Consistency of a Programming Logic for a Version of PCF Using Domain Theory

Andrés Sicard-Ramírez

Universidad EAFIT

Logic and Computation Seminar
Universidad EAFIT
5 April, 3 May 2013

A Core Functional Programming Language

Plotkin's PCF language

Theoretical Computer Science 5 (1977) 223–255.

LCF CONSIDERED AS A PROGRAMMING LANGUAGE

G.D. PLOTKIN
Department of Artificial Intelligence, University of Edinburgh, Hope Park Square, Meadow Lane, Edinburgh EH8 9NW, Scotland

Communicated by Robin Milner
Received July 1975

PCF Features [Plotkin 1977]

- Typed λ-calculus

Starting with a collection $\mathscr{L}$ of constants, each having a fixed type, and denumerably many variables α_i^σ $(i \geqslant 0)$ of each type, the *$\mathscr{L}$-terms* are given by the rules:

(1) Every variable α_i^σ is an $\mathscr{L}$-term of type σ.

(2) Every constant of type σ is an $\mathscr{L}$-term of type σ.

(3) If M and N are $\mathscr{L}$-terms of types $(\sigma \to \tau)$ and σ respectively then (MN) is an $\mathscr{L}$-term of type τ.

(4) If M is an $\mathscr{L}$-term of type τ then $(\lambda\alpha_i^\sigma M)$ is one of type $(\sigma \to \tau)$ $(i \geqslant 0)$.

PCF Features [Plotkin 1977]

- Basic data types: Natural numbers and Booleans

All languages, $\mathscr{L}$, considered include $\mathscr{L}_0$, the set of *standard* constants. These, together with their types, are:

$$tt : o,$$

$$ff : o,$$

$$\supset_\iota : (o, \iota, \iota, \iota),$$

$$\supset_o : (o, o, o, o),$$

$$Y_\sigma : ((\sigma \to \sigma) \to \sigma) \text{ (one for each } \sigma).$$

Generally we will be interested in a language $\mathscr{L}_A$ for arithmetic which also has:

$$k_n : \iota \text{ (one for each integer } n \geqslant 0),$$

$$(+1) : (\iota \to \iota),$$

$$(-1) : (\iota \to \iota),$$

$$Z : (\iota \to o).$$

A Programming Logic

Logical Theory of Constructions (LTC)

[Bove, Dybjer and Sicard-Ramírez 2009]

LTC = type-free version of PCF

(terms, conversion and discrimination rules)

+ first-order logic

+ inductive predicates (not considered in this talk)

LTC-Terms

Terms

$t ::= x$	variable
$\mid t \cdot t$	application
$\mid \lambda x.t$	λ-abstraction
$\mid \mathsf{fix}\ x.t$	fixed-point operator
$\mid \mathsf{true} \mid \mathsf{false} \mid \mathsf{if}$	Boolean constants
$\mid 0 \mid \mathsf{succ} \mid \mathsf{pred} \mid \mathsf{iszero}$	natural number constants

Convention

The binary application function symbol $\cdot$ is left-associative.

LTC-Formulae

Formulae

$A ::= \top \mid \bot$	truth, falsehood
$\mid A \Rightarrow A \mid A \wedge A \mid A \vee A$	binary logical connectives
$\mid \forall x.A \mid \exists x.A$	quantifiers
$\mid t = t$	equality
$\mid P(t, \ldots, t)$	predicate

Abbreviations

$$\neg A \stackrel{\text{def}}{=} A \Rightarrow \bot,$$

$$t \neq t' \stackrel{\text{def}}{=} \neg(t = t').$$

Conversion and Discrimination Rules of LTC

Conversion rules

$$\begin{aligned}
\forall t\ t'.\ \mathsf{if} \cdot \mathsf{true} \cdot t \cdot t' &= t, \\
\forall t\ t'.\ \mathsf{if} \cdot \mathsf{false} \cdot t \cdot t' &= t', \\
\mathsf{pred} \cdot 0 &= 0, \\
\forall t.\ \mathsf{pred} \cdot (\mathsf{succ} \cdot t) &= t, \\
\mathsf{iszero} \cdot 0 &= \mathsf{true}, \\
\forall t.\ \mathsf{iszero} \cdot (\mathsf{succ} \cdot t) &= \mathsf{false}, \\
\forall t\ t'.\ (\lambda x.t) \cdot t' &= t[x := t'], \\
\forall t.\ \mathsf{fix}\ x.t &= t[x := \mathsf{fix}\ x.t],
\end{aligned}$$

where $t[x := t']$ is the capture-free substitution of x for t' in t.

Conversion and Discrimination Rules of LTC

Discrimination rules

$$\mathsf{true} \neq \mathsf{false},$$
$$\forall t.\ 0 \neq \mathsf{succ} \cdot t.$$

LTC Consistency

How we know that LTC is a consistent theory?

LTC Consistency

How we know that LTC is a consistent theory?

Standard answer: To build a model for LTC
[Chang and Keisler 1992, theorem 1.3.21.]

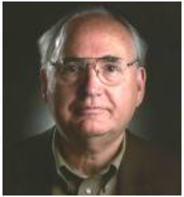
LTC Consistency

How we know that LTC is a consistent theory?

Standard answer: To build a model for LTC
[Chang and Keisler 1992, theorem 1.3.21.]
$\Rightarrow$ domain model for LTC

Introduction to Domain Theory

Motivation: Does λ-calculus have models?



"Historically my first model for the λ-calculus was discovered in 1969 and details were provided in Scott (1972) (written in 1971)." [Scott 1980, p. 226.]

Non-standard definitions

pre-domain, domain, complete partial order (cpo), ω-cpo, bottomless ω-cpo, Scott's domain, ...

Convention

domain $\equiv$ ω-complete partial order

Some bibliographic references

- Winskel, G. [1993] [1994]. The Formal Semantics of Programming Languages. An Introduction. Foundations of Computing Series. Second printing. MIT Press.

Some bibliographic references

- Winskel, G. [1993] [1994]. The Formal Semantics of Programming Languages. An Introduction. Foundations of Computing Series. Second printing. MIT Press.
- Mitchell, J. C. [1996]. Foundations for Programming Languages. MIT Press.

Some bibliographic references

- Winskel, G. [1993] [1994]. The Formal Semantics of Programming Languages. An Introduction. Foundations of Computing Series. Second printing. MIT Press.
- Mitchell, J. C. [1996]. Foundations for Programming Languages. MIT Press.
- Streicher, T. [2006]. Domain-Theoretic Foundations of Functional Programming. World Scientific Publishing Co. Pte. Ltd.

Some bibliographic references

- Winskel, G. [1993] [1994]. The Formal Semantics of Programming Languages. An Introduction. Foundations of Computing Series. Second printing. MIT Press.
- Mitchell, J. C. [1996]. Foundations for Programming Languages. MIT Press.
- Streicher, T. [2006]. Domain-Theoretic Foundations of Functional Programming. World Scientific Publishing Co. Pte. Ltd.
- Plotkin, G. [1992]. Post-graduate Lecture Notes in Advance Domain Theory (Incorporating the "Pisa Notes"). Electronic edition prepared by Yugo Kashiwagi and Hidetaka Kondoh. URL: http://homepages.inf.ed.ac.uk/gdp/ [visited on 29/07/2014].

Partially Ordered Sets

Definition (Partially ordered set)

A partially ordered set (poset) $(D, \sqsubseteq)$ is a set D on which the binary relation $\sqsubseteq$ satisfies the following properties:

$$\forall x.\ x \sqsubseteq x \qquad \text{(reflexive)}$$

$$\forall x\ y\ z.\ x \sqsubseteq y \wedge y \sqsubseteq z \Rightarrow x \sqsubseteq z \qquad \text{(transitive)}$$

$$\forall x\ y.\ x \sqsubseteq y \wedge y \sqsubseteq x \Rightarrow x = y \qquad \text{(antisymmetry)}$$

Partially Ordered Sets

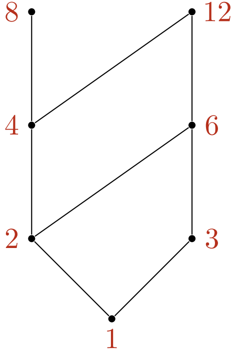
Examples

- $(\mathbb{Z}, \leq)$ is a poset.
- Let $a, b \in \mathbb{Z}$ with $a \neq 0$. The divisibility relation is defined by $a \mid b \stackrel{\text{def}}{=} \exists c\, (ac = b)$. Then $(\mathbb{Z}^+, |)$ is a poset.
- $(P(A), \subseteq)$ is a poset.

Partially Ordered Sets

Example

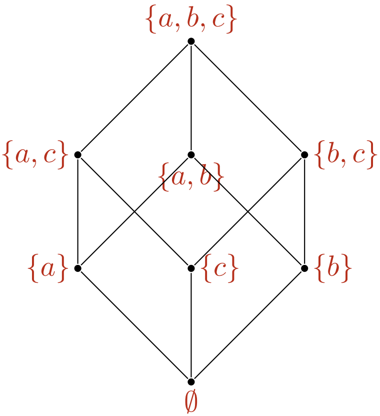
Hasse diagram for the poset $(\{1, 2, 3, 4, 6, 8, 12\}, |)$.



Partially Ordered Sets

Example

Hasse diagram for the poset $(\{a, b, c\}, \subseteq)$.



Monotone Functions

Definition (Monotone function)

Let $(D, \sqsubseteq)$ and $(D', \sqsubseteq')$ be two posets. A function $f : D \to D'$ is monotone if

$$\forall x\ y.\ x \sqsubseteq y \Rightarrow f(x) \sqsubseteq' f(y).$$

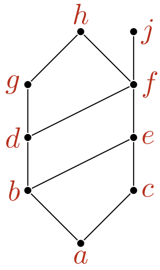
Notable Elements

Definition (Upper bound)

Let $(D, \sqsubseteq)$ be a poset and let $A \subseteq D$. Let $u \in D$ be an element such that $a \sqsubseteq u$ for all elements $a \in A$, then u is an upper bound of A.

Examples

- $A = \{a, b, c\}$
  Upper bounds: $\{e, f, j, h\}$
- $A = \{j, h\}$
  No upper bounds.
- $A = \{a, c, d, f\}$
  Upper bounds: $\{f, h, j\}$



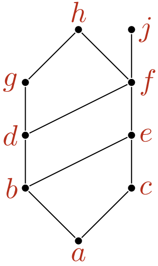
Notable Elements

Definition (Supremum or least upper bound)

An element x is the supremum or the least upper bound of the subset A, denoted by $\bigcup A$, if x is an upper bound that is less than every other upper bound of A.

Example

- $A = \{b, d, g\}$
  Upper bounds: $\{g, h\}$
  $\bigcup A = g$



ω-Complete Partial Orders

Definition (ω-chain)

Let $\mathbf{D} = (D, \sqsubseteq)$ be a poset. A ω-chain of $\mathbf{D}$ is an increasing chain $d_0 \sqsubseteq d_1 \sqsubseteq \cdots \sqsubseteq d_n \sqsubseteq \cdots$, where $d_i \in D$.

ω-Complete Partial Orders

Definition (ω-complete partial order)

Let $\mathbf{D} = (D, \sqsubseteq)$ be a poset. The poset $\mathbf{D}$ is a ω-complete partial order (ω-cpo) if [Plotkin 1992]

1. There is a least element $\bot \in D$, that is, $\forall x.\ \bot \sqsubseteq x$. The element $\bot$ is called *bottom*.
2. For every ω-chain $d_0 \sqsubseteq d_1 \sqsubseteq \cdots \sqsubseteq d_n \sqsubseteq \cdots$, the least upper bound $\bigsqcup_{n\in\omega} d_n \in D$ exists.

ω-Complete Partial Orders

Definition (Lifted set)

Let A be a set. The symbol $A_\perp$ denotes the ω-cpo whose elements $A \cup \{\perp\}$ are ordered by $x \sqsubseteq y$, if and only if, $x = \perp$ or $x = y$ [Mitchell 1996]. The ω-cpo $A_\perp$ is called A lifted.

ω-Complete Partial Orders

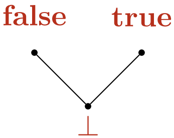
Definition (Lifted set)

Let A be a set. The symbol $A_\perp$ denotes the ω-cpo whose elements $A \cup \{\perp\}$ are ordered by $x \sqsubseteq y$, if and only if, $x = \perp$ or $x = y$ [Mitchell 1996]. The ω-cpo $A_\perp$ is called A lifted.

Examples

The lifted unit type and the lifted Booleans $\mathbf{B}_\perp$.


```
data () = ()
```

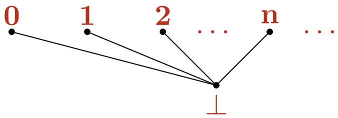



```
data Bool = True | False
```

ω-Complete Partial Orders

Example

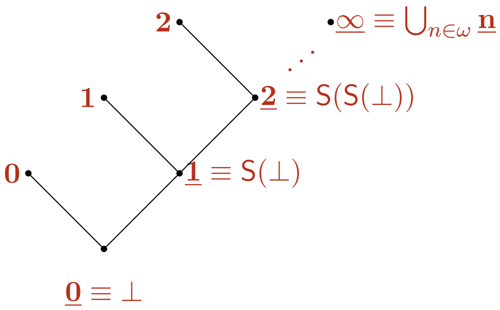
The lifted natural numbers $\mathbf{N}_\perp$.



ω-Complete Partial Orders

Example

The ω-cpo $\mathbf{LN}$ of lazy natural numbers arises from a non-strict successor function, that is, $\mathsf{S}(\perp) \neq \perp$ [Escardó 1993]




```
data Nat = Z
         | S Nat
```

ω-Complete Partial Orders

Definition (Continuous function)

Let $(D, \sqsubseteq)$ and $(D', \sqsubseteq')$ be two ω-cpos. A function $f : D \to D'$ is continuous if [Plotkin 1992]

1. The function is monotone.
2. The function preserves the least upper bounds of the ω-chains, that is,

$$\bigsqcup_{n \in \omega} f(d_n) = f(\bigsqcup_{n \in \omega} d_n),$$

   for all ω-chains $d_0 \sqsubseteq d_1 \sqsubseteq \cdots \sqsubseteq d_n \sqsubseteq \cdots$.

ω-Complete Partial Orders

Definition (Function space of continuous functions)

Let $(D, \sqsubseteq)$ and $(D', \sqsubseteq')$ be two ω-cpos. The function space of continuous functions is the set [Winskel 1994]

$$[D \to D'] = \{f : D \to D' \mid f \text{ is continous } \}.$$

ω-Complete Partial Orders

Definition (Function space of continuous functions)

Let $(D, \sqsubseteq)$ and $(D', \sqsubseteq')$ be two ω-cpos. The function space of continuous functions is the set [Winskel 1994]

$$[D \to D'] = \{f : D \to D' \mid f \text{ is continous } \}.$$

Theorem

The function space $[D \to D']$ is an ω-cpo.

- $[D \to D']$ can be partially ordered point-wise by

$$f \sqsubseteq g \Leftrightarrow \forall d \in D.\ f(d) \sqsubseteq' g(d).$$

- The bottom element is $\lambda x.\bot_{D'}$.

ω-Complete Partial Orders

Definition

Let $f : D \to D$ be a function, then

$$f^0(d) = d,$$
$$f^{n+1}(d) = f(f^n(d)).$$

Theorem (The Fixed-Point Theorem)

Let $(D, \sqsubseteq)$ be an ω-cpo. Given $f \in [D \to D]$, then

$$\mathrm{Fix}(f) = \bigcup_{n \in \omega} f^n(\bot),$$

is the least fixed-point of f [Winskel 1994], that is,

$$\forall d.\ f(d) \sqsubseteq d \Rightarrow \mathrm{Fix}(f) \sqsubseteq d,$$
$$f(\mathrm{Fix}(f)) = \mathrm{Fix}(f).$$

ω-Complete Partial Orders

Definition (Coalesced sum)

Let $\mathbf{D}_1 = (D_1, \sqsubseteq_1), \ldots, \mathbf{D}_n = (D_n, \sqsubseteq_n)$ be ω-cpos. The coalesced sum, that is, disjoint union with bottom elements identified $\mathbf{D}_1 \oplus \cdots \oplus \mathbf{D}_n$ is the ω-cpo [Plotkin 1992]

$$\left(\bigcup_{i \leq n} \{(i, d) \mid d \in D_i \wedge d \neq \perp\} \right) \cup \perp$$

with the order

$$x \sqsubseteq y \Leftrightarrow x = \perp \text{ or }$$
$$\exists i \leq n. \exists d, d' \in D_i.\ d \sqsubseteq_i d' \wedge x = (i, d) \wedge y = (i, d').$$

ω-Complete Partial Orders

Associated with the coalesced sum are the injection functions

$$in_i : \mathbf{D}_i \to \mathbf{D}_1 \oplus \cdots \oplus \mathbf{D}_n$$
$$in_i(d) = \begin{cases} \bot & \text{if } d = \bot, \\ (i, d) & \text{otherwise}. \end{cases}$$

Domain Model for LTC

Terms: The term language of LTC

From domain theory it is known that a domain model for Terms, where self-application is allowed and where the terms will have values in the Booleans or the lazy natural numbers is a solution to the recursive domain equation [Plotkin 1992]

$$\mathbf{D} \cong \mathbf{B}_{\perp} \oplus \mathbf{LN} \oplus (\mathbf{D} \to \mathbf{D})_{\perp}.$$

Domain Model for LTC

Notation

Let $\mathbf{D}$ be a domain and let ρ be a valuation on $\mathbf{D}$ (a function from the set of variables to $\mathbf{D}$).

- $\rho(x \mapsto \mathbf{d})$: the valuation which maps x to $\mathbf{d}$ and otherwise acts like ρ.
- $\lambda\mathbf{x}.\mathbf{e}$: λ-abstraction on $\mathbf{D}$.

Domain Model for LTC

Convention

$\mathbf{D}$: A solution to the recursive domain equation for LTC.

From terms to functions and viceverse

The domain $\mathbf{D}$ comes equipped with the continuous functions [Barendregt 2004]

$$\mathrm{F} : \mathbf{D} \to [\mathbf{D} \to \mathbf{D}],$$
$$\mathrm{G} : [\mathbf{D} \to \mathbf{D}] \to \mathbf{D}.$$

Domain Model for LTC

Interpretation function

$[\![\ ]\!]_\rho : \text{Terms} \to \mathbf{D}$: (Based on Pitts [1994])

$$
\begin{aligned}
[\![x]\!]_\rho &= \rho(x), \\
[\![\lambda x.t]\!]_\rho &= \mathrm{G}(\lambda \mathbf{d}.[\![t]\!]_{\rho(x \mapsto \mathbf{d})}), \\
[\![t \cdot t']\!]_\rho &= \begin{cases} \mathbf{f}([\![t']\!]_\rho) & \text{if } [\![t]\!]_\rho = \mathrm{G}(\mathbf{f}), \\ \bot & \text{otherwise}, \end{cases} \\
[\![\text{fix } x.t]\!]_\rho &= \mathrm{Fix}(\lambda \mathbf{d}.[\![t]\!]_{\rho(x \mapsto \mathbf{d})}), \\
[\![\text{true}]\!]_\rho &= \mathbf{true},
\end{aligned}
$$

$$
\begin{aligned}
[\![\text{false}]\!]_\rho &= \mathbf{false}, \\
[\![\text{if}]\!]_\rho &= \mathrm{G}(\mathbf{if}), \\
[\![0]\!]_\rho &= \mathbf{0}, \\
[\![\text{succ}]\!]_\rho &= \mathrm{G}(\mathbf{succ}), \\
[\![\text{pred}]\!]_\rho &= \mathrm{G}(\mathbf{pred}), \\
[\![\text{iszero}]\!]_\rho &= \mathrm{G}(\mathbf{iszero}),
\end{aligned}
$$

where

Domain Model for LTC

we omit the use of the injection functions in_i, and the continuous functions $\mathbf{if}$, $\mathbf{succ}$, $\mathbf{pred}$ and $\mathbf{iszero}$ from $\mathbf{D}$ to $\mathbf{D}$ are defined by

$$\mathbf{if}(d) = \begin{cases} \lambda \mathbf{xy.x} & \text{if } d = \mathbf{true}, \\ \lambda \mathbf{xy.y} & \text{if } d = \mathbf{false}, \\ \perp & \text{otherwise}, \end{cases}$$

$$\mathbf{succ}(d) = \begin{cases} \mathbf{n+1} & \text{if } d = \mathbf{n} \in \mathbf{LN}, \\ \underline{\mathbf{n+1}} & \text{if } d = \underline{\mathbf{n}} \in \mathbf{LN}, \\ \perp & \text{otherwise}, \end{cases}$$

Domain Model for LTC

$$\mathbf{pred}(d) = \begin{cases} \mathbf{0} & \text{if } d = \mathbf{0}, \\ d' & \text{if } d = \mathbf{succ}(d'), \\ \bot & \text{otherwise}, \end{cases}$$

$$\mathbf{iszero}(d) = \begin{cases} \mathbf{true} & \text{if } d = \mathbf{0}, \\ \mathbf{false} & \text{if } d = \mathbf{succ}(d'), \\ \bot & \text{otherwise}. \end{cases}$$

Domain Model for LTC

If the LTC equality is interpreted as the equality in $\mathbf{D}$, it is possible verify that the conversion and discrimination rules of LTC are satisfied in $\mathbf{D}$.

Bonus Slides

Monotone Functions

Example (Counter-example of monotone function)

$$\mathbf{halt} : \mathbf{N}_\perp \to \mathbf{B}_\perp$$

$$\mathbf{halt}(n) = \begin{cases} \mathbf{true} & \text{if } n \neq \perp, \\ \mathbf{false} & \text{if } n = \perp. \end{cases}$$

Let $n \in \mathbf{N}_\perp$. Since $\perp \sqsubseteq n$, and not necessarily $\mathbf{halt}(\perp) \sqsubseteq \mathbf{halt}(n)$, that is, $\mathbf{false} \not\sqsubseteq \mathbf{true}$, the $\mathbf{halt}$ function is non-monotone [Schmidt 1986].

Continuous Functions

Example (Monotone but non-continuous function[1])

$$f : [\mathrm{Bool}] \to \mathrm{Bool}$$

$$f(xs) = \begin{cases} \bot & \text{if } xs \text{ is finite}, \\ \mathrm{False} & \text{if } xs = [\mathrm{False}, \mathrm{False}, \ldots], \\ \mathrm{True} & \text{otherwise}. \end{cases}$$

Given $d_0 = []$, $d_1 = [\mathrm{False}]$, $d_2 = [\mathrm{False}, \mathrm{False}]$, $\ldots$, we have

$$\bigsqcup_{n \in \omega} f(d_n) = \bot \neq \mathrm{False} = f([\mathrm{False}, \mathrm{False}, \ldots]) = f\left(\bigsqcup_{n \in \omega} d_n\right),$$

that is, the function f is non-continuous.

[1] http://www.reddit.com/r/types/comments/1ahfh7/intuition_behind_continuity_in_winskels/.

Complete Partial Orders

Definition (Directed set)

Let $\mathbf{D} = (D, \sqsubseteq)$ be a poset. A subset $X \subseteq D$ is directed if $X \neq \emptyset$ and

$$\forall x\ y \in X. \exists z \in X.\ x \sqsubseteq z \wedge y \sqsubseteq z.$$

Definition (Complete partial order)

Let $\mathbf{D} = (D, \sqsubseteq)$ be a poset. The poset $\mathbf{D}$ is a complete partial order (cpo) if [Barendregt 2004]

1. There is a least element $\bot \in D$, that is, $\forall x.\ \bot \sqsubseteq x$. The element $\bot$ is called *bottom*.
2. For every directed $X \subseteq D$, the least upper bound $\bigsqcup X \in D$ exists.

Complete Partial Orders

Note

The Scott domains are built from complete partial orders. See, for example, Gunter and Scott [1990].

References

Barendregt, H. P. [1984] (2004). The Lambda Calculus. Its Syntax and Semantics. Revised edition, 6th impression. Vol. 103. Studies in Logic and the Foundations of Mathematics. Elsevier (cit. on pp. 40, 48).

Bove, A., Dybjer, P. and Sicard-Ramírez, A. (2009). Embedding a Logical Theory of Constructions in Agda. In: Proceedings of the 3rd Workshop on Programming Languages Meets Program Verification (PLPV 2009), pp. 59–66 (cit. on p. 5).

Chang, C. C. and Keisler, H. J. [1973] (1992). Model Theory. 3rd ed. Vol. 73. Studies in Logic and the Foundations of Mathematics. 3rd impression. North-Holland (cit. on pp. 10–12).

Escardó, M. H. (1993). On Lazy Natural Numbers with Applications to Computability Theory and Functional Programming. SIGACT News 24.1, pp. 61–67. DOI: 10.1145/152992.153008 (cit. on p. 31).

Gunter, C. A. and Scott, D. S. (1990). Semantics Domains. In: Handbook of Theoretical Computer Science. Ed. by van Leeuwen, J. Vol. B. Formal Models and Semantics. MIT Press. Chap. 12 (cit. on p. 49).

References

Mitchell, J. C. (1996). Foundations for Programming Languages. MIT Press (cit. on pp. 15–18, 28, 29).

Pitts, A. M. (1994). Computational Adequacy via 'Mixed' Inductive Definitions. In: Mathematical Foundations of Programming Semantics. Ed. by Brookes, S., Main, M., Melton, A., Mislove, M. and Schmidt, D. Vol. 802. Lecture Notes in Computer Science. Springer, pp. 72–82. DOI: 10.1007/3-540-58027-1_3 (cit. on p. 41).

Plotkin, G. D. (1977). LCF Considered as a Programming Language. Theoretical Computer Science 5.3, pp. 223–255. DOI: 10.1016/0304-3975(77)90044-5 (cit. on pp. 3, 4).

Plotkin, G. (1992). Post-graduate Lecture Notes in Advance Domain Theory (Incorporating the "Pisa Notes"). Electronic edition prepared by Yugo Kashiwagi and Hidetaka Kondoh. URL: http://homepages.inf.ed.ac.uk/gdp/ (visited on 29/07/2014) (cit. on pp. 15–18, 27, 32, 36, 38).

Schmidt, D. A. (1986). Denotational Semantics. A Methodology for Language Development. Allyn and Bacon (cit. on p. 46).

References

Scott, D. (1980). Lambda Calculus: Some Models, Some Philosophy. In: The Kleene Symposium. Ed. by Barwise, J., Keisler, H. J. and Kunen, K. Vol. 101. Studies in Logic and the Foundations of Mathematics. North-Holland Publishing Company, pp. 223–265 (cit. on p. 13).

Streicher, T. (2006). Domain-Theoretic Foundations of Functional Programming. World Scientific Publishing Co. Pte. Ltd. (cit. on pp. 15–18).

Winskel, G. [1993] (1994). The Formal Semantics of Programming Languages. An Introduction. Foundations of Computing Series. Second printing. MIT Press (cit. on pp. 15–18, 33–35).