A Model of PCF in Guarded Type Theory

Marco Paviotti[1]

IT University of Copenhagen

Rasmus Ejlers Møgelberg[2]

IT University of Copenhagen

Lars Birkedal[3]

Dept. of Comp. Science, Aarhus University

Abstract

Guarded recursion is a form of recursion where recursive calls are guarded by delay modalities. Previous work has shown how guarded recursion is useful for constructing logics for reasoning about programming languages with advanced features, as well as for constructing and reasoning about elements of coinductive types. In this paper we investigate how type theory with guarded recursion can be used as a metalanguage for denotational semantics useful both for constructing models and for proving properties of these. We do this by constructing a fairly intensional model of PCF and proving it computationally adequate. The model construction is related to Escardo's metric model for PCF, but here everything is carried out entirely in type theory with guarded recursion, including the formulation of the operational semantics, the model construction and the proof of adequacy.

Keywords: Denotational semantics, guarded recursion, type theory, PCF, synthetic domain theory

1 Introduction

Variations of type theory with guarded recursive types and guarded recursively defined predicates have proved useful for giving abstract accounts of operationally-based step-indexed models of programming languages with features that are challenging to model, such as recursive types and general references [1,4], countable nondeterminism [5], and concurrency [11]. Following observations of Nakano [10] and Atkey and McBride [2], guarded type theory also offers an attractive type-based approach to (1) ensuring productivity of definitions of elements of coinductive types [9], and (2) proving properties of elements of coinductive types [6]. One of

[1] Email: mpav@itu.dk

[2] Email: mogel@itu.dk

[3] Email: birkedal@cs.au.dk

the key features of guarded type theory is a modality on types, denotes $\triangleright$ and pronounced later. This modality is used to guard recursive definitions and the intuition is that elements of type $\triangleright A$ are only available one time step from now.

In this paper, we initiate an exploration of the use of guarded type theory for *denotational* semantics and use it to further test guarded type theory. More specifically, we present a model of PCF in guarded dependent type theory. To do so we, of course, need a way to represent possibly diverging computations in type theory. Here we follow earlier work of Escardo [7] and Capretta [**?**] and use a lifting monad L, which allows us to represent a possibly diverging computation of type X by a function into $L(X)$. In Capretta's work, L is defined using coinductive types. Here, instead, we use a guarded recursive type to define L. Using this approach we get a fairly intensional model of PCF which, intuitively keeps track of the number of computation steps, similar to [7]. We show this formally by proving that the denotational model is adequate with respect to a step-counting operational semantics. The definition of this step-counting operational semantics is delicate — to be able to show adequacy the steps in the operational semantics have to correspond to the abstract notion of time-steps used in the guarded type theory via the $\triangleright$ operator. Our adequacy result is related to one given by Escardo in [7]. To show adequacy, we define the operational semantics in guarded type theory and also define a logical relation *in* guarded type theory to relate the operational and denotational semantics. To carry out the logical relations proof, we make crucial use of some novel features of guarded dependent type theory recently proposed in [6], which, intuitively, allow us to reason now about elements that are only available later.

The remainder of the paper is organized as follows. In Section 2 we recall the core parts of guarded dependent type theory and the model thereof in the topos of trees [4,6]. Then we define a step-counting operational semantics of PCF in Section 3 and the denotational semantics is defined in Section 4. We prove adequacy in Section 5. In Section 6 we use the topos of trees model of the guarded type theory to summarize briefly what the results proved in guarded type theory mean externally, in standard set theory. Finally, we conclude and discuss future work in Section 7.

2 Guarded recursion

In this paper we work in a type theory with dependent types, natural numbers, inductive types and guarded recursion. The presentation of the paper will be informal, but the results of the paper can be formalised in gDTTas presented in [6] (we do not need the $\Box$ modality of gDTT). We start by recalling the core of this type theory (as described in [4]), introducing further constructions later on as needed.

A guarded recursive definition is a recursive definition where the recursive calls are guarded by time steps. The time steps are introduced via a type modality $\triangleright$ pronounced 'later'. If A is a type then $\triangleright A$ is the type of elements of A available only one time step from now. The type constructor $\triangleright$ is an applicative functor in the sense of [8], which means that there is a term $\mathsf{next} : A \to \triangleright A$ freezing an element of A so that it can be used one time step from now, and a 'later application'

$\circledast\colon \triangleright(A \to B) \to \triangleright A \to \triangleright B$ written infix, satisfying $\mathsf{next}(f) \circledast \mathsf{next}(t) = \mathsf{next}(f(t))$ among other axioms (see also [3]). In particular, $\triangleright$ extends to a functor mapping $f\colon A \to B$ to $\lambda x\colon \triangleright A.\, \mathsf{next}(f) \circledast x$.

Recursion on the level of terms is given by a fixed point operator $\mathsf{fix}\colon (\triangleright A \to A) \to A$ satisfying $f(\mathsf{next}(\mathsf{fix}(f))) = \mathsf{fix}(f)$. Intuitively, fix can compute the fixed point of any recursive definition, as long as that definition will only look at its argument later. This fixed point combinator is particularly useful in connection with guarded recursive types, i.e., types where the recursion variable occurs only guarded under a $\triangleright$ as, e.g., in the type of guarded streams:

$$\mathsf{Str}^g_A \simeq A \times \triangleright \mathsf{Str}^g_A$$

The cons operation cons^g for Str^g_A has type $A \to \triangleright \mathsf{Str}^g_A \to \mathsf{Str}^g_A$. Hence, we can define, e.g., constant streams as $\mathsf{constant}\, a = \mathsf{fix}(\lambda xs\colon \triangleright \mathsf{Str}^g_A\, .\, \mathsf{cons}^g\ a\, xs)$.

Guarded recursive types can be constructed using universes and fix as we now describe [3]. We shall assume a universe type $\mathcal{U}$ closed under both binary and dependent sums and products as usual, and containing a type of natural numbers. We write $\widehat{\mathbb{N}}$ for the code of natural numbers satisfying $\mathsf{El}(\widehat{\mathbb{N}}) \simeq \mathbb{N}$ and likewise $\widehat{\times}$ for the code of binary products satisfying $\mathsf{El}(A \mathbin{\widehat{\times}} B) \simeq \mathsf{El}(A) \times \mathsf{El}(B)$. The universe is also closed under $\triangleright$ in the sense that there exists an $\widehat{\triangleright}\colon \triangleright\mathcal{U} \to \mathcal{U}$ satisfying $\mathsf{El}(\widehat{\triangleright}(\mathsf{next}(A))) \simeq \triangleright \mathsf{El}(A)$. Using these, the type $\mathsf{Str}^g_{\mathbb{N}}$ can be defined as $\mathsf{El}(\widehat{\mathsf{Str}^g_{\mathbb{N}}})$ where $\widehat{\mathsf{Str}^g_{\mathbb{N}}} = \mathsf{fix}(\lambda B\colon \triangleright\mathcal{U}.\widehat{\mathbb{N}}\mathbin{\widehat{\times}}\widehat{\triangleright}B)$. Note that this satisfies the expected equality because

$$\mathsf{El}(\widehat{\mathsf{Str}^g_{\mathbb{N}}}) \simeq \mathsf{El}(\widehat{\mathbb{N}}\mathbin{\widehat{\times}}\widehat{\triangleright}(\mathsf{next}(\widehat{\mathsf{Str}^g_{\mathbb{N}}}))) \simeq \mathsf{El}(\widehat{\mathbb{N}}) \times \mathsf{El}(\widehat{\triangleright}(\mathsf{next}(\widehat{\mathsf{Str}^g_{\mathbb{N}}}))) \simeq \mathbb{N} \times \triangleright \mathsf{El}(\widehat{\mathsf{Str}^g_{\mathbb{N}}})$$

Likewise, guarded recursive (proof-relevant) predicates on a type A, i.e., terms of type $A \to \mathcal{U}$ can be defined using fix as we shall see an example of in Section 5.

2.1 The topos of trees model

The type theory gDTT can be modelled in the topos of trees [4], i.e., the category of presheaves over ω, the first infinite ordinal. Since this is a topos, it is a model of extensional type theory. A closed type is modelled as a family of sets $X(n)$ indexed by natural numbers together with restriction maps $r_n\colon X(n+1) \to X(n)$. Under the propositions-as-types interpretation, an element of $X(n)$ should be thought of as a proof that X is true after n computation steps. We say that X is *true at stage* n if $X(n)$ is inhabited. Note that if X is true at stage n, it is also true at stage k for all $k \leq n$. Thus, the intuition of this model is that a proposition is initially considered true and can only be falsified by further computation.

In the topos of trees model, the $\triangleright$ modality is interpreted as $\triangleright X(0) = 1$ and $\triangleright X(n+1) = X(n)$, i.e., from the logical point of view, the $\triangleright$ modality delays evaluation of a proposition by one time step. For example, if 0 is the constantly empty presheaf (corresponding to a false proposition), then $\triangleright^n 0$ is the proposition that appears true for the first n computation steps and is falsified after $n+1$ steps.

$$\frac{}{\Gamma, x:\sigma, \Delta \vdash x:\sigma}\ (\mathsf{Val}) \qquad \frac{\Gamma, x:\sigma \vdash M:\tau}{\Gamma \vdash (\lambda x:\sigma.M):\sigma\to\tau}\ (\mathsf{Lam})$$

$$\frac{\Gamma \vdash M:\sigma\to\tau \quad \Gamma\vdash N:\sigma}{\Gamma\vdash MN:\tau}\ (\mathsf{App}) \qquad \frac{\Gamma\vdash M:\sigma\to\sigma}{\Gamma\vdash \mathsf{Y}_\sigma\ M:\sigma}\ (\mathsf{Fix})$$

$$\frac{}{\Gamma\vdash \underline{n}:\mathbf{nat}}\ (\mathsf{Zero}) \qquad \frac{\Gamma\vdash M:\mathbf{nat}}{\Gamma\vdash \mathsf{succ}\ M:\mathbf{nat}}\ (\mathsf{Succ}) \qquad \frac{\Gamma\vdash M:\mathbf{nat}}{\Gamma\vdash \mathsf{pred}\ M:\mathbf{nat}}\ (\mathsf{Pred})$$

$$\frac{\Gamma\vdash L:\mathbf{nat} \quad \Gamma\vdash M:\sigma \quad \Gamma\vdash N:\sigma}{\Gamma\vdash \mathsf{ifz}\ L\ M\ N:\sigma}\ (\mathsf{IfZ})$$

Fig. 1. PCF typing rules

$$\Downarrow : \mathtt{Term}_{\mathtt{PCF}} \times \mathbb{N} \times (\mathtt{Value}_{\mathtt{PCF}} \to \mathcal{U}) \to \mathcal{U}$$

$$v \Downarrow^0 Q \stackrel{\text{def}}{=\!=} Q(v)$$

$$\mathsf{pred}\ M \Downarrow^k Q \stackrel{\text{def}}{=\!=} M \Downarrow^k \lambda x.\Sigma n\colon \mathbb{N}.x = \underline{n} \text{ and } Q(\underline{n-1})$$

$$\mathsf{succ}\ M \Downarrow^k Q \stackrel{\text{def}}{=\!=} M \Downarrow^k \lambda x.\Sigma n\colon \mathbb{N}.x = \underline{n} \text{ and } Q(\underline{n+1})$$

$$\mathsf{Y}_\sigma\ M \Downarrow^{k+1} Q \stackrel{\text{def}}{=\!=} \triangleright(M(\mathsf{Y}_\sigma\ M) \Downarrow^{k'} Q)$$

$$MN \Downarrow^{k+m} Q \stackrel{\text{def}}{=\!=} M \Downarrow^k Q' \quad \text{where } Q'(\lambda x.L) = L[N/x] \Downarrow^m Q$$

$$\mathsf{ifz}\ L\ M\ N \Downarrow^{k+m} Q \stackrel{\text{def}}{=\!=} L \Downarrow^k Q' \quad \text{where } Q'(0) = M \Downarrow^n Q \text{ and } Q'(v+1) = N \Downarrow^n Q$$

Fig. 2. Step-indexed Big-Step Operational Semantics for PCF

3 PCF

This section defines the syntax, typing judgements, and operational semantics of PCF. These should be read as judgements in guarded type theory, but as stated above we work *informally* in type theory, which here means that we ignore standard problems of representing syntax up to α-equality. Note that this is a perpendicular issue to the one we are trying to solve here.

Unlike the operational semantics to be defined below, the typing judgements of PCF are defined in an entirely standard way, see Figure 1. In the figure, v ranges over values of PCF, i.e., terms of the form $v = \underline{n}$, where n is a natural number or $v = \lambda x.M$. Note that we distinguish notationally between a natural number n and the corresponding PCF value $\underline{n}$. We denote by $\mathtt{Type}_{\mathtt{PCF}}$, $\mathtt{Term}_{\mathtt{PCF}}$and $\mathtt{Value}_{\mathtt{PCF}}$ the types of PCF types, *closed* terms, and values of PCF.

3.1 Big-step semantics

The big-step operational semantics defined in Figure 2 is a relation between terms, numbers and predicates on values. The statement $M \Downarrow^k Q$ should be read as M evaluates in k steps to a value satisfying Q. The relation can either be defined by a combination of guarded recursion and induction on M, or simply by ordinary

induction first on k then on M.

Figure 2 uses standard syntactic sugar, for example, only non-empty cases are mentioned, e.g, $v \Downarrow^k Q$ is defined to be 0 in case $k > 0$, and the case of function application should be read as

$$MN \Downarrow^l Q \stackrel{\text{def}}{=\!=} \sum k, m \colon \mathbb{N}.(k + m = l) \text{ and } M \Downarrow^k Q'$$

Note in particular that this means that $\mathsf{Y}_\sigma \ M \Downarrow^0 Q$ is always false.

As mentioned in the introduction, the formulation of the big-step operational semantics is quite delicate – the wrong definition will make the adequacy theorem false. First of all, the definition must ensure that the steps of PCF are synchronised with the steps on the meta level. This is the reason for the use of $\triangleright$ in the case of the fixed point combinator. Secondly, the use of predicates on values on the right hand side of $\Downarrow$ rather than simply values is necessary to ensure that the right hand side is not looked at before the term is fully evaluated. For example, a naive definition of the operational semantics using values on the right hand side and the rule

$$\mathsf{succ} \ M \Downarrow^k v \stackrel{\text{def}}{=\!=} \Sigma n \colon \mathbb{N}.(v = \underline{n} + 1) \text{ and } M \Downarrow^k \underline{n}$$

Would make $\mathsf{succ}\ (\mathsf{Y}_{\mathbb{N}}\ (\lambda x \colon \mathbb{N}.x)) \Downarrow^{42} \underline{0}$ false, but to obtain computational adequacy, we need this statement to be true for the first 42 steps before being falsified. (For an explanation of this point, see Remark 5.8 below.) In general, $M \Downarrow^k Q$ should be defined in such a way that in the topos of trees model it is true at stage n (using vocabulary from Section 2.1) iff either

- $k < n$ and M evaluates in precisely k steps to a value satisfying Q, or
- $k \geq n$ and evaluation of M takes more than k steps.

In particular, if M diverges, then $M \Downarrow^k Q$ should be true at stages $n \leq k$ false for $n > k$.

The use of predicates means that partial results of term evaluation are ignored, and comparison of the result to the right hand side of $\Downarrow$ is postponed until evaluation of the term is complete. The more standard big-step evaluation of terms to values can be defined as

$$M \Downarrow^k v \stackrel{\text{def}}{=\!=} M \Downarrow^k \lambda v'.v' = v$$

3.2 Small-step semantics

Figure 3 defines the small-step operational semantics. Just like the big step semantics, the small step semantics counts unfoldings of fixed points. The small steps semantics will be proved equivalent to the big-step semantics, but is introduced, because it is more suitable for the proofs of soundness and computational adequacy.

Note the following easy lemma.

Lemma 3.1 *The small-step semantics is deterministic: if $M \to^k N$ and $M \to^{k'} N'$, then $k = k'$ and $N = N'$.*

The transitive closure of the small step semantics is defined using $\triangleright$ to ensure that the steps of PCF are synchronised with the steps of the meta language.

$$\frac{}{(\lambda x:\sigma.M)(N) \to^0 M[N/x]}\ (\mathsf{SLam}) \qquad \frac{}{\mathsf{Y}_\sigma\ M \to^1 M(\mathsf{Y}_\sigma\ M)}\ (\mathsf{SFix})$$

$$\frac{}{\mathsf{pred}\ \underline{0} \to^0 \underline{0}}\ (\mathsf{SPredZ}) \qquad \frac{}{\mathsf{pred}\ \underline{n+1} \to^0 \underline{n}}\ (\mathsf{SPredN})$$

$$\frac{}{\mathsf{ifz}\ \underline{0}\ M\ N \to^0 M}\ (\mathsf{SIfZ}) \qquad \frac{}{\mathsf{ifz}\ (\underline{n+1})\ M\ N \to^0 N}\ (\mathsf{SIfN})$$

$$\frac{M \to^k M'}{M(N) \to^k M'(N)}\ (\mathsf{SApp}) \qquad \frac{M \to^k M'}{\mathsf{succ}\ M \to^k \mathsf{succ}\ M'}\ (\mathsf{SSucc})$$

$$\frac{M \to^k M'}{\mathsf{pred}\ M \to^k \mathsf{pred}\ M'}\ (\mathsf{SPred})$$

$$\frac{L \to^k L'}{\mathsf{ifz}\ L\ M\ N \to^k \mathsf{ifz}\ L'\ M\ N}\ (\mathsf{SIfZ})$$

Fig. 3. Step-Indexed Small Step semantics of PCF. In the rules, k can be 0 or 1.

Definition 3.2 Denote by $\to_*^0$ the reflexive, transitive closure of $\to^0$. The closure of the small step semantics, written $M \Rightarrow^k Q$ is a relation between closed terms, natural numbers, and predicates on closed terms, defined by induction on k as

$$M \Rightarrow^0 Q \stackrel{\text{def}}{=\!=} \Sigma N\colon \mathtt{Term}_{\mathtt{PCF}}.M \to_*^0 N \text{ and } Q(N)$$
$$M \Rightarrow^{k+1} Q \stackrel{\text{def}}{=\!=} \Sigma M', M''\colon \mathtt{Term}_{\mathtt{PCF}}.M \to_*^0 M' \text{ and } M' \to^1 M'' \text{ and } \triangleright(M'' \Rightarrow^k Q)$$

Similarly to the case of the big-step semantics we define $M \Rightarrow^k v \stackrel{\text{def}}{=\!=} M \Rightarrow^k \lambda N.v = N$

We will now prove the correspondence between the big-step and the small step operational semantics. First we need the following lemma.

Lemma 3.3 *Let M, N be closed terms of type τ, and let $Q : \mathtt{Term}_{\mathtt{PCF}} \to \mathcal{U}$.*

(i) *If $M \to^0 N$ and $N \Downarrow^k Q$ then $M \Downarrow^k Q$*

(ii) *If $M \to^1 N$ and $\triangleright(N \Downarrow^k Q)$ then $M \Downarrow^{k+1} Q$*

Proof sketch

(i) By induction on $M \to^0 N$. We consider che case $\mathsf{ifz}\ L\ M\ N \to^0 \mathsf{ifz}\ L'\ M\ N$. Assume $\mathsf{ifz}\ L'\ M\ N \Downarrow^k Q$. By definition $L' \Downarrow^k Q'$. By induction hypothesis $L \Downarrow^k Q'$ and by definition $\mathsf{ifz}\ L\ M\ N \Downarrow^k Q$. All the other cases are similar.

(ii) By induction on $M \to^1 N$. The base case is $\mathsf{Y}_\sigma\ M \to^1 M(\mathsf{Y}_\sigma\ M)$. Assume $\triangleright(M(\mathsf{Y}_\sigma\ M) \Downarrow^k Q)$. Then by definition $\mathsf{Y}_\sigma\ M \Downarrow^{k+1} Q$. We consider now the inductive cases $\mathsf{pred}\ M \to^1 \mathsf{pred}\ M'$. Assume $\triangleright(\mathsf{pred}\ M' \Downarrow^k Q)$. By definition $\triangleright(M' \Downarrow^k \lambda x.Q(x-1))$ and by induction hypothesis $M \Downarrow^{k+1} \lambda x.Q(x-1)$. By definition $\mathsf{pred}\ M \Downarrow^k Q$.

□

Lemma 3.4 *Let M be a closed term and $Q : \mathtt{Value}_{\mathtt{PCF}} \to \mathcal{U}$ a relation on values. If $M \Rightarrow^k (\lambda N.N \Downarrow^m Q)$ then $M \Downarrow^{k+m} Q$*

Proof. The proof is by induction on k. In the case where $k = k' + 1$ we have

as assumptions that $M \to^0_* N$ and $N \to^1 N'$ and $\triangleright(N' \Rightarrow^{k'+m} (\lambda N.N \Downarrow^m Q))$. By induction we have $\triangleright(N' \Downarrow^{k'+m} Q)$ and now by Lemma 3.3 also $M \Downarrow^{k+m} Q$ as desired. □

Now we can state the correspondence. Note that we have to massage the predicate of the $\Rightarrow$ relation to make things type check properly.

Lemma 3.5 *For all* $M : \mathtt{Term}_{\mathtt{PCF}}$ *and* $v : \mathtt{Value}_{\mathtt{PCF}}$ *,*
$M \Downarrow^k v$ *iff* $M \Rightarrow^k \lambda N.N \Downarrow^0 v$

Proof. We consider implication from left to right in the case of the fix-point. Assume $\mathsf{Y}_\sigma\ M \Downarrow^{k+1} v$. By definition $\triangleright(M(\mathsf{Y}_\sigma\ M) \Downarrow^k v)$. By induction hypothesis $\triangleright(M(\mathsf{Y}_\sigma\ M) \Rightarrow^k \lambda N.N \Downarrow^0 v)$. Together with $\mathsf{Y}_\sigma\ M \to^1 M(\mathsf{Y}_\sigma\ M)$ by definition $\mathsf{Y}_\sigma\ M \Rightarrow^{k+1} \lambda N.N \Downarrow^0 v$

The case from right to left follows from Lemma 3.4.

□

The following is the standard statement for operational correspondence and follows directly from Lemma 3.5.

Corollary 3.6 $M \Downarrow^k v \Leftrightarrow M \Rightarrow^k v$

4 Denotational semantics

We now define the denotational semantics of PCF. For this, we use the guarded recursive *lifting monad* on types, defined as the guarded recursive type

$$LA \stackrel{\text{def}}{=\!=} \mathsf{fix}\, X.(A + \triangleright X).$$

Let $i\colon A + \triangleright LA \cong LA$ be the isomorphism, let $\theta : \triangleright LA \to LA$ be the right inclusion composed with i and let $\eta\colon A \to LA$ (the unit of the monad) denote the left inclusion composed with i. Note that any element of LA is either of the form $\eta(a)$ or $\theta(r)$.

We can describe the universal property of LA as follows. Define a $\triangleright$-algebra to be a type B together with a map $\theta_B : \triangleright B \to B$. The lifting LA as defined above is the *free* $\triangleright$-algebra on A. Given $f : A \to B$ with B a $\triangleright$-algebra, the unique extension of f to a homomorphism of $\triangleright$-algebras $\hat{f}\colon LA \to B$ is defined as

$$\hat{f}(\eta(a)) \stackrel{\text{def}}{=\!=} f(a)$$
$$\hat{f}(\theta(r)) \stackrel{\text{def}}{=\!=} \theta_B(\mathsf{next}(\hat{f}) \circledast r)$$

which can be formally expressed as a fixed point of a term of type $\triangleright(LA \to B) \to LA \to B$.

The intuition the reader should have for L is that LA is the type of computations possibly returning an element of A, recording the number of steps used in the computation. The unit η gives an inclusion of values into computations, the composite $\delta = \theta \circ \mathsf{next}\colon LA \to LA$ is an operation that adds one time step to a computation, and the bottom element $\bot = \mathsf{fix}(\theta)$ is the diverging computation. In fact, any $\triangleright$-algebra has a bottom element and an operation δ as defined above, and homorphisms preserve this structure.

$$
\begin{aligned}
[\![x_1 : \sigma_1, \cdots, x_k : \sigma_k \vdash x_i]\!](\gamma) &= \pi_i \gamma \\
[\![\Gamma \vdash \underline{n} : \mathbf{nat}]\!](\gamma) &= \eta(n) \\
[\![\Gamma \vdash \mathsf{Y}_\sigma\ M]\!](\gamma) &= (\mathsf{fix}_{[\![\sigma]\!]})(\lambda x : \triangleright[\![\sigma]\!].\theta_\sigma(\mathsf{next}([\![M]\!](\gamma))) \circledast x)) \\
[\![\Gamma \vdash \lambda x.M]\!](\gamma) &= \lambda x.[\![M]\!](\gamma, x) \\
[\![\Gamma \vdash MN]\!](\gamma) &= [\![M]\!](\gamma)[\![N]\!](\gamma) \\
[\![\Gamma \vdash \mathsf{succ}\ M]\!](\gamma) &= L(\lambda x.x + 1)([\![M]\!](\gamma)) \\
[\![\Gamma \vdash \mathsf{pred}\ M]\!](\gamma) &= L(\lambda x.x - 1)([\![M]\!](\gamma)) \\
[\![\Gamma \vdash \mathsf{ifz}\ L\ M\ N]\!](\gamma) &= (\widehat{\mathsf{ifz}}([\![M]\!](\gamma), [\![N]\!](\gamma)))([\![L]\!](\gamma))
\end{aligned}
$$

Fig. 4. Interpretation of terms

4.1 Interpretation

The interpretation function $[\![\cdot]\!] : \mathsf{Type}_{\text{PCF}} \to \mathcal{U}$ is defined by induction.

$$
\begin{aligned}
[\![\mathbf{nat}]\!] &\stackrel{\text{def}}{=\!=} L\mathbb{N} \\
[\![\tau \to \sigma]\!] &\stackrel{\text{def}}{=\!=} [\![\tau]\!] \to [\![\sigma]\!]
\end{aligned}
$$

The denotation of every type is a $\triangleright$-algebra: the map $\theta_\sigma \colon \triangleright[\![\sigma]\!] \to [\![\sigma]\!]$ is defined by induction on σ by

$$\theta_{\sigma\to\tau} = \lambda f \colon \triangleright([\![\sigma]\!] \to [\![\tau]\!]).\lambda x \colon [\![\sigma]\!].\theta_\tau(f \circledast \mathsf{next}(x))$$

Typing judgements $\Gamma \vdash M : \sigma$ are interpreted as usual as functions from $[\![\Gamma]\!]$ to $[\![\sigma]\!]$, where the interpretation of contexts is defined as $[\![x_1 : \sigma_1, \cdots, x_k : \sigma_k]\!] \stackrel{\text{def}}{=\!=} [\![\sigma_1]\!] \times \cdots \times [\![\sigma_n]\!]$. Figure 4 defines the interpretation interpretation of judgements. Below we often write $[\![M]\!]$ rather than $[\![\Gamma \vdash M \colon \sigma]\!]$. Natural numbers in PCF are computations that produce a value in zero step, so we interpret them by using η. In the case of Y_σ we have by induction a map $[\![M]\!](\gamma)$ of type $[\![\sigma]\!] \to [\![\sigma]\!]$. Morally, $[\![\Gamma \vdash \mathsf{Y}_\sigma\ M]\!](\gamma)$ should be the fixed point of $[\![M]\!](\gamma)$ composed with δ, ensuring that each unfolding of the fixed point is recorded as a step in the model, but to get the types correct, we have to apply the functorial action of $\triangleright$ to $[\![M]\!](\gamma)$ and compose with θ instead of δ. The intuition given above is captured in the following lemma.

Lemma 4.1 *Let* $\Gamma \vdash M : \sigma \to \sigma$ *then* $[\![\mathsf{Y}_\sigma\ M]\!] = \delta_\sigma \circ [\![M(\mathsf{Y}_\sigma\ M)]\!]$

We now explain the interpretation of $\mathsf{ifz}\ L\ M\ N$. Define first a semantic $\mathsf{ifz} \colon [\![\sigma]\!] \to [\![\sigma]\!] \to \mathbb{N} \to [\![\sigma]\!]$ operation by

$$\mathsf{ifz}\, x\ y\ 0 \stackrel{\text{def}}{=\!=} x \qquad\qquad \mathsf{ifz}\, x\ y\ (n+1) \stackrel{\text{def}}{=\!=} y$$

The operation $\widehat{\mathsf{ifz}} \colon [\![\sigma]\!] \to [\![\sigma]\!] \to [\![\mathbf{nat}]\!] \to [\![\sigma]\!]$ is defined by $\widehat{\mathsf{ifz}}\ x\ y$ being the extension of $\mathsf{ifz}\ x\ y$ to a homomorphism of $\triangleright$-algebras. As a direct consequence of this definition we get

Lemma 4.2 (i)

$$[\![\ \lambda x\colon \mathbf{nat}.\,\mathsf{ifz}\ x\ M\ N]\!](\theta(r)) = \theta(\mathsf{next}([\![\ \lambda x\colon \mathbf{nat}.\,\mathsf{ifz}\ x\ M\ N]\!](\gamma)) \circledast r)$$

(ii) *If* $[\![L]\!](\gamma) = \delta([\![L']\!](\gamma))$, *then* $[\![\mathsf{ifz}\ L\ M\ N]\!](\gamma) = \delta[\![\mathsf{ifz}\ L'\ M\ N]\!](\gamma)$

4.2 Soundness

The soundness theorem states that if a program M evaluates to a value v in k steps then the interpretation of M is equal to the interpretation of v delayed k times by the semantic delay operation δ. Thus the soundness theorem captures not just extensional but also intensional behaviour of terms.

The theorem is proved using the small-step semantics. We first need a lemma for the single step reduction.

Lemma 4.3 *Let M be a closed term of type τ. If $M \to^k N$ then $[\![M]\!](*) = \delta^k[\![N]\!](*)$*

Proof. The proof goes by induction on $M \to^k N$, and here we only consider two cases. The case of $\mathsf{Y}_\sigma\ M \to^1 M(\mathsf{Y}_\sigma\ M)$ follows from Lemma 4.1. In the case of $\mathsf{ifz}\ M_1\ N_1\ N_2 \to^1 \mathsf{ifz}\ M_2\ N_1\ N_2$, the induction hypothesis gives $[\![M_1]\!] = \delta \circ [\![M_2]\!]$, and now Lemma 4.2 applies proving the case. □

We prove it now for $\Rightarrow^k$.

Lemma 4.4 *Let M be a closed term of type τ, if $M \Rightarrow^k N$ then $[\![M]\!](*) = \delta^k[\![N]\!](*)$*

Proof. By induction on k. The case $k = 0$ follows from Lemma 4.3. Assume $k = k' + 1$. By definition we have $M \to_*^0 M'$ and $M' \to^1 M''$ and $\triangleright(M'' \Rightarrow^{k'} N)$. By repeated application of Lemma 4.3 we get $[\![M]\!](*) = [\![M']\!](*)$ and $[\![M']\!](*) = \delta([\![M'']\!](*))$.

By induction hypothesis we get $\triangleright([\![M'']\!](*) = \delta^{k'}[\![N]\!](*))$. By gDTT rule $\mathsf{TY} - \mathsf{COM}_\triangleright$ this implies $\mathsf{next}([\![M'']\!](*)) = \mathsf{next}(\delta^{k'}[\![N]\!](*)))$ and since $\delta = \theta \circ \mathsf{next}$, this implies $\delta[\![M'']\!](*) = \delta^k[\![N]\!](*)$. By putting together the equations we get finally $[\![M]\!](*) = \delta^k[\![N]\!](*)$. □

The Soundness theorem follows from the fact that the small-step semantics is equivalent to the big step, which is Corollary 3.6.

Theorem 4.5 (Soundness) *Let M be a closed term of type τ, if $M \Downarrow^k v$ then $[\![M]\!](*) = \delta^k[\![v]\!](*)$*

5 Computational Adequacy

In this section we prove that the denotational semantics is computationally adequate with respect to the operational semantics. At a high level, we proceed in the standard way, by constructing a logical relation $\mathcal{R}_\sigma$ between denotations $[\![\sigma]\!]$ and terms $\mathtt{Term}_{\mathrm{PCF}}$ and then proving that open terms and their denotation respect this relation (Lemma 5.6 below). We define our logical relation in guarded dependent

type theory, so formally, it will be a map into the universe $\mathcal{U}$ of types. Thus we work with a proof-relevant logical relation, similar to what was recently done in work of Benton et. al. [**?**].

To formulate the definition of the logical relations and also to carry out the proof of the fundamental theorem of logical relations, we need some more sophisticated features of gDTT, which we now recall.

5.1 Guarded Dependent Type Theory

We recall some key features of gDTT; see [6] for more details.

As mentioned in Section 2, the later functor $\triangleright$ is an applicative functor. Guarded dependent type theory extends the later application $\circledast\colon \triangleright(A \to B) \to \triangleright A \to \triangleright B$ to the dependent case using a new notion of *delayed substitution*: If $\Gamma \vdash f : \triangleright\Pi(x : A).B$ and $\Gamma \vdash t : \triangleright A$, then the term $f \circledast t$ has type $\triangleright\,[x \leftarrow t]\,.B$, where $[x \leftarrow t]$ is a *delayed substitution*. Note that since t has the type $\triangleright A$, and not A, then we cannot substitute t for x in B. Intuitively, t will eventually reduce to some value $\mathsf{next}\, u$, and so the resulting type should be $\triangleright B[u/x]$. But when t is an open term, then we cannot perform this reduction, and thus cannot type this term. Hence we use the type mentioned earlier $\triangleright\,[x \leftarrow t]\,.B$, in whch x is bound in B. Definitional equality rules allow us to simplify this type when t has form $\mathsf{next}\, u$, i.e.,

$$\triangleright[x \leftarrow \mathsf{next}\, u].B \simeq \triangleright B[u/x]$$

as expected. Here we have just considered a single delayed substitution, in general, we may have sequences of delayed substitutions (such as $\triangleright\,[x \leftarrow t, y \leftarrow u]\,.C$). Delayed substitutions can also occur in terms, e.g., if $\Gamma, x\colon A \vdash t : B$ and $\Gamma \vdash u : \triangleright A$, then $\Gamma \vdash \mathsf{next}\,[x \leftarrow u]\,.t : \triangleright\,[x \leftarrow t]\,.B$

Recall from Section 2 that $\mathsf{next}\, f \circledast \mathsf{next}\, t \equiv \mathsf{next}(f(t))$. We can use delayed substitutions to express what a later application should be equal to, if one or both of the arguments are not of the form $\mathsf{next}(u)$ (for some term u). In gDTT, given f of type $\triangleright(X \to Y)$ and x of type $\triangleright X$, then $f \circledast x$ is equal to $\mathsf{next}\,[g \leftarrow f, y \leftarrow x]\,.(g(x))$. Definitional equality rules allow us to simplify such terms when one or both of f or x is of the form $\mathsf{next}\, u$. For example, $\mathsf{next}\,[g \leftarrow \mathsf{next}(f), y \leftarrow x]\,.(g(x))$ is definitionally equal to $\mathsf{next}\,[y \leftarrow x]\,.(f(x))$. Similarly for delayed substitutions in types.

5.2 Logical Relation

In this section we define a logical relation to prove the adequacy theorem. This relation is a function to $\mathcal{U}$.

We introduce the following notation:

Notation 1 *Let $\mathcal{R} : A \to B \to \mathcal{U}$ be a relation from A to B, t of type $\triangleright A$ and u of type $\triangleright B$. Define $t \triangleright\mathcal{R}\ u \stackrel{\text{def}}{=\!=} \triangleright\,[x \leftarrow t, y \leftarrow u]\,.(x\ \mathcal{R}\ y)$*

Note that $(\mathsf{next}(t) \triangleright\mathcal{R}\ \ \mathsf{next}(u)) \simeq \triangleright(tRu)$.

Lemma 5.1 *The mapping $\lambda R.\ \triangleright\mathcal{R} : (A \to B \to \mathcal{U}) \to \triangleright A \to \triangleright B \to \mathcal{U}$ is contractive, i.e., can be factored as $F \circ \mathsf{next}$ for some $F\colon \triangleright(A \to B \to \mathcal{U}) \to \triangleright A \to \triangleright B \to \mathcal{U}$.*

Proof. Define $F(S)\,x\,y = \widehat{\triangleright}(S \circledast x \circledast y)$. □

Definition 5.2 [Logical Relation] The logical relation $\mathcal{R}_\tau : [\![\tau]\!] \times \mathtt{Term}_{\mathtt{PCF}} \to \mathcal{U}$ is inductively defined on types.

$$\eta(v)\ \mathcal{R}_{\mathbf{nat}}\ M \stackrel{\text{def}}{=\!=} M \Downarrow^0 v$$
$$\theta_{\mathbf{nat}}(r)\ \mathcal{R}_{\mathbf{nat}}\ M \stackrel{\text{def}}{=\!=} \Sigma M', M'' \colon \mathtt{Term}_{\mathtt{PCF}}. M \to_*^0 M' \text{ and } M' \to^1 M'' \text{ and } r \triangleright \mathcal{R}_{\mathbf{nat}}\ \mathsf{next}(M'')$$
$$f\ \mathcal{R}_{\tau\to\sigma}\ M \stackrel{\text{def}}{=\!=} \Pi\alpha \colon [\![\tau]\!], N \colon \mathtt{Term}_{\mathtt{PCF}}. \alpha\ \mathcal{R}_\tau\ N \implies f(\alpha)\ \mathcal{R}_\sigma\ (MN)$$

The definition of $\mathcal{R}_{\mathbf{nat}}$ is by guarded recursion using Lemma 5.1.

We now prove a series of lemmas needed for the proof of computational adequacy. The first states that the applicative functor action $\circledast$ respects the logical relation.

Lemma 5.3 *If* $r \triangleright \mathcal{R}_\tau\ \mathsf{next}(L)$ *and* $f \triangleright \mathcal{R}_{\tau\to\sigma}\ \mathsf{next}(M)$ *then* $(f \circledast r) \triangleright \mathcal{R}_\sigma\ \mathsf{next}(ML)$

Proof. Assume $r \triangleright \mathcal{R}_\tau\ \mathsf{next}(L)$ and $f \triangleright \mathcal{R}_{\tau\to\sigma}\ \mathsf{next}(M)$ which by repeatedly unfolding is

$$\triangleright\,[g \leftarrow f]\,.(g\ \mathcal{R}_{\tau\to\sigma}\ M) \simeq \triangleright\,[g \leftarrow f]\,.(\Pi(y : [\![\sigma]\!])(L : \mathtt{Term}_{\mathtt{PCF}}).y\ \mathcal{R}_\tau\ L \to g(y)\ \mathcal{R}_\sigma\ ML)$$

Applying this to r, $\mathsf{next}(L)$ and $r \triangleright \mathcal{R}_{\mathbf{nat}}\ \mathsf{next}(L)$ gives $\triangleright\,[g \leftarrow f, y \leftarrow r]\,.(g(y)\ \mathcal{R}_\sigma\ ML)$, which can be reduced by type equalities to

$$\mathsf{next}\,[g \leftarrow f, y \leftarrow r]\,.(g(y)) \triangleright \mathcal{R}_\sigma\ \mathsf{next}(ML) \simeq (f \circledast r) \triangleright \mathcal{R}_\sigma\ \mathsf{next}(ML)$$

by distributing the next and applying the substitutions. □

The following lemma generalises the second case of $\mathcal{R}_{\mathbf{nat}}$ to all types.

Lemma 5.4 *Let* α *of type* $\triangleright[\![\sigma]\!]$ *and two terms* N *and* M*, if* $(\alpha \triangleright \mathcal{R}_\sigma\ \mathsf{next}(N))$ *and* $M \to^1 N$ *then* $\theta_\sigma(\alpha)\ \mathcal{R}_\sigma\ M$

Proof. The proof is by induction on σ. The base case $\sigma = \mathbf{nat}$ is by definition of $\mathcal{R}_{\mathbf{nat}}$.

For the induction step, suppose α of type $\triangleright[\![\tau_1 \to \tau_2]\!]$, and M, N are closed terms such that $\alpha \triangleright \mathcal{R}_{\tau_1\to\tau_2}\ \mathsf{next}(N)$ and $M \to^1 N$. We must show that if $\beta : [\![\tau_1]\!]$, $P : \mathtt{Term}_{\mathtt{PCF}}$ and $\beta\ \mathcal{R}_{\tau_1}\ P$ then $(\theta_{\tau_1\to\tau_2}(\alpha))(\beta)\ \mathcal{R}_{\tau_2}\ (MP)$.

So suppose $\beta\ \mathcal{R}_{\tau_1}\ P$, and thus also $\triangleright(\beta\ \mathcal{R}_{\tau_1}\ P)$ which is equal to $\mathsf{next}(\beta) \triangleright \mathcal{R}_{\tau_1}\ \mathsf{next}(P)$. By applying Lemma 5.3 to this and $\alpha \triangleright \mathcal{R}_{\tau_1\to\tau_2}\ \mathsf{next}(N)$ we get

$$\alpha \circledast (\mathsf{next}(\beta)) \triangleright \mathcal{R}_{\tau_2}\ \mathsf{next}(NP)$$

Since $M \to^1 N$ also $MP \to^1 NP$, and thus, by the induction hypothesis for τ_2, $\theta_{\tau_2}(\alpha \circledast (\mathsf{next}(\beta)))\ \mathcal{R}_{\tau_2}\ MP$. Since by definition $\theta_{\tau_1\to\tau_2}(\alpha) \circledast \beta = \theta_{\tau_2}(\alpha \circledast \mathsf{next}(\beta))$, this proves the case. □

Lemma 5.5 *If* $\alpha\ \mathcal{R}_\sigma\ M$ *and* $M \to^0 N$ *then* $\alpha\ \mathcal{R}_\sigma\ N$

Proof. The proof is by induction on σ. We show the case $\sigma = \mathbf{nat}$. We proceed by case analysis on α. We show the case when $\alpha = \theta_{\mathbf{nat}}(r)$. From the assumption

$\alpha\ \mathcal{R}_\sigma\ M$ we have that there exists M' and M'' such that $M \to_*^0 M'$ and $M' \to^1 M''$ and $\alpha \triangleright \mathcal{R}_{\mathbf{nat}}\ \mathsf{next}(M'')$. By determinism of the small-step semantics (Lemma 3.1) the reduction $M \to_*^0 M'$ must factor as $M \to N \to_*^0 M'$ and thus $\alpha\ \mathcal{R}_{\mathbf{nat}}\ N$ as desired. □

We can now finally prove the fundamental lemma, which can be thought of as a strengthened induction hypothesis for computational adequacy, generalised to open terms.

Lemma 5.6 (Fundamental Lemma) *Let* $\Gamma \vdash t : \tau$*, suppose* $\Gamma \equiv x_1 : \tau_1, \cdots, x : \tau$ *and* $t_i : \tau_i$*,* $\alpha_i : [\![\tau_i]\!]$ *and* $\alpha_i\ \mathcal{R}_{[\![\tau_i]\!]}\ t_i$ *for* $i \in \{1, \ldots, n\}$*, then* $[\![t]\!](\boldsymbol{\alpha})\ \mathcal{R}_\tau\ t[\boldsymbol{x} := \boldsymbol{t}]$

Proof. The proof is by induction on the height of the typing judgement, and we just show the two most difficult cases.

We start off by the case $\Gamma \vdash \mathsf{Y}_\sigma\ M : \sigma$. The argument is by guarded recursion: we assume

$$\triangleright([\![\mathsf{Y}_\sigma\ M]\!](\boldsymbol{\alpha})\ \mathcal{R}_\sigma\ (\mathsf{Y}_\sigma\ M)([\boldsymbol{x} := \boldsymbol{t}])) \tag{1}$$

and prove $[\![\mathsf{Y}_\sigma\ M]\!](\boldsymbol{\alpha})\ \mathcal{R}_\sigma\ (\mathsf{Y}_\sigma\ M)([\boldsymbol{x} := \boldsymbol{t}])$. By induction hypothesis we know $[\![M]\!](\boldsymbol{\alpha})\ \mathcal{R}_{\sigma\to\sigma}\ M[\boldsymbol{x} := \boldsymbol{t}]$, hence we derive $\triangleright([\![M]\!](\boldsymbol{\alpha})\ \mathcal{R}_{\sigma\to\sigma}\ M[\boldsymbol{x} := \boldsymbol{t}])$, i.e.,

$$\triangleright(\Pi\alpha : [\![\sigma]\!].N : \mathtt{Term}_{\mathtt{PCF}}.\ \alpha\ \mathcal{R}_\sigma\ N \Rightarrow [\![M]\!](\boldsymbol{\alpha})(\alpha)\ \mathcal{R}_\sigma\ \ (M[\boldsymbol{x} := \boldsymbol{t}]N)) \tag{2}$$

Applying (2) to (1) we get

$$\triangleright([\![M]\!](\boldsymbol{\alpha})([\![\mathsf{Y}_\sigma\ M]\!](\boldsymbol{\alpha}))\ \mathcal{R}_\sigma\ (M[\boldsymbol{x} := \boldsymbol{t}](\mathsf{Y}_\sigma\ M[\boldsymbol{x} := \boldsymbol{t}])))$$

which is equal as types to

$$\begin{aligned}\triangleright([\![M(\mathsf{Y}_\sigma\ M)]\!](\boldsymbol{\alpha})\ \mathcal{R}_\sigma\ (M(\mathsf{Y}_\sigma\ M))[\boldsymbol{x} := \boldsymbol{t}] \\ \simeq \mathsf{next}([\![M(\mathsf{Y}_\sigma\ M)]\!](\boldsymbol{\alpha}))\ \triangleright\mathcal{R}_\sigma\ \mathsf{next}((M(\mathsf{Y}_\sigma\ M))[\boldsymbol{x} := \boldsymbol{t}])\end{aligned}$$

Thus, by Lemma 5.4

$$\theta_\sigma(\mathsf{next}([\![M(\mathsf{Y}_\sigma\ M)]\!](\boldsymbol{\alpha})))\ \mathcal{R}_\sigma\ (\mathsf{Y}_\sigma\ M)([\boldsymbol{x} := \boldsymbol{t}])$$

and as $\delta_\sigma = \theta_\sigma \circ \mathsf{next}$, by Lemma 4.1

$$[\![\mathsf{Y}_\sigma\ M]\!](\boldsymbol{\alpha})\ \mathcal{R}_\sigma\ (\mathsf{Y}_\sigma\ M)([\boldsymbol{x} := \boldsymbol{t}])$$

as desired.

Now the case of $\Gamma \vdash \mathsf{ifz}\ L\ M\ N : \sigma$. This case can be shown by showing that

$$[\![\lambda x.\,\mathsf{ifz}\ x\ M\ N]\!](\boldsymbol{\alpha})\ \mathcal{R}_{\mathbf{nat}\to\sigma}\ \ (\lambda x.\,\mathsf{ifz}\ x\ M\ N)[\boldsymbol{x} := \boldsymbol{t}]$$

and then applying this to the induction hypothesis $[\![L]\!](\boldsymbol{\alpha})\ \mathcal{R}_{\mathbf{nat}}\ L[\boldsymbol{x} := \boldsymbol{t}]$. The argument is by guarded recursion. Assume

$$\triangleright([\![\lambda x.\,\mathsf{ifz}\ x\ M\ N]\!](\boldsymbol{\alpha})\ \mathcal{R}_{\mathbf{nat}\to\sigma}\ \ (\lambda x.\,\mathsf{ifz}\ x\ M\ N)[\boldsymbol{x} := \boldsymbol{t}]) \tag{3}$$

We must show that if $\beta : [\![\mathbf{nat}]\!]$, $L : \mathtt{Term}_{\mathtt{PCF}}$ and $\beta\ \mathcal{R}_{\mathbf{nat}}\ L$ then

$$[\![\lambda x.\,\mathsf{ifz}\ \ x\ M\ N]\!](\boldsymbol{\alpha})(\beta)\ \mathcal{R}_\sigma\ ((\lambda x.\,\mathsf{ifz}\ \ x\ M\ N)[\boldsymbol{x} := \boldsymbol{t}](L))$$

We proceed by case analysis on β. The interesting case is $\beta = \theta_{\mathbf{nat}}(r)$. Here r is of type $\triangleright[\![\mathbf{nat}]\!]$ and $L : \mathtt{Term}_{\mathtt{PCF}}$. The hypothesis $\theta_{\mathbf{nat}}(r)\ \mathcal{R}_{\mathbf{nat}}\ L$ states that there exist $L', L'' : \mathtt{Term}_{\mathtt{PCF}}$ s.t. $L \to_*^0 L'$, $L' \to^1 L''$ and

$$r\ \triangleright\mathcal{R}_{\mathbf{nat}}\ \ \mathsf{next}(L'') \tag{4}$$

Since (3) is equal to

$$(\mathsf{next}([\![\lambda x.\,\mathsf{ifz}\ \ x\ M\ N]\!](\boldsymbol{\alpha})))\ \triangleright\mathcal{R}_{\mathbf{nat}\to\sigma}\ \ \mathsf{next}((\lambda x.\,\mathsf{ifz}\ \ x\ M\ N)[\boldsymbol{x} := \boldsymbol{t}])$$

We can apply Lemma 5.3 to that and (4) to get

$$(\mathsf{next}([\![\lambda x.\,\mathsf{ifz}\ \ x\ M\ N]\!](\boldsymbol{\alpha}))\circledast r)\ \triangleright\mathcal{R}_\sigma\ \ \mathsf{next}((\mathsf{ifz}\ \ L''\ M[\boldsymbol{x} := \boldsymbol{t}]\ N[\boldsymbol{x} := \boldsymbol{t}]))$$

By Lemma 5.4 with $L' \to^1 L''$ this implies

$$\theta_\sigma(\mathsf{next}([\![\lambda x.\,\mathsf{ifz}\ \ x\ M\ N]\!](\boldsymbol{\alpha}))\circledast r)\ \mathcal{R}_\sigma\ \ \mathsf{next}((\mathsf{ifz}\ \ L'\ M[\boldsymbol{x} := \boldsymbol{t}]\ N[\boldsymbol{x} := \boldsymbol{t}]))$$

and by Lemma 4.2 along with repeated application of Lemma 5.5 this implies

$$[\![\lambda x.\,\mathsf{ifz}\ \ x\ M\ N]\!](\boldsymbol{\alpha})(\beta)\ \mathcal{R}_\sigma\ \ (\lambda x.\,\mathsf{ifz}\ \ x\ M\ N)[\boldsymbol{x} := \boldsymbol{t}](L)$$

thus getting what we wanted. □

We have now all the pieces in place to prove adequacy.

Theorem 5.7 (Computational Adequacy) *If M is a closed term of type* **nat** *then $M \Downarrow^k v$ iff $[\![M]\!](*) = \delta^k[\![v]\!]$*

Proof. The left to right implication is soundness (Theorem 4.5). For the right to left implication note first that the Fundamental Lemma (Lemma 5.6) implies $\delta^k[\![v]\!]\ \mathcal{R}_{\mathbf{nat}}\ M$. An easy induction on k then proves that $M \Downarrow^k v$. □

Remark 5.8 In the topos of trees model $[\![\mathbf{nat}]\!](n) \cong \{1, \ldots, n\} \times \mathbb{N} + \{\bot\}$. Values are modelled as elements of the form $(1, k)$ and δ is defined as $\delta(j, k) = (j + 1, k)$ if $j < n$ and $\delta(n, k) = \bot$. Thus, if a term M diverges, then $[\![M]\!](*) = \delta^k[\![v]\!]$ holds at stage n whenever $k > n$ explaining the need for $M \Downarrow^k v$ to be true also at stage n when $k > n$.

6 The external viewpoint

The adequacy theorem is a statement formulated entirely in gDTT, relating two notions of semantics also formulated entirely in gDTT. While we believe that gDTT is a natural setting to do semantics in, and that the result therefore is interesting in its own right, it is still natural to as what we proved in the "real world". One way of formulating this question more precisely is to use the interpretation of gDTT in

the topos of trees (henceforth denoted by $(\!|-|\!)$). For example, the sets of PCF types, terms and values are inductively defined types, which are interpreted as constant presheaves over the corresponding *sets* of types, terms and values. Types of PCF as understood in set theory, thus correspond bijectively to global element of $(\!|\mathtt{Type}_{\mathtt{PCF}}|\!)$, which by composing with the interpretation of PCF defined in gDTT gives rise to an object in the topos of trees. Likewise, a PCF term gives rise to a morphism in the topos of trees. Thus, essentially by composing the interpretation of PCF given above with the interpretation of gDTT, we get an interpretation of PCF into the topos of trees, which we will denote by $[\![-]\!]_{\mathtt{ext}}$.

We denote by $M \Downarrow^k_{\mathtt{ext}} v$ the usual external formulation of the big-step semantics for PCF (see e.g. [7]).

Lemma 6.1 *The type $(\!|M \Downarrow^k Q|\!)$ is globally inhabited iff there exists a value v such that $M \Downarrow^k_{ext} v$ and $(\!|Q(v)|\!)$ is globally inhabited.*

The lemma can be proved by induction over first k then M.

As a special case, Theorem 5.7 states that $(\!|M \Downarrow^k v|\!)$ is inhabited by a global element iff $(\!|[\![M]\!](*) = \delta^k[\![v]\!]|\!)$ is inhabited by a global element. Since the topos of trees is a model of extensional type theory, the latter holds precisely when $[\![M]\!]_{\mathtt{ext}} = \delta^k[\![v]\!]_{\mathtt{ext}}$.

Theorem 6.2 (Computational Adequacy, externally) *If $\vdash M : \sigma$ with σ a ground type, then $M \Downarrow^k_{ext} v$ iff $[\![M]\!]_{ext}(*) = \delta^k[\![v]\!]_{ext}$*

7 Discussion and Future Work

In earlier work, it has been shown how guarded type theory can be used to give abstract accounts of operationally-based step-indexed models [4,11]. There the operational semantics of the programming language under consideration is also defined inside guarded type theory, but there are no explicit counting of steps (indeed, part of the point is to avoid the steps). Instead, the operational semantics is defined by the transitive closure of a single-step relation — and, importantly, the transitive closure is defined by a fixed point using guarded recursion. Thus some readers might be surprised why we use a step-counting operational semantics here. The reason is simply that we want to show, in the type theory, that the denotational semantics is adequate with respect to an operational semantics and since the denotational semantics is intensional and steps thus matter, we also need to count steps in the operational semantics to formulate adequacy.

In previous work [4] we have studied the internal topos logic of the topos of trees model of guarded recursion and used this for reasoning about advanced programming languages. In this paper, we could have likewise chosen to reason in topos logic rather than type theory. We believe that the proofs of soundness and computational adequacy would have gone through also in this setting, but the interaction between the $\triangleright$ type modality and the existential quantifiers in the topos of trees, makes this an unnatural choice. For example, one can prove the statement $\exists k.\exists v.\, \mathsf{Y}_{\mathbf{nat}}\ (\lambda x.x) \Downarrow^k v$ in the internal logic using guarded recursion as follows: assume $\triangleright(\exists k.\exists v.\, \mathsf{Y}_{\mathbf{nat}}\ (\lambda x.x) \Downarrow^k v)$. Because **nat** is total and inhabited we can pull out the existentials by Theorem 2.7.4 in [4] and derive $\exists k.\exists v.\triangleright(\mathsf{Y}_{\mathbf{nat}}\ (\lambda x.x) \Downarrow^k v)$

which implies $\exists k.\exists v.\, \mathsf{Y}_{\mathbf{nat}}\ (\lambda x.x) \Downarrow^k v$. The same statement in type theory is not derivable as can be proved in the topos of trees. Intuitively the difference is the constructiveness of the dependent sum, which allows us to extract the witnesses k and n.

In future work, we would like to explore models of FPC (i.e., a PCF extended with recursive types) and also investigate how to define a more extensional model by quotienting the present intensional model. The latter would be related to Escardo's results in [7].

References

[1] 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.

[2] Robert Atkey and Conor McBride. Productive coprogramming with guarded recursion. In *ICFP*, pages 197–208, 2013.

[3] Nick Benton, Martin Hofmann, and Vivek Nigam. Abstract effects and proof-relevant logical relations. In *POPL*, 2014.

[4] Lars Birkedal and Rasmus Ejlers Møgelberg. Intensional type theory with guarded recursive types qua fixed points on universes. In *LICS*, pages 213–222, 2013.

[5] Lars Birkedal, Rasmus Ejlers Møgelberg, Jan Schwinghammer, and Kristian Støvring. First steps in synthetic guarded domain theory: step-indexing in the topos of trees. *LMCS*, 8(4), 2012.

[6] Ales Bizjak, Lars Birkedal, and Marino Miculan. A model of countable nondeterminism in guarded type theory. In *RTA-TLCA*, pages 108–123, 2014.

[7] Aleš Bizjak, Hans Bugge Grathwohl, Ranald Clouston, Rasmus Ejlers Møgelberg, and Lars Birkedal. Guarded dependent type theory with coinductive types, 2015. Submitted for publication. Extended version available at http://users-cs.au.dk/abizjak/documents/trs/gdtt-ext.pdf.

[8] Venanzio Capretta. General recursion via coinductive types. *Logical Methods in Computer Science*, 1(2), 2005.

[9] M.H. Escardo. A metric model of PCF. Laboratory for Foundations of Computer Science, University of Edinburgh, http://www.dcs.st-and.ac.uk/~mhe/, April 1999.

[10] C. McBride and R. Paterson. Applicative programming with effects. *Journal of Functional Programming*, 18(1), 2008.

[11] Rasmus Ejlers Møgelberg. A type theory for productive coprogramming via guarded recursion. In *CSL-LICS*, 2014.

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

[13] Kasper Svendsen and Lars Birkedal. Impredicative concurrent abstract predicates. In *ESOP*, 2014.