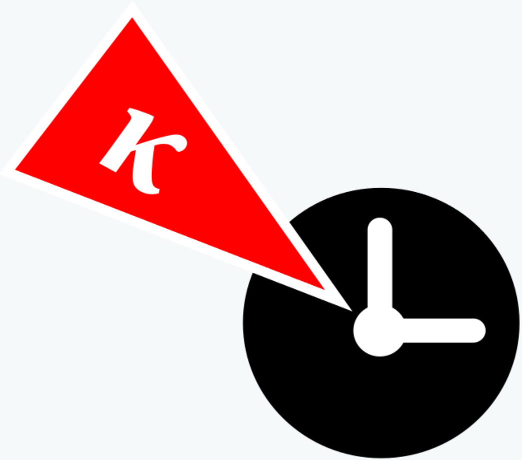
Guarded Computational Type Theory

Jonathan Sterling
Robert Harper
Carnegie Mellon University



The story begins almost 50 years ago with the invention of *domain theory* and *denotational semantics*.

Main idea: find category in which mixed variance operators have fixed points.

The story begins almost 50 years ago with the invention of *domain theory* and *denotational semantics*.

Main idea: find category in which mixed variance operators have fixed points.

Variety of techniques employed today, but the problem of mixed variance fixed points remained the fundamental struggle of PL semantics.

In 2001, Appel and McAllester invent new stratified semantic technique to construct fixed points of mixed variance, called *step-indexing*.

Main idea: index everything by its “stage” of construction. step-indexed predicate = monotone sequence of predicates

In 2001, Appel and McAllester invent new stratified semantic technique to construct fixed points of mixed variance, called *step-indexing*.

Main idea: index everything by its "stage" of construction. step-indexed predicate = monotone sequence of predicates

$$\llbracket \mu\alpha.A \rrbracket\rho \triangleq \{\mathtt{fold}(e) \mid e \in \llbracket A \rrbracket(\rho, \alpha \mapsto \llbracket \mu\alpha.A \rrbracket\rho)\}$$
$$\llbracket A \to B \rrbracket\rho \triangleq \{\lambda x.e \mid \forall v \in \llbracket A \rrbracket\rho.\ e[v/x] \in \llbracket B \rrbracket\rho\}$$

(no!! not well-defined)

In 2001, Appel and McAllester invent new stratified semantic technique to construct fixed points of mixed variance, called *step-indexing*.

Main idea: index everything by its "stage" of construction.
step-indexed predicate = monotone sequence of predicates

$$[\![i \mid \mu\alpha.A]\!]\rho \triangleq \{\texttt{fold}(e) \mid \forall j < i.\, e \in [\![j \mid A]\!](\rho, \alpha \mapsto [\![j+1 \mid \mu\alpha.A]\!]\rho)\}$$

$$[\![i \mid A \to B]\!]\rho \triangleq \{\lambda x.e \mid \forall j \leq i.\, \forall v \in [\![j \mid A]\!]\rho.\, e[v/x] \in [\![j \mid B]\!]\rho\}$$

Using ideas from Nakano [2000], abstract version of step-indexing factored in terms of *approximation modality* $\triangleright$, called "later" [Appel et al., 2007].

$$\llbracket i \mid \mu\alpha.A \rrbracket\rho \triangleq \{\mathtt{fold}(e) \mid \forall j < i.\ e \in \llbracket j \mid A \rrbracket(\rho, \alpha \mapsto \llbracket j+1 \mid \mu\alpha.A \rrbracket\rho)\}$$

$$\llbracket i \mid A \to B \rrbracket\rho \triangleq \{\lambda x.e \mid \forall j \leq i.\ \forall v \in \llbracket j \mid A \rrbracket\rho.\ e[v/x] \in \llbracket j \mid B \rrbracket\rho\}$$

Using ideas from Nakano [2000], abstract version of step-indexing factored in terms of *approximation modality* $\triangleright$, called "later" [Appel et al., 2007].

$$\llbracket \mu\alpha.A \rrbracket\rho \triangleq \mathbf{fix}\,\mathcal{R}.\,\{\mathtt{fold}(e) \mid \triangleright(e \in \llbracket A \rrbracket(\rho, \alpha \mapsto \mathcal{R}))\}$$
$$\llbracket A \to B \rrbracket\rho \triangleq \{\lambda x.e \mid \forall v \in \llbracket A \rrbracket\rho.\, e[v/x] \in \llbracket B \rrbracket\rho\}$$

Using ideas from Nakano [2000], abstract version of step-indexing factored in terms of *approximation modality* $\triangleright$, called "later" [Appel et al., 2007].

$$\llbracket \mu\alpha.A \rrbracket\rho \triangleq \textbf{fix}\, \mathcal{R}.\, \{\texttt{fold}(e) \mid \triangleright(e \in \llbracket A \rrbracket(\rho, \alpha \mapsto \mathcal{R}))\}$$
$$\llbracket A \to B \rrbracket\rho \triangleq \{\lambda x.e \mid \forall v \in \llbracket A \rrbracket\rho.\, e[v/x] \in \llbracket B \rrbracket\rho\}$$

not just for predicates! can also form guarded-recursive "sets":

$$\text{gstream} \cong \mathbb{N} \times \triangleright\text{gstream}$$

Programming Example: Guarded Streams

```
type gstr[X] = cons of X * |> gstr[X]

let head (xs : gstr[X]) : X =
  let cons (x, _) = xs in x

let tail (xs : gstr[X]) : |> gstr[X] =
  let cons (_, ys) = xs in ys

let zipWith (f : X -> Y -> Z) : gstr[X] -> gstr[Y] -> gstr[Z] =
  gfix F in
    fun (cons (x, xs)) (cons (y, ys)) ->
      cons (f x y, F <*> xs <*> ys)
```

Can't write unguarded `tail` function! Guarded-recursive types ensure that all observations are *causal*.

Constant modality

Models of guarded recursion often include another modal operator $\Box$ which *neutralizes* $\triangleright$:

$$\Box A \to A \qquad \Box A \to \Box\Box A \qquad \Box \triangleright A \to \Box A$$

Converts guarded recursion to coinduction.

$$\text{gstream} \cong \mathbb{N} \times \triangleright\text{gstream}$$

Constant modality

Models of guarded recursion often include another modal operator $\square$ which *neutralizes* $\triangleright$:

$$\square A \to A \qquad \square A \to \square\square A \qquad \square \triangleright A \to \square A$$

Converts guarded recursion to coinduction.

$$\mathsf{gstream} \cong \mathbb{N} \times \triangleright \mathsf{gstream}$$
$$\square\mathsf{gstream} \cong \mathbb{N} \times \square\mathsf{gstream}$$

Constant modality

Models of guarded recursion often include another modal operator $\Box$ which *neutralizes* $\triangleright$:

$$\Box A \to A \qquad \Box A \to \Box\Box A \qquad \Box \triangleright A \to \Box A$$

Converts guarded recursion to coinduction.

$$\text{gstream} \cong \mathbb{N} \times \triangleright\text{gstream}$$
$$\text{sequence} \cong \mathbb{N} \times \text{sequence}$$

Constant modality

Models of guarded recursion often include another modal operator $\Box$ which *neutralizes* $\triangleright$:

$$\Box A \to A \qquad \Box A \to \Box\Box A \qquad \Box \triangleright A \to \Box A$$

Converts guarded recursion to coinduction.

$$\text{gstream} \cong \mathbb{N} \times \triangleright\text{gstream}$$
$$\text{sequence} \cong \mathbb{N} \times \text{sequence}$$

Improved version [Atkey and McBride, 2013]: index $\triangleright_\kappa$ in "clocks" κ, re-cast $\Box$ as quantifier $\forall\kappa$.

Constant modality

Models of guarded recursion often include another modal operator $\Box$ which *neutralizes* $\triangleright$:

$$\Box A \to A \qquad \Box A \to \Box\Box A \qquad \Box \triangleright A \to \Box A$$

Converts guarded recursion to coinduction.

$$\mathsf{gstream}_\kappa \cong \mathbb{N} \times \triangleright_\kappa \mathsf{gstream}_\kappa$$
$$\forall\kappa.\mathsf{gstream}_\kappa \cong \mathbb{N} \times \forall\kappa.\mathsf{gstream}_\kappa$$

Improved version [Atkey and McBride, 2013]: index $\triangleright_\kappa$ in “clocks” κ, re-cast $\Box$ as quantifier $\forall\kappa$.

Main task: develop powerful metalanguages for guarded domain-theoretic semantics and programming.

Main task: develop powerful metalanguages for guarded domain-theoretic semantics and programming.

Most advances in the area of guarded higher-order logics, but **dependent type theory** indispensible.

Main task: develop powerful metalanguages for guarded domain-theoretic semantics and programming.

Most advances in the area of guarded higher-order logics, but **dependent type theory** indispensible.

Dependent type theory = semantic framework to unify the study of *programs* with the study of *programming languages.*

Guarded Dependent Type Theory

Bizjak and Møgelberg [2017] present elegant denotational account of guarded dependent type theory ($\Pi, \Sigma, \triangleright_{\kappa}, \forall\kappa, \ldots$)

A couple things gave us pause...

Guarded Dependent Type Theory

Bizjak and Møgelberg [2017] present elegant denotational account of guarded dependent type theory ($\Pi, \Sigma, \triangleright_\kappa, \forall\kappa, \ldots$)

A couple things gave us pause...

1. what about operational meaning? (status unknown for GDTT)

Guarded Dependent Type Theory

Bizjak and Møgelberg [2017] present elegant denotational account of guarded dependent type theory ($\Pi, \Sigma, \triangleright_\kappa, \forall\kappa, \ldots$)

A couple things gave us pause...

1. what about operational meaning? (status unknown for GDTT)
2. universes $\mathcal{U}_i$ are essential, but not directly supported in GDTT semantics

What's the deal with universes?

Adequacy of $\forall\kappa$ to encode coinduction requires that functions of clocks are constant. Called **clock irrelevance**.

But what about $(\lambda\kappa.\lambda A.\ \triangleright_{\kappa} A) \in \forall\kappa.(\mathcal{U} \to \mathcal{U})$?

What's the deal with universes?

Adequacy of $\forall\kappa$ to encode coinduction requires that functions of clocks are constant. Called **clock irrelevance**.

But what about $(\lambda\kappa.\lambda A.\ \triangleright_\kappa A) \in \forall\kappa.(\mathcal{U} \to \mathcal{U})$?

global orthogonality constraint

What's the deal with universes?

Adequacy of $\forall\kappa$ to encode coinduction requires that functions of clocks are constant. Called **clock irrelevance**.

But what about $(\lambda\kappa.\lambda A.\ \triangleright_{\kappa} A) \in \forall\kappa.(\mathcal{U} \to \mathcal{U})$?

global orthogonality constraint

Idea: get rid of ordinary universes, replace with weaker clock-context-indexed universes $\mathcal{U}_i^{\vec{\kappa}}$.

Guarded Computational Type Theory

Experience in behavioral type theoretic tradition (MLTT 1979, Nuprl/CTT) led us to believe we could surmount these problems.

Guarded Computational Type Theory

Experience in behavioral type theoretic tradition (MLTT 1979, Nuprl/CTT) led us to believe we could surmount these problems.

Idea: types aren't abstract collections of abstract things. Types are specifications of *program behavior.* Natural for programs to exhibit multiple behaviors, have multiple types.

Guarded Computational Type Theory

Experience in behavioral type theoretic tradition (MLTT 1979, Nuprl/CTT) led us to believe we could surmount these problems.

Idea: types aren't abstract collections of abstract things. Types are specifications of *program behavior.* Natural for programs to exhibit multiple behaviors, have multiple types.

Behavioral perspective led us immediately to an alternative solution (hard to see from more abstract vantage point).

Guarded Computational Type Theory

Experience in behavioral type theoretic tradition (MLTT 1979, Nuprl/CTT) led us to believe we could surmount these problems.

Idea: types aren't abstract collections of abstract things. Types are specifications of *program behavior*. Natural for programs to exhibit multiple behaviors, have multiple types.

Behavioral perspective led us immediately to an alternative solution (hard to see from more abstract vantage point).

We call it Guarded Computational Type Theory / CTT🕒.

Solution to universe problem

Inspired by extension of propositions-as-types discipline to *intersection* connective [Bickford and Constable, 2012, Allen et al., 2006].

Decompose clock quantifiers into two parts:

1. *uniform* quantifier (intersection): $\{\kappa \div \mathtt{clk}\} \to A_\kappa$
2. *non-uniform* quantifier (product): $(\kappa : \mathtt{clk}) \to A_\kappa$

Idea: don't impose global orthogonality condition, instead change the quantifier.

Solution to universe problem

Inspired by extension of propositions-as-types discipline to *intersection* connective [Bickford and Constable, 2012, Allen et al., 2006].

Decompose clock quantifiers into two parts:

1. *uniform* quantifier (intersection): $\{\kappa \div \texttt{clk}\} \to A_\kappa$
2. *non-uniform* quantifier (product): $(\kappa : \texttt{clk}) \to A_\kappa$

Idea: don't impose global orthogonality condition, instead change the quantifier.

Programs have more than one behavior: in particular, the specification $M \in \{\kappa \div \texttt{clk}\} \to A_\kappa$ means that M exhibits *all* the behaviors A_κ simultaneously.

"Who among us...?"

Initial version of **CTT**⌚ had subtle+critical bug, wasn't clear if it could be salvaged. Need for formalization.

Formalizing CTT$^{\circleddash}$

To gain confidence, formalized corrected version in **Coq** using internal topos-theoretic technique.

Basic idea: design a suitable presheaf topos $\mathcal{S}_{\circleddash}$ whose internal logic has clocks, $\triangleright_\kappa$ and $\forall\kappa$. See our paper, but think of presheaves on finitely many copies of ω.

The logic of clocks

There is a *presheaf of clocks* $\mathbb{K} : \mathcal{S}_{🕒}$.

The topos $\mathcal{S}_{🕒}$ contains a rich higher-order logic with a $\mathbb{K}$-indexed family of modalities $\triangleright_\kappa$, enjoying crucial principles including the following:

IRRELEVANCE

$\exists \kappa : \mathbb{K}.\, \top$

"IMPATIENCE"

$\forall \phi : \Omega^{\mathbb{K}}.\, (\forall \kappa : \mathbb{K}.\, \triangleright_\kappa \phi(\kappa)) \Rightarrow \forall \kappa : \mathbb{K}.\, \phi(\kappa)$

MONOTONICITY

$\forall \kappa : \mathbb{K}.\, \forall \phi : \Omega.\, \phi \Rightarrow \triangleright_\kappa \phi$

$\wedge$-DISTRIBUTIVITY

$\forall \kappa : \mathbb{K}.\, \forall \phi, \psi : \Omega.\, \triangleright_\kappa (\phi \wedge \psi) \equiv (\triangleright_\kappa \phi \wedge \triangleright_\kappa \psi)$

$\Rightarrow$-DISTRIBUTIVITY

$\forall \kappa : \mathbb{K}.\, \forall \phi, \psi : \Omega.\, \triangleright_\kappa (\phi \Rightarrow \psi) \equiv (\triangleright_\kappa \phi \Rightarrow \triangleright_\kappa \psi)$

STRONG LÖB INDUCTION

$\forall \kappa : \mathbb{K}.\, \forall \phi : \Omega.\, (\triangleright_\kappa \phi \Rightarrow \phi) \Rightarrow \phi$

Internal Syntax and Operational Semantics

Key idea: higher-order internal encoding of syntax for later modality and clock quantification[1], De Bruijn encoding for ordinary binders:

$$\frac{\kappa : \mathbb{K} \qquad A : \mathcal{P}rog}{\blacktriangleright_{\kappa} A : \mathcal{P}rog}$$

$$\frac{A : \mathcal{P}rog^{\mathbb{K}}}{(\kappa : \texttt{clk}) \to A(\kappa) : \mathcal{P}rog} \qquad \frac{A : \mathcal{P}rog^{\mathbb{K}}}{\{\kappa \div \texttt{clk}\} \to A(\kappa) : \mathcal{P}rog}$$

[1]See Fiore et al. [1999], Hofmann [1999].

Next, we construct the PER-model of type theory inside the *internal logic* of $\mathcal{S}_{\circlearrowleft}$.

Idea: internal inductive-recursive definition of a universe hierarchy, assigning relations to type-codes.

All clauses are standard, except the relations which define the clock-related connectives; approximately:

$$
\begin{aligned}
[\![\blacktriangleright_\kappa A]\!] &= \{(M, N) \mid \triangleright_\kappa((M, N) \in [\![A]\!])\} \\
[\![(\kappa : \mathtt{clk}) \to A(\kappa)]\!] &= \{(M, N) \mid \forall \kappa : \mathbb{K}.\, (\mathtt{app}(M, \kappa), \mathtt{app}(N, \kappa)) \in [\![A(\kappa)]\!]\} \\
[\![\{\kappa \div \mathtt{clk}\} \to A(\kappa)]\!] &= \{(M, N) \mid \forall \kappa : \mathbb{K}.\, (M, N) \in [\![A(\kappa)]\!]\} \\
&\vdots
\end{aligned}
$$

Programming in CTT$^{\circlearrowleft}$

Using a combination of both clock quantifiers and the later modality, we can program the type of guarded-recursive streams "qua fixed points on universes".[2]

$$\mathsf{gstr} \in (\_ : \mathtt{clk}) \to \mathbb{U}_0 \to \mathbb{U}_0$$
$$\mathsf{gstr} = \lambda\kappa.\, \mathtt{fix}\, F\, \mathtt{in}\, \lambda A.\, A \times \blacktriangleright_\kappa F(A)$$

$$\mathsf{seq} \in \mathbb{U}_0 \to \mathbb{U}_0$$
$$\mathsf{seq} = \lambda A.\, \{\kappa \div \mathtt{clk}\} \to \mathsf{gstr}\, \kappa\, A$$

$$\mathsf{gzipwith} \in (f : X \to Y \to Z) \to \{\kappa \div \mathtt{clk}\} \to \mathsf{gstr}\, \kappa\, X \to \mathsf{gstr}\, \kappa\, Y \to \mathsf{gstr}\, \kappa\, Z$$
$$\mathsf{gzipwith} = \lambda f.\, \mathtt{fix}\, F\, \mathtt{in}\, \lambda\alpha, \beta.\, \langle f\, \alpha.1\, \beta.1, F\, \alpha.2\, \beta.2\rangle$$

$$\mathsf{zipwith} \in (f : X \to Y \to Z) \to \mathsf{seq}\, X \to \mathsf{seq}\, Y \to \mathsf{seq}\, Z$$
$$\mathsf{zipwith} = \mathsf{gzipwith}$$

[2] Birkedal and Møgelberg [2013]

Summary of contribution

In **Guarded Computational Type Theory** / **CTT**$^{\circlearrowleft}$, we propose a guarded dependent type theory which supports a predicative hierarchy of standard/*non-indexed* type-theoretic universes $\mathtt{U}_i$.

1. standard dependent type connectives (Π,Σ, $\mathtt{Eq}$)
2. clock-indexed later modality ($\blacktriangleright_\kappa A$)
3. parametric clock quantifier ($\{\kappa \div \mathtt{clk}\} \to A$)
4. non-parametric clock quantifier ($(\kappa : \mathtt{clk}) \to A$)
5. type-theoretic universes $\mathtt{U}_i$ which contain $\blacktriangleright_\kappa A$
6. operational semantics, canonicity result
7. **formalized in Coq using internal-topos-theoretic technique**

(questions?)

Bibliography I

Andreas Abel, Andrea Vezzosi, and Theo Winterhalter. Normalization by evaluation for sized dependent types. *Proc. ACM Program. Lang.*, 1(ICFP):33:1–33:30, August 2017. ISSN 2475-1421.

S.F. Allen, M. Bickford, R.L. Constable, R. Eaton, C. Kreitz, L. Lorigo, and E. Moran. Innovations in computational type theory using nuprl. *Journal of Applied Logic*, 4(4):428 – 469, 2006. ISSN 1570-8683. Towards Computer Aided Mathematics.

Andrew W. Appel and David McAllester. An indexed model of recursive types for foundational proof-carrying code. *ACM Trans. Program. Lang. Syst.*, 23(5):657–683, September 2001. ISSN 0164-0925. doi: 10.1145/504709.504712. URL `http://doi.acm.org/10.1145/504709.504712`.

Bibliography II

Andrew W. Appel, Paul-André Melliès, Christopher D. Richards, and Jérôme Vouillon. A very modal model of a modern, major, general type system. In *Proceedings of the 34th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '07, pages 109–122, New York, NY, USA, 2007. ACM. ISBN 1-59593-575-4.

Robert Atkey and Conor McBride. Productive coprogramming with guarded recursion. In *Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming*, ICFP '13, pages 197–208, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2326-0.

P. Bahr, H. B. Grathwohl, and R. E. Møgelberg. The clocks are ticking: No more delays! In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12, June 2017.

Bibliography III

Mark Bickford and Robert L. Constable. *Polymorphic Logic*. Ontos Verlag, 2012. Festschrift for Helmut Schwichtenberg.

L. Birkedal and R. E. Møgelberg. Intensional type theory with guarded recursive types qua fixed points on universes. In *2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 213–222, June 2013. doi: 10.1109/LICS.2013.27.

Lars Birkedal, Rasmus Ejlers Møgelberg, Jan Schwinghammer, and Kristian Stovring. 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*, LICS '11, pages 55–64, Washington, DC, USA, 2011. IEEE Computer Society. ISBN 978-0-7695-4412-0.

Bibliography IV

Lars Birkedal, Aleš Bizjak, Ranald Clouston, Hans Bugge Grathwohl, Bas Spitters, and Andrea Vezzosi. Guarded Cubical Type Theory: Path Equality for Guarded Recursion. In Jean-Marc Talbot and Laurent Regnier, editors, *25th EACSL Annual Conference on Computer Science Logic (CSL 2016)*, volume 62 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 23:1–23:17, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN 978-3-95977-022-4.

Aleš Bizjak, Lars Birkedal, and Marino Miculan. A model of countable nondeterminism in guarded type theory. In Gilles Dowek, editor, *Rewriting and Typed Lambda Calculi*, pages 108–123, Cham, 2014. Springer International Publishing. ISBN 978-3-319-08918-8.

Bibliography V

Aleš Bizjak, Hans Bugge Grathwohl, Ranald Clouston, Rasmus E. Møgelberg, and Lars Birkedal. Guarded dependent type theory with coinductive types. In Bart Jacobs and Christof Löding, editors, *Foundations of Software Science and Computation Structures: 19th International Conference, FOSSACS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2–8, 2016, Proceedings*, pages 20–35, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg. ISBN 978-3-662-49630-5.

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

Aleš Bizjak and Rasmus Ejlers Møgelberg. A model of guarded recursion with clock synchronisation. *Electron. Notes Theor. Comput. Sci.*, 319(C):83–101, December 2015. ISSN 1571-0661.

Bibliography VI

Aleš Bizjak and Rasmus Ejlers Møgelberg. Denotational semantics for guarded dependent type theory. Draft, 2017.

R. L. Constable, S. F. Allen, H. M. Bromley, W. R. Cleaveland, J. F. Cremer, R. W. Harper, D. J. Howe, T. B. Knoblock, N. P. Mendler, P. Panangaden, J. T. Sasaki, and S. F. Smith. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1986. ISBN 0-13-451832-2.

D. Dreyer, A. Ahmed, and L. Birkedal. Logical step-indexed logical relations. In *2009 24th Annual IEEE Symposium on Logic In Computer Science*, pages 71–80, Aug 2009. doi: 10.1109/LICS.2009.34.

Marcelo Fiore, Gordon Plotkin, and Daniele Turi. Abstract syntax and variable binding. In *Proceedings of the 14th Symposium on Logic in Computer Science*, pages 193–202, 1999.

Bibliography VII

Martin Hofmann. Semantical analysis of higher-order abstract syntax. In *Proceedings of the 14th Annual IEEE Symposium on Logic in Computer Science*, LICS '99, pages 204–, Washington, DC, USA, 1999. IEEE Computer Society. ISBN 0-7695-0158-3. URL `http://dl.acm.org/citation.cfm?id=788021.788940`.

H. Nakano. A modality for recursion. In *Proceedings Fifteenth Annual IEEE Symposium on Logic in Computer Science (Cat. No.99CB36332)*, pages 255–266, New York, 2000. IEEE Computer Society.

Marco Paviotti, Rasmus Ejlers Møgelberg, and Lars Birkedal. A model of PCF in Guarded Type Theory. *Electronic Notes in Theoretical Computer Science*, 319(Supplement C):333 – 349, 2015. ISSN 1571-0661. The 31st Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXI).