Synthetic Domains in the 21st Century

September 28, 2023
Topos Institute Colloquium

Jonathan Sterling

Computer Laboratory
University of Cambridge

In 1949, Alan Turing presented one of the first “correctness proofs” for a computer program (an addition checker). He asks:

> *How can one check a routine in the sense of making sure that it is right?*

In 1949, Alan Turing presented one of the first "correctness proofs" for a computer program (an addition checker). He asks:

> *How can one check a routine in the sense of making sure that it is right?* ***In order that the [person] who checks may not have too difficult a task the programmer should make a number of definite assertions which can be checked individually, and from which the correctness of the whole programme easily follows.***

In 1949, Alan Turing presented one of the first "correctness proofs" for a computer program (an addition checker). He asks:

> *How can one check a routine in the sense of making sure that it is right?* ***In order that the [person] who checks may not have too difficult a task the programmer should make a number of definite assertions which can be checked individually, and from which the correctness of the whole programme easily follows.***

Turing's precocity:

1. **compositional reasoning** about programs
2. annotating programs with **local assertions** (*cf*. Floyd & Hoare)
3. **invariants** that cut across all steps of program execution

Goal of programming languages field: to give ***precise and reliable meaning*** to the "assertions" of Turing's verified addition checker.

Isn't it obvious what an assertion means? (No)

Think of a program with some assertions.

// x is an integer
set x **to** $2 * x$
// x is an even integer
print x
// x is an even integer

Isn't it obvious what an assertion means? (No)

Think of a program with some assertions.

// x is an integer
set x **to** $2 * x$
// x is an even integer
print x
// x is an even integer

The meaning of these assertions is *not* obvious.

1. What does a "variable" like x actually refer to?
2. Even "$2 * 5$" is so far only a *program expression*, so it is not an integer of any kind, much less an *even* integer.

Assertions and the meaning of program expressions

To give any kind of definite and reliable meaning to assertions, it is therefore necessary to explain what a program expression means. We also want this meaning to be related to what the computer really does.

Assertions and the meaning of program expressions

To give any kind of definite and reliable meaning to assertions, it is therefore necessary to explain what a program expression means. We also want this meaning to be related to what the computer really does.

Denotational semantics explains the meaning of a complex program $P(Q, R, S, \ldots)$ in terms of the meanings of its subroutines $Q, R, S, \ldots$; *cf.* Turing's compositionality criterion. Then, assertions are explained as predicates(*) on the meanings of the programs they concern.

Naïve denotational semantics in sets

Naïve denotational semantics in sets

1. A context Γ or a type τ refers to a *set* $[\![\Gamma]\!]$ or $[\![\tau]\!]$.

Naïve denotational semantics in sets

1. A context Γ or a type τ refers to a *set* $[\![\Gamma]\!]$ or $[\![\tau]\!]$.
2. A program $\Gamma \vdash M : \tau$ refers to a function $[\![M]\!] : [\![\Gamma]\!] \to [\![\tau]\!]$.

Naïve denotational semantics in sets

1. A context Γ or a type τ refers to a *set* $[\![\Gamma]\!]$ or $[\![\tau]\!]$.
2. A program $\Gamma \vdash M : \tau$ refers to a function $[\![M]\!] : [\![\Gamma]\!] \to [\![\tau]\!]$.
3. An assertion $\Gamma \mid \phi$ refers to a subset $[\![\phi]\!] \subseteq [\![\Gamma]\!]$.

Naïve denotational semantics in sets

1. A context Γ or a type τ refers to a *set* $[\![\Gamma]\!]$ or $[\![\tau]\!]$.
2. A program $\Gamma \vdash M : \tau$ refers to a function $[\![M]\!] : [\![\Gamma]\!] \to [\![\tau]\!]$.
3. An assertion $\Gamma \mid \phi$ refers to a subset $[\![\phi]\!] \subseteq [\![\Gamma]\!]$.
4. An entailment $\Gamma \mid \phi \vdash \psi$ refers to an *inclusion* $[\![\phi]\!] \subseteq [\![\psi]\!] \subseteq [\![\Gamma]\!]$.

Naïve denotational semantics in sets

1. A context Γ or a type τ refers to a *set* $[\![\Gamma]\!]$ or $[\![\tau]\!]$.
2. A program $\Gamma \vdash M : \tau$ refers to a function $[\![M]\!] : [\![\Gamma]\!] \to [\![\tau]\!]$.
3. An assertion $\Gamma \mid \phi$ refers to a subset $[\![\phi]\!] \subseteq [\![\Gamma]\!]$.
4. An entailment $\Gamma \mid \phi \vdash \psi$ refers to an *inclusion* $[\![\phi]\!] \subseteq [\![\psi]\!] \subseteq [\![\Gamma]\!]$.

We are taught almost from birth how to reason informally with sets. The benefit of naïve set theoretic semantics has nothing to do with "set theory" in the professional sense: it is good because we know how to think ***naïvely & reliably*** about collections and mappings between them.

Explaining the meanings of programs is hard because:

Explaining the meanings of programs is hard because:

1. some program have *general recursion* (in both terms and types!);

Explaining the meanings of programs is hard because:

1. some program have *general recursion* (in both terms and types!);
2. some programs are *stateful*;

Explaining the meanings of programs is hard because:

1. some program have *general recursion* (in both terms and types!);
2. some programs are *stateful*;
3. some programs are *polymorphic*;

Explaining the meanings of programs is hard because:

1. some program have *general recursion* (in both terms and types!);
2. some programs are *stateful*;
3. some programs are *polymorphic*;
4. some programs are *nondeterministic*;

Explaining the meanings of programs is hard because:

1. some program have *general recursion* (in both terms and types!);
2. some programs are *stateful*;
3. some programs are *polymorphic*;
4. some programs are *nondeterministic*;
5. some programs are *interactive*
6. . . .

Explaining the meanings of programs is hard because:

1. some program have *general recursion* (in both terms and types!);
2. some programs are *stateful*;
3. some programs are *polymorphic*;
4. some programs are *nondeterministic*;
5. some programs are *interactive*
6. …

Mere sets are too *discrete* to bring order to this complexity! Dana Scott's ***domain theory*** broke the logjam (Scott, 1970; Scott, 1972; Scott, 1976; Scott, 1982; Scott, 1993).

Denotational semantics of recursion via domains

Replace sets with some kind of ***space*** (“domain”) in which points have a specialization (pre)order supporting colimits of ascending chains.

Denotational semantics of recursion via domains

Replace sets with some kind of ***space*** (“domain”) in which points have a specialization (pre)order supporting colimits of ascending chains.

1. A context Γ or a type τ refers to a space $[\![\Gamma]\!]$ or $[\![\tau]\!]$.

Denotational semantics of recursion via domains

Replace sets with some kind of ***space*** (“domain”) in which points have a specialization (pre)order supporting colimits of ascending chains.

1. A context Γ or a type τ refers to a space $[\![\Gamma]\!]$ or $[\![\tau]\!]$.
2. A program $\Gamma \vdash M : \tau$ refers to a continuous function $[\![M]\!] : [\![\Gamma]\!] \to [\![\tau]\!]$.

Denotational semantics of recursion via domains

Replace sets with some kind of ***space*** (“domain”) in which points have a specialization (pre)order supporting colimits of ascending chains.

1. A context Γ or a type τ refers to a space $[\![\Gamma]\!]$ or $[\![\tau]\!]$.
2. A program $\Gamma \vdash M : \tau$ refers to a continuous function $[\![M]\!] : [\![\Gamma]\!] \to [\![\tau]\!]$.
3. A recursive program $\Gamma \vdash \mathbf{fix}\, f : \tau$ refers to the colimit of the chain $[\bot \leqslant [\![f]\!]\bot \leqslant [\![f]\!]^2\bot \leqslant \ldots]$.

Denotational semantics of recursion via domains

Replace sets with some kind of ***space*** (“domain”) in which points have a specialization (pre)order supporting colimits of ascending chains.

1. A context Γ or a type τ refers to a space $[\![\Gamma]\!]$ or $[\![\tau]\!]$.
2. A program $\Gamma \vdash M : \tau$ refers to a continuous function $[\![M]\!] : [\![\Gamma]\!] \to [\![\tau]\!]$.
3. A recursive program $\Gamma \vdash \mathbf{fix}\, f : \tau$ refers to the colimit of the chain $[\bot \leqslant [\![f]\!]\bot \leqslant [\![f]\!]^2\bot \leqslant \ldots]$.
4. An assertion $\Gamma \mid \phi$ refers to .... ??? (admissible subspaces, open subspaces, closed subspaces) ???

Downsides of classical domains for PL semantics

1. **Proliferation of obscure variations:** there's a ton of different kinds of domain (ω-dcpo, ω-cpo, Scott domain, strongly algebraic domain, *etc*.), each solving different problems.
2. **Abstraction is too low:** constant continuity side-obligations an impediment for everyday users of domain theory.
3. **No intrinsic notion of assertion**: many different possible ways to interpret assertions, but no unifying language.
4. **No intrinsic notion of *dependent type*:** thus impossible to reason "naïvely" in the language of domains, leading to an artificial boundary between programming and verification.
5. **Difficulty with "awkward" PL features**: higher-order store with parametric polymorphism; concurrency requires new kinds of domain (*e.g*. event structures, presheaf topoi, *etc*.).

The thesis of synthetic domain theory

Scott recognized the obscurity and complexity of classical domain theory, and initiated the field of ***synthetic domain theory*** to search for ***topoi*** that have domain-like spaces as full subcategories.

1. A ***topos*** is a model of extensional Intuitionistic Type Theory in which the propositions (subsingletons) form a univalent universe.
2. It is as easy to reason *rigorously and informally* in an arbitrary topos as it is in set theory. **Naïve denotational semantics for recursion.**
3. A ***full subcategory of domains*** means that you never have to check a continuity condition again.
4. Every notion of assertion (*e.g*. admissible subspace, open/closed subspace, *etc*.) easily expressed in terms of the ***subobject classifier***.
5. Automatic support for ***dependent types***: programming blends with verification.

Axioms of synthetic domain theory

Many possible axiom systems, but we will focus on a few core axioms that are sufficient in practice, inspired by Simpson (2004).

Let $\mathcal{S}$ be an elementary topos with a natural numbers object; we will work informally in the internal language.

Axiom (Dominance)

A subuniverse $\Sigma \subseteq \Omega$ closed under $\top$ and dependent sums $\sum_{x:\phi} \psi x$ where $\phi : \Sigma$ and $\psi : \phi \to \Sigma$.

Using this axiom, the *Σ-partial map classifier* construction gives a monad $\mathbb{L} = (L, \eta, \mu)$.

$$\begin{aligned} LA &:\equiv \textstyle\sum_{\phi:\Sigma} A^{\phi} \\ \eta_A a &:\equiv (\top, \lambda\_.a) \\ \mu_A(\phi, u) &:\equiv \left(\textstyle\sum_{x:\phi} (ux).1, \lambda(x,y).(ux).2y\right) \end{aligned}$$

This is a semantic partiality monad! We will later isolate the types in which partial functions can be defined by recursion.

Axiom (Empty Join)

The dominance $\Sigma \subseteq \Omega$ is closed under $\bot$.

We can also assume joins of higher arity, but this limits the models. Empty joins parameterize diverging computations $(\bot, \lambda()) : LA$; binary joins would parameterize *parallel* computations.

Let $L\omega \to \omega$ be the ***initial algebra*** for the lifting monad $\mathbb{L}$. For type theorists, this is the inductive type $\mathsf{W}_{\phi:\Sigma}\,\phi$.

Think of ω as the "generic ω-chain"; we have elements corresponding to natural numbers, but ω is somehow "thicker" than $\mathbb{N}$.

Let $\bar{\omega} \to L\bar{\omega}$ be the ***final coalgebra*** for $\mathbb{L}$; this is a coinductive type. We have $\omega \hookrightarrow \bar{\omega}$, and outside the image lies an infinite element $\infty : \bar{\omega}$.

Think of $\omega \hookrightarrow \bar{\omega}$ as the incidence relation between the generic omega chain and its colimit.

Definition

A type A is called ***complete*** when it is *orthogonal* to $\omega \hookrightarrow \bar{\omega}$, *i.e.* every figure $\alpha : \omega \to A$ extends to a unique figure $\bar{\alpha} : \bar{\omega} \to A$. We may write $\bigvee_{i:\omega} \alpha_i$ for $\bar{\alpha}_\infty$.

Axiom (Predomains)

There exists a *reflective full subfibration* $\mathcal{P} \subseteq \mathcal{S}$ whose objects are called ***predomains*** and are all complete and closed under $\mathbb{L}$.

Note: by above, $\mathcal{P}$ is automatically cartesian closed, and both complete & cocomplete in the fibered sense, with limits computed as in $\mathcal{S}$.

Definition

A ***domain*** is defined to be an $\mathbb{L}$-algebra whose underlying type is a predomain. A ***strict (linear) map*** between domains is an $\mathbb{L}$-algebra homomorphism.

Analogy: predomains $\sim$ unpointed cpos, domains $\sim$ pointed cpos.

Axiom (Optional)

The Kleisli category $\mathcal{P}_{\mathbb{L}}$ is algebraically compact as a fibration over $\mathcal{S}$.

In other words, we can compute recursive types.

Many more axioms can be imposed, to refine our picture of "domains"; important for *relating* synthetic constructions to ordinary math, but not needed for workaday denotational semantics.

Naïve denotational semantics of recursion

The internal intuitionistic type theory of any topos $\mathcal{S}$ satisfying our axioms serves as a *metalanguage* for naïve denotational semantics.

Naïve call-by-value interpretation of recursion

1. A context Γ or a type τ refers to a *predomain* $[\![\Gamma]\!]$ or $[\![\tau]\!]$.
2. A program $\Gamma \vdash M : \tau$ refers to a Kleisli mapping $[\![M]\!] : [\![\Gamma]\!] \to L[\![\tau]\!]$. (Continuity is automatic!)
   - Recursive functions computed using *completeness* of $L[\![\tau]\!]$, taking the "formal colimits" of a parameterized chain $[\![\Gamma]\!] \times \omega \to L[\![\tau]\!]$ defined using structural recursion on ω.
3. An assertion $\Gamma \mid \phi$ refers to a subset $[\![\phi]\!] \subseteq [\![\Gamma]\!]$; f.p. induction restricted to *complete* subsets.
4. An entailment $\Gamma \mid \phi \vdash \psi$ refers to an inclusion $[\![\phi]\!] \subseteq [\![\psi]\!] \subseteq [\![\Gamma]\!]$.

Scales effortlessly to parametric polymorphism, recursive types, first-order store, finite non-determinism, and thus interleaving concurrency. Higher-order store (storing closures) as well as true concurrency not accounted for in this environment.

Relating synthetic denotational semantics to "real math"

It is well and good to verify programs using the axioms of synthetic domain theory, but is this "sound" with respect to (1) classical domain theoretic semantics or (2) operational notions of equivalence?

Answering these questions means finding ***models*** of the axioms.

1. **Soundness for operational equivalence** ("computational adequacy") follows from a *nearly arbitrary* model of SDT thanks to Simpson (2004) and Marcelo P. Fiore and Plotkin (1994).
2. **Soundness for classical denotational semantics** follows because cpos, *etc*. embed nicely into sheaf models of SDT (Marcelo P. Fiore and Plotkin, 1996; Marcelo P. Fiore and Rosolini, 1997).

Metric domain theory in the 1980s

In the 1980s, a new kind of domain theory emerges. . .

Metric domain theory in the 1980s

In the 1980s, a new kind of domain theory emerges...

1. replace cpos and continuous maps with (complete, *etc.*) ***metric spaces*** and ***nonexpansive maps***.

Metric domain theory in the 1980s

In the 1980s, a new kind of domain theory emerges. . .

1. replace cpos and continuous maps with (complete, *etc*.) ***metric spaces*** and ***nonexpansive maps***.
2. **idea:** contractive maps (and locally contractive functors) have *unique* fixed points.

Metric domain theory in the 1980s

In the 1980s, a new kind of domain theory emerges…

1. replace cpos and continuous maps with (complete, *etc.*) ***metric spaces*** and ***nonexpansive maps***.
2. **idea:** contractive maps (and locally contractive functors) have *unique* fixed points.

See: Arnold and Nivat (1980), MacQueen, Plotkin, and Sethi (1984), and America and Rutten (1987).

Metric domain theory escapes the lab

Two decades later, programming language theorists gave their own take on metric domain theory under the name of ***step-indexing*** (Appel and McAllester, 2001). **A different ethos for a different era.**

Metric domain theory escapes the lab

Two decades later, programming language theorists gave their own take on metric domain theory under the name of ***step-indexing*** (Appel and McAllester, 2001). **A different ethos for a different era.**

1. **technically simple:** domain equations solved by recursion on concrete operational steps; everything is syntax.

Metric domain theory escapes the lab

Two decades later, programming language theorists gave their own take on metric domain theory under the name of ***step-indexing*** (Appel and McAllester, 2001). **A different ethos for a different era.**

1. **technically simple:** domain equations solved by recursion on concrete operational steps; everything is syntax.
2. **worse is better:** the rich categorical structure of domain theory thrown away, because who needs it? (Actually needed for scaling!)

Metric domain theory escapes the lab

Two decades later, programming language theorists gave their own take on metric domain theory under the name of ***step-indexing*** (Appel and McAllester, 2001). **A different ethos for a different era.**

1. **technically simple:** domain equations solved by recursion on concrete operational steps; everything is syntax.
2. **worse is better:** the rich categorical structure of domain theory thrown away, because who needs it? (Actually needed for scaling!)
3. **exceptionally strong results:** operational step-indexing the catalyst for solving many long-standing problems, *e.g.* semantic soundness of **System F**$_{\mu,\mathit{ref}}$ as in the *tour de force* thesis of Ahmed (2004).

The end of history?

A new synthetic domain theory from step-indexing

A strand of continuity between the world of metric domain theory and step-indexed PL semantics leads to a new synthetic domain theory.

A new synthetic domain theory from step-indexing

A strand of continuity between the world of metric domain theory and step-indexed PL semantics leads to a new synthetic domain theory.

Theorem (Birkedal, Møgelberg, Schwinghammer, and Støvring (2011))

*A **complete bisected ultrametric space** is more simply described as a presheaf on ω whose restriction maps are surjections / quotients (i.e. flabby presheaves). The inclusion into $\widehat{\omega}$ is **coreflective**.*

A new synthetic domain theory from step-indexing

A strand of continuity between the world of metric domain theory and step-indexed PL semantics leads to a new synthetic domain theory.

Theorem (Birkedal, Møgelberg, Schwinghammer, and Støvring (2011))

*A **complete bisected ultrametric space** is more simply described as a presheaf on ω whose restriction maps are surjections / quotients (i.e. flabby presheaves). The inclusion into $\widehat{\omega}$ is **coreflective**.*

Synthetic guarded domain theory generalizes the internal language of $\widehat{\omega}$, lifting the ill-advised (*) restriction to flabby presheaves.

A new synthetic domain theory from step-indexing

A strand of continuity between the world of metric domain theory and step-indexed PL semantics leads to a new synthetic domain theory.

Theorem (Birkedal, Møgelberg, Schwinghammer, and Støvring (2011))

*A **complete bisected ultrametric space** is more simply described as a presheaf on ω whose restriction maps are surjections / quotients (i.e. flabby presheaves). The inclusion into $\widehat{\omega}$ is **coreflective**.*

Synthetic guarded domain theory generalizes the internal language of $\widehat{\omega}$, lifting the ill-advised (*) restriction to flabby presheaves.

Axiomatizations: Birkedal, Møgelberg, Schwinghammer, and Støvring (2011), Milius and Litak (2017), and Palombi and Sterling (2023).

New features of synthetic guarded domain theory

New features of synthetic guarded domain theory

1. Unlike traditional SDT, no special classes of objects (*e.g*. complete, replete, *etc*.); **the “predomains” form a topos**.

New features of synthetic guarded domain theory

1. Unlike traditional SDT, no special classes of objects (*e.g*. complete, replete, *etc*.); **the “predomains” form a topos**.
2. Recursion introduced by an endofunctor $\blacktriangleright$, which corresponds to a single “unfolding” of a recursive domain equation; **“domains” are just $\blacktriangleright$-algebras.**

New features of synthetic guarded domain theory

1. Unlike traditional SDT, no special classes of objects (*e.g.* complete, replete, *etc.*); **the "predomains" form a topos**.
2. Recursion introduced by an endofunctor ►, which corresponds to a single "unfolding" of a recursive domain equation; **"domains" are just ►-algebras.**
3. **New feature:** the universe of *all* small predomains is a domain (*cf.* domains of *information systems* in classical domain theory, which classify only algebraic[. . .] domains).

Naïve denotational semantics in SGDT?

1. **Naïve denotational semantics of general recursion is both easy and elegant** (Paviotti, Møgelberg, and Birkedal, 2015; Møgelberg and Paviotti, 2016; Paviotti, 2016).

Naïve denotational semantics in SGDT?

1. **Naïve denotational semantics of general recursion is both easy and elegant** (Paviotti, Møgelberg, and Birkedal, 2015; Møgelberg and Paviotti, 2016; Paviotti, 2016).
2. Only a little bit harder is naïve denotational semantics of ***general recursion***, ***parametric polymorphism***, and ***higher-order store*** with semantic worlds, *etc*. (Sterling, Gratzer, and Birkedal, 2022).

Naïve denotational semantics in SGDT?

1. **Naïve denotational semantics of general recursion is both easy and elegant** (Paviotti, Møgelberg, and Birkedal, 2015; Møgelberg and Paviotti, 2016; Paviotti, 2016).
2. Only a little bit harder is naïve denotational semantics of ***general recursion***, ***parametric polymorphism***, and ***higher-order store*** with semantic worlds, *etc*. (Sterling, Gratzer, and Birkedal, 2022).
3. **A *new* result:** denotational semantics for full dependent type theory with higher-order store, parametricity, *etc*. (*op. cit*.).

Naïve denotational semantics in SGDT?

1. **Naïve denotational semantics of general recursion is both easy and elegant** (Paviotti, Møgelberg, and Birkedal, 2015; Møgelberg and Paviotti, 2016; Paviotti, 2016).
2. Only a little bit harder is naïve denotational semantics of ***general recursion***, ***parametric polymorphism***, and ***higher-order store*** with semantic worlds, *etc*. (Sterling, Gratzer, and Birkedal, 2022).
3. **A *new* result:** denotational semantics for full dependent type theory with higher-order store, parametricity, *etc*. (*op. cit*.).
4. Aagaard, Sterling, and Birkedal (2023) adapt Iris-style **higher-order separation logic** to denotational semantics, higher-order ghost state and invariants forthcoming.

Finally denotational semantics responds to Ahmed (2004), after which it seemed to many community members that operationally-based semantics was the only viable approach to higher-order store.

What's next? Technical & social prospects

1. Denotational semantics of **interleaving concurrency + higher-order effects** are too easy, but easy examples important.
2. ***True concurrency*** could be the "killer application" of denotational clarity in the era of relaxed memory. Let go of *functional bias* and study 2-dimensional domains!
3. **Education and outreach:** operational methods have dominated in an era in which ***sheer humanpower*** plays a bigger role than clarity. Soon, the pendulum swings again.

Thanks! **P.S. I'm hiring! Please get in touch.**

References I

Aagaard, Frederik Lerbjerg, Jonathan Sterling, and Lars Birkedal (Apr. 8, 2023). "A denotationally-based program logic for higher-order store". To appear, *Mathematical Foundations of Programming Semantics*. URL: https://www.cs.au.dk/~birke/papers/tulip.pdf.

Ahmed, Amal Jamil (2004). "Semantics of Types for Mutable State". PhD thesis. Princeton University. URL: http://www.ccs.neu.edu/home/amal/ahmedthesis.pdf.

America, Pierre and Jan J. M. M. Rutten (1987). "Solving Reflexive Domain Equations in a Category of Complete Metric Spaces". In: *Proceedings of the 3rd Workshop on Mathematical Foundations of Programming Language Semantics*. Berlin, Heidelberg: Springer-Verlag, pp. 254–288. ISBN: 3-540-19020-1.

Appel, Andrew W. and David McAllester (Sept. 2001). "An Indexed Model of Recursive Types for Foundational Proof-carrying Code". In: *ACM Transactions on Programming Languages and Systems* 23.5, pp. 657–683. ISSN: 0164-0925. DOI: 10.1145/504709.504712.

Arnold, A. and M. Nivat (1980). "Metric interpretations of infinite trees and semantics of non deterministic recursive programs". In: *Theoretical Computer Science* 11.2, pp. 181–205. ISSN: 0304-3975. DOI: 10.1016/0304-3975(80)90045-6.

Birkedal, Lars, Rasmus Ejlers Møgelberg, Jan Schwinghammer, and Kristian Støvring (2011). "First Steps in Synthetic Guarded Domain Theory: Step-Indexing in the Topos of Trees". In: *Proceedings of the 2011 IEEE 26th Annual Symposium on Logic in Computer Science*. Washington, DC, USA: IEEE Computer Society, pp. 55–64. ISBN: 978-0-7695-4412-0. DOI: 10.1109/LICS.2011.16. arXiv: 1208.3596 [cs.LO].

References II

Bizjak, Aleš (2016). "On semantics and applications of guarded recursion". PhD thesis. Aarhus University.

Cattani, Gian Luca, Marecelo P. Fiore, and Glynn Winskel (1998). "A Theory of Recursive Domains with Applications to Concurrency". In: *Proceedings of the 13th Annual IEEE Symposium on Logic in Computer Science*. USA: IEEE Computer Society. ISBN: 0-8186-8506-9.

Cattani, Gian Luca and Glynn Winskel (2005). "Profunctors, open maps and bisimulation". In: *Mathematical Structures in Computer Science* 15.3, pp. 553–614. DOI: 10.1017/S0960129505004718.

Escardó, Martín Hötzel (1999). "A metric model of PCF". In: *Workshop on Realizability Semantics and Applications*.

Fiore, Marcelo P. and Gordon D. Plotkin (1994). "An axiomatisation of computationally adequate domain theoretic models of FPC". In: *Proceedings Ninth Annual IEEE Symposium on Logic in Computer Science*, pp. 92–102. DOI: 10.1109/LICS.1994.316083.

— (1996). "An Extension of Models of Axiomatic Domain Theory to Models of Synthetic Domain Theory". In: *Computer Science Logic, 10th International Workshop, CSL '96, Annual Conference of the EACSL, Utrecht, The Netherlands, September 21-27, 1996, Selected Papers*. Ed. by Dirk van Dalen and Marc Bezem. Vol. 1258. Lecture Notes in Computer Science. Springer, pp. 129–149. DOI: 10.1007/3-540-63172-0_36.

References III

- Fiore, Marcelo P. and Giuseppe Rosolini (1997). “The category of cpos from a synthetic viewpoint”. In: *Thirteenth Annual Conference on Mathematical Foundations of Progamming Semantics, MFPS 1997, Carnegie Mellon University, Pittsburgh, PA, USA, March 23-26, 1997*. Ed. by Stephen D. Brookes and Michael W. Mislove. Vol. 6. Electronic Notes in Theoretical Computer Science. Elsevier, pp. 133–150. DOI: 10.1016/S1571-0661(05)80165-3.
- MacQueen, David, Gordon D. Plotkin, and Ravi Sethi (1984). “An Ideal Model for Recursive Polymorphic Types”. In: *Proceedings of the 11th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*. Salt Lake City, Utah, USA: Association for Computing Machinery, pp. 165–174. ISBN: 0-89791-125-3. DOI: 10.1145/800017.800528. URL: https://doi.org/10.1145/800017.800528.
- Milius, Stefan and Tadeusz Litak (2017). “Guard Your Daggers and Traces: Properties of Guarded (Co-)recursion”. In: *Fundamenta Informaticae* 150.3-4, pp. 407–449. DOI: 10.3233/FI-2017-1475.
- Møgelberg, Rasmus Ejlers and Marco Paviotti (2016). “Denotational Semantics of Recursive Types in Synthetic Guarded Domain Theory”. In: *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*. New York, NY, USA: Association for Computing Machinery, pp. 317–326. ISBN: 978-1-4503-4391-6. DOI: 10.1145/2933575.2934516.
- Morris, F. L. and C. B. Jones (1984). “An Early Program Proof by Alan Turing”. In: *Annals of the History of Computing* 6.2, pp. 139–143. DOI: 10.1109/MAHC.1984.10017.

References IV

- Palombi, Daniele and Jonathan Sterling (Feb. 2023). "Classifying topoi in synthetic guarded domain theory". In: *Electronic Notes in Theoretical Informatics and Computer Science* Volume 1 - Proceedings of MFPS XXXVIII. DOI: 10.46298/entics.10323. URL: https://entics.episciences.org/10323.
- Paviotti, Marco (2016). "Denotational semantics in Synthetic Guarded Domain Theory". PhD thesis. Denmark: IT-Universitetet i København. ISBN: 978-87-7949-345-2.
- Paviotti, Marco, Rasmus Ejlers Møgelberg, and Lars Birkedal (2015). "A Model of PCF in Guarded Type Theory". In: *Electronic Notes in Theoretical Computer Science* 319.Supplement C. The 31st Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXI), pp. 333–349. ISSN: 1571-0661. DOI: 10.1016/j.entcs.2015.12.020.
- Scott, Dana S. (Nov. 1970). *Outline of a Mathematical Theory of Computation*. Tech. rep. PRG02. Oxford University Computer Laboratory, p. 30.
- — (1972). "Continuous lattices". In: *Toposes, Algebraic Geometry and Logic*. Ed. by F. W. Lawvere. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 97–136. ISBN: 978-3-540-37609-5.
- — (1976). "Data Types as Lattices". In: *SIAM Journal on Computing* 5.3, pp. 522–587. DOI: 10.1137/0205037.
- — (1982). "Domains for denotational semantics". In: *Automata, Languages and Programming*. Ed. by Mogens Nielsen and Erik Meineche Schmidt. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 577–610. ISBN: 978-3-540-39308-5.

References V

- Scott, Dana S. (1993). "A type-theoretical alternative to ISWIM, CUCH, OWHY". In: *Theoretical Computer Science* 121.1, pp. 411–440. ISSN: 0304-3975. DOI: 10.1016/0304-3975(93)90095-B.
- Simpson, Alex (2004). "Computational adequacy for recursive types in models of intuitionistic set theory". In: *Annals of Pure and Applied Logic* 130.1. Papers presented at the 2002 IEEE Symposium on Logic in Computer Science (LICS), pp. 207–275. ISSN: 0168-0072. DOI: 10.1016/j.apal.2003.12.005.
- Sterling, Jonathan, Daniel Gratzer, and Lars Birkedal (July 2022). "Denotational semantics of general store and polymorphism". Unpublished manuscript. DOI: 10.48550/arXiv.2210.02169.
- Turing, Alan M. (1949). "Checking a Large Routine". In: *Report of a Conference on High Speed Automatic Calculation Machines*. Univ. Math. Lab, Cambridge, pp. 67–69.