Mathematical Structures in Computer Science (2021), **31**, pp. 1270–1300
doi:10.1017/S0960129521000153

PAPER

The Scott model of PCF in univalent type theory

Tom de Jong

School of Computer Science, University of Birmingham, Birmingham, UK
Email: t.dejong@pgr.bham.ac.uk

(Received 14 November 2019; revised 23 June 2021; accepted 24 June 2021; first published online 23 July 2021)

Abstract

We develop the Scott model of the programming language PCF in univalent type theory. Moreover, we work constructively and predicatively. To account for the non-termination in PCF, we use the lifting monad (also known as the partial map classifier monad) from topos theory, which has been extended to univalent type theory by Escardó and Knapp. Our results show that lifting is a viable approach to partiality in univalent type theory. Moreover, we show that the Scott model can be constructed in a predicative and constructive setting. Other approaches to partiality either require some form of choice or quotient inductive-inductive types. We show that one can do without these extensions.

Keywords: Scott model; PCF; univalent mathematics; type theory; lifting monad; partial map classifier

1. Introduction

We develop the Scott model of the programming language PCF in constructive predicative univalent mathematics. In 1969, Scott (1993) proposed a logic (LCF) for computing with functionals. In 1977, Plotkin (1977) considered LCF as a programming language (PCF), introducing operational semantics based on Scott's logic and proving (and formulating) soundness and computational adequacy. Later, the techniques of Scott and Plotkin were extended to many other programming languages (Plotkin 1983). These developments all took place in (informal) set theory with classical logic.

Our aim is to test these techniques in Voevodsky's constructive univalent type theory (The Univalent Foundations Program 2013). Our development differs from the classical approach (Streicher 2006) in three key ways. First of all, we have situated our development in the framework of univalent mathematics. Secondly, our work takes place in a constructive meta-theory. Thirdly, we work predicatively (meaning we do not assume propositional resizing).

The essential difference (for our development) between univalent type theory on the one hand, and set theory or systems like Coq on the other, is the treatment of truth values (propositions). We will discuss manifestations of this difference in Section 1.1.3 and throughout the paper.

1.1 Technical preliminaries

In this section, we briefly explain the syntax of PCF and its computational behavior. Moreover, we recall the notion of denotational semantics and the Scott model of PCF (in a classical setting) in particular. We also mention two fundamental properties that a model of PCF should enjoy: soundness and computational adequacy. Finally, we recall the lifting monad in the context of

https://doi.org/10.1017/S0960129521000153 Published online by Cambridge University Press

univalent type theory and sketch the construction of the Scott model in constructive univalent type theory.

1.1.1 PCF

PCF (Plotkin 1977) is a typed programming language. A detailed description of PCF is given in Section 5. We briefly discuss its most characteristic features here. PCF is a typed λ-calculus with additional constants. For example, we have numerals $\underline{n}$ of base type ι corresponding to natural numbers and basic operations on them, such as a predecessor term pred and a term ifz that allows us to perform case distinction on whether an input is zero or not. The most striking feature of PCF is its fixed point combinator fix_σ for every PCF type σ. The idea is that for a term t of function type $\sigma \Rightarrow \sigma$, the term $\mathsf{fix}_\sigma\ t$ of type σ is a fixed point of t. The use of fix is that it gives us general recursion.

The operational semantics of PCF is a reduction strategy that allows us to compute in PCF. We write $s \rhd t$ for s reduces to t. We show a few examples below:

$$\mathsf{pred}\ \underline{0} \rhd \underline{0}; \quad \mathsf{pred}\ \underline{n+1} \rhd \underline{n}; \quad \mathsf{ifz}\ s\ t\ \underline{0} \rhd s; \quad \mathsf{ifz}\ s\ t\ \underline{n+1} \rhd t; \quad \mathsf{fix} f \rhd f(\ \mathsf{fix} f).$$

We see that pred indeed acts as a predecessor function and that ifz performs case distinction on whether its third argument is zero or not. The reduction rule for fix reflects that $\mathsf{fix} f$ is a fixed point f and may be seen as an unfolding (of a recursive definition).

As an example of the use of fix, consider a function g on the natural numbers given by the recursive definition: $g(0) := s$ and $g(n+1) := t(g(n))$. We can define g in PCF as $\mathsf{fix}\, G$ where $G := \lambda(f : \iota \Rightarrow \iota).\lambda(x : \iota).\ \mathsf{ifz}\ s\ (t(\ \mathsf{pred}\ x))\ x$. Having general recursion also introduces non-termination, as for example, the successor function on naturals has no fixed point.

Instead of the formulation by Plotkin (1977), which features variables and λ-abstraction, we revert in Section 5 to the original, combinatory, formulation of the terms of LCF by Scott (1993) in order to simplify the technical development.

1.1.2 Models of PCF

We have seen that the operational semantics give meaning to the PCF terms by specifying computational behavior. Another way to give meaning to the PCF terms is through denotational semantics, that is, by giving a model of PCF. A model of PCF assigns to every PCF type σ some mathematical structure $[\![\sigma]\!]$ and to every PCF term t of type σ an element $[\![t]\!]$ of $[\![\sigma]\!]$.

1.1.2.1 Soundness and computational adequacy. Soundness and computational adequacy are important properties that a model of PCF should have.

Soundness states that if a PCF term s computes to t (according to the operational semantics), then their interpretations are equal in the model (symbolically, $[\![s]\!] = [\![t]\!]$).

Computational adequacy is completeness at the base type ι. It says that for every term t of type ι and every natural number n, if $[\![t]\!] = [\![\underline{n}]\!]$, then t computes to $\underline{n}$.

1.1.2.2 The Scott model, classically. To model PCF and its non-termination, Scott (1993) introduced the Scott model: a type is interpreted as a directed complete poset with a least element (or dcpo with $\bot$, for short). Concretely, PCF types are interpreted as follows.

Interpreting the base type ι. One proves that adding a least element $\bot_{\mathbb{N}}$ to the set $\mathbb{N}$ of natural numbers yields a dcpo with $\bot$, known as the *flat natural numbers*. This is then the interpretation of the base type ι. This least element $\bot_{\mathbb{N}}$ serves as the interpretation of a term of type ι that does not compute to a numeral, like $\mathsf{fix}\ \mathsf{succ}$ where succ denotes the successor map on ι.

https://doi.org/10.1017/S0960129521000153 Published online by Cambridge University Press

Interpreting function types. Function types are interpreted by considering continuous maps (i.e. monotone maps that preserve directed suprema) between two dcpos with $\bot$. Such maps can be ordered pointwise to form another dcpo with $\bot$.

A striking feature, and the crux of the Scott model, is that every continuous map has a (least) fixed point. Moreover, the assignment of a continuous map to its least fixed point is continuous. This allows us to soundly interpret the characteristic fix term of PCF.

The Scott model was proved sound and computationally adequate by Plotkin (1977). A modern presentation may be found in Streicher (2006).

1.1.2.3 Issues with constructivity. While the interpretation of function types goes through constructively, the above interpretation of the base type ι is problematic from a constructive viewpoint. Indeed, the proof that the flat natural numbers form a dcpo relies on classical reasoning in its analysis of the directed subsets: excluded middle allows us to prove that every directed subset of the flat natural numbers is exactly one of $\{\bot\}$, $\{\bot, n\}$, or $\{n\}$ for some natural number n. In fact, we can show that this reliance is in some sense essential: in Section 3, we prove that if the flat natural numbers form a dcpo, then the Limited Principle of Omniscience (LPO) holds. This principle asserts that every binary sequence is either 0 everywhere or it attains the value 1 at some point. LPO is not constructively acceptable (Bishop 1967, p. 9), it is even provably false in some varieties of constructive mathematics (Bridges and Richman 1987, pp. 3–4), and it is independent of Martin-Löf Type Theory (Escardó 2018).

1.1.3 Univalent type theory

As mentioned at the beginning of Section 1, an essential difference between univalent type theory on the one hand and set theory or systems like Coq on the other is the treatment of truth values (propositions). To illustrate this difference, consider the definition of a poset (cf. Definition 2).

Example 1. In set theory, the mathematical structure is provided by a set X and a binary relation $\leq$ on X. Moreover, this relation is required to be reflexive, transitive, and antisymmetric. Reflexivity, $\forall_{x \in X} x \leq x$ is a logical statement that is bivalent.

In type theory, if we define $\leq : X \to X \to \mathsf{Type}$, with Type some type universe, then the type encoding reflexivity, $\prod_{x:X} x \leq x$, may have more than one element. This is a fundamental difference with set theory.

In Coq, we could instead define $\leq : X \to X \to \texttt{Prop}$, where $\texttt{Prop}$ is Coq's special sort of propositions. This sort is defined such that (for instance) reflexivity, $\forall_{x:X}\, x \leq x$, is again in $\texttt{Prop}$.

The crucial difference between these approaches and the univalent approach is that in univalent type theory, we *prove* that something is a proposition (truth value). Following Voevodsky, we define a type to be a proposition (truth value, subsingleton) if it has at most one element with respect to its identity type, that is, up to propositional equality. To define posets, we then ask for a witness that the type $x \leq y$ is a proposition for every $x, y : X$. This allows us, in the presence of function extensionality (which is a consequence of the univalence axiom), to prove that reflexivity and transitivity are propositions. For example, for reflexivity, we wish to show that the type $\prod_{x:X} x \leq x$ is a proposition. So let f, g be two elements of this type. By function extensionality, it suffices to show that $f(x) = g(x)$ for every $x : X$. But the type of $f(x)$ and $g(x)$ is $x \leq x$, which is a proposition by requirement, so $f(x)$ and $g(x)$ must be (propositionally) equal, as desired. Finally, we require X to be a set: any two elements of X are equal in at most one way. This ensures, using function extensionality again, that antisymmetry is a proposition.

Sometimes, we will want to make a type into a proposition, by identifying its elements. This is achieved through the propositional truncation, a higher inductive type. For example, we will need it to define directed families (Definition 5) but also to define the reflexive transitive closure of a

https://doi.org/10.1017/S0960129521000153 Published online by Cambridge University Press

proposition-valued relation (Definition 35). We will further explain these examples in the main text. The universal property of the propositional truncation is described in Section 1.2. For more on propositions, sets, and propositional truncation in univalent type theory, see The Univalent Foundations Program (2013, Chapter 3).

1.2 Overview of results

We work in intensional Martin-Löf Type Theory with inductive types (including the empty $\mathsf{0}$, unit $\mathsf{1}$, natural numbers N, and identity types), $+$-, Σ-, and Π-types. As usual, we simply write $x = y$ for the identity type $\mathsf{Id}_X(x, y)$, use $\equiv$ for the judgmental equality, and write $\simeq$ for Voevodsky's notion of type equivalence.

We need (at least) two universes $\mathscr{U}_0$ and $\mathscr{U}_1$ closed under $+$-, Σ-, and Π-types, such that $\mathscr{U}_0$ contains $\mathsf{0}$, $\mathsf{1}$, and N, while $\mathscr{U}_1$ contains $\mathscr{U}_0$. We work predicatively, that is, we do not assume propositional resizing, so the type of propositions in $\mathscr{U}_0$, denoted by Ω, lives in the universe $\mathscr{U}_1$.

We also assume two extensionality axioms. The first is function extensionality which asserts that pointwise equal functions are equal. Given two (dependent) functions $f, g : \prod_{a:A} B(a)$, we write $f \sim g$ for the type $\prod_{a:A} f(a) = g(a)$, often called the type of homotopies between f and g. Function extensionality makes the type $f \sim g$ equivalent to the identity type $f = g$. The second is propositional extensionality which says that logically equivalent propositions are equal, that is, if P and Q are propositions, then $P \leftrightarrow Q$ implies $P = Q$. In the presence of function extensionality, this is equivalent to $(P \leftrightarrow Q) \simeq (P = Q)$.

Although we do not need the univalence axiom at any point, we remark that both extensionality axioms above follow from it. Moreover, we emphasize the importance of the idea of truncation levels, which is fundamental to univalent type theory.

Finally, we assume the existence of a single higher inductive type, the propositional truncation: given a type X in a universe $\mathscr{U}$, we assume that we have a *proposition* $\|X\|$ in $\mathscr{U}$ with a map $|-| : X \to \|X\|$ such that if P is a proposition in any universe and $f : X \to P$ is a map, then f factors through $|-|$. Diagrammatically,

$$\begin{tikzcd}
X \arrow[rr, "f"] \arrow[dr, "|-|"'] & & P \\
& \|X\| \arrow[ur, dashed, "\bar{f}"'] &
\end{tikzcd}$$

Observe that the factorization $\bar{f}$ is unique by function extensionality and the fact that P is a proposition.

Our paper can be summarized as follows:

Section 2. We introduce the theory of dcpos with $\perp$ (known as domain theory) in predicative constructive univalent type theory. We take the carriers of the dcpos to be sets (in the sense of univalent type theory) and the partial orders to be proposition-valued. Propositional truncation plays an import part in defining directedness.

Section 3. We elaborate on the issue with the classical construction of the Scott model in a constructive meta-theory (cf. the final paragraph of Section 1.1.2).

Section 4. To remedy this issue, we work instead with the lifting monad (also known as the partial map classifier monad) from topos theory (Kock 1991), which has been extended to constructive type theory by Reus and Streicher (1999) and recently to univalent type theory by Escardó and Knapp (2017) and Knapp (2018). The lifting $\mathscr{L}(X)$ of a type X is defined as $\mathscr{L}(X) :\equiv \sum_{P:\Omega} (P \to X)$, where Ω is the type of propositions in the first universe. We think of the elements (P, φ) of $\mathscr{L}(X)$ as partial elements of X: in case P holds, we get an element of X, but P may also fail to hold and then the partial element is thought of as undefined. In our constructive model, we interpret the base type of PCF as the lifting $\mathscr{L}(\mathsf{N})$ of the natural numbers.

Section 5. We define a combinatory version of PCF and its (small-step) operational semantics. We use the propositional truncation to obtain well-behaved relations in the small-step operational semantics.

Section 6. We define our constructive Scott model of PCF using the lifting monad.

Section 7. We show how the usual proofs of soundness and computational adequacy adapt to our constructive setting with propositional truncations.

Section 8. Recall that in our model the PCF type ι for natural numbers is interpreted as $\mathscr{L}(\mathsf{N})$, where N is the natural numbers type. Thus, if t is a PCF term of type ι, then we get an element $[\![t]\!] : \mathscr{L}(\mathsf{N})$. Hence, for every such term t, we have a proposition $\mathsf{pr}_1([\![t]\!]) : \Omega$. We show that such propositions are all semidecidable. This result should be contrasted with the fact that a restricted version of the lifting monad where we take a Σ-type over only semidecidable propositions is not adequate for our purposes, as we explain at the end of Section 8.

In proving our results, we take the opportunity to record some more general properties of reflexive transitive closures (Section 8.1) and indexed W-types (Section 8.2).

Section 9. We discuss the universe levels involved in our development. This is important, because we want our results to go through predicatively, that is, without propositional resizing.

Section 10. We summarize our main results and describe directions for future work.

1.3 Related work

Partiality in type theory has been the subject of recent study. We briefly discuss the different approaches.

Firstly, there are the delay monad by Capretta (2005) and its quotient by weak bisimilarity, as studied by Chapman et al. (2017). They used countable choice to prove that the quotient is again a monad. Escardó and Knapp (Escardó and Knapp 2017; Knapp 2018) showed that a weak form of countable choice is indeed necessary to prove this. However, Coquand et al. (2017, Corollary 2) have shown that countable choice cannot be proved in dependent type theory with one univalent universe and propositional truncation. Theorem 3.3 of (Coquand 2018) extends this to dependent type theory with a hierarchy of univalent universes and (some) higher inductive types. Moreover, Swan (2019*a*,b) recently showed that even the weak form of choice required is not provable in univalent type theory.

Another approach is laid out by Altenkirch et al. (2017). They postulated the existence of a particular quotient inductive-inductive type (QIIT) and showed that it satisfies the universal property of the free ω-cpo with a least element (Altenkirch et al. 2017, Theorem 5). Moreover, Altenkirch et al. showed that, assuming countable choice, their QITT coincides with the quotiented delay monad.

We stress that our approach does not need countable choice or QIITs.

Finally, Benton et al. (2009) used Capretta's delay monad to give a constructive approach to domain theory. Their approach used setoids, so that every object comes with an equivalence relation that maps must preserve. One cannot quotient these objects, because quotienting Capretta's delay monad requires (a weak form of) countable choice, as explained above. In our development, we instead use Martin-Löf's identity types as our notion of equality. Moreover, we do not make use of Coq's impredicative `Prop` universe and our treatment incorporates directed complete posets (dcpos) and not just ω-cpos.

1.4 Formalisation

All our results up to and including the proof of computational adequacy (and except for Section 3 and Remark 29) have been formalized in the proof assistant Coq using the UniMath library (Voevodsky et al. 2019) and Coq's `Inductive` types. The general results from Section 8 have also been formalized, but their direct applications to PCF, for example, 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`. Instructions for use can be found in the repository's `README.md` file. Browsable documentation for the formalization may be found at `https://tomdjong.github.io/Scott-PCF-UniMath/toc.html`. Definitions and proofs of lemmas, propositions, and theorems are labeled with their corresponding identifiers in the Coq name, for example as `pcf`, which also functions as a hyperlink to the appropriate definition in the documentation.

At present, it is not possible to verify universe levels in UniMath. Therefore, to verify the correctness of our development and our claims in Section 9 about universe levels in particular, we reformalized part of our development in Agda using Martín Escardó (2019) library. Our code is now part of the library. An HTML rendering may be found at: `https://www.cs.bham.ac.uk/~mhe/agda-new/PCFModules.html`.

2. Basic Domain Theory

We introduce basic domain theory in the setting of constructive predicative univalent mathematics. We adapt known definitions (cf. Abramsky and Jung 1994, Section 2.1 and Streicher 2006, Chapter 4) to constructive univalent type theory, paying special attention to how our definitions may involve propositional truncations.

2.1 Directed complete posets

Definition 2 (`PartialOrder`)**.** *A* poset $(X, \leq)$ *is a set* X *together with a proposition-valued binary relation* $\leq : X \to X \to \Omega$ *satisfying:*

(1) reflexivity: $\prod_{x:X} x \leq x$;
(2) antisymmetry: $\prod_{x,y:X} x \leq y \to y \leq x \to x = y$;
(3) transitivity: $\prod_{x,y,z:X} x \leq y \to y \leq z \to x \leq z$.

Remark 3. Notice that we require $\leq$ to take values in Ω, the type of propositions in $\mathscr{U}_0$, cf. Example 1. This allows us to prove (using function extensionality The Univalent Foundations Program 2013, Example 3.6.2) that reflexivity and transitivity are propositions, that is, there is at most one witness of reflexivity and transitivity. We also express this by saying that reflexivity and transitivity are properties, rather than structures. Moreover, we restrict to X being a set to ensure that antisymmetry is a property, rather than a structure.

Definition 4 (`posetmorphism`)**.** *Let* X *and* Y *be posets. A* poset morphism *from* X *to* Y *is a function between the underlying sets that preserves the order. We also say that the function is* monotone.

Definition 5 (`isdirected`)**.** *Let* $(X, \leq)$ *be a poset and* I *any type. Given a family* $u : I \to X$, *we often write* u_i *for* $u(i)$. *Such a family is called* directed *if* I *is inhabited (i.e.* $\|I\|$ *holds) and* $\prod_{i,j:I} \left\| \sum_{k:I} (u_i \leq u_k) \times (u_j \leq u_k) \right\|$.

https://doi.org/10.1017/S0960129521000153 Published online by Cambridge University Press

Remark 6. We use the propositional truncation in the definition above to ensure that being directed is a property, rather than a structure (`isaprop_isdirected`).

Firstly, we express that the type I is inhabited by requiring an element of $\|I\|$. This is different from requiring an element of I. It is akin to the difference (in set theory) between a set X such that $\exists x \in X$ holds and a pair (X, x) of a set with a chosen element $x \in X$.

Secondly, if we had used an untruncated Σ in the second clause of the definition, then we would have asked our poset to be equipped with an *operation* mapping pairs (x, y) of elements to some *specified* element greater than both x and y.

Definition 7 (`isupperbound`, `islub`, `isdirectedcomplete`)**.** *An element x of a poset X is an* upper bound *of a family $u : I \to X$ if $u_i \sqsubseteq x$ for every $i : I$. It is a* least upper bound *of u if it is an upper bound and $x \sqsubseteq y$ holds whenever y is an upper bound of u.*

A poset X is called $\mathscr{U}$-directed complete *for a type universe $\mathscr{U}$ if every directed family in X indexed by a type in $\mathscr{U}$ has a least upper bound in X, which we denote by $\bigsqcup_{i:I} u_i$. Symbolically, $\prod_{I:\mathscr{U}} \prod_{u:I\to X}\big(u \text{ is directed} \to \sum_{x:X} x \text{ is a least upper bound of } u\big)$.*

We call such a poset a $\mathscr{U}$-dcpo*. We shall often simply write dcpo, omitting reference to the type universe.*

Remark 8. Contrary to Definition 5, directed completeness is not phrased with a truncated Σ. This justifies having the least upper bound operator $\bigsqcup$. The reason for this definition of directed completeness is that least upper bounds are unique when they exist (`lubsareunique`). Moreover, the type expressing that an element is a least upper bound for a family can be shown to be a proposition using function extensionality (`isaprop_islub`). Hence, for any family u, the type of least upper bounds of u and its propositional truncation are equivalent. This observation also tells us, using function extensionality again, that the type expressing that a poset is directed complete is also a proposition (`isaprop_isdirectedcomplete`), that is, it is a property of the poset.

Remark 9. In classical mathematics, a dcpo is usually defined as a poset such that every directed *subset* has a least upper bound. We have formulated our version using *families*, because in our type-theoretic framework functions are primitive, unlike in set theory where sets are primitive and functions are encoded as particular sets. Another reason for preferring families is that we work in the absence of propositional resizing, so that we must pay attention to size and therefore only ask for least upper bounds of *small* directed subsets. This point is explained and worked out in detail in de Jong and Escardó (2021*b*, Section 5) to which we refer the interested reader. Here, we limit ourselves to saying that working with families is more direct, and that for the Scott model we will only need to consider simple N-indexed directed families anyway.

2.2 Morphisms of dcpos

Definition 10 (`isdcpomorphism`)**.** *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. That is, if $u : I \to D$ is a directed family, then $f\big(\bigsqcup_{i:I} u_i\big)$ is the least upper bound of $f \circ u : I \to E$.*

Thus, by definition, a dcpo morphism is required to be a poset morphism, that is, it must be monotone. However, as is well known in domain theory, requiring that the function is monotone is actually redundant, as the following lemma shows.

Lemma 11. *Let D and E be dcpos. If f is a function (on the underlying types) from D to E preserving least upper bounds of directed families, then f is order-preserving.*

https://doi.org/10.1017/S0960129521000153 Published online by Cambridge University Press

Proof (preservesdirectedlub_isdcpomorphism). Let $f : D \to E$ be a morphism of dcpos and suppose $x, y : D$ with $x \le y$. 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 $f(x) \le f(y)$. □

Lemma 12. *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.*

Proof (dcpomorphism_preservesdirected). Using monotonicity of f. □

Theorem 13. *Let D and E be dcpos. The morphisms from D to E form a dcpo with the pointwise order.*

Proof (dcpoofdcpomorphisms). The least upper bound of a directed family of dcpo morphisms is also given pointwise. The proof only differs from the standard proof of Streicher (2006, Theorem 4.2) in that it uses directed families, rather than subsets. One may consult the formalization for the technical details. □

2.3 Dcpos with ⊥

Definition 14 (dcpowithbottom). *A* dcpo with $\bot$ *is a dcpo D together with a least element in D.*

Theorem 15. *Let D be a dcpo and let E be a dcpo with $\bot$. Ordered pointwise, the morphisms from D to E form a dcpo with $\bot$, which we denote by E^D.*

Proof (dcpowithbottom_ofdcpomorphisms). 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 13. □

Dcpos with bottom elements are interesting because they admit least fixed points. Moreover, these least fixed points are themselves given by a continuous function.

Theorem 16. *Let D be a dcpo with $\bot$. There is a continuous function $\mu : D^D \to D$ that sends each continuous function to its least fixed point. In fact, μ satisfies:*

(1) $f(\mu(f)) = \mu(f)$ for every continuous $f : D \to D$;
(2) for every continuous $f : D \to D$ and each $d : D$, if $f(d) \le d$, then $\mu(f) \le d$.

Proof. (leastfixedpoint_isfixedpoint, leastfixedpoint_isleast). We have formalized the proof of Abramsky and Jung (1994, Theorem 2.1.19). We sketch the main construction here. For each natural number n, define $\mathsf{iter}(n) : D^D \to D$ as

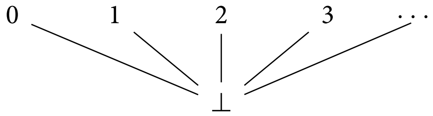
$$\mathsf{iter}(n)(f) :\equiv f^n(\bot) :\equiv \underbrace{f(f(\dots(f}_{n \text{ times}}(\bot))\dots)).$$

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 μ as the least upper bound of this directed family. Recall that least upper bounds in the exponential are given pointwise, so that $\mu(f) = \bigsqcup_{n:\mathbb{N}} f^n(\bot)$. □

https://doi.org/10.1017/S0960129521000153 Published online by Cambridge University Press

3. Constructive Issues with Partiality

In classical mathematics, a partial map from $\mathbb{N}$ to $\mathbb{N}$ can simply be seen as a 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:



Using excluded middle, a directed subset of $\mathbb{N}_\bot$ is either $\{\bot\}$, $\{n\}$ or $\{\bot, n\}$ (with n a natural number). The least upper bounds of which are easily computed as $\bot$, n and n, respectively. Thus, with excluded middle, $\mathbb{N}_\bot$ is directed complete.

One could hope that the above translates directly into constructive univalent mathematics, that is, that the poset $\mathsf{N}_\bot :\equiv (\mathsf{N} + 1, \leq_\bot)$ with $\leq_\bot$ the flat order (i.e. $\mathsf{inl}\,(\star)$ is the least element and all other elements are incomparable) is ($\mathcal{U}_0$-)directed complete (in the sense of Definition 7). However, we can prove that this implies Bishop's LPO, a constructive taboo (recall the final paragraph of Section 1.1.2), as follows.

Write 2 for the type $1 + 1$, and 0 and 1 for its inhabitants $\mathsf{inl}\,(\star)$ and $\mathsf{inr}\,(\star)$, respectively. In type theory, LPO may be formulated[1] as the following type:

$$\prod_{\alpha:\mathsf{N}\to 2}\left(\prod_{n:\mathsf{N}} \alpha(n) = 0\right) + \left(\sum_{k:\mathsf{N}} \alpha(k) = 1\right). \qquad \text{(LPO)}$$

Lemma 17. *Directed completeness of* $\mathsf{N}_\bot$ *implies* LPO.

Proof. Suppose that $\mathsf{N}_\bot$ is ($\mathcal{U}_0$-)directed complete. Let $\alpha : \mathsf{N} \to 2$ be an arbitrary binary sequence. Define the family $\beta : \mathsf{N} \to \mathsf{N}_\bot$ as

$$\beta(n) :\equiv \begin{cases} \mathsf{inl}\,(k) & \text{if } k \text{ is the least integer} \leq n \text{ such that } \alpha(k) = 1; \\ \mathsf{inr}\,(\star) & \text{else.} \end{cases}$$

Then β is directed, so by assumption, it has a supremum s in $\mathsf{N}_\bot$. By the induction principle of sum-types, we can decide whether $s = \mathsf{inl}\,(k)$ for some $k : \mathsf{N}$ or $s = \mathsf{inr}\,(\star)$. The former implies $\sum_{k:\mathsf{N}} \alpha(k) = 1$ and we claim that the latter implies $\prod_{n:\mathsf{N}} \alpha(n) = 0$. For suppose that $s = \mathsf{inr}\,(\star)$ and let $n : \mathsf{N}$. Since 2 has decidable equality, it suffices to show that $\alpha(n) \neq 1$. Assume for a contradiction that $\alpha(n) = 1$. Then $\beta(n) = \mathsf{inl}\,(k)$ for some natural number $k \leq n$. Using that s is the supremum of β yields $\mathsf{inl}\,(k) = \beta(n) \leq_\bot s = \mathsf{inr}\,(\star)$. By definition of the order, we also have the reverse inequality $\mathsf{inr}\,(\star) \leq_\bot \mathsf{inl}\,(k)$. Hence, $\mathsf{inr}\,(\star) = \mathsf{inl}\,(k)$ by antisymmetry, which is a contradiction, so $\alpha(n) \neq 1$ as desired. □

4. Partiality, Constructively

In this section, we present the lifting monad as a solution to the problem described in the previous section. Using the lifting monad in univalent type theory to deal with partiality originates with the work of Escardó and Knapp (2017), Knapp (2018) and aims to avoid countable choice.

We start by defining the lifting of a type and by characterizing its identity type. In Section 4.1, we prove that the lifting carries a monad structure, while in Section 4.2 we show that the lifting of a set is a dcpo with $\bot$. Most of the definitions and some of the results in this section can be found in Knapp (2018) or in Escardó and Knapp (2017). Exceptions are Lemma 22, Theorems 25, and 27. We note that our characterization of equality of the lifting, Lemma 22, is implicit in the fact that the order of Escardó and Knapp (2017) is antisymmetric. The order on the lifting in this paper (see Theorem 26) is different from the order presented in Escardó and Knapp (2017), Knapp (2018). The two orders are equivalent, however, as observed by in Escardó (2019, `LiftingUnivalentPrecategory`). We found the order in this paper to be more convenient.

Definition 18 (`lift`)**.** *Let X be any type. Define the* lifting of X *as:*

$$\mathscr{L}(X) :\equiv \sum_{P:\Omega} (P \to X).$$

Strictly speaking, we should have written $\mathsf{pr}_1(P) \to X$*, because elements of* Ω *are pairs of types and witnesses that these types are subsingletons. We will almost always suppress reference to these witnesses in this paper.*

Definition 19 (`liftorder_least`)**.** *For any type X, the type* $\mathscr{L}(X)$ *has a distinguished element:*

$$\bot_X :\equiv (0, \mathsf{from}\text{-}0_X) : \mathscr{L}(X),$$

where $\mathsf{from}\text{-}0_X$ *is the unique function from* 0 *to X.*

Definition 20 (`lift_embedding`)**.** *There is a canonical map* $\eta_X : X \to \mathscr{L}(X)$ *defined by:*

$$\eta_X(x) :\equiv (1, \lambda t.x).$$

Assuming LEM (i.e. $\prod_{P:\Omega}(P + \neg P)$), we can prove that the only propositions are 0 and 1, for if a proposition P holds, then it is equal (by propositional extensionality) to 1 and if it does not hold, then it is equal to 0. Hence, if we assume LEM then the two definitions above capture all of the lifting, since LEM implies:

$$\mathscr{L}(X) \equiv \left(\sum_{P:\Omega}(P \to X)\right) \simeq ((1 \to X) + (0 \to X)) \simeq (X + 1),$$

as $(1 \to X) \simeq X$ and there is a unique function from 0 to any type X. Constructively, things are more interesting, of course.

We proceed by defining meaningful projections.

Definition 21 (`isdefined`, `value`)**.** *We take* $\mathsf{isdefined} : \mathscr{L}(X) \to \Omega$ *to be the first projection. The function* $\mathsf{value} : \prod_{l:\mathscr{L}(X)} \mathsf{isdefined}(l) \to X$ *is given by:* $\mathsf{value}(P, \varphi)(p) :\equiv \varphi(p)$.

Since equality of Σ-types often requires $\mathsf{transport}$, it will be convenient to characterize the equality of $\mathscr{L}(X)$.

Lemma 22. *Let X be any type and let* $l, m : \mathscr{L}(X)$*. The following are logically equivalent*[2]

(1) $l = m$;
(2) $\sum_{e:\mathsf{isdefined}(l) \leftrightarrow \mathsf{isdefined}(m)} \mathsf{value}(l) \circ \mathsf{pr}_2(e) \sim \mathsf{value}(m)$.

Proof. (`lifteq_necc`, `lifteq_suff`).

First of all, the characterization of the identity type of Σ-types (The Univalent Foundations Program 2013, Theorem 2.7.2) yields

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

https://doi.org/10.1017/S0960129521000153 Published online by Cambridge University Press

Thus, we only have to show that the right-hand side of (†) is logically equivalent to (2) in the lemma. Suppose first that we have $e' : \mathsf{isdefined}\,(l) = \mathsf{isdefined}\,(m)$ and an equality $p : \mathsf{transport}\,(e', \mathsf{value}\,(l)) = \mathsf{value}\,(m)$. Then

$$e :\equiv \mathsf{eqtoiff}(e') : \mathsf{isdefined}\,(l) \leftrightarrow \mathsf{isdefined}\,(m).$$

Using path induction on e', we can prove that $\mathsf{value}\,(l) \circ \mathsf{pr}_2\,(e) = \mathsf{transport}\,(e', \mathsf{value}\,(l))$. Together with p, this equality implies $\mathsf{value}\,(l) \circ \mathsf{pr}_2\,(e) \sim \mathsf{value}\,(m)$, as desired.

Conversely, suppose $e : \mathsf{isdefined}\,(l) \leftrightarrow \mathsf{isdefined}\,(m)$ and $v : \mathsf{value}\,(l) \circ \mathsf{pr}_2\,(e) \sim \mathsf{value}\,(m)$. By propositional extensionality, we obtain $e' : \mathsf{isdefined}\,(l) = \mathsf{isdefined}\,(m)$ from e. From e', we can get an equivalence $\mathsf{idtoeqv}(e') : \mathsf{isdefined}\,(l) \simeq \mathsf{isdefined}\,(m)$. Furthermore, using path induction on e', one can prove that

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

Hence, it suffices to show that the right-hand side of (∗) is equal to $\mathsf{value}\,(m)$. The homotopy v yields $\mathsf{value}\,(l) \circ \mathsf{pr}_2\,(e) = \mathsf{value}\,(m)$ by function extensionality, so it suffices to prove that $(\mathsf{idtoeqv}(e'))^{-1} = \mathsf{pr}_2\,(e)$. But these are both functions with codomain $\mathsf{isdefined}\,(l)$, which is a proposition, so they are equal by function extensionality. □

4.1 The lifting monad

In this section, we prove that the lifting carries a monad structure.

This monad structure is most easily described as a Kleisli triple. The unit is given by Definition 20.

Definition 23 (`Kleisli_extension`). *Given $f : X \to \mathscr{L}(Y)$, the* Kleisli extension *$f^\# : \mathscr{L}(X) \to \mathscr{L}(Y)$ is defined by:*

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

where $\psi(p, d) :\equiv \mathsf{value}\,(f(\varphi(p)))(d)$.

Theorem 24 (Theorem 5.8 in Knapp 2018, Section 2.2 in Escardó and Knapp 2017)**.** *The above constructions yield a monad structure on $\mathscr{L}(X)$, that is, the Kleisli laws hold (pointwise):*

(1) $(\eta_X)^\# \sim \mathrm{id}_{\mathscr{L}(X)}$;
(2) $f^\# \circ \eta_X \sim f$ for any $f : X \to \mathscr{L}(Y)$;
(3) $g^\# \circ f^\# \sim (g^\# \circ f)^\#$ for any $f : X \to \mathscr{L}(Y)$ and $g : Y \to \mathscr{L}(Z)$.

Proof. (`eta_extension`, `fun_extension_after_eta`, `extension_comp`). The proofs are straightforward thanks to Lemma 22. Item (3) is essentially the associativity of Σ, that is, 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 ⊥

The goal of this section is to endow $\mathscr{L}(X)$ with a partial order that makes it into a dcpo with ⊥, provided that X is a set. We also show that the Kleisli extension from the previous section is continuous when regarded as a morphism between dcpos with ⊥.

Theorem 25. *If X is a set, then so is its lifting $\mathscr{L}(X)$.*

https://doi.org/10.1017/S0960129521000153 Published online by Cambridge University Press

Proof (`liftofhset_isaset`). As in the proof of Lemma 22, 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 Kraus et al. (2017, Lemma 3.11) (applied to the type of propositions), it suffices to give for every proposition R, a (weakly) constant (i.e. any two of its values are equal) endomap on $P = R$. But the composition

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

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

Theorem 26 (cf. Theorem 5.14 in Knapp 2018 and Theorem 1 in Escardó and Knapp 2017)**.** *If X is a set, then $\mathscr{L}\,(X)$ is a dcpo with $\bot$ with the following order:*

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

Proof (`liftdcpowithbottom`). First of all, we should prove that $\mathscr{L}\,(X)$ 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 using Lemma 22.

The $\bot$ element of $\mathscr{L}\,(X)$ is given by $\bot_X$ from Definition 19.

The construction of the least upper bound of a directed family is the most challenging part of the proof. Let $u : I \to \mathscr{L}\,(X)$ be a directed family in $\mathscr{L}\,(X)$. Consider the diagram (of solid arrows):

$$\begin{tikzcd}
\sum_{i:I} \mathsf{isdefined}\,(u_i) \arrow[rr, "\varphi:(i,d)\mapsto \mathsf{value}\,(u_i)(d)"] \arrow[dr, "|-|"'] & & X \\
& \left\| \sum_{i:I} \mathsf{isdefined}\,(u_i) \right\| \arrow[ur, dashed, "\psi"'] &
\end{tikzcd}$$

We are going to construct the dashed map ψ that makes the diagram commute and define the least upper bound of u as: $\left(\left\| \sum_{i:I} \mathsf{isdefined}\,(u_i) \right\|, \psi\right)$. Truncating the type is necessary, as $\sum_{i:I} \mathsf{isdefined}\,(u_i)$ may have more than one element if I is not a proposition. The difficulty lies in the fact that the universal property of the truncation only tells us how to define maps into *propositions*. But X is a *set*. We solve this problem using Kraus et al. (2017, Theorem 5.4), which says that every weakly constant function $f : A \to B$ to a set B factors through $\|A\|$. That f is weakly constant means that $f(a) = f(a')$ for every $a, a' : A$. So, to construct ψ, we only need to prove that the top map φ in the diagram is weakly constant. Let (i, d_i), (j, d_j) be two elements of the domain of φ. We are to prove that $\mathsf{value}\,(u_i)(d_i) = \mathsf{value}\,(u_j)(d_j)$. As X is a set, this is a proposition. Therefore, using that u is directed, we obtain $k : I$ with $u_i, u_j \sqsubseteq u_k$. But $d_i : \mathsf{isdefined}\,(u_i)$ and $d_j : \mathsf{isdefined}\,(u_j)$, so $u_i = u_k = u_j$ by definition of the order. Hence, $\varphi(i, d_i) = \mathsf{value}\,(u_i)(d_i) = \mathsf{value}\,(u_j)(d_j) = \varphi(j, d_j)$, as we wished to show. □

Theorem 27. *Let X and Y be sets and $f : X \to \mathscr{L}\,(Y)$ any function. The Kleisli extension $f^{\#} : \mathscr{L}\,(X) \to \mathscr{L}\,(Y)$ is a morphism of dcpos.*

https://doi.org/10.1017/S0960129521000153 Published online by Cambridge University Press

Proof (`Kleisli_extension_dcpo`). Let v be the least upper bound of a directed family $u : I \to \mathscr{L}(X)$ in $\mathscr{L}(X)$. 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 $l : \mathscr{L}(Y)$ is another upper bound for the family $f^{\#} \circ u$, that is, $l \sqsupseteq f^{\#}(u_i)$ 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 $\mathscr{L}(X)$ 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 28 (`liftfunctor_eq`)**.** Finally, one could define the functor $\mathscr{L}$ from the Kleisli extension and unit by putting $\mathscr{L}(f) :\equiv (\eta_Y \circ f)^{\#}$ for any $f : X \to Y$. However, it is equivalent and easier to directly define $\mathscr{L}(f)$ by post-composition: $\mathscr{L}(f)(P, \varphi) :\equiv (P, f \circ \varphi)$.

Remark 29. We remark that lifting may be regarded as a free construction, in more than one way in fact. This result should be compared to Altenkirch et al. (2017, Theorem 5), where Altenkirch et al. exhibit their QIIT as the free ω-cpo with a least element (cf. Section 1.3).

By de Jong and Escardó (2021*a*, Theorems 21 and 23), the lifting of a set X can be regarded both as the free pointed dcpo on X and as the free subsingleton complete poset on X. In our predicative setting, some care should be taken in formulating these statements. We do not go into the details here and instead refer the interested reader to de Jong and Escardó (2021*a*).

5. PCF and its Operational Semantics

This section formally defines the types and terms of PCF as well as the small-step operational semantics. It should be regarded as a formal counterpart to the informal introduction to PCF in Section 1.1.1.

To avoid dealing with free and bound variables (in the formalization), we opt to work in the combinatory version of PCF, as originally presented by Scott (1993). We note that it is possible to represent every closed λ-term in terms of combinators by a well-known technique (Hindley and Seldin 2008, Section 2C).

We inductively define combinatory PCF as follows.

Definition 30 (`type`)**.** *The* PCF types *are inductively defined as:*

(1) ι is a type, the base type*;*
(2) for every two types σ and τ, 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 31 (`term`)**.** *The* PCF terms of PCF type σ *are inductively generated by:*

$$\frac{}{\mathsf{zero}\ \textit{of type}\ \iota} \qquad \frac{}{\mathsf{succ}\ \textit{of type}\ \iota \Rightarrow \iota}$$

$$\frac{}{\mathsf{pred}\ \textit{of type}\ \iota \Rightarrow \iota} \qquad \frac{}{\mathsf{ifz}\ \textit{of type}\ \iota \Rightarrow \iota \Rightarrow \iota \Rightarrow \iota}$$

$$\frac{}{\mathsf{k}_{\sigma,\tau}\ \textit{of type}\ \sigma \Rightarrow \tau \Rightarrow \sigma} \qquad \frac{}{\mathsf{s}_{\sigma,\tau,\rho}\ \textit{of type}\ (\sigma \Rightarrow \tau \Rightarrow \rho) \Rightarrow (\sigma \Rightarrow \tau) \Rightarrow \sigma \Rightarrow \rho}$$

https://doi.org/10.1017/S0960129521000153 Published online by Cambridge University Press

$$\frac{}{\mathsf{fix}_\sigma \textit{ of type } (\sigma \Rightarrow \sigma) \Rightarrow \sigma} \qquad \frac{s \textit{ of type } \sigma \Rightarrow \tau \qquad t \textit{ of type } \tau}{(st) \textit{ of type } \tau}$$

We will often drop the parentheses in the final clause, as well as the PCF type subscripts in $\mathsf{k}_{\sigma,\tau}$, $\mathsf{s}_{\sigma,\tau,\rho}$, *and* fix_σ. *Finally, we employ the convention that the parentheses associate to the left, that is, we write rst for* $(rs)t$.

Definition 32 (numeral). *For any* $n : \mathsf{N}$, *let us write* $\underline{n}$ *for the nth PCF numeral, defined inductively as:*

$$\underline{0} :\equiv \mathsf{zero}; \quad \underline{n+1} :\equiv \mathsf{succ}\ \underline{n}.$$

To define the small-step operational semantics of PCF, we first define the following inductive type.

Definition 33 (smallstep', smallstep). *Define the* small-step pre-relation $\tilde{\triangleright}$ *of type:*

$$\prod_{\sigma:\textit{PCF types}} \textit{PCF terms of type } \sigma \to \textit{PCF terms of type } \sigma \to \mathscr{U}_0$$

as the inductive family generated by:

$$\frac{}{\mathsf{pred}\ \underline{0} \mathrel{\tilde{\triangleright}} \underline{0}} \qquad \frac{}{\mathsf{pred}\ \underline{n+1} \mathrel{\tilde{\triangleright}} \underline{n}} \qquad \frac{}{\mathsf{ifz}\ s\ t\ \underline{0} \mathrel{\tilde{\triangleright}} s} \qquad \frac{}{\mathsf{ifz}\ s\ t\ \underline{n+1} \mathrel{\tilde{\triangleright}} t}$$

$$\frac{}{\mathsf{k}st \mathrel{\tilde{\triangleright}} s} \qquad \frac{}{\mathsf{s}fgt \mathrel{\tilde{\triangleright}} ft(gt)} \qquad \frac{}{\mathsf{fix}\, f \mathrel{\tilde{\triangleright}} f(\,\mathsf{fix}\, f)} \qquad \frac{f \mathrel{\tilde{\triangleright}} g}{ft \mathrel{\tilde{\triangleright}} gt}$$

$$\frac{s \mathrel{\tilde{\triangleright}} t}{\mathsf{succ}\ s \mathrel{\tilde{\triangleright}} \mathsf{succ}\ t} \qquad \frac{s \mathrel{\tilde{\triangleright}} t}{\mathsf{pred}\ s \mathrel{\tilde{\triangleright}} \mathsf{pred}\ t} \qquad \frac{r \mathrel{\tilde{\triangleright}} r'}{\mathsf{ifz}\ s\ t\ r \mathrel{\tilde{\triangleright}} \mathsf{ifz}\ s\ t\ r'}$$

We have been unable to prove that $s \mathrel{\tilde{\triangleright}} t$ *is a proposition for every suitable PCF terms s and t. The difficulty is that one cannot perform induction on* both *s and t. However, conceptually,* $s \mathrel{\tilde{\triangleright}} t$ *should be a proposition, as (by inspection of the definition), there is at most one way by which we obtained* $s \mathrel{\tilde{\triangleright}} t$. *Moreover, for technical reasons that will become apparent later, we really want* $\tilde{\triangleright}$ *to be propostion-valued.*

We solve the problem by defining the small-step relation $\triangleright$ *as the propositional truncation of* $\tilde{\triangleright}$, *that is,* $s \triangleright t :\equiv \| s \mathrel{\tilde{\triangleright}} t \|$.

Remark 34. Benedikt Ahrens pointed out that in an impredicative framework, one could use propositional resizing and an impredicative encoding, that is, by defining $\triangleright$ as a Π-type of all suitable proposition-valued relations. This is similar to the situation in set theory, where one would define $\triangleright$ as an intersection. Specifically, say that a relation:

$$R : \prod_{\sigma:\text{PCF types}} \big(\text{PCF terms of type } \sigma \to \text{PCF terms of type } \sigma \to \Omega_{\mathscr{U}_0}\big)$$

is *suitable* if it closed under all the clauses of Definition 33, that is, $R\big(\iota, \mathsf{pred}\ \underline{0}, \underline{0}\big)$, $R\big(\iota, \mathsf{pred}\ \underline{n+1}, \underline{n}\big)$, etc., are all inhabited. We could define $s \triangleright_{\text{impred}} t :\equiv \prod_{R \text{ suitable}} R(\sigma, s, t)$. But notice the increase in universe level:

$$\triangleright_{\text{impred}} : \prod_{\sigma:\text{PCF types}} \big(\text{PCF terms of type } \sigma \to \text{PCF terms of type } \sigma \to \Omega_{\mathscr{U}_1}\big).$$

https://doi.org/10.1017/S0960129521000153 Published online by Cambridge University Press

So because of this increase, $\triangleright_{\text{impred}}$ itself is not one of the suitable relations. Therefore, $\triangleright_{\text{impred}}$ does not satisfy the appropriate universal property in being the least relation closed under the clauses in Definition 33. With propositional resizing, we could resize $\triangleright_{\text{impred}}$ to a $\mathcal{U}_0$-valued relation satisfying the appropriate universal property. The advantage of using the propositional truncation above is that it does satisfy the right universal property even without propositional resizing.

Let $R : X \to X \to \Omega$ be a relation on a type X. We might try to 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, even though R is. This is because we might add a pair (x, y) to R_* in more than one way, for example, once by an instance of extend and once by an instance of trans. Thus, we are led to the following definition.

Definition 35 (refl_trans_clos, refl_trans_clos_hrel)**.** *Let $R : X \to X \to \Omega$ be a relation on a type X. We define the* reflexive transitive closure *R^* of R by $xR^*y :\equiv \lVert xR_*y \rVert$, where R_* is as above.*

It is not hard to show that R^* is the least reflexive and transitive proposition-valued relation that extends R, so R^* satisfies the appropriate universal property (refl_trans_clos_univprop).

Some properties of $\triangleright$ reflect onto $\triangleright^*$ as the following lemma shows.

Lemma 36. *Let r', r, s, and t be PCF terms of type ι. If $r' \triangleright^* r$, then*

(1) $\mathsf{succ}\, r' \triangleright^* \mathsf{succ}\, r$;
(2) $\mathsf{pred}\, r' \triangleright^* \mathsf{pred}\, r$;
(3) $\mathsf{ifz}\, s\, t\, r' \triangleright^* \mathsf{ifz}\, s\, t\, r$.

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 σ.

Proof (succ_refltrans_smallstep, pred_refltrans_smallstep, ifz_refltrans_smallstep, app_refltrans_smallstep). We only prove (1) 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 refl or trans are easy. If p is formed by extend, then we get a term of type $r \triangleright r' \equiv \lVert r \widetilde{\triangleright} r' \rVert$. Again, as we are proving a proposition, we may suppose the existence of a term of type $r \widetilde{\triangleright} r'$. By Definition 33, we then get $\mathsf{succ}\, r' \widetilde{\triangleright}\, \mathsf{succ}\, r$. This in turn yields, $\mathsf{succ}\, r' \triangleright \mathsf{succ}\, r$ and finally we use extend to get the desired $\mathsf{succ}\, r' \triangleright^* \mathsf{succ}\, r$. □

6. The Scott Model of PCF Using the Lifting Monad

Next, we wish to give a denotational semantics for PCF, namely the Scott model, as explained in Definition 1.1.2. We recall that the idea is to assign some mathematical structure to each PCF type. The PCF terms are then interpreted as elements of the structure.

https://doi.org/10.1017/S0960129521000153 Published online by Cambridge University Press

Definition 37 (`denotational_semantics_type`)**.** *Inductively assign to each PCF type σ a dcpo with $\bot$ as follows:*

(1) $[\![\iota]\!] :\equiv \mathscr{L}(\mathsf{N})$;
(2) $[\![\sigma \Rightarrow \tau]\!] :\equiv [\![\tau]\!]^{[\![\sigma]\!]}$.

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

Next, we interpret PCF terms as elements of these dcpos with $\bot$, for which we will need that $\mathscr{L}$ is a monad (with unit η) and (in particular) a functor (recall Theorem 24 and Remark 28).

Definition 38 (`denotational_semantics_terms`)**.** *Define for each PCF term t of PCF type σ a term $[\![t]\!]$ of type $[\![\sigma]\!]$, by the following inductive clauses:*

(1) $[\![\mathsf{zero}]\!] :\equiv \eta(0)$;
(2) $[\![\mathsf{succ}]\!] :\equiv \mathscr{L}(s)$, *where* $s : \mathsf{N} \to \mathsf{N}$ *is the successor function;*
(3) $[\![\mathsf{pred}]\!] :\equiv \mathscr{L}(p)$, *where* $p : \mathsf{N} \to \mathsf{N}$ *is the predecessor function;*
(4) $[\![\mathsf{ifz}]\!] : [\![\iota \Rightarrow \iota \Rightarrow \iota \Rightarrow \iota]\!]$ *is defined using the Kleisli extension as:* $\lambda x, y.\left(\chi_{x,y}\right)^{\#}$, *where*

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

(5) $[\![\mathsf{k}]\!] :\equiv \lambda x, y.x$;
(6) $[\![\mathsf{s}]\!] :\equiv \lambda f, g, x.(f(x))(g(x))$;
(7) $[\![\mathsf{fix}]\!] :\equiv \mu$, *where μ is the least fixed point operator from Theorem 16.*

Remark 39. Of course, there are some things to be proved here. Namely, $[\![\mathsf{succ}]\!], [\![\mathsf{pred}]\!], \ldots, [\![\mathsf{fix}]\!]$ all need to be dcpo morphisms. In the case of $[\![\mathsf{succ}]\!]$ and $[\![\mathsf{pred}]\!]$, we simply appeal to Theorem 27 and Remark 28. For $[\![\mathsf{fix}]\!]$, this is Theorem 16. The continuity of $[\![\mathsf{k}]\!]$, $[\![\mathsf{s}]\!]$, and $[\![\mathsf{ifz}]\!]$ can be verified directly, as done in the formalization (`k_dcpo`, `s_dcpo`, `lifted_ifz`). 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.

Proposition 40. *For every natural number n, we have* $[\![\underline{n}]\!] = \eta(n)$.

Proof (`denotational_semantics_numerals`). 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,

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

as desired. □

7. Soundness and Computational Adequacy

In this section, we show that the denotational semantics and the operational semantics defined above are "in sync," as expressed by soundness and computational adequacy (cf. Section 1.1.2).

https://doi.org/10.1017/S0960129521000153 Published online by Cambridge University Press

Theorem 41 (Soundness). *Let s and t be any PCF terms of PCF type σ. If $s \rhd^* t$, then $[\![s]\!] = [\![t]\!]$.*

Proof (soundness). Since the carriers of dcpos are defined to be sets, the type $[\![s]\!] = [\![t]\!]$ is a proposition. Therefore, we can use induction on the derivation of $s \rhd^* t$. We use the Kleisli monad laws in proving some of the cases. 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:

$$\begin{aligned}
[\![\mathsf{ifz}\; s\; t\; \underline{n+1}]\!] &= [\![\mathsf{ifz}\; s\; t]\!]([\![\underline{n+1}]\!]) \\
&= [\![\mathsf{ifz}\; s\; t]\!](\eta(n+1)) && \text{(by Proposition 40)} \\
&= (\chi_{[\![s]\!],[\![t]\!]})^{\#}(\eta(n+1)) && \text{(by definition of } [\![\mathsf{ifz}]\!]) \\
&= \chi_{[\![s]\!],[\![t]\!]}(n+1) && \text{(by Theorem 24)} \\
&= [\![t]\!].
\end{aligned}$$

□

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 $\mathsf{k}\;\mathsf{zero} \rhd^* \mathsf{k}\,(\,\mathsf{succ}\,(\,\mathsf{pred}\;\mathsf{zero}))$ nor $\mathsf{k}\,(\,\mathsf{succ}\,(\,\mathsf{pred}\;\mathsf{zero})) \rhd^* \mathsf{k}\;\mathsf{zero}$ holds. We do, however, have the following.

Theorem 42 (Computational adequacy). *Let t be a PCF term of PCF type ι. Then,*

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

Equivalently, for every $n : \mathsf{N}$, it holds that $[\![t]\!] = [\![\underline{n}]\!]$ implies $t \rhd^* \underline{n}$.

We do not prove computational adequacy directly, as, unlike soundness, it does not allow for a straightforward proof by induction. Instead, we use the standard technique of logical relations (Streicher 2006, Chapter 7) and obtain the result as a direct corollary of Lemma 49.

Definition 43 (adequacy_relation). *For every PCF type σ, define a relation*

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

by induction on σ:

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

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

We sometimes omit the type subscript σ in R_σ.

Lemma 44. *Let s and t be PCF terms of type σ and let d be an element of $[\![\sigma]\!]$. If $s \rhd^* t$ and $tR_\sigma d$, then $sR_\sigma d$.*

Proof (adequacy_step). By induction on σ, making use of the last part of Lemma 36. □

Lemma 45. *For t equal to* zero, succ, pred, ifz, k *or* s, *we have $tR[\![t]\!]$.*

Proof (adequacy_zero, adequacy_succ, adequacy_pred, adequacy_ifz, adequacy_k, adequacy_s). By the previous lemma and Lemma 36. □

https://doi.org/10.1017/S0960129521000153 Published online by Cambridge University Press

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

Lemma 46. *Let σ be a PCF type and let $\perp$ be the least element of $[\![\sigma]\!]$. Then, $tR_\sigma \perp$ for any PCF term t of type σ.*

Proof (adequacy_bottom). By induction on σ. For the base type, this holds vacuously. For function types, it follows by induction hypothesis and the pointwise ordering. □

Lemma 47. *The logical relation is closed under directed suprema. That is, for every PCF term t of type σ and every directed family $d : I \to [\![\sigma]\!]$, if $tR_\sigma d_i$ for every $i : I$, then $tR_\sigma \bigsqcup_{i:I} d_i$.*

Proof (adequacy_lubs). This proof is somewhat different from the classical proof, so we spell out the details. We prove the lemma by induction on σ.

The case when σ 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 $\bigsqcup_{i:I} d_i$ is given by $\left(\left\| \sum_{i:I} \mathsf{isdefined}\,(d_i) \right\|, \varphi\right)$, where φ is the factorization of

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

through $\left\| \sum_{i:I} \mathsf{isdefined}\,(d_i) \right\|$.

We are tasked with proving that $t \rhd^* \underline{\varphi(p)}$ for every $p : \mathsf{isdefined}\left(\bigsqcup_{i:I} d_i\right)$. So assume that $p : \left\| \sum_{i:I} \mathsf{isdefined}\,(d_i) \right\|$. Since we are trying to prove a proposition (as $\rhd^*$ is proposition-valued), we may actually assume that we have $(j, p_j) : \sum_{i:I} \mathsf{isdefined}\,(d_i)$. By definition of φ, we have: $\varphi(p) = \mathsf{value}\,(d_j)(p_j)$ and by assumption we know that $t \rhd^* \underline{\mathsf{value}\,(d_j)(p_j)}$, so we are done. □

Lemma 48. *For every PCF type σ, we have $\mathsf{fix}_\sigma\ R_{(\sigma \Rightarrow \sigma) \Rightarrow \sigma} [\![\mathsf{fix}_\sigma]\!]$.*

Proof (adequacy_fixp). 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 μ and the previous lemma, it suffices to prove that $\mathsf{fix}\ tR_\sigma f^n(\perp)$ where $\perp$ 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 46.

Now suppose that $\mathsf{fix}\ tR_\sigma f^m(\perp)$. Then, using $tR_{\sigma \Rightarrow \sigma} f$, we find: $t(\,\mathsf{fix}\ t)R_\sigma f(f^m(\perp))$. Hence, by Lemma 44, we obtain the desired $\mathsf{fix}\ tR_\sigma f^{m+1}(\perp)$, completing our proof by induction. □

Lemma 49 (Fundamental Theorem). *For every PCF term t of type σ, we have $tR_\sigma [\![t]\!]$.*

Proof (adequacy_allterms). The proof is by induction on t. The base cases are taken care of by Lemma 45 and the previous lemma. For the inductive step, suppose t is a PCF term of type $\sigma \Rightarrow \tau$. By induction hypothesis, $tsR_\tau [\![ts]\!]$ for every PCF term s of type σ, but $[\![ts]\!] \equiv [\![t]\!][\![s]\!]$, so we are done. □

Computational adequacy is now a direct corollary of Lemma 49.

Proof of computational adequacy (adequacy, adequacy_alt, alt_adequacy). Take σ to be the base type ι in Lemma 49. □

https://doi.org/10.1017/S0960129521000153 Published online by Cambridge University Press

Using computational adequacy to compute. An interesting use of computational adequacy is that it allows one to argue semantically to obtain results about termination (i.e. reduction to a numeral) in PCF. Classically, every PCF program of type ι either terminates 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 σ be a PCF type. A *functional of type* σ is an element of $[\![\sigma]\!]$. By induction on PCF types, we define when a functional is said to be *total*:

(1) a functional i of type ι is total if $i = [\![\underline{n}]\!]$ for some natural number n;
(2) 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 τ for every total functional d of type σ.

Now, let s be a PCF term of type $\sigma_1 \Rightarrow \sigma_2 \Rightarrow \cdots \Rightarrow \sigma_n \Rightarrow \iota$. If we can prove that $[\![s]\!]$ is total, then computational adequacy lets us 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." Of course, this still requires us to prove that $[\![s]\!]$ is total, which may be challenging. But the point is that we can use domain-theoretic arguments to prove this about the denotation $[\![s]\!]$, whereas in a direct proof of termination we would only have the operational semantics available for our argument.

8. Semidecidable Propositions and PCF Terms of Base Type

In this section, we characterize those propositions that arise from the PCF interpretation, in the following sense. Every PCF term t of base type ι gives rise to a proposition via the Scott model, namely $\mathsf{isdefined}\,([\![t]\!])$. We wish to show that such propositions are semidecidable, which we define now. For ease of notation, we write $\exists$ for the propositional truncation of Σ.

Definition 50. *A proposition Q is* semidecidable *if it is equivalent to $\exists_{n_1:\mathsf{N}} \cdots \exists_{n_k:\mathsf{N}} P(n_1, \ldots, n_k)$ where k is some natural number and $P : \mathsf{N}^k \to \Omega$ is a proposition-valued family such that $P(m_1, \ldots, m_k)$ is decidable for every $(m_1, \ldots, m_k) : \mathsf{N}^k$.*

We will prove our goal that $\mathsf{isdefined}\,([\![t]\!])$ is semidecidable by showing that it is logically equivalent to $\exists_{n:\mathsf{N}}\exists_{k:\mathsf{N}}\, t \rhd^k \underline{n}$ and by proving that $t \rhd^k \underline{n}$ is decidable. Here $t \rhd^k \underline{n}$ says that t reduces to $\underline{n}$ in at most k steps. A first step toward this is the following, which is a consequence of soundness and computational adequacy.

Lemma 51. *Let t be a PCF term of type ι. We have the following logical equivalences:*

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

Proof (`char_pcf_propositions`). 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 $t \rhd^* \underline{\mathsf{value}\,([\![t]\!])(p)}$, so we are done.

Conversely, suppose that we are given a natural number n such that $t \rhd^* \underline{n}$. Soundness and Proposition 40 then yield $[\![t]\!] = \eta(n)$. Now $\star : \mathsf{isdefined}\,(\eta(n))$, so we may transport along the equality to get an element of $\mathsf{isdefined}\,([\![t]\!])$. □

https://doi.org/10.1017/S0960129521000153 Published online by Cambridge University Press

In order to characterize the propositions arising from PCF terms of base type as semidecidable, we wish to prove that $t \rhd^* \underline{n}$ is semidecidable for every PCF term t of type ι and natural number n. We do so by proving some more general results, which we present in Sections 8.1 and 8.2. Here, we outline our general strategy and highlight the main theorems and their applications to the problem at hand.

Given any (proposition-valued) relation R on a type X, we can define the k-step reflexive transitive closure R^k of R and prove that xR^*y if and only if $\exists_{k:\mathrm{N}} xR^k y$. Thus, we obtain the following (intermediate) result.

Lemma 52. *For every PCF term t of type ι, we have*

$$\mathsf{isdefined}(\llbracket t \rrbracket) \longleftrightarrow \exists_{n:\mathrm{N}} \exists_{k:\mathrm{N}} t \rhd^k \underline{n}.$$

Proof (`char_pcf_propositions'`). This follows from Lemmas 51 and 57. □

Thus, to prove that $s \rhd^* t$ is semidecidable, it suffices to show that $s \rhd^k t$ is decidable for every natural number k. To this end, we prove the following in Section 8.1.

Theorem (Theorem 61)**.** *Let R be relation on a type X. If*

(1) X has decidable equality;
(2) R is single-valued;
(3) $\sum_{y:X} xRy$ is decidable for every $x : X$;

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

Thus, $s \rhd^k t$ is decidable if it satisfies the Assumptions (1)–(3). Assumptions (2) and (3) can be verified by inspection of the small-step operational semantics once (1) has been proved.

Hence, we are to prove that the type of PCF terms has decidable equality. This can be done fairly directly by induction (as pointed out by one of the anonymous referees). However, we take it as an opportunity to study (in Section 8.2) a more general and powerful result on indexed W-types (see Theorem 73), which is interesting in its own right. For now, we take it as proved that the PCF terms have decidable equality and continue our study of propositions coming from PCF terms at the base type.

Theorem 53. *The propositions that arise from PCF terms t of type ι are all semidecidable, as witnessed by the following logical equivalence:*

$$\mathsf{isdefined}\ (\llbracket t \rrbracket) \longleftrightarrow \exists_{n:\mathrm{N}} \exists_{k:\mathrm{N}}\ t \rhd^k \underline{n}$$

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

Given this theorem, it is natural to ask whether we can construct the Scott model of PCF using a restricted version of the lifting monad. Write Ω_{sd} for the type of propositions that are semidecidable. Theorem 53 says that the map

$$\begin{aligned} \text{PCF terms of type } \iota &\to \Omega \\ t &\mapsto \mathsf{isdefined}(\llbracket t \rrbracket) \end{aligned}$$

factors through Ω_{sd}. Thus, could we also have constructed the Scott model of PCF using the restricted lifting $\mathscr{L}_{\mathrm{sd}}\ (X) :\equiv \sum_{P:\Omega_{\mathrm{sd}}} (P \to X)$?

https://doi.org/10.1017/S0960129521000153 Published online by Cambridge University Press

Of course, $\mathscr{L}_{\mathrm{sd}}(X)$ is not a dcpo, because, recalling our construction of suprema in $\mathscr{L}(X)$, given a directed family $u : I \to \mathscr{L}_{\mathrm{sd}}(X)$, the proposition $\left\| \sum_{i:I} \mathsf{isdefined}(u_i) \right\|$ need not be semidecidable. However, one might think that $\mathscr{L}_{\mathrm{sd}}(X)$ still has suprema of $\mathbb{N}$-indexed directed families (which would suffice for the Scott model), but proving this requires an instance of the axiom of countable choice, cf. Knapp (2018, Theorem 5.34) and Escardó and Knapp (2017, Theorem 5). Moreover, $\mathscr{L}_{\mathrm{sd}}$ is a monad if and only a particular choice principle (which is implied by countable choice) holds, see Escardó and Knapp (2017, Theorem 3) and Knapp (2018, Section 5.8). In fact, this choice principle is the one discussed in Section 1.3; Knapp (2018, Theorem 5.28) proves that if X is a set then $\mathscr{L}_{\mathrm{sd}}(X)$ is equivalent to the quotiented delay monad.

Again, as pointed out in Section 1.3, the problem is that this choice principle cannot be proved in constructive univalent type theory.

8.1 Decidability of the k-step reflexive transitive closure of a relation

In this section, we provide sufficient conditions on a relation for its k-step reflexive transitive closure to be decidable. The purpose of this section is to prove Theorem 61, whose use we have explained above.

Definition 54 (`hrel`). *A* relation on X *is a term of type* $X \to X \to \Omega$.

Definition 55 (`refltransclos_step`, `refltransclos_step_hrel`). *Let R be a relation on a type X. We wish to define the k-step reflexive transitive closure of R. As in Definition 35, we want this to be proposition-valued again. Therefore, we proceed as follows. For any natural number k, define xR_ky by induction on k:*

(1) $xR_0y :\equiv x = y$;
(2) $xR_{k+1}z :\equiv \sum_{y:X} xRy \times yR_kz$.

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 toward that.

Lemma 56. *Let R be a relation on X. Recall the untruncated reflexive transitive closure R_* from Definition 35. We have a logical equivalence for every x, y in X:*

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

Proof (`stepleftequiv`, `left_regular_equiv`). Define $xR'y$ 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 $\prod_{k:\mathbb{N}} (xR_ky \to$

https://doi.org/10.1017/S0960129521000153 Published online by Cambridge University Press

$xR'y$) by induction on k. This yields $\left(\sum_{k:\mathsf{N}} xR_k y\right) \to xR'y$. The converse is also easily established. Thus, $xR'y$ and $\sum_{k:\mathsf{N}} xR_k y$ are logically equivalent, finishing the proof. □

The next lemma extends the previous one to the propositional truncations.

Lemma 57. *Let R be a relation on X. For every $x, y : X$, we have a logical equivalence:*

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

Proof (`stepleftequiv_hrel`, `left_regular_equiv`). Let x and y be in X. By the previous lemma and functoriality of propositional truncation, we have

$$xR^*y \equiv \|xR_*y\| \longleftrightarrow \left\| \sum_{k:\mathsf{N}} xR_k y \right\|.$$

But the latter is equivalent to $\left\| \sum_{k:\mathsf{N}} \|xR_k y\| \right\| \equiv \left\| \sum_{k:\mathsf{N}} xR^k y \right\|$ by The Univalent Foundations Program (2013, Theorem 7.3.9). This may also be proved directly, as done in the formalization. □

Definition 58 (`is_singlevalued`)**.** *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 59 (`isdecidable_hrel`)**.** *A relation R on X is said to be* decidable *if the type xRy is decidable for every x and y in X.*

Lemma 60. *Let X be a type. If X is decidable, then so is $\|X\|$.*

Proof (`decidable_ishinh`). 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 0. But 0 is a proposition, so we may actually assume that we have $x : X$. Using $\neg X$, we then obtain 0, as desired. □

Theorem 61. *Let R be relation on a type X. If*

(1) X has decidable equality;
(2) R is single-valued;
(3) $\sum_{y:X} xRy$ is decidable for every $x : X$;

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

Proof (`decidable_step`). Suppose X and R satisfy conditions (1)–(3). By Lemma 60, 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

https://doi.org/10.1017/S0960129521000153 Published online by Cambridge University Press

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

to be decidable. By (3), 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, $yR_k z$ is decidable. If we have $yR_k z$, then we get $(*)$. So suppose that $\neg yR_k z$. We claim that $\neg(*)$. For suppose $(*)$, then we obtain $y' : X$ with xRy' and $y'R_k z$. But R is single-valued, so $y = y'$ and hence, $yR_k z$, contradicting our assumption. □

8.2 Decidable equality and indexed W-types

We wish to prove that a certain class of indexed W-types has decidable equality. Indexed W-types are a generalization of W-types that allows for many-sorted terms. One may consult (The Univalent Foundations Program 2013, Section 5.3) for an explanation of regular W-types. The PCF terms form a natural example of an indexed W-type, where the sorts will be the formal types of PCF terms. We apply the general result for indexed W-types to see that the PCF terms have decidable equality.

8.2.1 PCF terms as an indexed W-type

In this section, we explain what indexed W-types are and how PCF terms can encoded as such an indexed W-type.

Definition 62 (indexedWtype). *Let A and I be types and let B be a type family over A. Suppose we have $t : A \to I$ and $s : \left(\sum_{a:A} B(a)\right) \to I$. The* indexed W-type $\mathsf{W}_{s,t}$ specified by *s and t is the inductive type family over I generated by the following constructor:*

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

We have the following induction principle for indexed W-types. If $E : \prod_{i:I}\left(\mathsf{W}_{s,t}(i) \to \mathcal{U}\right)$, 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 $f : \prod_{b:B(a)} \mathsf{W}_{s,t}(s(a,b))$ 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, we 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 63. In this example, we 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 T is inductively defined as:

(1) zero is a term of type ι;
(2) succ is a term of type $\iota \Rightarrow \iota$;
(3) for every PCF type σ and τ, we have a term $\mathsf{app}_{\sigma,\tau}$ of type $(\sigma \Rightarrow \tau) \Rightarrow \sigma \Rightarrow \tau$.

https://doi.org/10.1017/S0960129521000153 Published online by Cambridge University Press

We can encode T as an indexed W-type. Let us write 2 for $1+1$ and 0_2 and 1_2 for its elements. Take I to be the type of PCF types and put $A :\equiv 2 + (I \times I)$. Define $B : A \to \mathscr{U}$ by

$$B(\,\mathsf{inl}\,(0_2)) :\equiv B(\,\mathsf{inl}\,(1_2)) :\equiv 0 \quad \text{and} \quad B(\,\mathsf{inr}\,(\sigma,\tau)) :\equiv 2.$$

Finally, define t by

$$t(\,\mathsf{inl}\,(0_2)) :\equiv \iota; \quad t(\,\mathsf{inl}\,(1_2)) :\equiv \iota \Rightarrow \iota; \quad \text{and} \quad t(\,\mathsf{inr}\,(\sigma,\tau)) :\equiv \tau;$$

and s by

$$s(\,\mathsf{inr}\,(\sigma,\tau),0_2) :\equiv \sigma \Rightarrow \tau; \quad \text{and} \quad s(\,\mathsf{inr}\,(\sigma,\tau),1_2) :\equiv \sigma;$$

on the other elements s is defined as the unique function from 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 W-type that are sufficient to conclude that an indexed W-type has decidable equality. We first need a few definitions before we can state the theorem.

Definition 64 (`WeaklyCompactTypes` in Escardó 2019, `picompact`)**.** *A type X is called* Π-compact *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 65 (`picompact_empty`, `picompact_unit`)**.** The empty type 0 is vacuously Π-compact. The unit type 1 is also easily seen to be Π-compact. There are also interesting examples of infinite types that are Π-compact, such as N_∞, the one-point compactification of the natural numbers (Escardó 2019, `WeaklyCompactTypes`).

We are now in position to state the general theorem about decidable equality on indexed W-types.

Theorem 66. *Let A and I be types and B a type family over A. Suppose $t : A \to I$ and $s : \left(\sum_{a:A} B(a)\right) \to I$. If A has decidable equality, $B(a)$ is Π-compact for every $a : A$ and I is a set, then $\mathsf{W}_{s,t}(i)$ has decidable equality for every $i : I$.*

The proof of Theorem 66 is quite technical, so we postpone it until Section 8.2.4. Instead, we next describe how to apply the theorem to prove that the PCF terms have decidable equality.

8.2.3 PCF terms have decidable equality

In this section, we show that the PCF terms have decidable equality by applying Theorem 66. Before we proceed, we record some useful lemmas.

Lemma 67. *Let X and Y be logically equivalent types. The type X is decidable if and only if Y is decidable.*

Proof (`decidable_iff`)*.* Straightforward. □

Definition 68. *A type X is called a* retract *of a type Y if there are maps $s : X \to Y$ (the* section*) and $r : Y \to X$ (the* retraction*) such that $\prod_{x:X} r(s(x)) = x$.*

https://doi.org/10.1017/S0960129521000153 Published online by Cambridge University Press

Lemma 69. *Let X be a retract of Y. If Y has decidable equality, then so does X.*

Proof (isdeceq_retract). Let $r : Y \to X$ and $s : X \to Y$ be respectively the retraction and section establishing X as a retract of Y. Let $a, b : X$. Since Y has decidable equality, we can consider two cases: $r(a) = r(b)$ and $r(a) \neq r(b)$. In the first case, we find $a = s(r(a)) = s(r(b)) = b$. In the second case, we immediately see that $a \neq b$. This finishes the proof. □

Lemma 70. *The Π-compact types are closed under binary coproducts.*

Proof (picompact_coprod). Let X and Y be Π-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 $F_X(x) :\equiv F(\mathsf{inl}\,(x))$ and $F_Y : Y \to \mathcal{U}$ 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 Π-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 67 now finishes the proof. □

Finally, let us see how to apply Theorem 66 to see that the PCF terms have decidable equality.

Theorem 71. *The PCF terms have decidable equality.*

Proof. As with Example 63, we only spell out the details for the fragment T. Recall that T may be encoded as a W-type, indexed by the PCF types. Using Example 65 and Lemma 70, we see that $B(a)$ is Π-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 Theorem 66. Define $A' :\equiv \mathbf{2}$ and define $B' : A' \to \mathcal{U}$ by $B'(\mathsf{inl}\,(\star)) :\equiv \mathbf{0}$ and $B'(\mathsf{inr}\,(\star)) :\equiv \mathbf{2}$. Let t' and s' be the unique functions to $\mathbf{1}$ from A' and $\sum_{x:A'} B'(x)$, respectively. One quickly verifies that the type of PCF types is a retract of $\mathsf{W}_{s',t'}(\star)$. Observe that $B'(x)$ is Π-compact for every $x : A'$ because of Example 65 and Lemma 70. Finally, $\mathbf{1}$ and $A' \equiv \mathbf{2}$ clearly have decidable equality, so by Theorem 66 the type $\mathsf{W}_{s',t'}(\star)$ has decidable equality. Thus, by Lemma 69, so do the PCF types. □

8.2.4 Proof of Theorem 66

In this section, we prove Theorem 66 by deriving it as a corollary of another result, namely Theorem 73 below. This result seems to have been first established by Jasper Hugunin, who reported on it in a post on the Homotopy Type Theory mailing list (Hugunin 2017*a*). Our proof of Theorem 73 is a simplified written-up account of Hugunin's Coq code (Hugunin 2017*b*, FiberProperties.v).

Definition 72 (Definition 2.4.2 in The Univalent Foundations Program 2013, hfiber). *Let $f : X \to Y$ be a map. The* fiber of f over a point $y : Y$ *is*

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

Theorem 73 Jasper Hugunin. *Let A and I be types and B a type family over A. Suppose $t : A \to I$ and $s : \left(\sum_{a:A} B(a)\right) \to I$. If $B(a)$ is Π-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 $i : I$.*

Let us see how to obtain Theorem 66 from Theorem 73.

https://doi.org/10.1017/S0960129521000153 Published online by Cambridge University Press

Proof of Theorem 66 (using Theorem 73) (`indexedWtype_deceq'`). 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. □

We now embark on a proof of Theorem 73. 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 $s : \left(\sum_{a:A} B(a)\right) \to I$.

We do not prove the theorem 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 74. *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 (`dec_depeq`). Suppose X is a set, $x : X$ and $y, y' : Y(x)$ with $e : (x, y) = (x, y')$. 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 75 (`subtrees`)**.** *For each $i : I$, define*

$$\mathsf{sub}_i : \mathsf{W}_{s,t}(i) \to \sum_{p:\mathsf{fib}_t(i)} \prod_{b:B(\mathsf{pr}_1(p))} \mathsf{W}_{s,t}(s(\mathsf{pr}_1(p), b))$$

by induction:

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

For notational convenience, we will omit the subscript of sub.

Lemma 76. *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 (`subtrees_eq`). Suppose $\mathsf{indexesup}(a, f) = \mathsf{indexedsup}(a, g)$. Then

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

As $\mathsf{fib}_t(i)$ is decidable, it is a set by Hedberg's Theorem (The Univalent Foundations Program 2013, Theorem 7.2.5). Therefore, $f = g$ by Lemma 74. □

Definition 77 (`getfib`)**.** *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 omit the subscript of getfib.

Lemma 78. *Let $i, j : I$ with a path $p : i = j$ and $w : \mathsf{W}_{s,t}(i)$. We have the following equality:*

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

Proof (`getfib_transport`). By path induction on p. □

We are now in position to state and prove the lemma from which Theorem 73 follows.

https://doi.org/10.1017/S0960129521000153 Published online by Cambridge University Press

Lemma 79. *Suppose that $B(a)$ is Π-compact for every $a : A$ and that the fiber of t over each $i : I$ has decidable equality. 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*

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

is decidable.

Proof (indexedWtype_deceq_transport). Suppose $i : I$ and $u : \mathsf{W}_{s,t}(i)$. We proceed by induction on u and so we assume that $u \equiv \mathsf{indexedsup}(a, f)$. 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 $p : t(a) = j$ 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 $t(a')$ has decidable equality. Hence, we can decide if $\left(a', \mathsf{refl}_{t(a')}\right)$ 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 78, the left-hand side is equal to $(a, \mathsf{refl}_{t(a)} \bullet p)$ which is in turn equal to (a, p), contradicting our assumption that $\left(a', \mathsf{refl}_{t(a')}\right)$ and (a, p) were not equal.

Now suppose that $\left(a', \mathsf{refl}_{t(a')}\right) = (a, p)$. From this, we obtain paths $e_1 : a' = a$ and $e_2 : \mathsf{transport}^{\lambda x:A.t(x)=t(a')}\left(e_1, \mathsf{refl}_{t(a')}\right) = p$. 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 the fact that $a \equiv a'$, the type $f(b) = f'(b)$ is decidable for every $b : B(a')$. Since $B(a')$ is Π-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 then, $\mathsf{indexedsup}(a', f)$ cannot be equal to $\mathsf{indexedsup}(a', f')$. For suppose that $\mathsf{indexedsup}(a', f) = \mathsf{indexedsup}(a', f')$. Then Lemma 76 yields $f = f'$, contradicting our assumption that $\neg \prod_{b:B(a)} f(b) = f'(b)$, and finishing the proof. □

Proof of Theorem 73 (indexedWtype_deceq). Let $i : I$ and $u, v : \mathsf{W}_{s,t}(i)$. Taking $j :\equiv i$ and $p :\equiv \mathsf{refl}_i$ in Lemma 79, we see that $u = v$ is decidable, as desired. □

9. Size Matters

In this penultimate section, we explain some of the subtleties regarding dcpos and universe levels. In particular, we revisit the dcpo of continuous functions while rigorously keeping track of universe levels. In the end, our analysis shows that, even in the absence of propositional resizing, the interpretation function $[\![-]\!]$ of the Scott model is well defined (Theorem 80). (For more on predicative domain theory, the reader may wish to consult our recent work de Jong and Escardó 2021*a*,b.)

As mentioned in the introduction, our results are formalized in Agda (Escardó 2019, `PCFModules`).

To study universe levels, let us suppose that we have a tower of type universes $\mathcal{U}_0 : \mathcal{U}_1 : \ldots$, indexed by meta natural numbers. (In the end, it will turn out that having just two universes $\mathcal{U}_0 : \mathcal{U}_1$ is sufficient for our purposes.) Let us fix some notation for (raising) universe levels. We write $\mathcal{U}_i^+$ for $\mathcal{U}_{i+1}$ and $\mathcal{U}_i \sqcup \mathcal{U}_j$ for $U_{\max(i,j)}$. The universes are assumed to be closed under $+$-, Σ- and Π-types and if $X : \mathcal{U}$ and $Y : X \to \mathcal{V}$, then $\sum_{x:X} Y(x), \prod_{x:X} Y(x) : \mathcal{U} \sqcup \mathcal{V}$. Finally, since $\mathcal{U} : \mathcal{U}^+$, we have $\sum_{X:\mathcal{U}} Y(X) : \mathcal{U}^+ \sqcup \mathcal{V}$ if $Y : \mathcal{U} \to \mathcal{V}$.

9.1 The lifting

In Section 1.2, we introduced Ω as the type of propositions in the universe $\mathcal{U}_0$. To see why we made this particular choice of type universe and to appreciate the considerations involved, it is helpful to consider a more general situation. Let us write $\Omega_{\mathcal{T}}$ for the propositions in some type universe $\mathcal{T}$. Define the (generalized) lifting $\mathcal{L}_{\mathcal{T}}(X)$ of a type X is as $\mathcal{L}_{\mathcal{T}}(X) :\equiv \sum_{P:\Omega_{\mathcal{T}}} (P \to X)$.

Now observe that if X is a type in a universe $\mathcal{U}$, then lifting (potentially) raises the universe level, as $\mathcal{L}_{\mathcal{T}}(X)$ is a type in universe $\mathcal{T}^+ \sqcup \mathcal{U}$. However, if X happens to be a type in $\mathcal{T}^+$, then $\mathcal{L}_{\mathcal{T}}(X)$ 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 $\mathcal{L}_{\mathcal{T}}(X)$ is as well. Despite the fact that lifting raises the universe level, one can write down the monad laws for $\mathcal{L}_{\mathcal{T}}$ and they typecheck.

Let X and I be types in universes $\mathcal{U}$ and $\mathcal{V}$, respectively. Suppose that $u : I \to \mathcal{L}_{\mathcal{T}}(X)$. Note that $\sum_{i:I} \mathsf{isdefined}\,(u_i)$ is in $\mathcal{V} \sqcup \mathcal{T}$. When considering $\mathcal{L}_{\mathcal{T}}(X)$ as a dcpo (cf. Theorem 26), we want $\sum_{i:I} \mathsf{isdefined}\,(u_i)$ to be in $\mathcal{T}$ again. One way to ensure this, is to take $\mathcal{V}$ to be $\mathcal{U}_0$. This would make $\mathcal{L}_{\mathcal{T}}(X)$ a $\mathcal{U}_0$-dcpo. Indeed, this is what we prove in the Agda formalization. In particular, this means that $\mathcal{L}_{\mathcal{T}}(X)$ has N-indexed directed suprema, which suffices for the Scott model of PCF.

9.2 The dcpo of continuous functions

In fact, we should be even more precise when it comes universe levels and dcpos than we have been so far. Write $\mathcal{W}\text{-}(\mathsf{DCPO}_\perp)_{\mathcal{U},\mathcal{V}}$ for the type of $\mathcal{W}$-directed complete posets with a least element whose underlying type is in $\mathcal{U}$ and whose underlying order takes values in $\mathcal{V}$.

Then $\mathcal{L}_{\mathcal{U}_0}(\mathsf{N}) \equiv \mathcal{L}(\mathsf{N})$ is of type $\mathcal{U}_0\text{-}(\mathsf{DCPO}_\perp)_{\mathcal{U}_1,\mathcal{U}_1}$, for example. (One easily checks that the order $\sqsubseteq$ from Theorem 26 has values in $\mathcal{U}_1$.)

Recall that $[\![\sigma \Rightarrow \tau]\!] \equiv [\![\tau]\!]^{[\![\sigma]\!]}$, the dcpo with $\perp$ of continuous functions from $[\![\sigma]\!]$ to $[\![\tau]\!]$, so let us investigate the universe levels surrounding the exponential. In general, we have

$$\begin{aligned} &\text{if } \mathcal{D} : \mathcal{W}\text{-}(\mathsf{DCPO}_\perp)_{\mathcal{U},\mathcal{V}} \text{ and } \mathcal{E} : \mathcal{W}\text{-}(\mathsf{DCPO}_\perp)_{\mathcal{U}',\mathcal{V}'},\\ &\text{then } \mathcal{E}^{\mathcal{D}} : \mathcal{W}\text{-}(\mathsf{DCPO}_\perp)_{\mathcal{W}^+\sqcup\mathcal{V}\sqcup\mathcal{V}'\sqcup\mathcal{U}\sqcup\mathcal{U}',\,\mathcal{U}\sqcup\mathcal{V}'}. \end{aligned} \tag{†}$$

We explain the universe levels involved as follows.

Let $\mathcal{D}$ be of type $\mathcal{W}\text{-}(\mathsf{DCPO}_\perp)_{\mathcal{U},\mathcal{V}}$ and write D and $\leq_{\mathcal{D}}$ for its underlying type and order, respectively. Further, let $\mathcal{E}$ be of type $\mathcal{W}\text{-}(\mathsf{DCPO}_\perp)_{\mathcal{U}',\mathcal{V}'}$ and write E and $\leq_{\mathcal{E}}$ for its underlying type and order, respectively.

The underlying type of the exponential $\mathcal{E}^{\mathcal{D}}$ is the type of functions from D to E that are continuous. The underlying order is the pointwise order: if f and g are continuous functions from D to E, then $f \leq_{\mathcal{E}^{\mathcal{D}}} g$ if $\prod_{x:D} f(x) \leq_{\mathcal{E}} g(x)$.

https://doi.org/10.1017/S0960129521000153 Published online by Cambridge University Press

Because D is in $\mathcal{U}$ and $\leq_{\mathcal{E}}$ takes values in $\mathcal{V}'$, we see that $\leq_{\mathcal{E}^{\mathcal{D}}}$ takes values in $\mathcal{U} \sqcup \mathcal{V}'$.

Furthermore, the type of functions from D to E is in $\mathcal{U} \sqcup \mathcal{U}'$. But the type of *continuous* functions also mentions $\leq_{\mathcal{D}}$ and $\leq_{\mathcal{E}}$ and *all* directed families indexed by a type in $\mathcal{W}$. In particular, the latter means that the definition of the type of continuous functions contains $\prod_{I:\mathcal{W}}$. Therefore the type of continuous functions is in $\mathcal{W}^+ \sqcup \mathcal{V} \sqcup \mathcal{V}' \sqcup \mathcal{U} \sqcup \mathcal{U}'$.

9.3 The Scott model of PCF

Given the increasing universe levels in (†), one might ask if there can be universes $\mathcal{U}, \mathcal{V}, \mathcal{W}$ such that

$$[\![-]\!] : \text{PCF types} \to \mathcal{U}\text{-}(\mathsf{DCPO}_\perp)_{\mathcal{V},\mathcal{W}}$$

typechecks.

As we mentioned, $\mathcal{L}_{\mathcal{U}_0}(\mathsf{N}) \equiv \mathcal{L}(\mathsf{N}) : \mathcal{U}_0\text{-}(\mathsf{DCPO}_\perp)_{\mathcal{U}_1,\mathcal{U}_1}$. Since, $[\![\iota]\!] \equiv \mathcal{L}(\mathsf{N})$, one would hope that

$$[\![-]\!] : \text{PCF types} \to \mathcal{U}_0\text{-}(\mathsf{DCPO}_\perp)_{\mathcal{U}_1,\mathcal{U}_1}.$$

And indeed, this is the case.

Theorem 80. *The interpretation function* $[\![-]\!]$ *from PCF types to dcpos with* $\perp$ *can be typed as:*

$$[\![-]\!] : \text{PCF types} \to \mathcal{U}_0\text{-}(\mathsf{DCPO}_\perp)_{\mathcal{U}_1,\mathcal{U}_1}.$$

Proof. If, in (†), we take $\mathcal{W}$ to be $\mathcal{U}_0$ and $\mathcal{U}, \mathcal{U}', \mathcal{V}, \mathcal{V}'$ all to be $\mathcal{U}_1$, then (†) reads

$$(-)^{(-)} : \mathcal{U}_0\text{-}(\mathsf{DCPO}_\perp)_{\mathcal{U}_1,\mathcal{U}_1} \to \mathcal{U}_0\text{-}(\mathsf{DCPO}_\perp)_{\mathcal{U}_1,\mathcal{U}_1} \to \mathcal{U}_0\text{-}(\mathsf{DCPO}_\perp)_{\mathcal{U}_1,\mathcal{U}_1},$$

as desired. □

10. Conclusion and Future Work

Our development confirms that univalent type theory is well adapted to the constructive formalization of domain-theoretic denotational semantics of programming languages like PCF, which was the original goal of this investigation. Moreover, our development is predicative. In particular, we have given a predicative version of directed complete posets. Our results show that partiality in univalent type theory via lifting works well. We rely crucially on Voevodsky's treatment of subsingletons as truth values. In particular, the propositional truncation plays a fundamental and interesting role in this work. Finally, we saw an interesting application of the abstract theory of indexed W-types in characterizing the propositions that come from PCF terms of the base type.

Regarding the Scott model of PCF, there are two questions for future research:

(1) Is there a natural extension of the map $[\![\iota]\!] \xrightarrow{\mathsf{pr}_1} \Omega$ to all PCF types? Can we characterize the propositions at types other than ι, for example, the propositions at type $\iota \Rightarrow \iota$? Are they still semidecidable?

(2) How can we better understand the fact that only semidecidable propositions occur for the Scott model, but that restricting to such propositions somehow needs a weak form of countable choice?

In de Jong and Escardó (2021*a*), we develop domain theory further in predicative and constructive univalent type theory, including continuous and algebraic dcpos, ideal completions and Scott's famous D_∞. Complementing this work, the paper (de Jong and Escardó 2021*b*) explores some aspects of domain theory that cannot be done predicatively.

https://doi.org/10.1017/S0960129521000153 Published online by Cambridge University Press

Acknowledgements. Firstly, I would like to thank Martín Escardó for suggesting and supervising this project. Secondly, I am grateful to Benedikt Ahrens for his support, his help with UniMath, and in particular for his feedback on earlier versions of this paper. I should also like to thank Andrej Bauer and Bernhard Reus for their comments and questions. Finally, I am indebted to the anonymous referees for their thorough and valuable reports that helped to improve the paper.

Conflicts of Interest. The author declares none.

Notes

[1] This formulation does not ensure that the type is a proposition, so one could also consider truncating the Σ or asking for the *least* k such that $\alpha(k) = 1$. But this version is sufficient for our purposes, and logically equivalent to the one with the truncated Σ.

[2] In fact, there is a type equivalence. One can prove this using univalence and a generalized structure identity principle, cf. Escardó (2019, `LiftingIdentityViaSIP`).

References

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

Altenkirch, T., Danielsson, N. A. and Kraus, N. (2017). Partiality, revisited: The partiality monad as a quotient inductive-inductive type. In: Esparza, J. and Murawski, A. S. (eds.) *Foundations of Software Science and Computation Structures*, Lecture Notes in Computer Science, vol. 10203, Springer, 534–549. doi:10.1007/978-3-662-54458-7_31.

Benton, N., Kennedy, A. and Varming, C. (2009). Some domain theory and denotational semantics in Coq. In: Berghofer, S., Nipkow, T., Urban, C. and Wenzel, M. (eds.) *Theorem Proving in Higher Order Logics*, Lecture Notes in Computer Science, vol. 5674, Berlin Heidelberg, Springer, 115–130. doi:10.1007/978-3-642-03359-9_10.

Bishop, E. (1967). *Foundations of Constructive Analysis*, McGraw-Hill Book Company.

Bridges, D. and Richman, F. (1987). *Varieties of Constructive Mathematics*, London Mathematical Society Lecture Note Series, vol. 97, Cambridge University Press.

Capretta, V. (2005). General recursion via coinductive types. *Logical Methods in Computer Science* **1** (2). doi:10.2168/LMCS-1(2:1)2005.

Chapman, J., Uustalu, T. and Veltri, N. (2017). Quotienting the delay monad by weak bisimilarity. *Mathematical Structures in Computer Science* **29** (1) 67–92. doi:10.1017/S0960129517000184.

Coquand, T. (2018). A survey of constructive presheaf models of univalence. *ACM SIGLOG News* **5** (3) 54–65.

Coquand, T., Mannaa, B. and Ruch, F. (2017). Stack semantics of type theory. In: *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, 1–11. doi:10.1109/LICS.2017.8005130.

de Jong, T. and Escardó, M. H. (2021a). Domain theory in constructive and predicative univalent foundations. In: Baier, C. and Goubault-Larrecq, J. (eds.) *29th EACSL Annual Conference on Computer Science Logic (CSL 2021)*, Leibniz International Proceedings in Informatics (LIPIcs), vol. 183, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 28:1–28:18. doi:10.4230/LIPIcs.CSL.2021.28.

de Jong, T. and Escardó, M. H. (2021b). Predicative aspects of order theory in univalent foundations. To appear in the Proceedings of *FSCD 2021*, LIPIcs, vol. 195. arXiv[math.LO]:2102.08812.

Escardó, M. H. (2018). Constructive mathematics in univalent type theory. Slides for a talk at *Homotopy Type Theory Electronic Seminar Talks*, 26 April. `https://www.uwo.ca/math/faculty/kapulkin/seminars/hottestfiles/Escardo-2018-04-26-HoTTEST.pdf`.

Escardó, M. H. (2019). TypeTopology — Various new theorems in constructive univalent mathematics written in Agda. `https://github.com/martinescardo/TypeTopology`.

Escardó, M. H. and Knapp, C. M. (2017). Partial elements and recursion via dominances in univalent type theory. In: Goranko, V. and Dam, M. (eds.) *26th EACSL Annual Conference on Computer Science Logic (CSL 2017)*, Leibniz International Proceedings in Informatics (LIPIcs), vol. 82, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 21:1–21:16. doi:10.4230/LIPIcs.CSL.2017.21.

Hindley, J. R. and Seldin, J. P. (2008). *Lambda-Calculus and Combinators, an Introduction*, 2nd edn., Cambridge University Press. doi:10.1017/cbo9780511809835.

Hugunin, J. (2017a). Characterizing the equality of Indexed W types. Post on the *Homotopy Type Theory* mailing list. `https://groups.google.com/d/msg/homotopytypetheory/qj2OvRvqf-Q/hGFBczJGAwAJ`.

Hugunin, J. (2017b). IWTypes — A Coq development of the theory of Indexed W types with function extensionality. `https://github.com/jashug/IWTypes`.

Knapp, C. (2018). *Partial Functions and Recursion in Univalent Type Theory*. PhD thesis, School of Computer Science, University of Birmingham.

Kock, A. (1991). Algebras for the partial map classifier monad. In: Carboni, A., Pedicchio, M. C. and Rosolini, G. (eds.) *Category Theory*, Lecture Notes in Mathematics, vol. 1488, Springer, 262–278. doi:10.1007/BFB0084225.

Kraus, N., Escardó, M., Coquand, T. and Altenkirch, T. (2017). Notions of anonymous existence in Martin-Löf Type Theory. *Logical Methods in Computer Science* **13**. doi:10.23638/LMCS-13(1:15)2017.

Plotkin, G. (1977). LCF considered as a programming language. *Theoretical Computer Science* **5** (3) 223–255. doi:10.1016/0304-3975(77)90044-5.

Plotkin, G. (1983). Domains. Lecture notes on domain theory, known as the *Pisa Notes*. https://homepages.inf.ed.ac.uk/gdp/publications/Domains_a4.ps.

Reus, B. and Streicher, T. (1999). General synthetic domain theory — a logical approach. *Mathematical Structures in Computer Science* **9** (2) 177–223. doi:10.1017/S096012959900273X.

Scott, D. S. (1993). A type-theoretical alternative to ISWIM, CUCH, OWHY. *Theoretical Computer Science* **121** (1) 411–440. doi:10.1016/0304-3975(93)90095-B.

Streicher, T. (2006). *Domain-Theoretic Foundations of Functional Programming*, World Scientific. doi:10.1142/6284.

Swan, A. W. (2019a). Choice, collection and covering in cubical sets. Talk in the electronic *HoTTEST* seminar, 6 November. Slides at https://www.uwo.ca/math/faculty/kapulkin/seminars/hottestfiles/Swan-2019-11-06-HoTTEST.pdf. Video recording at https://www.youtube.com/watch?v=r9KbEOzyr1g.

Swan, A. W. (2019b). Counterexamples in cubical sets. Slides for a talk at *Mathematical Logic and Constructivity: The Scope and Limits of Neutral Constructivism*, Stockholm, 20 August. http://logic.math.su.se/mloc-2019/slides/Swan-mloc-2019-slides.pdf.

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

Voevodsky, V., Ahrens, B., Grayson, D. (2019). UniMath — a computer-checked library of univalent mathematics. https://github.com/unimath/unimath#citing-unimath.

Cite this article: de Jong T (2021). The Scott model of PCF in univalent type theory. *Mathematical Structures in Computer Science* **31**, 1270–1300. https://doi.org/10.1017/S0960129521000153