Under consideration for publication in Math. Struct. in Comp. Science

Denotational semantics of recursive types in synthetic guarded domain theory

RASMUS E. MØGELBERG[1] † and MARCO PAVIOTTI[2]

[1] *IT University of Copenhagen, Copenhagen, Denmark.*
[2] *School of Computing, University of Kent, Canterbury, United Kingdom.*

Received 21 April 2017

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 reasoning operationally about programming languages with advanced features including general references, recursive types, countable non-determinism and concurrency.
Guarded recursion also offers a way of adding recursion to type theory while maintaining logical consistency. In previous work we initiated a programme of denotational semantics in type theory using guarded recursion, by constructing a computationally adequate model of the language PCF (simply typed lambda calculus with fixed points). This model was intensional in that it could distinguish between computations computing the same result using a different number of fixed point unfoldings.
In this work we show how also programming languages with recursive types can be given denotational semantics in type theory with guarded recursion. More precisely, we give a computationally adequate denotational semantics to the language FPC (simply typed lambda calculus extended with recursive types), modelling recursive types using guarded recursive types. The model is intensional in the same way as was the case in previous work, but we show how to recover extensionality using a logical relation.
All constructions and reasoning in this paper, including proofs of theorems such as soundness and adequacy, are by (informal) reasoning in type theory, often using guarded recursion.

Contents

† This research was supported by The Danish Council for Independent Research for the Natural Sciences (FNU), Grant no. 4002-00442.

1. Introduction

Recent years have seen great advances in formalisation of mathematics in type theory, in particular with the development of homotopy type theory [Uni13]. Such formalisations are an important step towards machine assisted verification of mathematical proofs. Rather than adapting classical set theory based mathematics to type theory, new synthetic approaches sometimes offer simpler and clearer presentations in type theory, as illustrated by the development of synthetic homotopy theory.

Just like any other branch of mathematics, domain theory and denotational semantics for programming languages with recursion should be formalised in type theory and, as was the case of homotopy theory, synthetic approaches can provide clearer and more abstract proofs.

Guarded recursion [Nak00] can be seen as a synthetic form of domain theory or, perhaps more accurately, a synthetic form of step-indexing [Bir+12; App+07]. Recent work has shown how guarded recursion can be used to construct syntactic models and operational reasoning principles for (also combinations of) advanced programming language features including general references, recursive types, countable non-determinism and concurrency [Bir+12; BBM14; SB14]. Our hope is that synthetic guarded domain theory can also provide denotational models of these features.

1.1. *Synthetic guarded domain theory*

The synthetic approach to domain theory is to assume that types are domains, rather than constructing a notion of domain as a type equipped with a certain structure. To model recursion a fixed point combinator is needed, but adding unrestricted fixed points makes the type theory inconsistent when read as a logical system. The approach of guarded recursion is to introduce a new type constructor $\triangleright$, pronounced "later". Elements of $\triangleright A$ are to be thought of as elements of type A available only one time step from now, and the introduction form $\mathsf{next}\colon A \to \triangleright A$ makes anything available now, available later. The fixed point operator has type

$$\mathsf{fix}\colon (\triangleright A \to A) \to A$$

and maps an f to a fixed point of $f \circ \mathsf{next}$. Guarded recursion also assumes solutions to all guarded recursive type equations, i.e., equations where all occurences of the type variable are under a $\triangleright$, as for example in the equation

$$LA \cong A + \triangleright LA \tag{1}$$

used to define the lifting monad L below, but guarded recursive equations can also have negative or even non-functorial occurences. Guarded recursion can be proved consistent with type theory using the topos of trees model and related variants [Bir+12; BM15; Biz+16]. In this paper we will be working in guarded dependent type theory (gDTT) [Biz+16], an extensional type theory with guarded recursion.

In previous work [PMB15], we initiated a study of denotational semantics inside guarded dependent type theory, constructing a model of PCF (simply typed lambda calculus with fixed points). By carefully aligning the fixpoint unfoldings of PCF with the steps of the metalanguage (represented by $\triangleright$), we proved a computational adequacy result for the model inside type theory. Guarded recursive types were used both in the denotational semantics (to define a lifting monad) and in the proof of computational adequacy. Likewise, the fixed point operator fix of gDTT was used both to model fixed points of PCF and as a proof principle.

1.2. *Contributions*

Here we extend our previous work in two ways. First we extend the denotational semantics and adequacy proof to languages with recursive types. More precisely, we consider the language FPC (simply typed lambda calculus extended with general recursive types), modelling recursive types using guarded recursive types. The proof of computational adequacy shows an interesting aspect of guarded domain theory. It uses a logical relation between syntax and semantics defined by induction over the structure of types. The case of recursive types requires a solution to a recursive type equation. In the setting of classical domain theory, the existence of this solution requires a separate argument [Pit96], but here it is simply a guarded recursive type.

The second contribution is a relation capturing extensionally equal elements in the model. Like the model for PCF in our previous work, the model for FPC constructed here distinguishes between programs computing the same value using a different number

of fixed point unfoldings. We construct a relation on the interpretation of types, relating elements that only differ by a finite number of computation steps. The relation is proved sound, meaning that, if the denotations of two terms are related, then the terms are contextually equivalent.

All constructions and proofs are carried out working informally in gDTT. This work illustrates the strength of gDTT, and indeed influenced the design of the type theory.

1.3. *Related work*

Escardó constructs a model of PCF using a category of ultrametric spaces [Esc99]. Since this category can be seen as a subcategory of the topos of trees [Bir+12], our previous work on PCF is a synthetic version of Escardó's model. Escardó's model also distinguishes between computations computing the same value in a different number of steps, and captures extensional behaviour using a logical relation similar to the one constructed here. Escardó however, does not consider recursive types. Although Escardó's model was useful for intuitions, the synthetic construction in type theory presented here is very different, in particular the proof of adequacy, which here is formulated in guarded dependent type theory.

Synthetic approaches to domain theory have been developed based on a wide range of models dating back to [Hyl91; Ros86]. Indeed, the internal languages of these models can be used to construct models of FPC and prove computational adequacy [Sim02]. A more axiomatic approach was developed in Reus's work [Reu96] where an axiomatisation of domain theory is postulated a priori inside the Extended Calculus of Constructions. Another approach is to endow the types with additional structure [BKV09; Ben+10] similar to an extensional version of the lifting monad we used in this paper. Unlike guarded synthetic domain theory, these models do not distinguish between computations using different numbers of steps. On the other hand, with the success of guarded recursion for syntactic models, we believe that the guarded approach could model languages with more advanced features.

The lifting monad used in this paper is a guarded recursive variant of the partiality monad considered by among others [Dan12; Cap05; BKV09; Ben+10]. Danielsson also defines a weak bisimulation on this monad, similar to the one defined in Definition 6.2. As reported by Danielsson, working with the partiality monad requires convincing Agda of productivity of coinductive definitions using workarounds. Here, productivity is ensured by the type system for guarded recursion.

This is an extended version of a conference publication [MP16]. A number of proofs that were omitted from the previous version due to space restrictions have been included in this version. There is also a slight difference in approach: The conference version defined a big-step operational semantics equivalent to the transitive closure of the small-step operational semantics of Definition 3.2 below. This operational semantics synchronises the steps of FPC with those of the meta-language, and capturing this in a big-step semantics was quite tricky. Here, instead, we define a simpler big-step operational semantics and prove this equivalent to the "global" small-step semantics (Lemma 3.3).

The paper is organized as follows. Section 2 gives a brief introduction to the most important concepts of gDTT. More advanced constructions of the type theory are introduced as needed. Section 3 defines the encoding of FPC and its operational semantics in gDTT. The denotational semantics is defined and soundness is proved in Section 4. Computational adequacy is proved in Section 5, and the relation capturing extensional equivalence is defined in Section 6. We conclude and discuss future work in Section 7.

Acknowledgements. We thank Nick Benton, Lars Birkedal, Aleš Bizjak, and Alex Simpson for helpful discussions and suggestions.

2. Guarded recursion

In this paper we work informally within a type theory with dependent types, inductive types and guarded recursion. Although inductive types are not mentioned in [Biz+16] the ones used here can be safely added – as they can be modelled in the topos of trees model – and so the arguments of this paper can be formalised in gDTT. We start by recalling some core features of this theory. In fact, for the first part of the development, we will need just the features of [BM13], which corresponds to the fragment of gDTT with a single clock and no delayed substitutions. Quantification over clocks and delayed substitutions will be introduced later, when needed.

When working in type theory, we use $\equiv$ for judgemental equality of types and terms and $=$ for propositional equality (sometimes $=_A$ when we want to be explicit about the type). We also use $=$ for (external) set theoretical equality.

The type constructor $\triangleright$ introduced in Section 1.1 is an applicative functor in the sense of [MP08], which means that there is a map next of type $A \to \triangleright A$ and a "later application" $\circledast \colon \triangleright(A \to B) \to \triangleright A \to \triangleright B$ written infix, satisfying

$$\mathsf{next}(f) \circledast \mathsf{next}(t) \equiv \mathsf{next}(f(t)) \tag{2}$$

among other axioms (see also [BM13]). In particular, $\triangleright$ extends to a functor mapping $f \colon A \to B$ to $\lambda x \colon \triangleright A.\, \mathsf{next}(f) \circledast x$. Moreover, the $\triangleright$ operator distributes over the identity type as follows

$$\triangleright(t =_A u) \equiv (\mathsf{next}\, t =_{\triangleright A} \mathsf{next}\, u) \tag{3}$$

Guarded dependent type theory comes with universes in the style of Tarski. In this paper, we will just use a single universe $\mathcal{U}$. Readers familiar with [Biz+16] should think of this as $\mathcal{U}_\kappa$, but since we work with a unique clock κ, we will omit the subscript. The universe comes with codes for type operations, including $\widehat{+} \colon \mathcal{U} \times \mathcal{U} \to \mathcal{U}$ for binary sum types, codes for dependent sums and products, and $\widehat{\triangleright} \colon \triangleright\mathcal{U} \to \mathcal{U}$ satisfying $\mathsf{El}(\widehat{\triangleright}(\mathsf{next}(A))) \equiv \triangleright\,\mathsf{El}(A)$, where we use $\mathsf{El}(A)$ for the type corresponding to an element $A \colon \mathcal{U}$. The type of $\widehat{\triangleright}$ allows us to solve recursive type equations using the fixed point combinator. For example, if A is small, i.e., has a code $\widehat{A}$ in $\mathcal{U}$, the type equation (1) can be solved by computing a code of LA as

$$\mathsf{fix}(\lambda X \colon \triangleright\mathcal{U}.\, \widehat{+}(\widehat{A}, \widehat{\triangleright} X)) \tag{4}$$

In this paper, we will only apply the monad L to small types A.

To ease presentation, we will usually not distinguish between types and type operations on the one hand, and their codes on the other. We generally leave El implicit.

2.1. *The topos of trees model*

The topos of trees model of guarded recursion [Bir+12] provides useful intuitions, and so we briefly recall it.

In the model, a closed type is modelled as a family of sets $X(n)$ indexed by natural numbers together with restriction maps $r_n^X : X(n+1) \to X(n)$ as in the following diagram

$$X(1) \longleftarrow X(2) \longleftarrow X(3) \longleftarrow X(4) \longleftarrow \dots$$

The $\triangleright$ type operator is modelled as $\triangleright X(1) = 1$, $\triangleright X(n+1) = X(n)$. Intuitively, $X(n)$ is the nth approximation for computations of type X, thus $X(n)$ describes the type X as it looks if we have n computational steps to reason about it.

Using the proposition-as-types principle, types like $\triangleright^{42} 0$ are non-standard truth values. Intuitively, this is the truthvalue of propositions that appear true for 42 computation steps, but then are falsified after 43.

For guarded recursive type equations, $X(n)$ describes the nth unfolding of the type equation. For example, fixing an object A, the unique solution to (1) is

$$LA(n) = 1 + A(1) + \dots + A(n)$$

with restriction maps defined using the restriction maps of A. In particular, if A is a constant presheaf, i.e., $A(n) = X$ for some fixed X and r_n^A identities, then we can think of $LA(n)$ as $\{0, \dots, n-1\} \times X + \{\bot\}$. The set of global elements of LA is then isomorphic to $\mathbb{N} \times X + \{\bot\}$. In particular, if $X = 1$, the set of global elements is $\bar{\omega}$, the natural numbers extended with a point at infinity.

Thus the global elements of LA correspond to the elements of Capretta's partiality monad [Cap05] L^{gl} defined as the coinductive solution to the type equation

$$L^{\mathrm{gl}} A \cong A + L^{\mathrm{gl}} A \tag{5}$$

3. FPC

This section defines the syntax, typing judgements and operational semantics of FPC. These are inductive types in guarded type theory, but, as mentioned earlier, we work informally in type theory, and in particular remain agnostic with respect to choice of representation of syntax with binding.

The typing judgements of FPC are defined in an entirely standard way. The grammar for terms of FPC

$$\begin{aligned} L, M, N ::= {} & \langle\rangle \mid x \mid \mathtt{inl}\ M \mid \mathtt{inr}\ M \\ & \mid \mathtt{case}\ L\ \mathtt{of}\ x_1.M; x_2.N \mid \langle M, N \rangle \\ & \mid \mathtt{fst}\ M \mid \mathtt{snd}\ M \\ & \mid \lambda x : \tau.M \mid MN \mid \mathtt{fold}\ M \mid \mathtt{unfold}\ N \end{aligned}$$

$$\Theta \in \text{Type Contexts} \stackrel{\text{def}}{=\!=} \langle\rangle \mid \langle \Theta, \alpha \rangle$$

$$\frac{}{\vdash \langle\rangle} \qquad \frac{\vdash \Theta}{\vdash \Theta, \alpha} \alpha \notin \Theta$$

$$\frac{\vdash \Theta}{\Theta \vdash \Theta_i} 1 \leq i \leq |\, \Theta \,| \qquad \frac{\vdash \Theta}{\Theta \vdash 1}$$

$$\frac{\Theta, \alpha \vdash \tau}{\Theta \vdash \mu\alpha.\tau} \qquad \frac{\Theta \vdash \tau_1 \quad \Theta \vdash \tau_2}{\Theta \vdash \tau_1 \,\mathtt{op}\, \tau_2} \text{ for } \mathtt{op} \in \{+, \times, \to\}$$

Fig. 1. Rules for welformed FPC types

$$\Gamma \in \text{Expression Contexts} \stackrel{\text{def}}{=\!=} \langle\rangle \mid \langle \Gamma, x : \tau \rangle$$

$$\frac{\vdash \Theta}{\Theta \vdash \langle\rangle} \qquad \frac{\Theta \vdash \Gamma \quad \Theta \vdash \tau}{\Theta \vdash \Gamma, x : \tau} x \notin \Gamma$$

Fig. 2. Rules for welformed FPC contexts

should be read as an inductive type of terms in the standard way. Likewise the grammars for types and contexts and the typing judgements defined in Figures 1, 2 and 3 should be read as defining inductive types in type theory, allowing us to do proofs by induction over e.g. typing judgements.

We denote by $\mathtt{Type}_{\text{FPC}}$, $\mathtt{Term}_{\text{FPC}}$ and $\mathtt{Value}_{\text{FPC}}$ the types of *closed* FPC types and terms, and values of FPC and by $\mathtt{OTerm}_{\text{FPC}}$ the type of all (also open) terms. By a value we mean a closed term matching the grammar

$$v ::= \langle\rangle \mid \mathtt{inl}\ M \mid \mathtt{inr}\ M \mid \langle M, N \rangle \mid \lambda x : \tau.M \mid \mathtt{fold}\ M$$

$$\frac{x : \sigma \in \Gamma \qquad \cdot \vdash \Gamma}{\Gamma \vdash x : \sigma} \qquad \frac{}{\Gamma \vdash \langle\rangle : 1}$$

$$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash (\lambda x : \sigma.M) : \sigma \to \tau}$$

$$\frac{\Gamma \vdash M : \sigma \to \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau}$$

$$\frac{\Gamma \vdash e : \tau_1}{\Gamma \vdash \mathtt{inl}\ e : \tau_1 + \tau_2} \qquad \frac{\Gamma \vdash e : \tau_2}{\Gamma \vdash \mathtt{inr}\ e : \tau_1 + \tau_2}$$

$$\frac{\Gamma \vdash L : \tau_1 + \tau_2 \quad \Gamma, x_1 : \tau_1 \vdash M : \sigma \quad \Gamma, x_2 : \tau_2 \vdash N : \sigma}{\Gamma \vdash \mathtt{case}\ L\ \mathtt{of}\ x_1.M; x_2.N : \sigma}$$

$$\frac{\Gamma \vdash M : \tau_1 \times \tau_2}{\Gamma \vdash \mathtt{fst}\ M : \tau_1} \qquad \frac{\Gamma \vdash M : \tau_1 \times \tau_2}{\Gamma \vdash \mathtt{snd}\ e : \tau_2}$$

$$\frac{\Gamma \vdash M : \tau_1 \quad \Gamma \vdash N : \tau_2}{\Gamma \vdash \langle M, N \rangle : \tau_1 \times \tau_2}$$

$$\frac{\Gamma \vdash M : \mu\alpha.\tau}{\Gamma \vdash \mathtt{unfold}\ M : \tau[\mu\alpha.\tau/\alpha]} \qquad \frac{\Gamma \vdash M : \tau[\mu\alpha.\tau/\alpha]}{\Gamma \vdash \mathtt{fold}\ M : \mu\alpha.\tau}$$

Fig. 3. Typing rules for FPC terms

$$\frac{}{v \Downarrow^0 v}$$

$$\frac{L \Downarrow^k \mathtt{inl}\ L' \quad M[L'/x_1] \Downarrow^m v}{\mathtt{case}\ L\ \mathtt{of}\ x_1.M; x_2.N \Downarrow^{m+k} v} \qquad \frac{L \Downarrow^k \mathtt{inr}\ L' \quad N[L'/x_2] \Downarrow^m v}{\mathtt{case}\ L\ \mathtt{of}\ x_1.M; x_2.N \Downarrow^{m+k} v}$$

$$\frac{L \Downarrow^k \langle M, N\rangle \quad M \Downarrow^l v}{\mathtt{fst}\ L \Downarrow^{k+l} v} \qquad \frac{L \Downarrow^k \langle M, N\rangle \quad N \Downarrow^l v}{\mathtt{snd}\ L \Downarrow^{k+l} v}$$

$$\frac{M \Downarrow^k \lambda x.L \quad L[N/x] \Downarrow^l v}{MN \Downarrow^{k+l} v}$$

$$\frac{M \Downarrow^k \mathtt{fold}\ N \quad N \Downarrow^m v}{\mathtt{unfold}\ M \Downarrow^{k+m+1} v}$$

Fig. 4. The big-step operational semantics.

3.1. *Big-step semantics*

We now define a big-step call-by-name operational semantics for FPC as relations between closed terms and values as in Figure 4. Since the denotational semantics of FPC are intensional, counting reduction steps, it is necessary to also count the steps in the operational semantics in order to state the soundness and adequacy theorems precisely. More precisely, the semantics counts the number of `unfold-fold` reductions in the same fashion in which Escardó counted fix-point reduction for PCF. The big-step relation is defined as an inductive type. The statement

$$M \Downarrow^k v \tag{6}$$

where M is a term, k a natural number, and v a value, should be read as 'M evaluates in k steps to a value v. We can define more standard big-step evaluation predicates as follows

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

3.2. *Small-step semantics*

Figure 5 defines the reductions of the small-step call-by-name operational semantics.

The semantics is trivially deterministic.

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

We define two transitive closures of the small-step semantics. The first is a defined in Figure 6 which should be read as a standard inductive type.

The second transitive closure synchronises the steps of FPC with those of the metalogic. This is needed for the statement of the soundness and adequacy theorems, and also allows for guarded recursion to be used in the proofs of these.

Definition 3.2. The guarded small step semantics $M \Rightarrow^k N$ is defined by induction on

$$(\lambda x : \sigma.M)(N) \to^0 M[N/x] \qquad \texttt{unfold}\ (\texttt{fold}\ M) \to^1 M$$
$$\texttt{case}\ (\texttt{inl}\ L)\ \texttt{of}\ x_1.M; x_2.N \to^0 M[L/x_1]$$
$$\texttt{case}\ (\texttt{inr}\ L)\ \texttt{of}\ x_1.M; x_2.N \to^0 N[L/x_2]$$
$$\texttt{fst}\ \langle M, N\rangle \to^0 M \qquad \texttt{snd}\ \langle M, N\rangle \to^0 N$$
$$\frac{M_1 \to^k M_2}{E[M_1] \to^k E[M_2]}$$

$$E ::= [\cdot] \mid EM \mid \texttt{case}\ E\ \texttt{of}\ x_1.M; x_2.N \mid \texttt{fst}\ E \mid \texttt{snd}\ E \mid \texttt{unfold}\ E$$

Fig. 5. Reductions of the small-step call-by-name operational semantics. In the last rule, k is either 0 or 1.

$$\frac{}{M \to_*^0 M} \qquad \frac{M \to^k M' \qquad M' \to_*^m N}{M \to_*^{k+m} N}$$

Fig. 6. Transitive closure on small-step semantics

k as

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

The more standard small-step evaluation of terms to values can be defined as

$$M \Rightarrow N \stackrel{\text{def}}{=\!=} \Sigma k.M \Rightarrow^k N$$

The two forms of transitive closures and the big-step operational semantics are related in Lemma 3.3 below.

3.3. *Examples*

As an example of a recursive FPC type, one can encode the natural numbers as

$$\mathbf{nat} \stackrel{\text{def}}{=\!=} \mu\alpha.1 + \alpha$$
$$\mathsf{zero} \stackrel{\text{def}}{=\!=} \texttt{fold}\ (\texttt{inl}\ (\langle\rangle))$$
$$\mathsf{succ}\ M \stackrel{\text{def}}{=\!=} \texttt{fold}\ (\texttt{inr}\ (M))$$

Using this definition we can define the term ifz of PCF. If L is a term of type **nat** and M,N are terms of type σ define ifz as

$$\mathsf{ifz}\ L\ M\ N \stackrel{\text{def}}{=\!=} \texttt{case}\ (\texttt{unfold}\ L)\ \texttt{of}\ x_1.M; x_2.N$$

where x_1, x_2 are fresh. It is easy to see that $\mathsf{ifz}\ \mathsf{zero}\ M\ N \Rightarrow^k v$ iff $\triangleright(M \Rightarrow^{k-1} v)$ and that $\mathsf{ifz}\ (\mathsf{succ}\ L)\ M\ N \Rightarrow^k v$ iff $\triangleright(N \Rightarrow^{k-1} v)$ for any L term of type **nat**. For example, $\mathsf{ifz}\ 1\ 0\ 1 \Rightarrow^2 42$ is $\triangleright 0$.

Recursive types introduce divergent terms. For example, given a type A, the Turing fixed point combinator on A can be encoded as follows:

$$
\begin{aligned}
&B \stackrel{\text{def}}{=\!=} \mu\alpha.(\alpha \to (A \to A) \to A)\\
&\theta : B \to (A \to A) \to A\\
&\theta \stackrel{\text{def}}{=\!=} \lambda x \lambda y.y(\mathtt{unfold}\ x\ x\ y)\\
&\mathsf{Y}_A \stackrel{\text{def}}{=\!=} \theta(\mathtt{fold}\ \theta)
\end{aligned}
$$

An easy induction shows that $(\mathsf{Y}_\sigma\ (\lambda x.x) \Rightarrow^k v) = \triangleright^k 0$, where 0 is the empty type.

If $M \to_*^k v$ with v a value and M a term, then

— $M \Rightarrow^k v$ is true
— $M \Rightarrow^n v$ is logically equivalent to $\triangleright^{\mathtt{min}(n,k)} 0$ if $n \neq k$, where 0 is the empty type

If, on the other hand, M is divergent in the sense that for any k there exists an N such that $M \to_*^k N$, then $M \Rightarrow^n v$ is equivalent to $\triangleright^n 0$.

3.4. *Operational semantics correspondence*

We would like to prove now that the standard inductive definition of big-step semantics defined Figure 6 coincides with our guarded small-step semantics in Definition 3.2. However, we cannot prove this directly. In fact, as stated in the example above, for a divergent term M, the statement $M \to_*^k v$ is false. Semantically, this is a constant object in the model. On the other hand $M \Rightarrow^k v$ is logically equivalent to $\triangleright^k 0$. Which is true for the first k steps in the model.

To relate the two semantics, we need a construction that gives a "global" view of a type, as can be computed using infinitely many steps. The global viewpoint will allow for the $\triangleright$ operators to be removed from a type up to logical equivalence.

3.4.1. *Universal quantification over clocks* The global view of a type is given by universal quantification over clocks in the sense of Atkey and McBride [AM13; Møg14]. We now briefly recall this as implemented in gDTT, referring to [Biz+16] for details.

In gDTT all types and terms are typed in a clock context, i.e., a finite set of names of clocks. For each clock κ, there is a type constructor $\triangleright_\kappa$, a fixed point combinator, and so on. The development of this paper so far has been in a context of a single implicit clock κ.

If A is a type in a context where κ does not appear, one can form the type $\forall\kappa.A$, binding κ. This construction behaves in many ways similarly to polymorphic quantification over types in System F. There is an associated binding introduction form $\Lambda\kappa.(-)$ (applicable to terms, where κ does not appear free in the context), and elimination form $t[\kappa']$ having type $A[\kappa'/\kappa]$ whenever $t\colon \forall\kappa.A$.

The type system allows for a restricted elimination rule for $\triangleright$. If t is of type $\triangleright_\kappa A$ in a

context where κ does not appear free, then $\mathsf{prev}\,\kappa.t$ has type $\forall\kappa.A$. Using $\mathsf{prev}\,\kappa.$ we can define a term force:

$$\begin{aligned} &\mathsf{force} : (\forall\kappa.\triangleright_\kappa A) \to \forall\kappa.A \\ &\mathsf{force} \stackrel{\text{def}}{=\!=} \lambda x.\, \mathsf{prev}\,\kappa.x[\kappa] \end{aligned} \tag{7}$$

For types A and B we say the two are type isomorphic if there exist two terms $f : A \to B$ and $g : B \to A$ such that $f(g(x)) \equiv x$ and $g(f(x)) = x$. When A is a type that does not mention any clock variable, $\forall\kappa.A$ is isomorphic to A. The map $A \to \forall\kappa.A$ is simply $\lambda x\colon A.\Lambda\kappa.x$. The other direction is given by application to a clock constant κ_0, which we assume exists. These can be proved to be inverses of each other using the clock irrelevance axiom. We refer to [Biz+16] for details.

Using force and the isomorphism $\forall\kappa.0 \cong 0$, one can prove that $\forall\kappa.\triangleright^n_\kappa 0$ is logically equivalent to 0. Recall that $\triangleright^n_\kappa 0$ is not equivalent to 0. When κ is not free in A and B is a type dependent on $x : A$ the following type isomorphism is derivable from the β and η rules of the constructors involved [Biz+16]

$$\forall\kappa.\Sigma(x : A).B \cong \Sigma(x : A).\forall\kappa.B \tag{8}$$

Semantically, gDTT is modelled by taking the limit of an object in the topos of trees, i.e., by taking the set of global elements of an object. This is a semantic justification for the terminology of the global view of a type. Note that this limit is a set, rather than an object in the topos of trees. Semantics of types and terms in contexts of more than one clock variable requires constructing a family of presheaf categories indexed over clock contexts. We refer to [BM15] for details.

As stated above, all the earlier development can be read as being done in a setting of a single clock κ. In the rest of the paper we shall continue leaving the clock κ implicit in most contexts, writing e.g. $\triangleright$ rather than $\triangleright_\kappa$.

3.4.2. *Relating the operational semantics* We can now state a relation between the three operational semantics given above.

Lemma 3.3. Let M and N be FPC terms, v a value and k a natural number. Then

1 $M \Downarrow^k v$ iff $M \to^k_* v$
2 $M \to^k_* N$ iff $\forall\kappa.M \Rightarrow^k N$

Proof. The first statement is a almost a textbook result on operational semantics, and we omit the proof.

For the second statement the proof from left to right is by induction on $M \to^k_* N$. The case of $M = N$ is trivial, so consider the case when $M \to^k M'$ and $M' \to^m_* N$. When $k = 0$, by definition $M \to^0_* M'$, and by induction hypothesis we know that $\forall\kappa.M' \Rightarrow^m N$. Thus, $M \Rightarrow^m N$ holds for any κ, and so also $\forall\kappa.M \Rightarrow^m N$, since κ is not free in the assumption $M \to^k_* N$. When $k = 1$ by induction hypothesis $\forall\kappa.M' \Rightarrow^m N$ and thus, for any κ, $M \to^1 M'$ and $\triangleright_\kappa(M' \Rightarrow^m N)$. As before, this allows us to conclude $\forall\kappa.M \Rightarrow^{m+1} N$.

The right to left implication is proved by induction on k. When $k = 0$ the clock κ is not free in $M \Rightarrow^k N$ and so $\forall\kappa.M \Rightarrow^k N$ is isomorphic to $M \Rightarrow^k N$, which implies $M \to^k_* N$.

When $k = k' + 1$ the assumption $\forall\kappa.M \Rightarrow^k N$ implies that $M \to_*^0 N'$, $N' \to^1 N''$ and $\forall\kappa.\triangleright_\kappa(N'' \Rightarrow^{k'} N)$. By the type isomorphism (7) the latter implies $\forall\kappa.(N'' \Rightarrow^{k'} N)$, which by the induction hypothesis implies $N'' \to_*^{k'} N$. Thus we conclude $M \to_*^k N$. □

4. Denotational Semantics

We now define the denotational semantics of FPC. First we recall the definition of the guarded recursive version of the *lifting monad* on types from [PMB15]. This is defined as the *unique* solution to the guarded recursive type equation

$$LA \cong A + \triangleright LA$$

which exists because the recursive variable is guarded by a $\triangleright$. Recall (Section 2) that guarded recursive types are defined as fixed points of endomaps on the universe, so LA is only defined for small types A. We will only apply L to small types in this paper.

The isomorphism induces a map $\theta_{LA} : \triangleright LA \to LA$ and a map $\eta\colon A \to LA$. An element of LA is either of the form $\eta(a)$ or $\theta(r)$. We think of these cases as values "now" or computations that "tick". Moreover, given $f : A \to B$ with B a $\triangleright$-algebra (i.e., equipped with a map $\theta_B\colon \triangleright B \to B$), we can lift f to a homomorphism of $\triangleright$-algebras $\hat{f}\colon LA \to B$ as follows

$$\begin{aligned} \hat{f}(\eta(a)) &\stackrel{\text{def}}{=\!=} f(a) \\ \hat{f}(\theta(r)) &\stackrel{\text{def}}{=\!=} \theta_B(\mathsf{next}(\hat{f}) \circledast r) \end{aligned} \tag{9}$$

Formally $\hat{f}$ is defined as a fixed point of a term of type $\triangleright(LA \to B) \to LA \to B$. Recall that $\lambda r.\, \mathsf{next}(\hat{f}) \circledast r$ is the application of the functor $\triangleright$ to the map $\hat{f}$, thus $\hat{f}$ is an algebra homomorphism.

Intuitively LA is the type of computations possibly returning an element of A, recording the number of steps used in the computation. We can define the divergent computation as $\bot \stackrel{\text{def}}{=\!=} \mathsf{fix}(\theta)$ and a "delay" map δ_{LA} of type $LA \to LA$ for any A as $\delta_{LA} \stackrel{\text{def}}{=\!=} \theta_{LA} \circ \mathsf{next}$. The latter can be thought of as adding a step to a computation. The lifting L extends to a functor. For a map $f : A \to B$ the action on morphisms can be defined using the unique extension as $L(f) \stackrel{\text{def}}{=\!=} \widehat{\eta \circ f}$.

4.1. *Interpretation of types*

A type judgement $\Theta \vdash \tau$ is interpreted as a map of type $\mathcal{U}^{|\Theta|} \to \mathcal{U}$, where $|\Theta|$ is the cardinality of the set of variables in Θ. This interpretation map is defined by a combination of induction and *guarded recursion* for the case of recursive types as in Figure 7.

More precisely, the case of recursive types is defined to be the fixed point of a map from $\triangleright(\mathcal{U}^{|\Theta|} \to \mathcal{U})$ to $\mathcal{U}^{|\Theta|} \to \mathcal{U}$ defined as follows:

$$\lambda X.\lambda\rho.\widehat{\triangleright}(\mathsf{next}([\![\tau]\!]) \circledast \mathsf{next}(\rho) \circledast (X \circledast \mathsf{next}(\rho))) \tag{10}$$

$$
\begin{aligned}
[\![\Theta \vdash \alpha]\!]\,(\rho) &\stackrel{\text{def}}{=\!=} \rho(\alpha)\\
[\![\Theta \vdash 1]\!]\,(\rho) &\stackrel{\text{def}}{=\!=} L1\\
[\![\Theta \vdash \tau_1 \times \tau_2]\!]\,(\rho) &\stackrel{\text{def}}{=\!=} [\![\Theta \vdash \tau_1]\!]\,(\rho) \times [\![\Theta \vdash \tau_2]\!]\,(\rho)\\
[\![\Theta \vdash \tau_1 + \tau_2]\!]\,(\rho) &\stackrel{\text{def}}{=\!=} L([\![\Theta \vdash \tau_1]\!]\,(\rho) + [\![\Theta \vdash \tau_2]\!]\,(\rho))\\
[\![\Theta \vdash \tau_1 \to \tau_2]\!]\,(\rho) &\stackrel{\text{def}}{=\!=} [\![\Theta \vdash \tau_1]\!]\,(\rho) \to [\![\Theta \vdash \tau_2]\!]\,(\rho)\\
[\![\Theta \vdash \mu\alpha.\tau]\!]\,(\rho) &\stackrel{\text{def}}{=\!=} \triangleright([\![\Theta, \alpha \vdash \tau]\!]\,(\rho, [\![\Theta \vdash \mu\alpha.\tau]\!]\,(\rho)))
\end{aligned}
$$

Fig. 7. Interpretation of FPC types

ensuring (using $\mathsf{El}(-)$ explicitly)

$$
\begin{aligned}
\mathsf{El}([\![\Theta \vdash \mu\alpha.\tau]\!]\,\rho) &\equiv \mathsf{El}(\widehat{\triangleright}(\mathsf{next}([\![\tau]\!]) \circledast \mathsf{next}(\rho) \circledast (\mathsf{next}([\![\Theta \vdash \mu\alpha.\tau]\!]) \circledast \mathsf{next}(\rho))))\\
&\equiv \mathsf{El}(\widehat{\triangleright}(\mathsf{next}([\![\tau]\!]\,(\rho, ([\![\Theta \vdash \mu\alpha.\tau]\!]\,\rho)))))\\
&\equiv \triangleright\,\mathsf{El}([\![\tau]\!]\,(\rho, ([\![\Theta \vdash \mu\alpha.\tau]\!]\,\rho)))
\end{aligned}
$$

The substitution lemma for types can be proved using guarded recursion in the case of recursive types.

Lemma 4.1 (Substitution Lemma for Types). Let σ be a well-formed type with variables in Θ and let ρ be of type $\mathcal{U}^{|\Theta|}$. If $\Theta, \beta \vdash \tau$ then

$$[\![\Theta \vdash \tau[\sigma/\beta]]\!]\,(\rho) = [\![\Theta, \beta \vdash \tau]\!]\,(\rho, [\![\Theta \vdash \sigma]\!]\,(\rho))$$

Proof. The proof is by induction on $\Theta, \beta \vdash \tau$. Most cases are straightforward, and we just show the case of $\Theta, \beta \vdash \mu\alpha.\tau$. The proof of this case is by *guarded recursion*, and thus we assume that

$$\triangleright([\![\Theta \vdash (\mu\alpha.\tau)[\sigma/\beta]]\!]\,(\rho) = [\![\Theta, \beta \vdash \mu\alpha.\tau]\!]\,(\rho, [\![\Theta \vdash \sigma]\!]\,(\rho))) \tag{11}$$

Assuming (without loss of generality) that α is not β we get the following series of equalities

$$
\begin{aligned}
[\![\Theta \vdash \mu\alpha.\tau[\sigma/\beta]]\!]\,(\rho) &= [\![\Theta \vdash \mu\alpha.(\tau[\sigma/\beta])]\!]\,(\rho)\\
&= \triangleright([\![\Theta, \alpha \vdash \tau[\sigma/\beta]]\!]\,(\rho, [\![\Theta \vdash \mu\alpha.(\tau[\sigma/\beta])]\!]\,(\rho)))\\
&= \triangleright([\![\Theta, \alpha, \beta \vdash \tau]\!]\,(\rho, [\![\mu\alpha.(\tau[\sigma/\beta])]\!]\,(\rho), [\![\Theta, \alpha \vdash \sigma]\!]\,(\rho, [\![\mu\alpha.(\tau[\sigma/\beta])]\!]\,(\rho))))\\
&= \triangleright([\![\Theta, \alpha, \beta \vdash \tau]\!]\,(\rho, [\![\mu\alpha.(\tau[\sigma/\beta])]\!]\,(\rho), [\![\Theta \vdash \sigma]\!]\,(\rho))) \qquad (12)
\end{aligned}
$$

using the induction hypothesis on τ and the fact that α is not free in σ. We want now to apply the guarded recursive hypothesis. To do this note that (12) is a guarded fixed point construction on universes as defined in Figure 7 and explained in (10). Thus, (12) is actually equal to

$$\mathsf{El}(\widehat{\triangleright}(\mathsf{next}([\![\Theta, \alpha, \beta \vdash \tau]\!]) \circledast \mathsf{next}(\rho) \circledast (\mathsf{next}([\![\mu\alpha.(\tau[\sigma/\beta])]\!]\,(\rho))) \circledast \mathsf{next}\,[\![\Theta \vdash \sigma]\!]\,(\rho))) \tag{13}$$

$$\begin{aligned}
\theta_1 &\stackrel{\text{def}}{=\!=} \lambda x : \triangleright [\![1]\!] . \theta_{L[\![1]\!]}(x) \\
\theta_{\tau_1 \times \tau_2} &\stackrel{\text{def}}{=\!=} \lambda x : \triangleright [\![\tau_1 \times \tau_2]\!] . \langle \theta_{\tau_1}(\triangleright(\pi_1)(x)), \theta_{\tau_2}(\triangleright(\pi_2)(x)) \rangle \\
\theta_{\tau_1 + \tau_2} &\stackrel{\text{def}}{=\!=} \lambda x : \triangleright [\![\tau_1 + \tau_2]\!] . \theta_{L[\![\tau_1 + \tau_2]\!]}(x) \\
\theta_{\sigma \to \tau} &\stackrel{\text{def}}{=\!=} \lambda f : \triangleright([\![\sigma]\!] \to [\![\tau]\!]) . \lambda x : [\![\sigma]\!] . \theta_\tau(f \circledast (\mathsf{next}(x))) \\
\theta_{\mu\alpha.\tau} &\stackrel{\text{def}}{=\!=} \lambda x : \triangleright [\![\mu\alpha.\tau]\!] . \, \mathsf{next}(\theta_{\tau[\mu\alpha.\tau/\alpha]}) \circledast (x)
\end{aligned}$$

Fig. 8. Definition of $\theta_\sigma : \triangleright [\![\sigma]\!] \to [\![\sigma]\!]$

modulo rewritings using (2). By (3), (11) implies

$$\mathsf{next}([\![\Theta \vdash (\mu\alpha.\tau)[\sigma/\beta]]\!]\,(\rho)) = \mathsf{next}([\![\Theta, \beta \vdash \mu\alpha.\tau]\!]\,(\rho, [\![\Theta \vdash \sigma]\!]\,(\rho)))$$

and so we can rewrite (12) as

$$\triangleright([\![\Theta, \alpha, \beta \vdash \tau]\!]\,(\rho, [\![\Theta, \beta \vdash \mu\alpha.\tau]\!]\,(\rho, [\![\Theta \vdash \sigma]\!]\,(\rho)), [\![\Theta \vdash \sigma]\!]\,(\rho)))$$

By switching the order of the arguments we get

$$\triangleright([\![\Theta, \beta, \alpha \vdash \tau]\!]\,(\rho, [\![\Theta \vdash \sigma]\!]\,(\rho), [\![\Theta, \beta \vdash \mu\alpha.\tau]\!]\,(\rho, [\![\Theta \vdash \sigma]\!]\,(\rho))))$$

and by definition this is equal to

$$[\![\Theta, \beta \vdash \mu\alpha.\tau]\!]\,(\rho, [\![\Theta \vdash \sigma]\!]\,(\rho))$$

thus concluding the case and the proof. □

The following lemma follows directly from the substitution lemma.

Lemma 4.2. For all types τ and environments ρ of type $\mathcal{U}^{|\Theta|}$,

$$[\![\Theta \vdash \mu\alpha.\tau]\!]\,(\rho) = \triangleright\,[\![\Theta \vdash \tau[\mu\alpha.\tau/\alpha]]\!]\,(\rho)$$

The interpretation of every *closed* type τ carries a $\triangleright$-algebra structure, i.e., a map $\theta_\tau : \triangleright [\![\tau]\!] \to [\![\tau]\!]$, defined by guarded recursion and structural induction on τ as in Figure 8. The case of recursive types is welltyped by Lemma 4.2, and can be formally constructed as a fixed point of a term of type

$$G : \triangleright(\Pi\sigma : \mathtt{Type}_{\text{FPC}} . (\triangleright [\![\sigma]\!] \to [\![\sigma]\!])) \to \Pi\sigma.(\triangleright [\![\sigma]\!] \to [\![\sigma]\!])$$

as follows. Suppose $F : \triangleright(\Pi\sigma : \mathtt{Type}_{\text{FPC}} . (\triangleright [\![\sigma]\!] \to [\![\sigma]\!]))$, and define $G(F)$ essentially as in Figure 8 but with the clause $G(F)_{\mu\alpha.\tau}$ for recursive types being defined as

$$\lambda x : \triangleright [\![\mu\alpha.\tau]\!] . (F_{\tau[\mu\alpha.\tau/\alpha]} \circledast x) \tag{14}$$

Here F_σ is defined as $F \circledast \mathsf{next}(\sigma)$ using a generalisation of $\circledast$ to dependent products to be defined in Section 5.1. Define θ as the fixed point of G. Then

$$\begin{aligned}
\theta_{\mu\alpha.\tau}(x) &\equiv G(\mathsf{next}\,(\theta))_{\mu\alpha.\tau}(x) \\
&\equiv \mathsf{next}\,(\theta)_{\tau[\mu\alpha.\tau/\alpha]} \circledast (x)
\end{aligned} \tag{15}$$

Using the θ we define the delay operation which, intuitively, takes a computation and adds one step.

$$\delta_\sigma \stackrel{\text{def}}{=\!=} \theta_\sigma \circ \mathsf{next}\,.$$

4.2. *Interpretation of terms*

Figure 9 defines the interpretation of judgements $\Gamma \vdash M : \sigma$ as functions from $[\![\Gamma]\!]$ to $[\![\sigma]\!]$ where $[\![x_1 : \sigma_1, \cdots, x_n : \sigma_n]\!] \stackrel{\text{def}}{=\!=} [\![\sigma_1]\!] \times \cdots \times [\![\sigma_n]\!]$. In the case of `case`, the function $\widehat{f}$ is the extension of f to a homomorphism defined as in (9) above, using the fact that all types carry a $\triangleright$-algebra structure. The interpretation of `fold` is welltyped because $\mathsf{next}([\![M]\!]\,(\gamma))$ has type $\triangleright\,[\![\tau[\mu\alpha.\tau/\alpha]]\!]$ which by Lemma 4.2 is equal to $[\![\mu\alpha.\tau]\!]$. In the case of `unfold`, since $[\![M]\!]\,(\gamma)$ has type $[\![\mu\alpha.\tau]\!]$, which by Lemma 4.2 is equal to $\triangleright\,[\![\tau[\mu\alpha.\tau/\alpha]]\!]$, the type of $\theta_{\tau[\mu\alpha.\tau/\alpha]}([\![M]\!]\,(\gamma))$ is $[\![\tau[\mu\alpha.\tau/\alpha]]\!]$.

$$\begin{aligned}
[\![\Gamma \vdash t : \sigma]\!] &: [\![\Gamma]\!] \to [\![\sigma]\!] \\
[\![\Gamma \vdash x]\!]\,(\gamma) &\stackrel{\text{def}}{=\!=} \gamma(x) \\
[\![\Gamma \vdash \langle\rangle]\!]\,(\gamma) &\stackrel{\text{def}}{=\!=} \eta(*) \\
[\![\Gamma \vdash \langle M, N\rangle]\!]\,(\gamma) &\stackrel{\text{def}}{=\!=} \langle [\![M]\!]\,(\gamma), [\![N]\!]\,(\gamma)\rangle \\
[\![\Gamma \vdash \mathtt{fst}\ M]\!]\,(\gamma) &\stackrel{\text{def}}{=\!=} \pi_1([\![M]\!]\,(\gamma)) \\
[\![\Gamma \vdash \mathtt{snd}\ M]\!]\,(\gamma) &\stackrel{\text{def}}{=\!=} \pi_2([\![M]\!]\,(\gamma)) \\
[\![\Gamma \vdash \lambda x.M]\!]\,(\gamma) &\stackrel{\text{def}}{=\!=} \lambda x.\,[\![M]\!]\,(\gamma, x) \\
[\![\Gamma \vdash MN]\!]\,(\gamma) &\stackrel{\text{def}}{=\!=} [\![M]\!]\,(\gamma)\,[\![N]\!]\,(\gamma) \\
[\![\Gamma \vdash \mathtt{inl}\ E]\!]\,(\gamma) &\stackrel{\text{def}}{=\!=} \eta(\iota_1 [\![E]\!]\,(\gamma)) \\
[\![\Gamma \vdash \mathtt{inr}\ E]\!]\,(\gamma) &\stackrel{\text{def}}{=\!=} \eta(\iota_2 [\![E]\!]\,(\gamma)) \\
[\![\Gamma \vdash \mathtt{case}\ L\ \mathtt{of}\ x_1.M; x_2.N]\!]\,(\gamma) &\stackrel{\text{def}}{=\!=} \widehat{f}([\![L]\!]\,(\gamma)) \\
\text{where } f(\iota_1(x_1)) &\stackrel{\text{def}}{=\!=} [\![M]\!]\,(\gamma, x_1) \\
f(\iota_2(x_2)) &\stackrel{\text{def}}{=\!=} [\![N]\!]\,(\gamma, x_2) \\
[\![\Gamma \vdash \mathtt{fold}\ M]\!]\,(\gamma) &\stackrel{\text{def}}{=\!=} \mathsf{next}([\![M]\!]\,(\gamma)) \\
[\![\Gamma \vdash \mathtt{unfold}\ M]\!]\,(\gamma) &\stackrel{\text{def}}{=\!=} \theta_{\tau[\mu\alpha.\tau/\alpha]}([\![M]\!]\,(\gamma))
\end{aligned}$$

Fig. 9. Interpretation of FPC terms

Lemma 4.3. If $\Gamma \vdash M : \tau[\mu\alpha.\tau/\alpha]$ then $[\![\mathtt{unfold}\ (\mathtt{fold}\ M)]\!]\,(\gamma) = \delta_{\tau[\mu\alpha.\tau/\alpha]}\,[\![M]\!]\,(\gamma)$.

Proof. Straightforward by definition of the interpretation and by the type equality from Lemma 4.2. □

Lemma 4.4 (Substitution Lemma). Let $\Gamma \equiv x_1 : \sigma_1, \cdots, x_k : \sigma_k$ be a context such

that $\Gamma \vdash M : \tau$, and let $\Delta \vdash N_i : \sigma_i$ be a term for each $i = 1, \dots k$. If further $\gamma \in [\![\Delta]\!]$, then

$$\left[\!\left[\Delta \vdash M[\vec{N}/x] : \tau\right]\!\right](\gamma) = [\![\Gamma \vdash M : \tau]\!]\left(\left[\!\left[\Delta \vdash \vec{N} : \vec{\sigma}\right]\!\right](\gamma)\right)$$

Proof. The proof is a straightforward induction on the typing judgement $\Gamma \vdash M : \tau$. □

We now aim to show a soundness theorem for the interpretation of FPC. We do this by first showing soundness of the single step reduction as in the next lemma.

Lemma 4.5. Let M be a closed term of type τ. If $M \to^k N$ then $[\![M]\!](*) = \delta^k [\![N]\!](*)$

The two most complicated cases of the proof of Lemma 4.5 are captured in the following two lemmas.

Lemma 4.6.

1 The interpretation of `case` is a homomorphism of $\triangleright$-algebras in the first variable, i.e.,

$$\begin{aligned}&[\![\ \lambda x\colon \tau_1 + \tau_2.\texttt{case}\ x\ \texttt{of}\ x_1.M; x_2.N]\!](\gamma)(\theta(r))\\ =&\theta(\mathsf{next}([\![\ \lambda x\colon \tau_1 + \tau_2.\texttt{case}\ x\ \texttt{of}\ x_1.M; x_2.N]\!](\gamma)) \circledast r)\end{aligned}$$

2 If $[\![L]\!](\gamma) = \delta([\![L']\!](\gamma))$, then

$$[\![\texttt{case}\ L\ \texttt{of}\ x_1.M; x_2.N]\!](\gamma) = \delta\,[\![\texttt{case}\ L'\ \texttt{of}\ x_1.M; x_2.N]\!](\gamma)$$

Proof. For the proof of the first part, we use the notation $\widehat{f}$ as in Figure 9. Since $\widehat{f}$ is a homomorphism of $\triangleright$-algebras we get

$$\begin{aligned}[\![\lambda x.\texttt{case}\ x\ \texttt{of}\ x_1.M; x_2.N]\!](\gamma)(\theta_{\tau_1+\tau_2}(r)) &= \widehat{f}(\theta_{\tau_1+\tau_2}(r))\\ &= \theta_\sigma(\mathsf{next}(\widehat{f}) \circledast r)\\ &= \theta_\sigma(\mathsf{next}\,[\![\lambda x.\texttt{case}\ x\ \texttt{of}\ x_1.M; x_2.N]\!](\gamma) \circledast r)\end{aligned}$$

For the second part, note that $\widehat{f}$ is $[\![\lambda x.\texttt{case}\ x\ \texttt{of}\ x_1.M; x_2.N]\!](\gamma)$, so

$$\begin{aligned}[\![\texttt{case}\ L\ \texttt{of}\ x_1.M; x_2.N]\!](\gamma) &= \widehat{f}([\![L]\!](\gamma))\\ &= \widehat{f}(\delta_{\tau_1+\tau_2}([\![L']\!](\gamma)))\\ &= \widehat{f}(\theta_{\tau_1+\tau_2}(\mathsf{next}([\![L']\!](\gamma))))\\ &= \theta_\sigma(\mathsf{next}(\widehat{f}) \circledast (\mathsf{next}([\![L']\!](\gamma))))\\ &= \theta_\sigma(\mathsf{next}(\widehat{f}([\![L']\!](\gamma))))\\ &= \delta_\sigma([\![\texttt{case}\ L'\ \texttt{of}\ x_1.M; x_2.N]\!](\gamma))\end{aligned}$$

□

Lemma 4.7. If $\mu\alpha.\tau$ is a closed FPC type then

1 $[\![\ \lambda x\colon \mu\alpha.\tau.\texttt{unfold}\ x]\!](\theta_{\mu\alpha.\tau}(r)) = \theta_{\tau[\mu\alpha.\tau/\alpha]}(\mathsf{next}(\theta_{\tau[\mu\alpha.\tau/\alpha]}) \circledast r)$

2 If $[\![M]\!](\gamma) = \delta_{\mu\alpha.\tau}([\![M']\!](\gamma))$, then

$$[\![\texttt{unfold}\ M]\!](\gamma) = \delta_{\tau[\mu\alpha.\tau/\alpha]}([\![\texttt{unfold}\ M']\!](\gamma))$$

Proof. The interpretation for $[\![\ \lambda x\colon \mu\alpha.\tau.\mathtt{unfold}\ x]\!]\,(\theta_{\mu\alpha.\tau}(r))$ yields $\theta_{\tau[\mu\alpha.\tau/\alpha]}(\theta_{\mu\alpha.\tau}(r))$. This type checks as r has type $\triangleright[\![\mu\alpha.\tau]\!]$, thus $(\theta_{\mu\alpha.\tau}(r))$ has type $[\![\mu\alpha.\tau]\!]$ which – by Lemma 4.2 – is equal to $\triangleright[\![\tau[\mu\alpha.\tau/\alpha]]\!]$. Thus the term $\theta_{\tau[\mu\alpha.\tau/\alpha]}(\theta_{\mu\alpha.\tau}(r))$ has type $[\![\tau[\mu\alpha.\tau/\alpha]]\!]$. Now by definition of $\theta_{\mu\alpha.\tau}$ this is equal to $\theta_{\tau[\mu\alpha.\tau/\alpha]}(\mathsf{next}(\theta_{\tau[\mu\alpha.\tau/\alpha]}) \circledast (r))$ which is what we wanted.

For the second statement, we compute

$$\begin{aligned}
[\![\mathtt{unfold}\ M]\!]\,(\gamma) &= [\![\ \lambda x\colon \mu\alpha.\tau.\mathtt{unfold}\ x]\!]\,([\![M]\!]\,(\gamma)) \\
&= [\![\ \lambda x\colon \mu\alpha.\tau.\mathtt{unfold}\ x]\!]\,(\delta_{\mu\alpha.\tau}([\![M']\!]\,(\gamma))) \\
&= [\![\ \lambda x\colon \mu\alpha.\tau.\mathtt{unfold}\ x]\!]\,(\theta_{\mu\alpha.\tau}(\mathsf{next}([\![M']\!]\,(\gamma)))) \\
&= \theta_{\tau[\mu\alpha.\tau/\alpha]}(\mathsf{next}(\theta_{\tau[\mu\alpha.\tau/\alpha]}) \circledast (\mathsf{next}([\![M']\!]\,(\gamma)))) && \text{(statement 1)} \\
&= \theta_{\tau[\mu\alpha.\tau/\alpha]}(\mathsf{next}(\theta_{\tau[\mu\alpha.\tau/\alpha]}([\![M']\!]\,(\gamma)))) && \text{(rule (2))} \\
&= \theta_{\tau[\mu\alpha.\tau/\alpha]}(\mathsf{next}([\![\mathtt{unfold}\ M']\!]\,(\gamma))) \\
&= \delta_{\tau[\mu\alpha.\tau/\alpha]}([\![\mathtt{unfold}\ M']\!]\,(\gamma))
\end{aligned}$$

□

Proof of Lemma 4.5 The proof is by induction on $M \to^k N$. Most of the cases are straightforward, some (β-reductions for function and sum types) using the substitution lemma (Lemma 4.4). The case $\mathtt{unfold}\ (\mathtt{fold}\ M) \to^1 M$ follows directly from Lemma 4.3.

Now we prove the inductive cases. For the case $M_1 N \to^k M_2 N$ we know that by definition $[\![M_1 N]\!]\,(*) = [\![M_1]\!]\,(*)\,[\![N]\!]\,(*)$. By induction hypothesis we know that $[\![M_1]\!]\,(*) = \delta^k_{\sigma\to\tau}([\![M_2]\!]\,(*))$, thus $[\![M_1]\!]\,(*)\,[\![N]\!]\,(*) = (\delta^k_{\sigma\to\tau}([\![M_2]\!]\,(*)))\,[\![N]\!]\,(*)$ By definition of δ and θ this is equal to $\delta^k_\tau([\![M_2]\!]\,(*)\,[\![N]\!]\,(*))$.

In the case of

$$\mathtt{case}\ L\ \mathtt{of}\ x_1.M; x_2.N \to^k \mathtt{case}\ L'\ \mathtt{of}\ x_1.M; x_2.N$$

the induction hypothesis gives $[\![L]\!]\,(*) = \delta_{\tau_1+\tau_2}\,[\![L']\!]\,(*)$, and so Lemma 4.6 applies proving the case.

Finally, the case for $\mathtt{unfold}\ M \to^k \mathtt{unfold}\ M'$. If $k = 0$ the case follows trivially from the induction hypothesis. If $k = 1$, the step from the induction hypothesis to the case is exactly the second statement of Lemma 4.7. □

Proposition 4.8 (Soundness). Let M be a closed term of type τ. If $M \Rightarrow^k N$ then $[\![M]\!]\,(*) = \delta^k\,[\![N]\!]\,(*)$.

Proof. By induction on k. When $k = 0$ Lemma 4.5 applies concluding the case. When $k = n + 1$ by definition we have $M \to^0_* M'$, $M' \to^1 M''$ and $\triangleright(M'' \Rightarrow^n N)$. By repeated application of Lemma 4.5 we get $[\![M]\!]\,(*) = [\![M']\!]\,(*)$ and $[\![M']\!]\,(*) = \delta([\![M'']\!]\,(*))$. By induction hypothesis we get $\triangleright([\![M'']\!]\,(*) = \delta^n\,[\![N]\!]\,(*))$ which implies $\mathsf{next}([\![M'']\!]\,(*)) = \mathsf{next}(\delta^n\,[\![N]\!]\,(*)))$ and since $\delta = \theta \circ \mathsf{next}$, this implies $\delta([\![M'']\!]\,(*)) = \delta^k([\![N]\!]\,(*))$. By putting together the equations we get finally $[\![M]\!]\,(*) = \delta^k\,[\![N]\!]\,(*)$. □

5. Computational Adequacy

Computational adequacy is opposite implication of Proposition 4.8 in the case of terms of unit type. It is proved by constructing a (proof relevant) logical relation between syntax and semantics. The relation cannot be constructed just by induction on the structure of types, since in the case of recursive types, the unfolding can be bigger than the recursive type. Instead, the relation is constructed by guarded recursion: we assume the relation exists *later*, and from that assumption construct the relation *now* by structural induction on types. Thus the well-definedness of the logical relation is ensured by the type system of gDTT, more specifically by the rules for guarded recursion. This is in contrast to the classical proof in domain theory [Pit96], where existence requires a separate argument.

The logical relation uses a lifting of relations on values available now, to relations on values available later. To define this lifting, we need *delayed substitutions*, an advanced feature of gDTT.

5.1. *Delayed substitutions*

In gDTT, if $\Gamma, x\colon A \vdash B\ \mathsf{type}$ is a well formed type and t has type $\triangleright A$ in context Γ, one can form the type $\triangleright\,[x \leftarrow t]\,.B$. One motivation for this is to generalise $\circledast$ (described in Section 2) to a dependent version: if $f\colon \triangleright(\Pi(x : A).B)$, then $f \circledast t\colon \triangleright\,[x \leftarrow t]\,.B$. The idea is that t will eventually reduce to a term of the form $\mathsf{next}\,u$, and then $\triangleright\,[x \leftarrow t]\,.B$ will be equal to $\triangleright B[u/x]$. But if t is open, we may not be able to do this reduction yet.

More generally, we define the notion of *delayed substitution* as follows.
Suppose $\Gamma, x_1\colon A_1 \dots x_n\colon A_n$ is a wellformed context, and all A_i are independent, i.e., no x_j appears in an A_i. A delayed substitution $\xi\colon \Gamma \twoheadrightarrow x_1\colon A_1 \dots x_n\colon A_n$ is a vector of terms $\xi = [x_1 \leftarrow t_1, \dots, x_n \leftarrow t_n]$ such that $\Gamma \vdash t_i : A_i$. [Biz+16] gives a more general definition of delayed substitution allowing dependencies between the A_i's, but for this paper we just need the definition above.

If $\xi\colon \Gamma \twoheadrightarrow \Gamma'$ is a delayed substitution and $\Gamma, \Gamma' \vdash B\ \mathsf{type}$ is a wellformed type, then the type $\triangleright\xi.B$ is wellformed in context Γ. The introduction form states $\mathsf{next}\,\xi.u\colon \triangleright\xi.B$ if $\Gamma, \Gamma' \vdash u : B$.

In Figure 10 we recall some rules from [Biz+16] needed below. Of these, (16) and (17) can be considered β and η laws, and (18) is a weakening principle. Rules (16), (18) and (19) also have obvious versions for types, e.g.,

$$\triangleright\xi\,[x \leftarrow \mathsf{next}\,\xi.t]\,.B \equiv \triangleright\xi.(B[t/x]) \tag{23}$$

Rather than be taken as primitive, later application $\circledast$ can be defined using delayed substitutions as

$$g \circledast y \stackrel{\text{def}}{=\!=} \mathsf{next}\,[f \leftarrow g, x \leftarrow y]\,.f(x) \tag{24}$$

Note that with this definition, the rule $\mathsf{next}(f(t)) \equiv \mathsf{next}\,f \circledast \mathsf{next}\,t$ from Section 2 generalises to

$$\mathsf{next}\,\xi.(f\,t) \equiv (\mathsf{next}\,\xi.f) \circledast (\mathsf{next}\,\xi.t) \tag{25}$$

which follows from (16). In fact, later application generalises to the setting of delayed

$$\mathsf{next}\,\xi\,[x \leftarrow \mathsf{next}\,\xi.t]\,.u \equiv \mathsf{next}\,\xi.(u[t/x]) \tag{16}$$
$$\mathsf{next}\,\xi\,[x \leftarrow t]\,.x \equiv t \tag{17}$$
$$\mathsf{next}\,\xi\,[x \leftarrow t]\,.u \equiv \mathsf{next}\,\xi.u \tag{18}$$
$$\mathsf{next}\,\xi\,[x \leftarrow t, y \leftarrow u]\,\xi'.v \equiv \mathsf{next}\,\xi\,[y \leftarrow u, x \leftarrow t]\,\xi'.u \tag{19}$$
$$\mathsf{next}\,\xi.\,\mathsf{next}\,\xi'.u \equiv \mathsf{next}\,\xi'.\,\mathsf{next}\,\xi.u \tag{20}$$
$$(\mathsf{next}\,\xi.t =_{\triangleright\xi.A} \mathsf{next}\,\xi.s) \equiv \triangleright\xi.(t =_A s) \tag{21}$$
$$\mathsf{El}(\widehat{\triangleright}(\mathsf{next}\,\xi.A)) \equiv \triangleright\xi.\,\mathsf{El}(A) \tag{22}$$

Fig. 10. The notation $\xi\,[x \leftarrow t]$ means the extension of the delayed substitution ξ with $[x \leftarrow t]$. Rule (18) requires x not free in u. Rule (20) requires that none of the variables in the codomains of ξ and ξ' appear in the type of u, and that the codomains of ξ and ξ' are independent.

substitutions: if $g : \triangleright\xi.\Pi x : A.B$ and $y : \triangleright\xi.A$ define

$$g \circledast y \stackrel{\text{def}}{=\!=} \mathsf{next}\,\xi\,[f \leftarrow g, x \leftarrow y]\,.f(x) : \triangleright\xi\,[x \leftarrow y]\,.B \tag{26}$$

Note that in the special case where $y = \mathsf{next}\,\xi.u$ we get

$$g \circledast \mathsf{next}\,\xi.u : \triangleright\xi.B[u/x]$$

Rules (17), (18) and (20) imply

$$\begin{aligned}\mathsf{next}\,\xi\,[x \leftarrow t]\,.\,\mathsf{next}\,x &\equiv \mathsf{next}(\mathsf{next}\,\xi\,[x \leftarrow t]\,.x)\\ &\equiv \mathsf{next}(t)\\ &\equiv \mathsf{next}\,\xi\,[x \leftarrow t]\,.t\end{aligned}$$

which by (21) gives an inhabitant of

$$\triangleright\xi\,[x \leftarrow t]\,.(\mathsf{next}\,x = t) \tag{27}$$

5.2. *A logical relation between syntax and semantics*

Figure 11 defines the logical relation between syntax and semantics. It uses the following operation lifting relations $\mathcal{R}$ from A to B to relations $\triangleright\mathcal{R}$ from $\triangleright A$ to $\triangleright B$:

$$t \mathrel{\triangleright\mathcal{R}} u \stackrel{\text{def}}{=\!=} \triangleright\,[x \leftarrow t, y \leftarrow u]\,.(x \mathrel{\mathcal{R}} y) \tag{28}$$

As a consequence of (23) the following statement holds:

$$(\mathsf{next}\,\xi.t) \mathrel{\triangleright\mathcal{R}} (\mathsf{next}\,\xi.u) \equiv \triangleright\xi.(t \mathrel{\mathcal{R}} u) \tag{29}$$

This lifting operation can also be expressed on codes mapping $\mathcal{R} : A \to B \to \mathcal{U}$ to

$$\lambda x : \triangleright A, y : \triangleright B.\widehat{\triangleright}(\mathsf{next}\,[x' \leftarrow x, y' \leftarrow y]\,.(x' \mathrel{\mathcal{R}} y'))$$

in fact, this operation can be shown to factor as $F \circ \mathsf{next}$, for $F : \triangleright(A \to B \to \mathcal{U}) \to A \to B \to \mathcal{U}$ defined as

$$\lambda S.\lambda x : \triangleright A, y : \triangleright B.\widehat{\triangleright}(\mathsf{next}\,[x' \leftarrow x, y' \leftarrow y,\ \mathcal{R} \ \leftarrow S]\,.(x' \mathrel{\mathcal{R}} y'))$$

$$\eta(*)\ \mathcal{R}_1\ M \stackrel{\text{def}}{=\!=} M \Rightarrow^0 \langle\rangle$$
$$\theta_1(x)\ \mathcal{R}_1\ M \stackrel{\text{def}}{=\!=} \Sigma M', M'' \colon \mathtt{Term}_{\mathtt{FPC}}. M \to^0_* M' \to^1 M'' \text{ and } x \triangleright \mathcal{R}_1\ \mathsf{next}(M'')$$
$$x\ \mathcal{R}_{\tau_1 \times \tau_2}\ M \stackrel{\text{def}}{=\!=} \pi_1(x)\ \mathcal{R}_{\tau_1}\ \mathtt{fst}\ (M) \text{ and } \pi_2(x)\ \mathcal{R}_{\tau_2}\ \mathtt{snd}\ (M)$$
$$\eta(\mathsf{inl}(x))\ \mathcal{R}_{\tau_1+\tau_2}\ M \stackrel{\text{def}}{=\!=} \Sigma L. M \Rightarrow^0 \mathtt{inl}\ L \text{ and } x\ \mathcal{R}_{\tau_1}\ L$$
$$\eta(\mathsf{inr}(x))\ \mathcal{R}_{\tau_1+\tau_2}\ M \stackrel{\text{def}}{=\!=} \Sigma L. M \Rightarrow^0 \mathtt{inr}\ L \text{ and } x\ \mathcal{R}_{\tau_2}\ L$$
$$\theta_{\tau_1+\tau_2}(x)\ \mathcal{R}_{\tau_1+\tau_2}\ L \stackrel{\text{def}}{=\!=} \Sigma M', M'' \colon \mathtt{Term}_{\mathtt{FPC}}. M \to^0_* M' \to^1 M'' \text{ and } x \triangleright \mathcal{R}_{\tau_1+\tau_2}\ \mathsf{next}(M'')$$
$$f\ \mathcal{R}_{\tau\to\sigma}\ M \stackrel{\text{def}}{=\!=} \Pi x \colon [\![\tau]\!], N \colon \mathtt{Term}_{\mathtt{FPC}}. x\ \mathcal{R}_\tau\ N \to f(x)\ \mathcal{R}_\sigma\ (MN)$$
$$x\ \mathcal{R}_{\mu\alpha.\tau}\ M \stackrel{\text{def}}{=\!=} \Sigma M' M''. \mathtt{unfold}\ M \to^0_* M' \to^1 M'' \text{ and } x \triangleright \mathcal{R}_{\tau[\mu\alpha.\tau/\alpha]}\ \mathsf{next}(M'')$$

Fig. 11. The logical relation $\mathcal{R}_\tau : [\![\tau]\!] \times \mathtt{Term}_{\mathtt{FPC}} \to \mathcal{U}$.

Using this, one can formally define the logical relation as a fixed point of a function of type

$$\triangleright(\Pi(\tau : \mathtt{Type}_{\mathtt{FPC}}).\, [\![\tau]\!] \times \mathtt{Term}_{\mathtt{FPC}} \to \mathcal{U}) \to (\Pi(\tau : \mathtt{Type}_{\mathtt{FPC}}).\, [\![\tau]\!] \times \mathtt{Term}_{\mathtt{FPC}} \to \mathcal{U})$$

similarly to the formal definition of θ in the equation (14).

5.3. *Proof of computational adequacy*

Computational adequacy follows from the fundamental lemma below, stating that all terms respect the logical relation. The proof of the fundamental lemma rests on the following key lemmas.

Lemma 5.1. If $f \triangleright \mathcal{R}_{\tau\to\sigma}\ \mathsf{next}(M)$ and $r \triangleright \mathcal{R}_\tau\ \mathsf{next}(L)$ then

$$(f \circledast r) \triangleright \mathcal{R}_\sigma\ \mathsf{next}(ML)$$

Proof. By definition $f \triangleright \mathcal{R}_{\tau\to\sigma}\ \mathsf{next}(M)$ is type equal to

$$\triangleright\,[x \leftarrow f]\,.(x\ \mathcal{R}_{\tau\to\sigma}\ M)$$

which by definition is

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

By applying the latter to r and $\mathsf{next}\, L$ using the generalised later application of (26) we get

$$\triangleright\,[x \leftarrow f, y \leftarrow r]\,.(y\ \mathcal{R}_\tau\ L \to x(y)\ \mathcal{R}_\sigma\ ML)$$

By further applying this to the hypothesis we get

$$\triangleright\,[x \leftarrow f, y \leftarrow r]\,.(x(y)\ \mathcal{R}_\sigma\ ML)$$

which is equivalent to $(f \circledast r) \triangleright \mathcal{R}_\sigma\ \mathsf{next}(ML)$, thus concluding the case. □

Lemma 5.2. If $M \to^0 N$ then $x\ \mathcal{R}_\sigma\ M$ iff $x\ \mathcal{R}_\sigma\ N$.

Proof. We prove first the left to right implication by induction on σ, and show just a few cases.

In the case of coproducts, we proceed by case analysis on x. In the case of $x = \eta(\iota_1(y))$, by the assumption we have that $M \to_*^0 \mathtt{inl}\ (N')$ and $y\ \mathcal{R}_{\tau_1}\ N'$. If $M = \mathtt{inl}\ (N')$, then by N must be of the form $\mathtt{inl}\ (N'')$ for some N'', such that $N' \to^0 N''$. In this case, by induction hypothesis $y\ \mathcal{R}_{\tau_1}\ N''$ and so $x\ \mathcal{R}_{\tau_1+\tau_2}\ N$. If the reduction $M \to_*^0 \mathtt{inl}\ (N')$ has positive length, by determinancy of the operational semantics (Lemma 3.1) we get $N \to_*^0 \mathtt{inl}\ N'$, and thus $x\ \mathcal{R}_{\tau_1+\tau_2}\ N$. The case where $x = \eta(\iota_2(y))$ is similar. When $x = \theta_{\tau_1+\tau_2}(y)$, by the assumption $x\ \mathcal{R}_{\tau_1+\tau_2}\ M$ there exist M' and M'' such that $M \to_*^0 M'$ and $M' \to^1 M''$ and $y \triangleright\mathcal{R}_1\ \mathsf{next}(M'')$. Again by determinancy of the operational semantics, $N \to_*^0 M'$ and thus we conclude $x\ \mathcal{R}_{\tau_1+\tau_2}\ N$.

Now we consider the case for recursive types. By assumption we know there exists M' and M'' such that $\mathtt{unfold}\ M \to_*^0 M'$ and $M' \to^1 M''$ and $x \triangleright\mathcal{R}_{\tau[\mu\alpha.\tau/\alpha]}\ \mathsf{next}(M'')$. Since $M \to^0 N$ then also $\mathtt{unfold}\ M \to^0 \mathtt{unfold}\ N$. Therefore, from the assumption and the fact that the operational semantics is deterministic (Lemma 3.1) we get $\mathtt{unfold}\ N \to_*^0 M'$. By definition of the logical relation we get $x\ \mathcal{R}_{\mu\alpha.\tau}\ N$, which concludes the proof.

The proof of the right to left implication is also by induction on the structure of σ. Again we just show a few cases.

In the case of the unit type, we proceed by case analysis on x. When $x = \eta(*)$ we have that $N \to_*^0 \langle\rangle$. Since $M \to^0 N$ we get $M \to_*^0 \langle\rangle$ as required. When x is $\theta_1(x')$ by assumption $x\ \mathcal{R}_1\ N$ implies that there exists N' and N'' such that $N \to_*^0 N'$ and $N' \to^1 N''$ and $x' \triangleright\mathcal{R}_1\ \mathsf{next}(N'')$. Since also $M \to_*^0 N'$ this implies $x\ \mathcal{R}_1\ M$.

In the case of recursive types, by assumption we have that $x\ \mathcal{R}_{\mu\alpha.\tau}\ N$ and $M \to^0 N$. From the former we derive that there exists M' and M'' such that $\mathtt{unfold}\ N \to_*^0 M'$,$M' \to^1 M''$ and $x\ \mathcal{R}_{\tau[\mu\alpha.\tau/\alpha]}\ \mathsf{next}(M'')$. Since $M \to^0 N$ then also $\mathtt{unfold}\ M \to^0 \mathtt{unfold}\ N$. Therefore, we know that $\mathtt{unfold}\ M \to_*^0 M'$, thus by definition of the logical relation we conclude. □

Lemma 5.3. If $x \triangleright\mathcal{R}_\tau\ \mathsf{next}(M)$ and $M' \to^1 M$ then $\theta_\tau(x)\ \mathcal{R}_\tau\ M'$.

Proof. The proof is by guarded recursion, so we assume that the lemma is "later true", i.e., that we have an inhabitant of the type obtained by applying $\triangleright$ to the statement of the lemma. We proceed by induction on τ.

The cases for the unit type and for the coproduct are straightforward by definition. Now the case for the product. By assumption we have

$$y \triangleright\mathcal{R}_{\tau_1\times\tau_2}\ \mathsf{next}(M)$$

by unfolding definitions we get

$$\triangleright\,[x \leftarrow y]\,.(\pi_1(x)\ \mathcal{R}_{\tau_1}\ (\mathtt{fst}\ M)) \text{ and } (\pi_2(x)\ \mathcal{R}_{\tau_2}\ \mathtt{fst}\ (M))$$

which implies

$$(\pi_1(y)) \triangleright\mathcal{R}_{\tau_1}\ \mathsf{next}(\mathtt{fst}\ M) \qquad \text{and} \qquad \pi_2(y) \triangleright\mathcal{R}_{\tau_2}\ \mathsf{next}(\mathtt{snd}\ M)$$

Since $M' \to^1 M$ then also $\mathtt{fst}\ M' \to^1 \mathtt{fst}\ M$ and $\mathtt{snd}\ M' \to^1 \mathtt{snd}\ M$, thus we can use

the induction hypothesis on τ_1 and τ_2 and get

$$\theta_{\tau_1}(\pi_1(y))\ \mathcal{R}_{\tau_1}\ \mathtt{fst}\ M' \qquad \text{and} \qquad \theta_{\tau_2}(\pi_2(y))\ \mathcal{R}_{\tau_2}\ \mathtt{snd}\ M'$$

by definition $\theta_{\tau_1\times\tau_2}$ commutes with π_1 and π_2. Thus, we obtain

$$\pi_1(\theta_{\tau_1\times\tau_2}(y))\ \mathcal{R}_{\tau_1}\ \mathtt{fst}\ M' \qquad \text{and} \qquad \pi_2(\theta_{\tau_1\times\tau_2}(y))\ \mathcal{R}_{\tau_2}\ \mathtt{snd}\ M'$$

which is by definition what we wanted.

Now the case for the function space. Assume $f \triangleright\mathcal{R}_{\tau_1\to\tau_2}\ \mathsf{next}(M)$ and $M' \to^1 M$. We must show that if $y : [\![\tau_1]\!]$, $N : \mathtt{Term}_{\mathtt{FPC}}$ and $y\ \mathcal{R}_{\tau_1}\ N$ then $(\theta_{\tau_1\to\tau_2}(f))(y)\ \mathcal{R}_{\tau_2}\ (MN)$. So suppose $y\ \mathcal{R}_{\tau_1}\ N$, and thus also $\triangleright(y\ \mathcal{R}_{\tau_1}\ N)$ which is equal to $\mathsf{next}(y) \triangleright\mathcal{R}_{\tau_1}\ \mathsf{next}(N)$. By applying Lemma 5.1 to this and $f \triangleright\mathcal{R}_{\tau_1\to\tau_2}\ \mathsf{next}(M)$ we get

$$f \circledast (\mathsf{next}(y)) \triangleright\mathcal{R}_{\tau_2}\ \mathsf{next}(MN)$$

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

The interesting case is the one of $\mu\alpha.\tau$. Assume $x \triangleright\mathcal{R}_{\mu\alpha.\tau}\ \mathsf{next}(M)$ and $M' \to^1 M$. By definition of $\triangleright\mathcal{R}$ this implies $\triangleright\,[y \leftarrow x]\,.(y\ \mathcal{R}_{\mu\alpha.\tau}\ M)$ which by definition of $\mathcal{R}_{\mu\alpha.\tau}$ is

$$\triangleright\,[y \leftarrow x]\,.\Sigma N'N''.\mathtt{unfold}\ M \to_*^0 N' \text{ and } N' \to^1 N'' \text{ and } (y \triangleright\mathcal{R}_{\tau[\mu\alpha.\tau/\alpha]}\ \mathsf{next}(N''))$$

Since zero-step reductions cannot eliminate outer `unfold`'s, N' must be on the form $\mathtt{unfold}\ N$ for some N, such that $M \to_*^0 N$. Thus, we can apply the guarded induction hypothesis to get

$$\triangleright\,[y \leftarrow x]\,.(\Sigma N.M \to_*^0 N \text{ and } (\theta_{\tau[\mu\alpha.\tau/\alpha]}(y)\ \mathcal{R}_{\tau[\mu\alpha.\tau/\alpha]}\ \mathtt{unfold}\ N))$$

Since $\mathtt{unfold}\ M \to_*^0 \mathtt{unfold}\ N$, by Lemma 5.2 we get

$$\triangleright\,[y \leftarrow x]\,.(\theta_{\tau[\mu\alpha.\tau/\alpha]}(y)\ \mathcal{R}_{\tau[\mu\alpha.\tau/\alpha]}\ \mathtt{unfold}\ M)$$

which by (29) is

$$\mathsf{next}\,[y \leftarrow x]\,.(\theta_{\tau[\mu\alpha.\tau/\alpha]}(y)) \triangleright\mathcal{R}_{\tau[\mu\alpha.\tau/\alpha]}\ \mathsf{next}(\mathtt{unfold}\ M)$$

By (24) this implies

$$\mathsf{next}(\theta_{\tau[\mu\alpha.\tau/\alpha]}) \circledast x \triangleright\mathcal{R}_{\tau[\mu\alpha.\tau/\alpha]}\ \mathsf{next}(\mathtt{unfold}\ M)$$

Since by assumption $M' \to^1 M$ also $\mathtt{unfold}\ M' \to^1 \mathtt{unfold}\ M$ thus, by definition of the logical relation

$$\mathsf{next}(\theta_{\tau[\mu\alpha.\tau/\alpha]}) \circledast x\ \mathcal{R}_{\mu\alpha.\tau}\ M'$$

By definition $\mathsf{next}(\theta_{\tau[\mu\alpha.\tau/\alpha]}) \circledast x$ is equal to $\theta_{\mu\alpha.\tau}(x)$ thus we can derive

$$\theta_{\mu\alpha.\tau}(x)\ \mathcal{R}_{\mu\alpha.\tau}\ M'$$

as we wanted. □

Lemma 5.4 (Fundamental Lemma). Suppose $\Gamma \vdash M : \tau$, for $\Gamma \equiv x_1 : \tau_1, \cdots, x_n : \tau_n$ and $N_i : \tau_i$, $\gamma_i : [\![\tau_i]\!]$ and $\gamma_i\ \mathcal{R}_{[\![\tau_i]\!]}\ N_i$ for $i \in \{1, \ldots, n\}$, then $[\![M]\!]\,(\vec{\gamma})\ \mathcal{R}_\tau\ M[\vec{N}/\vec{x}]$

Proof. The proof is by guarded recursion, and so we assume $\triangleright$ applied to the statement of the lemma. This implies that for all well-typed terms M with context Γ and type τ the following holds:

$$\triangleright(\llbracket M \rrbracket\,(\vec{\gamma})\ \mathcal{R}_\tau\ M[\vec{M}/\vec{x}])$$

Then we proceed by induction on the typing derivation $\Gamma \vdash M : \tau$, showing only the interesting cases.

Consider first the case of $\Gamma \vdash \lambda x.M : \sigma \to \tau$. Assuming $\gamma_{n+1}\ \mathcal{R}_\sigma\ M_{n+1}$, we must show $\llbracket \lambda x.M \rrbracket\,(\vec{\gamma})(\gamma_{n+1})\ \mathcal{R}_\tau\ \llbracket M \rrbracket\,(\vec{\gamma}, \gamma_{n+1})$. Since

$$\llbracket \lambda x.M \rrbracket\,(\vec{\gamma})(\gamma_{n+1}) = \llbracket M \rrbracket\,(\vec{\gamma}, \gamma_{n+1})$$
$$(\lambda x.M)[\vec{M}/\vec{x}](M_{n+1}) = \lambda x.(M[\vec{M}/\vec{x}])(M_{n+1})$$

and $\lambda x.(M[\vec{M}/\vec{x}])(M_{n+1}) \to^0 (M[\vec{M}/\vec{x}])[M_{n+1}/x]$, by Lemma 5.2 it suffices to prove

$$\llbracket M \rrbracket\,(\vec{\gamma}, \gamma_{n+1})\ \mathcal{R}_\tau\ M[\vec{M}/\vec{x}][M_{n+1}/x]$$

which follows from the induction hypothesis.

For the case $\Gamma \vdash \texttt{unfold}\ M : \tau[\mu\alpha.\tau/\alpha]$ we want to show that

$$\llbracket \texttt{unfold}\ M \rrbracket\,(\vec{\gamma})\ \mathcal{R}_{\tau[\mu\alpha.\tau/\alpha]}\ (\texttt{unfold}\ M)[\vec{N}/\vec{x}]$$

By induction hypothesis we know that $\llbracket M \rrbracket\,(\vec{\gamma})\ \mathcal{R}_{\mu\alpha.\tau}\ (M[\vec{N}/\vec{x}])$ which means that there exists M' and M'' such that $\texttt{unfold}\ (M[\vec{N}/\vec{x}]) \to_*^0 M'$ and $M' \to^1 M''$ and $\llbracket M \rrbracket\,(\vec{\gamma})\ \triangleright\mathcal{R}_{\tau[\mu\alpha.\tau/\alpha]}\ \mathsf{next}(M'')$. By Lemma 5.3 then $\theta_{\tau[\mu\alpha.\tau/\alpha]}(\llbracket M \rrbracket\,(\vec{\gamma}))\ \mathcal{R}_{\tau[\mu\alpha.\tau/\alpha]}\ M'$ and since $\texttt{unfold}\ (M[\vec{N}/\vec{x}]) \to_*^0 M'$ by repeated application of Lemma 5.2 we get

$$\theta_{\tau[\mu\alpha.\tau/\alpha]}(\llbracket M \rrbracket\,(\vec{\gamma}))\ \mathcal{R}_{\tau[\mu\alpha.\tau/\alpha]}\ \texttt{unfold}\ (M[\vec{N}/\vec{x}])$$

Since by definition $\llbracket \texttt{unfold}\ M \rrbracket\,(\vec{\gamma}) = \theta_{\tau[\mu\alpha.\tau/\alpha]}(\llbracket M \rrbracket\,(\vec{\gamma}))$ this finishes the proof of the case.

For the case $\Gamma \vdash \texttt{fold}\ M : \mu\alpha.\tau$ we want to show that

$$\llbracket \texttt{fold}\ M \rrbracket\,(\vec{\gamma})\ \mathcal{R}_{\mu\alpha.\tau}\ (\texttt{fold}\ M)[\vec{N}/\vec{x}]$$

By definition of the logical relation we have to show that there exist M' and M'' such that

$$\texttt{unfold}\ (\texttt{fold}\ (M[\vec{N}/\vec{x}])) \to_*^0 M'$$

$M' \to^1 M''$ and that $\llbracket \texttt{fold}\ M \rrbracket\,(\vec{\gamma})\ \triangleright\mathcal{R}_{\tau[\mu\alpha.\tau/\alpha]}\ \mathsf{next}(M'')$. Setting M'' to be $(M[\vec{N}/\vec{x}])$, we are left to show that

$$\llbracket \texttt{fold}\ M \rrbracket\,(\vec{\gamma})\ \triangleright\mathcal{R}_{\tau[\mu\alpha.\tau/\alpha]}\ \mathsf{next}(M[\vec{N}/\vec{x}])$$

which is equal by definition of the interpretation function to

$$\mathsf{next}(\llbracket M \rrbracket\,(\vec{\gamma}))\ \triangleright\mathcal{R}_{\tau[\mu\alpha.\tau/\alpha]}\ \mathsf{next}((M[\vec{N}/\vec{x}]))$$

The latter is equal by (29) to $\triangleright(\llbracket M \rrbracket\,(\vec{\gamma})\ \mathcal{R}_{\tau[\mu\alpha.\tau/\alpha]}\ (M[\vec{N}/\vec{x}]))$ which is true by the guarded recursive hypothesis.

For the case $\Gamma \vdash \mathtt{inl}\ M : \tau_1 + \tau_2$ we have to prove that

$$[\![\mathtt{inl}\ M]\!]\,(\vec{\gamma})\ \mathcal{R}_{\tau_1+\tau_2}\ \mathtt{inl}\ M[\vec{M}/\vec{x}]$$

By definition of the interpretation function $[\![\mathtt{inl}\ M]\!]\,(\vec{\gamma})$ is equal to $\eta(\iota_1([\![M]\!]\,(\vec{\gamma})))$. By definition of the logical relation we have to prove that there exists M' such that

$$(\mathtt{inl}\ M)[\vec{M}/\vec{x}] \Rightarrow^0 \mathtt{inl}\ M' \text{ and } [\![M]\!]\,(\vec{\gamma})\ \mathcal{R}_{\tau_1}\ M'.$$

The former is trivially true with $M' = M[\vec{M}/\vec{x}]$ and the latter is by induction hypothesis. The case for $\Gamma \vdash \mathtt{inr}\ N : \tau_1 + \tau_2$ is similar.

For the case $\Gamma \vdash \mathtt{case}\ L\ \mathtt{of}\ x_1.M; x_2.N : \sigma$ we have to prove that

$$[\![\mathtt{case}\ L\ \mathtt{of}\ x_1.M; x_2.N]\!]\,(\vec{\gamma})\ \mathcal{R}_\sigma\ (\mathtt{case}\ L\ \mathtt{of}\ x_1.M; x_2.N)[\vec{M}/\vec{x}]$$

For this it suffices to prove

$$[\![\lambda x.\mathtt{case}\ x\ \mathtt{of}\ x_1.M; x_2.N]\!]\,(\vec{\gamma})\ \mathcal{R}_{\tau_1+\tau_2\to\sigma}\ (\lambda x.\mathtt{case}\ x\ \mathtt{of}\ x_1.M; x_2.N)[\vec{M}/\vec{x}] \qquad (30)$$

and then applying this to $[\![L]\!]\,(\vec{\gamma})\ \mathcal{R}_{\tau_1+\tau_2}\ L[\vec{M}/\vec{x}]$. We prove (30) by guarded recursion thus assuming the statement is true later.

Assume y of type $[\![\tau_1 + \tau_2]\!]$, L a term, and $y\ \mathcal{R}_{\tau_1+\tau_2}\ L$. We proceed by case analysis on y which is of type $[\![\tau_1 + \tau_2]\!]$ which by definition is $L([\![\tau_1]\!] + [\![\tau_2]\!])$. In the case $y = \eta(\iota_1(z))$, where z is of type $[\![\tau_1]\!]$ we know by assumption that there exists L' s.t. $L \Rightarrow^0 \mathtt{inl}\ (L')$ and $z\ \mathcal{R}_{\tau_1}\ L'$. Since

$$[\![\lambda x.\mathtt{case}\ x\ \mathtt{of}\ x_1.M; x_2.N]\!]\,(\vec{\gamma})(\eta(\iota_1(z))) = [\![M]\!]\,(\vec{\gamma}, z)$$

and

$$\mathtt{case}\ L\ \mathtt{of}\ x_1.M[\vec{M}/\vec{x}]; x_2.N[\vec{M}/\vec{x}] \Rightarrow^0 M[\vec{M}/\vec{x}][L'/x_1]$$

by Lemma 5.2 we are left to prove

$$[\![M]\!]\,(\vec{\gamma}, \gamma)\ \mathcal{R}_\sigma\ \ M[\vec{M}/\vec{x}][L'/x_1]$$

which is true by induction hypothesis. The case $y = \eta(\iota_2(z))$ where z is of type $[\![\tau_2]\!]$ is similar.

Now consider the case of $y = \theta_{\tau_1+\tau_2}(z)$, where z is of type $\triangleright[\![\tau_1 + \tau_2]\!]$. By induction hypothesis we know that $\theta_{\tau_1+\tau_2}(z)\ \mathcal{R}_{\tau_1+\tau_2}\ L$, thus there exist L' and L'' of type $\mathtt{Term}_{\mathrm{FPC}}$ such that $L \to_*^0 L'$, $L' \to^1 L''$ and $z \triangleright\mathcal{R}_{\tau_1+\tau_2}\ \mathsf{next}(L'')$.

Recall that we have assumed $\triangleright$ of (30), i.e.,

$$\triangleright([\![\lambda x.\mathtt{case}\ x\ \mathtt{of}\ x_1.M; x_2.N]\!]\,(\vec{\gamma})\ \mathcal{R}_{\tau_1+\tau_2\to\sigma}\ (\lambda x.\mathtt{case}\ x\ \mathtt{of}\ x_1.M; x_2.N)[\vec{M}/\vec{x}])$$

which is type equal to

$$\mathsf{next}([\![\lambda x.\mathtt{case}\ x\ \mathtt{of}\ x_1.M; x_2.N]\!]\,(\vec{\gamma}))\ \triangleright\mathcal{R}_{\tau_1+\tau_2\to\sigma}\ \mathsf{next}((\lambda x.\mathtt{case}\ x\ \mathtt{of}\ x_1.M; x_2.N)[\vec{M}/\vec{x}])$$

By Lemma 5.1 we can apply this to the assumption $z \triangleright\mathcal{R}_{\tau_1+\tau_2}\ \mathsf{next}(L'')$ thus getting

$$\mathsf{next}([\![\lambda x.\mathtt{case}\ x\ \mathtt{of}\ x_1.M; x_2.N]\!]\,(\vec{\gamma}))\circledast z \triangleright\mathcal{R}_\sigma\ \mathsf{next}(((\lambda x.\mathtt{case}\ x\ \mathtt{of}\ x_1.M; x_2.N)[\vec{M}/\vec{x}])(L''))$$

Since $L' \to^1 L''$ we can apply Lemma 5.3 and obtain

$$\theta_\sigma(\mathsf{next}([\![\lambda x.\mathtt{case}\ x\ \mathtt{of}\ x_1.M; x_2.N]\!]\,(\vec{\gamma})) \circledast z)\ \mathcal{R}_\sigma\ \mathtt{case}\ L'\ \mathtt{of}\ x_1.M[\vec{M}/\vec{x}]; x_2.N[\vec{M}/\vec{x}]$$

By Lemma 5.2 with the fact that $L \to_*^0 L'$ we get

$$\theta_\sigma(\mathsf{next}(\llbracket \lambda x.\mathtt{case}\ x\ \mathtt{of}\ x_1.M; x_2.N \rrbracket\,(\vec{\gamma})) \circledast z)\ \mathcal{R}_\sigma\ \mathtt{case}\ L\ \mathtt{of}\ x_1.M[\vec{M}/\vec{x}]; x_2.N[\vec{M}/\vec{x}]$$

And finally by simplifying the left-hand side using Lemma 4.6:

$$\theta_\sigma(\mathsf{next}(\llbracket \lambda x.\mathtt{case}\ x\ \mathtt{of}\ x_1.M; x_2.N \rrbracket\,(\vec{\gamma})) \circledast z) = \llbracket \lambda x.\mathtt{case}\ x\ \mathtt{of}\ x_1.M; x_2.N \rrbracket\,(\vec{\gamma})(y)$$

thus getting

$$\llbracket \lambda x.\mathtt{case}\ x\ \mathtt{of}\ x_1.M; x_2.N \rrbracket\,(\vec{\gamma})(y)\ \mathcal{R}_\sigma\ (\lambda x.\mathtt{case}\ x\ \mathtt{of}\ x_1.M; x_2.N)[\vec{M}/\vec{x}](L)$$

as we wanted. □

From the Fundamental lemma we can now prove computational adequacy.

Theorem 5.5 (Intensional Computational Adequacy). If $M\colon 1$ is a closed term then $M \Rightarrow^k \langle\rangle$ iff $\llbracket M \rrbracket\,(*) = \delta^k(\eta(*))$.

Proof. The left to right implication is soundness (Proposition 4.8). For the right to left implication note first that the Fundamental Lemma (Lemma 5.4) implies $\delta^k(\eta(*))\ \mathcal{R}_1\ M$. To complete the proof it suffices to show that $\delta_1^k(\eta(*))\ \mathcal{R}_1\ M$ implies $M \Rightarrow^k \langle\rangle$.

This is proved by guarded recursion and induction on k: the case of $k = 0$ is immediate by definition of $\mathcal{R}_1$. If $k = k' + 1$ first assume $\delta_1^k(\eta(*))\ \mathcal{R}_1\ M$. By definition of $\mathcal{R}$ there exist M' and M'' such that $M \to_*^0 M'$, $M' \to^1 M''$ and $\mathsf{next}(\delta_1^{k'}(\eta(*)))\ \triangleright\mathcal{R}_1\ \mathsf{next}(M'')$ which is type equal to $\triangleright(\delta_1^{k'}(\eta(*))\ \mathcal{R}_1\ M'')$. By the guarded recursion assumption we get $\triangleright(M'' \Rightarrow^{k'} \langle\rangle)$ which by definition implies $M \Rightarrow^k \langle\rangle$. □

From Theorem 5.5 one can deduce that whenever two terms have equal denotations they are contextually equivalent in a very intensional way, as we now describe. By a context, we mean a term $C[-]$ with a hole, and we say that $C[-]$ has type $\Gamma, \tau \to (-, 1)$ if $C[M]$ is a closed term of type 1, whenever $\Gamma \vdash M : \tau$.

Corollary 5.6. Suppose $\Gamma \vdash M : \tau$ and $\llbracket M \rrbracket = \llbracket N \rrbracket$. If $C[-]$ has type $\Gamma, \tau \to (-, 1)$ and $C[M] \Rightarrow^k \langle\rangle$ also $C[N] \Rightarrow^k \langle\rangle$.

6. Extensional Computational Adequacy

Our model of FPC is intensional in the sense that it distinguishes between computations computing the same value in a different number of steps. In this section we define a logical relation which relates elements of the model if they differ only by a finite number of computation steps. In particular, this also means relating $\bot$ to $\bot$.

To do this we need to consider global behaviour of computations, as opposed to the local (or finitely computable) behaviour captured by the guarded recursive lifting monad L. To understand what this means, consider the interpretation of $L1$ in the topos of trees as described in Section 2.1. For each number n, the set

$$L1(n) = \{\bot, 0, 1, \ldots, n-1\}$$

describes computations terminating in at most $n - 1$ steps or using at least n steps

(corresponding to $\bot$). It cannot distinguish between termination in more than $n-1$ steps and real divergence. Our relation should relate a terminating value x in $L1(n)$ to any other terminating value, but not real divergence, which is impossible, if divergence cannot be distinguished from slow termination. Another, more semantic, way to phrase the problem is that termination as described by the subsets $\{0,1,\dots,n-1\}$ of $L1(n)$ for each n does not form a subobject of $L1$.

The global behaviour of a type is captured by the type constructor $\forall\kappa.(-)$ introduced in Section 3.4.1. This construction takes the guarded recursive type $\forall\kappa.L1$ to

$$L^{\mathrm{gl}}A \stackrel{\mathrm{def}}{=\!=} \forall\kappa.LA$$

which is a coinductive solution to the type equation

$$L^{\mathrm{gl}}A \cong A + L^{\mathrm{gl}}A$$

when κ is not free in A [Møg14]. Semantically $L1$ is modelled as the set $\mathbb{N}+\{\bot\}$, and termination is the subset of this corresponding to the left inclusion of $\mathbb{N}$. So on the global level we can, at least semantically, distinguish between termination and non-termination. This is reflected syntactically in Lemma 6.20.

We now define the global interpretation of types and terms and the logical relation.

6.1. *Global interpretation of types and terms*

Recall that the developments above should be read as taking place in a context of an implicit clock κ. To be consistent with the notation of the previous sections, κ will remain implicit in the denotations of types and terms, although one might choose to write e.g. $[\![\sigma]\!]^{\kappa}$ to make the clock explicit.

We define global interpretations of types and terms as follows:

$$[\![\sigma]\!]^{\mathrm{gl}} \stackrel{\mathrm{def}}{=\!=} \forall\kappa.\,[\![\sigma]\!]$$
$$[\![M]\!]^{\mathrm{gl}} \stackrel{\mathrm{def}}{=\!=} \Lambda\kappa.\,[\![M]\!]$$

such that if $\Gamma \vdash M : \tau$, then

$$[\![M]\!]^{\mathrm{gl}} \colon \forall\kappa.([\![\Gamma]\!] \to [\![\tau]\!])$$

Note that $[\![\sigma]\!]^{\mathrm{gl}}$ is a wellformed type, because $[\![\sigma]\!]$ is a wellformed type in context $\sigma\colon \mathtt{Type}_{\mathrm{FPC}}$ and $\mathtt{Type}_{\mathrm{FPC}}$ is an inductive type formed without reference to clocks or guarded recursion, thus κ does not appear in $\mathtt{Type}_{\mathrm{FPC}}$. By a similar argument $[\![M]\!]^{\mathrm{gl}}$ is welltyped.

Define for all σ the *delay* operator $\delta^{\mathrm{gl}}_{\sigma}\colon [\![\sigma]\!]^{\mathrm{gl}} \to [\![\sigma]\!]^{\mathrm{gl}}$ as follows

$$\delta^{\mathrm{gl}}_{\sigma}(x) \stackrel{\mathrm{def}}{=\!=} \Lambda\kappa.\delta_{\sigma}(x[\kappa]) \tag{31}$$

Similarly for LA, $\delta^{\mathrm{gl}}_{LA}(x) \stackrel{\mathrm{def}}{=\!=} \Lambda\kappa.\delta_{LA}(x[\kappa])$.

With these definitions we can lift the adequacy theorem to the global points.

Corollary 6.1 (Computational adequacy). If $M\colon 1$ is a closed term and n is a natural number, then $M \Downarrow^n \langle\rangle$ iff $\forall\kappa.\,[\![M]\!]\,(*) = \delta^n(\eta(*))$.

Proof. Since $\forall\kappa.(-)$ is functorial, Theorem 5.5 gives $\forall\kappa.\, [\![M]\!]\,(*) = \delta^n(\eta(*))$ iff $\forall\kappa. M \Rightarrow^n \langle\rangle$, which by Lemma 3.3 holds iff $M \Downarrow^n \langle\rangle$. □

6.2. *A weak bisimulation relation for the lifting monad*

Before defining the logical relation on the interpretation of types, we define a relational version of the guarded recursive lifting monad L. If applied to the identity relation on a type A in which κ does not appear, we obtain a weak bisimulation relation similar to the one defined by Danielsson [Dan12] for the coinductive partiality monad.

Definition 6.2. For a relation $R : A \times B \to \mathcal{U}$ define the lifting $LR : LA \times LB \to \mathcal{U}$ by guarded recursion and case analysis on the elements of LA and LB:

$$\begin{aligned}
\eta(x)\ LR\ \eta(y) &\stackrel{\text{def}}{=\!=} x\ R\ y \\
\eta(x)\ LR\ \theta_{LB}(y) &\stackrel{\text{def}}{=\!=} \Sigma n, y'.\theta_{LB}(y) = \delta^n_{LB}(\eta(y')) \text{ and } x\ R\ y' \\
\theta_{LA}(x)\ LR\ \eta(y) &\stackrel{\text{def}}{=\!=} \Sigma n, x'.\theta_{LA}(x) = \delta^n_{LA}(\eta(x')) \text{ and } x'\ R\ y \\
\theta_{LA}(x)\ LR\ \theta_{LB}(y) &\stackrel{\text{def}}{=\!=} x \triangleright LR\ y
\end{aligned}$$

The lifting of R, intuitively, captures computations that differ for a finite amount of computational steps or both diverge. For example, $\bot$ as defined in Section 4 is always related to itself which can be shown by guarded recursion as follows. Suppose $\triangleright(\bot\ LR\ \bot)$. Since $\bot = \theta(\mathsf{next}(\bot))$, to prove $\bot\ LR\ \bot$, we must prove $\mathsf{next}(\bot) \triangleright LR\ \mathsf{next}(\bot)$. But, this type is equal to the assumption $\triangleright(\bot\ LR\ \bot)$ by (29).

We can also prove that LR is closed under application of δ to either side.

Lemma 6.3. If $R : A \times B \to \mathcal{U}$, and $x\ LR\ y$ then $x\ LR\ \delta_{LB}(y)$ and $\delta_{LA}(x)\ LR\ y$.

Proof. Assume $x\ LR\ y$. We show $x\ LR\ \delta_{LB}(y)$. The proof is by guarded recursion, hence we first assume:

$$\triangleright(\Pi x : LA, y : LB.x\ LR\ y \Rightarrow x\ LR\ \delta_{LB}(y)). \tag{32}$$

We proceed by case analysis on x and y. If $x = \eta(x')$, then, since $x\ LR\ y$, there exist n and y' such that $y = \delta^n_{LB}(\eta(y'))$ and $x'\ R\ y'$. So then $\delta_{LB}(y) = \delta^{n+1}_{LB}(\eta(y'))$, from which it follows that $x\ LR\ \delta_{LB}(y)$.

For the case where $x = \theta_{LA}(x')$ and $y = \eta(v)$, it suffices to show that $\delta^n_{LA}(\eta(w))\ LR\ \eta(v)$ implies $\delta^n_{LA}(\eta(w))\ LR\ \delta_{LB}(\eta(v))$. The case of $n = 0$ was proved above. For $n = m + 1$ we know that if $\delta^n_{LA}(\eta(w))\ LR\ \eta(v)$ also $\delta^m_{LA}(\eta(w))\ LR\ \eta(v)$ holds by definition, and this implies

$$\triangleright(\delta^m_{LA}(\eta(w))\ LR\ \eta(v))$$

But this type can be rewritten as follows

$$\begin{aligned}
\triangleright(\delta^m_{LA}(\eta(w))\ LR\ \eta(v)) &\equiv \mathsf{next}(\delta^m_{LA}(\eta(w)) \triangleright LR\ \mathsf{next}(\eta(v))) \\
&\equiv \theta_{LA}(\mathsf{next}(\delta^m_{LA}(\eta(w))))\ LR\ \theta_{LB}(\mathsf{next}(\eta(v)))) \\
&\equiv \delta^n_{LA}(\eta(w))\ LR\ \delta_{LB}(\eta(v))
\end{aligned}$$

proving the case.

Finally, the case when $x = \theta_{LA}(x')$ and $y = \theta_{LB}(y')$. The assumption in this case is $x' \triangleright LR\ y'$, which means by (28),

$$\triangleright\,[x'' \leftarrow x', y'' \leftarrow y']\,.x''\ LR\ y''$$

By the guarded recursion hypothesis (32) we get

$$\triangleright\,[x'' \leftarrow x', y'' \leftarrow y']\,.x''\ LR\ \delta_{LB}(y'')$$

which can be rewritten to

$$\triangleright\,[x'' \leftarrow x', y'' \leftarrow y']\,.x''\ LR\ \theta_{LB}(\mathsf{next}(y'')) \tag{33}$$

By (27) there is an inhabitant of the type

$$\triangleright\,[x'' \leftarrow x', y'' \leftarrow y']\,.(\mathsf{next}(y'') = y')$$

and thus (33) implies $\triangleright\,[x'' \leftarrow x']\,.x''\ LR\ \theta_{LB}(y')$, which, by (29) and since $y = \theta_{LB}(y')$ equals $x' \triangleright LR\ \mathsf{next}(y)$. By definition, this is

$$\theta_{LA}(x')\ LR\ \theta_{LB}(\mathsf{next}(y))$$

which since $x = \theta_{LA}(x')$ is $x\ LR\ \delta_{LB}(y)$. □

We can lift this result to L^{gl} as follows. Suppose $R : A \times B \to \mathcal{U}$ and κ not in A or B. Define $L^{\mathrm{gl}}R : L^{\mathrm{gl}}A \times L^{\mathrm{gl}}B \to \mathcal{U}$ as

$$x\ L^{\mathrm{gl}}R\ y \stackrel{\mathrm{def}}{=\!=} \forall\kappa.x[\kappa]\ LR\ y[\kappa]$$

Lemma 6.4. Let $x : L^{\mathrm{gl}}A$ and $y : L^{\mathrm{gl}}B$. If $x\ L^{\mathrm{gl}}R\ y$ then $x\ L^{\mathrm{gl}}R\ \delta^{\mathrm{gl}}(y)$ and $\delta^{\mathrm{gl}}(x)\ L^{\mathrm{gl}}R\ y$.

Proof. Follows directly from Lemma 6.3. □

One might expect that $\delta_{LA}(x)\ LR\ \delta_{LB}(y)$ implies $x\ LR\ y$. This is not true, it only implies $\triangleright(x\ LR\ y)$. In the case of L^{gl}, however, we can use force to remove the $\triangleright$.

Lemma 6.5. For all $x : L^{\mathrm{gl}}A$ and $y : L^{\mathrm{gl}}B$ and for all $R : A \times B \to \mathcal{U}$, if $\delta^{\mathrm{gl}}_{LA}(x)\ L^{\mathrm{gl}}R\ \delta^{\mathrm{gl}}_{LB}(y)$ then $x\ L^{\mathrm{gl}}R\ y$.

Proof. Assume $\delta^{\mathrm{gl}}_{LA}(x)\ L^{\mathrm{gl}}R\ \delta^{\mathrm{gl}}_{LB}(y)$. We can rewrite this type by unfolding definitions and (29) as follows.

$$\begin{aligned}
\delta^{\mathrm{gl}}_{LA}(x)\ L^{\mathrm{gl}}R\ \delta^{\mathrm{gl}}_{LB}(y) &\equiv \forall\kappa.(\delta^{\mathrm{gl}}_{LA}(x))[\kappa]\ LR\ (\delta^{\mathrm{gl}}_{LB}(y))[\kappa] \\
&\equiv \forall\kappa.(\delta_{LA}(x[\kappa]))\ LR\ (\delta_{LB}(y[\kappa])) \\
&\equiv \forall\kappa.(\mathsf{next}(x[\kappa]) \triangleright LR\ \mathsf{next}(y[\kappa])) \\
&\equiv \forall\kappa.\triangleright(x[\kappa]\ LR\ (y[\kappa]))
\end{aligned}$$

Using force this implies $\forall\kappa.(x[\kappa]\ LR\ (y[\kappa]))$ which is equal to $x\ L^{\mathrm{gl}}R\ y$. □

Lemma 6.6. For all x of type $L^{\mathrm{gl}}A$ and y of type $L^{\mathrm{gl}}B$, if $\delta^{\mathrm{gl}}_{LA}(x)\ L^{\mathrm{gl}}R\ y$ then $x\ L^{\mathrm{gl}}R\ y$.

$$
\begin{aligned}
x \;\approx_1\; y &\stackrel{\text{def}}{=\!=} x\; L(=_1)\; y \\
x \;\approx_{\tau_1+\tau_2}\; y &\stackrel{\text{def}}{=\!=} x\; L(\;\approx_{\tau_1}\; +\; \approx_{\tau_2}\;)\; y \\
x \;\approx_{\tau_1\times\tau_2}\; y &\stackrel{\text{def}}{=\!=} \pi_1(x)\;\approx_{\tau_1}\;\pi_1(y) \text{ and } \pi_2(x)\;\approx_{\tau_2}\;\pi_2(y) \\
f \;\approx_{\sigma\to\tau}\; g &\stackrel{\text{def}}{=\!=} \Pi(x,y:[\![\sigma]\!]).x\;\approx_\sigma\; y \to f(x)\;\approx_\tau\; g(y) \\
x \;\approx_{\mu\alpha.\tau}\; y &\stackrel{\text{def}}{=\!=} x \triangleright \approx_{\tau[\mu\alpha.\tau/\alpha]}\; y
\end{aligned}
$$

Fig. 12. The logical relation $\approx_\tau$ is a predicate over denotations of τ of type $[\![\tau]\!] \times [\![\tau]\!] \to \mathcal{U}$

Proof. Assume $\delta^{\text{gl}}_{LA}(x)\; L^{\text{gl}}R\; y$. Then by applying Lemma 6.4 we get $\delta^{\text{gl}}_{LA}(x)\; L^{\text{gl}}R\; \delta^{\text{gl}}_{LB}(y)$ and by applying Lemma 6.5 we get $x\; L^{\text{gl}}R\; y$. □

6.3. *Relating terms up to extensional equivalence*

Figure 12 defines for each FPC type τ the logical relation $\approx_\tau : [\![\tau]\!] \times [\![\tau]\!] \to \mathcal{U}$. The definition is by guarded recursion, and well-definedness can be formalised using an argument similar to that used for well-definedness of θ in equation (14). The case of recursive types is well typed by Lemma 4.2. The figure uses the following lifting of relations to sum types.

Definition 6.7. Let $R : A \times B \to \mathcal{U}$ and $R' : A' \times B' \to \mathcal{U}$. Define $(R + R') : (A + A') \times (B + B') \to \mathcal{U}$ by case analysis as follows (omitting false cases)

$$
\begin{aligned}
\iota_1(x)\; (R + R')\; \iota_1(y) &\stackrel{\text{def}}{=\!=} x\; R\; y \\
\iota_2(x)\; (R + R')\; \iota_2(y) &\stackrel{\text{def}}{=\!=} x\; R'\; y
\end{aligned}
$$

The logical relation can be generalised to open terms and the global interpretation of terms as in the next two definitions.

Definition 6.8. For $\Gamma \equiv x_1 : \sigma_1, \cdots, x_n : \sigma_n$ and for f, g of type $[\![\Gamma]\!] \to [\![\tau]\!]$ define

$$
f \;\approx_{\Gamma,\tau}\; g \stackrel{\text{def}}{=\!=} \Pi(\vec{x},\vec{y} : [\![\vec{\sigma}]\!]).\vec{x}\;\approx_{\vec{\sigma}}\;\vec{y} \to f(\vec{x})\;\approx_\tau\; g(\vec{y})
$$

Definition 6.9. For x, y of type $[\![\Gamma \to \tau]\!]^{\text{gl}}$,

$$
x \;\approx^{\text{gl}}_{\Gamma,\tau}\; y \stackrel{\text{def}}{=\!=} \forall\kappa.x[\kappa]\;\approx_{\Gamma,\tau}\; y[\kappa]
$$

6.4. *Properties of* $\approx_{\Gamma,\tau}$

We now establish some basic properties of the logical relation needed for later developments. The first lemma states that delayed application $\circledast$ respects the logical relation.

Lemma 6.10. For all f, g of type $\triangleright[\![\tau \to \sigma]\!]$ and x, y of type $\triangleright[\![\tau]\!]$, if $f \triangleright \approx_{\tau\to\sigma} g$ and $x \triangleright \approx_\tau y$ then $(f \circledast x) \triangleright \approx_\sigma (g \circledast y)$.

Proof. Assume $f \triangleright \approx_{\tau\to\sigma} g$ and $x \triangleright \approx_\tau y$. By Definition 28 $f \triangleright \approx_{\tau\to\sigma} g$ is $\triangleright[f' \leftarrow f, g' \leftarrow g].(f' \approx_{\tau\to\sigma} g')$ which by unfolding the definition of $\approx_{\tau\to\sigma}$ is

$$\triangleright[f' \leftarrow f, g' \leftarrow g].(\Pi(x, y : [\![\sigma]\!]).x \approx_\tau y \to f'(x) \approx_\sigma g'(y))$$

By applying this to x, y and $x \triangleright \approx_\tau y$ using the dependent version of $\circledast$ defined in (26) we get

$$\triangleright[f' \leftarrow f, g' \leftarrow g, a \leftarrow x, b \leftarrow y].(f'(a) \approx_\sigma g'(b))$$

By (29) this is equal to

$$\mathsf{next}\,[f' \leftarrow f, a \leftarrow x].(f'(a)) \triangleright \approx_\sigma \mathsf{next}\,[g' \leftarrow g, b \leftarrow y].(g'(b))$$

which by rule (24) is equal to

$$(f \circledast x) \triangleright \approx_\sigma (g \circledast y)$$

□

Next we show that θ respects the logical relation.

Lemma 6.11. Let x, y of type $\triangleright[\![\sigma]\!]$, if $(x \triangleright \approx_\sigma y)$ then $\theta_\sigma(x) \approx_\sigma \theta_\sigma(y)$

Proof. We prove the statement by guarded recursion. Thus, we assume the statement holds "later" and we proceed by induction on σ. All the cases for the types that are interpreted using the lifting – namely the unit type and the sum type – in Definition 6.2 hold by definition of the lifting relation.

First the case for the function types: Assume $\sigma = \tau_1 \to \tau_2$ and assume f and g of type $\triangleright[\![\tau_1 \to \tau_2]\!]$ such that $f \triangleright \approx_{\tau_1\to\tau_2} g$. We must show that if $x, y : [\![\tau_1]\!]^\kappa$ and $x \approx_{\tau_1} y$ then $(\theta_{\tau_1\to\tau_2}(f))(x) \approx_{\tau_2} (\theta_{\tau_1\to\tau_2}(g))(y))$.

So suppose $x \approx_{\tau_1} y$, then also $\triangleright(x \approx_{\tau_1} y)$, which by (29) is equal to $\mathsf{next}(x) \triangleright \approx_{\tau_1} \mathsf{next}(y)$. By applying Lemma 6.10 to this and $f \triangleright \approx_{\tau_1\to\tau_2} g$ we get

$$f \circledast (\mathsf{next}\, x) \triangleright \approx_{\tau_2} g \circledast \mathsf{next}\, y$$

By induction hypothesis on τ_2, we get $\theta_{\tau_2}(f \circledast (\mathsf{next}\, x)) \approx_{\tau_2} \theta_{\tau_2}(g \circledast (\mathsf{next}\, y))$. We conclude by observing that by definition of θ, $\theta_{\tau_1\to\tau_2}(f)(x) = \theta_{\tau_2}(f \circledast \mathsf{next}(x))$.

The case of the product is straightforward.

For the case of recursive types, assume $\phi \triangleright \approx_{\mu\alpha.\tau} \psi$. This is type equal to

$$\triangleright[x \leftarrow \phi, y \leftarrow \psi].(x \approx_{\mu\alpha.\tau} y)$$

By definition this is equal to

$$\triangleright[x \leftarrow \phi, y \leftarrow \psi].(x \triangleright \approx_{\tau[\mu\alpha.\tau/\alpha]} y)$$

By guarded recursive hypothesis we get

$$\triangleright[x \leftarrow \phi, y \leftarrow \psi].(\theta_{\tau[\mu\alpha.\tau/\alpha]}(x) \approx_{\tau[\mu\alpha.\tau/\alpha]} \theta_{\tau[\mu\alpha.\tau/\alpha]}(y))$$

By (29) this is equal to

$$(\mathsf{next}\,[x \leftarrow \phi]\,.(\theta_{\tau[\mu\alpha.\tau/\alpha]}(x))) \triangleright \approx_{\tau[\mu\alpha.\tau/\alpha]} \ (\mathsf{next}\,[y \leftarrow \psi]\,.(\theta_{\tau[\mu\alpha.\tau/\alpha]}(y)))$$

This equals

$$(\mathsf{next}(\theta_{\tau[\mu\alpha.\tau/\alpha]}) \circledast \phi \triangleright \approx_{\tau[\mu\alpha.\tau/\alpha]} \ (\mathsf{next}(\theta_{\tau[\mu\alpha.\tau/\alpha]}) \circledast \psi$$

By Lemma 6.11 $\mathsf{next}(\theta_{\tau[\mu\alpha.\tau/\alpha]}) \circledast \phi$ is equal to $\theta_{\mu\alpha.\tau}(\phi)$ thus we can derive

$$\theta_{\mu\alpha.\tau}(\phi) \triangleright \approx_{\tau[\mu\alpha.\tau/\alpha]} \ \theta_{\mu\alpha.\tau}(\psi)$$

which by definition of $\approx_{\mu\alpha.\tau}$ is

$$\theta_{\mu\alpha.\tau}(\phi) \ \approx_{\mu\alpha.\tau} \ \theta_{\mu\alpha.\tau}(\psi)$$

□

Next we generalise Lemma 6.3 to hold for $\approx_\sigma$ for all σ.

Lemma 6.12. Let σ be a closed FPC type and let x and y of type $[\![\sigma]\!]$, if $x \approx_\sigma y$ then $\delta_\sigma(x) \approx_\sigma y$ and $x \approx_\sigma \delta_\sigma(y)$.

Proof. The proof is by guarded recursion and then by induction on the type σ. Thus, assume this lemma holds "later", and proceed by induction on σ. The cases of the unit type and coproduct follow from Lemma 6.3 and the case of products follows by induction from the fact that $\delta_{\tau_i}(\pi_i(x)) = \pi_i(\delta_{\tau_1 \times \tau_2}(x))$, for $i = 1, 2$. The case of function types follows from the fact that $\delta_{\sigma\to\tau}(f)(x) = \delta_\tau(f(x))$.

For the case of recursive types assume $x \approx_{\mu\alpha.\tau} y$. Note that

$$\begin{aligned} x \ \approx_{\mu\alpha.\tau} \ y &\equiv x \triangleright \approx_{\tau[\mu\alpha.\tau/\alpha]} \ y \\ &\equiv \triangleright\,[x' \leftarrow x, y' \leftarrow y]\,.x' \ \approx_{\tau[\mu\alpha.\tau/\alpha]} \ y' \end{aligned}$$

Using the dependent version of $\circledast$ as defined in (26) we can apply the guarded recursion assumption to conclude $\triangleright\,[x' \leftarrow x, y' \leftarrow y]\,.x' \ \approx_{\tau[\mu\alpha.\tau/\alpha]} \ \delta_{\tau[\mu\alpha.\tau/\alpha]}(y')$. Note that the delay operator is the composition $\theta \circ \mathsf{next}$, thus y' appears under next. We can thus employ (27) to derive that $\triangleright\,[x' \leftarrow x]\,.x' \ \approx_{\tau[\mu\alpha.\tau/\alpha]} \ \theta_{\tau[\mu\alpha.\tau/\alpha]}(y)$. From here we conclude by a simple computation:

$$\begin{aligned} \triangleright\,[x' \leftarrow x]\,.x' \ \approx_{\tau[\mu\alpha.\tau/\alpha]} \ \theta_{\tau[\mu\alpha.\tau/\alpha]}(y) &\equiv x \triangleright \approx_{\tau[\mu\alpha.\tau/\alpha]} \ \mathsf{next}(\theta_{\tau[\mu\alpha.\tau/\alpha]}(y)) \\ &\equiv x \triangleright \approx_{\tau[\mu\alpha.\tau/\alpha]} \ \mathsf{next}(\theta_{\tau[\mu\alpha.\tau/\alpha]}) \circledast \mathsf{next}(y) \\ &\equiv x \triangleright \approx_{\tau[\mu\alpha.\tau/\alpha]} \ \theta_{\mu\alpha.\tau}(\mathsf{next}(y)) \\ &\equiv x \ \approx_{\mu\alpha.\tau} \ \delta_{\mu\alpha.\tau}(y) \end{aligned}$$

□

Lemma 6.13. Let σ be a closed FPC type and let x, y of type $[\![\sigma]\!]^{\mathrm{gl}}$. If $x \approx^{\mathrm{gl}}_\sigma y$ then $x \approx^{\mathrm{gl}}_\sigma \delta^{\mathrm{gl}}_\sigma(y)$ and $\delta^{\mathrm{gl}}_\sigma(x) \approx^{\mathrm{gl}}_\sigma y$

Proof. Direct from Lemma 6.12. □

Finally, we now show that terms preserve the logical relation. Note that the relation

is not reflexive. For example the function $f\colon L1 \to L1$ defined by $f(\eta(*)) = \eta(*)$ and $f(\theta_{L1}(x)) = \bot$ does not satisfy $f \approx_{1\to 1} f$.

Lemma 6.14. If $\Gamma \vdash M : \sigma$, then $[\![M]\!] \approx_{\Gamma,\sigma} [\![M]\!]$.

Proof. The proof is by induction on M and we just show the interesting cases. In all cases we will assume $\Gamma \equiv x_1 : \sigma_1 .., x_n : \sigma_n$ and that we are given $\vec{x}$ and $\vec{y}$ such that $\vec{x} \approx_{\vec{\sigma}} \vec{y}$.

In the case of function application $M = NN'$ we know by induction hypothesis that $[\![N]\!] \approx_{\Gamma,\tau\to\sigma} [\![N]\!]$, and hence $[\![N]\!](\vec{x}) \approx_{\tau\to\sigma} [\![N]\!](\vec{y})$. Also by induction hypothesis we know that $[\![N']\!] \approx_{\Gamma,\tau} [\![N']\!]$, and hence

$$[\![N]\!](\vec{x})([\![N']\!](\vec{x})) \approx_{\sigma} [\![N]\!](\vec{y})([\![N']\!](\vec{y}))$$

Since $[\![NN']\!](\vec{x}) = [\![N]\!](\vec{x})([\![N']\!](\vec{x}))$ this concludes the case.

In the case for the lambda abstraction we have to prove that $[\![\lambda x.M]\!] \approx_{\Gamma,\tau\to\sigma} [\![\lambda x.M]\!]$. To prove this it suffices to show that

$$[\![\lambda x.M]\!](\vec{x}) \approx_{\tau\to\sigma} [\![\lambda x.M]\!](\vec{y})$$

for all related $\vec{x}$, $\vec{y}$ in the context $[\![\Gamma]\!]$. By definition of $\approx_{\tau\to\sigma}$ this means that, assuming $x \approx_{\tau} y$ we must prove $[\![\lambda x.M]\!](\vec{x})(x) \approx_{\sigma} [\![\lambda x.M]\!](\vec{y})(y)$. Since $[\![\lambda x.M]\!](\vec{x})(x) = [\![M]\!](\vec{x}, x)$ this follows directly from the induction hypothesis.

For case expressions, to prove that

$$[\![\texttt{case}\ L\ \texttt{of}\ x_1.M; x_2.N]\!](\vec{x}) \approx_{\tau} [\![\texttt{case}\ L\ \texttt{of}\ x_1.M; x_2.N]\!](\vec{y})$$

it suffices to prove that

$$[\![\lambda x.\texttt{case}\ x\ \texttt{of}\ x_1.M; x_2.N]\!](\vec{x}) \approx_{\sigma\to\tau} [\![\lambda x.\texttt{case}\ x\ \texttt{of}\ x_1.M; x_2.N]\!](\vec{y}) \tag{34}$$

Thus that for all x, y s.t. $x \approx_{\tau_1+\tau_2} y$

$$[\![\lambda x.\texttt{case}\ x\ \texttt{of}\ x_1.M; x_2.N]\!](\vec{x})(x) \approx_{\tau} [\![\lambda x.\texttt{case}\ x\ \texttt{of}\ x_1.M; x_2.N]\!](\vec{y})(y)$$

holds. We prove (34) by guarded recursion. Thus, we assume the statement holds "later" and we proceed by case analysis on x and y. When x is $\eta(x')$ and y is $\eta(y')$ either x' and y' are both in the left component or they are both in the right component of the sum. The former case $x' = \iota_1(x'')$ and $y' = \iota_1(y'')$ reduces to

$$[\![M]\!](\vec{x}, x'') \approx_{\tau} [\![M]\!](\vec{y}, y'')$$

which follows from the induction hypothesis, and the latter case is similar.

Now consider the case of $x = \theta_{\tau_1+\tau_2}(x')$ and $y = \eta(v)$. Since by assumption $x \approx_{\tau_1+\tau_2} y$ there exists n and w such that $x = \delta^n_{\tau_1+\tau_2}(\eta(w))$ and $w \approx_{\tau_1+\tau_2} v$. As before, v and w must be in the same component of the coproduct, so assume $w = \iota_1(w')$ and $v = \iota_1(v')$ such that $w' \approx_{\tau_1} v'$. By induction hypothesis we know that $[\![M]\!](\vec{x}) \approx_{\tau_1\to\tau} [\![M]\!](\vec{y})$ and thus that $[\![M]\!](\vec{x})(w') \approx_{\tau} [\![M]\!](\vec{y})(v')$. By Lemma 6.12 this implies $\delta^n_{\tau}([\![M]\!](\vec{x})(w')) \approx_{\tau} [\![M]\!](\vec{y})(v')$. Since

$$[\![M]\!](\vec{x})(w') = [\![\lambda x.\texttt{case}\ x\ \texttt{of}\ x_1.M; x_2.N]\!](\vec{x})(\eta(w)),$$

by Lemma 4.6 we get

$$\begin{aligned}\delta^n_\tau(\llbracket M\rrbracket(\vec{x})(w')) &= \delta^n_\tau(\llbracket \lambda x.\mathtt{case}\ x\ \mathtt{of}\ x_1.M; x_2.N\rrbracket)(\vec{x})(\eta(w)))\\ &= \llbracket \lambda x.\mathtt{case}\ x\ \mathtt{of}\ x_1.M; x_2.N\rrbracket(\vec{x})(\delta^n_{\tau_1+\tau_2}(\eta(w)))\end{aligned}$$

and thus we conclude

$$\llbracket \lambda x.\mathtt{case}\ x\ \mathtt{of}\ x_1.M; x_2.N\rrbracket(\vec{x})(\delta^n_{\tau_1+\tau_2}(\eta(w))) \approx_{\tau_1+\tau_2} \llbracket \lambda x.\mathtt{case}\ x\ \mathtt{of}\ x_1.M; x_2.N\rrbracket)(\vec{x})(\eta(v))$$

which is what we wanted to show.

The last case is when x is $\theta_{\tau_1+\tau_2}(x')$ and y is $\theta_{\tau_1+\tau_2}(y')$. By guarded recursion we know that

$$\triangleright(\llbracket \lambda x.\mathtt{case}\ x\ \mathtt{of}\ x_1.M; x_2.N\rrbracket(\vec{x}) \approx_{\tau_1+\tau_2\to\tau} (\llbracket \lambda x.\mathtt{case}\ x\ \mathtt{of}\ x_1.M; x_2.N\rrbracket)(\vec{y}))$$

By (29) we get

$$\mathsf{next}(\llbracket \lambda x.\mathtt{case}\ x\ \mathtt{of}\ x_1.M; x_2.N\rrbracket(\vec{x})) \triangleright \approx_{\tau_1+\tau_2\to\tau} \mathsf{next}(\llbracket \lambda x.\mathtt{case}\ x\ \mathtt{of}\ x_1.M; x_2.N\rrbracket(\vec{y}))$$

Since the assumption $\theta_{\tau_1+\tau_2}(x') \approx_{\tau_1+\tau_2} \theta_{\tau_1+\tau_2}(y')$, means that $x' \triangleright \approx_{\tau_1+\tau_2} y'$, by Lemma 6.10 this implies

$$\mathsf{next}(\llbracket \lambda x.\mathtt{case}\ x\ \mathtt{of}\ x_1.M; x_2.N\rrbracket)(\vec{x}) \circledast x' \triangleright \approx_\tau \mathsf{next}(\llbracket \lambda x.\mathtt{case}\ x\ \mathtt{of}\ x_1.M; x_2.N\rrbracket)(\vec{y}) \circledast y'$$

By Lemma 6.11 this implies

$$\theta_\tau(\mathsf{next}(\llbracket \lambda x.\mathtt{case}\ x\ \mathtt{of}\ x_1.M; x_2.N\rrbracket)(\vec{x}) \circledast x') \approx_\tau \theta_\tau(\mathsf{next}(\llbracket \lambda x.\mathtt{case}\ x\ \mathtt{of}\ x_1.M; x_2.N\rrbracket)(\vec{y}) \circledast y')$$

By Lemma 4.6 we conclude that

$$\llbracket \lambda x.\mathtt{case}\ x\ \mathtt{of}\ x_1.M; x_2.N\rrbracket(\vec{x})(\theta_{\tau_1+\tau_2}(x')) \approx_\tau \llbracket \lambda x.\mathtt{case}\ x\ \mathtt{of}\ x_1.M; x_2.N\rrbracket)(\vec{y})(\theta_{\tau_1+\tau_2}(y'))$$

proving the case.

Finally we prove the two cases for the recursive types. We first consider the case for $\mathtt{unfold}\ M$ of type $\tau[\mu\alpha.\tau/\alpha]$. We have to show that

$$\llbracket \mathtt{unfold}\ M\rrbracket(\vec{x}) \approx_{\tau[\mu\alpha.\tau/\alpha]} \llbracket \mathtt{unfold}\ M\rrbracket(\vec{y})$$

By induction hypothesis we know that $\llbracket M\rrbracket(\vec{x}) \approx_{\mu\alpha.\tau} \llbracket M\rrbracket(\vec{y})$ which by definition of $\approx_{\mu\alpha.\tau}$ is $\llbracket M\rrbracket(\vec{x}) \triangleright \approx_{\tau[\mu\alpha.\tau/\alpha]} \llbracket M\rrbracket(\vec{y})$. By Lemma 6.11 we get

$$\theta_{\tau[\mu\alpha.\tau/\alpha]}(\llbracket M\rrbracket(\vec{x})) \approx_{\tau[\mu\alpha.\tau/\alpha]} \theta_{\tau[\mu\alpha.\tau/\alpha]}(\llbracket M\rrbracket(\vec{y}))$$

and by definition of the interpretation function this is what we wanted.

Now the case for $\mathtt{fold}\ M$ of type $\mu\alpha.\tau$. By induction hypothesis we know that $\llbracket M\rrbracket(\vec{x}) \approx_{\tau[\mu\alpha.\tau/\alpha]} \llbracket M\rrbracket(\vec{y})$ which implies $\triangleright(\llbracket M\rrbracket(\vec{x}) \approx_{\tau[\mu\alpha.\tau/\alpha]} \llbracket M\rrbracket(\vec{y}))$ which is equal to

$$\mathsf{next}(\llbracket M\rrbracket(\vec{x})) \triangleright \approx_{\tau[\mu\alpha.\tau/\alpha]} \mathsf{next}(\llbracket M\rrbracket(\vec{y})).$$

By definition of $\approx_{\mu\alpha.\tau}$ this is precisely $\mathsf{next}(\llbracket M\rrbracket(\vec{x})) \approx_{\mu\alpha.\tau} \mathsf{next}(\llbracket M\rrbracket(\vec{y}))$ which by definition of the interpretation function is

$$\llbracket \mathtt{fold}\ M\rrbracket(\vec{x}) \approx_{\mu\alpha.\tau} \llbracket \mathtt{fold}\ M\rrbracket(\vec{y})$$

□

6.5. *Extensional computational adequacy*

Contextual equivalence of FPC is defined in the standard way by observing convergence at unit type.

Definition 6.15 (Contexts).

$$\begin{aligned}\mathsf{Ctx} := {} & [-] \mid \lambda x.\,\mathsf{Ctx} \mid \mathsf{Ctx}\,N \mid M\,\mathsf{Ctx} \\ & \mid \mathtt{inl}\ \mathsf{Ctx} \mid \mathtt{inr}\ \mathsf{Ctx} \mid \langle \mathsf{Ctx}, M\rangle \mid \langle M, \mathsf{Ctx}\rangle \mid \mathtt{fst}\ \mathsf{Ctx} \mid \mathtt{snd}\ \mathsf{Ctx} \\ & \mid \mathtt{case}\ \mathsf{Ctx}\ \mathtt{of}\ x_1.M; x_2.N \\ & \mid \mathtt{case}\ L\ \mathtt{of}\ x_1.\,\mathsf{Ctx}; x_2.N \mid \mathtt{case}\ L\ \mathtt{of}\ x_1.M; x_2.\,\mathsf{Ctx} \\ & \mid \mathtt{unfold}\ \mathsf{Ctx} \mid \mathtt{fold}\ \mathsf{Ctx}\end{aligned}$$

Intuitively, a context is a term that takes a term and returns a new term.

We define the "fill hole" function $\cdot[\cdot] : \mathsf{Ctx} \times \mathtt{OTerm}_{\text{FPC}} \to \mathtt{OTerm}_{\text{FPC}}$ by induction on the context in the standard way. Note that this may capture free variables in the term being substituted.

We say that a context C has type $(\Gamma, \sigma) \to (\Delta, \tau)$ if $\Delta \vdash C[M] : \tau$ whenever $\Gamma \vdash M : \sigma$. This can be captured by a typing relation on contexts as defined in Figure 13.

Definition 6.16. Let $\Gamma \vdash M, N : \tau$. We say that M, N are contextually equivalent, written $M \approx_{\mathtt{CTX}} N$, if for all contexts C of type $(\Gamma, \tau) \to (-, 1)$

$$C[M] \Downarrow \langle\rangle \iff C[N] \Downarrow \langle\rangle$$

Finally we can state the main theorem of this section.

Theorem 6.17 (Extensional Computational Adequacy). If $\Gamma \vdash M, N : \tau$ and $[\![M]\!]^{\text{gl}} \approx^{\text{gl}}_{\Gamma,\tau} [\![N]\!]^{\text{gl}}$ then $M \approx_{\mathtt{CTX}} N$.

To prove this theorem, we need the following lemma stating that contexts preserve the logical relation.

Lemma 6.18. Let $\Gamma \vdash M : \tau$ and $\Gamma \vdash N : \tau$ and suppose $[\![M]\!] \approx_{\Gamma,\tau} [\![N]\!]$. If C is a context such that $C : \Gamma, \tau \to \Delta, \sigma$ then $[\![C[M]]\!] \approx_{\Delta,\sigma} [\![C[N]]\!]$

Proof. The proof is by induction on C and most cases can be proved either very similarly to corresponding cases of Lemma 6.14, or by direct application of Lemma 6.14. We show how to do the latter in two cases.

For a context $\mathtt{unfold}\ C$ of type $(\Gamma, \sigma) \to (\Delta, \tau[\mu\alpha.\tau/\alpha])$ we have by induction that C has type $(\Gamma, \sigma) \to (\Delta, \mu\alpha.\tau)$ and thus induction hypothesis we know that $[\![C[M]]\!](\vec{x}) \approx_{\mu\alpha.\tau} [\![C[N]]\!](\vec{y})$. By Lemma 6.14 we know that $[\![\lambda x.\mathtt{unfold}\ x]\!] \approx_{(\mu\alpha.\tau)\to(\tau[\mu\alpha.\tau/\alpha])} [\![\lambda x.\mathtt{unfold}\ x]\!]$. By applying this latter fact to the induction hypothesis we obtain $[\![\mathtt{unfold}\ C[M]]\!](\vec{x}) \approx_{\tau[\mu\alpha.\tau/\alpha]} [\![\mathtt{unfold}\ C[N]]\!](\vec{y})$ which is what we wanted.

When the context binds a variable one has to be a bit more careful. For example, for a context of the form $\mathtt{case}\ L\ \mathtt{of}\ x_1.C; x_2.N'$ of type $(\Gamma, \tau) \to (\Delta, \sigma)$ we have by induction that C has type $(\Gamma, \tau) \to ((\Delta, x_1 \colon \tau_1), \sigma)$ and thus by induction hypothesis we know by applying the context parameters that $[\![C[M]]\!](\vec{x}) \approx_{\tau_1,\sigma} [\![C[N]]\!](\vec{y})$. From this we also

$$\frac{}{- : (\Gamma, \tau) \to (\Gamma, \tau)} \qquad \frac{C : (\Gamma, \tau) \to ((\Delta, x : \sigma'), \sigma)}{(\lambda x.C) : (\Gamma, \tau) \to (\Delta, \sigma' \to \sigma)}$$

$$\frac{C : (\Gamma, \tau) \to (\Delta, \tau' \to \sigma) \qquad \Delta \vdash N : \tau'}{CN : (\Gamma, \tau) \to (\Delta, \sigma)} \qquad \frac{C : (\Gamma, \sigma) \to (\Delta, \tau') \qquad \Delta \vdash M : \tau' \to \sigma}{MC : (\Gamma, \sigma) \to (\Delta, \sigma)}$$

$$\frac{C : (\Gamma, \sigma) \to (\Delta, \mu\alpha.\tau)}{\texttt{unfold}\, C : (\Gamma, \sigma) \to (\Delta, \tau[\mu\alpha.\tau/\alpha])} \qquad \frac{C : (\Gamma, \sigma) \to (\Delta, \tau[\mu\alpha.\tau/\alpha])}{\texttt{fold}\, C : (\Gamma, \sigma) \to (\Delta, \mu\alpha.\tau)}$$

$$\frac{C : (\Gamma, \tau) \to (\Delta, \tau_1 \times \tau_2)}{\texttt{fst}\, C : (\Gamma, \tau) \to (\Delta, \tau_1)} \qquad \frac{C : (\Gamma, \tau) \to (\Delta, \tau_1 \times \tau_2)}{\texttt{snd}\, C : (\Gamma, \tau) \to (\Delta, \tau_2)}$$

$$\frac{C : (\Gamma, \tau) \to (\Delta, \tau_1) \qquad \Delta \vdash N : \tau_2}{\langle C, N\rangle : (\Gamma, \tau) \to (\Delta, \tau_1 \times \tau_2)} \qquad \frac{C : (\Gamma, \tau) \to (\Delta, \tau_2) \qquad \Delta \vdash M : \tau_1}{\langle M, C\rangle : (\Gamma, \tau) \to (\Delta, \tau_1 \times \tau_2)}$$

$$\frac{C : (\Gamma, \tau) \to (\Delta, \tau_1 + \tau_2) \qquad \Delta, x_1 : \tau_1 \vdash M : \sigma \qquad \Delta, x_2 : \tau_2 \vdash N : \sigma}{\texttt{case}\, C\, \texttt{of}\, x_1.M; x_2.N : (\Gamma, \tau) \to (\Delta, \sigma)}$$

$$\frac{\Delta \vdash L : \tau_1 + \tau_2 \qquad C : (\Gamma, \tau) \to ((\Delta, x_1 : \tau_1), \sigma) \qquad \Delta, x_2 : \tau_2 \vdash N : \sigma}{\texttt{case}\, L\, \texttt{of}\, x_1.C; x_2.N : (\Gamma, \tau) \to (\Delta, \sigma)}$$

$$\frac{\Delta \vdash L : \tau_1 + \tau_2 \qquad \Delta, x_1 : \tau_1 \vdash M : \sigma \qquad C : (\Gamma, \tau) \to ((\Delta, x_2 : \tau_2), \sigma)}{\texttt{case}\, L\, \texttt{of}\, x_1.M; x_2.C : (\Gamma, \tau) \to (\Delta, \sigma)}$$

$$\frac{C : (\Gamma, \tau) \to (\Delta, \tau_1)}{\texttt{inl}\, C : (\Gamma, \tau) \to (\Delta, \tau_1 + \tau_2)} \qquad \frac{C : (\Gamma, \tau) \to (\Delta, \tau_2)}{\texttt{inr}\, C : (\Gamma, \tau) \to (\Delta, \tau_1 + \tau_2)}$$

Fig. 13. Typing judgment for contexts

know that

$$[\![\lambda x_1.C[M]]\!](\vec{x}) \;\approx_{\tau_1 \to \sigma}\; [\![\lambda x_1.C[N]]\!](\vec{y}). \tag{35}$$

By Lemma 6.14 we know that

$$[\![\lambda x.\texttt{case}\, L\, \texttt{of}\, x_1.x(x_1); x_2.N']\!](\vec{x}) \;\approx_{(\tau_1 \to \sigma) \to \sigma}\; [\![\lambda x.\texttt{case}\, L\, \texttt{of}\, x_1.x(x_1); x_2.N']\!](\vec{y}).$$

By applying this to (35) we conclude. □

As a direct consequence we get the following lemma.

Lemma 6.19. If $\Gamma \vdash M, N : \tau$ and $[\![M]\!]^{\mathrm{gl}} \;\approx^{\mathrm{gl}}_{\Gamma,\tau}\; [\![N]\!]^{\mathrm{gl}}$ then for all contexts C of type $(\Gamma, \tau) \to (-, 1)$, $[\![C[M]]\!]^{\mathrm{gl}} \;\approx^{\mathrm{gl}}_{(-,1)}\; [\![C[N]]\!]^{\mathrm{gl}}$

The next lemma states that if two computations of unit type are related then the first converges iff the second converges. Note that this lemma needs to be stated using the fact that the two computations are *globally related*.

Lemma 6.20. For all x, y of type $[\![1]\!]^{\mathrm{gl}}$, if $x \approx^{\mathrm{gl}}_{(-,1)} y$ then

$$\Sigma n.x = (\delta_1^{\mathrm{gl}})^n(\eta(*)) \Leftrightarrow \Sigma m.y = (\delta_1^{\mathrm{gl}})^m(\eta(*))$$

Proof. We show the left to right implication, so suppose $x = (\delta_1^{\mathrm{gl}})^n(\eta(*))$. The proof proceeds by induction on n. If $n = 0$ then since by assumption $\forall\kappa.x[\kappa] \approx_1 y[\kappa]$, by definition of $\approx_1$, for all κ, there exists an m such that $y[\kappa] = \delta_1^m(\eta(*))$. By type isomorphism (8), since m is a natural number, this implies there exists m such that for all κ, $y[\kappa] = \delta_1^m(\eta(*))$ which implies $y = \Lambda\kappa.y[\kappa] = (\delta_1^{\mathrm{gl}})^m(\eta(*))$.

In the inductive case $n = n'+1$, since by Lemma 6.6 $(\delta_1^{\mathrm{gl}})^{n'}([\![v]\!]^{\mathrm{gl}}) \approx_1^{\mathrm{gl}} y$, the induction hypothesis implies $\Sigma m.y = (\delta_1^{\mathrm{gl}})^m(\eta(*))$. □

Proof of Theorem 6.17 Suppose $[\![M]\!]^{\mathrm{gl}} \approx^{\mathrm{gl}}_{\Gamma,\tau} [\![N]\!]^{\mathrm{gl}}$ and that C has type $(\Gamma, \tau) \to (-, 1)$. We show that if $C[M] \Downarrow \langle\rangle$ also $C[N] \Downarrow \langle\rangle$. So suppose $C[M] \Downarrow \langle\rangle$. By definition this means $\Sigma n.C[M] \Downarrow^n \langle\rangle$. By Corollary 6.1 we get $\Sigma n.\forall\kappa.\, [\![C[M]]\!] = (\delta_1)^n(\eta(*))$ which is equivalent to $\Sigma n.[\![C[M]]\!]^{\mathrm{gl}} = (\delta_1^{\mathrm{gl}})^n(\eta(*))$. From the assumption and Lemma 6.19 we know that $[\![C[M]]\!]^{\mathrm{gl}} \approx_1^{\mathrm{gl}} [\![C[N]]\!]^{\mathrm{gl}}$, so by Lemma 6.20 there exists an m such that $[\![C[N]]\!]^{\mathrm{gl}} = (\delta_1^{\mathrm{gl}})^m(\eta(*))$. By applying the Corollary 6.1 once again we get $C[N] \Downarrow \langle\rangle$ as desired. □

7. Conclusions and Future Work

We have shown that programming languages with recursive types can be given sound and computationally adequate denotational semantics in guarded dependent type theory. The semantics is intensional, in the sense that it can distinguish between computations computing the same result in different number of steps, but we have shown how to capture extensional equivalence in the model by constructing a logical relation on the interpretation of types.

This work can be seen as a first step towards a formalisation of domain theory in type theory. Other, more direct formalisations have been carried out in Coq, e.g. [BKV09; Ben+10] but we believe that the synthetic viewpoint offers a more abstract and simpler presentation of the theory. Moreover, we hope that the success of guarded recursion for operational reasoning, mentioned in the introduction, can be carried over to denotational models of advanced programming language features in future work.

Future work also includes implementation of gDTT in a proof assistant, allowing for the theory of this paper to be machine verified. Currently, initial experiments are being carried out in this direction [Bir+16].

References

[AM13] Robert Atkey and Conor McBride. "Productive Coprogramming with Guarded Recursion". In: *ICFP*. 2013, pp. 197–208.

[App+07] 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*. 2007, pp. 109–122.

[BBM14] Aleš Bizjak, Lars Birkedal, and Marino Miculan. "A Model of Countable Nondeterminism in Guarded Type Theory". In: *RTA-TLCA*. 2014, pp. 108–123.

[Ben+10] Nick Benton, Lars Birkedal, Andrew Kennedy, and Carsten Varming. "Formalizing Domains, Ultrametric Spaces and Semantics of Programming Languages". 2010.

[Bir+12] L. Birkedal, R. Møgelberg, J. Schwinghammer, and K. Støvring. "First steps in synthetic guarded domain theory: step-indexing in the topos of trees". In: *LICS*. 2012.

[Bir+16] L. Birkedal, A. Bizjak, R. Clouston, H.B. Grathwohl, B. Spitters, and A. Vezzosi. "Guarded Cubical Type Theory: Path Equality for Guarded Recursion". In: *CSL 2016*. 2016.

[Biz+16] A. Bizjak, H. B. Grathwohl, R. Clouston, R. E. Møgelberg, and L. Birkedal. "Guarded Dependent Type Theory with Coinductive Types". In: *FoSSaCS*. 2016.

[BKV09] Nick Benton, Andrew Kennedy, and Carsten Varming. "Some Domain Theory and Denotational Semantics in Coq". In: *TPHOLs*. 2009.

[BM13] Lars Birkedal and Rasmus Ejlers Møgelberg. "Intensional Type Theory with Guarded Recursive Types qua Fixed Points on Universes". In: *LICS*. 2013, pp. 213–222.

[BM15] Aleš Bizjak and Rasmus Ejlers Møgelberg. "A model of guarded recursion with clock synchronisation". In: *MFPS*. 2015.

[Cap05] Venanzio Capretta. "General recursion via coinductive types". In: *Logical Methods in Computer Science* (2005).

[Dan12] Nils Anders Danielsson. "Operational semantics using the partiality monad". In: *ICFP*. 2012, pp. 127–138.

[Esc99] M.H. Escardó. "A metric model of PCF". Laboratory for Foundations of Computer Science, University of Edinburgh. 1999.

[Hyl91] J. Martin E. Hyland. "First steps in synthetic domain theory". In: *Category Theory*. 1991, pp. 131–156.

[MP08] C. McBride and R. Paterson. "Applicative Programming with Effects". In: *Journal of Functional Programming* 18.1 (2008).

[MP16] Rasmus Ejlers Møgelberg and Marco Paviotti. "Denotational Semantics of recursive types in Synthetic Guarded Domain Theory". In: *LICS*. 2016.

[Møg14] Rasmus Ejlers Møgelberg. "A type theory for productive coprogramming via guarded recursion". In: *CSL-LICS*. 2014.

[Nak00] Hiroshi Nakano. "A modality for recursion". In: *LICS*. 2000, pp. 255–266.

[Pit96] Andrew M. Pitts. "Relational Properties of Domains". In: *Inf. Comput.* 127.2 (1996), pp. 66–90.

[PMB15] Marco Paviotti, Rasmus Ejlers Møgelberg, and Lars Birkedal. "A Model of PCF in Guarded Type Theory". In: *Electr. Notes Theor. Comput. Sci.* (2015).

[Reu96] Bernhard Reus. "Synthetic Domain Theory in Type Theory: Another Logic of Computable Functions". In: *TPHOLs*. 1996.

[Ros86] G. Rosolini. "Continuity and effectiveness in topoi". PhD thesis. University of Oxford, 1986.

[SB14] Kasper Svendsen and Lars Birkedal. "Impredicative Concurrent Abstract Predicates". In: *ESOP*. 2014.

[Sim02] Alex K. Simpson. "Computational Adequacy for Recursive Types in Models of Intuitionistic Set Theory". In: *LICS*. 2002, pp. 287–298.

[Uni13] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study: `http://homotopytypetheory.org/book`, 2013.