Partial elements and recursion via dominances in univalent type theory

Martín H. Escardó[1] and Cory M. Knapp[1]

1 **School of Computer Science, University of Birmingham, UK**
{m.escardo,cmk497}@cs.bham.ac.uk

Abstract

We begin by revisiting partiality in univalent type theory via the notion of dominance. We then perform first steps in constructive computability theory, discussing the consequences of working with computability as property or structure, without assuming countable choice or Markov's principle. A guiding question is what, if any, notion of partial function allows the proposition "all partial functions on natural numbers are Turing computable" to be consistent.

1998 ACM Subject Classification F.4.1 Mathematical Logic: Lambda Calculus and related systems; F.1.1 Models of Computation: Computability Theory

Keywords and phrases univalent type theory, homotopy type theory, partial function, dominance, recursion theory, computability theory

Digital Object Identifier 10.4230/LIPIcs.CSL.2017.21

1 Introduction

We begin by revisiting partial functions in type theory [4, 3, 7, 5, 8, 21]. We work informally, but rigorously, in a univalent type theory as in the HoTT Book [23], with constructive content as developed in cubical type theory [9]. We then perform first steps in computability theory, where, in particular, we discuss the consequences of working with computability as property or as structure, in the absence of choice axioms, excluded middle and Markov's principle.

A guiding question is what, if any, notion of partial function allows the proposition "all partial functions from $\mathbb{N}$ to $\mathbb{N}$ are Turing computable" to be consistent in constructive univalent type theory. We begin with a natural, but too liberal, notion, that allows partial functions that are not computable in any reasonable sense to be easily constructed. We then use dominances [20] to try to restrict the notion of partial function (Section 2) so that this would be possible, discussing the difficulties that arise in this attempt (Section 3), and their relation to previous work by other authors (Section 4).

Dominances also occur in synthetic domain theory [11, 16, 24], synthetic computability theory [20, 1], and synthetic topology [20, 2]. In such synthetic approaches, one typically considers countable dependent choice and Markov's principle, in addition to axioms that contradict the principle of excluded middle. For example, in his synthetic approach to computability, Bauer [1] works with a non-classical axiom saying that there are enumerably many enumerable subsets of $\mathbb{N}$. We emphasize that, although we do use dominances, our approach is not synthetic, and, moreover, is compatible with classical logic.

In Section 2 we define a type $X \rightharpoonup Y$ of partial functions for any two types X and Y, so that partial functions correspond to ordinary (total) functions $A \to Y$ defined on a type A embedded into X, and show that it is equivalent to $X \to \mathcal{L}\,Y$ where $\mathcal{L}\,Y$ is a type of partial elements of Y. The lifted type $\mathcal{L}\,Y$ is a directed-complete partially ordered set if Y is a

set in the sense of univalent type theory, and this can be used to solve recursive definitions involving Scott continuous functions.

In Section 3, we define a Gödel encoding of Turing machines by natural numbers in the usual way, together with a Kleene-bracket function

$$\{-\} : \mathbb{N} \to (\mathbb{N} \to \mathcal{L}\,\mathbb{N}),$$

and define a function $\mathbb{N} \to \mathcal{L}\,\mathbb{N}$ to be computable if it is in the image of Kleene bracket. This can be formulated in two ways:

$$\begin{aligned} \text{computationalStructure}(f) &\stackrel{\text{def}}{=} \Sigma(e : \mathbb{N}), \{e\} = f, \\ \text{isComputable}(f) &\stackrel{\text{def}}{=} (\exists(e : \mathbb{N}), \{e\} = f) = \|\text{computationalStructure}(f)\|\,. \end{aligned}$$

The first definition says that a computational structure on f is a Gödel number of a Turing machine that computes f. The second definition says that f is computable if there is such a Gödel number, without exhibiting any. Here $\|-\|$ denotes the (propositional) truncation operator, which collapses a type to a subsingleton by identifying all its elements. Existence as a proposition is taken to be the truncation of existence as structure:

$$\exists(x : X), A(x) \stackrel{\text{def}}{=} \|\Sigma(x : X), A(x)\|\,.$$

We emphasize, for comparison, that in topos theory this is a theorem (often taken as a definition), expressing existence in terms of Σ, where $\|X\|$ is known as the *support* of the object X (the image of the unique map into the terminal object 1).

We can prove Rogers' version of Kleene's second recursion theorem, which says that for every computable total function $f : \mathbb{N} \to \mathbb{N}$, there is $e : \mathbb{N}$ with $\{e\} = \{f(e)\}$, in the following two versions:

$$\begin{aligned} &\Pi(f : \mathbb{N} \to \mathbb{N}), \text{computationalStructure}(f) \to \Sigma(e : \mathbb{N}), \{e\} = \{f(e)\}, \\ &\qquad \Pi(f : \mathbb{N} \to \mathbb{N}), \text{isComputable}(f) \to \exists(e : \mathbb{N}), \{e\} = \{f(e)\}, \end{aligned}$$

where the second version is a direct consequence of the first, by the universal property of propositional truncation.

For some statements it does matter whether computability is taken as property or structure. Consider, for example, the type

$$\Pi(f : \mathbb{N} \to \mathbb{N}), \text{computationalStructure}\, f$$

of functions that assign computational structure to every total function $\mathbb{N} \to \mathbb{N}$ (analogous to Lisp's quote operator). In the presence of function extensionality, the assumption that the above type is inhabited leads to a contradiction [22]. But the proposition

$$\Pi(f : \mathbb{N} \to \mathbb{N}), \text{isComputable}\, f$$

is consistent in a type theory with function extensionality and propositional truncation, with the effective topos as a model [12].

On the other hand, the statement that every *partial* function $\mathbb{N} \rightharpoonup \mathbb{N}$ is computable is false, even when the computational structure is not given. In fact, there is an easily defined partial function $f : \mathbb{N} \rightharpoonup \mathbb{N}$ that semidecides the complement of the Halting Set, in the sense that $f(x)$ is defined iff $\{x\}(x)$ is undefined, and hence is not computable (Section 3.4).

This motivates the consideration of dominances, to restrict the allowed domains of definition of partial elements (Section 2). Of particular interest is the *Rosolini dominance* [20], which consists of the propositions of the form $\exists(n : \mathbb{N}), \alpha(n) = 1$ with $\alpha : \mathbb{N} \to 2$ (Section 2.4). This is related to various partiality monads considered in the literature, as discussed in the concluding Section 4.

2 Partial functions and recursion via dominances

2.1 Partial functions

Assuming univalence, the type of functions $X \to Y$ is equivalent to the type of functional relations [17], where $\mathcal{U}$ is a type universe:

$$(X \to Y) \simeq \Sigma(R : X \to Y \to \mathcal{U}), \Pi(x : X), \mathrm{isContr}\,(\Sigma(y : Y), R\,x\,y)\,. \tag{1}$$

Recall that a type A is called contractible (or a singleton), written $\mathrm{isContr}(A)$, if we have $a_0 : A$, called a center of contraction of A, which is equal to every element of A:

$$\mathrm{isContr}(A) \stackrel{\mathrm{def}}{=} \Sigma(a_0 : A), \Pi(a : A), a_0 = a.$$

The above use of contractibility can be read as saying that there is a unique (y, r), with $y : Y$ and $r : R\,x\,y$, rather than the weaker statement that there is a unique $y : Y$ satisfying $R\,x\,y$, with possibly several $r : R\,x\,y$ for this y.

A type A is a called a *proposition* (or said to have at most one element, or to be a subsingleton), written $\mathrm{isProp}(A)$, if any two of its elements are equal:

$$\mathrm{isProp}(A) \stackrel{\mathrm{def}}{=} \Pi(a, b : A), a = b.$$

Then in the following definition of a type of partial functions, we replace unique existence by the existence of at most one pair (y, r) with $y : Y$ and $r : R\,x\,y$:

$$\begin{aligned} (X \rightharpoonup Y) &\stackrel{\mathrm{def}}{=} \Sigma(R : X \to Y \to \mathcal{U}), \Pi(x : X), \mathrm{isProp}\,(\Sigma(y : Y), R\,x\,y) && (2)\\ &\simeq \Sigma(A : \mathcal{U})(e : A \to X), \mathrm{isEmbedding}(e) \times (A \to Y), && (3)\end{aligned}$$

where

$$\mathrm{isEmbedding}(e : A \to X) \stackrel{\mathrm{def}}{=} \Pi(x : X), \mathrm{isProp}(\Sigma(a : A), e(a) = x).$$

Then the type of ordinary functions $(X \to Y)$ is embedded into the type $(X \rightharpoonup Y)$ of partial functions, and we say that a partial function is total if it is in the image of this embedding.

In topos theory one has an isomorphism as in (1) with $\mathcal{U}$ replaced by the subobject classifier and the proposition $\mathrm{isContr}\,(\Sigma(y : Y), R\,x\,y)$ replaced by the unique existence of some $y : Y$ with $R\,x\,y$. This replacement is also possible in univalent type theory in the special case where the type Y is a set (meaning that for any two $y, y' : Y$, the identity type $y = y'$ is a subsingleton). For the general case, however, we need to keep $\mathcal{U}$ and work with the contractibility of the Σ type as above. Notice also that, in our type theory, the right-hand side of (1) is in a higher universe than the left-hand side (cf. Section 2.9).

2.2 Partial functions via lifting

If we define the lifting of a type Y (also referred to as the type of *partial elements* of Y) by

$$\begin{aligned} \mathcal{L}\,Y &\stackrel{\mathrm{def}}{=} 1 \rightharpoonup Y && (4)\\ &\simeq \Sigma(A : Y \to \mathcal{U}), \mathrm{isProp}\,(\Sigma(y : Y), A\,y) && (5)\\ &\simeq \Sigma(P : \mathcal{U}), \mathrm{isProp}\,P \times (P \to Y), && (6)\end{aligned}$$

then partial functions reduce to ordinary functions with lifted codomain:

$$(X \rightharpoonup Y) \simeq (X \to \mathcal{L}\,Y).$$

Working with the representation (6) of $\mathcal{L}\,Y$, we denote the first and third projections by

$$\text{extent} : \mathcal{L}\,Y \to \mathcal{U}, \qquad \text{value} : \Pi(u : \mathcal{L}\,Y), \text{extent}\,u \to Y. \tag{7}$$

Then $\mathcal{L}$ has a strong-monad structure, which we present as a Kleisli triple. The unit $\eta_X : X \to \mathcal{L}\,X$ is given by $\eta_X(x) \stackrel{\text{def}}{=} (1, -, \lambda p.x)$, where we denote a suppressed witness by "$-$". Given $f : X \to \mathcal{L}\,Y$, we define its Kleisli extension $f^\sharp : \mathcal{L}\,X \to \mathcal{L}\,Y$ by

$$f^\sharp(P : \mathcal{U}, -, \phi : P \to X) \stackrel{\text{def}}{=} (Q : \mathcal{U}, -, \gamma : Q \to Y),$$

$$\text{where} \quad Q \stackrel{\text{def}}{=} \Sigma(p : P), \text{extent}(f(\phi(p)) \quad \text{and} \quad \gamma(p, e) \stackrel{\text{def}}{=} \text{value}(f(\phi(p)))(e).$$

Then Kleisli composition corresponds to relational composition, so that post-composition with η_Y is an embedding of $(X \to Y)$ into $(X \to \mathcal{L}\,Y)$, often taken as an implicit coercion.

We say that an element of $\mathcal{L}\,Y$ is *defined* (or *total*) if it is in the image of $\eta_Y : Y \to \mathcal{L}\,Y$, which is equivalent to saying that its extent of definition is inhabited (and hence contractible). The type $\mathcal{L}\,Y$ has an *undefined* element $\bot$, with empty extent of definition. The assertion that every partial element is defined or else undefined is equivalent to the principle of excluded middle (every proposition is inhabited or empty).

2.3 Recursive definitions

If the type Y is a set, then $\mathcal{L}\,Y$ is a directed-complete partially ordered set with least element $\bot$. Define $- \leq - : \mathcal{L}\,Y \to \mathcal{L}\,Y \to \mathcal{U}$ by

$$(u \leq v) \stackrel{\text{def}}{=} \Sigma(t : \text{extent}(u) \to \text{extent}(v)), \Pi(p : \text{extent}(u)), \text{value}(u)(p) = \text{value}(v)(t(p)).$$

As Y is a set, this relation is proposition-valued, and reflexive, transitive and antisymmetric.

▶ **Theorem 1.** *If Y is a set, then $(\mathcal{L}\,Y, \leq)$ is a directed-complete poset.*

Proof. Given a directed family $u_i : \mathcal{L}\,Y$, we construct its join $u_\infty : \mathcal{L}\,Y$ as follows. Firstly, we let $\text{extent}(u_\infty) \stackrel{\text{def}}{=} \|\Sigma_i, \text{extent}(u_i)\|$, so that, by construction, we have $\text{isProp}(\text{extent}(u_\infty))$. To define $\text{value}(u_\infty) : \text{extent}(u_\infty) \to Y$, we first define a function $\phi : (\Sigma_i, \text{extent}(u_i)) \to Y$ by $\phi(i, p) = \text{value}(u_i)(p)$. By the directedness of the family, we see that ϕ is constant, in the sense that any two of its values are equal, and hence, by the assumption that Y is a set, ϕ factors through a unique $\|\Sigma_i, \text{extent}(u_i)\| \to Y$ by [14, Theorem 5.4], which we take as $\text{value}(u_\infty)$. We omit the routine verification that u_∞ is the least upper bound of the family u_i. ◀

This can be used to construct least fixed points of continuous functions in the usual way, and hence solve recursive functional equations, using the fact that a function type $X \to \mathcal{L}\,Y$ is also directed complete under the pointwise order.

2.4 Partial elements with semidecidable extent of definition

To try to make the statement that partial functions $\mathbb{N} \to \mathcal{L}\,\mathbb{N}$ are computable consistent, we need to restrict the supply of propositions that are allowed to arise as extents of definition of partial elements, motivating the subject of dominances, mentioned above and to be discussed shortly. The propositions that can arise as the extents of definition of partial elements from Turing machines are characterized as those of the form

$$\langle\alpha\rangle \stackrel{\text{def}}{=} \Sigma(n : \mathbb{N}), \alpha(n) = 1,$$

with $\alpha : \mathbb{N} \to 2$ computable and $\mathrm{isProp}\langle\alpha\rangle$ (Section 3.5). Rosolini considered partial elements with extents of the above form [20], without assuming f computable, which we refer to as *Rosolini propositions*:

$$\begin{aligned} \mathrm{isRosolini}(A) &\stackrel{\mathrm{def}}{=} \exists(\alpha : \mathbb{N} \to 2), \mathrm{isProp}\langle\alpha\rangle \times (A = \langle\alpha\rangle). \\ \mathrm{isSemiDecidable}(A) &\stackrel{\mathrm{def}}{=} \exists(\alpha : \mathbb{N} \to 2), \mathrm{isComputable}(\alpha) \times \mathrm{isProp}\langle\alpha\rangle \times (A = \langle\alpha\rangle). \end{aligned}$$

Assuming the (consistent, internal) statement that all (total) functions $\mathbb{N} \to \mathbb{N}$ are computable, the Rosolini and semidecidable propositions coincide, of course.

2.5 Dominances

A dominance is a set of propositions closed under Σ, and having the proposition 1 as a member. It is convenient to present a dominance by the following data and properties:

D1. A function $d : \mathcal{U} \to \mathcal{U}$.
Then a type X is called *dominant* if $d(X)$ holds.

D2. A function $\Pi(X : \mathcal{U}), \mathrm{isProp}(d(X))$.
(Being dominant is a proposition.)

D3. A function $\Pi(X : \mathcal{U}), d(X) \to \mathrm{isProp}(X)$.
(Dominant types are propositions.)

D4. An element of $d(1)$ where 1 is the terminal type.
(The proposition 1 is dominant.)

D5. A function $\Pi(P : \mathcal{U})(Q : P \to \mathcal{U}), d(P) \to (\Pi(p : P), d(Q(p))) \to d\,(\Sigma(p : P), Q(p))$.
(Dominant types are closed under Σ.)

The required unnamed element and functions (D2)–(D5) are unique as their types are propositions, and hence constitute a property of d. Notice that $d \stackrel{\mathrm{def}}{=} \mathrm{isProp}$ satisfies the dominance axioms, that is, the set of all propositions forms a dominance. In the general case, (D5) is equivalent to the seemingly weaker condition

$$\Pi(P, Q' : \mathcal{U}), d(P) \to (P \to d(Q')) \to d(P \times Q'),$$

which corresponds to the original definition of dominance given by Rosolini. (In one direction, consider the constant family $Q(p) \stackrel{\mathrm{def}}{=} Q'$. In the other, consider the type $Q' \stackrel{\mathrm{def}}{=} \Sigma(p : P), Q(p)$.)

2.6 Partial functions induced by a dominance

Given a dominance d, we define the type of d-partial functions by

$$(X \rightharpoonup_d Y) \stackrel{\mathrm{def}}{=} \Sigma(R : X \to Y \to \mathcal{U}), \Pi(x : X), d\,(\Sigma(y : Y), R\,x\,y)\,. \tag{8}$$

The axiom that the proposition 1 is dominant ensures that the identity relation is a d-partial function, and the axiom that dominant propositions are closed under Σ ensures that d-partial functions are closed under composition. In fact, if we have relations $R : X \to Y \to \mathcal{U}$ and $S : Y \to Z \to \mathcal{U}$, their composition is

$$(R; S)\,x\,z \stackrel{\mathrm{def}}{=} \Sigma(y : Y), R\,x\,y \times S\,y\,z,$$

and the closure of d under Σ ensures that the assumptions $\Pi(x : X), d\,(\Sigma(y : Y), R\,x\,y)$ and $\Pi(y : Y), d\,(\Sigma(z : Z), S\,y\,z)$ on R and S together imply that their composite $R; S$ satisfies the required condition $\Pi(x : X), d\,(\Sigma(z : Z), (R; S)\,x\,z)$. Indeed, we have that the type $\Sigma(z : Z), (R; S)\,x\,z$ is equivalent to $\Sigma((y, r) : \Sigma(y : Y), R\,x\,y), \Sigma(z : Z), S\,y\,z$ by the reshuffling map $(z, (y, (r, s))) \mapsto ((y, r), (z, s))$. Being the sum of the dominant propositions

$\Sigma(z : Z), S\,y\,z$ indexed by a dominant proposition $\Sigma(y : Y), R\,x\,y$, this type is itself a dominant proposition, as required. Conversely, by constructing suitable relations R and S depending on a given family $Q : P \to \mathcal{U}$ of dominant propositions indexed by a dominant proposition P, we can see that closure under Σ is necessary.

2.7 Partial elements induced by a dominance

The d-lifting, or type of d-partial elements of Y, is defined by

$$\begin{aligned} \mathcal{L}_d\,Y &\stackrel{\text{def}}{=} (1 \rightharpoonup_d Y) && (9)\\ &\simeq (\Sigma(A : Y \to \mathcal{U}), d\,(\Sigma(y : Y), A\,y)) && (10)\\ &\simeq (\Sigma(P : \mathcal{U}), d(P) \times (P \to X)) && (11) \end{aligned}$$

so that

$$(X \rightharpoonup_d Y) \simeq (X \to \mathcal{L}_d\,Y).$$

The dominance axioms ensure that the monad structure on $\mathcal{L}$ discussed above restricts to $\mathcal{L}_d$, again with Kleisli composition corresponding to relational composition. Trivial examples of dominances are

$$d_1 \stackrel{\text{def}}{=} \text{isContr}, \quad d_2(X) \stackrel{\text{def}}{=} (X = 0) + (X = 1), \quad d_\Omega \stackrel{\text{def}}{=} \text{isProp},$$

with total spaces 1, 2, and Ω, where $2 \simeq 1 + 1$ and $\Omega \stackrel{\text{def}}{=} \Sigma(P : \mathcal{U}), \text{isProp}\,P$, which satisfy

$$\mathcal{L}_{d_1}(X) \simeq X, \quad \mathcal{L}_{d_2}(X) \simeq X + 1\ , \quad \mathcal{L}_{d_\Omega}(X) = \mathcal{L}\,X.$$

(And hence $\mathcal{L}\,X \simeq X + 1$ iff the canonical map $2 \to \Omega$ is an equivalence iff excluded middle holds.)

2.8 The Rosolini dominance and choice

A dominance often considered in topos theory, typically in realizability toposes, is the *Rosolini dominance* consisting of the Rosolini propositions defined above [20, 11]. Then the statement in the internal language that "all partial functions $\mathbb{N} \rightharpoonup \mathbb{N}$ whose values have Rosolini extents of definition are computable" is valid in the effective topos [12, 20] and hence consistent in a dependent type theory with function extensionality, propositional truncations *and some amount of choice*. Countable choice, which holds in the effective topos assuming a classical meta-theory for its study, is used to prove that the Rosolini propositions form a dominance [20, Chapter 5.2].

We work with the axiom of choice in the form "a product of inhabited sets is inhabited" [23]:

$$(\Pi(x : X), \|Y(x)\|) \to \|\Pi(x : X), Y(x)\|\,,$$

for a set $X : \mathcal{U}$ and a family of sets $Y : X \to \mathcal{U}$. When the set X is the type of natural numbers, this is *countable choice*. When X ranges over propositions, this is *propositional choice*, which is shown in [14] to be equivalent to

$$(P \to \|A\|) \to \|P \to A\|\,,$$

where P ranges over propositions and A over sets. Notice that propositional choice holds for decidable P, and hence excluded middle implies propositional choice. But the decidability of Rosolini propositions is known as LPO (the limited principle of omniscience), and is not constructively provable in any variety of constructive mathematics [6]. However, it is known that excluded middle doesn't imply countable choice, at least not in topos logic.

▶ **Theorem 2.** *For any set A, the following are equivalent:*
1. *Countable choice for families $Y(n) \stackrel{def}{=} (\alpha(n) = 1) \to A$ with $\alpha : \mathbb{N} \to 2$.*
2. *Propositional choice from Rosolini propositions to A.*

Proof. Consider the following propositions:
1. $\langle\alpha\rangle \to \|A\|$,
2. $(\Sigma(n : \mathbb{N}), \alpha(n) = 1) \to \|A\|$,
3. $\Pi(n : \mathbb{N}), ((\alpha(n) = 1) \to \|A\|)$,
4. $\Pi(n : \mathbb{N}), \|(\alpha(n) = 1) \to A\|$,
5. $\|\Pi(n : \mathbb{N}), (\alpha(n) = 1) \to A\|$,
6. $\|(\Sigma(n : \mathbb{N}), \alpha(n) = 1) \to A\|$,
7. $\|\langle\alpha\rangle \to A\|$.

The implication (4) → (5) is the above instance of countable choice, and the implication (1) → (7) is the above instance of propositional choice. Hence the chain of implications (1) → (2) → (3) → (4) → (5) → (6) → (7) gives propositional choice from countable choice, and the chain of implications (4) → (3) → (2) → (1) → (7) → (6) → (5) gives countable choice from propositional choice. ◀

Hence this particular case of countable choice is implied by excluded middle, and in fact already by LPO, because propositional choice for Rosolini propositions is. To show that the Rosolini propositions form a dominance, it is necessary and sufficient to have choice from Rosolini propositions P to *Rosolini structures* A, namely types of the form

$$\text{RosoliniStructure}(Q) \stackrel{\text{def}}{=} \Sigma(\alpha : \mathbb{N} \to 2), \text{isProp}\langle\alpha\rangle \times (Q = \langle\alpha\rangle)$$

with $Q : \mathcal{U}$. Notice that the type of Rosolini structures is a subtype of the *one-point compactification of* $\mathbb{N}$, namely the type $\mathbb{N}_\infty \stackrel{\text{def}}{=} \Sigma(\alpha : \mathbb{N} \to 2), \text{isProp}\langle\alpha\rangle$.

▶ **Theorem 3.** *The following are equivalent:*
1. *Choice from Rosolini propositions to Rosolini structures.*
2. *The Rosolini propositions form a dominance.*

Proof. (⇓): It suffices to show that, for any $\alpha : \mathbb{N} \to 2$ with $\text{isProp}\langle\alpha\rangle$ and $Q : \mathcal{U}$, we have $(\langle\alpha\rangle \to \text{isRosolini}\, Q) \to \text{isRosolini}(\langle\alpha\rangle \times Q)$, which amounts, by propositional univalence, to

$$(\langle\alpha\rangle \to \|\Sigma(\beta : \mathbb{N} \to 2), \text{isProp}\langle\beta\rangle \times (Q = \langle\beta\rangle)\|) \to \|\Sigma(\gamma : \mathbb{N} \to 2), (\langle\alpha\rangle \times Q) \iff \langle\gamma\rangle\|\,,$$

and hence, by Rosolini propositional choice, to

$$\|\langle\alpha\rangle \to \Sigma(\beta : \mathbb{N} \to 2), \text{isProp}\langle\beta\rangle \times (Q = \langle\beta\rangle)\| \to \|\Sigma(\gamma : \mathbb{N} \to 2), (\langle\alpha\rangle \times Q) \iff \langle\gamma\rangle\|\,,$$

and so, because truncation is functorial, it suffices to show that

$$(\langle\alpha\rangle \to \Sigma(\beta : \mathbb{N} \to 2), \text{isProp}\langle\beta\rangle \times (Q = \langle\beta\rangle)) \to \Sigma(\gamma : \mathbb{N} \to 2), (\langle\alpha\rangle \times Q) \iff \langle\gamma\rangle.$$

Assume an element ϕ of the premise, and define $\beta : \langle\alpha\rangle \to (\mathbb{N} \to 2)$ by $\beta(i, p) = \text{pr}_1(\phi(i, p))$. Then define $\gamma(n) = 1 \iff \Sigma(i, j \leq n), ((i = n) + (j = n)) \times \Sigma(p : \alpha(i) = 1), \beta(i, p)(j) = 1$. We omit the verification that $\text{isProp}\langle\gamma\rangle$ and that $(\langle\alpha\rangle \times Q) \iff \langle\gamma\rangle$.

(⇑): Assume $P \to \|\text{RosoliniStructure}\, Q\|$ for some Rosolini proposition P and some proposition Q, that is, $P \to \text{isRosolini}\, Q$. By the dominance axiom, $\text{isRosolini}(P \times Q)$. Because $P \to ((P \times Q) = Q)$, by propositional univalence, we have

$$\text{RosoliniStructure}(P \times Q) \to (P \to \text{RosoliniStructure}(Q)).$$

By functoriality of truncation, this gives $\text{isRosolini}(P \times Q) \to \|P \to \text{RosoliniStructure}(Q)\|$. Because the premise holds, $\|P \to \text{RosoliniStructure}(Q)\|$, as required. ◀

But it doesn't seem to be possible to replace countable choice by its weakening discussed above in Theorem 4 and Corollary 5 below. If we order propositions by $(P \leq Q) \stackrel{\text{def}}{=} (P \to Q)$, then any family of propositions has a least upper bound, given by the truncation of its sum.

▶ **Theorem 4.** *Assuming countable choice, the Rosolini propositions are closed under countable joins in the complete lattice of all propositions.*

Proof. We have to show that $\|\Sigma_i, P_i\|$ is a Rosolini proposition for any countable family P_i of Rosolini propositions. Countable choice gives

$$\|\Sigma(\alpha : \mathbb{N} \times \mathbb{N} \to 2), \Pi(i : \mathbb{N}), \mathrm{isProp}\langle\alpha(i,-)\rangle \times (P_i = \langle\alpha(i,-)\rangle)\|\,.$$

The last equation amounts to $P_i = (\Sigma_j, \alpha(i,j) = 1)$, which gives $(\Sigma_i, P_i) = (\Sigma_{i,j}, \alpha(i,j) = 1)$ and hence $\|\Sigma_i, P_i\| = \|\Sigma_{i,j}, \alpha(i,j) = 1\|$. Now if we define $\beta(n) = 1$ iff there is a minimal pair $i, j \leq n$ in the lexicographic order with $\alpha(i,j) = 1$, then β satisfies $\mathrm{isProp}\langle\beta\rangle$ and $\langle\beta\rangle \iff \|\Sigma_{i,j}, \alpha(i,j) = 1\|$, which shows that $\|\Sigma_i, P_i\|$ is a Rosolini proposition, as required. ◀

▶ **Corollary 5.** *If Y is a set, then countable choice implies that the Rosolini lifting of Y has joins of ascending sequences.*

Proof. The construction is as that of Theorem 1, with the additional step of showing that $\|\Sigma_i, \mathrm{extent}(u_i)\|$ is a Rosolini proposition, which is given by Theorem 4. ◀

However, Coquand, Mannaa, and Ruch [10] have shown that countable choice cannot be proved in dependent type theory with one univalent universe and propositional truncation, and discussions in research mailing lists for homotopy type theory and constructive mathematics with them and with Andrew Swan indicate that their negative result also applies to our weakening of countable choice.

2.9 Size matters

We now discuss the universe levels of the above constructions (which we have checked using Agda). We assume we have a function $d : \mathcal{U}_0 \to \mathcal{U}_0$, for example $d = \mathrm{isProp}$ or $d = \mathrm{isRosolini}$. We decorate the above constructions with universe levels j, k, l as superscripts. For a type $Y : \mathcal{U}_j$, we have $\mathcal{L}_d(Y) : \mathcal{U}_{\max(j,1)}$, so that lifting has the following type:

$$\mathcal{L}_d^j : \mathcal{U}_j \to \mathcal{U}_{\max(j,1)}.$$

In particular, we have $\mathcal{L}^0 : \mathcal{U}_0 \to \mathcal{U}_1$, and $\mathcal{L}^1 : \mathcal{U}_1 \to \mathcal{U}_1$, and $\mathcal{L}^{j+1} : \mathcal{U}_{j+1} \to U_{j+1}$, where we have elided the subscript d to avoid notational clutter. That is, lifting raises the universe level only at the lowest universe. With propositional resizing [25], assuming $\mathrm{isProp}(d(P))$, we would get that $\mathcal{L}^0 : \mathcal{U}_0 \to \mathcal{U}_0$ and hence $\mathcal{L}^j : \mathcal{U}_j \to U_j$ for every j so that universe levels are not raised. But resizing is not needed for our purposes, if we are willing to work with higher universes, and $\mathcal{U}_1$ suffices for our discussion of computation of partial functions in Section 3.

For the unit we have the type $\eta^j : \Pi(X : \mathcal{U}_j), X \to \mathcal{L}^j(X)$. Assuming that $d : \mathcal{U}_0 \to \mathcal{U}_0$ is closed under Σ in the sense discussed above, we get an extension operator $(-)^\sharp$ of type

$$\Pi(X : \mathcal{U}_j, Y : \mathcal{U}_k), (X \to \mathcal{L}^k(Y)) \to (\mathcal{L}^j(X) \to \mathcal{L}^k(Y)),$$

which for $j = k = 0$ specializes to $\Pi(X, Y : \mathcal{U}_0), (X \to \mathcal{L}^0(Y)) \to (\mathcal{L}^0(X) \to \mathcal{L}^0(Y))$. From this we get a Kleisli composition operator of type

$$\Pi(X : \mathcal{U}_j)(Y : \mathcal{U}_k)(Z : \mathcal{U}_l), (Y \to \mathcal{L}^l(Z)) \to (X \to \mathcal{L}^k(Y)) \to (X \to \mathcal{L}^l(Z)),$$

which specializes to $\Pi(X, Y, Z : \mathcal{U}_0), (Y \to \mathcal{L}^0(Z)) \to (X \to \mathcal{L}^0(Y)) \to (X \to \mathcal{L}^0(Z))$. With the general universe levels, the equations for Kleisli triples can be written down (they type check) and proved. For the multiplication induced by the extension operator, we have the type

$$\mu^j : \Pi(X : \mathcal{U}_j), \mathcal{L}^{\max(j,1)}(\mathcal{L}^j(X)) \to \mathcal{L}^j(X)$$

which specializes to $\Pi(X : \mathcal{U}_0), \mathcal{L}^1(\mathcal{L}^0(X)) \to \mathcal{L}^0(X)$ and $\Pi(X : \mathcal{U}_1), \mathcal{L}^1(\mathcal{L}^1(X)) \to \mathcal{L}^1(X)$. Again the monad laws, decorated with general universe levels, can be written down and proved. Of course, we don't have a monad because the universe levels give the above non-standard types for the unit and multiplication. Notice also that, because types don't form a category, but something like an ∞-category, a more refined notion of monad would be needed, even ignoring the issue with universe levels. In any case, our unrefined version is enough to get partial functions $(-) \to \mathcal{L}(-)$ with a natural Kleisli composition operation induced by the monad-like structure.

We observe that by the results of [18], if our type theory includes *graph quotients*, then the total space of $d \stackrel{\text{def}}{=} \text{isRosolini}$ is small, as it is the image of a function $\mathbb{N}_\infty \to \mathcal{U}_0$ into the locally small type $\mathcal{U}_0$, so that lifting, extension, unit and multiplication preserve universe levels even at the lowest universe.

3 Computable functions

We now demonstrate a way to develop computability theory constructively, and then in Section 3.5 relate this to our previous discussion of partial functions and dominances. A main point of this redevelopment is to make sure we don't use countable choice or Markov's principle, in addition to non-constructive principles. While Turing machines provide an intuitively convincing notion of computation and are easy to define in a constructive setting, they are cumbersome to work with directly. We will instead work with a model which is a modest abstraction of the notion of Turing machine.

3.1 Primitive recursive functions

Our model will make use of (unary) primitive recursive functions, so we quickly recall a few basic facts about primitive recursive functions. The classical presentation of the results mentioned here (such as that in [13], [19], or [15]) can be immediately adapted to fit in univalent type theory. The key point is that the functions used are all computable and total, so there's no place where non-constructive notions are likely to appear.

▶ **Definition 6.** The type family $\text{PR} : \mathbb{N} \to \mathcal{U}$ of *primitive recursive combinators (of arity n)* is defined inductively by

1. $\text{s} : \text{PR}_1$;
2. for any $n : \mathbb{N}$ and $k < n$ we have $\text{p}^n_k : \text{PR}_n$;
3. for any $n, k : \mathbb{N}$ we have $\text{c}^n_k : \text{PR}_n$;
4. if $f : \text{PR}_n$ and $g_i : \text{PR}_m$ for each $0 < i \leq n$, then $f\langle g_1, \ldots, g_n\rangle : \text{PR}_m$.
5. if $f : \text{PR}_{n+2}$ and $g : \text{PR}_n$, then $\text{r}_{f,g} : \text{PR}_{n+1}$.

We can define a function $\text{eval}_n : \text{PR}_n \to \mathbb{N}^n \to \mathbb{N}$ in the obvious way, such that $\text{eval}(t)$ is primitive recursive (in the usual sense) for each PR term t. We write $\text{isPR}_n(f)$ if $f : \mathbb{N}^n \to \mathbb{N}$ is in the image of eval, and say that f is primitive recursive if $\text{isPR}(f)$.

▶ **Theorem 7** (Enumeration Theorem for Primitive Recursive functions)**.** *For each $n : \mathbb{N}$, there is an injective function $e : \mathrm{PR}_n \to \mathbb{N}$, and a function $\{-\} : \mathbb{N} \to (\mathbb{N}^n \to \mathbb{N})$ such that for all $f : \mathrm{PR}_n$ and $x : \mathbb{N}^n$,*

$$\{e_f\}(x) = \mathrm{eval}(f, x).$$

A *primitive recursive characteristic function* for a predicate $R : \mathbb{N}^n \to \Omega$ is a primitive recursive function $\chi_R : \mathbb{N} \to \mathbb{N}$ such that for all $x : \mathbb{N}^n$

$$\chi_R(x) = 1 \Leftrightarrow R(x) \wedge \chi_R(x) = 0 \Leftrightarrow \neg R(x).$$

We will write $\mathrm{PR}(R)$ for the type of p.r. characteristic functions of R, and say that R is *primitive recursive* if $\mathrm{PR}(R)$. As $\mathrm{isPR}(f)$ is a proposition for any $f : \mathbb{N}^n \to \mathbb{N}$, so is $\mathrm{PR}(R)$. Observe that if R has a primitive recursive characteristic function, then R is complemented.

Many of our arguments will make use of functions with arity greater than 1, but we only use unary primitive recursive functions in what follows, so we will make use of primitive recursive pairing functions $\langle - \rangle : \mathbb{N}^n \to \mathbb{N}$, with primitive recursive projections. We will tacitly use the fact that if $f : \mathbb{N}^n \to \mathbb{N}$ is primitive recursive, then there is a primitive recursive function $\hat{f} : \mathbb{N} \to \mathbb{N}$ such that

$$\hat{f}\langle x_1, \ldots, x_n \rangle = f(x_1, \ldots, x_n).$$

Similarly, we will treat functions $\mathbb{N} + \mathbb{N} \to \mathbb{N}$ as functions $\mathbb{N} \to \mathbb{N}$, using a fixed bijection $\mathbb{N} + \mathbb{N} \simeq \mathbb{N}$ which makes the encodings of the inclusions inl and inr primitive recursive.

3.2 Abstract Turing machines

Our model abstracts away the details of the initialization and transition functions of a Turing machine, giving us the following definition.

▶ **Definition 8.** An *abstract Turing machine* is a pair (i, s) of functions $i : \mathbb{N} \to \mathbb{N}$ and $s : \mathbb{N} \to \mathbb{N} + \mathbb{N}$. The function i is called the *initialization function* and s is called the *transition function*. A *primitive recursive machine*, or recursive machine for short, is a pair $(i, s) : \mathrm{PR}_1 \times \mathrm{PR}_1$.

ATM is the type of abstract Turing machines, and RM is the type of recursive machines. We will treat recursive machines as ATMs by implicitly using the map defined by evaluating the combinators and composing along our fixed bijection $\mathbb{N} \to \mathbb{N} + \mathbb{N}$.

We can evaluate abstract Turing machines via a function $\mathrm{eval} : \mathrm{ATM} \to (\mathbb{N} \to \mathcal{L}\,\mathbb{N})$ as follows: Given $(i, s) : \mathrm{ATM}$, let $s' : \mathbb{N} + \mathbb{N} \to \mathbb{N} + \mathbb{N}$ be the function

$$s'(\mathrm{inl}\, x) = s(x),$$
$$s'(\mathrm{inr}\, y) = \mathrm{inr}\, y.$$

Now define for each $k : \mathbb{N}$ the function $\mathrm{run}_k : \mathrm{ATM} \to \mathbb{N} \to (\mathbb{N} + \mathbb{N})$ by

$$\mathrm{run}_k(m, x) \stackrel{\mathrm{def}}{=} s'^k(\mathrm{inl}(i(x)))$$

And so we have for fixed m,

$$R_m(x, y) \stackrel{\mathrm{def}}{=} \exists (k : \mathbb{N}), \mathrm{run}_k(m, x) = \mathrm{inr}\, y.$$

Then $\mathrm{eval}(m)$ is the partial function with R_m as its graph.

The partial function $\mathrm{eval}(m)$ is the function *computed by* m. We will sometimes abuse notation and write $m(x)$ for $\mathrm{eval}(m, x)$. For a partial function f, we let $\mathrm{computationalStructure}(f)$ be the type of recursive machines computing f, and $\mathrm{isComputable}(f)$ be the propositional truncation of $\mathrm{computationalStructure}(f)$. We let Comp be the type of computable functions:

$$\mathrm{Comp} \stackrel{\mathrm{def}}{=} \Sigma(f : \mathbb{N} \rightharpoonup \mathbb{N}), \mathrm{isComputable}(f).$$

The distinction between $\mathrm{computationalStructure}(f)$ and $\mathrm{isComputable}(f)$ is necessary. By the enumeration theorem for primitive recursive functions, we have a map

$$(\Sigma(f : \mathbb{N} \rightharpoonup \mathbb{N}), \mathrm{computationalStructure}(f)) \to \mathbb{N},$$

given by $(i, s) \mapsto \langle e_i, e_s \rangle$, which is easily seen to be an embedding.

On the other hand, if there is an embedding $F : \mathrm{Comp} \to \mathbb{N}$, then equality in Comp is complemented, since we have for $f, g : \mathrm{Comp}$ that $f = g \simeq F(f) = F(g)$, and equality between naturals is complemented. In particular, we would have a way to determine whether a (computable) function is equal to $\lambda x.0$, so we would have WLPO for computable functions. This means that we also cannot even expect to have an embedding

$$\mathrm{Comp} \to \Sigma(f : \mathbb{N} \rightharpoonup \mathbb{N}), \mathrm{computationalStructure}(f).$$

That is, we have no way of finding a program for a function just by knowing one exists.

As Turing machines can be encoded so that the initialization and transition functions are primitive recursive, every Turing machine is a recursive machine; conversely, given primitive recursive functions i and s, we can construct a Turing machine that runs i, and then iteratively runs s.

A simple programming exercise shows that if m and n are abstract Turing machines computing f and g respectively, then there is a machine $m; n$ computing $g \circ f$. Moreover, it is easy to see that when m and n are recursive machines, then so is $m; n$. Using the universal property of truncations this gives us,

$$\Pi(f, g : \mathbb{N} \rightharpoonup \mathbb{N}), \mathrm{isComputable}(f) \to \mathrm{isComputable}(g) \to \mathrm{isComputable}(g \circ f).$$

We have a minimization operator for partial functions, $\mu : (\mathbb{N} \to \mathcal{L}\,\mathbb{N}) \to \mathcal{L}\,\mathbb{N}$, with

$$\mathrm{extent}(\mu f) \stackrel{\mathrm{def}}{=} \Sigma(k : \mathbb{N}), f(k) = \eta 0 \times \Pi(j < k), \Sigma(n : \mathbb{N}), f(j) = \eta(n + 1),$$

and $\mathrm{value}(\mu f) \stackrel{\mathrm{def}}{=} \mathrm{pr}_1$. More generally, we may define the minimization $\mu y.R(y) : \mathcal{L}\,\mathbb{N}$ of a (decidable) predicate $R : \mathbb{N} \to \Omega$

$$\mathrm{extent}(\mu y.R(y)) \stackrel{\mathrm{def}}{=} \Sigma(y : \mathbb{N}), R(y) \times \Pi(x < y), \neg R(x).$$

and $\mathrm{value}(\mu y.R(y)) \stackrel{\mathrm{def}}{=} \mathrm{pr}_1$. When $R : \mathbb{N}^2 \to \Omega$ is primitive recursive, it is easy to see that $\mathrm{isComputable}(\lambda x.\mu y.R(x, y))$, by checking $R(x, n)$ for each n. Via dovetailing (by using run_k instead of eval), we can make this work with any *recursive* predicate, as expected from classical recursion theory.

3.3 Basic Recursion Theory

We are now able to show how to develop basic recursion theory via recursive machines. We roughly follow the presentation in Odifreddi [15]; in fact, the proofs given by Odifreddi for the results stated here translate to our framework with only minor adjustment. We begin with Kleene's Normal Form theorem.

▶ **Theorem 9** (Kleene's Normal Form Theorem). *There is an injection* $e : \mathrm{RM} \to \mathbb{N}$*, a primitive recursive predicate* T*, and a primitive recursive function* U *such that for any recursive machine* m *and* $x : \mathbb{N}$ *we have*

$$\mathrm{eval}(m, x) = (\Sigma(y : \mathbb{N}), \mathrm{T}(e_m, x, y)\ ,\ -\ ,\ \mathrm{U} \circ \mathrm{pr}_1)\,.$$

This says that $\mathrm{eval}(m, x)$ is defined iff there is a y such that $\mathrm{T}(e_m, x, y)$, and that when there is such a y, the value of $\mathrm{eval}(m, x)$ is $\mathrm{U}(y)$. As for any $e, x : \mathbb{N}$ there is a unique y such that $\mathrm{T}(e, x, y)$, we actually have that $\Sigma(y : \mathbb{N}), \mathrm{T}(e_m, x, y)$ is the extent of definition of $\mu y.\, \mathrm{T}(e_m, x, y)$, and we get the usual statement of the Normal Form Theorem.

▶ **Corollary 10.** *For any recursive machine* m *and* $x : \mathbb{N}$*, we have*

$$\mathrm{eval}(m, x) = \mathcal{L}\,\mathrm{U}(\mu y.\, \mathrm{T}(e_m, x, y)).$$

This result gives us the Kleene-bracket function $\{-\} : \mathbb{N} \to (\mathbb{N} \to \mathcal{L}\,\mathbb{N})$, defined by

$$\{e\}(x) \stackrel{\mathrm{def}}{=} \mathcal{L}\,\mathrm{U}(\mu y.\, \mathrm{T}(e, x, y)).$$

In order to be able to apply $\{-\}$ to a partial natural number, let $\{-\}^* : \mathcal{L}\,\mathbb{N} \to (\mathbb{N} \to \mathcal{L}\,\mathbb{N})$ be the function obtained by Kleisli extension in the first component.

Rogers' Fixed Point Theorem and Kleene's Second Recursion Theorem can now be proved in the standard way; observe, however, that the precise statement of Rogers' result must be modified to make sense in our setting, since the partial functions obtained from recursive machines (even total ones) have type $\mathbb{N} \to \mathcal{L}\,\mathbb{N}$. The proofs will require the S_n^m Theorem:

▶ **Theorem 11** (S_n^m Theorem). *There is a primitive recursive* $s : \mathbb{N}^2 \to \mathbb{N}$ *such that for all* $e, x, y : \mathbb{N}$ *we have* $\{s(e, x)\}y = \{e\}\langle x, y\rangle$*.*

▶ **Theorem 12** (Rogers' fixed point theorem). *For any recursive machine* m *such that* $\mathrm{eval}(m)$ *is total,* $\Sigma(n : \mathbb{N}), \{n\} = \{m(n)\}^*$*.*

Proof. Let $g : \mathbb{N} \to \mathbb{N}$ be defined by

$$g\langle x, y\rangle \stackrel{\mathrm{def}}{=} \{\{x\}(x)\}^*(y).$$

As this has a recursive machine, we have a code e_g. Defined $h(x) = s(e_g, x)$. Now let $e = e_{\mathrm{eval}(m)\circ h}$ and $n = h(e)$. Then we have

$$\{n\}y = \{h(e)\}y = \{s(e_g, e)\} = \{e_g\}\langle e, y\rangle = \{\{e\}(e)\}^*(y) = \{m(he)\}^*(y) = \{m(n)\}^*(y).$$ ◀

▶ **Theorem 13** (Kleene's Second Recursion Theorem). *For any recursive machine* m*,*

$$\Sigma(p : \mathbb{N}), \Pi(y : \mathbb{N}), (\{p\}(y) = \mathrm{eval}(m, \langle p, y\rangle))\,.$$

Proof. Let $f : \mathbb{N} \to \mathbb{N}$ be defined by

$$f(x) \stackrel{\mathrm{def}}{=} s\langle e_m, x\rangle.$$

This function f is primitive recursive, so there is a total recursive machine m computing f. By Rogers' fixed point theorem, there is then a p such that $\{p\} = \{m(p)\}^*$. Then,

$$\{p\}(y) = \{m(p)\}^*(y) = \{f(p)\}(y) = \{s\langle e_m, p\rangle\}(y) = m\langle p, y\rangle.$$ ◀

These two theorems have versions from the perspective of computability as property which follow as corollaries.

▶ **Corollary 14.** *For any total computable function* $f : \mathbb{N} \to \mathcal{L}\,\mathbb{N}$, $\exists(n : \mathbb{N}), \{n\} = \{f(n)\}^*$.

▶ **Corollary 15.** *For any computable function* $f : \mathbb{N} \to \mathcal{L}\,\mathbb{N}$,

$$\exists(p : \mathbb{N}), \Pi(y : \mathbb{N}), (\{p\}(y) = f(\langle p, y\rangle))\,.$$

A subset A of $\mathbb{N}$ is *recursive* if its characteristic function $\mathbb{N} \to \Omega$ factors through the embedding $2 \to \Omega$ via a computable function $\mathbb{N} \to 2$. It is *recursively enumerable* if it is the image of some computable function $f : \mathbb{N} \to \mathbb{N}$. As with primitive recursive predicates, any recursive set is complemented. We get the following form of a classical result.

▶ **Theorem 16.** *A subset* A *of* $\mathbb{N}$ *is recursive if and only if it is complemented and both* A *and its complement are recursively enumerable.*

Likewise we have the classical characterization of recursively enumerable sets.

▶ **Theorem 17.** *A subset of* $\mathbb{N}$ *is r.e. iff it is the domain of a computable function.*

A slightly better classical characterization requires some modification to be true constructively.

▶ **Theorem 18.** *If* A *is an inhabited subset of* $\mathbb{N}$*, then the following are equivalent.*

1. *A is the domain of a computable partial function.*
2. *A is the image of a computable partial function.*
3. *A is the image of a computable total function.*
4. *A is the image of a primitive recursive function.*

3.4 Not all partial functions are computable

As discussed in Section 2.8, it is consistent in topos logic that every Rosolini partial function is computable. However, the statement that every partial function $\mathbb{N} \to \mathcal{L}\,\mathbb{N}$ is computable is provably false in univalent type theory. In fact, if the partial function $f : \mathbb{N} \to \mathcal{L}\,\mathbb{N}$ defined by

$$\begin{aligned} \text{extent}\, f(x) &\stackrel{\text{def}}{=} (\{x\}(x) = \bot), \\ \text{value}(f(x))(p) &\stackrel{\text{def}}{=} 0 \end{aligned}$$

were computable, this would mean that the complement of the Halting Set is recursively enumerable.

3.5 Dominances and semidecidable propositions

We now compare Rosolini and semidecidable propositions, using the following technical lemma.

▶ **Lemma 19.** *If* X *is a type and* $P : X \to \mathcal{U}$ *is a family of types over* X *such that* $\text{isProp}\,(\Sigma(x : X), \|P(x)\|)$*, then* $(\Sigma(x : X), \|P(x)\|) = \|\Sigma(x : X), P(x)\|$.

Proof. By propositional extensionality, it is enough to construct functions in both directions. Right to left follows from the universal property of truncations. For the other, we have

$$((\Sigma(x : X), \|P(x)\|) \to \|\Sigma(x : X), P(x)\|) \simeq \Pi(x : X), (\|P(x)\| \to \|\Sigma(x : X), P(x)\|)\,.$$

and functoriality of truncation gives us an inhabitant of the right-hand type. ◀

Recall the relation R_m from Section 3.2 used in the definition of $\mathrm{eval}(m)$.

▶ **Theorem 20.** *For any machine $m : \mathrm{ATM}$ and any $x, y : \mathbb{N}$, the types $R_m(x, y)$ and $\Sigma(y : \mathbb{N}), R_m(x, y)$ are Rosolini propositions.*

Proof. For the first type, let $g : \mathbb{N} + \mathbb{N} \to 2$ be the function which takes value 1 on $\mathrm{inr}\, y$ and 0 otherwise. Define $\alpha_k \stackrel{\mathrm{def}}{=} g(\mathrm{run}_k(m, x))$. We have

$$(\alpha_k = 1) \simeq (g(\mathrm{run}_k(m, x)) = y),$$

so $R_m(x, y) \simeq \exists(k : \mathbb{N}), \alpha_k = 1$.

For the second type, let $h : \mathbb{N} + \mathbb{N} \to 2$ be the function which is 0 on $\mathrm{inl}\, j$ and 1 on $\mathrm{inr}\, j$ for all j, and again take $\alpha_k = h(\mathrm{run}_k(m, x))$. We have that

$$(\alpha_k = 1) \simeq (\Sigma(y : \mathbb{N}), \mathrm{run}_k(m, x) = \mathrm{inr}\, y).$$

Summing over all $k : \mathbb{N}$, rearranging and truncating takes us to

$$(\exists(k : \mathbb{N}), \alpha_k = 1) \simeq (\exists(y : \mathbb{N}), \Sigma(k : \mathbb{N}), \mathrm{run}_k(m, x) = \mathrm{inr}\, y).$$

By our technical lemma, we then have

$$(\exists(k : \mathbb{N}), \alpha_k = 1) \simeq \Sigma(y : \mathbb{N}), \exists(k : \mathbb{N}), \mathrm{run}_k(m, x) = \mathrm{inr}\, y. \qquad \blacktriangleleft$$

Moreover, the semidecidable propositions are exactly those which arise as the value of a program. That is,

▶ **Theorem 21.** *For all $A : \mathcal{U}$*

$$\mathrm{isSemiDecidable}(A) \iff \exists(f : \mathbb{N} \rightharpoonup \mathbb{N}), \mathrm{isComputable}(f) \times (A = \mathrm{extent}(f(0))).$$

Proof. Suppose we are given a function $f : \mathbb{N} \rightharpoonup \mathbb{N}$ with a recursive machine t such that $A = \mathrm{extent}(f(0))$. We may define

$$\alpha(n) = \begin{cases} 0 & \text{if } \mathrm{run}_n(t, 0) = \mathrm{inr}\, k \text{ for some } k, \text{ and } \alpha(m) = 1 \text{ for } m < n, \\ 1 & \text{otherwise.} \end{cases}$$

It is easy to see that α is recursive, that $\langle \alpha \rangle$ is a proposition and that $A = \langle \alpha \rangle$.

Conversely, suppose we are given a total recursive α. Consider the constant function $f(x) = \mu k.(\alpha(k) = 0)$. Since $\alpha(k) = 0$ is recursive, this is recursive and it is clear that $\mathrm{extent}(f(0)) = A$. The result follows from the universal property of truncations. $\blacktriangleleft$

4 Related work, discussion, conjectures, questions and further work

Capretta [7] introduced a certain *delay monad*, and Chapman, Uustalu and Veltri [8] considered its quotient by weak bisimilarity. In order to show that the result is again a monad, they used countable choice. This appearance of choice should be no accident: we conjecture that the quotient constructed in [8] is equivalent to the Rosolini lifting, which, as we saw, also requires some amount of countable choice to be a monad. As we have seen, countable choice is also needed to show that the Rosolini lifting is closed under countable ascending joins.

As discussed in the introduction, we would like to find a notion of partial function which can be seen to be closed under composition (the dominance axiom (D5)), without choice,

before we attempt to consistently postulate that all such partial functions from $\mathbb{N}$ to $\mathbb{N}$ are computable.

Altenkirch, Danielsson and Kraus [21] consider a higher inductive-inductive definition of a partiality monad with a constructor that closes under countable ascending joins, without choice, and show that, in the presence of countable choice, their construction is equivalent to the above construction [8], and hence would also be equivalent to the Rosolini lifting if the above conjecture is true. It is not hard to see that this construction can be equivalently described as the lifting induced by the dominance obtained by closing the Rosolini propositions under Σ and countable ascending joins.

However, this doesn't work for our purposes. In fact, consider again the Kleene-bracket function $\{-\} : \mathbb{N} \to (\mathbb{N} \to \mathcal{L}\,\mathbb{N})$ that associates a partial function to a Gödel number of a Turing machine. As discussed above, the extents of definition of the partial functions in the image of the Kleene-bracket are precisely the semidecidable propositions, which form a *subset* of the Rosolini propositions. Thus, enlarging the set of Rosolini propositions doesn't help.

Also, it doesn't seem to be possible to show that the subset consisting of the semidecidable propositions form a dominance without choice as above. Nevertheless, without any form of choice, we have that the partial functions in the image of Kleene-bracket, namely the computable functions, *are* closed under composition. It seems that even for $d = \mathrm{isSemiDecidable}$, in the absence of enough choice, the function space $\mathbb{N} \to \mathcal{L}_d\,\mathbb{N}$ may in principle contain more than the computable partial functions $\mathbb{N} \to \mathcal{L}\,\mathbb{N}$, as in Section 3.4 above. This motivates the following considerations.

The function $D \stackrel{\text{def}}{=} \mathrm{RosoliniStructure}$ satisfies the dominance axioms with the exception of (D2), where (D4) and (D5) then become structure rather than merely property. Crucially, a function (D5) that provides closure under Σ can be constructed without any form of choice. This induces a lift monad $\mathcal{L}_D$ with the same constructions discussed in Section 2, which turns out to be equivalent to Capretta's delay monad. With $d \stackrel{\text{def}}{=} \mathrm{isRosolini}$, we have a map $\mathcal{L}_D(Y) \to \mathcal{L}_d(Y)$ that truncates structure, which in turn induces a function $(X \to \mathcal{L}_D(Y)) \to (X \to \mathcal{L}_d(Y))$ by post-composition. We refer to the maps in the image $D(X, Y)$ of this function as *disciplined*. Although the maps $X \to \mathcal{L}_d(Y)$ are not closed under Kleisli composition unless the Rosolini propositions form a dominance, we have that the disciplined ones are. Moreover, it is easy to see that the maps $\mathbb{N} \to \mathcal{L}_d(\mathbb{N})$ in the image of Kleene–Bracket are all disciplined. Perhaps it is consistent that the converse also holds, which amounts to saying that all disciplined maps $D(\mathbb{N}, \mathbb{N})$ are computable. In other words, we speculate that the statement "all disciplined partial functions $\mathbb{N} \rightharpoonup \mathbb{N}$ are Turing computable" is consistent with univalent type theory.

Acknowledgements. We thank Thierry Coquand and Andrew Swan for discussions about countable choice, Nicolai Kraus and Ian Orton for comments on a draft version, Mike Shulman for discussions about dominances, and Martin Hyland, John Longley and Pino Rosolini for discussions about dominances and the effective topos.

References

1 Andrej Bauer. First steps in synthetic computability theory. *Electronic Notes in Theoretical Computer Science*, 155:5 – 31, 2006.

2 Andrej Bauer and Davorin Lešnik. Metric spaces in synthetic topology. *Annals of Pure and Applied Logic*, 163(2):87 – 100, 2012. Third Workshop on Formal Topology. doi:http://dx.doi.org/10.1016/j.apal.2011.06.017.

3 Ana Bove. *General Recursion in Type Theory*, pages 39–58. Springer, 2003.

4 Ana Bove and Venanzio Capretta. *Nested General Recursion and Partiality in Type Theory*, pages 121–125. Springer, 2001.

5 Ana Bove and Venanzio Capretta. *A Type of Partial Recursive Functions*, pages 102–117. Springer, Berlin, Heidelberg, 2008.

6 Douglas Bridges and Fred Richman. *Varieties of constructive mathematics*, volume 97 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, 1987.

7 Venancio Capretta. General recursion via coinductive types. *Logical Methods in Computer Science*, 15:1–28, 2005.

8 James Chapman, Tarmo Uustalu, and Niccolò Veltri. *Quotienting the Delay Monad by Weak Bisimilarity*, pages 110–125. Springer, Cham, 2015.

9 Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. Cubical type theory: a constructive interpretation of the univalence axiom. arXiv, November 2016. URL: `https://arxiv.org/abs/1611.02108`.

10 Thierry Coquand, Bassel Mannaa, and Fabian Ruch. Stack semantics of type theory. arXiv, 2017. URL: `https://arxiv.org/abs/1701.02571`.

11 J. M. E. Hyland. *First steps in synthetic domain theory*, pages 131–156. Springer, 1991.

12 J Martin E Hyland. The effective topos. *Studies in Logic and the Foundations of Mathematics*, 110:165–216, 1982.

13 S. Kleene. Introduction to Metamathematics, 1952.

14 N. Kraus, M.H. Escardó, T. Coquand, and T. Altenkirch. Notions of anonymous existence in Martin-Löf type theory. *Logical Methods in Computer Science*, 2017.

15 Piergiorgio Odifreddi. *Classical Recursion Theory*. Elsevier, 1989.

16 B. Reus and Th. Streicher. *General synthetic domain theory — A logical approach (extended abstract)*, pages 293–313. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997. `doi:10.1007/BFb0026995`.

17 Egbert Rijke. Homotopy type theory. Master's thesis, Utrecht University, 2012. URL: `"https://hottheory.files.wordpress.com/2012/08/hott2.pdf"`.

18 Egbert Rijke. The join contruction. arXiv:1701.07538, Jan 2017. URL: `https://arxiv.org/abs/1701.07538`.

19 H. Rogers, Jr. *Theory of Recursive Functions and Effective Computability*. MIT Press, 1987.

20 G. Rosolini. *Continuity and Effectiveness in Topoi*. PhD thesis, University of Oxford, 1986. URL: `ftp://ftp.disi.unige.it/person/RosoliniG/papers/coneit.ps.gz`.

21 Nils Anders Danielsson Thorsten Altenkirch and Nicolai Kraus. Partiality, revisited: The partiality monad as a quotient inductive-inductive type. In *FoSSaCS Proceedings*, 2017.

22 A. S. Troelstra. A note on non-extensional operations in connection with continuity and recursiveness. *Indag. Math.*, 39(5):455–462, 1977.

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

24 Jaap van Oosten and Alex K. Simpson. Axioms and (counter)examples in synthetic domain theory. *Annals of Pure and Applied Logic*, 104(1):233 – 278, 2000. `doi:http://dx.doi.org/10.1016/S0168-0072(00)00014-2`.

25 V. Voevodsky. Resizing rules. Invited talk at Types'2011, Bergen, September 2011. URL: `http://www.math.ias.edu/vladimir/files/2011_Bergen.pdf`.