OXFORD UNIVERSITY COMPUTING LABORATORY
PROGRAMMING RESEARCH GROUP

Lambda Calculus

C.-H. L. Ong

Aim

Recursive functions are representable as lambda terms, and definability in the calculus may be regarded as a definition of computability. This forms part of the standard foundations of computer science. Lambda calculus is the commonly accepted basis of functional programming languages; and it is folklore that the calculus *is* the prototypical functional language in purified form. The course investigates the syntax and semantics of lambda calculus both as a theory of functions from a foundational point of view, and as a minimal programming language.

Synopsis

Formal theory, fixed point theorems, combinatory logic: combinatory completeness, translations between lambda calculus and combinatory logic; reduction: Church-Rosser theorem; Böhm's theorem and applications; basic recursion theory; lambda calculi considered as programming languages; simple type theory and PCF: correspondence between operational and denotational semantics; current developments.

Relationship with other courses

Basic knowledge of logic and computability in paper B1 is assumed.

Selected references

- H. Barendregt. *The Lambda Calculus.* North-Holland, revised edition, 1984.
- J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types.* Cambridge University Press, 1989. Cambridge Tracts in Theoretical Computer Science 7.
- C. A. Gunter. *Semantics of Programming Languages: Structures and Techniques.* MIT Press, 1992.
- G. D. Plotkin. LCF considered as a programming language. *Theoretical Computer Science* 5:223–255, 1975.

[*Please send any correction to* `lo@comlab.ox.ac.uk`.]

Since s and t are not $\beta\eta$-equivalent, for some i, s_i and t_i are not $\beta\eta$-equivalent. Take $\pi_i \equiv \lambda x_1 \cdots x_n.x_i$.

$$\begin{aligned} \mathbf{B}_{\pi_i,z} \circ B(s^*) &= s_i^\dagger \\ \mathbf{B}_{\pi_i,z} \circ B(t^*) &= t_i^\dagger \end{aligned}$$

(Note that z does not occur free in $s_i^\dagger$ nor $t_i^\dagger$.) Clearly $\mathsf{size}(s_i) + \mathsf{size}(t_i) < \mathsf{size}(s) + \mathsf{size}(t)$. Hence, by the induction hypothesis, say B' is the required Böhm transformation for $s_i^\dagger$ and $t_i^\dagger$. The Böhm transformation required is just $B' \circ \mathbf{B}_{\pi_i,z} \circ B$.

Case (ii): $x \in \{x_1, \cdots, x_k\}$, say, $x = x_1$, and $y \notin \{x_1, \cdots, x_k\}$. Then, for every $n_1 > p$,

$$\begin{aligned} s^* &= \alpha_{n_1} s_1^* \cdots s_p^* = \lambda z_{p+1} \cdots z_{n_1} z.z s_1^* \cdots s_p^* z_{p+1} \cdots z_{n_1} \\ t^* &= y t_1^* \cdots t_q^*. \end{aligned}$$

Take $B = \mathbf{B}_z \circ \mathbf{B}_{z_{n_1}} \circ \cdots \circ \mathbf{B}_{z_{p+1}}$,

$$\begin{aligned} B(s^*) &= z s_1^* \cdots s_p^* z_{p+1} \cdots z_{n_1}, \\ B(t^*) &= y t_1^* \cdots t_q^* z_{p+1} \cdots z_{n_1} z. \end{aligned}$$

Since $y \neq z$, result follows from Lemma 4.2.4(1).

Case (iii): $x, y \in \{x_1, \cdots, x_k\}$.

Suppose $x = x_1$ and $y = x_2$ are distinct:

$$\begin{aligned} s^* &= \alpha_{n_1} s_1^* \cdots s_p^* = \lambda z_{p+1} \cdots z_{n_1} z.z s_1^* \cdots s_p^* z_{p+1} \cdots z_{n_1} \\ t^* &= \alpha_{n_2} t_1^* \cdots t_q^* = \lambda z_{q+1} \cdots z_{n_2} z.z t_1^* \cdots t_q^* z_{q+1} \cdots z_{n_2} \end{aligned}$$

taking $n_1 > p$, $n_2 > q$. Since $n_1 \neq n_2$ result follows from Lemma 4.2.4(2).

Suppose $x = y = x_1$, take $n_1 > p, q$:

$$\begin{aligned} s^* &= \alpha_{n_1} s_1^* \cdots s_p^* = \lambda z_{p+1} \cdots z_{n_1} z.z s_1^* \cdots s_p^* z_{p+1} \cdots z_{n_1} \\ t^* &= \alpha_{n_1} t_1^* \cdots t_q^* = \lambda z_{q+1} \cdots z_{n_1} z.z t_1^* \cdots t_q^* z_{q+1} \cdots z_{n_1}. \end{aligned}$$

If $p \neq q$ then by Lemma 4.2.4(2), $n_1 - p \neq n_1 - q$, result then follows. If $p = q$ then since s and t are not $\beta\eta$-equivalent, for some i, s_i and t_i are not $\beta\eta$-equivalent. Let $\pi_1 \equiv \lambda x_1 \cdots x_{n_1}.x_i$ and $B \stackrel{\text{def}}{=} \mathbf{B}_z \circ \mathbf{B}_{z_{n_1}} \circ \cdots \circ \mathbf{B}_{z_{n_{p+1}}}$. Then

$$\begin{aligned} \mathbf{B}_{\pi_1,z} \circ B(s^*) &= s_i^* \\ \mathbf{B}_{\pi_1,z} \circ B(t^*) &= t_i^*. \end{aligned}$$

Similar argument as before concludes the proof. □

Böhm's Theorem is an immediate consequence of Theorem 4.2.5. For if s is any closed term and B a Böhm transformation, then by Lemma 4.2.2 we have $Bs = su_1 \cdots u_k$ where $u_1, \cdots, u_k$ depend only on B. By applying Theorem 4.2.5 we therefore obtain $su_1 \cdots u_k = \mathbf{f}$ and $tu_1 \cdots u_k = \mathbf{t}$. (We may suppose that $\vec{u}$ are closed terms.)

5 Call-by-name and call-by-value lambda calculi

According to the so-called ***function paradigm*** of computation, the goal of every computation is to determine its *value*. Thus to compute is to *evaluate*. A (by now) standard way to implement evaluation is by a process of *reduction*. In this section we shall investigate a couple of important ideas that have arisen in semantics of functional computation in recent years. We take pure, untyped λ-calculus equipped with call-by-name (CBN) and call-by-value (CBV) reduction strategies as minimal (and prototypical) functional languages; and consider two operational or behavioural preorders over terms, namely, *applicative simulation* and *observational (or contextual) preorder*. We prove that they conincide in both CBN and CBV λ-calculi. In other words both languages satisfy the *context lemma*.

5.1 Motivations

The commonly accepted basis for functional programming is the λ-calculus; and it is folklore that the λ-calculus *is* the prototypical functional language in purified form. But what is the λ-calculus? The syntax is simple and classical; variables, abstraction and application in the pure calculus, with applied calculi obtained by adding constants. The further elaboration of the theory, covering conversion, reduction, theories and models, is laid out in Barendregt's already classical treatise [Bar84]. It is instructive to recall the following crux, which occurs rather early in that work (p. 39):

Meaning of λ-terms: first attempt

- The meaning of a λ-term is its normal form (if it exists).
- All terms without normal forms are identified.

This proposal incorporates such a simple and natural interpretation of the λ-calculus as a programming language, that if it worked there would surely be no doubt that it was the right one. However, it gives rise to an inconsistent theory!

Second attempt: sensible theory

- The meaning of λ-terms is based on ***head normal forms*** via the notion of ***Böhm tree***.
- All *unsolvable* terms (no head normal form) are identified.

This second attempt forms the central theme of Barendregt's book, and gives rise to a very beautiful and successful theory (henceforth referred to as the "standard theory"), as that work shows.

This, then, is the commonly accepted foundation for functional programming; more precisely, for the *lazy* functional languages [FW76, HM76], which represent the mainstream of current functional programming practice. Examples: Miranda [Tur85], LML [Aug84], Orwell [Wad85], Haskell, and Gofer. But do these languages as defined and implemented actually evaluate terms to head normal form? To the best of our knowledge, *not a single one of them does so*. Instead, they evaluate to *weak head normal form* i.e. they do not evaluate under abstractions (see [PJ87] for a comprehensive survey of the pragmatics of functional programming languages). E.g., $\lambda x.(\lambda y.y)s$ is in weak head normal form, but not in head normal form, since it contains the head redex $(\lambda y.y)s$.

So we have a fundamental *mismatch* between theory and practice. Since current practice is well-motivated by efficiency considerations and is unlikely to be abandoned readily, it makes sense to see if a good modified theory can be developed for it. To see that the theory really does need to be modified, we consider the following example.

Example 5.1.1 Let $\Omega \equiv (\lambda x.xx)(\lambda x.xx)$ be the standard unsolvable term. Then $\lambda x.\Omega = \Omega$ in the standard theory, since $\lambda x.\Omega$ is also unsolvable; but $\lambda x.\Omega$ is in weak head normal form, hence should be distinguished from Ω in our "lazy" theory.

We now turn to a second point in which the standard theory is not completely satisfactory.

Is the λ-calculus a programming language?

In the standard theory, the λ-calculus may be regarded as being characterized by the type equation

$$D \quad = \quad [D \to D]$$

(for justification of this in a general categorical framework, see *e.g.* [Sco80, Koy82, LS86]).

It is one of the most remarkable features of the various categories of domains used in denotational semantics that they admit non-trivial solutions of this equation. However, there is no *canonical* solution in any of these categories (in particular, the initial solution is trivial – the one-point domain).

We regard this as a symptom of the fact that the pure λ-calculus in the standard theory *is not a programming language.* Of course, this is to some extent a matter of terminology, but we feel that the expression "programming language" should be reserved for a formalism with a definite computational interpretation (an operational semantics). The pure λ-calculus as ordinarily conceived is too schematic to qualify.

5.2 Call-by-name or Lazy λ-calculus

We introduce a "toy" functional language that has closed λ-terms as ***programs*** and (closed) *abstractions* as ***values***. The operational semantics is given by a ***Martin-Löf style evaluation relation*** (which is also known as ***"big-step" reduction relation***) simulating a normal order (or leftmost) reduction strategy that terminates whenever the reduction reaches a weak head normal form (WHNF).

Definition 5.2.1 We define a family $\Downarrow_n$ ($n \in \omega$) of binary relations over closed λ-terms as follows. For each n, the relation $s \Downarrow_n v$ ("the program s converges to *value* v in n steps") is defined inductively by the following rules:

$$\lambda x.p \Downarrow_0 \lambda x.p \qquad \frac{s \Downarrow_m \lambda x.p \quad p[t/x] \Downarrow_n v}{st \Downarrow_{m+n+1} v}$$

Notation It is useful to fix some shorthand.

$$\begin{array}{rcll} s \Downarrow v & \stackrel{\text{def}}{=} & \exists n \in \omega . s \Downarrow_n v & \text{"}s \textbf{ converges to } v\text{"} \\ s\Downarrow & \stackrel{\text{def}}{=} & \exists v . s \Downarrow v & \text{"}s \textbf{ converges}\text{"} \\ s\Uparrow & \stackrel{\text{def}}{=} & \neg[s \Downarrow] & \text{"}s \textbf{ diverges}\text{"} \end{array}$$

For example, $\mathbf{i(ii)} \Downarrow \mathbf{i}$ and $\mathbf{k(ii)} \Downarrow \lambda y.\mathbf{ii}$, and $\Omega\Uparrow$. Take a λ-term s that is not in β-normal form. Informally the ***leftmost β-redex*** of s is the redex that literally "occurs leftmost" in s. We define a reduction strategy informally: at each step, contract the leftmost redex and stop as soon as an abstraction (***weak head normal form***) is reached. Convince yourself that for any program s, $s \Downarrow v$ if and only if s reduces to v by the reduction strategy.

Proposition 5.2.2 *(i) Show that* $(\lambda x.p)t\vec{r} \Downarrow_{n+1} v \iff p[t/x]\vec{r} \Downarrow_n v$.

(ii) Prove that $\Downarrow$ *is* deterministic *i.e. it defines a partial function from programs to values: whenever* $s \Downarrow v$ *and* $s \Downarrow v'$ *then* v *and* v' *are the same.* □

The CBN λ-calculus was first introduced by Plotkin in [Plo75]. An extensive study of the calculus can be found in [AO93].

5.3 Applicative simulation and context lemma

Under the reduction strategy $\Downarrow$, the possible "results" are of a particularly simple, indeed *atomic* kind. That is to say, a term s either converges to an abstraction (and according to this strategy, we have no clue as to the structure "under" the abstraction), or it diverges. The relation $\Downarrow$ by itself is too "shallow" to yield information about the behaviour of a term under all experiments.

Inspired by the work of Robin Milner [Mil80] and David Park [Par80] on concurrency, we shall use the reduction relation $\Downarrow$ as a building block to yield a deeper relation which we call ***applicative simulation***. To motivate this relation, let us spell out the observational scenario we have in mind: Given a closed term s, the only experiment of depth 1 we can do is to evaluate s and see if it converges to some abstraction (weak head normal form) $\lambda x.p_1$. If it does so, we can continue the experiment to depth 2 by supplying a term t_1 as input to $\lambda x.p_1$, and so on. Note that what the experimenter can observe at each stage is only the *fact* of convergence, not which term lies under the abstraction. We can picture matters thus:

$$\begin{array}{ll} \text{Stage 1 of experiment:} & s \Downarrow \lambda x.p_1; \\ & \text{environment "consumes" } \lambda, \\ & \text{produces } t_1 \text{ as input} \\ \text{Stage 2 of experiment:} & p_1[t_1/x] \Downarrow \ldots \\ \vdots & \end{array}$$

Definition 5.3.1 We define a family of binary relations $\lesssim_k$ $(k \in \omega)$ over Λ^o as follows:

- for any s and s', $s \lesssim_0 s'$.
- $s \lesssim_{k+1} s'$ provided $\forall \lambda x.p.[s \Downarrow \lambda x.p \implies \exists \lambda x.p'.[s' \Downarrow \lambda x.p' \ \ \& \ \ \forall r \in \Lambda^o.p[r/x] \lesssim_k p'[r/x]\,]]$.

We then define $s \lesssim s'$ to be $s \lesssim_k s'$ for all $k \in \omega$. The definition can be extended to all λ-terms by considering closures in the usual way *i.e.* for $s, s' \in \Lambda$,

$$s \lesssim s' \quad \stackrel{\text{def}}{=} \quad \forall \sigma : \mathsf{var} \longrightarrow \Lambda^o.s_\sigma \lesssim s'_\sigma$$

where s_σ means the "term that is obtained from s by simultaneously substituting $\sigma(x)$ for each free occurrence of x, with x ranging over the collection var of λ-calculus variables". For example $\Omega \lesssim x$ and $\Omega x \lesssim x$.

Write $s \sim s'$ to mean $s \lesssim s'$ and $s' \lesssim s$; and set

$$\lambda\ell \quad \stackrel{\text{def}}{=} \quad \{\, s = t : s \sim t \text{ where } s, t \in \Lambda^o \,\}.$$

We say that s and s' are ***applicatively bisimilar*** or simply ***bisimilar*** just in case $s \sim s'$. The theory $\lambda\ell$ is clearly (non-trivial and) consistent.

Exercise 5.3.2 (i) Show that $\precsim$ is a preorder over Λ i.e. a reflexive and transitive binary relation.

(ii) Show that $(\lambda x.xx)(\lambda x.xx) \sim (\lambda x.xxx)(\lambda x.xxx) \precsim \lambda x.(\lambda x.xx)(\lambda x.xx)$; show that $\lambda x.x$, $\mathbf{k}$ and $\mathbf{s}$ are pairwise incompatible w.r.t. $\precsim$.

(iii) Suppose $s\Uparrow$ and $t\Downarrow$. Show that $\lambda x_1 \cdots x_n.s \precsim \lambda x_1 \cdots x_n.t$.

(iv) Show that $\lambda x_1 \cdots x_n.s \precsim \lambda x_1 \cdots x_m.s$ iff $n \leqslant m$.

For an alternative description of $\precsim$, recall that the set $\mathcal{R}$ of binary relations over Λ^o is a complete lattice under set inclusion. Now, define $F : \mathcal{R} \longrightarrow \mathcal{R}$ by

$$F(R) \stackrel{\text{def}}{=} \{\, (s,s') : \forall \lambda x.p.[s \Downarrow \lambda x.p \implies \exists \lambda x.p'.[\, s' \Downarrow \lambda x.p' \;\&\; \forall t \in \Lambda^o.([p[t/x], p'[t/x]) \in R]\,]\,\}$$

It is easy to check that F is a monotone function with respect to the inclusion ordering. A relation $R \in \mathcal{R}$ is said to be a ***pre-simulation*** just in case $R \subseteq F(R)$ i.e. R is a ***post-fixpoint*** of F. Since F is monotone, by Tarski's Theorem [Tar55], it has a maximal pre-simulation given by

$$\bigcup_{R \subseteq F(R)} R$$

since the *closure ordinal* [Mos74] of $\langle \precsim_k : k \in \omega \rangle$ is ω. Note that the maximal post-fixpoint of F is also its maximal fixpoint (and this holds generally).

Lemma 5.3.3 *Applicative simulation is precisely the maximal pre-simulation.* □

We give a useful characterization of $\precsim$.

Theorem 5.3.4 (Characterization) *For any $s, s' \in \Lambda^o$, $s \precsim s'$ if and only if for any finite (possibly empty) sequence $\vec{t}$ of closed λ-terms, if $s\vec{t}\Downarrow$ then $s'\vec{t}\Downarrow$.* □

To prove the theorem, we first establish a useful result:

Lemma 5.3.5 *(i) If $s \Downarrow \lambda x.p$ and $s' \Downarrow \lambda x.p'$ then for any $r \in \Lambda^o$, for any $n \geqslant 0$,*

$$sr \precsim_n s'r \iff p[r/x] \precsim_n p'[r/x].$$

(ii) Hence if s and s' are both convergent then $s \precsim_{n+1} s' \iff \forall r \in \Lambda^o.sr \precsim_n s'r$.

Proof (i) The case of $n = 0$ is vacuous. Assume $s \Downarrow \lambda x.p$ and $s' \Downarrow \lambda x.p'$. Then $sr \Downarrow \lambda y.q$ iff $p[r/x] \Downarrow \lambda y.q$, and $s'r \Downarrow \lambda y.q'$ iff $p'[r/x] \Downarrow \lambda y.q'$ Now for the case of $n = l+1$: by definition, $sr \precsim_{l+1} s'r$ iff if $sr \Downarrow \lambda y.q$ then $s'r \Downarrow \lambda y.q'$ and for any closed t, $q[t/y] \precsim_l q'[t/y]$; i.e. iff if $p[r/x] \Downarrow \lambda y.q$ then $p'[r/x] \Downarrow \lambda y.q'$ and for any closed t, $q[t/y] \precsim_l q'[t/y]$; i.e. iff $p[r/x] \precsim_{l+1} p'[r/x]$. (ii) follows from (i) and the definition of $\precsim_{n+1}$. □

We define a family of relations $\lessdot_n$ with $n \geqslant 0$: $s \lessdot_0 s'$ holds for any s and s'; for $n \geqslant 0$ we define $s \lessdot_n s'$ by "for any finite sequence $\vec{t} \equiv t_1, \cdots, t_m$ such that $m < n$, if $s\vec{t}\Downarrow$ then $s'\vec{t}\Downarrow$". To prove the theorem, it suffices to show:

for all $n \geqslant 0$, $\lessdot_n$ and $\precsim_n$ are equal.

We shall prove it by induction on n. The base case is obvious. For the inductive case of $n = l+1$, we may assume w.l.o.g. that s and s' are both convergent. Observe that $s \lessdot_{n+1} s'$ iff "whenever $s\Downarrow$ then $s'\Downarrow$, and for any closed t, $st \lessdot_n s't$". Hence

$$\begin{array}{lll} & s \lessdot_{l+1} s' & \text{by the preceding and assumption} \\ \iff & \forall t.st \lessdot_l s't & \text{by induction hypothesis} \\ \iff & \forall t.st \sqsubseteq_l s't & \text{by Lemma 5.3.5(ii)} \\ \iff & s \sqsubseteq_{l+1} s'. & \end{array}$$

Hence the theorem is proved.

Recall that programs are closed terms. Thus ***program contexts*** are just closed contexts i.e. contexts that have no free λ-variables. We say that s ***observationally approximates*** s' just in case for any program context $C[X]$, if $C[s]$ converges then so must $C[s']$. Informally this means that whatever we can observe about s, the same can be observed about s'. (Note that convergence is the only thing we can observe about a computation in the CBN λ-calculus.)

Definition 5.3.6 The binary relation $\sqsubseteq^{\text{cxt}}$ over Λ^o, called ***observational*** or ***contextual preorder*** is defined as

$$s \sqsubseteq^{\text{cxt}} s' \stackrel{\text{def}}{=} \forall C[X] \in \Lambda^o.C[s]\Downarrow \implies C[s']\Downarrow.$$

Observational equivalence captures the intuitive idea that two program fragments are indisguishable in all possible programming contexts. Though observational preorder is clearly important, it is hard to reason about it *directly*. Try proving that $\lambda x.x\Omega \sqsubseteq^{\text{cxt}} \lambda x.xx$ or $\lambda x.xx \sqsubseteq^{\text{cxt}} \lambda x.(\lambda y.xy)$. Fortunately there is a convenient characterization.

Proposition 5.3.7 (Context lemma) *Applicative simulation and context preorder coincide.*

Proof This is a variation of Berry's proof of a Context Lemma in [Ber81].

It suffices to prove the following: Let s, s' range over Λ^o.

$$s \sqsubseteq s' \implies \forall l \in \omega.\forall C[X] \in \Lambda^o.C[s]\Downarrow_l \implies C[s']\Downarrow.$$

We prove the assertion by induction on l. The base case is obvious. Without loss of generality, consider the following two cases of closed contexts:

(1) $C[X] \equiv (\lambda x.P[X])(Q[X])R[\vec{X}]$,

(2) $C[X] \equiv X(P[X])Q[\vec{X}]$.

(1): Suppose $C[s]\Downarrow_{l+1}$. Define $D[X] \equiv (P[X])[Q[X]/x]R[\vec{X}]$. Then by Proposition 5.2.2 $D[s]\Downarrow_l$. Invoking the induction hypothesis, we have $D[s']\Downarrow$, which implies that $C[s']\Downarrow$.

(2): Let $s \equiv (\lambda x.p)\vec{q}$. Suppose $C[s]\Downarrow_{l+1}$. Define $D[X] \equiv (\lambda x.p)\vec{q}(P[X])Q[\vec{X}]$, a context of case (1). Note that $C[s] \equiv D[s]$. By an appeal to (1), we have $D[s']\Downarrow$. But $D[s'] \equiv sP[s']Q[\vec{s'}]$, and so by Theorem 5.3.4, because $s \sqsubseteq s'$, we have $s'P[s']Q[\vec{s'}]\Downarrow$, *i.e.* $C[s']\Downarrow$. □

Remark 5.3.8 **(i)** The above result says that if two programs are distinguishable by some program context then there is some *applicative context* that distinguishes them. In other words, the computational behaviour of CBN λ-calculus program is *functional*, which is what one would expect of a functional programming language. This property is called ***operational extensionality*** in [Blo88]. Milner [Mil77] proved a similar result in the case of simply typed combinatory algebra which he referred to as the ***Context Lemma***.

(ii) It follows immediately from the definition of $\precsim$ that the application operation in Λ^o is monotone in the *left* argument with respect to $\precsim$. Operational extensionality is equivalent to the monotonicity of the application operation in the *right* argument, *i.e.*

$$s \precsim s' \quad \Longrightarrow \quad \forall t \in \Lambda^o . ts \precsim ts';$$

which is the same as saying that $\precsim$ is a ***precongruence*** *i.e.*

$$s \precsim s' \ \& \ t \precsim t' \quad \Longrightarrow \quad st \precsim s't'.$$

5.4 Call-by-value λ-calculus

We let p, r, s and t range over λ-terms. ***Programs*** of Plotkin's *call-by-value* (CBV) λ-calculus are closed λ-terms, and ***values***, ranged over by u and v, are closed abstractions. Evaluation is defined by induction over the following rules: for programs $\lambda x.p, s$ and t

$$\lambda x.p \Downarrow \lambda x.p \qquad \frac{s \Downarrow \lambda x.p \quad t \Downarrow u \quad p[u/x] \Downarrow v}{st \Downarrow v}.$$

As before we read $s \Downarrow v$ as "program s converges or evaluates to value v", and write $s\Downarrow$ to mean $s \Downarrow v$ for some value v.

Notation: We shall not bother to distinguish *notationally* the evaluation relation of the CBV λ-calculus from that of the CBN λ-calculus, though they are of course distinct relations.

We present the operational semantics in terms of a ***Plotkin-style transition relation*** (which is also known as ***"small-step" reduction relation***) by induction over the following rules:

$$(\lambda x.p)v > p[v/x] \qquad \frac{s > s'}{E[s] > E[s']}$$

where $E[X]$ ranges over the collection of ***evaluation contexts*** defined by the following rules: v and s range over values and programs respectively

- X is an evaluation context
- if E is an evaluation context, then so is vE
- if E is an evaluation context, then so is Es.

Note that by definition, the hole occurs exactly once in every evaluation context. We call a term of the shape $(\lambda x.p)v$ a CBV β-redex, and write $\gg$ to be the reflexive, transitive closure of $>$.

Lemma 5.4.1 (Evaluation context) *For any program s, $s > s'$ iff there is a unique evaluation context $E[X]$ and a unique* CBV *redex $\Delta \equiv (\lambda x.p)v$ such that $E[\Delta] \equiv s$ and $s' \equiv E[p[v/x]]$. Hence big-step (Martin-Löf style evaluation relation) and small-step operational semantics coincide.* □

Proposition 5.4.2 (Equivalence) *For any program s, $s \Downarrow v$ iff $s \gg v$ where v is a value.* □

As in the case of CBN λ-calculus, for closed terms s and t, we define $s \sqsubseteq\!\!\!\approx t$, read s ***simulates*** t ***applicatively***, as the conjunction of a countable family of binary relations as follows:

- for any s and s', $s \sqsubseteq\!\!\!\approx_0 s'$.
- $s \sqsubseteq\!\!\!\approx_{k+1} s'$ just in case whenever $s \Downarrow \lambda x.p$ then $s' \Downarrow \lambda x.p'$ and for every value v, $p[v/x] \sqsubseteq\!\!\!\approx_k p'[v/x]$.

We then define $s \sqsubseteq\!\!\!\approx s'$ to be $s \sqsubseteq\!\!\!\approx_k s'$ for all $k \in \omega$. The relation can be extended to λ-terms in general: for any s and t, define $s \sqsubseteq\!\!\!\approx t$ just in case $s_\sigma \sqsubseteq\!\!\!\approx t_\sigma$ for every *value substitution* σ.

Proposition 5.4.3 *For any closed terms s and t, the following are equivalent:*

(i) $s \sqsubseteq\!\!\!\approx t$

(ii) for every finite sequence of closed terms $r_1, \cdots, r_n$, if $s\vec{r}\Downarrow$ then $t\vec{r}\Downarrow$

(ii) for every finite sequence of values $v_1, \cdots, v_n$, if $s\vec{v}\Downarrow$ then $t\vec{v}\Downarrow$.

□

5.5 Context lemma by Howe's method

Context lemma is valid for CBV λ-calculus but the argument in the proof of Proposition 5.3.7 does not work for the CBV calculus. We shall present a proof using what is known as Howe's method as an extended exercise.

A ***value substitution*** σ is just a function σ from variables to values. Suppose the variables occurring free in s are $x_1, \cdots, x_n$ then

$$s_\sigma \stackrel{\text{def}}{=} s[\sigma(x_1)/x_1, \cdots, \sigma(x_n)/x_n].$$

Exercise 5.5.1 *Prove the following:*

(i) $\sqsubseteq\!\!\!\approx$ *is a preorder.*

(ii) For any s and t (which are not necessarily closed) and for any value v,

$$s \sqsubseteq\!\!\!\approx t \implies s[v/x] \sqsubseteq\!\!\!\approx t[v/x].$$

Definition 5.5.2 (Pre-simulation) Let $\mathcal{R}$ be the set of binary relations over the set of closed λ-terms. Define a function $F : \mathcal{R} \longrightarrow \mathcal{R}$ by: for any $R \in \mathcal{R}$

$$F(R) \stackrel{\text{def}}{=} \{\, (s, s') : \forall v. s \Downarrow v \implies [\, \exists v'. s' \Downarrow v' \ \&\ \forall t. (vt, v't) \in R \,] \,\}.$$

F is a monotone function with respect to the inclusion ordering. A relation $R \in \mathcal{R}$ is said to be a *pre-simulation* just in case $R \subseteq F(R)$. Define $\lesssim$ to be the maximal pre-simulation i.e.

$$\lesssim \stackrel{\text{def}}{=} \bigcup_{R \subseteq F(R)} R.$$

Exercise 5.5.3 *Prove the following:*

(i) F *is a monotone function (with respect to the inclusion ordering).*

(ii) $\lesssim$ *is the same as* $\sqsubseteq$.

Our aim is to prove the Context Lemma.

Definition 5.5.4 (Precongruence candidate) Define a binary relation $\leqslant$, called *precongruence candidate*, over the collection of all (not just closed) λ-terms by induction over the following rules:

- if $x \sqsubseteq s$ then $x \leqslant s$
- if $s \leqslant s'$ and $t \leqslant t'$ and $s't' \sqsubseteq r$ then $st \leqslant r$
- if $s \leqslant s'$ and $\lambda x.s' \sqsubseteq r$ then $\lambda x.s \leqslant r$.

Exercise 5.5.5 *Prove the following:*

(i) Whenever $s \leqslant t$ *and* $t \sqsubseteq r$ *then* $s \leqslant r$.

(ii) $\leqslant$ *is a precongruence i.e. whenever* $s \leqslant s'$ *and* $t \leqslant t'$ *then* $st \leqslant s't'$, *and whenever* $s \leqslant s'$ *then* $\lambda x.s \leqslant \lambda x.s'$.

Exercise 5.5.6 *Prove that* $\leqslant$ *is reflexive. Hence deduce that* $\sqsubseteq$ *is contained in* $\leqslant$.

Lemma 5.5.7 (Substitution Lemma) *Prove that whenever* $s \leqslant s'$ *and values* $v \leqslant v'$ *then*

$$s[v/x] \quad \leqslant \quad s'[v'/x].$$

□

Exercise 5.5.8 *For closed* s *and* s', *if* $s \leqslant s'$ *and* $s \Downarrow v$, *then for some* v', $s' \Downarrow v'$ *and* $v \leqslant v'$.

[Hint: Define a notion of "convergence in n steps" $s \Downarrow_n v$, and prove by induction over n, using the Substitution Lemma.]

Exercise 5.5.9 *Prove that* $\leqslant$ *coincides with* $\sqsubseteq$. *Hence deduce the context lemma.*

[Hint: To prove that $\leqslant$ is contained in $\sqsubseteq$, it suffices to show that $\leqslant$ is a pre-simulation (why?).]

Problems

Unless otherwise specified, assume $\Downarrow$ *and* $\sqsubseteq$ *as defined in the* CBN λ*-calculus in the following.*

5.1 Formalize a small-step reduction for the CBN λ-calculus and prove that it is equivalent (in the sense of Proposition 5.4.2) to the big-step presentation.

5.2 Prove Proposition 5.2.2.

5.3 Prove Lemma 5.3.3.

5.4 (i) Show that $\Omega \equiv (\lambda x.xx)(\lambda x.xx)$ is a bottom element and $\mathbf{yk}$ a top element with respect to applicative simulation.

(ii) *A classification of closed λ-terms.*

For any (closed) λ-term s, say that s has ***order 0*** just in case s is not β-conertible to an abstraction. Suppose s is β-convertible to an abstraction. For $n \geqslant 1$, say that s has ***order*** n if n is largest k such that for some p, $\lambda\beta \vdash s = \lambda x_1 \cdots x_k.p$. We say that s has ***order*** ∞ just in case for no $n \in \omega$ is s of order n. Observe that every closed λ-term has a unique order.

Show that a λ-term is a bottom element w.r.t. applicative simulation iff it is of order 0; and top element iff it is of order ∞.

5.5 ***$\lambda\ell$ is a λ-theory***

(i) Is it true that if $\lambda\beta \vdash s = t$ then $s \sim t$? Is it true that if $s = s'$ and $t = t'$ in $\lambda\beta$ and if $s \precsim t$ then $s' \precsim t'$?

(ii) Prove that $\lambda\ell$ is a λ-theory.

(iii) Show that the axiom (η) is not valid in $\lambda\ell$. Rather a weaker version, called ***conditional-η***,

$$s{\downarrow} \implies \lambda x.sx = s$$

is valid, where we interpret $s{\downarrow}$ to mean "s converges".

5.6 (i) Show that $xx \precsim x(\lambda y.xy)$ in the CBN λ-calculus. Is it true in the CBV λ-calculus?

(ii) Are there $\beta\eta$-inequivalent β-normal forms that are equal in $\lambda\ell$?

(iii) The answer to (ii) is yes if we relax the β-normality requirement, or if the pair are only required to be β-inequivalent. Why?

5.7 ***Convergence testing*** ★

(i) A convergence test is a closed λ-term $\mathbf{c}$ such that $\mathbf{c}{\Downarrow}$, and for any $s \in \Lambda^o$

$$\begin{cases} s{\Downarrow} & \implies \quad \mathbf{c}s \Downarrow \lambda x.x \\ s{\Uparrow} & \implies \quad \mathbf{c}s{\Uparrow}. \end{cases}$$

Show that there is no convergence test in the CBN λ-calculus.

(ii) Let $\top$ be any order-∞ term, and $\bot$ any order-0 term. Let $p \equiv \lambda x.x(\lambda y.x\top\bot y)\top$ and $q \equiv \lambda x.x(x\top\bot)\top$. Prove that $p \sim q$.

(iii) Let p' and q' be obtained from p and q respectively by replacing $\top$ in them by $\lambda y.\bot$. Prove that we still have $p' \sim q'$.

(iv) Show that there is a convergence test in the CBV λ-calculus.

5.8 Describe, and characterize if possible, the least and greatest terms w.r.t. $\precsim$ in the CBV λ-calculus.

5.9 Use Howe's method to prove that $\precsim$ in the CBN λ-calculus is a precongruence.