A Metric Model of Lambda Calculus with Guarded Recursion

Lars Birkedal
IT University of Copenhagen
birkedal@itu.dk

Jan Schwinghammer
Saarland University
jan@ps.uni-saarland.de

Kristian Støvring
IT University of Copenhagen
kss@itu.dk

Abstract

We give a model for Nakano's typed lambda calculus with guarded recursive definitions in a category of metric spaces. By proving a computational adequacy result that relates the interpretation with the operational semantics, we show that the model can be used to reason about contextual equivalence.

1 Introduction

Recent work in semantics of programming languages and program logics has suggested that there might be a connection between metric spaces and Nakano's typed lambda calculus with guarded recursive definitions [7]. In this paper we show that is indeed the case by developing a model of Nakano's calculus in a category of metric spaces and by showing that the model is computationally adequate. The latter is done with the use of a logical relation, whose existence is non-trivial because of the presence of recursive types. We define the relation in a novel indexed way, inspired by step-indexed models for operational semantics [2].

For space reasons, we can only briefly hint at some of the motivating applications in semantics:

In [4], Birkedal et al. gave a model of a programming language with recursive types, impredicative polymorphism and general ML-style references. The model is a Kripke model, where the set of worlds is a recursively defined metric space (with the recursion variable in negative position). A simplified variant of the recursive equation can be formulated in a version of Nakano's calculus.

Pottier [9] has recently suggested to use Nakano's calculus as a calculus of kinds in an extension of F_ω with recursive kinds, which can serve as a target calculus for a store-passing translation of a language with ML-style references. The model we present here can be extended to a model of F_ω with recursive kinds by indexing complete uniform per's over metric spaces.

Hinze [5] has shown, in the context of Haskell streams, how the uniqueness of fixed points of contractive functions in an operational setting can be used for many applications. Nakano's calculus can be used to formalize such arguments and in our model unique fixed points for contractive functions do exist.

For reasoning about imperative interactive programs, Krishnaswami et al. [6] related an efficient imperative implementation to a simple model of functional reactive programming. In current unpublished work by Krishnaswami on extending this to higher-order programs, metric spaces are being used to model the functional reactive programs to capture that all uses of recursion are guarded. A logical relation is used to relate the functional reactive programs and the imperative implementation thereof; that logical relation is defined using ideas that seem similar to those we are using in our logical relation for the adequacy proof in Section 3.

In the remainder of the paper, we present (our version of) Nakano's calculus (Section 2), define the denotational semantics and prove that it is computationally adequate (Section 3), briefly discuss some closely related work (Section 4), and conclude with some comments about future work.

2 Nakano's Lambda Calculus

In this section we recall the typed lambda calculus of Nakano [7]. It allows certain forms of guarded recursive definitions, and tracks guardedness via a modality in the type system. We shall show that this

$$\frac{}{\Gamma, x{:}\tau \vdash x : \tau} \qquad \frac{}{\Gamma \vdash \underline{n} : \mathit{Int}} \qquad \frac{\Gamma, x{:}\tau \vdash t : \sigma}{\Gamma \vdash \lambda x{:}\tau.t : \tau \to \sigma} \qquad \frac{\Gamma \vdash t_1 : \bullet^n(\tau \to \sigma) \qquad \Gamma \vdash t_2 : \bullet^n \tau}{\Gamma \vdash t_1\, t_2 : \bullet^n \sigma}$$

$$\frac{\Gamma \vdash t_1 : \tau_1 \qquad \Gamma \vdash t_2 : \tau_2}{\Gamma \vdash (t_1, t_2) : \tau_1 \times \tau_2} \qquad \frac{\Gamma \vdash t : \bullet^n(\tau_1 \times \tau_2)}{\Gamma \vdash \mathit{proj}_i(t) : \bullet^n \tau_i} \qquad \frac{\bullet\Gamma \vdash t : \bullet\tau}{\Gamma \vdash t : \tau} \qquad \frac{\Gamma \vdash t : \bullet\tau_{i,j}}{\Gamma \vdash \mathit{In}_{i,j}(t) : ty_i}$$

$$\frac{\Gamma \vdash t : ty_i \qquad \Gamma, x_1{:}\bullet\tau_{i,1} \vdash t_1 : \tau \qquad \ldots \qquad \Gamma, x_{i_k}{:}\bullet\tau_{i,k_i} \vdash t_{k_i} : \tau}{\Gamma \vdash \mathit{case}\ t\ \mathit{of}\ \mathit{In}_{i,1}(x_1) \Rightarrow t_1 \mid \ldots \mid \mathit{In}_{i,k_i}(x_{k_i}) \Rightarrow t_{k_i} : \tau} \qquad \frac{\Gamma \vdash t : \tau \qquad \vdash \tau \leq \sigma}{\Gamma \vdash t : \sigma}$$

Figure 1: Typing rules of Nakano's lambda calculus

$$\frac{\vdash \tau \leq \sigma}{\vdash \bullet\tau \leq \bullet\sigma} \qquad \frac{\vdash \tau' \leq \tau \qquad \vdash \sigma \leq \sigma'}{\vdash \tau \to \sigma \leq \tau' \to \sigma'} \qquad \frac{\vdash \tau \leq \tau' \qquad \vdash \sigma \leq \sigma'}{\vdash \tau \times \sigma \leq \tau' \times \sigma'} \qquad \frac{}{\vdash \tau \leq \bullet\tau}$$

$$\frac{}{\vdash \tau \to \sigma \leq \bullet\tau \to \bullet\sigma} \qquad \frac{}{\vdash \bullet\tau \to \bullet\sigma \leq \bullet(\tau \to \sigma)} \qquad \frac{}{\vdash \tau \times \sigma \leq \bullet\tau \times \bullet\sigma} \qquad \frac{}{\vdash \bullet\tau \times \bullet\sigma \leq \bullet(\tau \times \sigma)}$$

Figure 2: Subtyping

modality can be understood semantically as a scaling operation of the distances in a metric space.

The precise language considered below differs from that presented by Nakano in [7] in that we assume a fixed number of "global" (possibly mutually recursive) datatype declarations. In contrast, Nakano includes equi-recursive types of the form $\mu X.A$ (subject to some well-formedness conditions that ensure formal contractiveness, and a type equivalence relation).

Syntax and typing. We assume that there is a fixed number of type identifiers $ty_1, \ldots, ty_n$, each associated with a recursive data type equation in the style of Haskell or ML that specifies constructors $\mathit{In}_{i,1}, \ldots, \mathit{In}_{i,k_i}$:

$$\mathit{data}\ ty_i = \mathit{In}_{i,1}\ \mathit{of}\ \tau_{i,1} \mid \ldots \mid \mathit{In}_{i,k_i}\ \mathit{of}\ \tau_{i,k_i}\ . \tag{1}$$

The types $\tau_{i,1}, \ldots, \tau_{i,k_i}$ that appear on the right hand side of (1) are built up from ground types b (for simplicity, we restrict ourselves to Int) and the identifiers $ty_1, \ldots, ty_n$, using product and function space and the unary type constructor '$\bullet$' (referred to as 'later' modality in recent work on step-indexed semantics [3]). One concrete instance of (1) is a type of infinite sequences of τ's: $\mathit{data}\ \mathit{seq}_\tau = \mathit{Cons}\ \mathit{of}\ \tau \times \mathit{seq}_\tau$. In general, these declarations need not be monotone, however, and the identifiers ty_i may also appear in negative positions. For instance, Nakano [7, Ex. 1] considers the type $\mathit{data}\ u = \mathit{Fold}\ \mathit{of}\ u \to \tau$. With this type, the fixed point combinator Y from untyped lambda calculus has type $(\bullet\tau \to \tau) \to \tau$; thus, it is not necessary to include a primitive for recursion.

Terms and their types are given in Figure 1. The type system uses the subtyping relation defined in Figure 2 (omitting the reflexivity and transitivity rules). Intuitively, the type $\bullet\tau$ may be understood as the set of values of type τ that need to be guarded by a constructor. This intuition explains the two rules for typing constructor applications and pattern matching in Figure 1: The application of a constructor lets one remove the outermost modal operator, and it also folds the recursive type. Conversely, the *case* construct removes a constructor, and the typing rule records this fact by applying the modality to the type of the bound variables. The typing rules for function application and component projections are also generalized to take care of the modal operator. We write $\mathit{Tm}(\tau)$ for the set of closed terms of type τ.

Operational semantics and contextual equivalence. The operational semantics consists of the usual (deterministic, call-by-name) reduction rules of lambda calculus with constructor types. One does not reduce under constructors: every term of the form $In_{i,j}(t)$ is a value. We take contextual equivalence as our notion of program equivalence. Formally, we observe the values that terms yield when plugged into closing, base-type valued contexts: Let us write $\vdash C : (\Gamma' \triangleright \tau') \multimap (\Gamma \triangleright \tau)$ for a context C if $\Gamma \vdash C[t] : \tau$ whenever $\Gamma' \vdash t : \tau'$. Then t_1 and t_2 are *contextually equivalent*, written $\Gamma \vdash t_1 \simeq t_2 : \tau$, if $\Gamma \vdash t_1 : \tau$ and $\Gamma \vdash t_2 : \tau$, and for all $\vdash C : (\Gamma \triangleright \tau) \multimap (\emptyset \triangleright Int)$ and $n \in \mathbb{N}$, $C[t_1] \xrightarrow{*} \underline{n} \Leftrightarrow C[t_2] \xrightarrow{*} \underline{n}$.

3 Denotational Semantics

We give a denotational semantics of types and terms in the category of complete, 1-bounded ultrametric spaces and non-expansive maps between them.

Ultrametric spaces. We briefly recall some basic definitions and results about metric spaces [10]. A *1-bounded ultrametric space* (X,d) is a metric space where the distance function $d : X \times X \to \mathbb{R}$ takes values in the closed interval $[0,1]$ and satisfies the "strong" triangle inequality $d(x,y) \leq \max\{d(x,z),d(z,y)\}$. A metric space is *complete* if every Cauchy sequence has a limit. Because of the use of max in the above inequality, rather than '+' as for the weaker triangle inequality of metric spaces in general, a sequence $(x_n)_{n\in\mathbb{N}}$ in an ultrametric space (X,d) is a Cauchy sequence if for every $\varepsilon > 0$ there exists $n \in \mathbb{N}$ such that $d(x_k,x_{k+1}) \leq \varepsilon$ for every $k \geq n$.

A function $f : X_1 \to X_2$ between metric spaces (X_1,d_1), (X_2,d_2) is *non-expansive* if $d_2(f(x),f(y)) \leq d_1(x,y)$ for all $x,y \in X_1$. It is *contractive* if there exists some $\delta < 1$ such that $d_2(f(x),f(y)) \leq \delta \cdot d_1(x,y)$ for all $x,y \in X_1$. The complete, 1-bounded, non-empty, ultrametric spaces and non-expansive functions between them form a Cartesian closed category $CBUlt_{ne}$. Products are given by the set-theoretic product where the distance is the maximum of the componentwise distances. The exponential $(X_1,d_1) \to (X_2,d_2)$ has the set of non-expansive functions from (X_1,d_1) to (X_2,d_2) as underlying set, and the distance function is given by $d_{X_1\to X_2}(f,g) = \sup\{d_2(f(x),g(x)) \mid x \in X_1\}$.

A functor $F : (CBUlt_{ne}^{\mathrm{op}})^n \times CBUlt_{ne}^n \longrightarrow CBUlt_{ne}$ is *locally non-expansive* if $d(F(f,g),F(f',g')) \leq \max\{d(f_1,f_1'),d(g_1,g_1'),\ldots,d(f_n,f_n'),d(g_n,g_n')\}$ for all non-expansive $f = (f_1,\ldots,f_n)$, $f' = (f_1',\ldots,f_n')$, $g = (g_1,\ldots,g_n)$ and $g' = (g_1',\ldots g_n')$. It is *locally contractive* if there exists some $\delta < 1$ such that $d(F(f,g),F(f',g')) \leq \delta \cdot \max\{d(f_1,f_1'),d(g_1,g_1'),\ldots,d(f_n,f_n'),d(g_n,g_n')\}$. Note that the factor δ is the same for all hom-sets. By multiplication of the distances of (X,d) with a non-negative shrinking factor $\delta < 1$, one obtains a new ultrametric space, $\delta \cdot (X,d) = (X,d')$ where $d'(x,y) = \delta \cdot d(x,y)$. By shrinking, a locally non-expansive functor F yields a locally contractive functor $(\delta \cdot F)(X,Y) = \delta \cdot (F(X,Y))$.

It is well-known that one can solve recursive domain equations in $CBUlt_{ne}$ by an adaptation of the inverse-limit method from classical domain theory:

Theorem 1 (America-Rutten [1])**.** *Let $F_i : (CBUlt_{ne}^{\mathrm{op}})^n \times CBUlt_{ne}^n \longrightarrow CBUlt_{ne}$ be locally contractive functors for $i = 1,\ldots,n$. Then there exists a unique (up to isomorphism) $X = (X_k,d_k)_k \in CBUlt_{ne}^n$ such that $(X_i,d_i) \cong F_i(X,X)$ for all i.*

The metric spaces we consider below are *bisected*, meaning that all non-zero distances are of the form 2^{-n} for some natural number $n \geq 0$. When working with bisected metric spaces, the notation $x \stackrel{n}{=} y$ means that $d(x,y) \leq 2^{-n}$. Each relation $\stackrel{n}{=}$ is an equivalence relation because of the ultrametric inequality; we are therefore justified in referring to the relation $\stackrel{n}{=}$ as "n-equality." Since the distance of a bisected metric space is bounded by 1, the relation $x \stackrel{0}{=} y$ always holds. Moreover, the n-equalities become finer as n increases, and $x = y$ if $x \stackrel{n}{=} y$ holds for all n. Finally, a function $f : X_1 \to X_2$ between bisected metric spaces is non-expansive iff $x_1 \stackrel{n}{=} x_1' \Rightarrow f(x_1) \stackrel{n}{=} f(x_1')$, and contractive iff $x_1 \stackrel{n}{=} x_1' \Rightarrow f(x_1) \stackrel{n+1}{=} f(x_1')$ for all n.

Denotational semantics of Nakano's lambda calculus. We can now define the interpretation of types and terms in $CBUlt_{ne}$. More precisely, by induction on the type τ we define locally non-expansive functors $F_\tau : (CBUlt_{ne}^{op})^n \times CBUlt_{ne}^n \longrightarrow CBUlt_{ne}$, by separating positive and negative occurrences of the identifiers $ty_1, \ldots, ty_n$ in τ:

$$\begin{aligned} F_{Int}(X,Y) &= (\mathbb{Z}, d), \text{ where } d \text{ is the discrete metric} \\ F_{ty_i}(X,Y) &= Y_i \\ F_{\tau_1 \times \tau_2}(X,Y) &= F_{\tau_1}(X,Y) \times F_{\tau_2}(X,Y) \\ F_{\tau_1 \to \tau_2}(X,Y) &= F_{\tau_1}(Y,X) \to F_{\tau_2}(X,Y) \\ F_{\bullet\tau}(X,Y) &= \tfrac{1}{2} \cdot F_\tau(X,Y) \end{aligned}$$

Now consider the functors $F_i : (CBUlt_{ne}^{op})^n \times CBUlt_{ne}^n \longrightarrow CBUlt_{ne}$ for $i = 1, \ldots, n$, defined by

$$F_i(X,Y) = \tfrac{1}{2} \cdot F_{\tau_{i,1}}(X,Y) + \ldots + \tfrac{1}{2} \cdot F_{\tau_{i,k_i}}(X,Y) \; . \tag{2}$$

Because of the shrinking factor $1/2$ in (2), each F_i is in fact locally contractive. Theorem 1 therefore gives a unique fixed point D with $\iota_i : F_i(D,D) \cong D_i$ for all i. We use D to give the semantics of types:

$$[\![\tau]\!] \stackrel{def}{=} F_\tau(D,D) \; .$$

Example 2 (Interpretation of streams). On streams, the semantics yields the "natural" metric. In fact, since $[\![seq_\tau]\!] \cong \frac{1}{2} \cdot ([\![\tau]\!] \times [\![seq_\tau]\!])$ we have

$$d_{[\![seq_\tau]\!]}(s_1 s_2 \ldots,\, s_1' s_2' \ldots) \leq 2^{-n} \;\Leftrightarrow\; d_{[\![\tau]\!]}(s_1, s_1') \leq 2^{-(n-1)} \;\wedge\; d_{[\![seq_\tau]\!]}(s_2 s_3 \ldots,\, s_2' s_3' \ldots) \leq 2^{-(n-1)} \; .$$

In particular, because of the discrete metric on $[\![Int]\!]$, for seq_{Int} this means $s \stackrel{n}{=} s'$ holds if $s_1 = s_1', \ldots, s_{n-1} = s_{n-1}'$, i.e., if the common prefix of s and s' has at least length $n-1$, and $s = s'$ if $s_i = s_i'$ for all $i \in \mathbb{N}$.

For each subtyping derivation Δ of a judgement $\vdash \tau \leq \sigma$ we define a corresponding coercion function, i.e., a non-expansive function $[\![\Delta]\!] : [\![\tau]\!] \to [\![\sigma]\!]$. First note that, for each of the 5 subtyping axioms $\vdash \tau \leq \sigma$ in Figure 2 (as well as the reflexivity axiom), the underlying sets of $[\![\tau]\!]$ and $[\![\sigma]\!]$ are equal. Thus we can define $[\![\Delta]\!]$ as the identity on the underlying sets, and it is easy to check that Δ is in fact non-expansive.[1] If Δ is obtained from Δ_1 and Δ_2 by an application of the transitivity rule, then $[\![\Delta]\!]$ is defined as the composition of $[\![\Delta_1]\!]$ and $[\![\Delta_2]\!]$. Finally, for each of the 3 structural rules in Figure 2, we use the functorial property of the respective type constructor to define $[\![\Delta]\!]$ from the coercions of the subderivations. It follows by induction that the coercion determined from any derivation Δ of $\vdash \tau \leq \sigma$ is the identity on the underlying set of $[\![\tau]\!]$, and hence independent of Δ.

Each term $x_1{:}\tau_1, \ldots, x_n{:}\tau_n \vdash t : \tau$ denotes a non-expansive function

$$[\![t]\!] : [\![\tau_1]\!] \times \ldots \times [\![\tau_n]\!] \to [\![\tau]\!]$$

which is defined by induction on the typing derivation. The cases of lambda abstraction, application, pairing and projections are given in terms of the cartesian closed structure on $CBUlt_{ne}$. From the definition of the type interpretation it follows that ι_i is an isomorphism between $\frac{1}{2} \cdot [\![\tau_{i,1}]\!] + \ldots + \frac{1}{2} \cdot [\![\tau_{i,k_i}]\!]$ and $[\![ty_i]\!]$. Together with the injections from $[\![\bullet\, \tau_{i,j}]\!] = \frac{1}{2} \cdot [\![\tau_{i,j}]\!]$ into the sum, this isomorphism is used to interpret the constructors and pattern matching of the calculus:

$$[\![In_{i,j}(t)]\!](\vec{a}) = (\iota_i \circ \mathrm{in}_j \circ [\![t]\!])(\vec{a}) \quad \text{and} \quad [\![\mathit{case}\; t \;\mathit{of}\; In_{i,1}(x_1) \Rightarrow t_1 \mid \ldots \mid In_{i,k_i}(x_{k_i}) \Rightarrow t_{k_i}]\!](\vec{a}) = [\![t_j]\!](\vec{a}, a)$$

where $\iota_i^{-1}([\![t]\!](\vec{a})) = \mathrm{in}_j(a)$ for some $1 \leq j \leq k_i$ and $a \in [\![\tau_{i,j}]\!]$. We obtain a model that is sound:

[1] Note however that even though the underlying sets are the same, the inclusion of $[\![\bullet(\tau \to \sigma)]\!]$ into $[\![\bullet\,\tau \to \bullet\,\sigma]\!]$ fails to be non-expansive. Thus we cannot have $\bullet(\tau \to \sigma) \leq \bullet\,\tau \to \bullet\,\sigma$ in general.

Theorem 3 (Soundness). *If $\emptyset \vdash t : \tau$ and $t_1 \to t_2$ then $[\![t_1]\!] = [\![t_2]\!]$ in $[\![\tau]\!]$.*

Remark 4 (Recursive definitions). In [7, Ex. 1] Nakano shows that the fixed point combinator Y from untyped lambda calculus can be represented by $\lambda f.\Delta_f\,(\mathit{Fold}(\Delta_f))$ where $\Delta_f = f(\mathit{case}\,x\,\mathit{of}\,\mathit{Fold}(y) \Rightarrow yx)$ and the data type $\mathit{data}\,u = \mathit{Fold}\,\mathit{of}\,u \to \tau$ is assumed, and that this term has type $(\bullet\,\tau \to \tau) \to \tau$. Note that in the above interpretation, where the modality is interpreted by scaling the distances by $1/2$, the type $\bullet\,\tau \to \tau$ denotes the set of all non-expansive functions $\frac{1}{2} \cdot [\![\tau]\!] \to [\![\tau]\!]$, or equivalently (by the bisectedness of $[\![\tau]\!]$) the contractive functions on $[\![\tau]\!]$. Such functions have a unique fixed point in $[\![\tau]\!]$ by the Banach fixed point theorem, and we can show that Y indeed returns this fixed point since $f([\![Y]\!]\,f) = [\![Y]\!]\,f$.

As an alternative to this coding, we could introduce a recursion operator $\mathit{rec} : (\bullet\,\tau \to \tau) \to \tau$ for each τ as a primitive, with reduction rule $\mathit{rec}\,t \to t\,(\mathit{rec}\,t)$, and use the above observation for its interpretation.

Computational adequacy. We now relate the interpretation of lambda terms in the metric model and their operational behaviour. In particular, we prove that the semantics is sound for reasoning about contextual equivalence: *if two terms have the same denotation then they are contextually equivalent*.

The general idea of the adequacy proof is standard: the universal quantification over contexts prevents a direct inductive proof, so we use the compositionality of the denotational semantics and construct a *(Kripke) logical relation* between semantics and syntax. More precisely, we consider the family $(R^k_\tau)^k_\tau$ of relations indexed by types τ and natural numbers k, where $R^k_\tau \subseteq [\![\tau]\!] \times \mathit{Tm}(\tau)$ is given by:

$$\begin{aligned}
n\,R^k_{\mathit{Int}}\,t &\;\Leftrightarrow\; t \xrightarrow{*} \underline{n} \\
(a_1,a_2)\,R^k_{\tau_1\times\tau_2}\,t &\;\Leftrightarrow\; a_1\,R^k_{\tau_1}\,\mathit{proj}_1(t) \;\wedge\; a_2\,R^k_{\tau_2}\,\mathit{proj}_2(t) \\
f\,R^k_{\tau_1\to\tau_2}\,t &\;\Leftrightarrow\; \forall j,a_1,t_1.\; j \le k \;\wedge\; a_1\,R^j_{\tau_1}\,t_1 \;\Rightarrow\; f a_1\,R^j_{\tau_2}\,t\,t_1 \\
a\,R^k_{\bullet\tau}\,t &\;\Leftrightarrow\; k > 0 \;\Rightarrow\; a\,R^{k-1}_{\tau}\,t \\
a\,R^k_{ty_i}\,t &\;\Leftrightarrow\; \exists a',t'.\; a = (\iota_i \circ \mathrm{in}_j)(a') \;\wedge\; t \xrightarrow{*} \mathrm{In}_{i,j}(t') \;\wedge\; a'\,R^k_{\bullet\tau_{i,j}}\,t'
\end{aligned}$$

Note that it is the natural number index which lets us define the relations R^k_τ inductively in the presence of recursive types ty_i. We prove the 'fundamental property' by induction on typing derivations:

Proposition 5. *If $\Gamma \vdash t : \tau$, $k \in \mathbb{N}$, and $a_i\,R^k_{\tau_i}\,t_i$ for all $x_i{:}\tau_i$ in Γ, then $[\![t]\!]\,(\vec{a})\,R^k_\tau\,t[\vec{x}{:=}\vec{t}]$.*

Proof sketch. By induction on the derivation of $\Gamma \vdash t : \tau$. The proof uses the closure of the relations under the operational semantics, the downwards closure (Kripke monotonicity) in k, and the closure under the coercions determined by the subtyping relation. (See Section A.3 for details.) □

Now adequacy is easily proved. In fact, given $\Gamma \vdash t_1 : \tau$, $\Gamma \vdash t_2 : \tau$ and $\vdash C : (\Gamma \triangleright \tau) \multimap (\emptyset \triangleright \mathit{Int})$, and $C[t_1] \xrightarrow{*} \underline{n}$ for some $n \in \mathbb{N}$, then $[\![C[t_2]]\!] = [\![C[t_1]]\!] = n$ follows from the assumption $[\![t_1]\!] = [\![t_2]\!]$ and Theorem 3. By Proposition 5, $[\![C[t_2]]\!]\,R^k_{\mathit{Int}}\,C[t_2]$, and therefore $n\,R^k_{\mathit{Int}}\,C[t_2]$ holds. But by definition this means $C[t_2] \xrightarrow{*} \underline{n}$, and with a symmetric argument the claim $[\![t_1]\!] = [\![t_2]\!] \;\Rightarrow\; \Gamma \vdash t_1 \simeq t_2 : \tau$ follows.

Apart from using the semantics directly to reason about contextual equivalence, we can also use computational adequacy to derive more abstract proof principles from it. For instance, by the characterization of the metric on $\mathit{seq}_{\mathit{Int}}$ given in Example 2 and the fact that there are closed terms $\mathit{get}_n : \mathit{seq}_{\mathit{Int}} \to \bullet^n \mathit{Int}$ that yield the n-th element of a sequence, we obtain a variant of Bird and Wadler's *take lemma*: if $\mathit{get}_n\,t_1$ and $\mathit{get}_n\,t_2$ reduce to the same value, for each $n \in \mathbb{N}$, then $\emptyset \vdash t_1 \simeq t_2 : \mathit{seq}_{\mathit{Int}}$. Similarly, we can exploit the uniqueness of fixed points of contractive equations also in the operational setting: if there exists a closed term $f : \bullet\,\tau \to \tau$ such that $f\,t_1$ and t_1 are convertible, and also $f\,t_2$ and t_2 are convertible, then $\emptyset \vdash t_1 \simeq t_2 : \tau$. See, for instance, Hinze's article [5] for numerous applications of such a unique fixed point principle phrased in the context of Haskell streams, and Pottier's work [9] for a similar application to establish type equivalences in the context of F_ω with recursive kinds.

4 Related Work

Metric semantics of PCF. Escardó [**?**] presents a metric model of PCF. One can, almost,[2] factor his interpretation of PCF into two parts: (1) a syntactic translation from PCF to Nakano's calculus, extended with constants for integer operations and a basic type of booleans, and (2) the metric interpretation of Nakano's calculus presented here.

The basic idea of the syntactic translation is that a potentially divergent PCF term of integer type is translated into a term of the recursive type $Int_s = Done\ of\ Int \mid Next\ of\ Int_s$. After evaluating such a term, one either obtains an actual integer answer, *Done(n)*, or a new term that one can evaluate in the hope of eventually getting an integer answer. Function types in PCF are translated to function types.

Now, for all types τ that are used to interpret PCF types, one defines a term $delay : \bullet\tau \to \tau$; this term is used to define the fixed-point combinator of PCF in terms of the fixed-point combinator of our calculus (Remark 4). See Escardó for more details. Intuitively, the idea is that unfolding a recursive definition takes a "computation step," which will be visible as an extra *Next* in the final answer of type Int_s. Escardó shows an adequacy result which implies that the semantics of a PCF term does indeed match the number of unfoldings of the fixed-point combinator; the same information can be obtained from our adequacy proof using the definition of the logical relation on recursive types. Escardó gives two adequacy proofs, the first of which uses a Kripke logical relation as in our adequacy proof.

Since Escardó considers PCF, he does not treat recursively defined types, as we do here.

5 Conclusion and Future Work

We have presented a computationally adequate metric model of lambda calculus with guarded recursion.

We conjecture that an adaptation of the present model can be used to give a model for focusing proof systems with recursive types [11].

References

[1] Pierre America and Jan J. M. M. Rutten. Solving reflexive domain equations in a category of complete metric spaces. *Journal of Computer and System Sciences*, 39(3):343–375, 1989.

[2] Andrew W. Appel and David McAllester. An indexed model of recursive types for foundational proof-carrying code. *ACM Transactions on Programming Languages and Systems*, 23(5):657–683, September 2001.

[3] Andrew W. Appel, Paul-André Melliès, Christopher D. Richards, and Jérôme Vouillon. A very modal model of a modern, major, general type system. In *POPL*, pages 109–122, 2007.

[4] Lars Birkedal, Kristian Støvring, and Jacob Thamsborg. Realizability semantics of parametric polymorphism, general references, and recursive types. In *FOSSACS'09*, pages 456–470, 2009.

[5] Ralf Hinze. Functional pearl: streams and unique fixed points. In *ICFP*, pages 189–200. ACM, 2008.

[6] N. Krishnaswami, L. Birkedal, and J. Aldrich. Verifying event-driven programs using ramified frame properties. In *Proceedings of TLDI'2010*, 2010.

[7] Hiroshi Nakano. A modality for recursion. In *LICS*, pages 255–266, 2000.

[8] Andrew M. Pitts. Relational properties of domains. *Information and Computation*, 127:66–90, 1996.

[9] François Pottier. A typed store-passing translation for general references. Submitted, April 2010.

[10] Michael B. Smyth. Topology. In *Handbook of Logic in Computer Science*, volume 1. Oxford University Press, 1992.

[2] Due to the syntactic restrictions on recursive types in our variant of Nakano's calculus, the metrics on PCF ground types (and hence on all types) differ slightly in our model and Escardó's model.

[11] Noam Zeilberger. *The Logical Basis of Evaluation Order and Pattern-Matching*. PhD thesis, Carnegie Mellon University, 2009.

$$(\lambda x{:}\tau.t)\,t' \to t[x{:=}t']$$
$$proj_i(t_1,t_2) \to t_i$$
$$case\ In_{i,j}(t)\ of\ In_{i,1}(x_1) \Rightarrow t_1 | \ldots | In_{i,k_i}(x_{k_i}) \Rightarrow t_{k_i} \to t_j[x_j{:=}t]$$
$$E[t] \to E[t'] \qquad \text{if } t \to t'$$

where $E ::= [\,] \mid E\,t \mid proj_i E \mid case\ E\ of\ In_{i,1}(x_1) \Rightarrow t_1 | \ldots | In_{i,k_i}(x_{k_i}) \Rightarrow t_{k_i}$ defines the set of evaluation contexts.

Figure 3: Operational semantics

A Proofs

A.1 Operational Semantics and Soundness

Figure 3 gives the reduction relation on (closed) terms. The denotational semantics is sound with respect to this operational semantics.

Theorem 6 (Soundness)**.** *If* $\vdash t : \tau$ *and* $t_1 \to t_2$ *then* $[\![t_1]\!] = [\![t_2]\!]$ *in* $[\![\tau]\!]$.

Proof. By induction on the operational semantics. For instance, if the case of pattern matching we have

$$[\![case\ (In_{i,j}(t))\ of\ In_{i,1}(x_1) \Rightarrow t_1 \mid \ldots \mid In_{i,k_i}(x_{k_i}) \Rightarrow t_{k_i}]\!] = [\![t_j]\!]\,(a)$$

for a such that $\mathrm{in}_j(a) = \iota_i^{-1}([\![In_{i,j}(t)]\!]) = \mathrm{in}_j([\![t]\!])$. Thus, by a substitution lemma,

$$[\![case\ (In_{i,j}(t))\ of\ In_{i,1}(x_1) \Rightarrow t_1 \mid \ldots \mid In_{i,k_i}(x_{k_i}) \Rightarrow t_{k_i}]\!] = [\![t_j]\!]\,([\![t]\!]) = [\![t_j[x_j{:=}t]]\!]\ .$$

The other cases are similar. □

A.2 Coercions for Subtyping

Lemma 7. *Let* $(X,d_X),(Y,d_Y) \in CBUlt_{ne}$*. Then the underlying sets of* $X \to Y$*, of* $\frac{1}{2}\cdot X \to \frac{1}{2}\cdot Y$*, and of* $\frac{1}{2}\cdot(X \to Y)$ *all agree.*

Proof. That the underlying sets of $X \to Y$ and $\frac{1}{2}\cdot(X \to Y)$ agree is immediate from the definition of scaling. Now suppose f is a non-expansive function $X \to Y$, and let $x,y \in \frac{1}{2}\cdot X$ such that $d_{\frac{1}{2}\cdot X}(x,y) \leq 2^{-n}$. Then $d_X(x,y) \leq 2^{-n+1}$. But this means $d_Y(fx,fy) \leq 2^{-n+1}$ and thus $d_{\frac{1}{2}\cdot Y}(fx,fy) \leq 2^{-n}$, i.e., f is also a non-expansive function $\frac{1}{2}\cdot X \to \frac{1}{2}\cdot Y$. The other direction is similar. □

A.3 Logical Relation

Lemma 8. *Suppose* $a\ R^k_\tau\ t$ *and* $t_1 \xrightarrow{*} t \xrightarrow{*} t_2$*. Then* $a\ R^k_\tau\ t_1$ *and* $a\ R^k_\tau\ t_2$

Proof. By induction on τ, using the determinacy of the operational semantics. □

Lemma 9. *Suppose* $a\ R^k_\tau\ t$ *and* $j \leq k$*. Then* $a\ R^j_\tau\ t$.

Proof. By lexicographic induction on (k,τ). □

Lemma 10. *Suppose* $a\ R^k_\tau\ t$ *and* $\vdash \tau \leq \sigma$. *Then* $(\llbracket \tau \leq \sigma \rrbracket\, a)\ R^k_\sigma\ t$.

Proof. By induction on $\vdash \tau \leq \sigma$. □

Theorem 11 (Fundamental property)**.** *If* $\Gamma \vdash t : \tau$, $k \in \mathbb{N}$, *and* $a_i\ R^k_{\tau_i}\ t_i$ *for all* $x_i{:}\tau_i$ *in* Γ, *then* $\llbracket t \rrbracket\,(\vec{a})\ R^k_\tau\ t[\vec{x}{:=}\vec{t}]$.

Proof. By induction on the derivation of $\Gamma \vdash t : \tau$.

- The case of variables is by assumption.
- The case of an integer constant is immediate by definition of R^k_{Int}.
- Consider the abstraction case, where we have to show that $\llbracket \lambda x{:}\tau.t \rrbracket\,(\vec{a})\ R^k_{\tau\to\sigma}\ (\lambda x{:}\tau.t)[\vec{x}{:=}\vec{t}]$. Let $j \leq k$ and $a'\ R^j_\tau\ t'$. By Kripke monotonicity (Lemma 9), $a_i\ R^j_{\tau_i}\ t_i$ for all $x_i{:}\tau_i$ in Γ. Thus, by induction hypothesis, $\llbracket t \rrbracket\,(\vec{a},a')\ R^j_\sigma\ t[\vec{x}{:=}\vec{t},x{:=}t']$. Since $\llbracket \lambda x{:}\tau.t \rrbracket\,(\vec{a})(a') = \llbracket t \rrbracket\,(\vec{a},a')$ and $(\lambda x{:}\tau.t)[\vec{x}{:=}\vec{t}]\,t'$ reduces to $t[\vec{x}{:=}\vec{t},x{:=}t']$, this case follows from the closure of R^j_σ under the operational semantics (Lemma 8).
- Consider the case of the (generalized) application rule. We have to show that

  $$\llbracket t_1\, t_2 \rrbracket\,(\vec{a})\ R^k_{\bullet^n\sigma}\ (t_1\, t_2)[\vec{x}{:=}\vec{t}]\ ,$$

  assuming that $\Gamma \vdash t_1 : \bullet^n(\tau \to \sigma)$ and $\Gamma \vdash t_2 : \bullet^n\tau$. By definition of $R_{\bullet^n\sigma}$ we may assume that $k > n$; otherwise there is nothing to show. But then the induction hypothesis yields $\llbracket t_1 \rrbracket\,(\vec{a})\ R^{k-n}_{\tau\to\sigma}\ t_1[\vec{x}{:=}\vec{t}]$ and $\llbracket t_2 \rrbracket\,(\vec{a})\ R^{k-n}_\tau\ t_2[\vec{x}{:=}\vec{t}]$, so we can conclude $\llbracket t_1\, t_2 \rrbracket\,(\vec{a})\ R^{k-n}_\sigma\ (t_1\, t_2)[\vec{x}{:=}\vec{t}]$ from $\llbracket t_1\, t_2 \rrbracket\,(\vec{a}) = \llbracket t_1 \rrbracket\,(\vec{a})(\llbracket t_2 \rrbracket\,(\vec{a}))$ and the definition of $R^{k-n}_{\tau\to\sigma}$. The result follows by definition of $R_{\bullet^n\sigma}$.
- The cases of the pairing and (generalized) projection rules are similar to abstraction and application, respectively.
- Next, consider the $\bullet$ rule which infers $\Gamma \vdash t : \tau$ from $\bullet\Gamma \vdash t : \bullet\tau$. By definition of the logical relation and the assumption, $a_i\ R^{k+1}_{\bullet\tau_i}\ t_i$ for all $x_i{:}\tau_i$ in Γ. So $\llbracket t \rrbracket\,(\vec{a})\ R^{k+1}_{\bullet\tau}\ t[\vec{x}{:=}\vec{t}]$ by induction hypothesis, from which the required $\llbracket t \rrbracket\,(\vec{a})\ R^k_\tau\ t[\vec{x}{:=}\vec{t}]$ follows.
- Consider the case of a constructor application. We must show that $\llbracket In_{i,j}(t) \rrbracket\,(\vec{a})\ R^k_{ty_i}\ In_{i,j}(t)[\vec{x}{:=}\vec{t}]$ holds. By definition of the relation $R^k_{ty_i}$ and since $\llbracket In_{i,j}(t) \rrbracket\,(\vec{a}) = (\iota_i \circ \mathrm{in}_j)(\llbracket t \rrbracket\,\vec{a})$ it suffices to show that $\llbracket t \rrbracket\,(\vec{a})\ R^k_{\bullet\tau_{i,j}}\ t[\vec{x}{:=}\vec{t}]$. This is exactly what the induction hypothesis yields.
- Consider the case construct. Writing s for $case\ t\ of\ In_{i,1}(x_1) \Rightarrow t_1 \mid \ldots \mid In_{i,k_i}(x_{k_i}) \Rightarrow t_{k_i}$, we must show that $\llbracket s \rrbracket\,(\vec{a})\ R^k_\tau\ s[\vec{x}{:=}\vec{t}]$. Applying the induction hypothesis to $\Gamma \vdash t : ty_i$, there exists a' and t' such that $\llbracket t \rrbracket\,(\vec{a}) = (\iota_i \circ \mathrm{in}_j)(a')$ and $t[\vec{x}{:=}\vec{t}] \xrightarrow{*} In_{i,j}(t')$ and $a'\ R^k_{\bullet\tau_{i,j}}\ t'$. From this, applying the induction hypothesis to $\Gamma, x_j{:}\bullet\tau_{i,j} \vdash t_j : \tau$, we obtain $\llbracket t_j \rrbracket\,(\vec{a},a')\ R^k_\tau\ t_j[\vec{x}{:=}\vec{t},x_j{:=}t']$. Note that $\llbracket s \rrbracket\,(\vec{a}) = \llbracket t_j \rrbracket\,(\vec{a},a')$ and

  $$s[\vec{x}{:=}\vec{t}]\ \xrightarrow{*}\ case\ (In_{i,j}(t'))\ of\ In_{i,1}(x_1) \Rightarrow t_1[\vec{x}{:=}\vec{t}] \mid \ldots \mid In_{i,k_i}(x_{k_i}) \Rightarrow t_{k_i}[\vec{x}{:=}\vec{t}]\ \to\ t_j[\vec{x}{:=}\vec{t},x_j{:=}t']\ ,$$

  hence $\llbracket s \rrbracket\,(\vec{a})\ R^k_\tau\ s[\vec{x}{:=}\vec{t}]$ follows from Lemma 8.
- Finally, the case of the subsumption rule follows from Lemma 10. □