Constructive domain theory in Univalent Foundations

Tom de Jong

University of Birmingham, United Kingdom

8 April, 2020

UNIVERSITY OF BIRMINGHAM

Outline

1. Introduction

Outline

1. Introduction
2. Technical background
   - Univalent Foundations
     - Subsingletons and sets
     - Propositional truncation
     - Univalence
   - Constructivity and predicativity
   - Domain theory (classically)

Outline

1. Introduction
2. Technical background
   - Univalent Foundations
     - Subsingletons and sets
     - Propositional truncation
     - Univalence
   - Constructivity and predicativity
   - Domain theory (classically)
3. Our work
   - Predicative dcpos in UF
   - Scott model of PCF

Outline

Our aim and motivation

Develop domain theory, but constructively and predicatively in Univalent Foundations.

Our aim and motivation

Develop domain theory, but constructively and predicatively in Univalent Foundations.

Why domain theory?

- Classical topic in theoretical computer science
- Applications in:
    - semantics of programming languages
    - topology

Our aim and motivation

Develop domain theory, but constructively and predicatively in Univalent Foundations.

Why domain theory?

- Classical topic in theoretical computer science
- Applications in:
  - semantics of programming languages
  - topology

Why constructively and predicatively?

- More general
- Relevance in:
  - computer science (algorithm extraction)
  - pointfree/formal topology
- No constructive justification of impredicativity axioms (yet)

Our aim and motivation

Develop domain theory, but constructively and predicatively in Univalent Foundations.

Why Univalent Foundations?

- Implemented in proof assistants
- Constructive and predicative by default
- Novel and natural interpretation of mathematical equality

We could also extend our foundations with more higher inductive types, but so far, we haven't had any need for it.

By developing domain theory constructively in UF, we have also improved our understanding of the foundations themselves.

Further, domain theory serves as a testing ground for (formalisation in) UF.

Outline

Univalent Foundations

Intensional Martin-Löf Type Theory with:

- extensionality axioms
- propositional truncation

Vladimir Voevodsky

I will assume some familiarity with dependent type theory, e.g. $\Pi, \Sigma, +$-types.

Specifically, we need function extensionality (pointwise equal functions are equal) and propositional extensionality (logically equivalent propositions are equivalent) (and sometimes, univalence).

I will explain the propositional truncation shortly.

Univalent Foundations

Intensional Martin-Löf Type Theory with:

- extensionality axioms
- propositional truncation

Vladimir Voevodsky

Notation:

- For $x, y : X$, write $x = y$ for $\mathsf{Id}_X(x, y)$.
- Use $\equiv$ for judgemental equality.

I will assume some familiarity with dependent type theory, e.g. $\Pi, \Sigma, +$-types.

Specifically, we need function extensionality (pointwise equal functions are equal) and propositional extensionality (logically equivalent propositions are equivalent) (and sometimes, univalence).

I will explain the propositional truncation shortly.

Subsingletons and sets

Definition

A type X is a subsingleton (or proposition) if we have an element of

$$\text{is-a-prop}(X) :\equiv \prod_{x:X} \prod_{y:X} x = y.$$

There is a stratification of types in terms of the complexity of their identity types: Voevodsky's hlevels or truncation levels.

For this talk, we only need to consider two hlevels: the subsingletons and sets.

In a subsingleton, all elements are identified/equal. There is at most one element (up to $=$).
In a set, elements are identified/equal in at most one way.

Subsingletons and sets

Definition

A type X is a subsingleton (or proposition) if we have an element of

$$\text{is-a-prop}(X) :\equiv \prod_{x:X} \prod_{y:X} x = y.$$

Definition

A type X is a set if we have an element of

$$\text{is-a-set}(X) :\equiv \prod_{x:X} \prod_{y:X} \text{is-a-prop}(x = y).$$

There is a stratification of types in terms of the complexity of their identity types: Voevodsky's hlevels or truncation levels.

For this talk, we only need to consider two hlevels: the subsingletons and sets.

In a subsingleton, all elements are identified/equal. There is at most one element (up to $=$).
In a set, elements are identified/equal in at most one way.

Propositional truncation

For every type X, there is a *proposition* $\|X\|$ and a map $X \to \|X\|$, such that every map from X to a *proposition* P factors through it.

$$\begin{tikzcd} X \arrow[rr] \arrow[dr] & & P \\ & \|X\| \arrow[ur, dashed] & \end{tikzcd}$$

Borrowing terminology from category theory, we might call propositional truncation subsingleton reflection.

The dashed map is necessarily unique, because of function extensionality and the fact that P is a subsingleton.

The propositional truncation does *not* erase witnesses. (For instance: if A is a decidable predicate (i.e. proposition-valued family) on $\mathbf{N}$, then we have maps:

$$\left\| \sum_{n:\mathbf{N}} A(n) \right\| \to \sum_{k:\mathbf{N}} (k \text{ is the least } n : \mathbf{N} \text{ such that } A(n) \text{ holds}) \to \sum_{n:\mathbf{N}} A(n),$$

where the first map exists, because the second type may be shown to be a proposition and because A is decidable.)

What about the univalence axiom?

- The univalence axiom is an extensionality axiom for type universes.
- It implies function and propositional extensionality.
- Univalent Foundations is about much more than the univalence axiom!

We usually do not need full univalence, because the types under consideration are all propositions and sets (dcpos).

Arguably, univalent type theory is much more about the concept of truncation levels than about the univalence axiom.

Outline

Constructivity

Definition

Excluded middle (*EM*) in UF: $P + \neg P$ for all *propositions* P.

Constructivity

Definition

Excluded middle (*EM*) in UF: $P + \neg P$ for all *propositions* P.

Definition

Bishop's *Limited Principle of Omniscience* (*LPO*):

$$\prod_{\alpha : \mathbf{N} \to \mathbf{2}} \left(\left(\prod_{n : \mathbf{N}} \alpha(n) = 0 \right) + \left(\sum_{k : \mathbf{N}} k \text{ is least with } \alpha(k) = 1 \right) \right).$$

Constructivity

Definition

Excluded middle (*EM*) in UF: $P + \neg P$ for all *propositions* P.

Definition

Bishop's *Limited Principle of Omniscience* (*LPO*):

$$\prod_{\alpha : \mathbf{N} \to \mathbf{2}} \left(\left(\prod_{n : \mathbf{N}} \alpha(n) = 0 \right) + \left(\sum_{k : \mathbf{N}} k \text{ is least with } \alpha(k) = 1 \right) \right).$$

- EM implies LPO.
- LPO and EM are constructive taboos: they cannot be proved or disproved constructively.

Predicativity in Univalent Foundations

*Im*predicativity

The type of propositions in a universe $\mathcal{U}$

$$\Omega_{\mathcal{U}} :\equiv \sum_{P:\mathcal{U}} \text{is-a-prop}(P)$$

is (essentially) small, i.e. has an (equivalent) copy in $\mathcal{U}$.

Here $\simeq$ refers to Voevodsky's notion of (type) equivalence.

Predicativity in Univalent Foundations

***Im*predicativity**

The type of propositions in a universe $\mathcal{U}$

$$\Omega_{\mathcal{U}} :\equiv \sum_{P:\mathcal{U}} \text{is-a-prop}(P)$$

is (essentially) small, i.e. has an (equivalent) copy in $\mathcal{U}$.

Theorem

EM *implies* Impredicativity.

Proof.

With EM, there are only two propositions: $\mathbf{0}$ and $\mathbf{1}$, so $\Omega_{\mathcal{U}} \simeq \mathbf{2} : \mathcal{U}$. □

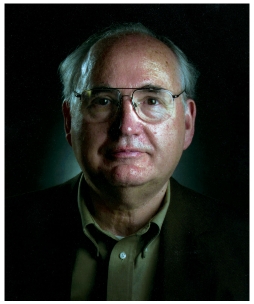
Here $\simeq$ refers to Voevodsky's notion of (type) equivalence.

Outline

Domain theory (classically)

Domain theory is a branch of order theory with applications in

- semantics of programming languages
- topology



Dana S. Scott

Domain theory was pioneered by Dana Scott [Sco72; Sco93] and developed further by many others: Plotkin [Plo83], Lawson, Keimel, Abramsky, Jung [AJ94], Simpson and Escardó, just to name a few.

Order theory studies partially ordered sets (posets).

Basic objects in domain theory

Definition

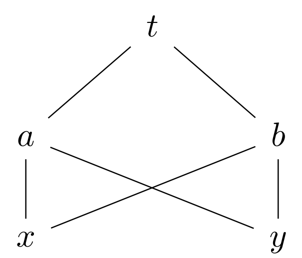
A poset $(P, \leq)$ is *directed* if it is non-empty and for every $x, y \in P$, there exists some $z \in P$ such that $x \leq z$ and $y \leq z$.

For some (computational) intuition: think of a directed set as a set of approximations (or computations). Given two approximations, we can find a better one.

Basic objects in domain theory

Definition

A poset $(P, \leq)$ is *directed* if it is non-empty and for every $x, y \in P$, there exists some $z \in P$ such that $x \leq z$ and $y \leq z$.



An example of a directed set.

For some (computational) intuition: think of a directed set as a set of approximations (or computations). Given two approximations, we can find a better one.

Basic objects in domain theory

Definition

A poset $(P, \leq)$ is *directed* if it is non-empty and for every $x, y \in P$, there exists some $z \in P$ such that $x \leq z$ and $y \leq z$.

Definition

A *directed complete poset* (*dcpo*) is a poset $(P, \leq)$ such that every directed subset of P has a least upper bound in P.

For some (computational) intuition: think of a directed set as a set of approximations (or computations). Given two approximations, we can find a better one.

In a dcpo, we require that all approximations converge to a value (the least upper bound).

Basic objects in domain theory

Definition

A *directed complete poset* (*dcpo*) is a poset $(P, \leq)$ such that every directed subset of P has a least upper bound in P.

$$\begin{tikzcd}[column sep=small] & \pi \arrow[d, dash, dotted] & \\ & 3.141 \arrow[d, dash] & \\ & 3.14 \arrow[dl, dash] \arrow[dr, dash] & \\ 3.13 \arrow[dr, dash] & & 3.15 \arrow[dl, dash] \\ & 3 & \end{tikzcd}$$

An example of a dcpo (classically).

For some (computational) intuition: think of a directed set as a set of approximations (or computations). Given two approximations, we can find a better one.

In a dcpo, we require that all approximations converge to a value (the least upper bound).

Outline

Predicative dcpos in UF

For predicativity reasons, we use families rather than subsets.

Definition

Let $(P, \leq)$ be a poset. A family $u : I \to P$ is *directed* if $\|I\|$ and $\prod_{i,j:I} \|\sum_{k:I} u_i \leq u_k \times u_j \leq u_k\|$.

Note the use of the propositional truncation.

We use the propositional truncation here:

- to ensure that being directed is property (rather than structure);
- because for $i, j : I$, there might be many $k : I$ with $u_i \leq u_k \times u_j \leq u_k$ and we don't mean to specify a choice.

Similarly, asking for an element of I (rather than $\|I\|$) would be asking for a *pointed* (rather than an inhabited) type.

Predicative dcpos in UF

For predicativity reasons, we use families rather than subsets.

Definition

Let $(P, \leq)$ be a poset. A family $u : I \to P$ is *directed* if $\|I\|$ and $\prod_{i,j:I} \|\sum_{k:I} u_i \leq u_k \times u_j \leq u_k\|$.

Note the use of the propositional truncation.

Fix a universe $\mathcal{V}$ of "small" types.

Definition

A *$\mathcal{V}$-dcpo* is a poset $(P, \leq)$ such that every directed family $I \to P$ with I *small* has a least upper bound in $(P, \leq)$.

We use the propositional truncation here:

- to ensure that being directed is property (rather than structure);
- because for $i, j : I$, there might be many $k : I$ with $u_i \leq u_k \times u_j \leq u_k$ and we don't mean to specify a choice.

Similarly, asking for an element of I (rather than $\|I\|$) would be asking for a *pointed* (rather than an inhabited) type.

In a predicative framework, we must be careful about size, which is why we only ask that directed families indexed by types in a fixed universe have least upper bounds.

Scott model of PCF

- PCF: typed programming language with a fixed point combinator for general recursion. PCF types:
  - type ι for natural numbers
  - function types

Scott model of PCF

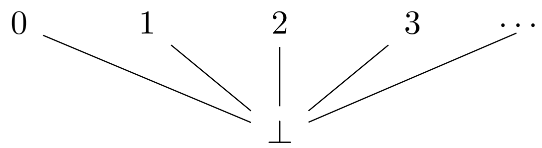
- PCF: typed programming language with a fixed point combinator for general recursion. PCF types:
  - type ι for natural numbers
  - function types
- Scott model of PCF: interpret PCF types as dcpos with a least element that represents non-termination.

Because of the fixed point combinator, a "standard" set-theoretic interpretation will not work (i.e. one where function types are interpreted as exponentials in Set).

A map between dcpos (with bottom) is continuous if it preserves directed suprema. The point is that such maps have fixed points.
The continuous maps between two dcpos with bottom form another dcpo with bottom with the pointwise ordering. This allows us to interpret the function types of PCF.

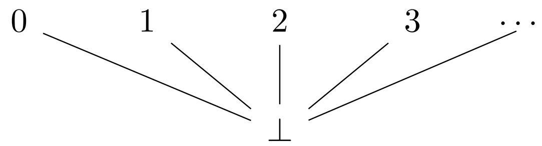
How to represent the type of natural numbers?

Classically:



How to represent the type of natural numbers?

Classically:



But,

$$(\mathbf{N} + \{\bot\} \text{ with this order) is a dcpo } \Rightarrow \text{LPO}.$$

Recall that LPO is:

$$\prod_{\alpha:\mathbf{N}\to\mathbf{2}} \left(\left(\prod_{n:\mathbf{N}} \alpha(n) = 0 \right) + \left(\sum_{k:\mathbf{N}} k \text{ is least with } \alpha(k) = 1 \right) \right).$$

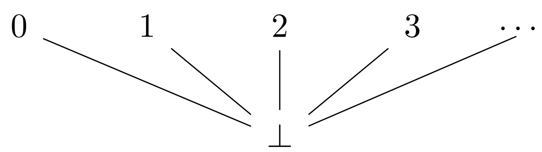
Proof of the implication: given $\alpha : \mathbf{N} \to \mathbf{2}$, define $\beta : \mathbf{N} \to \mathbf{N} + \mathbf{1}$ by:

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

Then β is directed and therefore, if $\mathbf{N} + \mathbf{1}$ is directed complete, has a least upper bound s.
But we can decide if $s = \mathsf{inl}(k)$ for some $k : \mathbf{N}$ or if $s = \mathsf{inr}(\star)$. But the former implies $\alpha(k) = 1$, while the latter implies $\prod_{n:\mathbf{N}} \alpha(n) = 0$.

How to represent the type of natural numbers?

Classically:



But,

$$(\mathbf{N} + \{\bot\} \text{ with this order}) \text{ is a dcpo } \Rightarrow \text{LPO}.$$

So constructively, this is no good.

Lifting

Definition

The *lifting* of a type X is: $\mathcal{L}(X) :\equiv \sum_{P:\Omega}(P \to X)$.

Lifting

Definition

The *lifting* of a type X is: $\mathcal{L}(X) :\equiv \sum_{P:\Omega}(P \to X)$.

Definition

We can embed a type into its lifting:

$$\begin{aligned} \eta_X &: X \to \mathcal{L}(X) \\ x &\mapsto (\mathbf{1}, \lambda(u:\mathbf{1}).x) \end{aligned}$$

Lifting

Definition

The *lifting* of a type X is: $\mathcal{L}(X) :\equiv \sum_{P:\Omega}(P \to X)$.

Definition

We can embed a type into its lifting:

$$\begin{aligned} \eta_X : X &\to \mathcal{L}(X) \\ x &\mapsto (\mathbf{1}, \lambda(u:\mathbf{1}).x) \end{aligned}$$

Theorem (Knapp, Escardó)

$\mathcal{L}$ is monad (on sets) with unit η (modulo size).

Note that $\mathcal{L}$ (potentially) raises universe levels, so that it is a "monad across universes". Moreover, for types that are not sets, this would be some kind ∞-monad, because it is missing coherence conditions.

Lifting

Definition

The *lifting* of a type X is: $\mathcal{L}(X) :\equiv \sum_{P:\Omega}(P \to X)$.

Definition

We can embed a type into its lifting:

$$\begin{aligned} \eta_X &: X \to \mathcal{L}(X) \\ x &\mapsto (\mathbf{1}, \lambda(u:\mathbf{1}).x) \end{aligned}$$

Theorem (Knapp, Escardó)

$\mathcal{L}$ is monad (on sets) with unit η (modulo size).

There is a distinguished element: $\bot_X :\equiv (\mathbf{0}, \text{from-empty}_X) : \mathcal{L}(X)$.

Note that $\mathcal{L}$ (potentially) raises universe levels, so that it is a "monad across universes". Moreover, for types that are not sets, this would be some kind ∞-monad, because it is missing coherence conditions.

With Excluded Middle, this is all, i.e. $\mathcal{L}(X) \simeq X + \mathbf{1}$.

Definition

The *lifting* of a type X is: $\mathcal{L}(X) :\equiv \sum_{P:\Omega}(P \to X)$.

Definition

Let is-defined : $\mathcal{L}(X) \to \Omega$ be: $(P, \varphi) = P$.

Definition

The *lifting* of a type X is: $\mathcal{L}(X) :\equiv \sum_{P:\Omega}(P \to X)$.

Definition

Let is-defined : $\mathcal{L}(X) \to \Omega$ be: $(P, \varphi) = P$.

Definition

Define a partial order $\sqsubseteq$ on $\mathcal{L}(X)$ by:

$$l \sqsubseteq m :\equiv \text{is-defined}(l) \to l = m.$$

Definition

The *lifting* of a type X is: $\mathcal{L}(X) :\equiv \sum_{P:\Omega}(P \to X)$.

Definition

Let is-defined : $\mathcal{L}(X) \to \Omega$ be: $(P, \varphi) = P$.

Definition

Define a partial order $\sqsubseteq$ on $\mathcal{L}(X)$ by:

$$l \sqsubseteq m :\equiv \text{is-defined}(l) \to l = m.$$

Theorem (Knapp, Escardó)

The pair $(\mathcal{L}(X), \sqsubseteq)$ *is a dcpo if* X *is a set.*

Soundness and computational adequacy

Using:

- $(\mathcal{L}(\mathbf{N}), \sqsubseteq)$ to interpret the PCF type of natural numbers
- the monad structure on $\mathcal{L}$

we can define the Scott model of PCF

Soundness and computational adequacy

Using:

- $(\mathcal{L}(\mathbf{N}), \sqsubseteq)$ to interpret the PCF type of natural numbers
- the monad structure on $\mathcal{L}$

we can define the Scott model of PCF and prove:

- soundness: if a PCF program s computes to a term t, then s and t are equal in the model;

Soundness and computational adequacy

Using:

- $(\mathcal{L}(\mathbf{N}), \sqsubseteq)$ to interpret the PCF type of natural numbers
- the monad structure on $\mathcal{L}$

we can define the Scott model of PCF and prove:

- soundness: if a PCF program s computes to a term t, then s and t are equal in the model;
- computational adequacy: if a PCF program t is equal to $\eta(n)$ with $n : \mathbf{N}$, then t computes to the term $\underline{n}$ (that represents n in PCF).

What is especially nice about having a constructive proof of computational adequacy is that it allows us run a PCF program once we prove that it is total, cf. [19, end of Section 7].

Conclusion and current work

Conclusion

Constructive and predicative domain theory in Univalent Foundations

- soundness and computational adequacy of Scott model of PCF using lifting monad
- important use of propositional truncation
- formalised in Agda (some in Coq/UniMath)

Conclusion and current work

Conclusion

Constructive and predicative domain theory in Univalent Foundations

- soundness and computational adequacy of Scott model of PCF using lifting monad
- important use of propositional truncation
- formalised in Agda (some in Coq/UniMath)

Current work

- ✓ bases of dcpos & continuous and algebraic dcpos
- ✓ formalise Scott's D_∞
- exponentials for continuous dcpos (e.g. SFP domains)
- (predicative version of) Pataraia's fixed point theorem

Conclusion and current work

Conclusion

Constructive and predicative domain theory in Univalent Foundations

- soundness and computational adequacy of Scott model of PCF using lifting monad
- important use of propositional truncation
- formalised in Agda (some in Coq/UniMath)

Current work

- ✓ bases of dcpos & continuous and algebraic dcpos
- ✓ formalise Scott's D_∞
- exponentials for continuous dcpos (e.g. SFP domains)
- (predicative version of) Pataraia's fixed point theorem

The Scott model of PCF in univalent type theory. Nov. 2019. arXiv: 1904.09810 [math.LO].

Bases for dcpos

- A dcpo is *continuous* if it has a *basis* that "generates" the whole dcpo.
- Predicatively, we need to strengthen the notion of basis.

In our predicative framework, given a dcpo D, we say that $\beta : B \to D$ is a *basis* if, in addition to the usual axioms of a basis, B is *small* and the way-below/approximation relation of D is *small* when restricted to elements of the form $\beta(b)$.

Bases for dcpos

- A dcpo is *continuous* if it has a *basis* that "generates" the whole dcpo.
- Predicatively, we need to strengthen the notion of basis.

Examples:

- $\mathcal{L}(X)$ has a very simple basis: $X + \mathbf{1}$.

In our predicative framework, given a dcpo D, we say that $\beta : B \to D$ is a *basis* if, in addition to the usual axioms of a basis, B is *small* and the way-below/approximation relation of D is *small* when restricted to elements of the form $\beta(b)$.

Bases for dcpos

- A dcpo is *continuous* if it has a *basis* that "generates" the whole dcpo.
- Predicatively, we need to strengthen the notion of basis.

Examples:

- $\mathcal{L}(X)$ has a very simple basis: $X + \mathbf{1}$.
- $\mathcal{P}(X)$ has the Kuratowski finite subsets of X as a basis.

In our predicative framework, given a dcpo D, we say that $\beta : B \to D$ is a *basis* if, in addition to the usual axioms of a basis, B is *small* and the way-below/approximation relation of D is *small* when restricted to elements of the form $\beta(b)$.

References I

[AJ94] S. Abramsky and A. Jung. 'Domain Theory'. In: *Handbook of Logic in Computer Science*. Ed. by S. Abramsky, D. M. Gabbay and T. S. E. Maibaum. Vol. 3. Updated online version available at: `https://www.cs.bham.ac.uk/~axj/pub/papers/handy1.pdf`. Clarendon Press, 1994, pp. 1–168.

[BKV09] Nick Benton, Andrew Kennedy and Carsten Varming. 'Some Domain Theory and Denotational Semantics in Coq'. In: *Theorem Proving in Higher Order Logics*. Ed. by Stefan Berghofer et al. Vol. 5674. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009, pp. 115–130. DOI: 10.1007/978-3-642-03359-9_10.

References II

[EK17] Martín H. Escardó and Cory M. Knapp. ‘Partial Elements and Recursion via Dominances in Univalent Type Theory’. In: *26th EACSL Annual Conference on Computer Science Logic (CSL 2017)*. Ed. by Valentin Goranko and Mads Dam. Vol. 82. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2017, 21:1–21:16. DOI: `10.4230/LIPIcs.CSL.2017.21`.

[Esc20] Martín Hötzel Escardó. *Introduction to Univalent Foundations of Mathematics with Agda*. Feb. 2020. arXiv: `1911.00580 [cs.LO]`.

References III

[Koc91] Anders Kock. 'Algebras for the partial map classifier monad'. In: *Category Theory*. Ed. by Aurelio Carboni, Maria Cristina Pedicchio and Guiseppe Rosolini. Vol. 1488. Lecture Notes in Mathematics. Springer Berlin Heidelberg, 1991, pp. 262–278. DOI: 10.1007/BFb0084225.

[Plo77] G.D. Plotkin. 'LCF considered as a programming language'. In: *Theoretical Computer Science* 5.3 (1977), pp. 223–255. DOI: 10.1016/0304-3975(77)90044-5.

[Plo83] Gordon Plotkin. 'Domains'. Lecture notes on domain theory, known as the *Pisa Notes*. 1983. URL: https://homepages.inf.ed.ac.uk/gdp/publications/Domains_a4.ps.

References IV

[RS99] Bernhard Reus and Thomas Streicher. ‘General synthetic domain theory – a logical approach’. In: *Mathematical Structures in Computer Science* 9.2 (1999), pp. 177–223. DOI: 10.1017/S096012959900273X.

[SVV96] Giovanni Sambin, Silvio Valentini and Paolo Virgili. ‘Constructive domain theory as a branch of intuitionistic pointfree topology’. In: *Theoretical Computer Science* 159.2 (June 1996), pp. 319–341. DOI: 10.1016/0304-3975(95)00169-7.

References V

[Sco72] Dana Scott. 'Continuous lattices'. In: *Toposes, Algebraic Geometry and Logic*. Ed. by F.W. Lawvere. Vol. 274. Lecture Notes in Mathematics. 1972, pp. 97–136. DOI: 10.1007/BFb0073967.

[Sco93] Dana S. Scott. 'A type-theoretical alternative to ISWIM, CUCH, OWHY'. In: *Theoretical Computer Science* 121.1 (1993), pp. 411–440. DOI: 10.1016/0304-3975(93)90095-B.

[Str06] Thomas Streicher. *Domain-Theoretic Foundations of Functional Programming*. World Scientific, 2006. DOI: 10.1142/6284.

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