Axioms for Recursion in Call-by-Value[†]

Masahito Hasegawa (hassei@kurims.kyoto-u.ac.jp) and
Yoshihiko Kakutani (kakutani@kurims.kyoto-u.ac.jp)
Research Institute for Mathematical Sciences, Kyoto University, Kyoto 606-8502 Japan

Abstract. We propose an axiomatization of fixpoint operators in typed call-by-value programming languages, and give its justifications in two ways. First, it is shown to be sound and complete for the notion of uniform T-fixpoint operators of Simpson and Plotkin. Second, the axioms precisely account for Filinski's fixpoint operator derived from an iterator (infinite loop constructor) in the presence of first-class continuations, provided that we define the uniformity principle on such an iterator via a notion of effect-freeness (centrality). We then explain how these two results are related in terms of the underlying categorical structures.

Keywords: recursion, iteration, call-by-value, continuations, categorical semantics.

1. Introduction

While the equational theories of *fixpoint operators* in call-by-name programming languages and in domain theory have been extensively studied and now there are some canonical axiomatizations (including the *iteration theories* [1] and *Conway theories*, equivalently *traced cartesian categories* [12] – see [27] for the latest account), there seems no such widely-accepted result in the context of *call-by-value (cbv) programming languages*, possibly with side effects. Although the implementation of recursion in "impure" programming language has been well-known, it seems that the underlying semantic nature of recursive computation in the presence of side-effects has not been studied at a sufficiently general level. Regarding the widespread use of call-by-value programming languages and the importance of recursion in real life programming, it is desirable to have theoretically motivated and justified principles for reasoning about recursive computation in a call-by-value setting.

In this paper we propose a candidate of such an axiomatization, which consists of three simple axioms, including a uniformity principle analogous to that in the call-by-name setting. Our axiomatization, of *stable uniform call-by-value fixpoint operators* to be introduced below, is justified by the following two main results:

[†] An extended abstract of this work appeared in *Proc. Foundations of Software Science and Computation Structures (FoSSaCS 2001)*, Springer LNCS Vol. 2030.

1. The λ_c*-calculus* (*computational lambda calculus*) [18] with a stable uniform cbv fixpoint operator is sound and complete for the models based on the notion of *uniform T-fixpoint operators* of Simpson and Plotkin [27].

2. In the *call-by-value* $\lambda\mu$*-calculus* [25] (= the λ_c-calculus plus *first-class continuations*) there is a bijective correspondence between stable uniform cbv fixpoint operators and *uniform iterators*, via Filinski's construction of *recursion from iteration* [5].

The notion of uniform T-fixpoint operators arose from the context of *Axiomatic Domain Theory* [7, 26]. By letting T be a lifting monad on a category of predomains, a uniform T-fixpoint operator amounts to a uniform fixpoint operator on domains (the least fixpoint operator in the standard order-theoretic setting). In general, T can be any strong monad on a category with finite products, thus a uniform T-fixpoint operator makes sense for any model of the computational lambda calculus in terms of strong monads [18], and Simpson and Plotkin [27] suggest the possibility of using uniform T-fixpoint operators for modelling call-by-value recursion. This line of considerations leads us to our first main result. In fact, we distill our axioms from the uniform T-fixpoint operators.

A surprise is the second one, in that the axioms precisely account for Filinski's cbv fixpoint operator derived from an *iterator* (infinite loop constructor) and first-class continuations, provided that we refine Filinski's notion of uniformity, for which the distinction between values and effect-insensitive programs (characterised by the notion of *centrality*) [22, 28, 10] is essential. Using our axioms, we establish the bijectivity result between fixpoint operators and iterators. Therefore here is an interesting coincidence of a category-theoretic axiomatics (of Simpson and Plotkin) with a program construction (of Filinski).

However, we also show that, after sorting out the underlying categorical semantics, Filinski's construction combined with the *Continuation-Passing Style (CPS) transformation* can be understood within the abstract setting of Simpson and Plotkin. The story is summarised as follows. As noted by Filinski, the CPS-transform of an iterator is a usual (call-by-name) fixpoint operator on the types of the form R^A in the target $\lambda\beta\eta$-calculus, where R is the answer type. If we let T be the continuation monad $R^{R^{(-)}}$, then the uniform T-fixpoint operator precisely amounts to the uniform fixpoint operator on the types R^A. Since our first main result is that the stable uniform cbv fixpoint operator is sound and complete for such uniform T-fixpoint operators, it turns out that Filinski's construction combined with the CPS transformation can be regarded as a consequence of the general categorical axiomatics; by specialising it to the setting with a continuation monad, we obtain a semantic version of the second main result.

Construction of this paper
In Section 2 we recall the λ_c-calculus and the call-by-value $\lambda\mu$-calculus, which will be used as our working languages in this paper. In Section 3 we introduce our axioms for fixpoint operators in these calculi (Definition 3) and give basic syntactic results. Section 4 demonstrates how our axioms are used for establishing Filinski's correspondence between recursion and iteration (which in fact gives a syntactic proof of the second main result). Up to this section, all results are presented in an entirely syntactic manner. In Section 5 we start to look at the semantic counterpart of our axiomatization, by recalling the categorical models of the λ_c-calculus and the call-by-value $\lambda\mu$-calculus. We then recall the notion of uniform T-fixpoint operators on these models in Section 6, and explain how our axioms are distilled from the uniform T-fixpoint operators (Theorem 2, the first main result). In Section 7, we specialise the result in the previous section to the models of the call-by-value $\lambda\mu$-calculus, and give a semantic proof of the second main result (Theorem 4). Section 8 gives some concluding remarks.

2. The Call-by-Value Calculi

The λ_c-calculus (computational lambda calculus) [18], an improvement of the call-by-value λ-calculus [21], is sound and complete for

1. categorical models based on *strong monads* (Moggi [18])

2. Continuation-Passing Style transformation into the $\lambda\beta\eta$-calculus (Sabry and Felleisen [23])

and has proved useful for reasoning about call-by-value programs. In particular, it can be seen as the theoretical backbone of (the typed version of) the theory of *A-normal forms* [8], which enables us to optimise call-by-value programs directly without performing the CPS transformation.

For these reasons, we take the λ_c-calculus as a basic calculus for typed call-by-value programming languages. We also use an extension of the λ_c-calculus with first-class continuations, called the call-by-value $\lambda\mu$-calculus, for which the soundness and completeness results mentioned above have been extended by Selinger [25].

2.1. The λ_c-calculus

The syntax, typing rules and axioms on the well-typed terms of the λ_c-calculus are summarised in Figure 1. The types, terms and typing judgements are those of the standard simply typed lambda calculus (including the unit $\top$ and binary products $\times$).[1] c^σ ranges over the constants of type σ. As an abbreviation, we write $\mathsf{let}\ x^\sigma\ \mathsf{be}\ M\ \mathsf{in}\ N$ for $(\lambda x^\sigma.N)M$. $\mathrm{FV}(M)$ denotes the set of free variables in M. (As long as there is no confusion, we may use italic small letters for both variables and values. Capital letters usually range over terms, though we may also use some capital letters like F, G, H for higher-order functional values.) The crucial point is that we have the notion of values, and the axioms are designed so that the above-mentioned completeness results hold. Below we may call a term a value if it is provably equal to a value defined by the grammar. We write $g \circ f$ for the composition $\lambda x.g\,(f\,x)$ of values f and g, and id_σ for the identity function $\lambda x^\sigma.x$.

In the sequel, we are concerned not just about the pure λ_c-calculus but also about its extensions with additional constructs and axioms. A *λ_c-theory* is a typed equational theory on the well-typed expressions of the λ_c-calculus (possibly with additional constructs) which is a congruence on all term constructions and contains the axioms of the λ_c-calculus. A λ_c-theory can be typically specified by the additional axioms (as the congruence generated from them), or as the equational theory induced by a model in the sense of Section 5, i.e. $\Gamma \vdash M = N : \sigma$ iff $[\![\Gamma \vdash M : \sigma]\!] = [\![\Gamma \vdash N : \sigma]\!]$.

Centre and focus

In call-by-value languages, we often regard values as representing effect-free (finished or suspended) computations. While this intuition is valid, the converse may not always be justified; in fact, the answer depends on the computational effects under consideration.

DEFINITION 1 (centre, focus). *In a λ_c-theory, we say that a term $M : \sigma$ is* central *if it commutes with any other computational effect, that is,*

$$\mathsf{let}\ x^\sigma\ \mathsf{be}\ M\ \mathsf{in}\ \mathsf{let}\ y^\tau\ \mathsf{be}\ N\ \mathsf{in}\ L \;=\; \mathsf{let}\ y^\tau\ \mathsf{be}\ N\ \mathsf{in}\ \mathsf{let}\ x^\sigma\ \mathsf{be}\ M\ \mathsf{in}\ L \;:\; \theta$$

holds for any $N : \tau$ and $L : \theta$, where x and y are not free in M and N. In addition, we say that $M : \sigma$ is focal *if it is central and moreover* copyable *and* discardable*, i.e.,* $\mathsf{let}\ x^\sigma\ \mathsf{be}\ M\ \mathsf{in}\ \langle x, x\rangle = \langle M, M\rangle : \sigma \times \sigma$ *and* $\mathsf{let}\ x^\sigma\ \mathsf{be}\ M\ \mathsf{in}\ * = * : \top$ *hold.*

[1] We do not include the "computation types" $T\sigma$ and associated constructs, as they can be defined by $T\sigma = \top \to \sigma$, $[M] = \lambda u^\top.M$ and $\mu(M) = M\,*$.

$$
\begin{array}{llll}
\text{Types} & \sigma,\tau & ::= & b \mid \sigma\to\tau \mid \top \mid \sigma\times\tau \quad \text{where } b \text{ ranges over base types}\\
\text{Terms} & L,M,N & ::= & x \mid c^\sigma \mid \lambda x^\sigma.M \mid M\,N \mid * \mid \langle M,N\rangle \mid \pi_1\,M \mid \pi_2\,M\\
\text{Values} & V,U & ::= & x \mid c^\sigma \mid \lambda x^\sigma.M \mid * \mid \langle V,U\rangle \mid \pi_1\,V \mid \pi_2\,V
\end{array}
$$

Typing Rules:

$$
\frac{}{\Gamma\vdash x:\sigma}\;x\!:\!\sigma\!\in\!\Gamma \qquad \frac{}{\Gamma\vdash c^\sigma:\sigma} \qquad \frac{\Gamma,x:\sigma\vdash M:\tau}{\Gamma\vdash\lambda x^\sigma.M:\sigma\to\tau} \qquad \frac{\Gamma\vdash M:\sigma\to\tau\quad\Gamma\vdash N:\sigma}{\Gamma\vdash M\,N:\tau}
$$

$$
\frac{}{\Gamma\vdash *:\top} \qquad \frac{\Gamma\vdash M:\sigma\quad\Gamma\vdash N:\tau}{\Gamma\vdash\langle M,N\rangle:\sigma\times\tau} \qquad \frac{\Gamma\vdash M:\sigma\times\tau}{\Gamma\vdash\pi_1\,M:\sigma} \qquad \frac{\Gamma\vdash M:\sigma\times\tau}{\Gamma\vdash\pi_2\,M:\tau}
$$

Axioms:

$$
\begin{array}{lcll}
\mathsf{let}\ x^\sigma\ \mathsf{be}\ V\ \mathsf{in}\ M & = & M[V/x] & \\
\lambda x^\sigma.V\,x & = & V & (x\notin\mathrm{FV}(V))\\
V & = & * & (V:\top)\\
\pi_i\langle V_1,V_2\rangle & = & V_i & \\
\langle\pi_1\,V,\pi_2\,V\rangle & = & V & \\
\mathsf{let}\ x^\sigma\ \mathsf{be}\ M\ \mathsf{in}\ x & = & M & \\
\multicolumn{4}{l}{\mathsf{let}\ y^\tau\ \mathsf{be}\ (\mathsf{let}\ x^\sigma\ \mathsf{be}\ L\ \mathsf{in}\ M)\ \mathsf{in}\ N = \mathsf{let}\ x^\sigma\ \mathsf{be}\ L\ \mathsf{in}\ \mathsf{let}\ y^\tau\ \mathsf{be}\ M\ \mathsf{in}\ N}\\
 & & & (x\notin\mathrm{FV}(N))\\
M\,N & = & \mathsf{let}\ f^{\sigma\to\tau}\ \mathsf{be}\ M\ \mathsf{in}\ \mathsf{let}\ x^\sigma\ \mathsf{be}\ N\ \mathsf{in}\ f\,x & \\
 & & & (M:\sigma\to\tau, N:\sigma)\\
\langle M,N\rangle & = & \mathsf{let}\ x^\sigma\ \mathsf{be}\ M\ \mathsf{in}\ \mathsf{let}\ y^\tau\ \mathsf{be}\ N\ \mathsf{in}\ \langle x,y\rangle & \\
 & & & (M:\sigma, N:\tau)\\
\pi_i\,M & = & \mathsf{let}\ x^{\sigma\times\tau}\ \mathsf{be}\ M\ \mathsf{in}\ \pi_i\,x & (M:\sigma\times\tau)
\end{array}
$$

where $\mathsf{let}\ x^\sigma\ \mathsf{be}\ M\ \mathsf{in}\ N$ stands for $(\lambda x^\sigma.N)M$

Figure 1. The λ_c-calculus

It is worth emphasising that a value is always focal, but the converse is not true (see Section 7.3). A detailed analysis of these concepts in several λ_c-theories is found in [10]; see also discussions in Section 5.

2.2. The call-by-value $\lambda\mu$-calculus

Our call-by-value $\lambda\mu$-calculus, summarised in Figure 2, is the version due to Selinger [25]. We regard it as an extension of the λ_c-calculus with first-class continuations and sum types (the empty type $\bot$ and binary sums $+$). We write $\neg\sigma$ for the type $\sigma\to\bot$ ("negative type").

The typing judgements take the form $\Gamma\vdash M:\sigma\mid\Delta$ where $\Delta = \alpha_1:\tau_1,\ldots,\alpha_n:\sigma_n$ is a sequence of *names* (ranged over by α, β,...) with their types. A judgement $x_1:\sigma_1,\ldots,x_m:\sigma_m\vdash M:\tau\mid\alpha_1:\tau_1,\ldots,\alpha_n:\tau_n$ represents a well-typed term M with at most m *free variables* $x_1,\ldots,x_m$ and n *free names* $\alpha_1,\ldots,\alpha_n$. We write $\mathrm{FN}(M)$ for

$$
\begin{array}{llll}
\text{Types} & \sigma, \tau & ::= & \cdots \mid \bot \mid \sigma + \tau \\
\text{Terms} & L, M, N & ::= & \cdots \mid [\alpha]M \mid \mu\alpha^{\sigma}.M \mid [\alpha,\beta]M \mid \mu(\alpha^{\sigma},\beta^{\tau}).M \\
\text{Values} & V, U & ::= & \cdots \mid \mu(\alpha^{\sigma},\beta^{\tau}).[\alpha]V \mid \mu(\alpha^{\sigma},\beta^{\tau}).[\beta]V \\
 & & & \qquad \text{where } \alpha, \beta \notin \mathrm{FN}(V)
\end{array}
$$

Additional Typing Rules:

$$
\frac{\Gamma \vdash M : \sigma \mid \Delta}{\Gamma \vdash [\alpha]M : \bot \mid \Delta}\ \alpha{:}\sigma \in \Delta
\qquad
\frac{\Gamma \vdash M : \bot \mid \alpha : \sigma, \Delta}{\Gamma \vdash \mu\alpha^{\sigma}.M : \sigma \mid \Delta}
$$

$$
\frac{\Gamma \vdash M : \sigma + \tau \mid \Delta}{\Gamma \vdash [\alpha,\beta]M : \bot \mid \Delta}\ \alpha{:}\sigma, \beta{:}\tau \in \Delta
\qquad
\frac{\Gamma \vdash M : \bot \mid \alpha : \sigma, \beta : \tau, \Delta}{\Gamma \vdash \mu(\alpha^{\sigma},\beta^{\tau}).M : \sigma + \tau \mid \Delta}
$$

Additional Axioms:

$$
\begin{array}{lll}
V\,(\mu\alpha^{\sigma}.M) & = \mu\beta^{\tau}.M[[\beta](V(-))/[\alpha](-)] & (V : \sigma \to \tau) \\
[\alpha'](\mu\alpha^{\sigma}.M) & = M[\alpha'/\alpha] & \\
\mu\alpha^{\sigma}.[\alpha]M & = M & (\alpha \notin \mathrm{FN}(M)) \\
[\alpha',\beta'](\mu(\alpha^{\sigma},\beta^{\tau}).M) & = M[\alpha'/\alpha, \beta'/\beta] & \\
\mu(\alpha^{\sigma},\beta^{\tau}).[\alpha,\beta]M & = M & (\alpha, \beta \notin \mathrm{FN}(M)) \\
[\alpha]M & = M & (M : \bot) \\
[\alpha]M & = \mathsf{let}\ x^{\sigma}\ \mathsf{be}\ M\ \mathsf{in}\ [\alpha]x & (M : \sigma) \\
[\alpha,\beta]M & = \mathsf{let}\ x^{\sigma+\tau}\ \mathsf{be}\ M\ \mathsf{in}\ [\alpha,\beta]x & (M : \sigma + \tau)
\end{array}
$$

Figure 2. The call-by-value $\lambda\mu$-calculus

the set of free names in M. In this judgement, M can be thought of as a proof of the sequent $\sigma_1, \ldots, \sigma_m \vdash \tau, \tau_1, \ldots, \tau_n$ or the proposition $(\sigma_1 \wedge \ldots \wedge \sigma_m) \to (\tau \vee \tau_1 \vee \ldots \vee \tau_n)$ in the classical propositional logic. Among the additional axioms, the first one involves the *mixed substitution* $M[C(-)/[\alpha](-)]$ for a term M, a context $C(-)$ and a name α, which is the result of recursively replacing any subterm of the form $[\alpha]N$ by $C(N)$ and any subterm of the form $[\alpha_1, \alpha_2]N$ (with $\alpha = \alpha_1$ or $\alpha = \alpha_2$) by $C(\mu\alpha.[\alpha_1, \alpha_2]N)$. See [25] for further details on these syntactic conventions.

Remark 1. We have chosen the cbv $\lambda\mu$-calculus as our working language firstly because we intend the results in this paper to be compatible with the duality result of the second author [15] (see Section 7) which is based on Selinger's work on the $\lambda\mu$-calculus [25], and secondly because it has a well-established categorical semantics, again thanks to Selinger. However our results are not specific to the $\lambda\mu$-calculus; they apply also to any other language with similar semantics – for example, we could have used Hofmann's axiomatization of control operators [13]. Also, strictly speaking, the inclusion of sum types (coproducts) is not necessary in the main development of this paper, though they enable us to describe iterators more naturally (as general feedback operators, see Remark 3 in Section 4) and are also used in some principles on iterators like diagonal property (see Section 8), and crucially needed for the duality result in [25, 15].

Example 1. As an example, we can define terms for the "double-negation elimination" and the "initial map":

$$\begin{array}{lcl}\mathcal{C}_\sigma & = & \lambda m^{\neg\neg\sigma}.\mu\alpha^\sigma.m\,(\lambda x^\sigma.[\alpha]x) : \neg\neg\sigma \to \sigma \\ \mathcal{A}_\sigma & = & \lambda x^\perp.\mu\alpha^\sigma.x : \perp \to \sigma\end{array}$$

One can check that these combinators satisfy Hofmann's axioms in [13]. Also, we can use $\mathcal{C}_\sigma\,(\lambda k^{\neg\sigma}.M[k(-)/[\alpha](-)])$ for $\mu\alpha^\sigma.M$ (as will be done in the programming example in SML/NJ in Section 4).

Centre and focus

In the presence of first-class continuations, central and focal terms coincide [28, 25], and enjoy a simple characterisation (*thunkability* [28]).

LEMMA 1. *In a cbv $\lambda\mu$-theory, the following conditions on a term $M : \sigma$ are equivalent.*

1. *M is central.*
2. *M is focal.*
3. *(thunkability)* $\mathsf{let}\ x^\sigma\ \mathsf{be}\ M\ \mathsf{in}\ \lambda u^\top.x = \lambda u^\top.M : \top \to \sigma$ *holds.*[2]

We also note that central terms and values agree at function types [25].

LEMMA 2. *In a cbv $\lambda\mu$-theory, a term $M : \sigma \to \tau$ is central if and only if it is a value, i.e., $M = \lambda x^\sigma.M\,x$ (with x not free in M) holds.*

3. Axioms for Recursion

Throughout this section, we work in a λ_c-theory.

3.1. Rigid functionals

The key in our axiomatization of call-by-value fixpoint operators is the notion of *uniformity.* In the call-by-name setting, we define the

[2] Equivalently: $\mathsf{let}\ x^\sigma\ \mathsf{be}\ M\ \mathsf{in}\ \lambda k^{\neg\sigma}.k\,x = \lambda k^{\neg\sigma}.k\,M : \neg\neg\sigma$.

uniformity for fixpoint operators with respect to the *strict maps*, i.e., those that preserve the bottom element (divergence). In a call-by-value setting, however, we cannot define uniformity via this particular notion of strict maps, simply because everything is strict – if an input does not terminate, the whole program cannot terminate. Instead we propose to define the uniformity principle with respect to a class of functionals that use their argument functions in a constrained way.

DEFINITION 2 (rigid functional). *A value* $H : (\sigma \to \tau) \to \sigma' \to \tau'$ *is called* rigid *if* $H\,(\lambda x.M\,x) = \lambda y.H\,M\,y$ *holds for any* $M : \sigma \to \tau$*, where* x *and* y *are not free in* H *and* M*.*

The word "rigid" was coined by Filinski in [5] (see discussions in Section 7.3). Intuitively, a rigid functional uses its argument exactly once, and it does not matter whether the argument is evaluated beforehand or evaluated at its actual use.

LEMMA 3. $\mathsf{id}_{\sigma\to\tau} : (\sigma \to \tau) \to \sigma \to \tau$ *is rigid. Also, if* $H : (\sigma \to \tau) \to \sigma' \to \tau'$ *and* $H' : (\sigma' \to \tau') \to \sigma'' \to \tau''$ *are rigid, so is* $H' \circ H : (\sigma \to \tau) \to \sigma'' \to \tau''$*.*

LEMMA 4. *If* $H : (\sigma \to \tau) \to \sigma' \to \tau'$ *is rigid,* $H = \lambda f^{\sigma\to\tau}.\lambda y^{\sigma'}.H\,f\,y$ *holds (where* f *and* y *are not free in* H*).*

Example 2. The reader may want to see rigid functionals in more concrete ways. In the case of settings with first-class continuations, we have such a characterisation of rigid functionals, see Section 7.3. In general cases, a rigid functional typically takes the following form:

$$H \;=\; \lambda f^{\sigma\to\tau}.\lambda y^{\sigma'}.\mathsf{let}\ x\ \mathsf{be}\ f\,(h\,y)\ \mathsf{in}\ N : (\sigma \to \tau) \to \sigma' \to \tau'$$

where $h : \sigma' \to \sigma$ satisfies the following property: $h\,v$ is central for any value $v : \sigma'$ – later, such an h will be called "total" (Definition 4). N can be any term, possibly with side effects. It is easily seen that such an H satisfies $H(\lambda x.M\,x) = \lambda y.H\,M\,y$ in any λ_c-theory. On the other hand, in the presence of side effects, many purely functional terms fail to be rigid – e.g., constant functionals, as well as functionals like $\lambda f.f \circ f$.

3.2. Axioms for recursion

Now we are ready to state the main definition of this paper: our axiomatization of the call-by-value fixpoint operators.

DEFINITION 3 (stable uniform call-by-value fixpoint operator). *A type-indexed family of closed values* $\mathsf{fix}^{\mathsf{v}}_{\sigma\to\tau} : ((\sigma \to \tau) \to \sigma \to \tau) \to \sigma \to \tau$ *is called a* stable uniform call-by-value fixpoint operator *if the following conditions are satisfied:*

1. *(cbv fixpoint) For any value* $F : (\sigma \to \tau) \to \sigma \to \tau$
   $\mathsf{fix}^{\mathsf{v}}_{\sigma\to\tau}\, F \;=\; \lambda x^{\sigma}.F\,(\mathsf{fix}^{\mathsf{v}}_{\sigma\to\tau}\, F)\, x$
   (where x is not free in F)

2. *(stability) For any value* $F : (\sigma \to \tau) \to \sigma \to \tau$
   $\mathsf{fix}^{\mathsf{v}}_{\sigma\to\tau}\, F \;=\; \mathsf{fix}^{\mathsf{v}}_{\sigma\to\tau}\,(\lambda f^{\sigma\to\tau}.\lambda x^{\sigma}.F\, f\, x)$
   (where f, x are not free in F)

3. *(uniformity) For values* $F : (\sigma \to \tau) \to \sigma \to \tau$, $G : (\sigma' \to \tau') \to \sigma' \to \tau'$ *and a rigid* $H : (\sigma \to \tau) \to \sigma' \to \tau'$, *if* $H \circ F = G \circ H$ *holds, then* $H\,(\mathsf{fix}^{\mathsf{v}}_{\sigma\to\tau}\, F) \;=\; \mathsf{fix}^{\mathsf{v}}_{\sigma'\to\tau'}\, G$

$$\begin{tikzcd} \sigma \to \tau \arrow[r, "F"] \arrow[d, "H"'] & \sigma \to \tau \arrow[d, "H"] \\ \sigma' \to \tau' \arrow[r, "G"'] & \sigma' \to \tau' \end{tikzcd} \quad \Rightarrow \quad H\,(\mathsf{fix}^{\mathsf{v}} F) = \mathsf{fix}^{\mathsf{v}} G$$

The first axiom is known as the *call-by-value fixpoint equation*; the eta-expansion in the right-hand-side means that $\mathsf{fix}^{\mathsf{v}}_{\sigma\to\tau}\, F$ is equal to a value. The second axiom says that, though the functionals F and $\lambda f.\lambda x.F\, f\, x$ may behave differently, their fixpoints, when applied to values, satisfy the same fixpoint equation and cannot be distinguished. The last axiom is a call-by-value variant of Plotkin's *uniformity principle*; here the *rigid functionals* play the rôle of *strict functions* in the uniformity principle for the call-by-name fixpoint operators. Our uniformity axiom can be justified by the fact that $H\,(\mathsf{fix}^{\mathsf{v}}_{\sigma\to\tau}\, F)$ satisfies the same fixpoint equation as $\mathsf{fix}^{\mathsf{v}}_{\sigma'\to\tau'}\, G$ when H is rigid and $H \circ F = G \circ H$ holds:

$$\begin{array}{lll} & H\,(\mathsf{fix}^{\mathsf{v}}_{\sigma\to\tau}\, F) & \\ = & H\,(\lambda x^{\sigma}.F\,(\mathsf{fix}^{\mathsf{v}}_{\sigma\to\tau}\, F)\, x) & \text{cbv fixpoint equation for } \mathsf{fix}^{\mathsf{v}}_{\sigma\to\tau}\, F \\ = & \lambda y^{\sigma'}.H\,(F\,(\mathsf{fix}^{\mathsf{v}}_{\sigma\to\tau}\, F))\, y & H \text{ is rigid} \\ = & \lambda y^{\sigma'}.G\,(H\,(\mathsf{fix}^{\mathsf{v}}_{\sigma\to\tau}\, F))\, y & H \circ F = G \circ H \end{array}$$

The following consideration confirms that the rigidness assumption cannot be dropped from the uniformity axiom. Let $H : (\sigma \to \tau) \to \sigma' \to \tau'$ be any value so that $H = \lambda f.\lambda y.H\, f\, y$ holds. Take any term M of type $\sigma \to \tau$. Define $F : (\sigma \to \tau) \to \sigma \to \tau$ and $G : (\sigma' \to \tau') \to \sigma' \to \tau'$

by $F = \lambda f.M$ and $G = \lambda g.\mathsf{let}\ f\ \mathsf{be}\ M\ \mathsf{in}\ \lambda y.H\, f\, y$ (with f, g not free in M). Then we have $H \circ F = G \circ H$, and the uniformity would ask $H\,(\mathsf{fix}^{\mathsf{v}}\, F) = \mathsf{fix}^{\mathsf{v}}\, G$ to be hold; it is easily seen that $\mathsf{fix}^{\mathsf{v}}\, F = \lambda x.M\, x$ and $\mathsf{fix}^{\mathsf{v}}\, G = \lambda y.H\, M\, y$, hence H must be rigid.

Remark 2. As easily seen, the uniformity implies that any rigid functional preserves the fixpoints of the identity maps: $H\,(\mathsf{fix}^{\mathsf{v}}_{\sigma\to\tau}\ \mathsf{id}_{\sigma\to\tau}) = \mathsf{fix}^{\mathsf{v}}_{\sigma'\to\tau'}\ \mathsf{id}_{\sigma'\to\tau'}$ for any rigid $H : (\sigma \to \tau) \to \sigma' \to \tau'$. It is tempting to define a notion of "call-by-value strictness" as this preservation of the fixpoints of identities. In the pure functional settings (like the call-by-value PCF) where the divergence is the only effect, this call-by-value strictness actually coincides with the rigidness (this can be verified by inspecting the standard domain-theoretic model; see also Section 6). However, in the presence of other effects (in particular the case with first-class continuations which we will study in the Section 4), rigidness is a much stronger requirement than this call-by-value strictness; for instance, the constant functional $\lambda f.\mathsf{fix}^{\mathsf{v}}\,\mathsf{id}$ as well as the "twice" functional $\lambda f.f \circ f$ are call-by-value strict (hence rigid in a pure functional setting), but they are not rigid in many λ_c-theories and cannot be used in the uniformity principle.

3.3. On the axiomatizations of uniformity

There are some alternative ways of presenting the axioms of stable uniform cbv fixpoint operators. In particular, in [5] Filinski proposed a single uniformity axiom which amounts to our stability and uniformity axioms.

LEMMA 5. *For values $F : (\sigma \to \tau) \to \sigma \to \tau$, $G : (\sigma' \to \tau') \to \sigma' \to \tau'$ and a rigid $H : (\sigma \to \tau) \to \sigma' \to \tau'$, $H \circ F = G \circ H$ implies $H \circ (\lambda f.\lambda x.F\, f\, x) = (\lambda g.\lambda y.G\, g\, y) \circ H$ (where f, x are not free in F and g, y are not free in G).*

$$
\begin{tikzcd}
\sigma \to \tau \arrow[r, "F"] \arrow[d, "H"'] & \sigma \to \tau \arrow[d, "H"] & & \sigma \to \tau \arrow[r, "\lambda f.\lambda x.F\,f\,x"] \arrow[d, "H"'] & \sigma \to \tau \arrow[d, "H"] \\
\sigma' \to \tau' \arrow[r, "G"'] & \sigma' \to \tau' & \Rightarrow & \sigma' \to \tau' \arrow[r, "\lambda g.\lambda y.G\,g\,y"'] & \sigma' \to \tau'
\end{tikzcd}
$$

Proof.

$$
\begin{aligned}
H \circ (\lambda f.\lambda x.F\, f\, x) &= \lambda f.H\,(\lambda x.F\, f\, x) \\
&= \lambda f.\lambda y.H\,(F\, f)\, y \quad && H \text{ is rigid} \\
&= \lambda f.\lambda y.G\,(H\, f)\, y \quad && H \circ F = G \circ H \\
&= (\lambda g.\lambda y.G\, g\, y) \circ H && \square
\end{aligned}
$$

PROPOSITION 1. *The stability axiom and uniformity axiom are equivalent to the following* Filinski's uniformity axiom *[5]:*

For values $F : (\sigma \to \tau) \to \sigma \to \tau$, $G : (\sigma' \to \tau') \to \sigma' \to \tau'$ and a rigid $H : (\sigma \to \tau) \to \sigma' \to \tau'$, if $H \circ (\lambda f.\lambda x.F\, f\, x) = G \circ H$ holds (with x, f not free in F), then $H\,(\mathsf{fix}^{\mathsf{v}} F) = \mathsf{fix}^{\mathsf{v}} G$.

$$\begin{tikzcd} \sigma \to \tau \arrow[r, "\lambda f.\lambda x.F\,f\,x"] \arrow[d, "H"'] & \sigma \to \tau \arrow[d, "H"] \\ \sigma' \to \tau' \arrow[r, "G"'] & \sigma' \to \tau' \end{tikzcd} \quad \Rightarrow \quad H\,(\mathsf{fix}^{\mathsf{v}} F) = \mathsf{fix}^{\mathsf{v}} G$$

Proof. Stability and uniformity imply Filinski's uniformity, because, if $H \circ (\lambda f^{\sigma\to\tau}.\lambda x^{\sigma}.F\, f\, x) = G \circ H$,

$$\begin{array}{rcll} H\,(\mathsf{fix}^{\mathsf{v}}_{\sigma\to\tau}\, F) & = & H\,(\mathsf{fix}^{\mathsf{v}}_{\sigma\to\tau}\,(\lambda f^{\sigma\to\tau}.\lambda x^{\sigma}.F\, f\, x)) & \text{stability} \\ & = & \mathsf{fix}^{\mathsf{v}}_{\sigma'\to\tau'}\, G & \text{uniformity} \end{array}$$

Conversely, Filinski's uniformity implies stability and uniformity. First, for a value $F : (\sigma \to \tau) \to \sigma \to \tau$, we have $\mathsf{id}_{\sigma\to\tau} \circ (\lambda f^{\sigma\to\tau}.\lambda x^{\sigma}.F\, f\, x) = (\lambda f^{\sigma\to\tau}.\lambda x^{\sigma}.F\, f\, x) \circ \mathsf{id}_{\sigma\to\tau}$. Since $\mathsf{id}_{\sigma\to\tau}$ is rigid, by Filinski's uniformity we have the stability $\mathsf{fix}^{\mathsf{v}}_{\sigma\to\tau}\, F \;=\; \mathsf{fix}^{\mathsf{v}}_{\sigma\to\tau}\,(\lambda f^{\sigma\to\tau}.\lambda x^{\sigma}.F\, f\, x)$.
For uniformity, suppose that we have values $F : (\sigma \to \tau) \to \sigma \to \tau$, $G : (\sigma' \to \tau') \to \sigma' \to \tau'$ and a rigid $H : (\sigma \to \tau) \to \sigma' \to \tau'$ such that $H \circ F = G \circ H$ holds. Then, by Lemma 5, we have $H \circ (\lambda f.\lambda x.F\, f\, x) = (\lambda g.\lambda y.G\, g\, y) \circ H$. By applying Filinski's uniformity axiom, we obtain

$$H\,(\mathsf{fix}^{\mathsf{v}}_{\sigma\to\tau}\, F) \;=\; \mathsf{fix}^{\mathsf{v}}_{\sigma'\to\tau'}\,(\lambda g^{\sigma'\to\tau'}.\lambda y^{\sigma'}.G\, g\, y)$$

Since we have already seen that stability follows from Filinski's uniformity, it follows that $\mathsf{fix}^{\mathsf{v}}_{\sigma'\to\tau'}\,(\lambda g^{\sigma'\to\tau'}.\lambda y^{\sigma'}.G\, g\, y) = \mathsf{fix}^{\mathsf{v}}_{\sigma'\to\tau'}\, G$, therefore we have $H\,(\mathsf{fix}^{\mathsf{v}}_{\sigma\to\tau}\, F) \;=\; \mathsf{fix}^{\mathsf{v}}_{\sigma'\to\tau'}\, G$. □

4. Recursion from Iteration

For grasping the rôle of our axioms, it is best to look at the actual construction in the second main result: the correspondence of recursors and iterators in the presence of first-class continuations due to Filinski [5]. So we shall describe this syntactic development before going into the semantic investigation which is the main issue of this paper. In this section we work in a call-by-value $\lambda\mu$-theory, unless otherwise stated.

4.1. Axioms for iteration

As the case of recursion, we introduce a class of functions for determining the uniformity principle for an iterator.

DEFINITION 4 (total value). *In a λ_c-theory, a value $h : \sigma \to \tau$ is called* total *if $h\,v : \tau$ is central for any value $v : \sigma$.*

The word "total" is due to Filinski [5], though in his original definition $h\,v$ is asked to be a *value* rather than a central term.[3]

DEFINITION 5 (uniform iterator). *A type-indexed family of closed values $\mathsf{loop}_\sigma : (\sigma \to \sigma) \to \neg\sigma$ is called a* uniform iterator *if the following conditions are satisfied:*

1. *(iteration) For any value $f : \sigma \to \sigma$,* $\quad \mathsf{loop}_\sigma\, f \;=\; \lambda x^\sigma.\mathsf{loop}_\sigma\, f\,(f\,x)$ *(i.e.* $\mathsf{loop}_\sigma\, f \;=\; (\mathsf{loop}_\sigma\, f) \circ f$*)*

2. *(uniformity) For values $f : \sigma \to \sigma$, $g : \sigma' \to \sigma'$ and $h : \sigma \to \sigma'$, if h is* total *and $h \circ f = g \circ h$ holds, then* $(\mathsf{loop}_{\sigma'}\, g) \circ h = \mathsf{loop}_\sigma\, f$

$$\begin{tikzcd} \sigma \arrow[r, "f"] \arrow[d, "h"'] & \sigma \arrow[d, "h"] \\ \sigma' \arrow[r, "g"'] & \sigma' \end{tikzcd} \qquad \Rightarrow \qquad (\mathsf{loop}\, g) \circ h = \mathsf{loop}\, f$$

If $h \circ f = g \circ h$, we have $(\mathsf{loop}\, g) \circ h = (\mathsf{loop}\, g) \circ g \circ h = (\mathsf{loop}\, g) \circ h \circ f$. So it is natural to expect that $(\mathsf{loop}\, g) \circ h$ behaves in the same way as $\mathsf{loop}\, f$ for "well-behaved" h. The uniformity axiom claims that this is the case when h is total.

It seems that this totality assumption is necessary. For example, let $f = g = \mathsf{id}_\sigma$ and $h = \lambda x^\sigma.\mu\alpha^\sigma.[\beta^\sigma]x$ (an always-jumping function which is not total); then $(\mathsf{loop}\, g) \circ h = \lambda x.[\beta]x$ performs the jump to the label β, while $\mathsf{loop}\, f$ just diverges.

[3] We shall warn that there is yet another use of the word "total" by Filinski [4] where a term is called total when it is discardable (in the sense of Definition 1); see [29] for a detailed analysis on this concept. Another possible source of confusion is that our notion of totality does not correspond to the standard notions of "total relations" or "total maps (in domain theory)". However, in this paper we put our priority on the compatibility with Filinski's development in [5].

Remark 3. Despite of its very limited form, the expressive power of an iterator is not so weak, as we can derive a general *feedback operator*

$$\begin{aligned}\mathsf{feedback}_{\sigma,\tau} &= \lambda f^{\sigma\to\sigma+\tau}.\lambda a^{\sigma}.\mu\beta^{\tau}.\mathsf{loop}\,(\lambda x^{\sigma}.\mu\alpha^{\sigma}.[\alpha,\beta](f\,x))\,a \\ &\qquad : (\sigma\to\sigma+\tau)\to\sigma\to\tau\end{aligned}$$

from an iterator using sums and first-class continuations, which satisfies (with a syntax sugar for sums)

$$\mathsf{feedback}_{\sigma,\tau}\,f\,a \;=\; \mathsf{case}\;f\,a\;\mathsf{of}\;(\mathsf{in}_1\,x^{\sigma}\Rightarrow\mathsf{feedback}_{\sigma,\tau}\,f\,x\mid\mathsf{in}_2\,y^{\tau}\Rightarrow y)$$

for values $f:\sigma\to\sigma+\tau$ and $a:\sigma$.

4.2. The construction

Surprisingly, in the presence of first-class continuations, there is a bijective correspondence between the stable uniform cbv fixpoint operators and the uniform iterators. We recall the construction which is essentially the same as that in [5].

The construction is divided into two parts. For the first part, we introduce a pair of contravariant constructions:

$$\begin{aligned}\mathsf{step}_{\sigma,\tau} &= \lambda F^{\neg\tau\to\neg\sigma}.\lambda x^{\sigma}.\mu\beta^{\tau}.F\,(\lambda y^{\tau}.[\beta]y)\,x \;:\; (\neg\tau\to\neg\sigma)\to\sigma\to\tau\\ \mathsf{pets}_{\sigma,\tau} &= \lambda f^{\sigma\to\tau}.\lambda k^{\neg\tau}.\lambda x^{\sigma}.k\,(f\,x) \;:\; (\sigma\to\tau)\to\neg\tau\to\neg\sigma\end{aligned}$$

Note that here we need first-class continuations to implement $\mathsf{step}_{\sigma,\tau}$ (it has "classical" type). One can easily verify that

LEMMA 6.

- $\mathsf{step}_{\sigma,\tau}\circ\mathsf{pets}_{\sigma,\tau}=\mathsf{id}_{\sigma\to\tau}$ *holds.*
- $\mathsf{pets}_{\sigma,\tau}\circ\mathsf{step}_{\sigma,\tau}=\lambda F^{\neg\tau\to\neg\sigma}.\lambda k^{\neg\tau}.\lambda x^{\sigma}.F\,k\,x$ *holds.*

LEMMA 7.

- *For values* $F:\neg\sigma''\to\neg\sigma'$ *and* $G:\neg\sigma'\to\neg\sigma$, $\mathsf{step}_{\sigma,\sigma''}\,(G\circ F)=(\mathsf{step}_{\sigma',\sigma''}\,F)\circ(\mathsf{step}_{\sigma,\sigma'}\,G)$ *holds if either G is rigid or F is total.*
- $\mathsf{pets}_{\sigma,\sigma''}\,(g\circ f)=(\mathsf{pets}_{\sigma,\sigma'}\,f)\circ(\mathsf{pets}_{\sigma',\sigma''}\,g)$ *holds for values* $f:\sigma\to\sigma'$ *and* $g:\sigma'\to\sigma''$.

The following observation implies that the two notions of uniformity for recursors and iterators are intimately related by this contravariant correspondence.

LEMMA 8. $\mathsf{step}_{\sigma,\tau}(-)$ *and* $\mathsf{pets}_{\sigma,\tau}(-)$ *give a bijective correspondence between rigid functionals of* $\neg\tau \to \neg\sigma$ *and total functions of* $\sigma \to \tau$.

Proof. The only non-trivial part is that $\mathsf{step}_{\sigma,\tau}(-)$ sends a rigid functional to a total function (the other direction and the bijectivity follow immediately from Lemma 4 and Lemma 6). Suppose that $H : \neg\tau \to \neg\sigma$ is rigid. We show that $\mathsf{step}_{\sigma,\tau}\,H = \lambda x^\sigma.\mu\beta^\tau.H\,(\lambda y^\tau.[\beta]y)\,x : \sigma \to \tau$ is total, that is, $\mu\beta^\tau.H\,(\lambda y^\tau.[\beta]y)\,x : \tau$ (with a free variable $x : \sigma$) is central. This can be verified as follows:

$$\begin{array}{ll} & \mathsf{let}\ u\ \mathsf{be}\ \mu\beta.H\,(\lambda y.[\beta]y)\,x\ \mathsf{in}\ \mathsf{let}\ v\ \mathsf{be}\ M\ \mathsf{in}\ N \\ = & \mu\alpha.H\,(\lambda u.[\alpha](\mathsf{let}\ v\ \mathsf{be}\ M\ \mathsf{in}\ N))\,x \\ = & \mu\alpha.H\,(\lambda u.\mathsf{let}\ v\ \mathsf{be}\ M\ \mathsf{in}\ [\alpha]N)\,x \end{array}$$

$$\begin{array}{ll} & \mathsf{let}\ v\ \mathsf{be}\ M\ \mathsf{in}\ \mathsf{let}\ u\ \mathsf{be}\ \mu\beta.H\,(\lambda y.[\beta]y)\,x\ \mathsf{in}\ N \\ = & \mathsf{let}\ v\ \mathsf{be}\ M\ \mathsf{in}\ \mu\alpha.H\,(\lambda u.[\alpha]N)\,x \\ = & \mu\alpha.(\mathsf{let}\ v\ \mathsf{be}\ M\ \mathsf{in}\ H\,(\lambda u.[\alpha]N)\,x) \\ = & \mu\alpha.H\,(\mathsf{let}\ v\ \mathsf{be}\ M\ \mathsf{in}\ \lambda u.[\alpha]N)\,x \end{array}$$

Since H is rigid, it follows that

$$H\,(\lambda u.\mathsf{let}\ v\ \mathsf{be}\ M\ \mathsf{in}\ [\alpha]N)\,x = H\,(\mathsf{let}\ v\ \mathsf{be}\ M\ \mathsf{in}\ \lambda u.[\alpha]N)\,x.$$

□

We are then able to see that, if loop is a uniform iterator, the composition

$$\mathsf{loop}_\sigma \circ \mathsf{step}_{\sigma,\sigma}\ :\ (\neg\sigma \to \neg\sigma) \to \neg\sigma$$

yields a stable uniform fixpoint operator restricted on the negative types $\neg\sigma$. The cbv fixpoint axiom is verified as (by noting the equation $k^{\neg\tau}\,(\mathsf{step}_{\sigma,\tau}\,F^{\neg\tau\to\neg\sigma}\,x^\sigma) = F\,k\,x$)

$$\begin{array}{rcl} (\mathsf{loop}_\sigma \circ \mathsf{step}_{\sigma,\sigma})\,F & = & \mathsf{loop}_\sigma\,(\mathsf{step}_{\sigma,\sigma}\,F) \\ & = & \lambda x^\sigma.\mathsf{loop}_\sigma\,(\mathsf{step}_{\sigma,\sigma}\,F)\,(\mathsf{step}_{\sigma,\sigma}\,F\,x) \\ & = & \lambda x^\sigma.F\,(\mathsf{loop}_\sigma\,(\mathsf{step}_{\sigma,\sigma}\,F))\,x \\ & = & \lambda x^\sigma.F\,((\mathsf{loop}_\sigma \circ \mathsf{step}_{\sigma,\sigma})\,F)\,x \end{array}$$

The stability axiom holds as $\mathsf{step}\,(\lambda f.\lambda x.F\,f\,x) = \mathsf{step}\,F$. The uniformity axiom follows from Lemma 7 and Lemma 8. If $H^{\neg\sigma\to\neg\tau} \circ F^{\neg\sigma\to\neg\sigma} = G^{\neg\tau\to\neg\tau} \circ H^{\neg\sigma\to\neg\tau}$ and H is rigid (hence total by Lemma 4), the first half of Lemma 7 implies $(\mathsf{step}_{\sigma,\sigma}\,F) \circ (\mathsf{step}_{\tau,\sigma}\,H) = (\mathsf{step}_{\tau,\sigma}\,H) \circ (\mathsf{step}_{\tau,\tau}\,G)$. Since $\mathsf{step}_{\tau,\sigma}\,H$ is total by Lemma 8, by the uniformity of loop we have

$$\begin{array}{rcl} \mathsf{loop}_{\tau,\tau}\,(\mathsf{step}_\tau\,G) & = & (\mathsf{loop}_{\sigma,\sigma}\,(\mathsf{step}_\sigma\,F)) \circ (\mathsf{step}_{\tau,\sigma}\,H) \\ & = & \lambda y^\tau.H\,(\mathsf{loop}_\sigma\,(\mathsf{step}_{\sigma,\sigma}\,F))\,y \\ & = & H\,(\mathsf{loop}_\sigma\,(\mathsf{step}_{\sigma,\sigma}\,F)) \end{array}$$

Conversely, if $\mathsf{fix}^{\mathsf{v}}$ is a stable uniform fixpoint operator,

$$\mathsf{fix}^{\mathsf{v}}_{\neg\sigma} \circ \mathsf{pets}_{\sigma,\sigma} \;:\; (\sigma \to \sigma) \to \neg\sigma$$

gives a uniform iterator:

$$\begin{array}{rcl}(\mathsf{fix}^{\mathsf{v}}_{\neg\sigma} \circ \mathsf{pets}_{\sigma,\sigma})\, f & = & \mathsf{fix}^{\mathsf{v}}_{\neg\sigma}\,(\mathsf{pets}_{\sigma,\sigma}\, f) \\ & = & \lambda x^{\sigma}.(\mathsf{pets}_{\sigma,\sigma}\, f)\,(\mathsf{fix}^{\mathsf{v}}_{\neg\sigma}\,(\mathsf{pets}_{\sigma,\sigma}\, f))\, x \\ & = & \lambda x^{\sigma}.(\lambda k^{\neg\sigma}.\lambda y^{\sigma}.k\,(f\, y))\,(\mathsf{fix}^{\mathsf{v}}_{\neg\sigma}\,(\mathsf{pets}_{\sigma,\sigma}\, f))\, x \\ & = & \lambda x^{\sigma}.(\mathsf{fix}^{\mathsf{v}}_{\neg\sigma}\,(\mathsf{pets}_{\sigma,\sigma}\, f))\,(f\, x) \\ & = & \lambda x^{\sigma}.(\mathsf{fix}^{\mathsf{v}}_{\neg\sigma} \circ \mathsf{pets}_{\sigma,\sigma})\, f\,(f\, x)\end{array}$$

Again, the uniformity is a consequence of Lemma 7 and Lemma 8. One direction of the bijectivity of these constructions is guaranteed by the stability axiom (while the other direction follows from $\mathsf{step}_{\sigma,\sigma} \circ \mathsf{pets}_{\sigma,\sigma} = \mathsf{id}_{\sigma\to\sigma}$):

$$\mathsf{fix}^{\mathsf{v}}_{\neg\sigma} \circ \mathsf{pets}_{\sigma,\sigma} \circ \mathsf{step}_{\sigma,\sigma} = \lambda F.\mathsf{fix}^{\mathsf{v}}_{\neg\sigma}(\lambda k.\lambda x.F\, k\, x) = \lambda F.\mathsf{fix}^{\mathsf{v}}_{\neg\sigma}\, F = \mathsf{fix}^{\mathsf{v}}_{\neg\sigma}$$

So we have established

PROPOSITION 2. *There is a bijective correspondence between uniform iterators and stable uniform cbv fixpoint operators restricted on negative types.*

The second part is to reduce fixpoints on an arrow type $\sigma \to \tau$ to those on a negative type $\neg(\sigma \times \neg\tau)$. This is possible because we can implement a pair of isomorphisms between these types (again using first-class continuations):

$$\begin{array}{l}\mathsf{switch}_{\sigma,\tau} = \lambda l^{\neg(\sigma\times\neg\tau)}.\lambda x^{\sigma}.\mu\beta^{\tau}.l\,\langle x, \lambda y^{\tau}.[\beta]y\rangle \;:\; \neg(\sigma\times\neg\tau) \overset{\simeq}{\rightarrow} \sigma \to \tau \\ \mathsf{switch}^{-1}_{\sigma,\tau} = \lambda f^{\sigma\to\tau}.\lambda\langle x^{\sigma}, k^{\neg\tau}\rangle.k(f\, x) \;:\; (\sigma \to \tau) \overset{\simeq}{\rightarrow} \neg(\sigma\times\neg\tau)\end{array}$$

It is routinely seen that both $\mathsf{switch}^{-1}_{\sigma,\tau} \circ \mathsf{switch}_{\sigma,\tau} = \mathsf{id}_{\neg(\sigma\times\neg\tau)}$ and $\mathsf{switch}_{\sigma,\tau} \circ \mathsf{switch}^{-1}_{\sigma,\tau} = \mathsf{id}_{\sigma\to\tau}$ hold. It is also easy to verify (by direct calculation or by applying Proposition 8 in Section 7) that

LEMMA 9. $\mathsf{switch}_{\sigma,\tau}$ *and* $\mathsf{switch}^{-1}_{\sigma,\tau}$ *are rigid.*

By applying the uniformity axiom to the trivial equation $\mathsf{switch}_{\sigma,\tau} \circ (\mathsf{switch}^{-1}_{\sigma,\tau} \circ F \circ \mathsf{switch}_{\sigma,\tau}) = F \circ \mathsf{switch}_{\sigma,\tau}$ we have

$$\mathsf{fix}^{\mathsf{v}}_{\sigma\to\tau}\, F \;=\; \mathsf{switch}_{\sigma,\tau}\,(\mathsf{fix}^{\mathsf{v}}_{\neg(\sigma\times\neg\tau)}\,(\mathsf{switch}^{-1}_{\sigma,\tau} \circ F \circ \mathsf{switch}_{\sigma,\tau}))$$

PROPOSITION 3. *There is a bijective correspondence between stable uniform cbv fixpoint operators restricted on negative types and those on general function types.*

Proof. From a stable uniform cbv fixpoint operator restricted on negative types, one can define that on general function types by taking the equation above as definition; because of the uniformity, this in fact is the unique possibility of extending the operator to that on all function types. The only nontrivial point is that the uniformity axiom on this defined fixpoint operator on general function typed can be derived from the uniformity axiom on the fixpoint operator on negative types, which we shall spell out below. Suppose that we have values $F : (\sigma \to \tau) \to \sigma \to \tau$, $G : (\sigma' \to \tau') \to \sigma' \to \tau'$ and a rigid $H : (\sigma \to \tau) \to \sigma' \to \tau'$ such that $H \circ F = G \circ H$ holds. Since rigid functionals are closed under composition (Lemma 3) and switch and switch^{-1} are rigid (Lemma 9), $\mathsf{switch}^{-1} \circ H \circ \mathsf{switch}$ is also rigid. By applying the uniformity axiom (on negative types) to the equation

$$\begin{aligned}(\mathsf{switch}^{-1}_{\sigma',\tau'} \circ H \circ \mathsf{switch}_{\sigma,\tau}) \circ (\mathsf{switch}^{-1}_{\sigma,\tau} \circ F \circ \mathsf{switch}_{\sigma,\tau}) = \\ (\mathsf{switch}^{-1}_{\sigma',\tau'} \circ G \circ \mathsf{switch}_{\sigma',\tau'}) \circ (\mathsf{switch}^{-1}_{\sigma',\tau'} \circ H \circ \mathsf{switch}_{\sigma,\tau})\end{aligned}$$

we obtain

$$\begin{aligned}\mathsf{fix}^{\mathsf{v}}_{\neg(\sigma' \times \neg\tau')}\, (\mathsf{switch}^{-1}_{\sigma',\tau'} \circ G \circ \mathsf{switch}_{\sigma',\tau'}) = \\ \mathsf{switch}^{-1}_{\sigma',\tau'}(H(\mathsf{switch}_{\sigma,\tau}(\mathsf{fix}^{\mathsf{v}}_{\neg(\sigma \times \neg\tau)}\, (\mathsf{switch}^{-1}_{\sigma,\tau} \circ F \circ \mathsf{switch}_{\sigma,\tau}))))\end{aligned}$$

which implies (by applying $\mathsf{switch}_{\sigma',\tau'}$ to both sides of the equation) $\mathsf{fix}^{\mathsf{v}}_{\sigma' \to \tau'}\, G = H\, (\mathsf{fix}^{\mathsf{v}}_{\sigma \to \tau}\, F)$. □

In summary, we conclude that, in the presence of first-class continuations, stable uniform cbv fixpoint operators are precisely those derived from uniform iterators, and vice versa:

THEOREM 1. *There is a bijective correspondence between uniform iterators and stable uniform cbv fixpoint operators.*

$$\begin{aligned}\mathsf{fix}^{\mathsf{v}}_{\sigma \to \tau}\, F \;&=\; \mathsf{switch}_{\sigma,\tau}\, (\mathsf{loop}_{\sigma \times \neg\tau}\, (\mathsf{step}_{\sigma \times \neg\tau, \sigma \times \neg\tau}\, (\mathsf{switch}^{-1}_{\sigma,\tau} \circ F \circ \mathsf{switch}_{\sigma,\tau}))) \\ \mathsf{loop}_{\sigma}\, f \;&=\; \mathsf{fix}^{\mathsf{v}}_{\neg\sigma}\, (\mathsf{pets}_{\sigma,\sigma}\, f)\end{aligned}$$

Sample code written in SML/NJ [17, 11] is found in Figure 3.

```
(* an empty type "bot" with an initial map A : bot -> 'a *)
datatype bot = VOID of bot;
fun A (VOID v) = A v;
(* the C operator, C : (('a -> bot) -> bot) -> 'a *)
fun C f =
 SMLofNJ.Cont.callcc
   (fn k => A (f (fn x => (SMLofNJ.Cont.throw k x) : bot)));

(* basic combinators *)
fun step F x = C (fn k => F k x);
fun pets f k x = k (f x) : bot;
fun switch l x = C (fn q => l (x,q));
fun switch_inv f (x, k) = k (f x) : bot;
(* step : (('a -> bot) -> 'b -> bot) -> 'b -> 'a
   pets : ('a -> 'b) -> ('b -> bot) -> 'a -> bot
   switch : ('a * ('b -> bot) -> bot) -> 'a -> 'b
   switch_inv : ('a -> 'b) -> 'a * ('b -> bot) -> bot *)

(* an iterator, loop : ('a -> 'a) -> 'a -> bot *)
fun loop f x = loop f (f x) : bot;

(* recursion from iteration *)
fun fix F = switch (loop (step (switch_inv o F o switch)));
(* fix : (('a -> 'b) -> 'a -> 'b) -> 'a -> 'b *)
```

Figure 3. Coding in SML/NJ (versions based on SML '97 [17])

5. Categorical Semantics

The rest of this paper is devoted to investigating the semantic counterpart of our stable uniform cbv fixpoint operators and for giving our two main results in a coherent way. In this section we recall some preliminaries on the underlying categorical structures which will be used in our semantic development.

5.1. Models of the λ_c-calculus

Let $\mathbb{C}$ be a category with finite products and a strong monad $T = (T, \eta, \mu, \theta)$, where η and μ are the unit and multiplication of the monad T, and θ is the tensorial strength with respect to the finite products of $\mathbb{C}$ (see e.g. [18, 19] for these category-theoretic concepts). We write $\mathbb{C}_T$ for the Kleisli category of T, and $J : \mathbb{C} \to \mathbb{C}_T$ for the associated left adjoint functor; explicitly, J is the identity on objects and sends $f \in \mathbb{C}(X, Y)$ to $\eta_Y \circ f \in \mathbb{C}_T(X, Y) = \mathbb{C}(X, TY)$. We assume that $\mathbb{C}$ has Kleisli exponentials, i.e., for every X in $\mathbb{C}$ the functor $J((-) \times X) : \mathbb{C} \to \mathbb{C}_T$

has a right adjoint $X \Rightarrow (-) : \mathbb{C}_T \to \mathbb{C}$. This gives the structure for modelling computational lambda calculus [18]. Specifically, we fix an object $[\![b]\!]$ for each base type b and define the interpretation of types as $[\![\sigma \to \tau]\!] = [\![\sigma]\!] \Rightarrow [\![\tau]\!]$, $[\![\sigma \times \tau]\!] = [\![\sigma]\!] \times [\![\tau]\!]$ and $[\![\top]\!] = 1$. A well-typed term $\Gamma \vdash M : \sigma$ is interpreted inductively as a morphism of $\mathbb{C}_T([\![\Gamma]\!], [\![\sigma]\!]) = \mathbb{C}([\![\Gamma]\!], T[\![\sigma]\!])$ (where $[\![\Gamma]\!] = [\![\sigma_1]\!] \times \ldots \times [\![\sigma_n]\!]$ for $\Gamma = x_1 : \sigma_1, \ldots, x_n : \sigma_n$), once we fix the interpretations of constants; see Appendix A for a summary. Following Moggi, we call such a structure a *computational model.*

PROPOSITION 4. *[18] The computational models provide a sound and complete class of models of the computational lambda calculus.*

In fact, we can use the λ_c-calculus as an internal language of a computational model – up to the choice of the base category $\mathbb{C}$ (which may correspond to either syntactically defined values, or more semantic values like thunkable terms, or even something between them; see [10] for a detailed consideration on this issue) – in a similar sense that the simply typed lambda calculus is used as an internal language of a cartesian closed category [16].

5.2. Models of the call-by-value $\lambda\mu$-calculus

Let $\mathbb{C}$ be a distributive category, i.e., a category with finite products and coproducts so that $(-) \times A : \mathbb{C} \to \mathbb{C}$ preserves finite coproducts for each A. We call an object R a *response object* if there exists an exponential R^A for each A, i.e., $\mathbb{C}(- \times A, R) \simeq \mathbb{C}(-, R^A)$ holds. Given such a structure, we can model the cbv $\lambda\mu$-calculus in the Kleisli category $\mathbb{C}_T$ of the strong monad $T = R^{R^{(-)}}$ [25]. A term $\Gamma \vdash M : \sigma \mid \Delta$ is interpreted as a morphism of $\mathbb{C}_T([\![\Gamma]\!], [\![\sigma]\!] + [\![\Delta]\!])$ (where $[\![\Delta]\!] = [\![\tau_1]\!] + \ldots + [\![\tau_n]\!]$ for $\Delta = \alpha_1 : \tau_1, \ldots, \alpha_n : \tau_n$). The interpretation is in fact a typed version of the call-by-value CPS transformation [21, 25], as sketched in Appendix B. Following Selinger, we call $\mathbb{C}$ a *response category* and the Kleisli category $\mathbb{C}_T$ a *category of continuations* and write $R^{\mathbb{C}}$ for $\mathbb{C}_T$ (though in [25] a category of continuations means the opposite of $R^{\mathbb{C}}$).

PROPOSITION 5. *[25] The categories of continuations provide a sound and complete class of models of the cbv $\lambda\mu$-calculus.*

As the case of the λ_c-calculus, we can use the cbv $\lambda\mu$-calculus as an internal language of a category of continuations [25].

5.3. Centre and focus

We have already seen the notion of centre and focus in the λ_c-calculus and the cbv $\lambda\mu$-calculus in a syntactic form (Definition 1). However, these concepts originally arose from the analysis on the category-theoretic models given as above. Following the discovery of the *premonoidal structure* on the Kleisli category part $\mathbb{C}_T$ ($R^{\mathbb{C}}$) of these models [22], Thielecke [28] proposed a *direct* axiomatization of $R^{\mathbb{C}}$ not depending on the base category $\mathbb{C}$ (which may be seen as a chosen category of "values") but on the subcategory of "effect-free" morphisms of $R^{\mathbb{C}}$, which is the *focus* (equivalently *centre*) of $R^{\mathbb{C}}$. Führmann [10] carries out further study on models of the λ_c-calculus along this line.

DEFINITION 6 (centre, semantic definition). *Given a computational model with the base category $\mathbb{C}$ and the strong monad T, an arrow $f : X \to Y$ in $\mathbb{C}_T$ is called* central *if, for any $g : X' \to Y'$ in $\mathbb{C}_T$, the compositions $(Y \times g) \circ (f \times X')$ and $(f \times Y') \circ (X \times g)$ agree.*[4]

Note that the products are not necessarily bifunctorial on $\mathbb{C}_T$; they form premonoidal products in the sense of [22] (the reader familiar with this notion might prefer to use $\otimes$ instead of $\times$ for indicating that they are not cartesian products). This notion of centrality amounts to the semantic version of centrality in Definition 1.

In this paper we do not go into the further details of these semantic analyses. However, we will soon see that these concepts naturally arise in our analysis of the uniformity principles for recursors and iterators. In particular, a total value $h : \sigma \to \tau$ (equivalently the term $x : \sigma \vdash h\,x : \tau$) precisely corresponds to the central morphisms in the semantic models. In the case of the models of the cbv $\lambda\mu$-calculus, the centre can be characterised in terms of the category of algebras, for which our uniformity principles are defined; that is, we have

PROPOSITION 6. *$f \in R^{\mathbb{C}}(A, B) \simeq \mathbb{C}(R^B, R^A)$ is central if and only if its counterpart in $\mathbb{C}$ is an algebra morphism from the algebra (R^B, R^{η_B}) to (R^A, R^{η_A}).*

We discuss more about this in Section 7; there this observation turns out to be essential in relating the uniformity principles for recursion and iteration in the cbv $\lambda\mu$-theories. We note that this result has been observed in various forms in [28, 25, 10].

[4] In terms of $\mathbb{C}$, $f \in \mathbb{C}_T(X, Y) = \mathbb{C}(X, TY)$ is central if $\varphi_{X',Y'} \circ (f \times g) = \varphi'_{X',Y'} \circ (f \times g)$ holds for any $g \in \mathbb{C}_T(X', Y') = \mathbb{C}(X', TY')$ where φ and φ' are the left-first and right-first pairings (Appendix A).

6. Uniform T-Fixpoint Operators

In this section we shall consider a computational model with the base category $\mathbb{C}$ and a strong monad T.

6.1. Uniform T-Fixpoint Operators

We first recall the notion of *uniform T-fixpoint operator* of Simpson and Plotkin [27], which arose from considerations on fixpoint operators in Axiomatic Domain Theory (ADT) [7, 26]. In ADT, we typically start with a category $\mathbb{C}$ of *predomains*, for example the category of ω-complete partial orders (possibly without bottom) and continuous functions. Then we consider the *lifting monad* T on $\mathbb{C}$, which adds a bottom element to ω-cpo's. Then objects of the form TX are pointed cpo's (ω-cpo's with bottom), on which we have the least fixpoint operator. It is also easily checked that such a pointed cpo has a unique T-algebra structure (in fact any T-algebra arises in this way in this setting, though we will soon see that this is not the case if we take a continuation monad as T), and an algebra morphism is precisely the bottom-preserving maps, i.e., the strict ones. As is well known, the least fixpoint operator enjoys the uniformity principle with respect to such strict maps. By abstracting this situation we have:

DEFINITION 7 (uniform T-fixpoint operator [27]). *A* T-fixpoint operator on $\mathbb{C}$ *is a family of functions*

$$(-)^* : \mathbb{C}(TX, TX) \to \mathbb{C}(1, TX)$$

such that, for any $f : TX \to TX$, $f \circ f^* = f^*$ holds. It is called uniform *if, for any $f : TX \to TX$, $g : TY \to TY$ and $h : TX \to TY$, $h \circ \mu = \mu \circ Th$ and $g \circ h = h \circ f$ imply $g^* = h \circ f^*$.*

$$\begin{tikzcd} TTX \arrow[r, "\mu"] \arrow[d, "Th"'] & TX \arrow[d, "h"] \\ TTY \arrow[r, "\mu"'] & TY \end{tikzcd} \qquad \begin{tikzcd} TX \arrow[r, "f"] \arrow[d, "h"'] & TX \arrow[d, "h"] \\ TY \arrow[r, "g"'] & TY \end{tikzcd} \quad \Rightarrow \quad g^* = h \circ f^*$$

Thus a T-fixpoint operator is given as a fixpoint operator restricted on the objects of the form TX. One may easily check that, in the domain-theoretic example sketched as above, the condition $h \circ \mu = \mu \circ Th$ says that h is a strict map.

This limited form of fixpoint operators, however, turns out to be sufficient to model a call-by-value fixpoint operator. To see this, suppose that we are given an object A with a T-algebra structure $\alpha : TA \to A$ (that is, we ask $\alpha \circ \eta = id$ and $\alpha \circ \mu = \alpha \circ T\alpha$ hold). Given $f : A \to A$, we have $\alpha \circ (\eta \circ f \circ \alpha)^* : 1 \to A$ and

$$\begin{array}{rcl} \alpha \circ (\eta \circ f \circ \alpha)^* & = & \alpha \circ \eta \circ f \circ \alpha \circ (\eta \circ f \circ \alpha)^* \\ & = & f \circ \alpha \circ (\eta \circ f \circ \alpha)^* \end{array}$$

Therefore we can extend a T-fixpoint operator $(-)^*$ to be a fixpoint operator $(-)^*$ on objects with T-algebra structure by defining $f^* = \alpha \circ (\eta \circ f \circ \alpha)^*$.

Moreover, given a *uniform* T-fixpoint operator $(-)^*$, it is easy to see that this extended fixpoint operator $(-)^*$ on T-algebras is *uniform* in the following sense: for T-algebras $(A, \alpha : TA \to A)$ and $(B, \beta : TB \to B)$, if $f : A \to A$, $g : B \to B$ and $h : A \to B$ satisfy $h \circ \alpha = \beta \circ Th$ (i.e., h is a T-algebra morphism) and $g \circ h = h \circ f$, then $g^* = h \circ f^*$.

$$\begin{tikzcd} TA \arrow[r, "\alpha"] \arrow[d, "Th"'] & A \arrow[d, "h"] \\ TB \arrow[r, "\beta"'] & B \end{tikzcd} \qquad \begin{tikzcd} A \arrow[r, "f"] \arrow[d, "h"'] & A \arrow[d, "h"] \\ B \arrow[r, "g"'] & B \end{tikzcd} \quad \Rightarrow \quad g^* = h \circ f^*$$

Furthermore, such a uniform extension is unique: given a uniform fixpoint operator $(-)^*$ on objects with T-alpgebra structure, by applying this uniformity to $\alpha \circ (\eta \circ f \circ \alpha) = f \circ \alpha$ for a T-algebra (A, α) and $f : A \to A$, we obtain $f^* = \alpha \circ (\eta \circ f \circ \alpha)^*$, hence $(-)^*$ is completely determined by its restriction on free algebras (TX, μ_X), i.e., a uniform T-fixpoint operator.

In particular, Kleisli exponentials $X \Rightarrow Y$ fit in this scheme, where the T-algebra structure $\alpha_{X,Y} : T(X \Rightarrow Y) \to X \Rightarrow Y$ is given as the adjoint mate (currying) of

$$T(X \Rightarrow Y) \times X \stackrel{\theta'}{\to} T((X \Rightarrow Y) \times X) \stackrel{T\mathsf{ev}}{\to} T^2Y \stackrel{\mu}{\to} TY$$

(see Appendix A for notations). Since we interpret a function type as a Kleisli exponential, this fact enables us to use a uniform T-fixpoint operator for dealing with a fixpoint operator on function types.

We note that $\eta \circ \alpha_{X,Y} : T(X \Rightarrow Y) \to T(X \Rightarrow Y)$ corresponds to an eta-expansion in the λ_c-calculus. That is, if a term $\Gamma \vdash M : X \to Y$ represents an arrow $f : A \to T(X \Rightarrow Y)$ in $\mathbb{C}$, then $\Gamma \vdash \lambda x^X.M\,x : X \to Y$ represents $\eta \circ \alpha_{X,Y} \circ f : A \to T(X \Rightarrow Y)$.

LEMMA 10. *For any* $\Gamma \vdash M : \sigma \to \tau$,

$$[\![\Gamma \vdash \lambda x^\sigma.M\,x : \sigma \to \tau]\!] = \eta \circ \alpha_{[\![\sigma]\!],[\![\tau]\!]} \circ [\![\Gamma \vdash M : \sigma \to \tau]\!]$$

holds.

Proof.

$$\begin{array}{rl} & [\![\Gamma \vdash \lambda x^\sigma.M\,x : \sigma \to \tau]\!] \\ = & \eta \circ \mathsf{cur}([\![\Gamma, x : \sigma \vdash M\,x : \tau]\!]) \\ = & \eta \circ \mathsf{cur}(\mu \circ T\mathsf{ev} \circ \varphi \circ \langle [\![\Gamma, x : \sigma \vdash M : \sigma \to \tau]\!], [\![\Gamma, x : \sigma \vdash x : \sigma]\!] \rangle) \\ = & \eta \circ \mathsf{cur}(\mu \circ T\mathsf{ev} \circ \varphi \circ ([\![\Gamma \vdash M : \sigma \to \tau]\!] \times \eta)) \\ = & \eta \circ \mathsf{cur}(\mu \circ T\mathsf{ev} \circ \theta' \circ ([\![\Gamma \vdash M : \sigma \to \tau]\!] \times id)) \\ = & \eta \circ \mathsf{cur}(\mu \circ T\mathsf{ev} \circ \theta') \circ [\![\Gamma \vdash M : \sigma \to \tau]\!] \end{array}$$ □

This observation is frequently used in distilling the axioms of the stable uniform cbv fixpoint operators below.

6.2. Axiomatization in the λ_c-calculus

Using the λ_c-calculus as an internal language of $\mathbb{C}_T$, the equation $f^* = f \circ f^*$ on $X \Rightarrow Y$ can be represented as

$$F^* \;=\; \lambda x^X.F\,F^*\,x \quad \text{where } F = \lambda f^{X \to Y}.\lambda x^X.F\,f\,x : (X \to Y) \to X \to Y$$

The side condition $F = \lambda f^{X \to Y}.\lambda x^X.F\,f\,x$ means that F corresponds to an arrow in $\mathbb{C}(X \Rightarrow Y, X \Rightarrow Y)$, not $\mathbb{C}_T(X \Rightarrow Y, X \Rightarrow Y)$. However, the operator $(-)^* : \mathbb{C}(X \Rightarrow Y, X \Rightarrow Y) \to \mathbb{C}(1, X \Rightarrow Y)$ can be equivalently axiomatized by a slightly different operator

$$(-)^\ddagger : \mathbb{C}(X \Rightarrow Y, T(X \Rightarrow Y)) \to \mathbb{C}(1, X \Rightarrow Y)$$

subject to $f^\ddagger = \alpha_{X,Y} \circ f \circ f^\ddagger$, with an additional condition $f^\ddagger = (\eta \circ \alpha_{X,Y} \circ f)^\ddagger$. In fact, we can define such a $(-)^\ddagger$ as $(\alpha_{X,Y} \circ (-))^*$ and conversely $(-)^*$ by $(\eta \circ (-))^\ddagger$, and it is easy to see that these are in bijective correspondence. The condition $f^\ddagger = \alpha_{X,Y} \circ f \circ f^\ddagger$, equivalently $\eta \circ f^\ddagger = \eta \circ \alpha_{X,Y} \circ f \circ f^\ddagger$, is axiomatized in the λ_c-calculus as (by recalling that $\eta \circ \alpha_{X,Y} \circ (-)$ gives an eta-expansion)

$$F^\ddagger \;=\; \lambda x.F\,F^\ddagger\,x \text{ for any value } F : (X \to Y) \to X \to Y$$

which is precisely the cbv fixpoint axiom. The additional condition $f^\ddagger = (\eta \circ \alpha_{X,Y} \circ f)^\ddagger$ is axiomatized as

$$F^\ddagger = (\lambda f.\lambda x.F\,f\,x)^\ddagger \quad \text{where } F \text{ is a value}$$

This is no other than the stability axiom. We thus obtain the first two axioms of our stable uniform cbv fixpoint operators, which are precisely modelled by T-fixpoint operators.

6.3. Uniformity axiom

Next, we shall see how the uniformity condition on T-fixpoint operators can be represented in the λ_c-calculus. Following the previous discussions, we consider $H \in \mathbb{C}(X \Rightarrow Y, X' \Rightarrow Y')$ as "strict" if it is an algebra morphism from $(X \Rightarrow Y, \alpha_{X,Y})$ to $(X' \Rightarrow Y', \alpha_{X',Y'})$.[5] Spelling out this condition, we ask H to satisfy $H \circ \alpha_{X,Y} = \alpha_{X',Y'} \circ T(H)$, equivalently $T(H) \circ \eta \circ \alpha_{X,Y} = \eta \circ \alpha_{X',Y'} \circ T(H)$. In terms of the λ_c-calculus, this means that an eta-expansion commutes with the application of H; therefore, in the λ_c-calculus, we ask $H : (X \to Y) \to X' \to Y'$ to be a value such that

$$H(\lambda x^X.M\,x) \;=\; \lambda y^{X'}.H\,M\,y \;:\; X' \to Y'$$

holds for any $M : X \to Y$. We have called such an H rigid, and defined the uniformity condition with respect to such rigid functionals.

Remark 4. Actually the uniformity condition obtained by the argument above is as follows, which is slightly weaker than stated in Definition 3:

> For values $F : (\sigma \to \tau) \to \sigma \to \tau$, $G : (\sigma' \to \tau') \to \sigma' \to \tau'$ and a rigid $H : (\sigma \to \tau) \to \sigma' \to \tau'$, if $H \circ (\lambda f.\lambda x.F\,f\,x) = (\lambda g.\lambda y.G\,g\,y) \circ H$ holds, then $H\,(\mathsf{fix}^{\mathsf{v}} F) = \mathsf{fix}^{\mathsf{v}} G$.

$$\begin{tikzcd}
\sigma \to \tau \arrow[r, "\lambda f.\lambda x.F\,f\,x"] \arrow[d, "H"'] & \sigma \to \tau \arrow[d, "H"] \\
\sigma' \to \tau' \arrow[r, "\lambda g.\lambda y.G\,g\,y"'] & \sigma' \to \tau'
\end{tikzcd} \qquad \Rightarrow \qquad H\,(\mathsf{fix}^{\mathsf{v}} F) = \mathsf{fix}^{\mathsf{v}} G$$

However, thanks to Lemma 5, we can justify the uniformity axiom in Definition 3.

6.4. Soundness and completeness

Now we give one of the main result of this paper.

THEOREM 2. *The computational models with a uniform T-fixpoint operator provide a sound and complete class of models of the computational lambda calculus with a stable uniform call-by-value fixpoint operator.*

[5] A characterisation of rigid functions (on computation types) in the same spirit is given in Filinski's thesis [6] (Section 2.2.2) though unrelated to the uniformity of fixpoint operators.

This extends Proposition 4 with the stable uniform call-by-value fixpoint operator and uniform T-fixpoint operators. Most part of soundness follows from a routine calculation. However, the interpretation of the stable uniform call-by-value fixpoint operator and the verification of the axioms do require some care: we need to consider a parameterized fixpoint operator (with parameterized uniformity) for interpreting the free variables. Thus we have to parameterize the considerations in Section 6.1. This can be done along the line of Simpson's work [26]. Below we outline the constructions and results needed for our purpose.

PROPOSITION 7. *A uniform T-fixpoint operator uniquely extends to a family of functions*

$$(-)^*_{X,A,\alpha} : \mathbb{C}(X \times A, A) \to \mathbb{C}(X, A)$$

where X ranges over objects of $\mathbb{C}$ and $(A, \alpha : TA \to A)$ over T-algebras, such that

1. *(parameterized fixpoint) For $f : X \times A \to A$ with a T-algebra (A, α), $f^*_{X,A,\alpha} = f \circ \langle id_X, f^*_{X,A,\alpha}\rangle : X \to A$ holds*

2. *(parameterized uniformity) For $f : X \times A \to A$, $g : X \times B \to B$, $h : X \times A \to B$ with T-algebras (A, α) and (B, β), $h \circ (id_X \times \alpha) = \beta \circ Th \circ \theta_{X,A}$ and $h \circ \langle \pi_{X,A}.f\rangle = g \circ \langle \pi_{X,A}, h\rangle$ imply $g^*_{X,B,\beta} = h \circ \langle id_X, f^*_{X,A,\alpha}\rangle$*

$$\begin{tikzcd}
X \times TA \arrow[r, "id_X \times \alpha"] \arrow[d, "Th \circ \theta_{X,A}"'] & X \times A \arrow[d, "h"] & X \times A \arrow[r, "\langle \pi_{X,A}, f\rangle"] \arrow[d, "\langle \pi_{X,A}, h\rangle"'] & X \times A \arrow[d, "h"] \\
TB \arrow[r, "\beta"'] & B & X \times B \arrow[r, "g"'] & B
\end{tikzcd}$$

$$\Downarrow$$

$$g^*_{X,B,\beta} = h \circ \langle id_X, f^*_{X,A,\alpha}\rangle$$

Here we only give the construction of $f^*_{X,A,\alpha} : X \to A$ from $f : X \times A \to A$ and omit the proof (which largely consists of lengthy diagram chasings and we shall leave it for interested readers – see also [26]). Let $\rho_{X,A,\alpha} = \alpha \circ \mathsf{ev} \circ (\alpha_{X,A} \times id_X) : T(X \Rightarrow A) \times X \to A$ (recall that $\alpha_{X,A} : T(X \Rightarrow A) \to X \Rightarrow A$ is the T-algebra structure on $X \Rightarrow A$). Using ρ, we define $\lceil f\rceil : T(X \Rightarrow A) \to T(X \Rightarrow A)$ by

$$\lceil f\rceil = \eta_{X\Rightarrow A} \circ \mathsf{cur}(\eta_A \circ f \circ \langle \pi', \rho\rangle)$$

Finally we have $f^*_{X,A,\alpha} = \rho \circ \langle \lceil f \rceil^* \circ !_X, id_X \rangle : X \to A$. By trivialising the parameterization and by considering just the free algebras, one can recover the original uniform T-fixpoint operator. The uniqueness of the extension follows from the uniformity (essentially in the same way as described in Section 6.1).

Using this parametrically uniform parameterized fixpoint operator, now it is not hard to interpret a stable uniform call-by-value fixpoint operator in a computational model with a uniform T-fixpoint operator and check that all axioms are validated.

Completeness is shown by constructing a term model, for which there is no difficulty. Since the uniform T-fixpoint operator on this term model is directly defined by the stable uniform call-by-value fixpoint operator on the types $T\sigma = \top \to \sigma$, and also because we have already observed that rigid functionals are characterized as the algebra morphisms in this model, this part is truly routine.

7. Recursion from Iteration Revisited

7.1. Iteration in the category of continuations

Let $\mathbb{C}$ be a response category with a response object R. An *iterator* on the category of continuations $R^{\mathbb{C}}$ is a family of functions $(-)_* : R^{\mathbb{C}}(A, A) \to R^{\mathbb{C}}(A, 0)$ so that $f_* = f_* \circ f$ holds for $f \in R^{\mathbb{C}}(A, A)$. Spelling out this definition in $\mathbb{C}$, to give an iterator on $R^{\mathbb{C}}$ is to give a family of functions $(-)^* : \mathbb{C}(R^A, R^A) \to \mathbb{C}(1, R^A)$ so that $f^* = f \circ f^*$ holds for $f \in \mathbb{C}(R^A, R^A)$. Thus an iterator on $R^{\mathbb{C}}$ (hence in the cbv $\lambda\mu$-calculus) is no other than a fixpoint operator on $\mathbb{C}$ (hence the target call-by-name calculus) restricted on objects of the form R^A ("negative objects").

Example 3. We give a simple-minded model of the cbv $\lambda\mu$-calculus with an iterator. Let $\mathbb{C}$ be the category of ω-cpo's (possibly without bottom) and continuous maps, and let R be an ω-cpo with bottom. Since $\mathbb{C}$ is a cartesian closed category with finite coproducts, it serves as a response category with the response object R. Moreover there is a least fixpoint operator on the negative objects R^A because R^A has a bottom element, thus we have an iterator on $R^{\mathbb{C}}$ (which in fact is a unique uniform iterator in the sense below).

Remark 5. A careful reader may notice that we actually need a parameterized version of the iterator for interpreting free variables as well as free names: $(-)_*$ should be defined as a function from $R^{\mathbb{C}}(X \times A, A + Y)$ to $R^{\mathbb{C}}(X \times A, Y)$. However, this parameterization, including that on uniformity discussed below, can be done in the same way as in the previous section (and is much easier); a uniform iterator uniquely extends to a parametrically uniform parameterized iterator – we leave the detail to the interested reader.

7.2. Relation to uniform T-fixpoint operators

For any object A, the negative object R^A canonically has a T-algebra structure

$$\alpha_A = R^{\eta_A} = \lambda m^{R^{R^{R^A}}}.\lambda x^A.m\,(\lambda f^{R^A}.f\,x) : R^{R^{R^A}} \to R^A$$

for the monad $T = R^{R^{(-)}}$. Thus the consideration on the uniform T-fixpoint operators applies to this setting: if this computational model has a uniform T-fixpoint operator, then we have a fixpoint operator on negative objects, hence we can model an iterator of the cbv $\lambda\mu$-calculus in the category of continuations.

Conversely, if we have an iterator on $R^{\mathbb{C}}$, then it corresponds to a fixpoint operator on negative objects in $\mathbb{C}$, which of course include objects of the form $TA = R^{R^A}$. Therefore we obtain a T-fixpoint operator. It is then natural to expect that (along the consideration in Section 6.1), if the iterator satisfies a suitable uniformity condition, then it bijectively corresponds to a uniform T-fixpoint operator. This uniformity condition on an iterator must be determined again with respect to algebra morphisms. So we regard $h \in R^{\mathbb{C}}(A,B) \simeq \mathbb{C}(R^B, R^A)$ as "strict" when its counterpart in $\mathbb{C}(R^B, R^A)$ is an algebra morphism from (R^B, α_B) to (R^A, α_A), i.e., $h \circ \alpha_B = \alpha_A \circ R^{R^h}$ holds in $\mathbb{C}$. We say that an iterator $(-)_*$ on $R^{\mathbb{C}}$ is *uniform* if $f_* = g_* \circ h$ holds for $f : A \to A$, $g : B \to B$ and "strict" $h : A \to B$ such that $h \circ f = g \circ h$.

THEOREM 3. *Given a response category $\mathbb{C}$ with a response object R, to give a uniform $R^{R^{(-)}}$-fixpoint operator on $\mathbb{C}$ is to give a uniform iterator on $R^{\mathbb{C}}$.*

Proof. Immediate, since a uniform $R^{R^{(-)}}$-fixpoint operator uniquely extends to a uniform fixpoint operator on negative objects (hence a uniform iterator) – the uniqueness of the extension follows from the uniformity (by the same argument as given in Section 6.1). □

Fortunately, the condition to be an algebra morphism is naturally represented in a cbv $\lambda\mu$-theory. A value $h : A \to B$ represents an algebra morphism if and only if

$$x : A \vdash \mathsf{let}\ y^B\ \mathsf{be}\ h\,x\ \mathsf{in}\ \lambda u^\top.y \ = \ \lambda u^\top.h\,x \ : \ \top \to B$$

holds – in fact, the CPS transformation (see Appendix B) of this equation is no other than the equation $h \circ \alpha_B = \alpha_A \circ R^{R^h}$. By Lemma 1, in a cbv $\lambda\mu$-theory, this requirement is equivalent to saying that $h\,x$ is a central term for each value x (this also implies Proposition 6 in Section 5), hence h is total. Therefore we obtain the uniformity condition for an iterator in Section 4. This is remarkable, as it says that the idea of defining the uniformity principle of fixpoint operators with respect to algebra morphisms (from ADT) and the idea of defining the uniformity principle of iterators with respect to effect-free morphisms (from Filinski's work) *coincide* in the presence of first-class continuations, despite their very different origins; technically, this is the substance of the left-to-right implication of Proposition 6. In summary, we have semantically shown Theorem 1:

THEOREM 4 (Theorem 1 restated). *In a cbv $\lambda\mu$-theory, there is a bijective correspondence between the stable uniform cbv fixpoint operators and the uniform iterators.*

In a sense, the syntactic proof in Section 4 gives an example of *direct style* reasoning, whereas this semantic proof provides a *continuation-passing style* reasoning on the same result. We can choose either stable uniform fixpoint operators (in syntactic, direct style) or uniform T-fixpoint operators (in semantic, monadic or continuation-passing style) as the tool for reasoning about recursion in call-by-value setting; they are as good as the other (thanks to Theorem 2).

7.3. On Filinski's uniformity

In [5] Filinski introduced uniformity principles for both cbv fixpoint operators and iterators, for establishing a bijective correspondence between them. While his definitions turn out to be sufficient for his purpose, in retrospect they seem to be somewhat ad hoc and are strictly weaker than our uniformity principles. Here we give a brief comparison.

First, Filinski calls a value $h : \sigma \to \tau$ "total" when $h\,v$ is a *value* for each value $v : \sigma$. However, while a value is always central, the converse is not true. Note that, while the notion of centre is uniquely determined for each cbv $\lambda\mu$-theory (and category of continuations), the notion of value is not canonically determined (a category of continuations can arise from different response categories [25]). Since the uniformity principle is determined not in terms of the base category $\mathbb{C}$ but in terms of the category of algebras, it seems natural that it corresponds to the notion of centre which is determined not by $\mathbb{C}$ but by $\mathbb{C}_T$.

Second, Filinski calls a value $H : (\sigma \to \tau) \to \sigma' \to \tau'$ "rigid" when there are total $h_1 : \sigma' \to \tau \to \tau'$ and $h_2 : \sigma' \to \sigma$ such that

$$H = \lambda f^{\sigma\to\tau}.\lambda y^{\sigma'}.h_1\, y\,(f\,(h_2\, y)) : (\sigma \to \tau) \to \sigma' \to \tau'$$

holds (cf. Example 2). It is easily checked that if H is rigid in the sense of Filinski, it is also rigid in our sense – but the converse does not hold, even if we change the notion of total values to ours (for instance, $\mathsf{switch}_{\sigma,\tau}$ in Section 4 is not rigid in the sense of Filinski). By closely inspecting the correspondence of rigid functionals and total functions via the $\mathsf{step}/\mathsf{pets}$ and switch constructions, we can strengthen Filinski's formulation to match ours:

PROPOSITION 8. *In a cbv $\lambda\mu$-theory, $H : (\sigma \to \tau) \to \sigma' \to \tau'$ is rigid if and only if there are total $h_1 : \sigma' \to \tau \to \tau'$ and $h_2 : (\sigma' \times \neg\tau') \to \sigma$ such that $H = \lambda f^{\sigma\to\tau}.\lambda y^{\sigma'}.\mu\gamma^{\tau'}.[\gamma](h_1\, y\,(f\,(h_2\,\langle y, \lambda z^{\tau'}.[\gamma]z\rangle)))$ holds.*

Proof. By pre- and post-composing switch and switch^{-1}, rigid functionals of $(\sigma \to \tau) \to \sigma' \to \tau'$ are in bijective correspondence with those of $\neg(\sigma \times \neg\tau) \to \neg(\sigma' \times \neg\tau')$, which are, by Lemma 8, in bijective correspondence with the total functions of $(\sigma' \times \neg\tau') \to (\sigma \times \neg\tau)$ via the $\mathsf{step}/\mathsf{pets}$ construction. A total function of $(\sigma' \times \neg\tau') \to (\sigma \times \neg\tau)$ is equal to $\lambda\langle y,k\rangle.\langle h_2\,\langle y,k\rangle, h_1\,\langle y,k\rangle\rangle : (\sigma' \times \neg\tau') \to (\sigma \times \neg\tau)$ for some total functions $h_2 : (\sigma' \times \neg\tau') \to \sigma$ and $h_1 : (\sigma' \times \neg\tau') \to \neg\tau$. We note that total functions of $(\sigma' \times \neg\tau') \to \neg\tau$ are in bijective correspondence with those of $\sigma' \to \tau \to \tau'$ – for $f : (\sigma' \times \neg\tau') \to \neg\tau$ we have $\lambda y.\lambda x.\mu\beta.f\,\langle y, \lambda z.[\beta]z\rangle\, x : \sigma' \to \tau \to \tau'$ and for $g : \sigma' \to \tau \to \tau'$ we take $\lambda\langle y,k\rangle.\lambda x.k\,(g\, y\, x) : (\sigma' \times \neg\tau') \to \neg\tau$.

In summary, for any rigid functional $H : (\sigma \to \tau) \to \sigma' \to \tau'$ we have total $h_1 : \sigma' \to \tau \to \tau'$ and $h_2 : (\sigma' \times \neg\tau') \to \sigma$ such that

$$H = \mathsf{switch} \circ (\mathsf{pets}\,(\lambda\langle y,k\rangle.\langle h_2\langle y,k\rangle, \lambda x.k\,(h_1\, y\, x)\rangle)) \circ \mathsf{switch}^{-1}$$

holds. By simplifying the right hand side of this equation, we obtain the result. □

This subsumes Filinski's rigid functionals as special cases where h_2 does not use the second argument.

8. Conclusion and Further Work

We have proposed an axiomatization of fixpoint operators in typed call-by-value programming languages, and have shown that it can be justified in two different ways: as a sound and complete axiomatization for uniform T-fixpoint operators of Simpson and Plotkin [27], and also by Filinski's bijective correspondence between recursion and iteration in the presence of first-class continuations [5]. We also have shown that these results are closely related, by inspecting the semantic structure behind Filinski's construction, which turns out to be a special case of the uniform T-fixpoint operators.

We think that our axioms are reasonably simple, and we expect they can be a practical tool for direct-style reasoning about call-by-value programs involving recursion, just in the same way as the equational theory of the computational lambda calculus is the theoretical basis of the theory of A-normal forms [23, 8].

8.1. Further principles for call-by-value recursion

It is an interesting challenge to strengthen the axioms in some systematic ways. Below we give some results and perspectives.

Dinaturality, diagonal property, and Iteration Theories
By adding other natural axioms on an iterator in the presence of first-class continuations, one may derive the corresponding axioms on the cbv fixpoint operator. In particular, we note that the *dinaturality*

$$\mathsf{loop}\,(g \circ f) \;=\; \mathsf{loop}\,(f \circ g) \circ f$$

on an iterator loop precisely amounts to the axiom

$$\mathsf{fix}^{\mathsf{v}}\,(G \circ (\lambda f.\lambda y.F\,f\,y))) \;=\; \lambda z.G\,(\mathsf{fix}^{\mathsf{v}}\,(F \circ (\lambda g.\lambda x.G\,g\,x)))\,z$$

on the corresponding cbv fixpoint operator $\mathsf{fix}^{\mathsf{v}}$ (note that this axiom implies both the cbv fixpoint axiom and the stability axiom). Similarly, the *diagonal property* on the iterator

$$\mathsf{loop}\,(\lambda x.\mu\alpha.[\alpha,\alpha](f\,x)) \;=\; \mathsf{loop}\,(\lambda x.\mu\alpha.\mathsf{loop}\,(\lambda y.\mu\beta.[\alpha,\beta](f\,y))\,x)$$

corresponds to that on the fixpoint operator

$$\mathsf{fix}^{\mathsf{v}}\,(\lambda f.F\,f\,f) \;=\; \mathsf{fix}^{\mathsf{v}}\,(\lambda f.\mathsf{fix}^{\mathsf{v}}\,(\lambda g.F\,f\,g)).$$

These can be seen axiomatizing the call-by-value counterpart of Conway theories [1, 12]. In [27], Simpson and Plotkin have shown that the equational theory induced by a uniform Conway operator (provided it is consistent) is the smallest iteration theory of Bloom and Ésik [1], which enjoys very general completeness theorem. Regarding this fact, we conjecture that our axioms for stable uniform cbv fixpoint operators together with the dinaturality and diagonal property capture all the valid identities on the cbv fixpoint operators, at least in the presence of first-class continuations.

Mutual recursion and extensions to product types

One may further consider the call-by-value version of the *Bekič property* (another equivalent axiomatization of dinatural and diagonal properties [12]) along this line, which could be used for reasoning about mutual recursion. For this purpose it is natural to extend the definition of fixpoint operators on product types of function types, and also extend the notion of rigid functionals to those with multiple parameters. These extensions are syntactically straightforward and semantically natural (as the category of algebras is closed under finite products). Spelling this out, for $\phi = \sigma \to \tau$ and $\phi' = \sigma' \to \tau'$, we can (uniquely) extend the fixpoint operator on $\phi \times \phi'$ by (using the idea of Section 6)

$$\mathsf{fix}^{\mathsf{v}}_{\phi\times\phi'}\, F^{\phi\times\phi'\to\phi\times\phi'} \;=\; \alpha\,(\mathsf{fix}^{\mathsf{v}}_{\top\to\phi\times\phi'}\,(\lambda h^{\top\to\phi\times\phi'}.\lambda u^{\top}.F\,(\alpha\, h)))$$

where $\alpha = \lambda h^{\top\to\phi\times\phi'}.\langle \lambda x^{\sigma}.\pi_1\,(h\,*)\,x,\ \lambda y^{\sigma'}.\pi_2\,(h\,*)\,y\rangle$. Bekič property is stated as, for $F_1 : \phi\times\phi' \to \phi$ and $F_2 : \phi\times\phi' \to \phi'$,

$$\mathsf{fix}^{\mathsf{v}}_{\phi\times\phi'}\,(\lambda z^{\phi\times\phi'}.\langle F_1\, z,\ F_2\, z\rangle) \;=\; \langle V,\ \mathsf{fix}^{\mathsf{v}}_{\phi'}\,(\lambda g^{\phi'}.F_2\,\langle V,\ g\rangle)\rangle \;:\; \phi\times\phi'$$

where $V = \mathsf{fix}^{\mathsf{v}}_{\phi}\,(\lambda f^{\phi}.F_1\,\langle f, \mathsf{fix}^{\mathsf{v}}_{\phi'}\,(\lambda g^{\phi'}.F_2\,\langle f, g\rangle)\rangle) : \phi$. For example, from Bekič property and uniformity, we can show equations like $\mathsf{fix}^{\mathsf{v}}\, F = \mathsf{fix}^{\mathsf{v}}\,(F \circ (\lambda f.\lambda x.F\, f\, x))$.

Fixpoint objects

Another promising direction is the approach based on *fixpoint objects* [2], as a uniform T-fixpoint operator is canonically derived from a fixpoint object whose universal property implies strong proof principles. For instance, in Example 3, a uniform iterator is unique because the monad $R^{R^{(-)}}$ has a fixpoint object. For the setting with first-class continuations, it might be fruitful to study the implications of the existence of a fixpoint object of continuation monads.

Graphical axioms

Jeffrey [14] argues the possibility of *partial traces* as a foundation of graphical reasoning on recursion in call-by-value languages. Schweimeier and Jeffrey [24] demonstrate that such graphical axioms can be used to verify the closure conversion phase of a compiler. Similar consideration is found in Führmann's thesis [10]. It follows that most of the equalities proposed in these approaches can be derived (up to some syntactic differences) from the axioms for stable uniform call-by-value fixpoint operators with dinatural and diagonal (or Bekič) property; detailed comparisons, however, are left as a future work.

In a related but different direction, Erkök and Launchbury [3] propose graphical axioms for reasoning about recursion with monadic effects in lazy functional programming languages. Friedman and Sabry [9] also discuss about recursion in such settings ("unfolding recursion" versus "updating recursion") and propose an implementation of "updating recursion" via a monadic effect. Although these approaches have the common underlying semantic structure as the present work, the problems cosidered are rather of different nature and it is not clear how they can be compared with our work.

8.2. Relating recursion in call-by-name and call-by-value

The results reported here can be nicely combined with *Filinski's duality* [4] between call-by-value and call-by-name languages with first-class control primitives. In his MSc thesis [15], the second author demonstrates that recursion in the *call-by-name $\lambda\mu$-calculus* [20] exactly corresponds to iteration in the call-by-value $\lambda\mu$-calculus via this duality, by extending Selinger's work [25]. Together with the results in this paper, we obtain a bijective correspondence between call-by-name recursion and call-by-value recursion (both subject to suitable uniformity principles)

Recursion in cbn $\lambda\mu$-calculus	$\Leftrightarrow$	Iteration in cbv $\lambda\mu$-calculus	$\Leftrightarrow$	Recursion in cbv $\lambda\mu$-calculus

which seems to open a way to relate the reasoning principles on recursive computations under these two calling strategies.

Acknowledgements

We thank Shin-ya Katsumata and Carsten Führmann for helpful discussions and their interests on this work, and the anonymous reviewers of FoSSaCS 2001 and this submission for numerous insightful suggestions. Part of this work was done while the first author was visiting Laboratory for Foundations of Computer Science, University of Edinburgh.

References

1. Bloom, S. and Esik, Z. (1993) *Iteration Theories.* EATCS Monographs on Theoretical Computer Science, Springer-Verlag.
2. Crole, R.L. and Pitts, A.M. (1992) New foundations for fixpoint computations: FIX-hyperdoctrines and the FIX-logic. *Inform. and Comput.* **98**(2), 171–210.
3. Erkök, L. and Launchbury, J. (2000) Recursive monadic bindings. In *Proc. International Conference on Functional Programming*, pp. 174–185.
4. Filinski, A. (1989) Declarative continuations: an investigation of duality in programming language semantics. In *Proc. Category Theory and Computer Science*, Springer Lecture Notes in Comput. Sci. **389**, pp. 224–249.
5. Filinski, A. (1994) Recursion from iteration. *Lisp and Symbolic Comput.* **7**(1), 11–38.
6. Filinski, A. (1996) *Controlling Effects.* PhD thesis, Carnegie Mellon University, CMU-CS-96-119.
7. Fiore, M. (1994) *Axiomatic Domain Theory in Categories of Partical Maps.* PhD thesis, University of Edinburgh, ECS-LFCS-94-307; also in Distinguished Dissertation Series, Cambridge University Press, 1996.
8. Flanagan, C., Sabry, A., Duba, B.F. and Felleisen, M. (1993) The essence of compiling with continuations. In *Proc. ACM Conference on Programming Languages Design and Implementation*, pp. 237–247.
9. Friedman, D.P. and Sabry, A. (2000) Recursion is a computational effect. Technical Report No. 546, Computer Science Department, Indiana University.
10. Führmann, C. (2000) *The Structure of Call-by-Value.* PhD thesis, University of Edinburgh.
11. Harper, R., Duba, B.F. and MacQueen, D. (1993) Typing first-class continuations in ML. *J. Funct. Programming* **3**(4), 465–484.
12. Hasegawa, M. (1997) *Models of Sharing Graphs: A Categorical Semantics of* let *and* letrec. PhD thesis, University of Edinburgh, ECS-LFCS-97-360; also in Distinguished Dissertation Series, Springer-Verlag, 1999.
13. Hofmann, M. (1995) Sound and complete axiomatisations of call-by-value control operators. *Math. Structures Comput. Sci.* **5**(4), 461–482.
14. Jeffrey, A. (1998) Premonoidal categories and a graphical view of programs. Manuscript.
15. Kakutani, Y. (2001) *Duality between Call-by-Name Recursion and Call-by-Value Iteration.* MSc thesis, Kyoto University.
16. Lambek, J. and Scott, P. (1986) *Introduction to Higher Order Categorical Logic.* Cambridge University Press.
17. Milner, R., Tofte, M., Harper, R. and MacQueen, D. (1997) *The Definition of Standard ML (Revised).* MIT Press.
18. Moggi, E. (1989) Computational lambda-calculus and monads. In *Proc. 4th Annual Symposium on Logic in Computer Science*, pp. 14–23; a different version available as Technical Report ECS-LFCS-88-86, University of Edinburgh, 1988.
19. Moggi, E. (1991) Notions of computation and monads. *Inform. and Comput.* **93**(1), 55–92.
20. Parigot, M. (1992) $\lambda\mu$-calculus: an algorithmic interpretation of classical natural deduction. In *Proc. International Conference on Logic Programming and Automated Reasoning*, Springer Lecture Notes in Comput. Sci. **624**, pp. 190–201.
21. Plotkin, G.D. (1975) Call-by-name, call-by-value, and the λ-calculus. *Theoret. Comput. Sci.* **1**(1), 125–159.

22. Power, A.J. and Robinson, E.P. (1997) Premonoidal categories and notions of computation. *Math. Structures Comput. Sci.* **7**(5), 453–468.
23. Sabry, A. and Felleisen, M. (1992) Reasoning about programs in continuation-passing style. In *Proc. ACM Conference on Lisp and Functional Programming*, pp. 288–298; extended version in *Lisp and Symbolic Comput.* **6**(3/4), 289–360, 1993.
24. Schweimeier, R. and Jeffrey, A. (1999) Compilation of higher-order languages in graphical form. In *Proc. 15th Conference on Mathematical Foundations of Programming Semantics*, Electron. Notes Theor. Comput. Sci. **20**.
25. Selinger, P. (2001) Control categories and duality: on the categorical semantics of the lambda-mu calculus. *Math. Structures Comput. Sci.* **11**(2), 207–260.
26. Simpson, A.K. (1992) Recursive types in Kleisli categories. Manuscript.
27. Simpson, A.K. and Plotkin, G.D. (2000) Complete axioms for categorical fixed-point operators. In *Proc. 15th Annual Symposium on Logic in Computer Science*, pp. 30–41.
28. Thielecke, H. (1997) *Categorical Structure of Continuation Passing Style*. PhD thesis, University of Edinburgh, ECS-LFCS-97-376.
29. Thielecke, H. (1999) Using a continuation twice and its implications for the expressive power of call/cc. *Higher-Order and Symbolic Comput.* **12**(1), 47–73.

Appendix

A. Semantic interpretation of the λ_c-calculus

Below we work in the base category $\mathbb{C}$. Recall that the each component of the unit η, multiplication μ and tensorial strength θ of a strong monad T has the following typing.

$$\eta_X : X \to TX \qquad \mu_X : T^2X \to TX \qquad \theta_{X,Y} : X \times TY \to T(X \times Y)$$

We use the following notations for the adjoint mate (currying) and counit (evaluation map) of the Kleisli exponentials:

$$\frac{f : A \times X \to TY}{\mathsf{cur}(f) : A \to X \Rightarrow Y} \qquad \mathsf{ev}_{X,Y} : (X \Rightarrow Y) \times X \to TY$$

We define the "left-first pairing" $\varphi_{X,Y} : TX \times TY \to T(X \times Y)$ by

$$\varphi_{X,Y} = TX \times TY \stackrel{\theta'_{X,TY}}{\longrightarrow} T(X \times TY) \stackrel{T\theta_{X,Y}}{\longrightarrow} T^2(X \times Y) \stackrel{\mu_{X\times Y}}{\longrightarrow} T(X \times Y)$$

where θ' is obtained from θ by pre- and post-composing suitable symmetry morphisms. (By exchanging θ and θ', we also obtain the right-first pairing $\varphi'_{X,Y} : TX \times TY \to T(X \times Y)$.) Using these notations, the interpretation is given as follows.

$$
\begin{array}{rcl}
[\![\Gamma \vdash x_i : \sigma_i]\!] & = & [\![\Gamma]\!] \xrightarrow{\mathsf{proj}_i} [\![\sigma_i]\!] \xrightarrow{\eta_{[\![\sigma_i]\!]}} T[\![\sigma_i]\!] \quad (\Gamma = x_1 : \sigma_1, \ldots, x_n : \sigma_n) \\
[\![\Gamma \vdash \lambda x^\sigma . M : \sigma \to \tau]\!] & = & [\![\Gamma]\!] \xrightarrow{\mathsf{cur}([\![\Gamma, x:\sigma \vdash M:\tau]\!])} [\![\sigma]\!] \Rightarrow [\![\tau]\!] \xrightarrow{\eta_{[\![\sigma]\!] \Rightarrow [\![\tau]\!]}} T([\![\sigma]\!] \Rightarrow [\![\tau]\!]) \\
[\![\Gamma \vdash M^{\sigma \to \tau} N^\sigma : \tau]\!] & = & [\![\Gamma]\!] \xrightarrow{\langle [\![\Gamma \vdash M:\sigma\to\tau]\!], [\![\Gamma \vdash N:\sigma]\!] \rangle} T([\![\sigma]\!] \Rightarrow [\![\tau]\!]) \times T[\![\sigma]\!] \xrightarrow{\varphi} \\
 & & T(([\![\sigma]\!] \Rightarrow [\![\tau]\!]) \times [\![\sigma]\!]) \xrightarrow{T\mathsf{ev}_{[\![\sigma]\!],[\![\tau]\!]}} T^2[\![\tau]\!] \xrightarrow{\mu_{[\![\tau]\!]}} T[\![\tau]\!] \\
[\![\Gamma \vdash * : \top]\!] & = & [\![\Gamma]\!] \xrightarrow{!_{[\![\Gamma]\!]}} 1 \xrightarrow{\eta_1} T1 \\
[\![\Gamma \vdash \langle M, N \rangle : \sigma \times \tau]\!] & = & [\![\Gamma]\!] \xrightarrow{\langle [\![\Gamma \vdash M:\sigma]\!], [\![\Gamma \vdash N:\tau]\!] \rangle} T[\![\sigma]\!] \times T[\![\tau]\!] \xrightarrow{\varphi} T([\![\sigma]\!] \times [\![\tau]\!]) \\
[\![\Gamma \vdash \pi_i\, M^{\sigma_1 \times \sigma_2} : \sigma_i]\!] & = & [\![\Gamma]\!] \xrightarrow{[\![\Gamma \vdash M:\sigma_1 \times \sigma_2]\!]} T([\![\sigma_1]\!] \times [\![\sigma_2]\!]) \xrightarrow{T\mathsf{proj}_i} T[\![\sigma_i]\!]
\end{array}
$$

B. Semantic interpretation of the cbv $\lambda\mu$-calculus

We give the interpretation in the response category $\mathbb{C}$. We use the simply typed $\lambda\beta\eta$-calculus with products and sums for describing the morphisms in $\mathbb{C}$ (although we need only function types of the form R^X). This enables us to state the interpretation as a CPS transformation. Recall that, for the continuation monad $T = R^{R^{(-)}}$, we have the interpretation of types as $[\![\sigma \to \tau]\!] = R^{(R^{[\![\tau]\!]})^{[\![\sigma]\!]}}$, $[\![\top]\!] = 1$, $[\![\sigma \times \tau]\!] = [\![\sigma]\!] \times [\![\tau]\!]$, $[\![\bot]\!] = 0$ and $[\![\sigma + \tau]\!] = [\![\sigma]\!] + [\![\tau]\!]$. The unit, multiplication and tensorial strength are given by

$$
\begin{array}{rcl}
\eta_X(x^X) & = & \lambda k^{R^X}.k\, x \\
\mu_X(m^{R^{R^{R^{R^X}}}}) & = & \lambda k^{R^X}.m\, (\lambda h^{R^{R^X}}.h\, k) \\
\theta_{X,Y}(x^X, h^{R^{R^Y}}) & = & \lambda k^{R^{X \times Y}}.h\, (\lambda y^Y.k\, \langle x, y \rangle)
\end{array}
$$

The derived transformation is

$$
\begin{array}{rcl}
\overline{x} & = & \lambda k.k\, x \\
\overline{\lambda x.N} & = & \lambda k.k\, (\lambda x.\overline{M}) \\
\overline{M\, N} & = & \lambda k.\overline{M}\, (\lambda m.\overline{N}\, (\lambda n.m\, n\, k)) \\
\overline{*} & = & \lambda k.k\, * \\
\overline{\langle M, N \rangle} & = & \lambda k.\overline{M}\, (\lambda m.\overline{N}\, (\lambda n.k\, \langle m, n \rangle)) \\
\overline{\pi_i M} & = & \lambda k.\overline{M}(\lambda m.k\, (\pi_i\, m)) \\
\overline{[\alpha]M} & = & \lambda k.\overline{M}\alpha \\
\overline{\mu\alpha.M} & = & \lambda \alpha.\overline{M}\, \diamondsuit \\
\overline{[\alpha,\beta]M} & = & \lambda k.\overline{M}(\lambda z.\mathsf{case}\ z\ \mathsf{of}\ (\mathsf{in}_1\, x \Rightarrow \alpha\, x \mid \mathsf{in}_2\, y \Rightarrow \beta\, y)) \\
\overline{\mu(\alpha,\beta).M} & = & \lambda \gamma.\overline{M}[\lambda x.\gamma\, (\mathsf{in}_1\, x)/\alpha,\ \lambda y.\gamma\, (\mathsf{in}_2\, y)/\beta]\, \diamondsuit
\end{array}
$$

where $\diamondsuit$ is the unique inhabitant of R^0.
Given $x_1 : \sigma_1, \ldots x_m : \sigma_m \vdash M : \sigma \mid \alpha_1 : \tau_1, \ldots, \alpha_n : \tau_n$, we have

$$x_1 : [\![\sigma_1]\!], \ldots, x_m : [\![\sigma_m]\!], \alpha_1 : R^{[\![\tau_1]\!]}, \ldots, \alpha_n : R^{[\![\tau_n]\!]} \vdash \overline{M} : R^{R^{[\![\sigma]\!]}}$$

which amounts to a morphism of

$$\begin{array}{rl} & \mathbb{C}([\![\sigma_1]\!] \times \ldots \times [\![\sigma_m]\!] \times R^{[\![\tau_1]\!]} \times \ldots \times R^{[\![\tau_n]\!]},\, R^{R^{[\![\sigma]\!]}}) \\ \simeq & \mathbb{C}([\![\sigma_1]\!] \times \ldots \times [\![\sigma_m]\!],\, R^{R^{[\![\sigma]\!]+[\![\tau_1]\!]+\ldots+[\![\tau_n]\!]}}) \\ \simeq & \mathbb{C}_T([\![\sigma_1]\!] \times \ldots \times [\![\sigma_m]\!],\, [\![\sigma]\!] + [\![\tau_1]\!] + \ldots + [\![\tau_n]\!]) \quad . \end{array}$$