Intensionality, Intensional Recursion, and the Gödel-Löb Axiom

G. A. Kavvos
Department of Computer Science, University of Oxford
Wolfson Building, Parks Road, Oxford OX1 3QD, United Kingdom
alex.kavvos@bristol.ac.uk

Abstract

The use of a necessity modality in a typed λ-calculus can be used to separate it into two regions. These can be thought of as intensional vs. extensional data: data in the first region, the modal one, are available as code, and their description can be examined. In contrast, data in the second region are only available as values up to ordinary equality. This allows us to add non-functional operations at modal types whilst maintaining consistency. In this setting, the Gödel-Löb axiom acquires a novel constructive reading: it affords the programmer the possibility of a very strong kind of recursion which enables them to write programs that have access to their own code. This is a type of computational reflection that is strongly reminiscent of Kleene's Second Recursion Theorem.

1 Introduction

This paper is about putting a logical twist on two old pieces of programming lore:

- First, it is about using *modal types* to treat *programs-as-data* in a type-safe manner.
- Second, it is about noticing that—in the context of intensional programming—a constructive reading of the Gödel-Löb axiom, i.e. $\Box(\Box A \to A) \to \Box A$, amounts to a strange kind of recursion, namely *intensional recursion*.

We will introduce a *typed λ-calculus with modal types* that supports both of these features. We will call it *Intensional PCF*, after the simply-typed λ-calculus with $\mathbf{Y}$ introduced by Scott [34] and Plotkin [32].

This is a revised version of the third chapter of [23], which is in turn based on a paper presented at the 7th Workshop on Intuitionistic Modal Logic and Applications (IMLA 2017).

1.1 Intensionality and Programs-as-data

To begin, we want to discuss our notion of *programs-as-data*. We mean it in a way that is considerably stronger than the higher-order functional programming with which we are already familiar, i.e. 'functions as first-class citizens.' In addition to that, our notion hints at a kind of *homoiconicity*, similar to the one present in the Lisp family of languages. It refers to the ability given to a programmer to *quote* code, and carry it around as a datum; see [5] for an instance of that in Lisp. This ability can be used for *metaprogramming*, which is the activity of writing programs that write other programs. Indeed, this is what Lisp macros excel at [12], and what the *metaprogramming community* has been studying for a long time; see e.g. [37, 39]. Considering programs as data—but in an untyped manner—was also the central idea in the work of the *partial evaluation community*: see [19, 16, 17].

But we would like to go even further. In Lisp, a program is able to process code by treating it as mere symbols, thereby disregarding its function and behaviour. This is what we call *intensionality*: an operation is *intensional* if it is *finer than equality*. This amounts to a kind of *non-functional computation*. That this may be done type-theoretically was suspected by Davies and Pfenning [31, 9], who introduced modal types to programming language theory. A system based on nominal techniques that fleshed out those ideas was presented by Nanevski [29]. The notions of intensional and extensional equality implicit in this system were studied using logical relations by Pfenning and Nanevski [30]. However, none of these papers studied whether the induced equational systems are consistent. We show that, no matter the intensional mechanism at use, modalities enable consistent intensional programming.

To our knowledge, this paper presents the first consistency proof for intensional programming.

1.2 Intensional Recursion

We also want to briefly explain what we mean by *intensional recursion*; a fuller discussion may be found in [1, 23]. Most modern programming languages support *extensional recursion*: in the body of a function definition, the programmer may make a finite number of calls to the definiendum itself. Operationally, this leads a function to examine its own values at a finite set of points at which it has hopefully already been defined. In the *untyped λ-calculus*, with $=_\beta$ standing for β-convertibility, this is modelled by the *First Recursion Theorem (FRT)* [4, §6.1]:

Theorem 1 (First Recursion Theorem)**.** $\forall f \in \Lambda.\ \exists u \in \Lambda.\ u =_\beta fu$.

However, as Abramsky [1] notes, in the *intensional paradigm* we have described above a stronger kind of recursion is attainable. Instead of merely examining the

result of a finite number of recursive calls, the definiendum can recursively have access to a *full copy of its own source code*. This is embodied in Kleene's *Second Recursion Theorem (SRT)* [24]. Here is a version of the SRT in the untyped λ-calculus, where $\ulcorner u \urcorner$ means 'the Gödel number of the term u' [4, §6.5, Thm. 6.5.9].

Theorem 2 (Second Recursion Theorem)**.** $\forall f \in \Lambda.\ \exists u \in \Lambda.\ u =_\beta f \ulcorner u \urcorner$.

Kleene also proved the following, where Λ^0 is the set of closed λ-terms:

Theorem 3 (Existence of Interpreter)**.** $\exists \mathbf{E} \in \Lambda^0.\ \forall M \in \Lambda^0.\ \mathbf{E} \ulcorner M \urcorner \to^* M$

It is not hard to see that, using Theorem 3, the SRT implies the FRT for closed terms: given $f \in \Lambda^0$ we let $F \stackrel{\text{def}}{=} \lambda y.\ f(\mathbf{E}\, y)$, so that the SRT applied to F yields a term u such that

$$u =_\beta F \ulcorner u \urcorner =_\beta f\,(\mathbf{E} \ulcorner u \urcorner) =_\beta f\, u$$

It is not at all evident whether the converse holds. This is because the SRT is a *first-order theorem* that is about diagonalisation, Gödel numbers and source code, whereas the FRT really is about *higher types*: see the discussion in [23, §2].

Hence, in the presence of intensional operations, the SRT affords us with a much stronger kind of recursion. In fact, it allows for a certain kind of *computational reflection*, or *reflective programming*, of the same kind envisaged by Brian Cantwell Smith [35]. But the programme of Smith's *reflective tower* involved a rather mysterious construction with unclear semantics [10, 42, 8], eventually leading to a theorem that—even in the presence of a mild reflective construct, the so-called **fexpr**—observational equivalence of programs collapses to α-conversion: see Wand [41]. Similar forays have also been attempted by the partial evaluation community: see [14, 15, 18].

We will use modalities to stop intension from flowing back into extension, so that the aforementioned theorem in [41]—which requires unrestricted quoting—will not apply. We will achieve reflection by internalising the SRT. Suppose that our terms are typed, and that $u : A$. Suppose as well that there is a type constructor $\Box$, so that $\Box A$ means 'code of type A.' Then certainly $\ulcorner u \urcorner : \Box A$, and f is forced to have type $\Box A \to A$. A logical reading of the SRT is then the following: for every $f : \Box A \to A$, there exists a $u : A$ such that $u = f \ulcorner u \urcorner$. This corresponds to *Löb's rule* from *provability logic* [7], namely

$$\frac{\Box A \to A}{A}$$

which is equivalent to adding the Gödel-Löb axiom to the logic. In fact, the punchline of this paper is that *the type of the Second Recursion Theorem is the Gödel-Löb axiom of provability logic.*

To our knowledge, this paper presents the first sound, type-safe attempt at reflective programming.

1.3 Prospectus

In §2 we will introduce the syntax of iPCF, and in §3 we will show that it satisfies basic metatheoretic properties. Following that, in section §4 we will add intensional operations to iPCF. By proving that the resulting notion of reduction is confluent, we will obtain consistency for the system. We then look at the computational behaviour of some important terms in §5, and conclude with two key examples of the new powerful features of our language in §6.

2 Introducing Intensional PCF

Intensional PCF (iPCF) is a typed λ-calculus with modal types. As discussed before, the modal types work in our favour by separating intension from extension, so that the latter does not leak into the former. Given the logical flavour of our previous work on categorical models of intensionality [22], we shall model the types of iPCF after the *constructive modal logic* $\mathsf{S4}$, in the dual-context style pioneered by Pfenning and Davies [31, 9]. Let us seize this opportunity to remark that (a) there are also other ways to capture $\mathsf{S4}$, for which see the survey [20], and that (b) dual-context formulations are not by any means limited to $\mathsf{S4}$: they began in the context of *intuitionistic linear logic* [3], but have recently been shown to also encompass other modal logics: see [21].

iPCF is *not* related to the language Mini-ML that is introduced by [9]: that is a call-by-value, ML-like language, with ordinary call-by-value fixed points. In contrast, ours is a call-by-name language with a new kind of fixed point, namely intensional fixed points. These fixed points will afford the programmer the full power of *intensional recursion*. In logical terms they correspond to throwing the Gödel-Löb axiom $\Box(\Box A \to A) \to \Box A$ into $\mathsf{S4}$. Modal logicians might object to this, as, in conjunction with the T axiom $\Box A \to A$, it will make every type inhabited. We remind them that a similar situation occurs in PCF, where the $\mathbf{Y}_A : (A \to A) \to A$ combinator allows one to write a term $\mathbf{Y}_A(\lambda x : A.\, x)$ at every type A. As in the study of PCF, we care less about the logic and more about the underlying computation: *it is the terms that matter, and the types are only there to stop basic type errors from happening.*

The syntax and the typing rules of iPCF may be found in Figure 1. These are largely the same as Pfenning and Davies' $\mathsf{S4}$, save the addition of some constants (drawn from PCF), and a rule for intensional recursion. The introduction rule for the

modality restricts terms under a $\mathsf{box}\ (-)$ to those containing only modal variables, i.e. variables carrying only intensions or code, but never 'live values:'

$$\frac{\Delta\,;\cdot \vdash M : A}{\Delta\,;\Gamma \vdash \mathsf{box}\ M : \Box A}$$

There is also a rule for intensional recursion:

$$\frac{\Delta\,;z : \Box A \vdash M : A}{\Delta\,;\Gamma \vdash \mathsf{fix}\ z\ \mathsf{in}\ M : A}$$

This will be coupled with the reduction $\mathsf{fix}\ z\ \mathsf{in}\ M \longrightarrow M[\mathsf{box}\ (\mathsf{fix}\ z\ \mathsf{in}\ M)/z]$. This rule is actually just *Löb's rule* with a modal context, and including it in the Hilbert system of a (classical or intuitionistic) modal logic is equivalent to including the Gödel-Löb axiom: see [7] and [40]. Finally, let us record the fact that erasing the modality from the types appearing in either Löb's rule or the Gödel-Löb axiom yields the type of $\mathbf{Y}_A : (A \to A) \to A$, as a rule in the first case, or axiomatically internalised as a constant in the second (both variants exist in the literature: see [13] and [27]). A similar observation for a stronger form of the Löb axiom underlies the stream of work on *guarded recursion* [28, 6]; we recommend the survey [25] for a broad coverage of constructive modalities with a provability-like flavour.

3 Metatheory

iPCF satisfies the expected basic results: structural and cut rules are admissible. This is no surprise given its origin in the well-behaved Davies-Pfenning calculus. We assume the typical conventions for λ-calculi: terms are identified up to α-equivalence, for which we write $\equiv$, and substitution $[\cdot/\cdot]$ is defined in the ordinary, capture-avoiding manner. Bear in mind that we consider occurrences of u in N to be bound in $\mathsf{let\ box}\ u \Leftarrow M\ \mathsf{in}\ N$. Contexts Γ, Δ are lists of type assignments $x : A$. Furthermore, we shall assume that whenever we write a judgement like $\Delta\,;\Gamma \vdash M : A$, then Δ and Γ are *disjoint*, in the sense that $\mathsf{Vars}\,(\Delta) \cap \mathsf{Vars}\,(\Gamma) = \emptyset$, where $\mathsf{Vars}\,(x_1 : A_1, \ldots, x_n : A_n) \stackrel{\text{def}}{=} \{x_1, \ldots, x_n\}$. We write Γ, Γ' for the concatenation of disjoint contexts. Finally, we sometimes write $\vdash M : A$ whenever $\cdot;\cdot \vdash M : A$.

Theorem 4 (Structural & Cut)**.** *The following rules are admissible in iPCF:*

1. *(Weakening)*

$$\frac{\Delta\,;\Gamma, \Gamma' \vdash M : A}{\Delta\,;\Gamma, x : A, \Gamma' \vdash M : A}$$

$$\textbf{Ground Types}\quad G \quad ::= \quad \mathsf{Nat} \mid \mathsf{Bool}$$

$$\textbf{Types}\quad A, B \quad ::= \quad G \mid A \to B \mid \Box A$$

$$\textbf{Terms}\quad M, N \quad ::= \quad x \mid \lambda x{:}A.\ M \mid MN \mid \mathsf{box}\ M \mid \mathsf{let\ box}\ u \Leftarrow M\ \mathsf{in}\ N \mid \widehat{n} \mid \mathsf{true} \mid \mathsf{false} \mid \mathsf{succ} \mid \mathsf{pred} \mid \mathsf{zero?} \mid \supset_G \mid \mathsf{fix}\ z\ \mathsf{in}\ M$$

$$\textbf{Contexts}\quad \Gamma, \Delta \quad ::= \quad \cdot \mid \Gamma, x : A$$

$$\frac{}{\Delta\,;\,\Gamma \vdash \widehat{n} : \mathsf{Nat}} \qquad \frac{}{\Delta\,;\,\Gamma \vdash b : \mathsf{Bool}}\ (b \in \{\mathsf{true}, \mathsf{false}\})$$

$$\frac{}{\Delta\,;\,\Gamma \vdash \mathsf{zero?} : \mathsf{Nat} \to \mathsf{Bool}} \qquad \frac{}{\Delta\,;\,\Gamma \vdash f : \mathsf{Nat} \to \mathsf{Nat}}\ (f \in \{\mathsf{succ}, \mathsf{pred}\})$$

$$\frac{}{\Delta\,;\,\Gamma \vdash \supset_G : \mathsf{Bool} \to G \to G \to G}$$

$$\frac{}{\Delta\,;\,\Gamma, x{:}A, \Gamma' \vdash x : A}\ (\mathsf{var}) \qquad \frac{}{\Delta, u{:}A, \Delta'\,;\,\Gamma \vdash u : A}\ (\Box\mathsf{var})$$

$$\frac{\Delta\,;\,\Gamma, x{:}A \vdash M : B}{\Delta\,;\,\Gamma \vdash \lambda x{:}A.\ M : A \to B}\ (\to \mathcal{I}) \qquad \frac{\Delta\,;\,\Gamma \vdash M : A \to B \quad \Delta\,;\,\Gamma \vdash N : A}{\Delta\,;\,\Gamma \vdash MN : B}\ (\to \mathcal{E})$$

$$\frac{\Delta\,;\,\cdot \vdash M : A}{\Delta\,;\,\Gamma \vdash \mathsf{box}\ M : \Box A}\ (\Box\mathcal{I}) \qquad \frac{\Delta\,;\,\Gamma \vdash M : \Box A \qquad \Delta, u{:}A\,;\,\Gamma \vdash N : C}{\Delta\,;\,\Gamma \vdash \mathsf{let\ box}\ u \Leftarrow M\ \mathsf{in}\ N : C}\ (\Box\mathcal{E})$$

$$\frac{\Delta\,;\,z : \Box A \vdash M : A}{\Delta\,;\,\Gamma \vdash \mathsf{fix}\ z\ \mathsf{in}\ M : A}\ (\Box\mathsf{fix})$$

Figure 1: Syntax and Typing Rules for Intensional PCF

2. *(Exchange)*
$$\frac{\Delta\,;\Gamma, x : A, y : B, \Gamma' \vdash M : C}{\Delta\,;\Gamma, y : B, x : A, \Gamma' \vdash M : C}$$

3. *(Contraction)*
$$\frac{\Delta\,;\Gamma, x : A, y : A, \Gamma' \vdash M : A}{\Delta\,;\Gamma, w : A, \Gamma' \vdash M[w, w/x, y] : A}$$

4. *(Cut)*
$$\frac{\Delta\,;\Gamma \vdash N : A \qquad \Delta\,;\Gamma, x : A, \Gamma' \vdash M : A}{\Delta\,;\Gamma, \Gamma' \vdash M[N/x] : A}$$

Theorem 5 (Modal Structural & Cut)**.** *The following rules are admissible:*

1. *(Modal Weakening)*
$$\frac{\Delta, \Delta'\,;\Gamma \vdash M : C}{\Delta, u : A, \Delta'\,;\Gamma \vdash M : C}$$

2. *(Modal Exchange)*
$$\frac{\Delta, x : A, y : B, \Delta'\,;\Gamma \vdash M : C}{\Delta, y : B, x : A, \Delta'\,;\Gamma \vdash M : C}$$

3. *(Modal Contraction)*
$$\frac{\Delta, x : A, y : A, \Delta'\,;\Gamma \vdash M : C}{\Delta, w : A, \Delta'\,;\Gamma \vdash M[w, w/x, y] : C}$$

4. *(Modal Cut)*
$$\frac{\Delta\,;\cdot \vdash N : A \quad \Delta, u : A, \Delta'\,;\Gamma \vdash M : C}{\Delta, \Delta'\,;\Gamma \vdash M[N/u] : C}$$

3.1 Free variables

In this section we prove a theorem regarding the occurrences of free variables in well-typed terms of iPCF. It turns out that, if a variable occurs free under a $\mathsf{box}\ (-)$ construct, then it has to be in the modal context. This is the property that enforces that *intensions can only depend on intensions.*

Definition 1 (Free variables)**.**

1. The *free variables* $\mathsf{fv}\,(M)$ of a term M are defined by induction on the structure of the term:

$$\begin{aligned} \mathsf{fv}\,(x) &\stackrel{\text{def}}{=} \{x\} & \mathsf{fv}\,(MN) &\stackrel{\text{def}}{=} \mathsf{fv}\,(M) \cup \mathsf{fv}\,(N) \\ \mathsf{fv}\,(\lambda x : A.\ M) &\stackrel{\text{def}}{=} \mathsf{fv}\,(M) - \{x\} & \mathsf{fv}\,(\mathsf{box}\ M) &\stackrel{\text{def}}{=} \mathsf{fv}\,(M) \\ \mathsf{fv}\,(\mathsf{fix}\ z\ \mathsf{in}\ M) &\stackrel{\text{def}}{=} \mathsf{fv}\,(M) - \{z\} \end{aligned}$$

   as well as

$$\mathsf{fv}\,(\mathsf{let\ box}\ u \Leftarrow M\ \mathsf{in}\ N) \stackrel{\text{def}}{=} \mathsf{fv}\,(M) \cup (\mathsf{fv}\,(N) - \{u\})$$

   and $\mathsf{fv}\,(c) \stackrel{\text{def}}{=} \emptyset$ for any constant c.

2. The *unboxed free variables* $\mathsf{fv}_0\,(M)$ of a term are those that do *not* occur under the scope of a $\mathsf{box}\ (-)$ or $\mathsf{fix}\ z\ \mathsf{in}\ (-)$ construct. They are formally defined by replacing the following clauses in the definition of $\mathsf{fv}\,(-)$:

$$\mathsf{fv}_0\,(\mathsf{box}\ M) \stackrel{\text{def}}{=} \emptyset \qquad \mathsf{fv}_0\,(\mathsf{fix}\ z\ \mathsf{in}\ M) \stackrel{\text{def}}{=} \emptyset$$

3. The *boxed free variables* $\mathsf{fv}_{\geq 1}\,(M)$ of a term M are those that *do* occur under the scope of a $\mathsf{box}\ (-)$ construct. They are formally defined by replacing the following clauses in the definition of $\mathsf{fv}\,(-)$:

$$\begin{aligned} \mathsf{fv}_{\geq 1}\,(x) &\stackrel{\text{def}}{=} \emptyset & \mathsf{fv}_{\geq 1}\,(\mathsf{box}\ M) &\stackrel{\text{def}}{=} \mathsf{fv}\,(M) \\ \mathsf{fv}_{\geq 1}\,(\mathsf{fix}\ z\ \mathsf{in}\ M) &\stackrel{\text{def}}{=} \mathsf{fv}\,(M) - \{z\} \end{aligned}$$

Theorem 6 (Free variables)**.**

1. *For every term* M, $\mathsf{fv}\,(M) = \mathsf{fv}_0\,(M) \cup \mathsf{fv}_{\geq 1}\,(M)$.
2. *If and* $\Delta\,;\Gamma \vdash M : A$, *then*

$$\begin{aligned} \mathsf{fv}_0\,(M) &\subseteq \mathsf{Vars}\,(\Gamma) \cup \mathsf{Vars}\,(\Delta) \\ \mathsf{fv}_{\geq 1}\,(M) &\subseteq \mathsf{Vars}\,(\Delta) \end{aligned}$$

Proof.

1. Trivial induction on M.
2. By induction on the derivation of $\Delta\,;\Gamma \vdash M : A$.

□

4 Consistency of Intensional Operations

In this section we shall prove that the modal types of iPCF enable us to consistently add intensional operations on the modal types. These are *non-functional operations on terms* which are not ordinarily definable because they violate equality. All we have to do is assume them as constants at modal types, define their behaviour by introducing a notion of reduction, and then prove that the compatible closure of this notion of reduction is confluent. A known corollary of confluence is that the equational theory induced by the reduction is *consistent*, i.e. does not equate all terms.

There is a caveat involving extension flowing into intension. That is: we need to exclude from consideration terms where a variable bound by a λ occurs under the scope of a $\mathsf{box}\ (-)$ construct. These will never be well-typed, but—since we discuss types and reduction orthogonally—we also need to explicitly exclude them here too.

4.1 Adding intensionality

Davies and Pfenning [31] suggested that the $\Box$ modality can be used to signify intensionality. In fact, in [31, 9] they had prevented reductions from happening under $\mathsf{box}\ (-)$ construct, " [...] since this would violate its intensional nature." But the truth is that neither of these presentations included any genuinely non-functional operations at modal types, and hence their only use was for homogeneous staged metaprogramming. Adding intensional, non-functional operations is a more difficult task. Intensional operations are dependent on *descriptions* and *intensions* rather than *values* and *extensions*. Hence, unlike reduction and evaluation, they cannot be blind to substitution. This is something that quickly came to light as soon as Nanevski [29] attempted to extend the system of Davies and Pfenning to allow 'intensional code analysis' using nominal techniques.

A similar task was also recently taken up by Gabbay and Nanevski [11], who attempted to add a construct $\mathsf{is\text{-}app}$ to the system of Davies and Pfenning, along with the reduction rules

$$\begin{aligned} \mathsf{is\text{-}app}\ (\mathsf{box}\ PQ) &\longrightarrow \mathsf{true} \\ \mathsf{is\text{-}app}\ (\mathsf{box}\ M) &\longrightarrow \mathsf{false} \qquad \text{if } M \text{ is not of the form } PQ \end{aligned}$$

The function computed by $\mathsf{is\text{-}app}$ is truly intensional, as it depends solely on the syntactic structure of its argument: it merely checks if it syntactically is an application or not. As such, it can be considered a *criterion of intensionality*, albeit an extreme one: its definability conclusively confirms the presence of computation up to syntax.

Gabbay and Nanevski tried to justify the inclusion of is-app by producing denotational semantics for modal types in which the semantic domain $[\![\Box A]\!]$ directly involves the actual closed terms of type $\Box A$. However, something seems to have gone wrong with substitution. In fact, we believe that their proof of soundness is wrong: it is not hard to see that their semantics is not stable under the second of these two reductions: take M to be u, and let the semantic environment map u to an application PQ, and then notice that this leads to $[\![\mathsf{true}]\!] = [\![\mathsf{false}]\!]$. We can also see this in the fact that their notion of reduction is *not confluent*. Here is the relevant counterexample: we can reduce like this:

$$\mathsf{let\ box}\ u \Leftarrow \mathsf{box}\ (PQ)\ \mathsf{in\ is\text{-}app}\ (\mathsf{box}\ u) \longrightarrow \mathsf{is\text{-}app}\ (\mathsf{box}\ PQ) \longrightarrow \mathsf{true}$$

But we could have also reduced like that:

$$\mathsf{let\ box}\ u \Leftarrow \mathsf{box}\ (PQ)\ \mathsf{in\ is\text{-}app}\ (\mathsf{box}\ u) \longrightarrow \mathsf{let\ box}\ u \Leftarrow \mathsf{box}\ (PQ)\ \mathsf{in\ false} \longrightarrow \mathsf{false}$$

This example is easy to find if one tries to plough through a proof of confluence: it is very clearly *not* the case that $M \longrightarrow N$ implies $M[P/u] \longrightarrow N[P/u]$ if u is under a box $(-)$, exactly because of the presence of intensional operations such as is-app.

Perhaps the following idea is more workable: let us limit intensional operations to a chosen set of functions $f : \mathcal{T}(A) \to \mathcal{T}(B)$ from terms of type A to terms of type B, and then represent them in the language by a constant $\tilde{f}$, such that $\tilde{f}(\mathsf{box}\ M) \longrightarrow \mathsf{box}\ f(M)$. This set of functions would then be chosen so that they satisfy some sanity conditions. Since we want to have a let construct that allows us to substitute code for modal variables, the following general situation will occur: if $N \longrightarrow N'$, we have

$$\mathsf{let\ box}\ u \Leftarrow \mathsf{box}\ M\ \mathsf{in}\ N \longrightarrow N[M/u]$$

but also

$$\mathsf{let\ box}\ u \Leftarrow \mathsf{box}\ M\ \mathsf{in}\ N \longrightarrow \mathsf{let\ box}\ u \Leftarrow \mathsf{box}\ M\ \mathsf{in}\ N' \longrightarrow N'[M/u]$$

Thus, in order to have confluence, we need $N[M/u] \longrightarrow N'[M/u]$. This will only be the case for reductions of the form $\tilde{f}(\mathsf{box}\ M) \to \mathsf{box}\ f(M)$ if

$$f(N[M/u]) \equiv f(N)[M/u]$$

i.e. if f is *substitutive*. But then a simple naturality argument gives that $f(N) \equiv f(u[N/u]) \equiv f(u)[N/u]$, and hence $\tilde{f}$ is already definable by

$$\lambda x : \Box A.\ \mathsf{let\ box}\ u \Leftarrow x\ \mathsf{in\ box}\ f(u)$$

so such a 'substitutive' function is not intensional after all.

In fact, the only truly intensional operations we can add to our calculus will be those acting on *closed* terms. We will see that this circumvents the problems that arise when intensionality interacts with substitution. Hence, we will limit intensional operations to the following set:

Definition 2 (Intensional operations)**.** Let $\mathcal{T}_0(A)$ be the set of (α-equivalence classes of) closed terms M such that $\cdot\,;\cdot \vdash M : A$. Then, the set of *intensional operations*, $\mathcal{F}(A, B)$, is defined to be the set of all functions $f : \mathcal{T}_0(A) \to \mathcal{T}_0(B)$.

We will include all of these intensional operations $f : \mathcal{T}_0(A) \to \mathcal{T}_0(B)$ in our calculus as constants:

$$\frac{}{\Delta\,;\Gamma \vdash \tilde{f} : \Box A \to \Box B}$$

with reduction rule $\tilde{f}(\mathsf{box}\ M) \to \mathsf{box}\ f(M)$, under the proviso that M is closed. Of course, these also includes operations on terms that *might not be computable*. However, we are interested in proving consistency of intensional operations in the most general setting. The questions of which intensional operations are computable, and which primitives or mechanisms can and should be used to express them, are beyond the scope of this paper, and largely still open.

4.2 Reduction and Confluence

We introduce a notion of reduction for iPCF, which we present in Figure 2. Unlike many studies of PCF-inspired languages, we do not consider a reduction strategy but ordinary 'non-deterministic' β-reduction. We do so because are trying to show consistency of the induced equational theory.

The equational theory induced by this notion of reduction is a symmetric version of it, annotated with types. It is easy to write down, so we omit it. Note the fact that, like the calculus of Davies and Pfenning, we do *not* include the following congruence rule for the modality:

$$\frac{\Delta\,;\cdot \vdash M = N : A}{\Delta\,;\Gamma \vdash \mathsf{box}\ M = \mathsf{box}\ N : \Box A}\ (\Box\mathsf{cong})$$

In fact, the very absence of this rule is what will allow modal types to become intensional. Otherwise, the only new rules are intensional recursion, embodied by the rule ($\Box\mathsf{fix}$), and intensional operations, exemplified by the rule ($\Box\mathsf{int}$).

We note that it seems perfectly reasonable to think that we should allow reductions under fix, i.e. admit the rule

$$\frac{M \longrightarrow N}{\mathsf{fix}\ z\ \mathsf{in}\ M \longrightarrow \mathsf{fix}\ z\ \mathsf{in}\ N}$$

$$\frac{}{(\lambda x : A.\ M)N \longrightarrow M[N/x]}\ (\longrightarrow \beta) \qquad \frac{M \longrightarrow N}{\lambda x : A.\ M \longrightarrow \lambda x : A.\ N}\ (\mathsf{cong}_\lambda)$$

$$\frac{M \longrightarrow N}{MP \longrightarrow NP}\ (\mathsf{app}_1) \qquad \frac{P \longrightarrow Q}{MP \longrightarrow MQ}\ (\mathsf{app}_2)$$

$$\frac{}{\mathsf{let\ box}\ u \Leftarrow \mathsf{box}\ M\ \mathsf{in}\ N \longrightarrow N[M/u]}\ (\Box\beta)$$

$$\frac{}{\mathsf{fix}\ z\ \mathsf{in}\ M \longrightarrow M[\mathsf{box}\ (\mathsf{fix}\ z\ \mathsf{in}\ M)/z]}\ (\Box\mathsf{fix})$$

$$\frac{M\ \text{closed},\ M \in \mathrm{dom}(f)}{\tilde{f}(\mathsf{box}\ M) \longrightarrow \mathsf{box}\ f(M)}\ (\Box\mathsf{int})$$

$$\frac{M \longrightarrow N}{\mathsf{let\ box}\ u \Leftarrow M\ \mathsf{in}\ P \longrightarrow \mathsf{let\ box}\ u \Leftarrow N\ \mathsf{in}\ P}\ (\mathsf{let\text{-}cong}_1)$$

$$\frac{P \longrightarrow Q}{\mathsf{let\ box}\ u \Leftarrow M\ \mathsf{in}\ P \longrightarrow \mathsf{let\ box}\ u \Leftarrow M\ \mathsf{in}\ Q}\ (\mathsf{let\text{-}cong}_2)$$

$$\frac{}{\mathsf{zero?}\ \widehat{0} \longrightarrow \mathsf{true}}\ (\mathsf{zero?}_1) \qquad \frac{}{\mathsf{zero?}\ \widehat{n+1} \longrightarrow \mathsf{false}}\ (\mathsf{zero?}_2)$$

$$\frac{}{\mathsf{succ}\ \widehat{n} \longrightarrow \widehat{n+1}}\ (\mathsf{succ}) \qquad \frac{}{\mathsf{pred}\ \widehat{n} \longrightarrow \widehat{n \dot{-} 1}}\ (\mathsf{pred})$$

$$\frac{}{\supset_G\ \mathsf{true}\ M\ N \longrightarrow M}\ (\supset_1) \qquad \frac{}{\supset_G\ \mathsf{false}\ M\ N \longrightarrow N}\ (\supset_2)$$

Figure 2: Reduction for Intensional PCF

as M and N are expected to be of type A, which need not be modal. However, the reduction $\mathsf{fix}\ z\ \mathsf{in}\ M \longrightarrow M[\mathsf{box}\ (\mathsf{fix}\ z\ \mathsf{in}\ M)/z]$ 'freezes' M under an occurrence of $\mathsf{box}\ (-)$, so that no further reductions can take place within it. Thus, the above rule would violate the intensional nature of boxes. We were likewise compelled to define $\mathsf{fv}_0\ (\mathsf{fix}\ z\ \mathsf{in}\ M) \stackrel{\text{def}}{=} \emptyset$ in the previous section: we should already consider M to be intensional, or under a box.

We can now show that

Theorem 7. *The reduction relation $\longrightarrow$ is confluent.*

The easiest route to that theorem is to use a proof like that in [21], i.e. the method of *parallel reduction*. This kind of proof was originally discovered by Tait and Martin-Löf, and is nicely documented in [38]. Because of the intensional nature of our $\mathsf{box}\ (-)$ constructs, ours will be more nuanced and fiddly. The proof can of course be skipped on a first reading.

Proof of confluence We will use a variant of the proof in [21], i.e. the method of *parallel reduction*. This kind of proof was originally discovered by Tait and Martin-Löf, and is nicely documented in [38]. Because of the intensional nature of our $\mathsf{box}\ (-)$ constructs, ours will be more nuanced and fiddly than any in *op. cit.* The method is this: we will introduce a second notion of reduction,

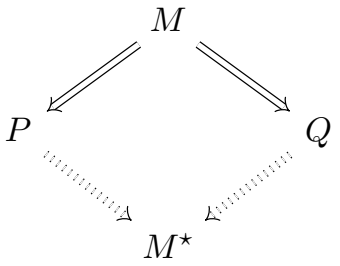
$$\Longrightarrow\ \subseteq \Lambda \times \Lambda$$

which we will 'sandwich' between reduction proper and its transitive closure:

$$\longrightarrow\ \subseteq \Longrightarrow \subseteq \longrightarrow^*$$

We will then show that $\Longrightarrow$ has the diamond property. By the above inclusions, the transitive closure $\Longrightarrow^*$ of $\Longrightarrow$ is then equal to $\longrightarrow^*$, and hence $\longrightarrow$ is Church-Rosser.

In fact, we will follow [38] in doing something better: we will define for each term M its *complete development*, $M^\star$. The complete development is intuitively defined by 'unrolling' all the redexes of M at once. We will then show that if $M \Longrightarrow N$, then $N \Longrightarrow M^\star$. $M^\star$ will then suffice to close the diamond:



$$\frac{}{M \Longrightarrow M}\,(\mathsf{refl}) \qquad \frac{M \Longrightarrow N \qquad P \Longrightarrow Q}{(\lambda x : A.\ M)P \Longrightarrow N[Q/x]}\,(\to \beta)$$

$$\frac{M \Longrightarrow N}{\lambda x : A.\ M \Longrightarrow \lambda x : A.\ N}\,(\mathsf{cong}_\lambda) \qquad \frac{M \Longrightarrow N \qquad P \Longrightarrow Q}{MP \Longrightarrow NQ}\,(\mathsf{app})$$

$$\frac{P \Longrightarrow P'}{\supset_G\ \mathsf{true}\ P\ Q \Longrightarrow P'}\,(\supset_1) \qquad \frac{Q \Longrightarrow Q'}{\supset_G\ \mathsf{false}\ P\ Q \Longrightarrow Q'}\,(\supset_2)$$

$$\frac{M \Longrightarrow N}{\mathsf{let\ box}\ u \Leftarrow \mathsf{box}\ P\ \mathsf{in}\ M \Longrightarrow N[P/u]}\,(\Box\beta)$$

$$\frac{M \Longrightarrow N}{\mathsf{fix}\ z\ \mathsf{in}\ M \Longrightarrow N[\mathsf{box}\ (\mathsf{fix}\ z\ \mathsf{in}\ M)/z]}\,(\Box\mathsf{fix})$$

$$\frac{M\ \text{closed},\ M \in \mathrm{dom}(f)}{\tilde{f}(\mathsf{box}\ M) \Longrightarrow \mathsf{box}\ f(M)}\,(\Box\mathsf{int})$$

$$\frac{M \Longrightarrow N \qquad P \Longrightarrow Q}{\mathsf{let\ box}\ u \Leftarrow M\ \mathsf{in}\ P \Longrightarrow \mathsf{let\ box}\ u \Leftarrow N\ \mathsf{in}\ Q}\,(\Box\mathsf{let\text{-}cong})$$

Remark. In addition to the above, one should also include rules for the constants, but these are restatements of the rules in Figure 2.

Figure 3: Parallel Reduction

The parallel reduction $\Longrightarrow$ is defined in Figure 3. Instead of the axiom (refl) we would more commonly have an axiom for variables, $x \Longrightarrow x$, and $M \Longrightarrow M$ would be derivable. However, we do not have a congruence rule neither for $\mathsf{box}\ (-)$ nor for Löb's rule, so that possibility would be precluded. We are thus forced to include $M \Longrightarrow M$, which slightly complicates the lemmas that follow.

The main lemma that usually underpins the confluence proof is this: if $M \Longrightarrow N$ and $P \Longrightarrow Q$, $M[P/x] \Longrightarrow N[Q/x]$. However, this is intuitively wrong: no reductions should happen under boxes, so this should only hold if we are substituting for a variable *not* occurring under boxes. Hence, this lemma splits into three different ones:

- $P \Longrightarrow Q$ implies $M[P/x] \Longrightarrow M[Q/x]$, if x does not occur under boxes: this

is the price to pay for replacing the variable axiom with (refl).

- $M \Longrightarrow N$ implies $M[P/u] \Longrightarrow N[P/u]$, even if u is under a box.
- If x does not occur under boxes, $M \Longrightarrow N$ and $P \Longrightarrow Q$ indeed imply $M[P/x] \Longrightarrow N[Q/x]$

Lemma 1. *If $M \Longrightarrow N$ then $M[P/u] \Longrightarrow N[P/u]$.*

Proof. By induction on the generation of $M \Longrightarrow N$. Most cases trivially follow, or consist of simple invocations of the IH. In the case of $(\to \beta)$, the known substitution lemma suffices. Let us look at the cases involving boxes.

Case($\Box\beta$). Then $M \Longrightarrow N$ is $\mathsf{let\ box}\ v \Leftarrow \mathsf{box}\ R\ \mathsf{in}\ S \Longrightarrow S'[R/v]$ with $S \Longrightarrow S'$. By the IH, we have that $S[P/u] \Longrightarrow S'[P/u]$, so

$$\mathsf{let\ box}\ v \Leftarrow \mathsf{box}\ R[P/u]\ \mathsf{in}\ S[P/u] \Longrightarrow S'[P/u][R[P/u]/v]$$

and this last is α-equivalent to $S'[R/v][P/u]$ by the substitution lemma.

Case($\Box\mathsf{fix}$). A similar application of the substitution lemma.

Case($\Box\mathsf{int}$). Then $M \Longrightarrow N$ is $\tilde{f}(\mathsf{box}\ Q) \Longrightarrow \mathsf{box}\ f(Q)$. Hence

$$\left(\tilde{f}(\mathsf{box}\ Q)\right)[P/u] \equiv \tilde{f}(\mathsf{box}\ Q) \Longrightarrow \mathsf{box}\ f(Q) \equiv (\mathsf{box}\ f(Q))\,[P/u]$$

simply because both Q and $f(Q)$ are closed.

□

Lemma 2. *If $P \Longrightarrow Q$ and $x \notin \mathsf{fv}_{\geq 1}(M)$, then $M[P/x] \Longrightarrow M[Q/x]$.*

Proof. By induction on the term M. The only non-trivial cases are those for M a variable, $\mathsf{box}\ M'$ or $\mathsf{fix}\ z\ \mathsf{in}\ M'$. In the first case, depending on which variable M is, use either (refl), or the assumption $P \Longrightarrow Q$. In the latter two, $(\mathsf{box}\ M')[P/x] \equiv \mathsf{box}\ M' \equiv (\mathsf{box}\ M')[Q/x]$ as x does not occur under a box, so use (refl), and similarly for $\mathsf{fix}\ z\ \mathsf{in}\ M'$. □

Lemma 3. *If $M \Longrightarrow N$, $P \Longrightarrow Q$, and $x \notin \mathsf{fv}_{\geq 1}(M)$, then*

$$M[P/x] \Longrightarrow N[Q/x]$$

Proof. By induction on the generation of $M \Longrightarrow N$. The cases for most congruence rules and constants follow trivially, or from the IH. We prove the rest.

CASE(refl). Then $M \Longrightarrow N$ is actually $M \Longrightarrow M$, so we use Lemma 2 to infer $M[P/x] \Longrightarrow M[Q/x]$.

CASE($\Box$int). Then $M \Longrightarrow N$ is actually $\tilde{f}(\mathsf{box}\ M) \Longrightarrow \mathsf{box}\ f(M)$. But M and $f(M)$ are closed, so $\left(\tilde{f}(\mathsf{box}\ M)\right)[P/x] \equiv \tilde{f}(\mathsf{box}\ M) \Longrightarrow \mathsf{box}\ f(M) \equiv (\mathsf{box}\ f(M))[Q/x]$.

CASE($\supset_i$). Then $M \Longrightarrow N$ is $\supset_G\ \mathsf{true}\ M\ N \Longrightarrow M'$ with $M \Longrightarrow M'$. By the IH, $M[P/x] \Longrightarrow M'[Q/x]$, so

$$\supset_G\ \mathsf{true}\ M[P/x]\ N[P/x] \Longrightarrow M'[Q/x]$$

by a single use of ($\supset_1$). The case for false is similar.

CASE($\to \beta$). Then $(\lambda x'{:}A.\ M)N \Longrightarrow N'[M'/x']$, where $M \Longrightarrow M'$ and $N \Longrightarrow N'$. Then

$$((\lambda x'{:}A.\ M)N)[P/x] \equiv (\lambda x'{:}A.\ M[P/x])(N[P/x])$$

But, by the IH, $M[P/x] \Longrightarrow M'[Q/x]$ and $N[P/x] \Longrightarrow N'[Q/x]$. So by ($\to \beta$) we have

$$(\lambda x'{:}A.\ M[P/x])(N[P/x]) \Longrightarrow M'[Q/x]\ [N'[Q/x]/x']$$

But this last is α-equivalent to $(M'[N'/x'])\,[Q/x]$ by the substitution lemma.

CASE($\Box\beta$). Then $\mathsf{let\ box}\ u' \Leftarrow \mathsf{box}\ M\ \mathsf{in}\ N \Longrightarrow N'[M/u']$ where $N \Longrightarrow N'$. By assumption, we have that $x \notin \mathsf{fv}\,(M)$ and $x \notin \mathsf{fv}_{\geq 1}\,(N)$. Hence, we have by the IH that $N[P/x] \Longrightarrow N'[Q/x]$, so by applying ($\Box\beta$) we get

$$\begin{aligned}(\mathsf{let\ box}\ u' \Leftarrow \mathsf{box}\ M\ \mathsf{in}\ N)[P/x] &\equiv \mathsf{let\ box}\ u' \Leftarrow \mathsf{box}\ M[P/x]\ \mathsf{in}\ N[P/x] \\ &\equiv \mathsf{let\ box}\ u' \Leftarrow \mathsf{box}\ M\ \mathsf{in}\ N[P/x] \\ &\Longrightarrow N'[Q/x][M/u']\end{aligned}$$

But this last is α-equivalent to $N'[M/u'][Q/x]$, by the substitution lemma and the fact that x does not occur in M.

CASE($\Box$fix). Then $\mathsf{fix}\ z\ \mathsf{in}\ M \Longrightarrow M'[\mathsf{box}\ (\mathsf{fix}\ z\ \mathsf{in}\ M)/z]$, with $M \Longrightarrow M'$. As $x \notin \mathsf{fv}_{\geq 1}\,(\mathsf{fix}\ z\ \mathsf{in}\ M)$, we have that $x \notin \mathsf{fv}\,(M)$, and by Lemma 5, $x \notin \mathsf{fv}\,(M')$ either, so

$$(\mathsf{fix}\ z\ \mathsf{in}\ M)[P/x] \equiv \mathsf{fix}\ z\ \mathsf{in}\ M$$

and

$$M'[\mathsf{fix}\ z\ \mathsf{in}\ M/z][Q/x] \equiv M'[Q/x][\mathsf{fix}\ z\ \mathsf{in}\ M[Q/x]/z] \equiv M'[\mathsf{fix}\ z\ \mathsf{in}\ M/z]$$

Thus, a single use of ($\Box\mathsf{fix}$) suffices.

□

We now pull the following definition out of the hat:

Definition 3 (Complete development)**.** The *complete development* $M^\star$ of a term M is defined by the following clauses:

$$\begin{aligned}
x^\star &\stackrel{\text{def}}{=} x \\
c^\star &\stackrel{\text{def}}{=} c \qquad (c \in \{\tilde{f}, \widehat{n}, \mathsf{zero?}, \dots\}) \\
(\lambda x{:}A.\ M)^\star &\stackrel{\text{def}}{=} \lambda x{:}A.\ M^\star \\
\left(\tilde{f}(\mathsf{box}\ M)\right)^\star &\stackrel{\text{def}}{=} \mathsf{box}\ f(M) \qquad \text{if } M \text{ is closed} \\
((\lambda x{:}A.\ M)\ N)^\star &\stackrel{\text{def}}{=} M^\star[N^\star/x] \\
(\supset_G\ \mathsf{true}\ M\ N)^\star &\stackrel{\text{def}}{=} M^\star \\
(\supset_G\ \mathsf{false}\ M\ N)^\star &\stackrel{\text{def}}{=} N^\star \\
(MN)^\star &\stackrel{\text{def}}{=} M^\star N^\star \\
(\mathsf{box}\ M)^\star &\stackrel{\text{def}}{=} \mathsf{box}\ M \\
(\mathsf{let\ box}\ u \Leftarrow \mathsf{box}\ M\ \mathsf{in}\ N)^\star &\stackrel{\text{def}}{=} N^\star[M/u] \\
(\mathsf{let\ box}\ u \Leftarrow M\ \mathsf{in}\ N)^\star &\stackrel{\text{def}}{=} \mathsf{let\ box}\ u \Leftarrow M^\star\ \mathsf{in}\ N^\star \\
(\mathsf{fix}\ z\ \mathsf{in}\ M)^\star &\stackrel{\text{def}}{=} M^\star[\mathsf{box}\ (\mathsf{fix}\ z\ \mathsf{in}\ M)/z]
\end{aligned}$$

We need the following two technical results as well.

Lemma 4. $M \Longrightarrow M^\star$

Proof. By induction on the term M. Most cases follow immediately by (refl), or by the IH and an application of the relevant rule. The case for $\mathsf{box}\ M$ follows by (refl), the case for $\mathsf{fix}\ z\ \mathsf{in}\ M$ follows by ($\Box\mathsf{fix}$), and the case for $\tilde{f}(\mathsf{box}\ M)$ by ($\Box\mathsf{int}$). □

Lemma 5 (BFV antimonotonicity)**.** *If* $M \Longrightarrow N$ *then* $\mathsf{fv}_{\geq 1}(N) \subseteq \mathsf{fv}_{\geq 1}(M)$.

Proof. By induction on $M \Longrightarrow N$. □

And here is the main result:

Theorem 8. *If $M \Longrightarrow P$, then $P \Longrightarrow M^\star$.*

Proof. By induction on the generation of $M \Longrightarrow P$. The case of (refl) follows by Lemma 4, and the cases of congruence rules follow from the IH. We show the rest.

> Case($\to \beta$). Then we have $(\lambda x{:}A.\ M)N \Longrightarrow M'[N'/x]$, with $M \Longrightarrow M'$ and $N \Longrightarrow N'$. By the IH, $M' \Longrightarrow M^\star$ and $N' \Longrightarrow N^\star$. We have that $x \notin \mathsf{fv}_{\geq 1}(M)$, so by Lemma 5 we get that $x \notin \mathsf{fv}_{\geq 1}(M')$. Hence, by Lemma 3 we get $M'[N'/x] \Longrightarrow M^\star[N^\star/x] \equiv ((\lambda x{:}A.\ M)\,N)^\star$.
>
> Case($\Box\beta$). Then we have
>
> $$\mathsf{let\ box}\ u \Leftarrow \mathsf{box}\ M\ \mathsf{in}\ N \Longrightarrow N'[M/u]$$
>
> where $N \Longrightarrow N'$. By the IH, $N' \Longrightarrow N^\star$, so it follows that
>
> $$N'[M/u] \Longrightarrow N^\star[M/u] \equiv (\mathsf{let\ box}\ u \Leftarrow \mathsf{box}\ M\ \mathsf{in}\ N)^\star$$
>
> by Lemma 1.
>
> Case($\Box\mathsf{fix}$). Then we have
>
> $$\mathsf{fix}\ z\ \mathsf{in}\ M \Longrightarrow M'[\mathsf{box}\ (\mathsf{fix}\ z\ \mathsf{in}\ M)/z]$$
>
> where $M \Longrightarrow M'$. By the IH, $M' \Longrightarrow M^\star$. Hence
>
> $$M'[\mathsf{box}\ (\mathsf{fix}\ z\ \mathsf{in}\ M)/z] \Longrightarrow M^\star[\mathsf{box}\ (\mathsf{fix}\ z\ \mathsf{in}\ M)/z] \equiv (\mathsf{fix}\ z\ \mathsf{in}\ M)^\star$$
>
> by Lemma 1.
>
> Case($\Box\mathsf{int}$). Similar.

□

5 Some important terms

Let us look at the kinds of terms we can write in iPCF.

From the axioms of S4 First, we can write a term corresponding to axiom K, the *normality axiom* of modal logics:

$$\mathsf{ax_K} \stackrel{\text{def}}{=} \lambda f : \Box(A \to B).\ \lambda x : \Box A.\ \mathsf{let\ box}\ g \Leftarrow f\ \mathsf{in\ let\ box}\ y \Leftarrow x\ \mathsf{in\ box}\ (g\,y)$$

Then $\vdash \mathsf{ax_K} : \Box(A \to B) \to (\Box A \to \Box B)$. An intensional reading of this is the following: any function given as code can be transformed into an *effective operation* that maps code of type A to code of type B.

The rest of the axioms correspond to evaluating and quoting. Axiom T takes code to value, or intension to extension:

$$\vdash \mathsf{eval}_A \stackrel{\text{def}}{=} \lambda x : \Box A.\ \mathsf{let\ box}\ y \Leftarrow x\ \mathsf{in}\ y : \Box A \to A$$

and axiom $\mathsf{4}$ quotes code into code-for-code:

$$\vdash \mathsf{quote}_A \stackrel{\text{def}}{=} \lambda x : \Box A.\ \mathsf{let\ box}\ y \Leftarrow x\ \mathsf{in\ box}\ (\mathsf{box}\ y) : \Box A \to \Box\Box A$$

The Gödel-Löb axiom: intensional fixed points Since ($\Box\mathsf{fix}$) is Löb's rule, we expect to be able to write down a term corresponding to the Gödel-Löb axiom of provability logic. We can, and it is an *intensional fixed-point combinator*:

$$\mathbb{Y}_A \stackrel{\text{def}}{=} \lambda x : \Box(\Box A \to A).\ \mathsf{let\ box}\ f \Leftarrow x\ \mathsf{in\ box}\ (\mathsf{fix}\ z\ \mathsf{in}\ f\ z)$$

and $\vdash \mathbb{Y}_A : \Box(\Box A \to A) \to \Box A$. We observe that

$$\mathbb{Y}_A(\mathsf{box}\ M) \longrightarrow^* \mathsf{box}\ (\mathsf{fix}\ z\ \mathsf{in}\ (M\ z))$$

Undefined The combination of eval and intensional fixed points leads to non-termination, in a style reminiscent of the term $(\lambda x.\, xx)(\lambda x.\, xx)$ of the untyped λ-calculus. Let

$$\Omega_A \stackrel{\text{def}}{=} \mathsf{fix}\ z\ \mathsf{in}\ (\mathsf{eval}_A\ z)$$

Then $\vdash \Omega_A : A$, and

$$\Omega_A \longrightarrow\ \mathsf{eval}_A\ (\mathsf{box}\ \Omega_A)\ \longrightarrow^*\ \Omega_A$$

Extensional Fixed Points Perhaps surprisingly, the ordinary PCF $\mathbf{Y}$ combinator is also definable in the iPCF. Let

$$\mathbf{Y}_A \stackrel{\text{def}}{=} \mathsf{fix}\ z\ \mathsf{in}\ \lambda f : A \to A.\ f(\mathsf{eval}\ z\ f)$$

Then $\vdash \mathbf{Y}_A : (A \to A) \to A$, so that

$$\begin{aligned} \mathbf{Y}_A &\longrightarrow^*\ \lambda f : A \to A.\ f(\mathsf{eval}\ (\mathsf{box}\ \mathbf{Y}_A)\ f)) \\ &\longrightarrow^*\ \lambda f : A \to A.\ f(\mathbf{Y}_A\, f) \end{aligned}$$

6 Two intensional examples

No discussion of an intensional language with intensional recursion would be complete without examples that use these two novel features. Our first example uses intensionality, albeit in a 'extensional' way, and is drawn from the study of PCF and issues related to sequential vs. parallel (but not concurrent) computation. Our second example uses intensional recursion, so it is slightly more adventurous: it is a computer virus.

6.1 'Parallel or' by dovetailing

In [32] Gordon Plotkin proved the following theorem: there is no term $\mathsf{por} : \mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool}$ of PCF such that $\mathsf{por}\ \mathsf{true}\ M \twoheadrightarrow_\beta \mathsf{true}$ and $\mathsf{por}\ M\ \mathsf{true} \twoheadrightarrow_\beta \mathsf{true}$ for any $\vdash M : \mathsf{Bool}$, whilst $\mathsf{por}\ \mathsf{false}\ \mathsf{false} \twoheadrightarrow_\beta \mathsf{false}$. Intuitively, the problem is that por has to first examine one of its two arguments, and this can be troublesome if that argument is non-terminating. It follows that the *parallel or* function is not definable in PCF. In order to regain the property of so-called *full abstraction* for the *Scott model* of PCF, a constant denoting this function has to be manually added to PCF, and endowed with the above rather clunky operational semantics. See [32, 13, 27, 36].

However, the parallel or function is a computable *partial recursive functional* [36, 26]. The way to prove that is intuitively the following: given two closed terms $M, N : \mathsf{Bool}$, take turns in β-reducing each one for a one step: this is called *dovetailing*. If at any point one of the two terms reduces to true, then output true. But if at any point both reduce to false, then output false.

This procedure is not definable in PCF because a candidate term por does not have access to a code for its argument, but can only inspect its value. However, in iPCF we can use the modality to obtain access to code, and intensional operations to implement reduction. Suppose we pick a reduction strategy $\longrightarrow_r$. Then, let us include a constant $\mathsf{tick} : \Box\mathsf{Bool} \to \Box\mathsf{Bool}$ that implements one step of this reduction strategy on closed terms:

$$\frac{M \longrightarrow_r N,\ M, N \text{ closed}}{\mathsf{tick}\ (\mathsf{box}\ M) \longrightarrow \mathsf{box}\ N}$$

Also, let us include a constant $\mathsf{done?} : \Box\mathsf{Bool} \to \mathsf{Bool}$, which tells us if a closed term under a box is a normal form:

$$\frac{M \text{ closed, normal}}{\mathsf{done?}\ (\mathsf{box}\ M) \longrightarrow \mathsf{true}} \qquad \frac{M \text{ closed, not normal}}{\mathsf{done?}\ (\mathsf{box}\ M) \longrightarrow \mathsf{false}}$$

These two can be subsumed under our previous scheme for introducing intensional operations. The above argument is now implemented by the following term:

$$\begin{aligned}\mathsf{por} :\equiv\ \mathbf{Y}(\lambda\,\mathsf{por}.\,\lambda x : \Box\mathsf{Bool}.\,\lambda y : \Box\mathsf{Bool}.\\ \supset_{\mathsf{Bool}}\,(\mathsf{done?}\ x)\ (\mathsf{lor}\ (\mathsf{eval}\ x)(\mathsf{eval}\ y))\\ (\supset_{\mathsf{Bool}}\,(\mathsf{done?}\ y)\quad (\mathsf{ror}\ (\mathsf{eval}\ x)(\mathsf{eval}\ y))\\ (\mathsf{por}\ (\mathsf{tick}\ x)(\mathsf{tick}\ y)))\end{aligned}$$

where $\mathsf{lor}, \mathsf{ror} : \mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool}$ are terms defining the left-strict and right-strict versions of the 'or' connective respectively. Notice that the type of this term is $\Box\mathsf{Bool} \to \Box\mathsf{Bool} \to \mathsf{Bool}$: we require *intensional access* to the terms of boolean type in order to define this function!

6.2 A computer virus

Abstract computer virology is the study of formalisms that model computer viruses. There are many ways to formalise viruses. We will use the model of Adleman [2], where files can be interpreted either as data, or as functions. We introduce a data type F of files, and two constants

$$\mathsf{in} : \Box(F \to F) \to F \quad \text{and} \quad \mathsf{out} : F \to \Box(F \to F)$$

If F is a file, then $\mathsf{out}\ F$ is that file interpreted as a program, and similarly for in. We ask that $\mathsf{out}\ (\mathsf{in}\ M) \longrightarrow M$, making $\Box(F \to F)$ a retract of F. This might seem the same as the situation where $F \to F$ is a retract of F, which yields models of the (untyped) λ-calculus, and is not trivial to construct [4, §5.4]. However, in our case it is not nearly as worrying: $\Box(F \to F)$ is populated by programs and codes, not by actual functions. Under this interpretation, the pair $(\mathsf{in}, \mathsf{out})$ corresponds to a kind of Gödel numbering—especially if F is $\mathbb{N}$.

In Adleman's model, a *virus* is given by its infected form, which either *injures*, *infects*, or *imitates* other programs. The details are unimportant in the present discussion, save from the fact that the virus needs to have access to code that it can use to infect other executables. One can hence construct such a virus from its *infection routine*, by using Kleene's SRT. Let us model it by a term

$$\vdash \mathsf{infect} : \Box(F \to F) \to F \to F$$

which accepts a piece of viral code and an executable file, and it returns either the file itself, or a version infected with the viral code. We can then define a term

$$\vdash \mathsf{virus} \stackrel{\text{def}}{=} \mathsf{fix}\ z\ \mathsf{in}\ (\mathsf{infect}\ z) : F \to F$$

so that

$$\mathsf{virus} \longrightarrow^* \mathsf{infect}\ (\mathsf{box}\ \mathsf{virus})$$

which is a program that is ready to infect its input with its own code.

7 Conclusion

We have achieved the desideratum of an intensional programming calculus with intensional recursion. There are two main questions that result from this development.

First, does there exist a good set of *intensional primitives* from which all others are definable? Is there perhaps *more than one such set*, hence providing us with a choice of programming primitives? Previous attempts aiming to answer this question include those of [33, 29].

Second, what is the exact kind of programming power that we have unleashed? Does it lead to interesting programs that we have not been able to write before? We have outlined some speculative applications for intensional recursion in [23, §§1–2]. Is iPCF a useful tool when it comes to attacking these?

Acknowledgements

The author would like to thank Mario Alvarez-Picallo for their endless conversations on types and metaprogramming, Neil Jones for his careful reading and helpful comments, and Samson Abramsky for suggesting the topic of intensionality. This work was supported by the EPSRC (award reference 1354534).

References

[1] Samson Abramsky. Intensionality, Definability and Computation. In Alexandru Baltag and Sonja Smets, editors, *Johan van Benthem on Logic and Information Dynamics*, pages 121–142. Springer-Verlag, 2014.

[2] Leonard M. Adleman. An Abstract Theory of Computer Viruses. In *Advances in Cryptology - CRYPTO' 88*, volume 403 of *Lecture Notes in Computer Science*, pages 354–374. Springer New York, New York, NY, 1990.

[3] Andrew Graham Barber. Dual Intuitionistic Linear Logic. Technical report, ECS-LFCS-96-347, Laboratory for Foundations of Computer Science, University of Edinburgh, 1996.

[4] Henk Barendregt. *Lambda Calculus: Its Syntax and Semantics*. North-Holland, Amsterdam, 1984.

[5] Alan Bawden. Quasiquotation in LISP. In *Proceedings of the 6th ACM SIGPLAN Workshop on Partial Evaluation and Semantics-Based Program Manipulation (PEPM '99)*, 1999.

[6] Lars Birkedal, Rasmus Møgelberg, Jan Schwinghammer, and Kristian Støvring. First steps in synthetic guarded domain theory: step-indexing in the topos of trees. *Logical Methods in Computer Science*, 8(4):1–45, oct 2012.

[7] George S. Boolos. *The Logic of Provability*. Cambridge University Press, Cambridge, 1994.

[8] Olivier Danvy and Karoline Malmkjaer. Intensions and extensions in a reflective tower. In *Proceedings of the 1988 ACM conference on LISP and functional programming (LFP '88)*, pages 327–341, New York, New York, USA, 1988. ACM Press.

[9] Rowan Davies and Frank Pfenning. A modal analysis of staged computation. *Journal of the ACM*, 48(3):555–604, 2001.

[10] Daniel P. Friedman and Mitchell Wand. Reification: Reflection without metaphysics. In *Proceedings of the 1984 ACM Symposium on LISP and functional programming (LFP '84)*, pages 348–355, New York, New York, USA, 1984. ACM Press.

[11] Murdoch J. Gabbay and Aleksandar Nanevski. Denotation of contextual modal type theory (CMTT): Syntax and meta-programming. *Journal of Applied Logic*, 11(1):1–29, mar 2013.

[12] Paul Graham. *On LISP: Advanced Techniques for Common LISP*. Prentice Hall, 1993.

[13] Carl A. Gunter. *Semantics of programming languages: structures and techniques*. Foundations of Computing. The MIT Press, 1992.

[14] Torben Amtoft Hansen, Thomas Nikolajsen, Jesper Larsson Träff, and Neil D. Jones. Experiments with Implementations of Two Theoretical Constructions. In *Proceedings of the Symposium on Logical Foundations of Computer Science: Logic at Botik '89*, pages 119–133, London, UK, 1989. Springer-Verlag.

[15] Neil D. Jones. Computer Implementation and Applications of Kleene's S-M-N and Recursion Theorems. In Yiannis N Moschovakis, editor, *Logic from Computer Science: Proceedings of a Workshop Held November 13-17, 1989 [at MSRI]*, volume 21 of *Mathematical Sciences Research Institute Publications*, pages 243–263. Springer New York, 1992.

[16] Neil D. Jones. An introduction to partial evaluation. *ACM Computing Surveys*, 28(3):480–503, 1996.

[17] Neil D. Jones. *Computability and Complexity: From a Programming Perspective*. Foundations of Computing. MIT Press, 1997.

[18] Neil D. Jones. A Swiss Pocket Knife for Computability. *Electronic Proceedings in Theoretical Computer Science*, 129:1–17, sep 2013.

[19] Neil D. Jones, Carsten K. Gomard, and Peter Sestoft. *Partial Evaluation and Automatic Program Generation*. Prentice Hall International, 1993.

[20] G. A. Kavvos. The Many Worlds of Modal λ-calculi: I. Curry-Howard for Necessity, Possibility and Time. *CoRR*, 2016.

[21] G. A. Kavvos. Dual-context calculi for modal logic. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. IEEE, 2017.

[22] G. A. Kavvos. On the Semantics of Intensionality. In Javier Esparza and Andrzej S. Murawski, editors, *Proceedings of the 20th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS)*, volume 10203 of *Lecture Notes in Computer Science*, pages 550–566. Springer-Verlag Berlin Heidelberg, 2017.

[23] Georgios Alexandros Kavvos. *On the Semantics of Intensionality and Intensional Recursion*. DPhil thesis, University of Oxford, 2017.

[24] Stephen C. Kleene. On notation for ordinal numbers. *The Journal of Symbolic Logic*, 3(04):150–155, 1938.

[25] Tadeusz Litak. Constructive Modalities with Provability Smack. In Guram Bezhanishvili, editor, *Leo Esakia on duality in modal and intuitionistic logics*, pages 179–208. Springer, 2014.

[26] John R. Longley and Dag Normann. *Higher-Order Computability*. Theory and Applications of Computability. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

[27] John C. Mitchell. *Foundations for programming languages*. Foundations of Computing. The MIT Press, 1996.

[28] Hiroshi Nakano. A modality for recursion. *Proceedings Fifteenth Annual IEEE Symposium on Logic in Computer Science (Cat. No.99CB36332)*, 2000.

[29] Aleksandar Nanevski. Meta-programming with names and necessity. *ACM SIGPLAN Notices*, 37:206–217, 2002.

[30] Aleksandar Nanevski and Frank Pfenning. Staged computation with names and necessity. *Journal of Functional Programming*, 15(06):893, 2005.

[31] Frank Pfenning and Rowan Davies. A judgmental reconstruction of modal logic. *Mathematical Structures in Computer Science*, 11(4):511–540, 2001.

[32] Gordon D. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5(3):223–255, 1977.

[33] Carsten Schürmann, Joëlle Despeyroux, and Frank Pfenning. Primitive recursion for higher-order abstract syntax. *Theoretical Computer Science*, 266(1-2):1–57, 2001.

[34] Dana S. Scott. A type-theoretical alternative to ISWIM, CUCH, OWHY. *Theoretical Computer Science*, 121(1-2):411–440, 1993.

[35] Brian Cantwell Smith. Reflection and Semantics in LISP. In *Proceedings of the 11th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL '84)*, pages 23–35, New York, New York, USA, 1984. ACM Press.

[36] Thomas Streicher. *Domain-theoretic Foundations of Functional Programming*. World Scientific, 2006.

[37] Walid Taha and Tim Sheard. MetaML and multi-stage programming with explicit annotations. *Theoretical Computer Science*, 248(1-2):211–242, 2000.

[38] M. Takahashi. Parallel Reductions in λ-Calculus. *Information and Computation*, 118(1):120–127, apr 1995.

[39] Takeshi Tsukada and Atsushi Igarashi. A logical foundation for environment classifiers.

Logical Methods in Computer Science, 6(4):1–43, 2010.

[40] Aldo Ursini. A modal calculus analogous to K4W, based on intuitionistic propositional logic. *Studia Logica*, 38(3):297–311, 1979.

[41] Mitchell Wand. The Theory of Fexprs is Trivial. *LISP and Symbolic Computation*, 10(3):189–199, 1998.

[42] Mitchell Wand and Daniel P. Friedman. The mystery of the tower revealed: A nonreflective description of the reflective tower. *Lisp and Symbolic Computation*, 1(1):11–38, jun 1988.

Received 14 October 2017