]\!]^-_{\mathrm{cps}}$: $\begin{cases} (\bot_{[\![\alpha]\!]_{\mathrm{dir}}\to[\![\beta]\!]_{\mathrm{dir}}})(e) & = \bot_{[\![\beta]\!]_{\mathrm{dir}}} \\ (\bot_{[\![\alpha\to\beta]\!]^-_{\mathrm{cps}}})(e') & = \bot_{[\![\beta]\!]^+_{\mathrm{cps}}}, \end{cases}$

the case is established for $i = 0$.

induction step: Let e and e' be respectively elements of $[\![\alpha]\!]_{\mathrm{dir}}$ and $[\![\alpha]\!]^-_{\mathrm{cps}}$. Then, by the semantics given above:

$$\begin{cases} F_{i+1}(e) & = [\![M]\!]_{\mathrm{dir}}\, \rho[e/d, up(strict(F_i))/f] \\ F'_{i+1}(e') & = [\![M]\!]_{\mathrm{cps}}\, \rho'[e'/d, (F'_i))/f]. \end{cases}$$

Suppose that $e \sim \lambda\!\!\lambda c \cdot c(e')$. By auxiliary induction hypothesis, we then have: $up\,(strict\,(F_i)) \sim \lambda\!\!\lambda c \cdot c(F'_i))$ and by principal induction hypothesis, we finally get $F_{i+1}(e) \sim F'_{i+1}(e')$. □

We can now conclude that, as far as non-exceptional terms are concerned, the CPS-semantics above possesses the property of computational adequacy, which is expressed by the following corollary:

Corollary 2 *Let $M : \iota$ be a non-exceptional term such that $FV(M) = \emptyset$. Then: $M \uparrow_v \iff [\![M]\!]_{\mathrm{cps}} = \bot_{[\![\iota]\!]^+_{\mathrm{cps}}}$.*

Proof. By Corollary 1 and Proposition 1. □

6 Conclusion

The previous results seem to indicate that the CPS-semantics we propose for λ_{exn} is convenient for modeling the evaluation rules of the calculus, even if, of course, it remains to prove the property of computational adequacy for exceptional terms.

The next step will be then to study what properties are conserved by addition of the recursion operator and if others constructions, like for example *if-then-else* may in turn be added with benefit to make λ_{exn} a really expressive functional language.

References

[de Groote,1995] Ph. de Groote, *A Simple Calculus of Exception Handling*. Lectures Notes in Computer Science 902, 1995

[Winskel,1993] G. Winskel, *The Formal Semantics of Programming Languages*. MIT Press, 1993

[Gunter,1992] Carl A. Gunter, *Semantics of Programming Languages*. MIT Press, 1992

[Paulson,1987] L. C. Paulson, *Logic and Computation: Interactive Proof with Cambridge LCF*. Cambridge University Press, 1987

[Reynolds,1974] John C. Reynolds, *On the Relation between Direct and Continuation Semantics*. Proceedings of the second Colloquium on Automata, Language and Programming, LNCS 14, 1974

Appendix A: Proof of the Lemma 1

We shall proceed by induction on the type of a_i and b_i:

Base case: $\forall i:\ a_i \in [\![\iota]\!]_{\mathrm{dir}},\ b_i \in [\![\iota]\!]^+_{\mathrm{cps}}$

first case: we have, for all i: $a_i = \bot_{[\![\iota]\!]_{\mathrm{dir}}}$, $b_i = \bot_{[\![\iota]\!]^+_{\mathrm{cps}}}$.
Hence, $\bigsqcup_i(a_i) = \bot_{[\![\iota]\!]_{\mathrm{dir}}}$, $\bigsqcup_i(b_i) = \bot_{[\![\iota]\!]^+_{\mathrm{cps}}}$,
which gives, by the definition of $\sim$: $\bigsqcup_i(a_i) \sim \bigsqcup_i(b_i)$.

second case: $\exists i_0$ such that $\forall i,\ i \geq i_0$ implies: $a_i = *,\ b_i = \lambda c \cdot c(*)$.
Then, we have: $\bigsqcup_i(a_i) = *,\ \bigsqcup_i(b_i) = \lambda c \cdot c(*)$,
which are related by definition.

induction case: $\forall i:\ a_i \in [\![\alpha \to \beta]\!]_{\mathrm{dir}},\ b_i \in [\![\alpha \to \beta]\!]^+_{\mathrm{cps}}$

first case: $\forall i:\ (a_i) = \bot_{[\![\alpha\to\beta]\!]_{\mathrm{dir}}},\ (b_i) = \bot_{[\![\alpha\to\beta]\!]^+_{\mathrm{cps}}}$.
As before, we get $\bigsqcup_i(a_i) \sim \bigsqcup_i(b_i)$ by definition.

second case: $\exists i_0$ such that $\forall i,\ i \geq i_0$ implies

$$\exists f_i, f_i':\ a_i = up\,(strict\,(f_i)),\ b_i = \lambda c \cdot c(f_i')$$

and we know then from the third assumption that:

$$\forall e \in [\![\alpha]\!]_{\mathrm{dir}},\ e' \in [\![\alpha]\!]^-_{\mathrm{cps}}:\ e \sim \lambda c \cdot c(e') \Longrightarrow f_i(e) \sim f_i'(e').$$

Hence we have: $\bigsqcup_i(a_i) \neq \bot_{[\![\alpha\to\beta]\!]_{\mathrm{dir}}}$ and $\bigsqcup_i(b_i) \neq \bot_{[\![\alpha\to\beta]\!]^+_{\mathrm{cps}}}$.
Then, by the definition of $\sim$ for the functional types, in order to get $\bigsqcup_i(a_i) \sim \bigsqcup_i(b_i)$, we need to prove:

$$\exists f \in [\![\alpha]\!]_{\mathrm{dir}} \to [\![\beta]\!]_{\mathrm{dir}}\ ,\ f' \in [\![\alpha\to\beta]\!]^-_{\mathrm{cps}} = [\![\alpha]\!]^-_{\mathrm{cps}} \to [\![\beta]\!]^+_{\mathrm{cps}}$$
$$\text{such that:} \begin{cases} \bigsqcup_i(a_i) = up\,(strict\,(f)), \\ \bigsqcup_i(b_i) = \lambda\!\!\lambda c \cdot c(f'), \\ \forall e \in [\![\alpha]\!]_{\mathrm{dir}}, e' \in [\![\alpha]\!]^-_{\mathrm{cps}} : e \sim \lambda\!\!\lambda c \cdot c(e') \Longrightarrow f(e) \sim f'(e'). \end{cases}$$

As the functions UP, STRICT and the continuations c are continuous and as the function space we use is that of continuous functions, we have:

$$\begin{cases} \bigsqcup_i(a_i) = \bigsqcup_i(up\,(strict(f_i))) = up\,(strict\,(\bigsqcup_i(f_i))) \\ \bigsqcup_i(b_i) = \bigsqcup_i(\lambda\!\!\lambda c \cdot c(f'_i)) \qquad\quad = \lambda\!\!\lambda c \cdot c(\bigsqcup_i(f'_i)) \end{cases}$$
$$\text{where:} \begin{cases} \forall i : f_i \in [\![\alpha]\!]_{\mathrm{dir}} \to [\![\beta]\!]_{\mathrm{dir}},\ f'_i \in [\![\alpha\to\beta]\!]^-_{\mathrm{cps}} \\ \forall i,j :\ i<j \Longrightarrow (f_i \sqsubseteq f_j,\ f'_i \sqsubseteq f'_j). \end{cases}$$

As the sets $[\![\alpha]\!]_{\mathrm{dir}} \to [\![\beta]\!]_{\mathrm{dir}}$ and $[\![\alpha\to\beta]\!]^-_{\mathrm{cps}}$ are CPOs whenever $\alpha\to\beta$ is a type, we have $\bigsqcup_i(f_i) \in [\![\alpha]\!]_{\mathrm{dir}} \to [\![\beta]\!]_{\mathrm{dir}}$
and $\bigsqcup_i(f'_i) \in [\![\alpha\to\beta]\!]^-_{\mathrm{cps}}$.
Let $e \in [\![\alpha]\!]_{\mathrm{dir}}$, $e' \in [\![\alpha]\!]^-_{\mathrm{cps}}$ be such that $e \sim \lambda\!\!\lambda c \cdot c(e')$.
We have: $(\bigsqcup_i f_i)(e) = \bigsqcup_i(f_i(e))$ and $(\bigsqcup_i f'_i)(e') = \bigsqcup_i(f'_i(e'))$.
From the second assumption above, we know that: $\forall i,\ f_i(e) \sim f'_i(e')$ and we have too: $\forall i : f_i(e) \in [\![\beta]\!]_{\mathrm{dir}},\ f'_i(e') \in [\![\beta]\!]^+_{\mathrm{cps}}$.
Hence, by induction hypothesis, we can conclude $\bigsqcup_i(f_i(e)) \sim \bigsqcup_i(f'_i(e'))$. □