# The Scott model of PCF in Univalent Type Theory

Tom de Jong

University of Birmingham, United Kingdom

**Abstract**

We develop the Scott model of the programming language PCF in constructive predicative univalent type theory. Moreover, we define the operational semantics of PCF and prove soundness and computational adequacy of the model with respect to the operational semantics. We use the lifting monad to account for the non-termination in PCF. Our results show that lifting is a viable approach to partiality in type theory.

## 1 Introduction

We develop the Scott model of the programming language PCF [Str06] in constructive predicative univalent mathematics. Our development differs from the classical approach in three key ways. First of all, our work is constructive. Secondly, we work predicatively and do not assume propositional resizing. This has led us to consider directed *families*, rather than subsets. Thirdly, we situate our development in the framework of univalent mathematics.

The programming language PCF has general recursion and hence non-termination, and this is what makes a constructive type theoretic treatment challenging. In classical mathematics, the construction of adding a (least) element to a set is used to deal with partiality. Moreover, this construction yields a domain. Constructively, this is no longer true, so to account for partiality, we instead work with the partial map classifier monad (also known as the lifting monad) from topos theory [Koc91], which has been extended to univalent type theory by Cory Knapp and Martín Escardó [Kna18, EK17].

The framework of univalent mathematics has led us to consider the following technical issues. Firstly, we use propositional truncation to find a suitable encoding of the reflexive transitive closure of a (proposition valued) relation. Secondly, to show that PCF types and terms have decidable equality, we have formalised more general results on indexed W-types with decidable equality.

Finally, let us discuss two applications of this work. First of all, our results serve as validation that partiality via lifting is a suitable approach to partiality in constructive univalent type theory. Secondly, computational adequacy (see Theorem 7.2) can be used to compute total functions as described in Section 7.1.

### 1.1 Related work

Partiality in type theory has been the subject of recent study. We briefly discuss the different approaches. First, there are Capretta's delay monad and its quotient by weak bisimilarity, which have been studied by Uustula, Chapman and Veltri [CUV17]. A drawback of the quotient is that some form of choice is needed (countable choice suffices) to show that it is again a monad. Another approach is laid out in [ADK17] by Altenkirch, Danielsson and Kraus. They construct (essentially by definition) the free $\omega$-cpo with a least element using a higher inductive-inductive type. Moreover, Altenkirch et al. show that, assuming countable choice, their free $\omega$-cpo coincides with the quotiented delay monad. In [Kna18], Knapp showed that, assuming countable choice, a restricted version (using a dominance) of the lifting is isomorphic to the quotiented delay monad.

One way in which the lifting distinguishes itself from the other approaches is that the lifting of a set can be seen as the free subsingleton complete poset with a least element, rather than the free $\omega$-cpo with a least element.

A constructive approach (a formalisation even) to domain theory and denotational semantics has already been given in 2009 by [BKV09] using a version of Capretta's delay monad. However, the objects have the "wrong equality", so that every object comes with an equivalence relation that maps must preserve. The framework of univalent mathematics in which we have placed our development provides a more natural approach. Moreover, we do not make use of Coq's impredicative `Prop` universe and our treatment incorporates directed complete posets (dcpos) and not just $\omega$-cpos.

## 1.2 Overview

The first section introduces basic domain theory in the framework of univalent type theory. We develop the material right up to the least fixed point operator, as this will suffice for our purposes.

We proceed by discussing the constructive issues with the classical approach to partiality and flat dcpos. In Section 4 we introduce the lifting monad as a solution to partiality in constructive type theory. We show that it is a monad and prove that the lifting may be seen as a free construction.

Section 5 lays out PCF and its operational semantics, while Section 6 discusses the constructive Scott model of PCF. In Section 7 we prove that the operational and denotational semantics work well together in the form of soundness and computational adequacy.

Section 8 aims to characterise propositions that arise from PCF terms of the base type as semidecidable propositions. To do so, we study the reflexive transitive closure of a relation and indexed W-types in general.

Finally, the appendix will discuss universe level issues surrounding the lifting and directed completeness.

## 1.3 Framework and formalisation

We work in a type theory with $\sum$, $\prod$ and inductive types (including the empty, unit and natural numbers types). We write Prop for the type of propositions (or subsingletons). Moreover, we assume propositional truncation and propositional extensionality. Finally, we need function extensionality. In particular, Theorem 8.20 in Section 8.2 requires a strong (dependent) form of function extensionality: $\prod_{f,g:\prod_{t:T} P(t)}(\prod_{t:T} f(t) = g(t)) \to f = g$ for every type $T$ and type family $P$ over $T$. However, we do not need full univalence at any point.

All our results up to Section 7.1 (and except for Section 3) have been formalised in the proof assistant Coq using the UniMath library [VAG+]. The general results from Section 8 have also been formalised, but their direct applications to PCF, e.g. single-valuedness of the operational semantics and PCF as an indexed W-type, have not. The code may be found at `https://github.com/tomdjong/UniMath/tree/paper`.

## 1.4 Acknowledgements

Acknowledgements. I would like to thank Martín Escardó for suggesting and supervising this project. I have also benefited from Benedikt Ahrens's support and his help with UniMath.

# 2 Basic domain theory

We introduce basic domain theory in the setting of constructive univalent mathematics. All definitions and theorems are straightforward adaptions of the usual ones in [AJ94, Section 2.1] and [Str06, Chapter 4].

## 2.1 Directed complete posets

**Definition 2.1.** A *poset* $(X, \leq)$ is a set $X$ together with a proposition valued binary relation $\leq\colon X \to X \to \mathsf{Prop}$ satisfying:

(i) *reflexivity*: $\prod_{x:X} x \leq x$;

(ii) *antisymmetry*: $\prod_{x,y:X} x \leq y \to y \leq x \to x = y$;

(iii) *transitivity*: $\prod_{x,y,z:X} x \leq y \to y \leq z \to x \leq z$.

**Definition 2.2.** Let $P$ and $Q$ be posets. A *poset morphism* from $P$ to $Q$ is a function on the underlying sets that preserves the order. One also says that the function is *monotone*.

**Definition 2.3.** Let $(X, \leq)$ be a poset and $I$ any type. Given a family $u : I \to X$, we write $u_i$ for $u(i)$. Such a family is called *directed* if it is inhabited (i.e. $\|I\|$) and $\prod_{i,j:I} \|\sum_{k:I}(u_i, u_j \leq u_k)\|$.

Observe that being directed is property, rather than structure.

**Definition 2.4.** Let $\mathcal{U}$ be a type universe. A poset $P$ is called *$\mathcal{U}$-directed complete* if every directed family in $P$ indexed by a type in $\mathcal{U}$ has a least upper bound in $P$. We call such a poset a *$\mathcal{U}$-dcpo*.

We shall often simply write dcpo, omitting reference to the type universe. A discussion on size/universe issues can be found in the appendix.

Finally, we denote least upper bounds of a directed family $u : I \to X$ by $\bigsqcup_{i:I} u_i$.

## 2.2 Morphisms of dcpos

**Definition 2.5.** Let $D$ and $E$ be dcpos. A poset morphism from $D$ to $E$ is a *dcpo morphism* or *continuous* if it preserves least upper bounds of directed families.

In fact, requiring that the function is monotone is redundant.

**Lemma 2.6.** *Let $D$ and $E$ be dcpos. If $f$ is a function (on the underlying sets) from $D$ to $E$ preserving least upper bounds of directed families, then $f$ is order preserving.*

*Proof.* Let $f : D \to E$ be a morphism of dcpos and suppose $x, y : D$ with $x \leq y$. Consider the family $1 + 1 \to D$ defined as $\mathsf{inl}(\star) \mapsto x$ and $\mathsf{inr}(\star) \mapsto y$. This family is easily seen to be directed and its least upper bound is $y$. Now $f$ preserves this least upper bound, so $f(x) \leq f(y)$. ∎

**Corollary 2.7.** *Every morphism of dcpos preserves directed families. That is, if $f : D \to E$ is a morphism of dcpos and $u$ is a directed family in $D$, then $f \circ u$ is a directed family in $E$.*

**Theorem 2.8.** *Let $D$ and $E$ be dcpos. The morphisms from $D$ to $E$ form a dcpo with the pointwise order.*

*Proof.* The least upper bound of a directed family of dcpo morphisms is also given pointwise. The proof only differs from the standard proof of [Str06, Theorem 4.2] in that it uses directed families, rather than subsets. One may consult the formalisation for the technical details. ∎

### 2.3 Dcpos with bottom

**Definition 2.9.** A *dcpo with bottom* is a dcpo $D$ together with a least element in $D$.

**Theorem 2.10.** *Let $D$ be a dcpo and let $E$ be a dcpo with bottom. Ordered pointwise, the morphisms from $D$ to $E$ form a dcpo with bottom.*

*Proof.* Since the order is pointwise, the least morphism from $D$ to $E$ is simply given by mapping every element in $D$ to the least element in $E$. The rest is as in Theorem 2.8. ■

**Definition 2.11.** Let $D$ and $E$ be dcpos with bottom. We will write $E^D$ for the dcpo with bottom that has the dcpo morphisms from $D$ to $E$ as its underlying set.

Dcpos with bottoms are interesting because of fixed points.

**Theorem 2.12.** *Let $D$ be a dcpo with bottom. There is a continuous function $\mu : D^D \to D$ that sends each continuous function to its least fixed point. In fact, $\mu$ satisfies:*

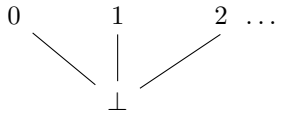
(i) *$f(\mu(f)) = \mu(f)$ for every continuous $f : D \to D$;*

(ii) *for every continuous $f : D \to D$ and each $d : D$, if $f(d) \leq d$, then $\mu(f) \leq d$.*

*Proof.* We have formalised the proof of [AJ94, Theorem 2.1.19]. We sketch the main construction here. For each natural number $n$, define $\mathsf{iter}(n) : D^D \to D$ as $\mathsf{iter}(n)(f) \equiv f^n(\bot) \equiv \underbrace{f(f(\dots(f}_{n \text{ times}}(\bot))\dots))$. By induction on $n$, one may show that every $\mathsf{iter}(n)$ is continuous. Then, the assignment $n \mapsto \mathsf{iter}(n)$ is a directed family in $D^{(D^D)}$. Finally, one defines $\mu$ as the least upper bound of this directed family. Recall that least upper bounds in the exponential are given pointwise, so that $\mu(f) = \sqcup_{n:\mathbb{N}} f^n(\bot)$. ■

## 3 Constructive issues with partiality

In classical mathematics, a partial map from $\mathbb{N}$ to $\mathbb{N}$ can simply be seen as total map from $\mathbb{N}$ to $\mathbb{N} \cup \{\bot\}$, where $\bot$ is some fresh element not in $\mathbb{N}$. The *flat dcpo* $\mathbb{N}_\bot$ is $\mathbb{N} \cup \{\bot\}$ ordered as in the following Hasse diagram:




Classically, it is easy to show that $\mathbb{N}_\bot$ is a directed complete poset (viz. every finite subset of $\mathbb{N}_\bot$ has a least upper bound in $\mathbb{N}_\bot$).

One could hope that the above translates directly into constructive univalent mathematics, that is, that $\mathbb{N} + 1$ is directed complete (in the sense of Definition 2.4). However, we can prove that this implies the Weak Limited Principle of Omniscience (WLPO), a constructive taboo, as follows.

In type theory, WLPO can be formulated as the following type:

$$\prod_{\alpha:\mathbb{N}\to 2} \left( \prod_{n:\mathbb{N}} \alpha(n) = 0 \right) + \left( \neg \prod_{n:\mathbb{N}} \alpha(n) = 0 \right). \tag{WLPO}$$

Define the type $\mathbb{N}_\infty$ of increasing sequences binary sequences by:

$$\mathbb{N}_\infty \equiv \sum_{\alpha:\mathbb{N}\to 2} \prod_{n:\mathbb{N}} (\alpha(n) = 1 \to \alpha(n+1) = 1).$$

**Lemma 3.1.** WLPO *is equivalent to a restricted version of* WLPO *where* $\alpha$ *just runs over increasing sequences, viz. the type:*

$$\prod_{\alpha:\mathbb{N}_\infty} \left(\prod_{n:\mathbb{N}} \alpha(n) = 0\right) + \left(\neg \prod_{n:\mathbb{N}} \alpha(n) = 0\right). \tag{$*$}$$

*Proof.* Since both types are propositions, proving the bi-implication suffices. Of course, one implication is trivial, because there is an obvious embedding of $\mathbb{N}_\infty$ into $\mathbb{N} \to 2$.

For the converse, suppose we have a term $p$ of $(*)$. Let $\alpha$ be any binary sequence. Define $\tilde{\alpha} : \mathbb{N}_\infty$ by:

$$\tilde{\alpha}(n) \equiv \begin{cases} 0 & \text{if } \prod_{k\leq n} \alpha(k) = 0; \\ 1 & \text{else.} \end{cases}$$

Observe that $\prod_{k\leq n} \alpha(k) = 0$ is decidable for every $n : \mathbb{N}$, so this definition is constructively sound. Finally, it is easily seen that $\prod_{n:\mathbb{N}} \alpha(n) = 0$ is equivalent to $\prod_{n:\mathbb{N}} \tilde{\alpha}(n) = 0$, so $p(\tilde{\alpha})$ is a proof of WLPO. ∎

**Lemma 3.2.** *Directed completeness of* $\mathbb{N} + 1$ *implies* WLPO.

*Proof.* Suppose that $\mathbb{N} + 1$ is directed complete. We employ the previous lemma. Let $\alpha : \mathbb{N}_\infty$. Define a sequence $u : \mathbb{N} \to \mathbb{N} + 1$ by:

$$u_n \equiv \begin{cases} \mathsf{inl}(n) & \text{if } \alpha(n) = 1; \\ \mathsf{inr}(\star) & \text{else.} \end{cases}$$

Since $\alpha$ is increasing, the family $u$ is directed. Thus, $u$ has a least upper bound $l$ in $\mathbb{N} + 1$. By definition of $u$, it follows that $l = \mathsf{inr}(\star)$ if and only if $\prod_{n:\mathbb{N}} \alpha(n) = 0$. But $l = \mathsf{inr}(\star)$ is decidable, so we get $(\prod_{n:\mathbb{N}} \alpha(n) = 0) + (\neg \prod_{n:\mathbb{N}} \alpha(n) = 0)$. ∎

# 4 Dealing with partiality constructively

In his PhD thesis [Kna18], Cory Knapp studied partial functions and recursion in univalent type theory. Most of the results in this section can be found in [Kna18] or in the paper [EK17] by Knapp and his supervisor Martín Escardó. Exceptions are Lemma 4.3, Theorem 4.5 and Theorem 4.7. The former, however, is implicit in the antisymmetry of the order on the lifting. The order on the lifting in this paper (see Theorem 4.6) is different from the order presented in [EK17] and [Kna18]. The two orders are equivalent however and this was first observed by Martín Escardó.

**Definition 4.1.** Let $X$ be any type. Define the *lifting of* $X$ as

$$\mathcal{L}\, X \equiv \sum_{P:\mathsf{Prop}} (P \to X).$$

We now define meaningful projections.

**Definition 4.2.** The function $\mathsf{isdefined} : \mathcal{L}\,X \to \mathsf{Prop}$ is defined as the first projection. The function $\mathsf{value} : \prod_{l:\mathcal{L}\,X} \mathsf{isdefined}(l) \to X$ is defined as: $\mathsf{value}(P,\varphi)(p) \equiv \varphi(p)$.

Since equality of $\sum$-types often requires $\mathsf{transport}$, it will be convenient to characterise the equality of $\mathcal{L}\,X$.

**Lemma 4.3.** *Let $X$ be any type and let $l, m : \mathcal{L}\,X$. Then $l = m$ and*

$$\sum_{e:\mathsf{isdefined}(l)\leftrightarrow\mathsf{isdefined}(m)} \mathsf{value}(l)\circ\mathsf{pr}_2(e) \sim \mathsf{value}(m)$$

*are logically equivalent.*[1]

*Proof.* Suppose we have a path $p : l = m$. By the characterisation of the identity type of $\sum$-types, we obtain $p_1 : \mathsf{isdefined}(l) = \mathsf{isdefined}(m)$ and $p_2 : \mathsf{transport}(p_1, \mathsf{value}(l)) = \mathsf{value}(m)$. Obviously, $p_1$ yields a term $\mathsf{eqtoiff}(p_1) : \mathsf{isdefined}(l) \leftrightarrow \mathsf{isdefined}(m)$. Using path induction on $p_1$, we can prove that $\mathsf{value}(l)\circ\mathsf{pr}_2(\mathsf{eqtoiff}(p_1)) = \mathsf{transport}(p_1, \mathsf{value}(l))$. Together with $p_2$, this equality implies $\mathsf{value}(l)\circ\mathsf{pr}_2(e) \sim \mathsf{value}(m)$, as desired.

Conversely, suppose we have $e : \mathsf{isdefined}(l) \leftrightarrow \mathsf{isdefined}(m)$ and a homotopy $v : \mathsf{value}(l)\circ\mathsf{pr}_2(e) \sim \mathsf{value}(m)$. By the characterisation of the identity type of $\sum$-types:

$$l = m \simeq \sum_{e':\mathsf{isdefined}(l)=\mathsf{isdefined}(m)} \mathsf{transport}(e', \mathsf{value}(l)) = \mathsf{value}(m).$$

By propositional extensionality, we obtain $e' : \mathsf{isdefined}(l) = \mathsf{isdefined}(m)$ from $e$. From $e'$ we get an equivalence $\mathsf{idtoeqv}(e') : \mathsf{isdefined}(l) \simeq \mathsf{isdefined}(m)$. Furthermore, using path induction on $e'$, one can prove that

$$\mathsf{transport}(e', \mathsf{value}(l)) = \mathsf{value}(l)\circ(\mathsf{idtoeqv}(e'))^{-1}. \tag{$*$}$$

Using function extensionality and our homotopy $v$ we have

$$\mathsf{value}(m) = \mathsf{value}(l)\circ\mathsf{pr}_2(e)$$

and we are to prove that

$$\mathsf{value}(m) = \mathsf{transport}(e', \mathsf{value}(l)).$$

Thus, by $(*)$ is suffices to establish that

$$(\mathsf{idtoeqv}(e'))^{-1} = \mathsf{pr}_2(e).$$

But the domain and codomain of these maps are propositions, so this is immediate. ∎

## 4.1 The lifting monad

The lifting carries a monad structure that is most easily described as a Kleisli triple. The unit $\eta_X : X \to \mathcal{L}(X)$ is given by $\eta_X(x) \equiv (1, \hat{x})$, where $\hat{x}$ denotes the function $\star \mapsto x$. Given $f : X \to \mathcal{L}(Y)$, the Kleisli extension $f^\# : \mathcal{L}(X) \to \mathcal{L}(Y)$ is defined by:

$$f^\#(P,\varphi) \equiv \left(\sum_{p:P} \mathsf{isdefined}(f(\varphi(p))), \psi\right),$$
$$\text{where } \psi(p,d) \equiv \mathsf{value}(f(\varphi(p)))(d).$$

[1] In fact, there is a type equivalence; one can prove this using univalence and a generalised structure identity principle, c.f. http://www.cs.bham.ac.uk/~mhe/agda-new/LiftingIdentityViaSIP.html

**Theorem 4.4.** *The above constructions yield a monad structure on $\mathcal{L}(X)$, i.e. the Kleisli laws hold (pointwise):*

(i) $\eta_X^\# \sim \mathrm{id}_{\mathcal{L}(X)}$;

(ii) $f^\# \circ \eta_X \sim f$ *for any* $f : X \to \mathcal{L}(Y)$;

(iii) $g^\# \circ f^\# \sim (g^\# \circ f)^\#$ *for any* $f : X \to \mathcal{L}(Y)$ *and* $g : Y \to \mathcal{L}(Z)$.

*Proof.* Using Lemma 4.3 the proofs become easy. Item (iii) is essentially the equivalence between $\sum_{a:A} \sum_{b:B(a)} C(a,b)$ and $\sum_{(a,b):\sum_{a:A} B(a)} C(a,b)$. ∎

## 4.2 The lifting as a dcpo with bottom

**Theorem 4.5.** *If $X$ is a set, then so is its lifting $\mathcal{L}\,X$.*

*Proof.* As in the proof of Lemma 4.3, we have:

$$l = m \simeq \sum_{e:\mathsf{isdefined}(l)=\mathsf{isdefined}(m)} \mathsf{transport}(e, \mathsf{value}(l)) = \mathsf{value}(m).$$

Since $X$ is a set, the type $\mathsf{transport}(e, \mathsf{value}(l)) = \mathsf{value}(m)$ is a proposition. So, if we can prove that $\mathsf{isdefined}(l) = \mathsf{isdefined}(m)$ is a proposition, then the right hand side is a proposition indexed sum of propositions, which is again a proposition.

So let us prove that if $P$ and $Q$ are propositions, then so is $P = Q$. At first glance, it might seem like one needs univalence (for propositions) to prove this, but in fact propositional extensionality suffices. By [KECA17, Lemma 3.11], it suffices to give a weakly constant (i.e. any two of its values are equal) endomap on $P = Q$ (one may safely replace the type $X$ in [KECA17, Lemma 3.11] with $\mathsf{Prop}$, c.f. our formalisation). But the composition

$$(P = Q) \to (P \leftrightarrow Q) \xrightarrow{\text{PropExt}} (P = Q)$$

is weakly constant, because $P \leftrightarrow Q$ is a proposition, so this finishes the proof. ∎

**Theorem 4.6.** *If $X$ is a set, then $\mathcal{L}\,X$ is a dcpo with bottom with the following order:*

$$l \sqsubseteq m \equiv \mathsf{isdefined}(l) \to l = m.$$

*Proof.* First of all, we should prove that $\mathcal{L}(X)$ is a poset with the specified order. In particular, $\sqsubseteq$ should be proposition valued. If $X$ is a set, then $\mathsf{isdefined}(l) \to l = m$ is a function type into a proposition and therefore a proposition itself.

Reflexivity and transitivity of $\sqsubseteq$ are easily verified. Moreover, $\sqsubseteq$ is seen to be antisymmetric by virtue of Lemma 4.3.

The bottom element of $\mathcal{L}(X)$ is given by $(\mathsf{0}, !_X)$, where $!_X$ is the unique function from $\mathsf{0} \to X$.

The construction of the least upper bound of a directed family is the most challenging part of the proof. If $u : I \to \mathcal{L}\,X$ is a directed family in $\mathcal{L}\,X$, then its least upper bound is

$$\left( \left\| \sum_{i:I} \mathsf{isdefined}(u_i) \right\|, \varphi \right) : \mathcal{L}\,X,$$

where $\varphi$ is the factorisation of the function

$$\psi : \sum_{i:I} \mathsf{isdefined}(u_i) \to X, \quad (i, d) \mapsto \mathsf{value}(u_i)d$$

through $\|\sum_{i:I} \mathsf{isdefined}(u_i)\|$. That this factorisation exists follows from [KECA17, Theorem 5.4], as $X$ is a set and $\psi$ is weakly constant (viz. any two of its values are equal), which we prove now.

Suppose $(i_0, d_0)$ and $(i_1, d_1)$ are terms of type $\sum_{i:I} \mathsf{isdefined}(u_i)$. We want to prove that $\psi(i_0, d_0) = \psi(i_1, d_1)$. Since $X$ is a set, this is a proposition, so we may use directedness of $u$ to obtain $k : I$ with $u_{i_0}, u_{i_1} \sqsubseteq u_k$. From $d_0$ and $d_1$, we now get $d, d' : \mathsf{isdefined}(k)$ satisfying:

$$\psi(i_0, d_0) = \psi(k, d) \quad \text{and} \quad \psi(i_1, d_1) = \psi(k, d')$$

but $d = d'$ since $\mathsf{isdefined}(k)$ is a proposition, so $\psi(i_0, d_0) = \psi(i_1, d_1)$, as desired. ∎

**Theorem 4.7.** *Let $X$ and $Y$ be sets and $f : X \to \mathcal{L}(Y)$ any function. The Kleisli extension $f^\# : \mathcal{L}(X) \to \mathcal{L}(Y)$ is a morphism of dcpos.*

*Proof.* Let $v$ be the upper bound of a directed family $u : I \to \mathcal{L}(X)$ in $\mathcal{L}(X)$. Proving that $f^\#$ is monotone is quite easy. By monotonicity, $f^\#(v)$ is an upper bound for the family $f^\# \circ u$. We are left to prove that it is the least. Suppose that $l : \mathcal{L}(Y)$ is another upper bound for the family $f^\# \circ u$, i.e. $l \sqsupseteq f^\#(u_i)$ for every $i : I$. We must show that $f^\#(v) \sqsubseteq l$. To this end, assume we have $q : \mathsf{isdefined}(f^\#(v))$. We must prove that $f^\#(v) = l$.

From $q$, we obtain $p : \mathsf{isdefined}(v)$ by definition of $f^\#$. By our construction of suprema in $\mathcal{L}(X)$ and the fact that $f^\#(v) = l$ is a proposition, we may in fact assume that we have an element $i : I$ and $d_i : \mathsf{isdefined}(u_i)$. But $l \sqsupseteq f^\#(u_i)$, so using $d_i$ we get the equality $l = f^\#(u_i)$. Since $v$ is an upper bound for $u$, the term $d_i$ also yields $u_i = v$. In particular, $l = f^\#(u_i) = f^\#(v)$, as desired. ∎

*Remark* 4.8. Finally, one could define the functor $\mathcal{L}$ from the Kleisli triple by putting $\mathcal{L}(f) = (\eta_Y \circ f)^\#$ for any $f : X \to Y$. However, it is equivalent and easier to directly define $\mathcal{L}(f)$ by postcomposition: $\mathcal{L}(f)(P, \varphi) \equiv (P, f \circ \varphi)$.

## 4.3 The lifting as a free construction

**Definition 4.9.** Let us call a poset subsingleton complete if it has least upper bounds of proposition (subsingleton) indexed families.

**Definition 4.10.** Let $\mathsf{Subcpo}_\perp$ be the category with as objects subsingleton complete posets with a least element and maps that preserve proposition indexed suprema as morphisms.

**Theorem 4.11.** *If $X$ is a set, then $(\mathcal{L}(X), \sqsubseteq)$ is a subsingleton complete poset with a least element.*

*Proof.* We have already seen that $(\mathcal{L}(X), \sqsubseteq)$ is a poset with a least element. Let $u : P \to \mathcal{L}(X)$ be a proposition indexed family. It is straightforward to verify that the least upper bound of $u$ is given by

$$\left( \sum_{p:P} \mathsf{isdefined}(u_p), (p, d) \mapsto \mathsf{value}(u_p)(d) \right) : \mathcal{L}(X). \qquad ∎$$

**Theorem 4.12.** *The lifting functor $\mathcal{L}$ is left adjoint to the forgetful functor $U : \mathsf{Subcpo}_\perp \to \mathsf{Set}$.*

*Proof.* Let $X$ be a set and $D$ a subsingleton complete poset with a least element. Suppose $f : X \to U(D)$ is a map of sets. We must construct a unique morphism $\widetilde{f} : \mathcal{L}(X) \to D$ in $\mathsf{Subcpo}_\perp$ such that $U(\widetilde{f})(\eta_X(x)) = f(x)$.

We do so by defining $\widetilde{f} : \mathcal{L}(X) \to D$ as

$$(P, \varphi) \mapsto \bigsqcup_{p:P} f(\varphi(p)).$$

Note that this is well-defined, because $D$ is assumed to be subsingleton complete. It is not hard to show that $\widetilde{f}$ preserves proposition indexed suprema. Hence, it remains to prove that $\widetilde{f}$ is the unique such morphism. To this end, suppose $g : \mathcal{L}(X) \to D$ is a morphism in $\mathsf{Subcpo}_\perp$ with $g(\eta_X(x)) = f(x)$. We must show that $g(l) = \widetilde{f}(l)$ for every $l : \mathcal{L}(X)$. Suppose that we have $(P, \varphi) : \mathcal{L}(X)$. Note that

$$(P, \varphi) = \bigsqcup_{p:P} \eta_X(\varphi(p)),$$

so that we have the following chain of equalities

$$\begin{aligned} g(P, \varphi) &= g\left(\bigsqcup_{p:P} \eta_X(\varphi(p))\right) \\ &= \bigsqcup_{p:P} g(\eta_X(\varphi(p))) && \text{(as } g \text{ preserves proposition indexed suprema)} \\ &= \bigsqcup_{p:P} f(\varphi(p)) && \text{(by assumption on } g) \\ &\equiv \widetilde{f}(P, \varphi), \end{aligned}$$

as desired. ∎

# 5 PCF and its operational semantics

To avoid dealing with (free and bound) variables, we opt to work in the combinatory version of PCF. It is inductively defined as follows.

**Definition 5.1.** The *PCF types* are inductively defined as:

(i) $\iota$ is a type, the *base type*;

(ii) for every two types $\sigma$ and $\tau$, there is a *function type* $\sigma \Rightarrow \tau$.

As usual, $\Rightarrow$ will be right associative, so we write $\sigma \Rightarrow \tau \Rightarrow \rho$ for $\sigma \Rightarrow (\tau \Rightarrow \rho)$.

**Definition 5.2.** The *PCF terms of PCF type $\sigma$* are inductively defined as:

(i) there is a PCF term $\mathsf{zero}$ of PCF type $\iota$;

(ii) there is a PCF term $\mathsf{succ}$ of PCF type $\iota \Rightarrow \iota$;

(iii) there is a PCF term $\mathsf{pred}$ of PCF type $\iota \Rightarrow \iota$;

(iv) there is a PCF term $\mathsf{ifz}$ of PCF type $\iota \Rightarrow \iota \Rightarrow \iota \Rightarrow \iota$;

(v) for any types $\sigma$ and $\tau$, there is a PCF term $\mathsf{k}_{\sigma,\tau}$ of PCF type $\sigma \Rightarrow \tau \Rightarrow \sigma$;

(vi) for any types $\sigma, \tau$ and $\rho$, there is a PCF term $\mathsf{s}_{\sigma,\tau,\rho}$ of PCF type $(\sigma \Rightarrow \tau \Rightarrow \rho) \Rightarrow (\sigma \Rightarrow \tau) \Rightarrow \sigma \Rightarrow \rho$;

(vii) for any type $\sigma$, there is a PCF term $\mathsf{fix}_\sigma$ of PCF type $(\sigma \Rightarrow \sigma) \Rightarrow \sigma$;

(viii) if $s$ is a PCF term of PCF type $\sigma \Rightarrow \tau$ and $t$ is a PCF term of PCF type $\sigma$, then $(st)$ is a PCF term of PCF type $\tau$.

We will often drop the parenthesis in (viii) as well as the PCF type subscripts in (v) – (vii). Finally, we employ the convention that the parenthesis associate to the left, i.e. we write $rst$ for $(rs)t$.

**Definition 5.3.** For any $n : \mathbb{N}$, let us write $\underline{n}$ for the $n$th PCF numeral, defined inductively as:

$$\underline{0} = \mathsf{zero}; \quad \underline{n+1} = \mathsf{succ}\,\underline{n}.$$

To define the smallstep operational semantics of PCF, we first define the following inductive type.

**Definition 5.4.** Define the *smallstep pre-relation* $\widetilde{\rhd}$ of type

$$\prod_{\sigma:\text{PCF types}} \text{PCF terms of type } \sigma \to \text{PCF terms of type } \sigma \to \mathsf{Type}$$

as the inductive type generated by (we leave out the PCF type argument of $\widetilde{\rhd}$ and use infix notation for $\widetilde{\rhd}$):

(i) $\mathsf{pred}\,\mathsf{zero} \mathrel{\widetilde{\rhd}} \mathsf{zero}$;

(ii) for any natural number $n$, $\mathsf{pred}\,\underline{n+1} \mathrel{\widetilde{\rhd}} \underline{n}$;

(iii) for any PCF terms $s$ and $t$ of PCF type $\iota$, $\mathsf{ifz}\,t\,s\,\mathsf{zero} \mathrel{\widetilde{\rhd}} t$;

(iv) for any PCF terms $s$ and $t$ of PCF type $\iota$ and natural number $n$, $\mathsf{ifz}\,s\,t\,(\underline{n+1}) \mathrel{\widetilde{\rhd}} t$;

(v) for any PCF term $s$ of PCF type $\sigma$ and PCF term $t$ of PCF type $\tau$, $\mathsf{k}ts \mathrel{\widetilde{\rhd}} t$;

(vi) for any PCF term $f$ of PCF type $\sigma \Rightarrow \tau \Rightarrow \rho$, PCF term $g$ of PCF type $\sigma \Rightarrow \tau$ and PCF term $t$ of PCF type $\sigma$, $\mathsf{s}fgt \mathrel{\widetilde{\rhd}} (ft)gt$;

(vii) for any PCF term $f$ of PCF type $\sigma \Rightarrow \sigma$, $\mathsf{fix}\,f \mathrel{\widetilde{\rhd}} f(\mathsf{fix}\,f)$;

(viii) for any PCF terms $f$ and $g$ of PCF type $\sigma \Rightarrow \tau$ and any PCF term $t$ of PCF type $\sigma$, if $f \mathrel{\widetilde{\rhd}} g$, then $ft \mathrel{\widetilde{\rhd}} gt$;

(ix) for any PCF terms $s$ and $t$ of PCF type $\iota$, if $s \mathrel{\widetilde{\rhd}} t$, then $\mathsf{succ}\,s \mathrel{\widetilde{\rhd}} \mathsf{succ}\,t$;

(x) for any PCF terms $s$ and $t$ of PCF type $\iota$, if $s \mathrel{\widetilde{\rhd}} t$, then $\mathsf{pred}\,s \mathrel{\widetilde{\rhd}} \mathsf{pred}\,t$;

(xi) for any PCF terms $r', r, s$ and $t$ of PCF type $\iota$, if $r \mathrel{\widetilde{\rhd}} r'$, then $\mathsf{ifz}\,s\,t\,r \mathrel{\widetilde{\rhd}} \mathsf{ifz}\,s\,t\,r'$.

It seems rather hard to prove that $s \mathrel{\widetilde{\triangleright}} t$ is a proposition for every suitable PCF terms $s$ and $t$. However, conceptually, $s \mathrel{\widetilde{\triangleright}} t$ should be a proposition, as (by inspection of the definition), there is at most one way by which we obtained $s \mathrel{\widetilde{\triangleright}} t$. Moreover, for technical reasons that will become apparent later, we really want $\widetilde{\triangleright}$ to be propostion-valued.

We solve the problem by defining the *smallstep relation* $\triangleright$ as the propositional truncation of $\widetilde{\triangleright}$, i.e. $s \triangleright t \equiv \|s \mathrel{\widetilde{\triangleright}} t\|$.

**Definition 5.5.** Let $R : X \to X \to \mathsf{Prop}$ be a relation on a type $X$. We could define the reflexive transitive closure $R_*$ of $R$ as an inductive type, generated by three constructors:

$$\begin{aligned} \mathsf{extend} &: \prod_{x,y:X} xRy \to xR_*y; \\ \mathsf{refl} &: \prod_{x:X} xR_*x; \\ \mathsf{trans} &: \prod_{x,y,z:X} xR_*y \to yR_*z \to xR_*z. \end{aligned}$$

But $R_*$ is not necessarily proposition valued. However, one can show that the relation $R^*$ defined as $xR^*y \equiv \|xR_*y\|$ is. Moreover, it is the least reflexive, transitive, proposition valued relation that extends $R$. Therefore, we define $R^*$ to be the *reflexive transitive closure* of $R$.

The above properties (viii)–(xi) of $\triangleright$ reflect onto $\triangleright^*$ as the following lemma shows.

**Lemma 5.6.** *Let $r', r, s$ and $t$ be PCF terms of type $\iota$. If $r' \triangleright^* r$, then*

(i) $\mathsf{succ}\, r' \triangleright^* \mathsf{succ}\, r$;

(ii) $\mathsf{pred}\, r' \triangleright^* \mathsf{pred}\, r$;

(iii) $\mathsf{ifz}\, s\, t\, r' \triangleright^* \mathsf{ifz}\, s\, t\, r$.

*Moreover, if $f$ and $g$ are PCF terms of type $\sigma \Rightarrow \tau$ and $f \triangleright^* g$, then $ft \triangleright^* gt$ for any PCF term $t$ of type $\sigma$.*

*Proof.* We only prove (i), the rest is similar. Suppose $r' \triangleright^* r'$. Since $\mathsf{succ}\, r' \triangleright^* \mathsf{succ}\, r$ is a proposition, we may assume that we actually have a term $p$ of type $r' \triangleright_* r'$. Now we can perform induction on $p$. The cases were $p$ is formed using $\mathsf{refl}$ or $\mathsf{trans}$ are easy. If $p$ is formed by $\mathsf{extend}$, then we get a term of type $r \triangleright r' \equiv \|r \mathrel{\widetilde{\triangleright}} r'\|$. Again, as we are proving a proposition, we may suppose the existence of a term of type $r \mathrel{\widetilde{\triangleright}} r'$. By (ix) of Definition 5.4, we then get $\mathsf{succ}\, r' \mathrel{\widetilde{\triangleright}} \mathsf{succ}\, r$. This in turn yields, $\mathsf{succ}\, r' \triangleright \mathsf{succ}\, r$ and finally we use $\mathsf{extend}$ to get the desired $\mathsf{succ}\, r' \triangleright^* \mathsf{succ}\, r$. ∎

# 6 The Scott model of PCF using the lifting monad

We have already introduced the operational semantics of PCF. Next, we wish to give another sort of semantics for PCF, known as denotational semantics. The idea is to assign some mathematical structure to each PCF type. The PCF terms will then be interpreted as elements of the structure.

Moreover, we want the denotational semantics to be in sync with the operational semantics, as expressed by soundness and computational adequacy in the next section.

**Definition 6.1.** Inductively assign to each PCF type $\sigma$ a dcpo with bottom as follows:

(i) $[\![\iota]\!] = \mathcal{L}(\mathbb{N})$;

(ii) $[\![\sigma \Rightarrow \tau]\!] = [\![\tau]\!]^{[\![\sigma]\!]}$.

Recall that if $D$ and $E$ are dcpos with bottom, then $E^D$ is the dcpo with bottom of dcpo morphisms from $D$ to $E$, with pointwise ordering and pointwise least upper bounds.

Next, we interpret terms as elements of these dcpos with bottom.

**Definition 6.2.** Define for each PCF term $t$ of PCF type $\sigma$ a term $[\![t]\!]$ of type $[\![\sigma]\!]$, by the following inductive clauses:

(i) $[\![\mathsf{zero}]\!] \equiv \eta 0$;

(ii) $[\![\mathsf{succ}]\!] \equiv \mathcal{L}(s)$, where $s : \mathbb{N} \to \mathbb{N}$ is the successor function;

(iii) $[\![\mathsf{pred}]\!] \equiv \mathcal{L}(p)$, where $p : \mathbb{N} \to \mathbb{N}$ is the predecessor function;

(iv) $[\![\mathsf{ifz}]\!] : [\![\iota \Rightarrow \iota \Rightarrow \iota \Rightarrow \iota]\!]$ is defined using the Kleisli extension as: $\lambda x, y.(\chi_{x,y})^{\#}$, where

$$\chi_{x,y}(n) \equiv \begin{cases} x & \text{if } n = 0; \\ y & \text{else.} \end{cases}$$

(v) $[\![\mathsf{k}]\!] \equiv \lambda x, y.x$;

(vi) $[\![\mathsf{s}]\!] \equiv \lambda f, g, x.(f(x))(g(x))$;

(vii) $[\![\mathsf{fix}]\!] \equiv \mu$, where $\mu$ is the least fixed point operator from Theorem 2.12.

*Remark* 6.3. Of course, there are some things to be proved here. Namely, $[\![\mathsf{succ}]\!], [\![\mathsf{pred}]\!], \ldots$ all need to be dcpo morphisms. In the case of $[\![\mathsf{succ}]\!]$ and $[\![\mathsf{pred}]\!]$, we simply appeal to Theorem 4.7 and Remark 4.8. For $[\![\mathsf{fix}]\!]$, this is Theorem 2.12. The continuity of $[\![\mathsf{k}]\!], [\![\mathsf{s}]\!]$ and $[\![\mathsf{ifz}]\!]$ can be verified directly, as done in the formalisation. It is however, unenlightning and tedious, so we omit the details here.

As a first result about our denotational semantics, we show that the PCF numerals have a canonical interpretation in the denotational semantics.

**Lemma 6.4.** *For every natural number $n$, we have $[\![\underline{n}]\!] = \eta(n)$.*

*Proof.* We proceed by induction on $n$. The $n \equiv 0$ case is by definition of $[\![\underline{0}]\!]$. Suppose $[\![\underline{m}]\!] = \eta(m)$ for a natural number $m$. Then,

$$\begin{aligned} [\![\underline{m+1}]\!] &= [\![\mathsf{succ}]\!]([\![\underline{m}]\!]) \\ &= \mathcal{L}(s)(\eta(m)) && \text{(by induction hypothesis)} \\ &= \eta(m+1) && \text{(by definition of the lift functor),} \end{aligned}$$

as desired. ∎

# 7 Soundness and computational adequacy

**Theorem 7.1** (Soundness)**.** *Let $s$ and $t$ be any PCF term of PCF type $\sigma$. If $s \triangleright^* t$, then $[\![s]\!] = [\![t]\!]$.*

*Proof.* By induction on the derivation of $s \triangleright^* t$, using the Kleisli monad laws. For example, one step is to prove that

$$[\![\mathsf{ifz}\, s\, t\, \underline{n+1}]\!] = [\![t]\!].$$

This may be proved by the following chain of equalities:

$$\begin{aligned}
[\![\mathsf{ifz}\, s\, t\, \underline{n+1}]\!] &= [\![\mathsf{ifz}\, s\, t]\!]([\![\underline{n+1}]\!]) \\
&= [\![\mathsf{ifz}\, s\, t]\!](\eta(n+1)) && \text{(by Lemma 6.4)} \\
&= (\chi_{[\![s]\!],[\![t]\!]})^{\#}(\eta(n+1)) && \text{(by definition of } [\![\mathsf{ifz}]\!]) \\
&= \chi_{[\![s]\!],[\![t]\!]}(n+1) && \text{(by Theorem 4.4(ii))} \\
&= [\![t]\!].
\end{aligned}$$

■

Ideally, we would like a converse to soundness. However, this is not possible, as for example, $[\![\mathsf{k}\,\mathsf{zero}]\!] = [\![\mathsf{k}\,(\mathsf{succ}\,(\mathsf{pred}\,\mathsf{zero}))]\!]$, but neither $\mathsf{k}\,\mathsf{zero} \triangleright^* \mathsf{k}\,(\mathsf{succ}\,(\mathsf{pred}\,\mathsf{zero}))$ nor $\mathsf{k}\,(\mathsf{succ}\,(\mathsf{pred}\,\mathsf{zero})) \triangleright^* \mathsf{k}\,\mathsf{zero}$ holds. We do, however, have the following.

**Theorem 7.2** (Computational adequacy)**.** *Let $t$ be a PCF term of PCF type $\iota$. Then,*

$$\prod_{p:\mathsf{isdefined}([\![t]\!])} t \triangleright^* \underline{\mathsf{value}([\![t]\!])(p)}.$$

We will not prove computational adequacy directly, as, unlike soundness, it does not allow for a straightforward proof by induction. Instead, we obtain the result as a direct corollary of Lemma 7.9. The rest of the section will be devoted to a particular logical relation that is the key to proving computational adequacy.

**Definition 7.3.** For every PCF type $\sigma$, define a relation

$$R_\sigma : \text{PCF terms of type } \sigma \to [\![\sigma]\!] \to \mathsf{Prop}$$

by induction on $\sigma$:

(i) $tR_\iota d \equiv \prod_{p:\mathsf{isdefined}(d)} t \triangleright^* \underline{\mathsf{value}(d)(p)}$;

(ii) $sR_{\tau\Rightarrow\rho} f \equiv \prod_{t:\text{PCF terms of type } \tau} \prod_{d:[\![\tau]\!]} (tR_\tau d \to stR_\rho f(d))$.

We will sometimes omit the type subscript $\sigma$ in $R_\sigma$.

**Lemma 7.4.** *Let $s$ and $t$ be PCF terms of type $\sigma$ and let $d$ be an element of $[\![\sigma]\!]$. If $s \triangleright^* t$ and $tR_\sigma d$, then $sR_\sigma d$.*

*Proof.* By induction on $\sigma$, making use of the last part of Lemma 5.6. ■

**Lemma 7.5.** *For $t$ equal to $\mathsf{zero}, \mathsf{succ}, \mathsf{pred}, \mathsf{ifz}, \mathsf{k}$ or $\mathsf{s}$, we have: $tR[\![t]\!]$.*

*Proof.* By the previous lemma and Lemma 5.6(i)–(iii). ■

Next, we wish to extend the previous lemma the case where $t \equiv \mathsf{fix}_\sigma$ for any PCF type $\sigma$. This is slightly more complicated and we will need two intermediate lemmas. Only the second has a non-trivial proof.

**Lemma 7.6.** *Let $\sigma$ be a PCF type and let $\bot$ be the least element of $[\![\sigma]\!]$. Then, $tR_\sigma\bot$ for any PCF term $t$ of type $\sigma$.*

*Proof.* By induction on $\sigma$. For the base type, this holds vacuously. For function types, it follows by induction hypothesis and the pointwise ordering. ■

**Lemma 7.7.** *The logical relation is closed under directed suprema. That is, for every PCF term $t$ of type $\sigma$ and every directed family $d : I \to [\![\sigma]\!]$, if $tR_\sigma d_i$ for every $i : I$, then $tR_\sigma \sqcup_{i:I} d_i$.*

*Proof.* This proof is somewhat different from the classical proof, so we spell out the details. We prove the lemma by induction on $\sigma$.

The case when $\sigma$ is a function type is easy, because least upper bounds are calculated pointwise and so it reduces to an application of the induction hypothesis. We concentrate on the case when $\sigma \equiv \iota$ instead.

Recall that $\sqcup_{i:I} d_i$ is given by $(\|\sum_{i:I} \mathsf{isdefined}(d_i)\|, \varphi)$, where $\varphi$ is the factorisation of

$$\sum_{i:I} \mathsf{isdefined}(d_i) \to \mathcal{L}\,\mathbb{N}, \quad (i, p_i) \mapsto \mathsf{value}(d_i)(p_i)$$

through $\|\sum_{i:I} \mathsf{isdefined}(d_i)\|$. Let us write $d_\infty$ for the $\sqcup_{i:I} d_i$.

We are tasked with proving that $t \triangleright^* \underline{\varphi(p)}$ for every $p : \mathsf{isdefined}(d_\infty)$. So assume $p : \mathsf{isdefined}(d_\infty) = \|\sum_{i:I} \mathsf{isdefined}(d_i)\|$. Since we are trying to prove a proposition (as $\triangleright^*$ is proposition valued), we may actually assume that we have $(i_0, p_{i_0}) : \sum_{i:I} \mathsf{isdefined}(d_i)$.

The rest of the proof is easy. By definition of $\varphi$ we have: $\varphi(p) = \mathsf{value}(d_{i_0})(p_{i_0})$ and by assumption we know that $t \triangleright^* \underline{\mathsf{value}(d_{i_0})(p_{i_0})}$, so we are done. ■

**Lemma 7.8.** *For every PCF type $\sigma$, we have $\mathsf{fix} R_{(\sigma\Rightarrow\sigma)\Rightarrow\sigma} [\![\mathsf{fix}]\!]$.*

*Proof.* Let $t$ be a PCF term of type $\sigma \Rightarrow \sigma$ and let $f : [\![\sigma \Rightarrow \sigma]\!]$ such that $tR_{\sigma\Rightarrow\sigma} f$. We are to prove that $\mathsf{fix}\, tR_\sigma \mu(f)$.

By definition of $\mu$ and the previous lemma, it suffices to prove that $\mathsf{fix}\, tR_\sigma f^n(\bot)$ where $\bot$ is the least element of $[\![\sigma]\!]$ for every natural number $n$. We do so by induction on $n$.

The base case is an application of Lemma 7.6.

Now suppose that $\mathsf{fix}\, tR_\sigma f^m(\bot)$. Then, using $tR_{\sigma\Rightarrow\sigma} f$, we find: $t(\mathsf{fix}\, t)R_\sigma f(f^m(\bot))$. Hence, by Lemma 7.4, we obtain the desired $\mathsf{fix}\, tR_\sigma f^{m+1}(\bot)$, completing our proof by induction. ■

**Lemma 7.9** (Main Lemma)**.** *For every PCF term $t$ of type $\sigma$, we have $tR_\sigma [\![t]\!]$.*

*Proof.* The proof is by induction on $t$. The base cases are taken care of by Lemma 7.5 and the previous lemma. For the inductive step, suppose $s$ is a PCF term of type $\sigma \Rightarrow \tau$. By induction hypothesis, $stR_\tau [\![st]\!]$ for every PCF term $t$ of type $\sigma$, but $[\![st]\!] \equiv [\![s]\!][\![t]\!]$, so we are done. ■

Computational adequacy is now a direct corollary of the Main Lemma.

### 7.1 Using computational adequacy to compute

Classically, every PCF program of type $\iota$ either terminates (i.e. reduces to a numeral) or it does not. From a constructive point of view, we wait for a program to terminate, with no a priori knowledge of termination. The waiting could be indefinite. Less naively, we could limit the number of computation steps to avoid indefinite waiting, with an obvious shortcoming: how many steps are enough? Instead, one could use computational adequacy to compute as follows.

Let $\sigma$ be a PCF type. A *functional of type* $\sigma$ is an element of $[\![\sigma]\!]$. By induction on PCF types, we define when a functional is said to be *total*:

(i) a functional $i$ of type $\iota$ is total if $i = [\![\underline{n}]\!]$ for some natural number $n$;

(ii) a functional $f$ of type $\sigma \Rightarrow \tau$ is total if it maps total functionals to total functionals, viz. $f(d)$ is a total functional of type $\tau$ for every total functional $d$ of type $\sigma$.

Now, let $s$ be a PCF term of type $(\cdots(\sigma_1 \Rightarrow \sigma_2) \Rightarrow \cdots \Rightarrow \sigma_n) \Rightarrow \iota$. If we can prove that $[\![s]\!]$ is total, then computational adequacy allows us to conclude that for all total inputs $[\![t_1]\!] : [\![\sigma_1]\!], \ldots, [\![t_n]\!] : [\![\sigma_n]\!]$, the term $s(t_1, \ldots, t_n)$ reduces to the numeral representing $[\![s]\!]([\![t_1]\!], \ldots, [\![t_n]\!])$. Thus, the semantic proof of totality plays the role of "enough steps".

## 8 Characterising the propositions arising from PCF terms of the base type

Every PCF term $t$ of base type $\iota$ gives rise to a proposition via the Scott model, namely $\mathsf{isdefined}([\![t]\!])$. In this section, we will show that these propositions are all semidecidable.

A consequence of soundness and computational adequacy is the following.

**Theorem 8.1.** *Let $t$ be a PCF term of type $\iota$. We have the following logical equivalences*

$$\mathsf{isdefined}([\![t]\!]) \longleftrightarrow \sum_{n:\mathbb{N}} t \triangleright^* \underline{n} \longleftrightarrow \left\| \sum_{n:\mathbb{N}} t \triangleright^* \underline{n} \right\|.$$

*Proof.* We start by proving the first logical equivalence. The second then follows from the fact that $\mathsf{isdefined}([\![t]\!])$ is a proposition.

Suppose $p$ is of type $\mathsf{isdefined}([\![t]\!])$. By computational adequacy, we find that $t \triangleright^* \underline{\mathsf{value}([\![t]\!])(p)}$, so we are done.

Conversely, suppose that we are given a natural number $n$ such that $t \triangleright^* \underline{n}$. Soundness and Lemma 6.4 then yield $[\![t]\!] = \eta(n)$. Now $\star : \mathsf{isdefined}(\eta(n))$, so we may transport along the equality to get an element of $\mathsf{isdefined}([\![t]\!])$. ∎

In order to characterise the propositions arising from PCF terms of base type as semidecidable, we will prove that $t \triangleright^* \underline{n}$ is semidecidable for every PCF term $t$ of type $\iota$ and natural number $n$. To do so, we first develop some general machinery in the next two sections.

### 8.1 Decidability of the reflexive transitive closure of a relation

**Definition 8.2.** A *relation on* $X$ is a term of type $X \to X \to \mathsf{Prop}$.

**Definition 8.3.** Let $R$ be a relation on a type $X$. We wish to define the $k$-step reflexive transitive closure of $R$. As in Definition 5.5, we want this to be proposition valued again. Therefore, we proceed as follows. For any natural number $k$, define $R_k : X \to X \to \mathcal{U}$ by induction on $k$:

(i) $xR_0y \equiv x = y$;

(ii) $xR_{k+1}z \equiv \sum_{y:X} xRy \times yR_kz$.

The *$k$-step reflexive transitive closure $R^k$ of $R$* is now defined as the relation on $X$ given by $xR^ky \equiv \|xR_ky\|$.

We wish to prove that $xR^*y$ if and only if $\left\|\sum_{k:\mathbb{N}} xR^ky\right\|$. The following lemma is the first step towards that.

**Lemma 8.4.** *Let $R$ be a relation on $X$. Recall the untruncated reflexive transitive closure $R_*$ from Definition 5.5. We have a logical equivalence for every $x, y$ in $X$:*

$$xR_*y \longleftrightarrow \sum_{k:\mathbb{N}} xR_ky.$$

*Proof.* Define $R' : X \to X \to \mathcal{U}$ inductively by:

$$\begin{aligned} \mathsf{refl}' &: \prod_{x:X} xR'x; \\ \mathsf{left} &: \prod_{xyz:X} xRy \to yR'z \to xR'z. \end{aligned}$$

It is not hard to verify that $R'$ is reflexive, transitive and that it extends $R$. Using this, one shows that $xR'y$ and $xR_*y$ are logically equivalent for every $x, y : X$. Now one easily proves $\prod_{k:\mathbb{N}}(xR_ky \to xR'y)$ by induction on $k$. This yields $(\sum_{k:\mathbb{N}} xR_ky) \to xR'y$. The converse is also easily established. Thus, $xR'y$ and $\sum_{k:\mathbb{N}} xR_ky$ are logically equivalent, finishing the proof. ∎

The next lemma extends the previous to the propositional truncations.

**Lemma 8.5.** *Let $R$ be a relation on $X$. For every $x, y : X$, we have a logical equivalence:*

$$xR^*y \longleftrightarrow \left\|\sum_{k:\mathbb{N}} xR^ky\right\|.$$

*Proof.* Let $x$ and $y$ be in $X$. By the previous lemma and functoriality of propositional truncation, we have

$$xR^*y \equiv \|xR_*y\| \to \left\|\sum_{k:\mathbb{N}} xR_ky\right\| \to \left\|\sum_{k:\mathbb{N}} \|xR_ky\|\right\| \equiv \left\|\sum_{k:\mathbb{N}} xR^ky\right\|.$$

Conversely, observe that by the previous lemma, $xR_ky \to xR_*y$ for every natural number $k$. Hence, by functoriality of propositional truncation, $xR^ky \to xR^*y$ for every natural number $k$. Thus,

$$\left(\sum_{k:\mathbb{N}} xR^ky\right) \to xR^*y.$$

But $xR^*y$ is a proposition, so $\left\|\sum_{k:\mathbb{N}} xR^ky\right\| \to xR^*y$, as desired. ∎

**Definition 8.6.** A relation $R$ on $X$ is said to be *single-valued* if for every $x, y, z : X$ with $xRy$ and $xRz$ we have $y = z$.

**Definition 8.7.** A relation $R$ on $X$ is said to be *decidable* if the type $xRy$ is decidable for every $x$ and $y$ in $X$.

**Definition 8.8.** A relation $R$ on $X$ is said to be *right-decidable* if $\sum_{y:X} xRy$ is decidable for every $x : X$.

**Lemma 8.9.** *Let $X$ be a type. If $X$ is decidable, then so is $\|X\|$.*

*Proof.* Suppose that $X$ is decidable. Then there are two cases to consider. Either we have $x : X$ or $\neg X$. If we have $x : X$, then obviously we have $|x| : \|X\|$.

So suppose that $\neg X$. We claim that $\neg\|X\|$. Assuming $\|X\|$, we must find a term of type $\mathsf{0}$. But $\mathsf{0}$ is a proposition, so we may actually assume that we have $x : X$. Using $\neg X$, we then obtain $\mathsf{0}$, as desired. ∎

**Theorem 8.10.** *Let $R$ be relation on a type $X$. If*

(i) *$X$ has decidable equality;*

(ii) *$R$ is single-valued;*

(iii) *$R$ is right-decidable;*

*then, the $k$-step reflexive transitive closure of $R$ is decidable for every natural number $k$.*

*Proof.* Suppose $X$ and $R$ satisfy conditions (i) – (iii). By Lemma 8.9, it suffices to prove that the untruncated version of $R^k$, that is $R_k$, is decidable by induction on $k$.

For the base case, let $x$ and $y$ be elements of $X$. We need to decide $xR_0y$. By definition this means deciding $x = y$, which we can, since $X$ is assumed to have decidable equality.

Now suppose $x$ and $z$ are elements of $X$ and that $aR_kb$ is decidable for every $a, b : X$. We need to show that $xR_{k+1}z$ is decidable. By definition this means that we must prove

$$\sum_{y:X} xRy \times yR_kz \tag{$*$}$$

to be decidable. By (iii), we can decide $\sum_{y:X} xRy$. Obviously, if we have $\neg\sum_{y:X} xRy$, then $\neg(*)$. So assume that we have $y : X$ such that $xRy$. By induction hypothesis, $yR_kz$ is decidable. If we have $yR_kz$, then we get $(*)$. So suppose that $\neg yR_kz$. We claim that $\neg(*)$. For suppose $(*)$, then we obtain $y' : X$ with $xRy'$ and $y'R_kz$. But $R$ is single-valued, so $y = y'$ and hence, $yR_kz$, contradicting our assumption. ∎

By Lemma 8.5, $s{\triangleright}^*t$ is logically equivalent to $\left\|\sum_{k:\mathbb{N}} s \triangleright^k t\right\|$ and $s{\triangleright}^k t$ is decidable if it satisfies the assumptions of Theorem 8.10. Assumptions (ii) and (iii) can be verified by inspection of the smallstep operational semantics once (i) has been proved.

The next section is devoted to proving (i). We do so by proving a more general result.

## 8.2 Indexed W-types with decidable equality

Indexed W-types are a generalisation of W-types. One may consult [Uni13, Section 5.3] for an explanation of regular W-types. We will see that the PCF terms form a natural example of an indexed W-type. In this section we will prove that the PCF terms have decidable equality by proving a more general result for indexed W-types. This result seems to have been first established by Jasper Hugunin, who first reported on it in a post on the Homotopy Type Theory mailing list. Theorem 8.20 and its proof are simplifications of his code, which may be found at `https://github.com/jashug/IWTypes/blob/master/FiberProperties.v`.

Before we proceed, we record some useful, albeit trivial to prove, lemmas.

**Lemma 8.11.** *Let $X$ and $Y$ be logically equivalent types. The type $X$ is decidable if and only if $Y$ is decidable.*

**Definition 8.12.** A type $R$ is called a *retract* of a type $X$ if there are maps $s : R \to X$ (the *section*) and $r : X \to R$ (the *retraction*) such that the diagram

$$\begin{array}{ccccc} R & \xrightarrow{s} & X & \xrightarrow{r} & R \\ & \searrow & & \nearrow_{\mathrm{id}_X} & \end{array}$$

commutes pointwise.

**Lemma 8.13.** *Let $R$ be a retract of $X$. If $X$ has decidable equality, then so does $R$.*

*Proof.* Let $r : R \to X$ and $s : X \to R$ be respectively the retraction and section establishing $R$ as a retract of $X$. Let $a, b : R$. Since $X$ has decidable equality, we can consider two cases: $r(a) = r(b)$ and $\neg(r(a) = r(b))$.

In the first case, we find $a = s(r(a)) = s(r(b)) = b$. In the second case, we immediately see that $\neg(a = b)$. This finishes the proof. ∎

### 8.2.1 Indexed W-types

**Definition 8.14.** Let $A$ and $I$ be types and let $B$ be a type family over $A$. Suppose we have $t : A \to I$ and $s : \sum_{a:A} B(a) \to I$. The *indexed* $\mathsf{W}$*-type* $\mathsf{W}_{s,t}$ *specified by* $s$ *and* $t$ is the inductive type family over $I$ generated by the following constructor:

$$\mathsf{indexedsup} : \prod_{a:A} \big(B(a) \to \mathsf{W}_{s,t}(s(a,b))\big) \to \mathsf{W}_{s,t}(t(a)).$$

We have the following induction principle for indexed W-types. If $E : \prod_{i:I}(\mathsf{W}_{s,t}(i) \to \mathcal{U})$, then to prove $\prod_{i:I} \prod_{w:\mathsf{W}_{s,t}(i)} E(i, w)$, it suffices to show that for any $a : A$ and $f : B(a) \to \mathsf{W}_{s,t}(s(a, b))$ satisfying $E(s(a, b), f(b))$ for every $b : B(a)$ (the *induction hypothesis*), we have a term of type $E(t(a), \mathsf{indexedsup}(a, f))$.

Just as with regular W-types, one can think of indexed W-types as encoding a particular class of inductive types. In this interpretation, $A$ encodes the constructors of the inductive type, whereas $B$ encodes the arity of each constructor. However, each constructor has a "sort" given by $t(a) : I$. Given a constructor $a : A$ and a label of an argument $b : B(a)$, the sort of this argument is given by $s(a, b)$.

**Example 8.15.** In this example, we will show that a fragment of the PCF terms can be encoded as an indexed W-type. One could extend the encoding to capture all PCF terms, but we do not spell out the tedious details here, as a fragment suffices to get the idea across.

The type family $\mathsf{T}$ is inductively defined as:

(i) $\mathsf{zero}$ is a term of type $\iota$;

(ii) $\mathsf{succ}$ is a term of type $\iota \Rightarrow \iota$;

(iii) for every PCF type $\sigma$ and $\tau$, we have a term $\mathsf{app}_{\sigma,\tau}$ of type $(\sigma \Rightarrow \tau) \Rightarrow \sigma \Rightarrow \tau$.

We can encode $\mathsf{T}$ as an indexed $\mathsf{W}$-type. Let us write $\mathsf{2}$ for $\mathsf{1}+\mathsf{1}$ and $0_{\mathsf{2}}$ and $1_{\mathsf{2}}$ for its elements. Take $I$ to be the type of PCF types and put $A \equiv \mathsf{2} + (I \times I)$. Define $B$ by

$$B(\mathsf{inl}(0_{\mathsf{2}})) \equiv B(\mathsf{inl}(1_{\mathsf{2}})) \equiv \mathsf{0} \quad \text{and} \quad B(\mathsf{inr}(\sigma,\tau)) \equiv \mathsf{2}.$$

Finally, define $t$ by

$$t(\mathsf{inl}(0_{\mathsf{2}})) \equiv \iota; \quad t(\mathsf{inl}(1_{\mathsf{2}})) \equiv \iota \Rightarrow \iota; \quad \text{and} \quad t(\mathsf{inr}(\sigma,\tau)) \equiv \tau$$

and $s$ by

$$s(\mathsf{inr}(\sigma,\tau), 0_{\mathsf{2}}) \equiv \sigma \Rightarrow \tau; \quad \text{and} \quad s(\mathsf{inr}(\sigma,\tau), 1_{\mathsf{2}}) \equiv \sigma;$$

on the other elements $s$ is defined as the unique function from $\mathsf{0}$.

One can check that given a PCF type $\sigma : I$, there is a type equivalence $T(\sigma) \simeq W_{s,t}(\sigma)$.

### 8.2.2 Indexed W-types with decidable equality

We wish to isolate some conditions on the parameters of an indexed $\mathsf{W}$-type that are sufficient to conclude that an indexed $\mathsf{W}$-type has decidable equality. We first need a few definitions before we can state the theorem.

**Definition 8.16.** A type $X$ will be called $\prod$-*compact*[2] when every type family $Y$ over $X$ satisfies: if $Y(x)$ is decidable for every $x : X$, then so is the dependent product $\prod_{x:X} Y(x)$.

**Example 8.17.** The empty type $\mathsf{0}$ is vacuously $\prod$-compact. The unit type $\mathsf{1}$ is also easily seen to be $\prod$-compact. There are interesting examples of infinite types that are $\prod$-compact, such as the one-point compactification of the natural numbers[2], but these will not be of interest to us here.

**Lemma 8.18.** *The $\prod$-compact types are closed under binary coproducts.*

*Proof.* Let $X$ and $Y$ be $\prod$-compact types. Suppose $F$ is a type family over $X + Y$ such that $F(z)$ is decidable for every $z : X + Y$. We must show that $\prod_{z:X+Y} F(z)$ is decidable.

Define $F_X : X \to \mathcal{U}$ by $F_X(x) \equiv F(\mathsf{inl}(x))$. Similarly, define $F_Y : Y \to \mathcal{U}$ as $F_Y(y) \equiv F(\mathsf{inr}(y))$. By our assumption on $F$, the types $F_X(x)$ and $F_Y(y)$ are decidable for every $x : X$ and $y : Y$. Hence, since $X$ and $Y$ are assumed to be $\prod$-compact, the dependent products $\prod_{x:X} F_X(x)$ and $\prod_{y:Y} F_Y(y)$ are decidable.

Finally, $\prod_{z:X+Y} F(z)$ is logically equivalent to $\prod_{x:X} F_X(x) \times \prod_{y:Y} F_Y(y)$. Since the product of two decidable types is again decidable, an application of Lemma 8.11 now finishes the proof. ∎

**Definition 8.19** (Definition 2.4.2 in [Uni13])**.** Let $f : X \to Y$ be a map. The *fiber of $f$ over a point $y : Y$* is

$$\mathsf{fib}_f(y) \equiv \sum_{x:X} (f(x) = y).$$

We wish to prove the following theorem.

**Theorem 8.20** (Jasper Hugunin)**.** *Let $A$ and $I$ be types and $B$ a type family over $I$. Suppose $t : A \to I$ and $s : \sum_{a:A} B(a) \to I$. If $B(a)$ is $\prod$-compact for every $a : A$ and the fiber of $t$ over $i$ has decidable equality for every $i : I$, then $\mathsf{W}_{s,t}(i)$ also has decidable equality for every $i : I$.*

[2] c.f. http://www.cs.bham.ac.uk/~mhe/agda-new/WeaklyCompactTypes.html

A corollary that is perhaps more readily useable is the following.

**Corollary 8.21.** *Let $A$ and $I$ be types and $B$ a type family over $I$. Suppose $t : A \to I$ and $s : \sum_{a:A} B(a) \to I$. If $A$ has decidable equality, $B(a)$ is $\prod$-compact for every $a : A$ and $I$ is a set, then $\mathsf{W}_{s,t}(i)$ has decidable equality for every $i : I$.*

*Proof.* Suppose that $A$ has decidable equality and $I$ is a set. We are to show that the fiber of $t$ over $i$ has decidable equality for every $i : I$. Let $i : I$ be arbitrary. Suppose we have $(a, p)$ and $(a', p')$ in the fiber of $t$ over $i$. Since $A$ has decidable equality, we can decide whether $a$ and $a'$ are equal or not. If they are not, then certainly $(a, p) \neq (a', p')$. If they are, then we claim that the dependent pairs $(a, p)$ and $(a', p')$ are also equal. If $e : a = a'$ is the supposed equality, then it suffices to show that $\mathsf{transport}^{\lambda x:A.t(x)=i}(e, p) = p'$, but both these terms are paths in $I$ and $I$ is a set, so they must be equal. ∎

For the remainder of this section, let us fix types $A$ and $I$, a type family $B$ over $A$ and maps $t : A \to I$ and $s : \sum_{a:A} B(a) \to I$.

We will not prove Theorem 8.20 directly. The statement makes it impossible to assume two elements $u, v : \mathsf{W}_{s,t}(i)$ and proceed by induction on *both* $u$ and $v$. Instead, we will state and prove a more general result that is amenable to a proof by induction. But first, we need more general lemmas and some definitions.

**Lemma 8.22.** *Let $X$ be a type and let $Y$ be a type family over it. If $X$ is a set, then the right pair function is injective, in the following sense: if $(x, y) = (x, y')$ as terms of $\sum_{a:X} Y(a)$, then $y = y'$.*

*Proof.* Suppose $X$ is a set, $x : X$, $y, y' : Y(x)$ and $e : (x, y) = (x, y')$. From $e$, we obtain $e_1 : x = x$ and $e_2 : \mathsf{transport}^Y(e_1, y) = y'$. Since $X$ is a set, we must have that $e_1 = \mathsf{refl}_x$, so that from $e_2$ we obtain a term of type $y \equiv \mathsf{transport}^Y(\mathsf{refl}_x, y) = y'$, as desired. ∎

**Definition 8.23.** For each $i : I$, define $\mathsf{sub}_i : \mathsf{W}_{s,t}(i) \to \sum_{p:\mathsf{fib}_t(t(a))} \prod_{b:B(a)} \mathsf{W}_{s,t}(s(\mathsf{pr}_1(p), b))$ by induction:

$$\mathsf{sub}_{t(a)}(\mathsf{indexedsup}(a, f)) \equiv \big(\big(a, \mathsf{refl}_{t(a)}\big), f\big).$$

For notational convenience, we will omit the subscript of $\mathsf{sub}$.

**Lemma 8.24.** *Let $a : A$ and $f, g : \prod_{b:B(a)} \mathsf{W}_{s,t}(s(a, b))$. If the fiber of $t$ over $i$ has decidable equality for every $i : I$, then $\mathsf{indexedsup}(a, f) = \mathsf{indexedsup}(a, g)$ implies $f = g$.*

*Proof.* Suppose $\mathsf{indexesup}(a, f) = \mathsf{indexedsup}(a, g)$. Then

$$\big(\big(a, \mathsf{refl}_{t(a)}\big), f\big) \equiv \mathsf{sub}(\mathsf{indexedsup}(a, f)) = \mathsf{sub}(\mathsf{indexedsup}(a, g)) \equiv \big(\big(a, \mathsf{refl}_{t(a)}\big), g\big)$$

So by Lemma 8.22 and Hedberg's Theorem, $f = g$. ∎

**Definition 8.25.** For every $i : I$, define a function $\mathsf{getfib}_i : \mathsf{W}_{s,t}(i) \to \mathsf{fib}_t(i)$ inductively by

$$\mathsf{getfib}_{t(a)}(\mathsf{indexedsup}(a, f)) \equiv (a, \mathsf{refl}_{t(a)}).$$

In future use, we will omit the subscript of $\mathsf{getfib}$.

**Lemma 8.26.** *Let $i, j : I$ with a path $p : i = j$ and $w : \mathsf{W}_{s,t}(i)$. We have the following equality:*

$$\mathsf{getfib}(\mathsf{transport}^{\mathsf{W}_{s,t}}(p, w)) = (\mathsf{pr}_1(\mathsf{getfib}(w)), (\mathsf{pr}_2(\mathsf{getfib}(w)) \cdot p)).$$

*Proof.* By path induction on $p$. ∎

We are now in position to state and prove the lemma from which Theorem 8.20 will follow.

**Lemma 8.27.** *Suppose that $B(a)$ is $\prod$-compact for every $a : A$ and that the fiber of $t$ over each $i : I$ is decidable. For any $i : I$, $u : \mathsf{W}_{s,t}(i)$, $j : I$, path $p : i = j$ and $v : \mathsf{W}_{s,t}(j)$, the type*

$$\mathsf{transportf}^{\mathsf{W}_{s,t}}(p, u) = v$$

*is decidable.*

*Proof.* Suppose $i : I$ and $u : \mathsf{W}_{s,t}(i)$. We proceed by induction on $u$. The induction hypothesis reads:

$$\prod_{b:B(a)} \prod_{j':I} \prod_{p':s(a,b)=j'} \prod_{v':\mathsf{W}_{s,t}(j')} (\mathsf{transport}^{\mathsf{W}_{s,t}}(p, f(b)) = v) \text{ is decidable .} \qquad (*)$$

Suppose we have $j : I$ with path $p : i = j$ and $v : \mathsf{W}_{s,t}(j)$. By induction, we may assume that $v \equiv \mathsf{indexedsup}(a', f')$. We are tasked to show that

$$\mathsf{transport}^{\mathsf{W}_{s,t}}(p, \mathsf{indexedsup}(a, f)) = \mathsf{indexedsup}(a', f') \qquad (\dagger)$$

is decidable, where $p : t(a) = t(a')$.

By assumption the fiber of $t$ over $t(a')$ is decidable. Hence, we can decide if $\big(a', \mathsf{refl}_{t(a')}\big)$ and $(a, p)$ are equal or not. Suppose first that the pairs are not equal. We claim that in this case $\neg(\dagger)$. For suppose we had $e : (\dagger)$, then

$$\mathsf{ap}_{\mathsf{getfib}}(e) : \mathsf{getfib}(\mathsf{transport}^{\mathsf{W}_{s,t}}(p, \mathsf{indexedsup}(a, f))) = \mathsf{getfib}(\mathsf{indexedsup}(a', f')).$$

By definition, the right hand side is $(a', \mathsf{refl}_{t(a')})$. By Lemma 8.26, the left hand side is equal to $(a, \mathsf{refl}_{t(a)} \cdot p)$ which is in turn equal to $(a, p)$, contradicting our assumption on $\big(a', \mathsf{refl}_{t(a')}\big)$ and $(a, p)$.

Now suppose that $\big(a', \mathsf{refl}_{t(a')}\big) = (a, p)$. From this, we obtain $e_1 : a' = a'$ and $e_2 : \mathsf{transport}^{\lambda x:A.\, t(x)=t(a')}\big(e_1, \mathsf{refl}_{t(a')}\big) = p$. By path induction, we may assume $e_1 \equiv \mathsf{refl}_{a'}$, so that from $e_2$ we obtain a path

$$\rho : \mathsf{refl}_{t(a')} = p.$$

Using this path, we see that the left hand side of $(\dagger)$ is equal to $\mathsf{indexedsup}(a', f)$, so we are left to show that

$$\mathsf{indexedsup}(a', f) = \mathsf{indexedsup}(a', f')$$

is decidable.

By induction hypothesis $(*)$ and $\rho$, the type $f(b) = f'(b)$ is decidable for every $b : B(a')$. Since $B(a')$ is $\prod$-compact, this implies that $\prod_{b:B(a')} f(b) = f'(b)$ is decidable.

Suppose first that $\prod_{b:B(a')} f(b) = f'(b)$. Function extensionality then yields $f = f'$, so that $\mathsf{indexedsup}(a', f) = \mathsf{indexedsup}(a', f')$.

On the other hand, suppose $\neg \prod_{b:B(a')} f(b) = f'(b)$. We claim that $\mathsf{indexedsup}(a', f)$ cannot be equal to $\mathsf{indexedsup}(a', f')$. For suppose that $\mathsf{indexedsup}(a', f) = \mathsf{indexedsup}(a', f')$. But then Lemma 8.24 yields $f = f'$, contradicting our assumption that $\neg \prod_{b:B(a)} f(b) = f'(b)$, finishing the proof. ∎

*Proof of Theorem 8.20.* Let $i : I$ and $u, v : \mathsf{W}_{s,t}(i)$. Taking $j \equiv i$ and $p \equiv \mathsf{refl}_i$ in Lemma 8.27, we see that $u = v$ is decidable, as desired. ∎

Finally, let us see how to apply Corollary 8.21 to see that the PCF terms have decidable equality. As with Example 8.15, we will only spell out the details for the fragment $\mathsf{T}$. Recall that $\mathsf{T}$ may be encoded as a $\mathsf{W}$-type, indexed by the PCF types. Using Example 8.17 and Lemma 8.18, we see that $B(a)$ is $\prod$-compact for every $a : A$. Note that $A$ has decidable equality if $I$ does. So it remains to prove that $I$, the type of PCF types, has decidable equality.

This will be another application of Corollary 8.21. Define $A' \equiv \mathsf{2}$ and define $B' : A' \to \mathcal{U}$ by $B'(\mathsf{inl}(\star)) \equiv \mathsf{0}$ and $B'(\mathsf{inr}(\star)) \equiv \mathsf{2}$. Let $t'$ and $s'$ be the unique functions to $\mathsf{1}$ from $A'$ and $\sum_{x:A'} B'(x)$, respectively. One quickly verifies that the type of PCF types is a retract of $\mathsf{W}_{s',t'}$. Observe that $B'(x)$ is $\prod$-compact for every $x : A'$ because of Example 8.17 and Lemma 8.18. Finally, $\mathsf{1}$ and $A'$ clearly have decidable equality, so by Corollary 8.21 $\mathsf{W}_{s',t'}$ has decidable equality. Thus, by Lemma 8.13, so do the PCF types.

We summarise the results of this section in the following theorem.

**Theorem 8.28.** *The propositions that arise from PCF terms of type $\iota$ are all semidecidable, as witnessed by the following logical equivalence:*

$$\mathsf{isdefined}(\llbracket t \rrbracket) \longleftrightarrow \left\| \sum_{n:\mathbb{N}} \sum_{k:\mathbb{N}} t \triangleright^k \underline{n} \right\|$$

*and the decidability of $t \triangleright^k \underline{n}$.*

# A Size matters

In this appendix, we rigorously keep track of the universe levels when working with the lifting (as a dcpo). If $\mathcal{U}$ is a type universe, then we write $\mathcal{U}^+$ for the successor type universe of $\mathcal{U}$. If $\mathcal{U}$ and $\mathcal{V}$ are two type universes, then $\mathcal{U} \sqcup \mathcal{V}$ denotes the least type universe bigger than $\mathcal{U}$ and $\mathcal{V}$.

First of all, the propositions used in $\mathcal{L}$ are taken to be in a fixed universe $\mathcal{T}$, i.e. the lifting $\mathcal{L}(X)$ of a type $X$ is defined as $\mathcal{L}(X) \equiv \sum_{P:\mathsf{Prop}_\mathcal{T}} P \to X$, where $\mathsf{Prop}_\mathcal{T} \equiv \sum_{P:\mathcal{T}} \mathsf{isaprop}(P)$.

Now observe that if $X$ is a type in a universe $\mathcal{U}$, then lifting (potentially) raises the universe level, as $\mathcal{L}(X)$ is a type in universe $\mathcal{T}^+ \sqcup \mathcal{U}$. However, if $X$ happens to be a type in $\mathcal{T}^+$, then $\mathcal{L}(X)$ also lives in $\mathcal{T}^+$. Moreover, repeated applications of $\mathcal{L}$ do not raise the universe level any further, because if $X$ is in $\mathcal{T}^+ \sqcup \mathcal{U}$, then $\mathcal{L}(X)$ is as well. These claims have been verified by Martín Escardó using Agda[3]. Moreover, one can write down the monad laws and they typecheck[4].

Let $X$ and $I$ be types in universes $\mathcal{U}$ and $\mathcal{V}$, respectively. Suppose that $u : I \to \mathcal{L}(X)$. Note that $\sum_{i:I} \mathsf{isdefined}(u_i)$ is in $\mathcal{V} \sqcup \mathcal{T}$. When considering $\mathcal{L}(X)$ as a dcpo (c.f. Theorem 4.6), we want $\sum_{i:I} \mathsf{isdefined}(u_i)$ to be in $\mathcal{T}$ again. Hence, $\mathcal{L}(X)$ is really a $\mathcal{W}$-dcpo, where $\mathcal{W}$ is a type universe below $\mathcal{T}$. In particular, $\mathcal{L}(X)$ has $\mathbb{N}$-indexed directed suprema, because the natural numbers are in the lowest type universe.

# References

[ADK17] Thorsten Altenkirch, Nils Anders Danielsson, and Nicolai Kraus. Partiality, revisited: The partiality monad as a quotient inductive-inductive type. In Javier Esparza and Andrzej S. Murawski, editors, *Foundations of Software Science and Computation Structures*, pages 534–549. Springer Berlin Heidelberg, 2017.

[3] https://www.cs.bham.ac.uk/~mhe/agda-new/LiftingSize.html
[4] https://www.cs.bham.ac.uk/~mhe/agda-new/LiftingMonad.html

[AJ94] S. Abramsky and A. Jung. Domain theory. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 3, pages 1–168. Clarendon Press, 1994. Updated online version available at: https://www.cs.bham.ac.uk/~axj/pub/papers/handy1.pdf.

[BKV09] Nick Benton, Andrew Kennedy, and Carsten Varming. Some domain theory and denotational semantics in coq. In *Lecture Notes in Computer Science*, pages 115–130. Springer Berlin Heidelberg, 2009.

[CUV17] James Chapman, Tarmo Uustalu, and Niccolò Veltri. Quotienting the delay monad by weak bisimilarity. *Mathematical Structures in Computer Science*, 29(1):67–92, 2017.

[EK17] Martín H. Escardó and Cory M. Knapp. Partial elements and recursion via dominances in univalent type theory. In Valentin Goranko and Mads Dam, editors, *26th EACSL Annual Conference on Computer Science Logic (CSL 2017)*, volume 82 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 21:1–21:16. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2017.

[KECA17] Nicolai Kraus, Martín Escardó, Thierry Coquand, and Thorsten Altenkirch. Notions of anonymous existence in Martin-Löf Type Theory. *Logical Methods in Computer Science*, 13, 2017.

[Kna18] Cory Knapp. *Partial Functions and Recursion in Univalent Type Theory*. PhD thesis, School of Computer Science, University of Birmingham, June 2018.

[Koc91] Anders Kock. Algebras for the partial map classifier monad. In *Lecture Notes in Mathematics*, pages 262–278. Springer Berlin Heidelberg, 1991.

[Str06] Thomas Streicher. *Domain-Theoretic Foundations of Functional Programming*. World Scientific, 2006.

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

[VAG+] Vladimir Voevodsky, Benedikt Ahrens, Daniel Grayson, et al. UniMath — a computer-checked library of univalent mathematics. available at https://github.com/UniMath/UniMath.