Decalf: A Directed, Effectful Cost-Aware Logical Framework

Harrison Grodin[1] Yue Niu[1] Jonathan Sterling[2] Robert Harper[1]

POPL 2024

[1]Carnegie Mellon University

[2]University of Cambridge

Motivation

Example (List Map)

$$map : (A \rightarrow B) \rightarrow \mathrm{list}(A) \rightarrow \mathrm{list}(B)$$
$$map\ f\ [] = []$$
$$map\ f\ (x :: xs) = f\ x :: map\ f\ xs$$

Motivation

Example (List Map)

$$map : (A \to B) \to \mathsf{list}(A) \to \mathsf{list}(B)$$
$$map\ f\ [] = []$$
$$map\ f\ (x :: xs) = f\ x :: map\ f\ xs$$

Goal

What is the cost of *map*?

- higher-order function
- argument may perform effects

decalf (embedded in Agda) gives an elegant, linguistic answer.

Core Language

- ↯ effects
- $\leq$ program inequality
- φ phase distinction

Effects in Call-By-Push-Value [Levy, 2003]

Value Types

$$
\begin{aligned}
A, B, C ::= \; & \mathsf{U}(X) \\
& 0 \quad A + B \\
& 1 \quad A \times B \\
& A \to B \\
& \mathsf{nat} \\
& \mathsf{list}(A) \\
& \vdots
\end{aligned}
$$

Computation Types

$$
\begin{aligned}
X, Y, Z ::= \; & \mathsf{F}(A) \\
& 1 \quad X \times Y \\
& A \to X \\
& \vdots
\end{aligned}
$$

Effects in Call-By-Push-Value [Levy, 2003]

Value Types

$$A, B, C ::= \mathsf{U}(X) \mid 0 \mid A + B \mid 1 \mid A \times B \mid A \to B \mid \mathsf{nat} \mid \mathsf{list}(A) \mid \vdots$$

Computation Types

$$X, Y, Z ::= \mathsf{F}(A) \mid 1 \mid X \times Y \mid A \to X \mid \vdots$$

These support effects while retaining equational reasoning principles (*e.g.*, β/η equality and pointwise function equality).

Abstract Cost as an Effect [Danielsson, 2008]

Assume some value type $\mathbb{C}$ representing cost monoid (*e.g.*, $(\mathbb{N}, 0, +)$).

Definition (Cost Effect)

$$\frac{\Gamma \vdash c : \mathbb{C} \qquad \Gamma \vdash e : X}{\Gamma \vdash \mathsf{step}^{c}(e) : X}$$

$$\mathsf{step}^{0}(e) = e$$
$$\mathsf{step}^{c_1}(\mathsf{step}^{c_2}(e)) = \mathsf{step}^{c_1 + c_2}(e)$$

$\leq$ Exact Cost [Niu et al., 2022]

Big Idea

To show an exact cost bound, use program equality.

Big Idea

To show an exact cost bound, use program equality.

Definition (Exact Cost in calf)

$$\mathrm{hasCost}_A(e, c) := \sum_{a:A} (e = \mathrm{step}^c(\mathrm{ret}(a)))$$

Exact Cost [Niu et al., 2022]

Big Idea

To show an exact cost bound, use program equality.

Definition (Exact Cost in calf)

$$\mathrm{hasCost}_A(e, c) := \sum_{a:A} (e = \mathrm{step}^c(\mathrm{ret}(a)))$$

Example (Merge Sort)

For all l : list(nat), we have $\mathrm{hasCost}(\mathit{msort}\ l, |l| \log_2 |l|)$.

$\leq$ Inequality [Licata and Harper, 2011, Riehl and Shulman, 2017]

$$\boxed{e \leq_X e'}$$

Intuition

Both e and e' compute the same result, but e may be cheaper.

$\leq$ Inequality [Licata and Harper, 2011, Riehl and Shulman, 2017]

$$\boxed{e \leq_X e'}$$

Intuition

Both e and e' compute the same result, but e may be cheaper.

Example

$$\mathsf{step}^3(\mathsf{ret}(\texttt{"hi"})) \leq_{\mathsf{F}(\mathsf{string})} \mathsf{step}^{12}(\mathsf{ret}(\texttt{"hi"}))$$

Inequality [Licata and Harper, 2011, Riehl and Shulman, 2017]

$$\boxed{e \leq_X e'}$$

Intuition

Both e and e' compute the same result, but e may be cheaper.

Example

$$\mathsf{step}^3(\mathsf{ret}(\texttt{"hi"})) \leq_{\mathsf{F(string)}} \mathsf{step}^{12}(\mathsf{ret}(\texttt{"hi"}))$$

Remark

Inequality verifies cost *and* behavior by comparing programs.

$\leq$ Inequality [Licata and Harper, 2011, Riehl and Shulman, 2017]

$$\boxed{e \leq_X e'}$$

Intuition

Both e and e' compute the same result, but e may be cheaper.

Example

$$\mathsf{step}^3(\mathsf{ret}(\texttt{"hi"})) \leq_{\mathsf{F(string)}} \mathsf{step}^{12}(\mathsf{ret}(\texttt{"hi"}))$$

Remark

Inequality verifies cost *and* behavior by comparing programs.

Remark

Inspired by *directed type theory*.

$\leq$ Inexact Cost

Big Idea

To show an <u>in</u>exact cost bound, use program <u>in</u>equality.

Inexact Cost

Big Idea

To show an inexact cost bound, use program inequality.

Definition (Inexact Cost)

$$\mathsf{isBounded}_A(e, c) :\equiv \sum_{a:A} (e \leq \mathsf{step}^c(\mathsf{ret}(a)))$$

Big Idea

To show an inexact cost bound, use program inequality.

Definition (Inexact Cost)

$$\mathrm{isBounded}_A(e, c) := \sum_{a:A} (e \leq \mathrm{step}^c(\mathrm{ret}(a)))$$

Example (Insertion Sort)

For all l : list(nat), we have isBounded($isort\ l, |l|^2$).

$\leq$ Equality vs. Inequality

Equality $=$

- reflexive
- transitive
- symmetric
- congruence:
  $a = a'$ implies $f(a) = f(a')$
- pointwise on functions

Inequality $\leq$

- reflexive
- transitive
- N/A
- monotone:
  $a \leq a'$ implies $f(a) \leq f(a')$
- pointwise on functions

$\leq$ Equality vs. Inequality

Equality $=$

- reflexive
- transitive
- symmetric
- congruence:
  $a = a'$ implies $f(a) = f(a')$
- pointwise on functions

Inequality $\leq$

- reflexive
- transitive
- N/A
- monotone:
  $a \leq a'$ implies $f(a) \leq f(a')$
- pointwise on functions

Compositional cost analysis via inequality reasoning.

Example (List Insert)

$$
\begin{aligned}
&\mathit{insert} : \mathsf{nat} \to \mathsf{list}(\mathsf{nat}) \to \mathsf{F}(\mathsf{list}(\mathsf{nat}))\\
&\mathit{insert}\ x\ [] = \mathsf{ret}(x :: [])\\
&\mathit{insert}\ x\ (y :: ys) =\\
&\quad \mathsf{bind}\ b \leftarrow \mathsf{step}^1(x \leq^? y)\ \mathsf{in}\\
&\quad \mathsf{if}\ b\ \mathsf{then}\ \mathsf{ret}(x :: y :: ys)\ \mathsf{else}\\
&\quad \mathsf{bind}\ ys' \leftarrow \mathit{insert}\ x\ ys\ \mathsf{in}\ \mathsf{ret}(y :: ys')
\end{aligned}
$$

Example (List Insert)

$$
\begin{aligned}
&\mathit{insert} : \mathsf{nat} \to \mathsf{list}(\mathsf{nat}) \to \mathsf{F}(\mathsf{list}(\mathsf{nat})) \\
&\mathit{insert}\ x\ [] = \mathsf{ret}(x :: []) \\
&\mathit{insert}\ x\ (y :: ys) = \\
&\quad \mathsf{bind}\ b \leftarrow \mathsf{step}^1(x \leq^? y)\ \mathsf{in} \\
&\quad \mathsf{if}\ b\ \mathsf{then}\ \mathsf{ret}(x :: y :: ys)\ \mathsf{else} \\
&\quad \mathsf{bind}\ ys' \leftarrow \mathit{insert}\ x\ ys\ \mathsf{in}\ \mathsf{ret}(y :: ys')
\end{aligned}
$$

Theorem (Closed Form Bound)

$$\mathit{insert} \leq \lambda x.\ \lambda l.\ \mathsf{step}^{|l|}(\mathsf{ret}(\mathit{insert}_{spec}\ x\ l))$$

Theorem (Closed Form Bound)

$$\mathit{insert} \leq \lambda x.\, \lambda l.\, \mathsf{step}^{|l|}(\mathsf{ret}(\mathit{insert_{spec}}\ x\ l))$$

Proof Excerpt.

$$
\begin{aligned}
&\mathsf{begin}\\
&\quad \mathsf{step}^1(\mathsf{bind}\ \mathit{ys}' \leftarrow \mathit{insert}\ x\ \mathit{ys}\ \mathsf{in}\ \mathsf{ret}(x :: \mathit{ys}'))\\
&\leq\langle \text{ monotonicity, IH } \rangle\\
&\quad \mathsf{step}^1(\mathsf{bind}\ \mathit{ys}' \leftarrow \mathsf{step}^{|\mathit{ys}|}(\mathsf{ret}(\mathit{insert}_{\mathsf{spec}}\ x\ \mathit{ys}))\ \mathsf{in}\ \mathsf{ret}(x :: \mathit{ys}'))\\
&=\langle\rangle\\
&\quad \mathsf{step}^{1+|\mathit{ys}|}(\mathsf{ret}(y :: \mathit{insert}_{\mathsf{spec}}\ x\ \mathit{ys}))\\
&=\langle\rangle\\
&\quad \mathsf{step}^{|y::\mathit{ys}|}(\mathsf{ret}(\mathit{insert}_{\mathsf{spec}}\ x\ \mathit{ys}))\\
&\blacksquare
\end{aligned}
$$

Extensional Phase [Sterling and Harper, 2021, Sterling, 2021]

Definition (Extensional Phase)

Proposition ext for isolating behavior. If ext holds:

- $\mathbb{C} \cong \mathbf{1}$
- $a \leq a'$ implies $a = a'$

Modality $\bigcirc A := (\mathsf{ext} \to A)$ isolates behavioral part of A.

Extensional Phase [Sterling and Harper, 2021, Sterling, 2021]

Definition (Extensional Phase)

Proposition ext for isolating behavior. If ext holds:

- $\mathbb{C} \cong \mathbf{1}$
- $a \leq a'$ implies $a = a'$

Modality $\bigcirc A := (\mathsf{ext} \to A)$ isolates behavioral part of A.

Corollary (Noninterference)

If $\bigcirc A \cong \mathbf{1}$, then every function $A \to \bigcirc B$ is constant.

Cost does not impact behavior.

Cost Removal [Niu et al., 2022]

Corollary (Cost Removal)

If ext *holds, then* $\mathbb{C} \cong \mathbf{1}$. *So, every* $c : \mathbb{C}$ *equals* 0*:*

$$\mathsf{step}^c(e) = \mathsf{step}^0(e) = e$$

Cost Removal [Niu et al., 2022]

Corollary (Cost Removal)

If ext *holds, then* $\mathbb{C} \cong \mathbf{1}$*. So, every* $c : \mathbb{C}$ *equals* 0*:*

$$\mathsf{step}^c(e) = \mathsf{step}^0(e) = e$$

Example

If ext holds, then *isort* = *msort*.

Effects

Effects in decalf [Plotkin and Power, 2002]

decalf supports algebraic effects beyond cost.

Examples:

- errors
- nondeterminism
- probabilistic choice
- global state

Probabilistic Choice

Definition (Biased Coin Flip)

$$\frac{\Gamma \vdash p : \mathbb{Q}_{[0,1]} \qquad \Gamma \vdash e_0 : X \qquad \Gamma \vdash e_1 : X}{\Gamma \vdash \mathrm{flip}_p(e_0,\, e_1) : X}$$

$$\begin{aligned} \mathrm{flip}_p(e_0,\, e_1) &= \mathrm{flip}_{1-p}(e_1,\, e_0) \\ \mathrm{flip}_p(e,\, e) &= e \\ &\vdots \end{aligned}$$

$$\mathrm{step}^c(\mathrm{flip}_p(e_0,\, e_1)) = \mathrm{flip}_p(\mathrm{step}^c(e_0),\, \mathrm{step}^c(e_1))$$

Randomized Quicksort [Hoare, 1961, Hoare, 1962]

Example

Randomized parallel quicksort:

$$qsort : \mathsf{list}(\mathsf{nat}) \to \mathsf{F}(\mathsf{list}(\mathsf{nat}))$$

Benign randomization; same value always returned. So:

$$qsort \leq \lambda l.\, \mathsf{step}^{|l|^2}(\mathsf{ret}(sort_{\mathsf{spec}}\ l))$$

Proof by induction.

Randomized Quicksort [Hoare, 1961, Hoare, 1962]

Example

Randomized parallel quicksort:

$$qsort : \mathsf{list}(\mathsf{nat}) \to \mathsf{F}(\mathsf{list}(\mathsf{nat}))$$

Benign randomization; same value always returned. So:

$$qsort \leq \lambda l.\, \mathsf{step}^{|l|^2}(\mathsf{ret}(sort_{\mathsf{spec}}\ l))$$

Proof by induction.

Corollary (Correctness)

$$\bigcirc(qsort = \lambda l.\, \mathsf{ret}(sort_{spec}\ l))$$

Random Sublist

Example (Random Sublist)

$$
\begin{aligned}
&\mathit{sublist} : \mathsf{list}(\mathsf{nat}) \to \mathsf{F}(\mathsf{list}(\mathsf{nat}))\\
&\mathit{sublist}\ [] = \mathsf{ret}([])\\
&\mathit{sublist}\ (x :: xs) =\\
&\quad \mathsf{bind}\ xs' \leftarrow \mathit{sublist}\ xs\ \mathsf{in}\\
&\quad \mathsf{flip}_{\frac{1}{2}}(\mathsf{ret}(xs'), \mathsf{step}^1(\mathsf{ret}(x :: xs')))
\end{aligned}
$$

Random Sublist

Example (Random Sublist)

$$
\begin{aligned}
&\mathit{sublist} : \mathsf{list}(\mathsf{nat}) \to \mathsf{F}(\mathsf{list}(\mathsf{nat}))\\
&\mathit{sublist}\ [] = \mathsf{ret}([])\\
&\mathit{sublist}\ (x :: xs) =\\
&\quad \mathsf{bind}\ xs' \leftarrow \mathit{sublist}\ xs\ \mathsf{in}\\
&\quad \mathsf{flip}_{\frac{1}{2}}(\mathsf{ret}(xs'), \mathsf{step}^1(\mathsf{ret}(x :: xs')))
\end{aligned}
$$

Example (Binomial Cost)

$$
\begin{aligned}
&\mathit{binomial} : \mathsf{nat} \to \mathsf{F}(1)\\
&\mathit{binomial}\ \mathsf{zero} = \mathsf{ret}(\star)\\
&\mathit{binomial}\ (\mathsf{suc}(n)) =\\
&\quad \mathsf{flip}_{\frac{1}{2}}(\mathit{binomial}\ n, \mathsf{step}^1(\mathit{binomial}\ n))
\end{aligned}
$$

Random Sublist Analysis

Definition (Result Erasure)

$$\|-\| : \mathsf{F}(A) \to \mathsf{F}(1)$$
$$\|e\| = e \,;\, \mathsf{ret}(\star)$$

Random Sublist Analysis

Definition (Result Erasure)

$$\|-\| : \mathsf{F}(A) \to \mathsf{F}(1)$$
$$\|e\| = e \mathbin{;} \mathsf{ret}(\star)$$

Theorem (Random Sublist Cost)

$$\lambda l.\, \|\mathit{sublist}\ l\| = \lambda l.\, \mathit{binomial}\ |l|$$

Random Sublist Analysis

Definition (Result Erasure)

$$\|-\| : \mathsf{F}(A) \to \mathsf{F}(1)$$
$$\|e\| = e \,;\, \mathsf{ret}(\star)$$

Theorem (Random Sublist Cost)

$$\begin{aligned} \lambda l.\, \|\mathit{sublist}\ l\| &= \lambda l.\, \mathit{binomial}\ |l| \\ &\leq \lambda l.\, \mathsf{step}^{|l|}(\mathsf{ret}(\star)) \end{aligned}$$

List Map

Example (List Map)

$$
\begin{aligned}
&\mathit{map} : \mathrm{U}(A \to \mathrm{F}(B)) \to \mathrm{list}(A) \to \mathrm{F}(\mathrm{list}(B)) \\
&\mathit{map}\ f\ [] = \mathrm{ret}([]) \\
&\mathit{map}\ f\ (x :: \mathit{xs}) = \\
&\quad \mathrm{bind}\ \mathit{ys} \leftarrow \mathit{map}\ f\ \mathit{xs}\ \mathrm{in} \\
&\quad \mathrm{bind}\ y \leftarrow f\ x\ \mathrm{in} \\
&\quad \mathrm{ret}(y :: \mathit{ys})
\end{aligned}
$$

If f can perform arbitrary effects, there's no hope for a succinct, informative bound!

List Map Bounds

Theorem (Trivial Bound)

Always, map $\leq$ map.

List Map Bounds

Theorem (Trivial Bound)

Always, map $\leq$ map.

Theorem (Pure Bound)

If $\|f\ x\| \leq \mathsf{step}^c(\mathsf{ret}(\star))$, *then*

$$\|map\ f\ l\| \leq \mathsf{step}^{c|l|}(\mathsf{ret}(\star)).$$

List Map Bounds

Theorem (Trivial Bound)

Always, map $\leq$ *map.*

Theorem (Pure Bound)

If $\|f\ x\| \leq \mathsf{step}^{c}(\mathsf{ret}(\star))$, *then*

$$\|map\ f\ l\| \leq \mathsf{step}^{c|l|}(\mathsf{ret}(\star)).$$

Theorem (Randomized Bound)

If $\|f\ x\| \leq binomial\ n$, *then*

$$\|map\ f\ l\| \leq binomial\ (n|l|).$$

Semantics

Inequality via an Interval [Hyland, 1991, Phoa, 1991]

Definition (Path Relation)

Let $(\mathbb{I}, 0, 1)$ be an interval. Then, the *path relation* $x \sqsubseteq_A y$ is:

$$\exists p \colon \mathbb{I} \to A.\ (p\ 0 = x) \wedge (p\ 1 = y)$$

Inequality via an Interval [Hyland, 1991, Phoa, 1991]

Definition (Path Relation)

Let $(\mathbb{I}, 0, 1)$ be an interval. Then, the *path relation* $x \sqsubseteq_A y$ is:

$$\exists p \colon \mathbb{I} \to A.\ (p\ 0 = x) \wedge (p\ 1 = y)$$

Goal

- ☑ reflexive
- ☐ transitive
- ☑ monotone: $a \sqsubseteq_A a'$ implies $f(a) \sqsubseteq_B f(a')$
- ☐ pointwise on functions
- ☐ extensionally discrete: $\bigcirc(x \sqsubseteq_A y)$ implies $\bigcirc(x = y)$

Inequality via an Interval [Hyland, 1991, Phoa, 1991]

Extensional Discreteness

Require that $\bigcirc\mathbb{I} \cong \mathbf{1}$.

Under ext, any map $\mathbb{I} \to A$ is constant, so $x \sqsubseteq_A y$ is $x = y$.

Goal

- ☑ reflexive
- ☐ transitive
- ☑ monotone: $a \sqsubseteq_A a'$ implies $f(a) \sqsubseteq_B f(a')$
- ☐ pointwise on functions
- ☐ extensionally discrete: $\bigcirc(x \sqsubseteq_A y)$ implies $\bigcirc(x = y)$

Inequality via an Interval [Hyland, 1991, Phoa, 1991]

Extensional Discreteness

Require that $\bigcirc\mathbb{I} \cong \mathbf{1}$.

Under ext, any map $\mathbb{I} \to A$ is constant, so $x \sqsubseteq_A y$ is $x = y$.

Goal

- ☑ reflexive
- ☐ transitive
- ☑ monotone: $a \sqsubseteq_A a'$ implies $f(a) \sqsubseteq_B f(a')$
- ☐ pointwise on functions
- ☑ extensionally discrete: $\bigcirc(x \sqsubseteq_A y)$ implies $\bigcirc(x = y)$

Inequality via an Interval [Hyland, 1991, Phoa, 1991]

Transitivity and Pointwise Ordering

Path relation $x \sqsubseteq_A y$ is not transitive/pointwise on all A. So, isolate a class of A's (reflective subuniverse) for which it is.

Goal

- [x] reflexive
- [ ] transitive
- [x] monotone: $a \sqsubseteq_A a'$ implies $f(a) \sqsubseteq_B f(a')$
- [ ] pointwise on functions
- [x] extensionally discrete: $\bigcirc(x \sqsubseteq_A y)$ implies $\bigcirc(x = y)$

Inequality via an Interval [Hyland, 1991, Phoa, 1991]

Transitivity and Pointwise Ordering

Path relation $x \sqsubseteq_A y$ is not transitive/pointwise on all A. So, isolate a class of A's (reflective subuniverse) for which it is.

Goal

- ☑ reflexive
- ☑ transitive
- ☑ monotone: $a \sqsubseteq_A a'$ implies $f(a) \sqsubseteq_B f(a')$
- ☑ pointwise on functions
- ☑ extensionally discrete: $\bigcirc(x \sqsubseteq_A y)$ implies $\bigcirc(x = y)$

A Non-Trivial Model

Axioms

1. Interval $\mathbb{I}$,
2. discrete type $\mathbb{N}$,
3. proposition ext,
4. and $\bigcirc(\mathbb{I} \cong \mathbf{1})$.

A Non-Trivial Model

Axioms

1. Interval $\mathbb{I}$,
2. discrete type $\mathbb{N}$,
3. proposition ext,
4. and $\bigcirc(\mathbb{I} \cong \mathbf{1})$.

Example (Augmented Simplicial Sets)

Simplicial sets, but where initial object $[-1]$ is added to the simplex category.

$$\mathbb{I} := \mathbf{y}[1]$$

$$\mathsf{ext} := \mathbf{y}[-1]$$

Cost Model as a Directed QIT

Example (Cost Model ω as a QIT)

data ω **where**

$\mathsf{zero} : \omega$

$\mathsf{suc} : \omega \to \omega$

$\_ : (n : \omega) \to n \sqsubseteq_\omega \mathsf{suc}\ n$

Cost Model as a Directed QIT

Example (Cost Model ω as a QIT)

data ω **where**

zero : ω

suc : $\omega \to \omega$

$\_ : (n : \omega) \to n \sqsubseteq_\omega \mathsf{suc}\ n$

Theorem ($\mathbb{C} :\equiv \omega$ is a Valid Cost Model)

$$\bigcirc(\omega \cong \mathbf{1})$$

Conclusion

Future Work

Amortized Analysis [Grodin and Harper, 2023]

Amortized upper bounds (using coinduction)?

Abstraction

Abstract data types and cost signatures? Separating cost from correctness?

Parallelism and Effects

Effects in parallel (commutative)? Non-algebraic effects (*e.g.*, unbounded recursion)?

Advanced Probabilistic Analysis

Expected/with-high-probability cost analysis?

Conclusion

Contribution

calf does synthetic cost analysis at $\mathsf{F}(-)$ types. **decalf** adds:

- $\lightning$ *support for effects and higher-order programs and*
- $\leq$ *program inequality for inexact bounds*
- φ *harmonious with extensional reasoning.*

Conclusion

Contribution

calf does synthetic cost analysis at $\mathsf{F}(-)$ types. **decalf** adds:

- $\lightning$ *support for effects and higher-order programs and*
- $\leq$ *program inequality for inexact bounds*
- φ *harmonious with extensional reasoning.*

Justification

1. Topos-theoretically via augmented simplicial sets, and
2. practically via full-scale examples embedded in Agda.

References i

Danielsson, N. A. (2008).
Lightweight semiformal time complexity analysis for purely functional data structures.
In *Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '08, page 133–144, New York, NY, USA. Association for Computing Machinery.

Grodin, H. and Harper, R. (2023).
Amortized Analysis via Coinduction.
In Baldan, P. and de Paiva, V., editors, *10th Conference on Algebra and Coalgebra in Computer Science (CALCO 2023)*, volume 270 of *Leibniz International*

Proceedings in Informatics (LIPIcs), pages 23:1–23:6, Dagstuhl, Germany. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

Hoare, C. A. R. (1961).
Algorithm 64: Quicksort.
Communications of the ACM, 4(7):321.

Hoare, C. A. R. (1962).
Quicksort.
The Computer Journal, 5(1):10–16.

Hyland, J. M. E. (1991).
First steps in synthetic domain theory.
In Carboni, A., Pedicchio, M. C., and Rosolini, G., editors, *Category Theory*, pages 131–156, Berlin, Heidelberg. Springer Berlin Heidelberg.

Levy, P. B. (2003).
Adjunction models for call-by-push-value with stacks.
Electronic Notes in Theoretical Computer Science, 69:248–271.
CTCS'02, Category Theory and Computer Science.

References iv

Licata, D. R. and Harper, R. (2011).
2-dimensional directed type theory.
Electronic Notes in Theoretical Computer Science, 276:263–289.
Twenty-seventh Conference on the Mathematical Foundations of Programming Semantics (MFPS XXVII).

Niu, Y., Sterling, J., Grodin, H., and Harper, R. (2022).
A cost-aware logical framework.
Proceedings of the ACM on Programming Languages, 6(POPL).

Phoa, W. (1991).
Domain Theory in Realizability Toposes.
PhD thesis, University of Edinburgh.

References v

Plotkin, G. D. and Power, J. (2002).
Notions of computation determine monads.
In *Proceedings of the 5th International Conference on Foundations of Software Science and Computation Structures*, pages 342–356, Berlin, Heidelberg. Springer-Verlag.

Riehl, E. and Shulman, M. (2017).
A type theory for synthetic ∞-categories.
Higher Structures, 1:147–224.

Sterling, J. (2021).
First Steps in Synthetic Tait Computability: The Objective Metatheory of Cubical Type Theory.
PhD thesis, Carnegie Mellon University.
Version 1.1, revised May 2022.

Sterling, J. and Harper, R. (2021).
Logical relations as types: Proof-relevant parametricity for program modules.
Journal of the ACM, 68(6).