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`.]

Contents

1 Syntax of the λ-calculus

In this section we introduce the syntax of the untyped λ-calculus and fix some notations. Substitution is a key operation of the λ-calculus, which should be contrasted with context substitution. There are fixed-point operators in the λ-calculus – this has to do with the possibility of self-application in the *untyped* λ-calculus. The theories $\boldsymbol{\lambda\beta}$ and $\boldsymbol{\lambda\beta\eta}$ are introduced.

1.1 Basic definitions

The syntax of the λ-calculus is remarkably simple. λ-terms are defined by induction over the following rules:

- any variable is a λ-term
- if s and t are λ-terms then so is $(s \cdot t)$ which is called an ***application***
- if s is a λ-term then the λ-*abstraction* (or simply ***abstraction***) $(\lambda x.s)$ is a λ-term.

Remark 1.1.1 (i) We use *meta*-variables s, s', s_i, t, t', etc. to range over λ-terms, and x, y, z, x_i, x' etc. to range over (denumerably many) variables. (Do not confuse the *object*-variables i.e. x, y, z etc. with meta-variables.)

(ii) The symbols (,), $\cdot$ are part of the language. They play an important rôle in disambiguating the structure of expressions. It is possible to minimize their use in a safe way. We write $(s \cdot t)$ simply as st and $(\lambda x.s)$ as $\lambda x.s$, omitting $\cdot$ and as many parentheses as we can get away with, subject to the following convention:

- *abstraction associates to the right*: $\lambda x_1 \cdots x_n.s$ means $\lambda x_1.(\lambda x_2. \cdots (\lambda x_n.s) \cdots)$.
- *application associates to the left*: $s_1 \cdots s_n$ means $(\cdots (s_1 s_2) \cdots s_n)$.

(iii) $\lambda \vec{x}.s$ and $\vec{s}$ are shorthand for $\lambda x_1 \cdots x_n.s$ and $s_1 \cdots s_n$ respectively, for $n \geqslant 0$. So for example $s\vec{t}u$ is a shorthand for $st_1 \cdots t_n u$ for some $n \geqslant 0$.

1.2 Variables

An occurrence of a variable x in s is said to be ***bound*** if it is in the scope of some abstraction $\lambda x.-$ in s; otherwise x is ***free*** in s. Formally we define the set $\mathsf{fv}(s)$ of free variables of s by recursion as follows:

$$\begin{aligned}\mathsf{fv}(x) &\stackrel{\text{def}}{=} \{\, x \,\} \\ \mathsf{fv}(st) &\stackrel{\text{def}}{=} \mathsf{fv}(s) \cup \mathsf{fv}(t) \\ \mathsf{fv}(\lambda x.s) &\stackrel{\text{def}}{=} \mathsf{fv}(s) - \{\, x \,\}.\end{aligned}$$

A term is said to be ***closed*** if every variable occurrence in it is bound. We write $\mathbf{\Lambda}$ for the set of λ-terms, and $\mathbf{\Lambda}^o$ for the set of closed λ-terms.

1.3 Important convention

α-convertibility Two terms s and t are said to be ***α-convertible***, written $s =_\alpha t$, if one is obtainable from the other by renaming bound variables. E.g.

$$\lambda xy.x \quad =_\alpha \quad \lambda zy.z \quad =_\alpha \quad \lambda zx.z.$$

We regard α-convertible terms as *identical* at the syntactic level; they are to all intents and purposes equal. We shall use $\equiv$ to mean *syntactic* equality; and reserve the more common symbol $=$ for β-convertibility. So $s =_\alpha t$ implies $s \equiv t$.

Variable convention We state the convention informally as:

> "We shall assume that there is an inexhaustible supply of fresh variable names so that given any (finite) number of λ-terms $s_1, \cdots, s_n$, bound variables occurring in them are renamed where necessary in such a way that none is the same as any variable occurring free in $s_1, \cdots, s_n$."

1.4 β-conversion and substitution

What do λ-terms denote? λ-calculus is a theory of functions. Application is a binary operator. The λ-abstractor "$\lambda x.-$" in any abstraction $\lambda x.s$ can be thought of as a term-constructor of arity one. A term may act both as an operator (function) and as an operand (argument). E.g. x in the term xx.

β-conversion Think of λ-terms as programs for the moment. What happens when a λ-term (an abstraction) is applied to another?

$$(\lambda x.s)t \quad = \quad s[t/x]$$

where $s[t/x]$ means "in s substitute t for every *free* occurrence of x". **Substitution** is a very important operation in formal logic. In the λ-calculus

- substitution is an *implicit* operation i.e. the expression "$s[t/x]$" is not part of the object language; we are to understand "$s[t/x]$" as *denoting* the λ-term that is obtained from s by substituting t for every free occcurrence of x in s.
- substitution is an *unrestricted* operation; any term may be substituted for any variable. This is to be contrasted with the substitution mechanism of, say, the π-calculus of Milner, Parrow and Walker [MPW92], in which only names (as opposed to all π-terms) may participate in the operation.

Substitution may be defined by recursion as follows:

$$\begin{aligned}
x[s/y] &\stackrel{\text{def}}{=} \begin{cases} s & \text{if } x \equiv y \\ x & \text{otherwise} \end{cases} \\
(uv)[s/y] &\stackrel{\text{def}}{=} (u[s/y]v[s/y]) \\
(\lambda x.t)[s/y] &\stackrel{\text{def}}{=} \lambda x.(t[s/y]).
\end{aligned}$$

Note that in the *absence* of the variable convention, the last clause should be replaced by

$$(\lambda x.t)[s/y] \quad \stackrel{\text{def}}{=} \quad \begin{cases} \lambda x'.(t[x'/x][s/y]) & \text{if } x \equiv y \text{ or } x \in \mathsf{fv}(s) \\ \lambda x.(t[s/y]) & \text{otherwise} \end{cases}$$

For example $(\lambda xy.zy)[yy/z]$ is $\lambda xu.(yy)u$.

Proposition 1.4.1 (Nested substitution) *For any variable x distinct from y, if x does not occur free in u,*

$$s[t/x][u/y] \quad \equiv \quad s[u/y][t[u/y]/x].$$

Proof We prove by induction on the structure of s. Consider the base case of s being a variable. If $s \equiv x$ then both lhs and rhs are $t[u/y]$. If $s \equiv y$ then the lhs is u; the rhs $y[u/y][t[u/y]/x]$ is $u[t[u/y]/x]$ which is u since x does not occur free in u. For the remaining case of s being a variable distinct from x and y, both sides give s. Next suppose $s \equiv s_1 s_2$.

$$\begin{array}{rcll} (s_1 s_2)[t/x][u/y] & \equiv & (s_1[t/x][u/y])(s_2[t/x][u/y]) & \text{by induction hypo.} \\ & \equiv & (s_1[u/y][t[u/y]/x])(s_2[u/y][t[u/y]/x]) & \\ & \equiv & (s_1[u/y] s_2[u/y])[t[u/y]/x] & \\ & \equiv & (s_1 s_2)[u/y][t[u/y]/x]. & \end{array}$$

The case of s being an abstraction is left as an easy exercise. □

1.5 Formal theories $\lambda\beta$ and $\lambda\beta\eta$

[We shall assume knowledge of elementary logic; see for example Hamilton's [Ham88] or Mendelson's book [Men87].] A *theory* is a collection of formulae closed under a notion of provability or derivability. In this course terms are just the λ-terms and formulae are equations between terms, written $s = t$.

Proof system $\lambda\beta$

There are three groups of axiom and rule *schema*.

(1) equivalence: these are the rules that define $=$ to be an equivalence relation

$$\textbf{(reflexivity)} \qquad s = s$$

$$\textbf{(symmetry)} \qquad \frac{s = t}{t = s}$$

$$\textbf{(transitivity)} \qquad \frac{s = t \qquad t = u}{s = u}$$

(2) *compatible closure*: these rules ensure that $=$ is a *congruence* i.e. $=$ is preserved by all contexts

$$\textbf{(application)} \quad \frac{s = s' \qquad t = t'}{st = s't'}$$

$$\textbf{(abstraction)} \quad \frac{s = t}{\lambda x.s = \lambda x.t}$$

(3) β-conversion

$$(\beta) \qquad (\lambda x.s)t \quad = \quad s[t/x].$$

The formal theory $\boldsymbol{\lambda\beta\eta}$ is $\boldsymbol{\lambda\beta}$ extended by the following axiom scheme

$$\lambda x.sx = s \qquad \text{provided } x \text{ does not occur free in } s.$$

We write $\boldsymbol{\lambda\beta} \vdash s = t$ to mean that $s = t$ is provable in the theory $\boldsymbol{\lambda\beta}$. Similarly for $\boldsymbol{\lambda\beta\eta}$.

Notation We shall often write $\boldsymbol{\lambda\beta} \vdash s = t$ simply as $s = t$.

Here are some questions that we should ask about the theories:

(1) Is $\boldsymbol{\lambda\beta}$ (or $\boldsymbol{\lambda\beta\eta}$) consistent? (A theory is said to be *consistent* if there is a formula which is not a theorem. Warning: consistency has several subtly different meanings in logic.)

(2) Is $\boldsymbol{\lambda\beta}$ (or $\boldsymbol{\lambda\beta\eta}$) maximally consistent (i.e. for any s and t, either $\boldsymbol{\lambda\beta} \vdash s = t$, or $\boldsymbol{\lambda\beta} + (s = t)$ – the theory obtained by augmenting $\boldsymbol{\lambda\beta}$ by the equation $(s = t)$ – is inconsistent)?

(3) Is equality $=$ in $\boldsymbol{\lambda\beta}$ (or in $\boldsymbol{\lambda\beta\eta}$) decidable?

1.6 Fixed points

For λ-terms f and u, u is said to be a ***fixed point*** of f if $fu = u$. A ***fixed-point combinator*** is a (closed) term f such that $s(fs) = fs$ for all λ-term s. (A ***combinator*** is just a closed λ-term, for which more anon.) In the λ-calculus, the so-called first recursion theorem is (almost) a triviality.

Proposition 1.6.1 (First Recursion Theorem) *There are (many) fixed-point combinators in the λ-calculus.* □

Here are two well-known ones:

- Curry's "paradoxical" combinator: $\mathbf{y} \stackrel{\text{def}}{=} \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$
- Turing's fixed-point combinator: $\boldsymbol{\Theta} \stackrel{\text{def}}{=} AA$ where A is defined to be $\lambda xy.y(xxy)$.

For example $\mathbf{y}g = (\lambda x.g(xx))(\lambda x.g(xx)) = g((\lambda x.g(xx))(\lambda x.g(xx))) = g(\mathbf{y}g)$.

1.7 Contexts

Intuitively these are λ-terms that contain "holes". We use meta-variables X, X', Y, Y' etc. to denote such "holes" – call them *hole-variables*. Examples of context: $\lambda x.XXx$ (or more suggestively $\lambda x.[\,][\,]x$), $X(zyX(\lambda x.Y))$. Contexts are ranged over by C, C', D etc.

Definition 1.7.1 ***Contexts*** (or λ-*contexts*) are defined by the following BNF rule:

$$C \quad ::= \quad x \quad | \quad X \quad | \quad (CC) \quad | \quad (\lambda x.C).$$

As usual we adopt the convention of leaving out as many parentheses as we can get away with. We often write a context C in a more informative way as $C[X_1, \cdots, X_n]$ whenever the hole-variables occurring in C belong to the set $\{ X_1, \cdots, X_n \}$.

Context substitution

It is important to distinguish context substitution from variable substitution: in the former, ***variable capture*** may happen i.e. variables may become bound as a result of the operation; but not in the latter. For example take $C[X]$ to be $\lambda x.Xyx$. Then $C[x]$ is $\lambda x.xyx$ – x is bound or "captured" as a result; contrast this with $(\lambda x.zyx)[x/z]$, which is $\lambda u.xyu$.

Formally we define $C[s_1, \cdots, s_n]$, the ***context-substitution*** of λ-terms $s_1, \cdots, s_n$ for hole-variables $X_1, \cdots, X_n$ in $C \equiv C[X_1, \cdots, X_n]$, as follows: (we shall write $C[\vec{s}]$ as a short hand for $C[s_1, \cdots, s_n]$)

$$C[s_1, \cdots, s_n] \quad \stackrel{\text{def}}{=} \quad \begin{cases} C & \text{if } C \text{ is a term variable} \\ s_i & \text{if } C \text{ is } X_i \\ C_1[\vec{s}]C_2[\vec{s}] & \text{if } C \equiv C_1C_2 \\ \lambda x.C'[\vec{s}] & \text{if } C \equiv \lambda x.C'. \end{cases}$$

Context is an important tool for reasoning about properties of syntax.

Problems

Problems contained in this exercise (and in future installments) supplement the lectures. Students are advised to work through them. Problems marked with ★ may be difficult.

1.1 (i) Rewrite $((xy)(\lambda y(\lambda z.(z(xy)))))$ using the minimum number of parentheses

(ii) Fill in all possible parentheses in $(\lambda xyz.xy(xz))\lambda xy.x$.

1.2 Perform the following substitutions:

(i) $(\lambda x.yx)[yz/x]$

(ii) $(\lambda y.xy)[yx/x]$

(iii) $(\lambda z.(\lambda x.yx)xz)[zx/x]$

(iv) $C[yz]$ where $C[X] \equiv \lambda z.(\lambda x.yx)Xz$

(v) $C[yx]$ where $C[X] \equiv \lambda y.Xy$.

1.3 A ***proof*** of the formal system $\lambda\beta$ is a finite sequence l of formulae (=equations) such that every formula θ of l is either an instance of an axiom, or it is the conclusion of an instance of a rule whose corresponding instances of the premises occcur to the left of θ in l.

Write down a proof of $(\lambda x.(\lambda z.z)((\lambda y.y)x))u = (\lambda x.xx)u$, and construct its ***proof tree***.

1.4 Prove the following:

(i) if $s = t$ then $s[u/x] = t[u/x]$ for any u

(ii) if $s = t$ then $u[s/x] = u[t/x]$ for any u

(iii) if $s = t$ and $p = q$ then $s[p/x] = t[q/x]$.

1.5 Prove that if $s = t$ then for any context $C[-]$, $C[s] = C[t]$.

1.6 Show that there exists s such that $st = ss$ for all t.

1.7 Show that there is no λ-term f satisfying the following property:

for any λ-terms s and t, $f(st) = s$.

[Hint: use the Fixed Point Theorem.]

1.8 Use the Fixed-Point Theorem to construct:

1. a closed λ-term t such that $t = t\mathbf{s}$ where $\mathbf{s}$ is the standard S-combinator.
2. a closed λ-term M such that $M\mathbf{iss} = M\mathbf{s}$ where $\mathbf{i}$ is the standard identity combinator.

1.9 ★ Show that every fixed-point combinator can be characterized as a fixed point of a term G. Find G.

1.10 ★ Show that there are denumerably many (β-inequivalent) fixed-point combinators. Generate these combinators by a "uniform" procedure.

1.11 It is known that the η-axiom is not derivable from the formal system $\lambda\beta$. († Can you show it?) However for any s which is β-equivalent to a λ-abstraction, $\lambda x.sx = s$, for x not occurring free in s. Why is this so?

2 Reduction

Using β-reduction as the main example, we introduce the basic notions of term rewriting such as weak and strong normalization, and Church-Rosser. β-reduction is shown to be Church-Rosser.

2.1 Preliminaries: rule induction

For an introduction to *rule induction* see e.g. the treatment in chapter 4 of Winskel's book [Win93] (especially the Principle of Rule Induction); for a more foundational approach, see Aczel's chapter in the Handbook of Mathematical Logic [Bar77].

Subterm of a λ-term is defined by induction as follows:

- a λ-term is a subterm of itself
- if u is a subterm of s then it is a subterm of $\lambda x.s$
- if u is a subterm of s the it is a subterm of both st and ts.

Let $\mathcal{T}$ be a set of terms. Typically $\mathcal{T}$ is defined by induction over a set of ***formation rules***. The formation rule $\mathcal{R}_{\mathsf{c}}$ defining a constructor c of $\mathcal{T}$ has the general form:

$$\mathcal{R}_{\mathsf{c}} \qquad \frac{s_1 \in \mathcal{T} \quad \cdots \quad s_n \in \mathcal{T}}{\mathsf{c}(s_1, \cdots, s_n) \in \mathcal{T}}$$

where c is a term constructor. For example the collection $\mathbf{\Lambda}$ of λ-terms can be defined by induction over the following formation rule *schema*:

$$x \in \mathbf{\Lambda} \qquad \frac{s \in \mathbf{\Lambda} \qquad t \in \mathbf{\Lambda}}{(s \cdot t) \in \mathbf{\Lambda}} \qquad \frac{s \in \mathbf{\Lambda}}{(\lambda x.s) \in \mathbf{\Lambda}}$$

2.2 Some basic notions of term rewriting

Term rewriting is a subject in theoretical computer science in its own right: for a survey, see the respective chapters in the MIT Press Handbook of Theoretical Computer Science [vL90] and the OUP Handbook of Logic in Computer Science [AGM93]. We shall not study term rewriting in general in this section but rather regard λ-calculus as a particular term rewriting system.

A ***redex rule*** (or ***notion of reduction***) R over $\mathcal{T}$ is just a binary relation R over $\mathcal{T}$ (of a certain kind). Take $\mathcal{T}$ to be the set of λ-terms, and R the redex rule

$$\beta \quad = \quad \{ \langle (\lambda x.s)t, s[t/x] \rangle : s \text{ and } t \text{ are } \lambda\text{-terms, } x \text{ a variable} \}.$$

We define the corresponding ***one-step β-reduction*** by induction over the following rule schema:

$$\frac{}{s \to t} \langle s, t \rangle \in \beta \qquad \frac{s \to s'}{st \to s't} \qquad \frac{t \to t'}{st \to st'} \qquad \frac{s \to s'}{\lambda x.s \to \lambda x.s'}$$

The first rule scheme is only applied whenever the predicate on the r.h.s., known as the ***side condition***, is satisfied.

Though β is the main redex rule we shall study in this course, the idea of one-step reduction is quite general. Given an arbitrary redex rule or notion of reduction R over a set $\mathcal{T}$ of terms, we define the corresponding one-step reduction, which we call ***one-step R-reduction***, as follows.

Definition 2.2.1 (Informal) A binary relation R over $\mathcal{T}$ is said to be closed under the formation rule $\mathcal{R}_{\mathsf{c}}$ (as above) *argumentwise* just in case for any $s_1, \cdots, s_n$, and for each i, if $s_i \, R \, s'_i$ then

$$\mathsf{c}(s_1, \cdots, s_i, \cdots, s_n) \, R \, \mathsf{c}(s_1, \cdots, s'_i, \cdots, s_n).$$

The ***compatible closure*** of the redex rule R, or ***one-step*** R***-reduction***, is defined to be the least (w.r.t. inclusion) binary relation containing R and closed under all the formation rules argumentwise.

Notation Let R be a redex rule or notion of reduction over $\mathcal{T}$.

$$\begin{array}{rcl} \to_R & \stackrel{\text{def}}{=} & \text{compatible closure of } R \text{ or one-step } R\text{-reduction} \\ \twoheadrightarrow_R & \stackrel{\text{def}}{=} & \text{reflexive, transitive closure of } \to_R \\ \to_R^+ & \stackrel{\text{def}}{=} & \text{transitive closure of } \to_R \\ =_R & \stackrel{\text{def}}{=} & \text{reflexive, symmetric, transitive closure of } \to_R. \end{array}$$

We shall be a little vague about what exactly is a redex rule or notion of reduction. Intuitively a notion of reduction is a binary relation from which we derive the corresponding one-step reduction (by taking compatible closure).

Proposition 2.2.2 *For* λ*-terms* s *and* t, $\boldsymbol{\lambda\beta} \vdash s = t$ *if and only if* $s =_\beta t$. $\square$

The following are obvious:

- $s \twoheadrightarrow_R t$ if and only if for some $n \geqslant 0$ and for some $s_1, \cdots, s_n$,
$$s \equiv s_0 \to_R s_1 \to_R \cdots \to_R s_n \equiv t;$$
- $s \to_R^+ t$ if and only if for some $n \geqslant 1$ and for some $s_1, \cdots, s_n$,
$$s \equiv s_0 \to_R s_1 \to_R \cdots \to_R s_n \equiv t.$$

Intuitively an R***-redex*** is the "smallest" syntactic unit that contributes to (an instance of) one-step reduction. A β***-redex*** is a λ-term that has the general shape $(\lambda x.s)t$, i.e., the shape of the lhs of the redex rule β.

Remark 2.2.3 Let R be a redex rule over $\mathcal{T}$. The following is generally valid:

$$s \to_R s' \quad \Longleftrightarrow \quad \begin{cases} \text{for some "one-holed" } \mathcal{T}\text{-context } C[X] \text{ and} \\ \text{for some } R\text{-redex } \Delta,\ C[\Delta] \equiv s \text{ and } s' \equiv C[\Delta'] \text{ and } \Delta \, R \, \Delta'. \end{cases}$$

A one-holed context is one in which the hole occurs exactly once.

2.3 Some desirable properties of term rewriting systems

(For the rest of this section, we shall assume that R is a redex rule over $\mathcal{T}$.)

A term s of $\mathcal{T}$ is said to be an R***-normal form*** (R-NF or simply ***normal form*** if R is clear from the context) provided there is no t for which $s \to_R t$. By definition of $\to_R$, a term s is an R-NF if and only if no subterm of s is an R-redex. A term s ***has an*** R***-normal form*** just in case s reduces to an R-normal form i.e. $s \to_R s_1 \to_R s_2 \to_R \cdots \to_R s_n$ and s_n is an R-normal form, for some terms $s_1, \cdots, s_n$.

Example 2.3.1 (i) $\Delta_n \equiv \lambda x.\underbrace{x \cdots x}_{n}$, $\mathbf{s} \equiv \lambda xyz.xz(yz)$, $\mathbf{k} \equiv \lambda xy.x$ are in β-NF.

(ii) $\mathbf{\Omega} \equiv (\lambda x.xx)(\lambda x.xx)$, $\Delta_m\Delta_n$ for $m,n \geqslant 2$, $\mathbf{yi}$ and $\mathbf{yk}$ do not have any β-NF, where $\mathbf{y}$ is any fixed-point combinator.

Definition 2.3.2 (i) A term s is said to be ***normalizable*** (w.r.t. a one-step reduction $\to_R$) if s has an R-normal form.

(ii) A term s is said to be ***strongly normalizable*** (w.r.t. $\to_R$) if there is no infinite one-step reduction emanating from s; equivalently every one-step reduction sequence emanating from s terminates at an R-normal form after finitely many steps.

(iii) The one-step reduction $\to_R$ is ***weakly normalizing*** if there is a reduction strategy that reduces every term to its R-normal form. A reduction ***strategy*** is a map that associates to each non-normal term a subterm that is a redex. A reduction strategy that reduces every term to its R-normal form if it has one is called ***normalizing***.

(iv) The one-step reduction $\to_R$ is ***strongly normalizing*** if every term s is strongly normalizable w.r.t. $\to_R$; equivalently there is no infinite one-step R-reduction sequence.

Clearly if s is strong normalizable then it is weakly normalizable. Hence if a one-step reduction is strongly normalizing then it is weakly normalizing. In untyped λ-calculus β-reduction is neither weakly nor strongly normalizing. However β-reduction in simply-typed λ-calculus and second-order polymorphic λ-calculus (or *System F*) is strongly normalizing.

Let R be a redex rule over $\mathcal{T}$. We say that $\to_R$ satisfies

(i) ***diamond property*** if $s \to_R t_1$ and $s \to_R t_2$ implies $t_1 \to_R t$ and $t_2 \to_R t$ for some t.

(ii) ***Church-Rosser property*** if $s \twoheadrightarrow_R t_1$ and $s \twoheadrightarrow_R t_2$ implies $t_1 \twoheadrightarrow_R t$ and $t_2 \twoheadrightarrow_R t$ for some t; equivalently if $\twoheadrightarrow_R$ satisfies the diamond property.

How are the two properties related?

Lemma 2.3.3 *Let R_0 be a binary relation. If R_0 satisfies the diamond property then the transitive closure of R_0 satisfies the diamond property.*

Proof By a "diagram chase". □

We say that a redex rule R (over $\mathcal{T}$), or $\to_R$, has ***unique normal form property*** if whenever a term has normal forms they are equal.

2.4 Church-Rosser property of β-reduction

The rest of the section is devoted to a proof of the following result:

Theorem 2.4.1 *The one-step β-reduction is Church-Rosser.*

We shall prove this theorem by a method of "parallel reduction" due to P. Martin-Löf and W. W. Tait. Define a notion of *parallel β-reduction* as a binary relation $\gg$ over λ-terms by induction over the following rules:

$$
\begin{array}{ll}
\text{(refl)} & s \gg s \\[2ex]
\text{(app)} & \dfrac{s \gg s' \qquad t \gg t'}{st \gg s't'} \\[2ex]
\text{(abs)} & \dfrac{s \gg s'}{\lambda x.s \gg \lambda x.s'} \\[2ex]
(||\text{-}\beta) & \dfrac{s \gg s' \qquad t \gg t'}{(\lambda x.s)t \gg s'[t'/x]}
\end{array}
$$

Our strategy shall be to prove

(1) $\gg$ satisfies the diamond property, and

(2) $\twoheadrightarrow_\beta$ is the transitive closure of $\gg$.

Hence by Lemma 2.3.3 $\twoheadrightarrow_\beta$ satisfies the diamond property. To establish (1) we first prove:

Lemma 2.4.2 (Substitution) *If $s \gg s'$ and $t \gg t'$ then $s[t/x] \gg s'[t'/x]$.*

Proof We prove by induction over the structure of s and by case analysis of the definition of $s \gg s'$ by rule induction.

- **(refl)** Suppose $s \equiv s'$.

s	$s[t/x]$	$s'[t'/x] \equiv s[t'/x]$
x	t	t'
y	y	y
pq	$p[t/x]q[t/x]$	$p[t'/x]q[t'/x]$
$\lambda y.p$	$\lambda y.(p[t/x])$	$\lambda y.(p[t'/x])$

- **(app)** Write $s \equiv s_1 s_2$, $s' \equiv s'_1 s'_2$. By supposition $s_1 \gg s'_1$ and $s_2 \gg s'_2$. Since s_1 and s_2 are smaller than s, by the induction hypothesis, $s_1[t/x] \gg s'_1[t'/x]$ and $s_2[t/x] \gg s'_2[t'/x]$. Hence the result follows from **(app)**.

- **(abs)** Exercise.

- **($||$-β)** Suppose $s \equiv (\lambda y.p)q$ and $s' \equiv p'[q'/y]$ with $p \gg p'$ and $q \gg q'$. By the induction hypothesis, $p[t/x] \gg p'[t'/x]$ and $q[t/x] \gg q'[t'/x]$. Thus

$$
\begin{array}{rcll}
(\lambda y.p)q[t/x] & \equiv & (\lambda y.p[t/x])(q[t/x]) & \\
& \gg & p'[t'/x][q'[t'/x]/y] & \text{by } (||\text{-}\beta) \\
& = & (p'[q'/y])[t'/x]. & \text{by Prop. 1.4.1 (Nested substitution)}
\end{array}
$$

$\square$

Observe that by definition of $\gg$ we have

(i) If $\lambda x.s \gg t$ then t has the shape $\lambda x.s'$ and $s \gg s'$

(ii) If $st \gg u$ then

- *either* u has the shape $s't'$ and $s \gg s'$ and $t \gg t'$
- *or* s has the shape $\lambda x.p$ and $u \equiv p'[t'/x]$ with $p \gg p'$ and $t \gg t'$.

Proposition 2.4.3 $\gg$ *satisfies the diamond property.*

Proof Suppose $s \gg s_1$ and $s \gg s_2$. We show by structural induction and by case analysis of the definition of $s \gg s_1$ that $s_1 \gg t$ and $s_2 \gg t$, for some t.

- **(refl)** By assumption $s \gg s \equiv s_1$. Take t to be s_2.
- **(app)** By assumption $s \equiv pq$, $s_1 \equiv p_1 q_1$ with $p \gg p_1$ and $q \gg q_1$. By the preceding observation we consider two subcases:
  - $s_2 \equiv p_2 q_2$ with $p \gg p_2$ and $q \gg q_2$. By the induction hypothesis $p_1 \gg \underline{p}$ and $p_2 \gg \underline{p}$, $q_1 \gg \underline{q}$ and $q_2 \gg \underline{q}$ for some $\underline{p}$ and $\underline{q}$. Hence the result follows by taking t to be $\underline{pq}$ and by (**app**).
  - $p \equiv \lambda x.u$ and $s_2 \equiv u_2[q_2/x]$ with $u \gg u_2$ and $q \gg q_2$. Suppose $p_1 \equiv \lambda x.u_1$ with $u \gg u_1$. By the induction hypothesis we have, for some $\underline{u}$, $u_1 \gg \underline{u}$ and $u_2 \gg \underline{u}$; and for some $\underline{q}$, $q_1 \gg \underline{q}$ and $q_2 \gg \underline{q}$. Hence by $(||\text{-}\beta)$, $(\lambda x.u_1)q_1 \gg \underline{u}[\underline{q}/x]$ and, by the Substitution Lemma 2.4.2, $u_2[q_2/x] \gg \underline{u}[\underline{q}/x]$.
- **(abs)** Exercise.
- $(||\text{-}\beta)$ By assumption $s \equiv (\lambda x.p)q$ and $s_1 \equiv p_1[q_1/x]$ with $p \gg p_1$ and $q \gg q_1$. By the previous observation there are two cases:
  - $s_2 \equiv (\lambda x.p_2)q_2$ with $p \gg p_2$ and $q \gg q_2$. By the induction hypothesis, for some $\underline{p}$, p_1 and $p_2 \gg \underline{p}$, and for some $\underline{q}$, q_1 and $q_2 \gg \underline{q}$. Hence, by the Substitution Lemma, $p_1[q_1/x] \gg \underline{p}[\underline{q}/x]$, and by $(||\text{-}\beta)$, $(\lambda x.p_2)q_2 \gg \underline{p}[\underline{q}/x]$.
  - $s_2 \equiv p_2[q_2/x]$ with $p \gg p_2$ and $q \gg q_2$, same as before.

□

Finally we check that

Lemma 2.4.4 $\twoheadrightarrow_\beta$ *is the transitive closure of* $\gg$.

Proof It suffices to show that

$$\text{“reflexive closure of } \rightarrow_\beta\text{”} \quad \subseteq \quad \gg \quad \subseteq \quad \twoheadrightarrow_\beta .$$

The first inclusion is obvious; the second is easily verified by induction over the rules that define $\gg$. □

Hence we see that β is Church-Rosser.

Other forms of reduction in λ-calculus

η-reduction and *η-expansion* over untyped λ-terms are the respective compatible closures of the following notions of reduction:

$$\begin{aligned} \eta^{\text{red}} &\stackrel{\text{def}}{=} \{\langle\, \lambda x.sx, s\,\rangle : x \text{ is not free in } s\,\} \\ \eta^{\text{exp}} &\stackrel{\text{def}}{=} \{\langle\, s, \lambda x.sx\,\rangle : x \text{ is not free in } s\,\}. \end{aligned}$$

The notion of reduction $\beta\eta$ is the union of β and η^{red}.

Proposition 2.4.5 *The one-step $\beta\eta$-reduction is Church-Rosser.* □

A proof can be found e.g. in [Bar84, p. 66].

2.5 Why is the Church-Rosser property important?

Church-Rosser property is a standard test for consistency of an equational theory in a sense which we shall make clear shortly. Let $\mathcal{E}$ be a formal theory of equations over terms of $\mathcal{T}$. We say that a notion of reduction R ***implements*** $\mathcal{E}$ just when the equivalence relation $=_R$ induced by R coincides with $\mathcal{E}$.

Proposition 2.5.1 *Suppose R implements $\mathcal{E}$. If R is Church-Rosser and if there are distinct R-normal forms then $\mathcal{E}$ is* ***consistent*** *(i.e. there are distinct terms s and t that are not provably equal in $\mathcal{E}$).*

Proof Note that by definition $s =_R t$ if and only if

$$\begin{tikzcd} s \arrow[dr, two heads] & & t_1 \arrow[dl, two heads] \arrow[dr, two heads] & & \cdots & t_{n-1} \arrow[dr, two heads] & & t_n \equiv t \arrow[dl, two heads] \\ & s_1 & & s_2 & & & s_n & \end{tikzcd}$$

for some $s_1, t_1, \cdots, s_n, t_n$. We claim: if R is Church-Rosser then $s =_R t$ if and only if for some u, both s and $t \twoheadrightarrow_R u$.

Now take s and t to be two distinct R-normal forms. Suppose R is Church-Rosser. For a contradiction, suppose $s = t$ is provable in $\mathcal{E}$. Hence by R-implementability, $s =_R t$, which implies, by the claim, that both s and $t \twoheadrightarrow_R u$, for some u — a contradiction. □

Corollary 2.5.2 *Since there are distinct β-normal forms, and β is Church-Rosser, $\boldsymbol{\lambda\beta}$ is consistent.*

Proposition 2.5.3 *Suppose R implements $\mathcal{E}$. If $\to_R$ is both weakly normalizing and Church-Rosser then $\mathcal{E}$ is decidable.*

Proof Take any terms s and t. Weak normalization gives a strategy that reduces s and t to normal form in finitely many steps. Since R implements $\mathcal{E}$, $s = t$ in $\mathcal{E}$ if and only if they have the same normal form. It then remains to appeal to Church's Thesis. □

Problems

2.1 List all the subterms of $\lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$.

2.2 Give pairs of λ-terms to show that the following inclusions are strict:

$$\to_\beta \;\subset\; \to_\beta^+ \;\subset\; \twoheadrightarrow_\beta \;\subset\; =_\beta .$$

2.3 Show that Remark 2.2.3 is true in the case of β-reduction over λ-terms.

2.4 Show that the following λ-terms have a β-normal form: set $\mathbf{s} \equiv \lambda xyz.xz(yz)$

(i) $(\lambda yz.zy)((\lambda x.xxx)(\lambda x.xxx))(\lambda wx.x)$

(ii) $(\lambda y.yyy)((\lambda ab.a)(\lambda x.x)(\mathbf{ss}))$

2.5 Prove the following:

(i) If $\to_R$ is Church-Rosser then it has unique normal form property, but not *vice versa.*

(ii) If $\to_R$ is strongly normalizing and has unique normal form property then it is Church-Rosser. Is the statement still valid if $\to_R$ is only weakly normalizing?

2.6 Consider β-reduction. Give three λ-terms that are:

(i) in β-normal form

(ii) not in β-normal form but strongly normalizable

(iii) normalizable but not strongly normalizable

(iv) non-normalizable.

2.7 Prove the ***Subsitution Lemma*** for β-reduction: for any terms s, t and u, and for any variable x, if $s \twoheadrightarrow_\beta t$ then $s[u/x] \twoheadrightarrow_\beta t[u/x]$.

2.8 Define by recursion the collection of β-normal forms. Describe the collection of $\beta\eta$-normal forms.

2.9 Write out the argument for the case of **(abs)** in Lemma 2.4.2 and the same in Lemma 2.4.3.

2.10 Prove that the notion of reduction β implements the proof system $\lambda\beta$. That is to say, for any λ-terms s and t, $\lambda\beta \vdash s = t$ if and only if $s =_\beta t$ (the latter is defined w.r.t. the notion of reduction β).

2.11 We say that λ-terms s and t are ***incompatible*** just in case the formal theory obtained by augmenting $\boldsymbol{\lambda\beta}$ with $s = t$ is inconsistent. Prove by contradiction that (the usual combinators considered as λ-terms) $\mathbf{s}$ and $\mathbf{k}$ are incompatible. [Hint: By applying both sides of the equation $\mathbf{s} = \mathbf{k}$ to appropriate λ-terms p, q and r, show that $\mathbf{i} = s$ for all s.]

Show further that

(i) $\mathbf{i}$ and $\mathbf{k}$ are incompatible

(ii) $\mathbf{i}$ and $\mathbf{s}$ are incompatible

(iii) xy and yx are incompatible.

[We shall see later in the course by Böhm's Theorem that any λ-theory that equates any two $\beta\eta$-inequivalent normal forms is inconsistent.]

2.12 Let $\rightarrow_1$ and $\rightarrow_2$ be two binary relations on a set $\mathcal{T}$. Say that $\rightarrow_1$ and $\rightarrow_2$ ***commute*** just in case whenever $s \rightarrow_1 t_1$ and $s \rightarrow_2 t_2$ then there is some t such that $t_1 \rightarrow_2 t$ and $t_2 \rightarrow_1 t$.

A lemma of Hindley and Rosen states that if $\rightarrow_1$ and $\rightarrow_2$ both satisfy the diamond property and commute with each other then the reflexive transitive closure of $(\rightarrow_1 \cup \rightarrow_2)$ satisfies the diamond property.

Prove the lemma. Hence show that if $\rightarrow_1$ and $\rightarrow_2$ are Church-Rosser and $\twoheadrightarrow_1$ commutes with $\twoheadrightarrow_2$ then $(\rightarrow_1 \cup \rightarrow_2)$ is Church-Rosser.

2.13 By using the preceding lemma of Hindley and Rosen, prove that the notion of reduction $\boldsymbol{\lambda\beta\eta}^{\mathbf{red}}$ is Church-Rosser.

2.14 Add to $\mathbf{\Lambda}$ (the collection of λ-terms) constants δ and ϵ. Define on the extended terms the following notion of reduction $\boldsymbol{\delta}$:

$$\delta s s \quad \rightarrow \quad \epsilon.$$

Show that $\boldsymbol{\lambda\beta\delta}$ is not Church-Rosser.

3 Combinatory logic

The section gives a brief introduction to the theory of combinatory logic. Models of combinatory logic are called combinatory algebras which may be characterized as applicative structures that satisfy the axiom of combinatory completeness. Various extensionality axioms are introduced. A major theme is the translation between combinatory logic and the λ-calculus, and the nature of their relationship as theories.

This section assumes basic knowledge of first-order predicate logic; see e.g. the relevant chapters of [Ham88] or [Men87].

3.1 Combinatory algebra

We shall consider theories in first-order predicate calculus with equality.

Notation Fix such a language $\mathcal{L}_0$ that has a binary function symbol "$\cdot$". For terms a, b of $\mathcal{L}_0$, we write $a \cdot b$ simply as ab; and for any variable x of $\mathcal{L}_0$, $a[b/x]$ shall mean the term obtained from a by substituting b for every occurrence of x. We shall assume the same notational convention for application as before i.e. $abcd$ means $((ab)c)d$ etc. A model for $\mathcal{L}_0$ is called an ***applicative structure***.

Definition 3.1.1 Let $\mathcal{L}$ be the language obtained by adding constant symbols $\mathbf{k}, \mathbf{s}$ to $\mathcal{L}_0$. Consider the axioms

$$(\mathbf{C}_0) \qquad \begin{cases} \mathbf{k}xy &= x \\ \mathbf{s}xyz &= xz(yz). \end{cases}$$

Note that it is the same as considering the closure of $(\mathbf{C}_0)$, i.e.,

$$\begin{cases} \forall xy.\mathbf{k}xy &= x \\ \forall xyz.\mathbf{s}xyz &= xz(yz). \end{cases}$$

A model of this system of axioms is called a (total) ***combinatory algebra***.

We shall use the following notations:

- $\mathcal{M} \models F$ means the formula F is satisfied in the model $\mathcal{M}$ of $\mathcal{L}$
- $S \vdash F$ means the closed formula F is a consequence of the set S of formulae.

3.2 Abstraction algorithm

Define an operation $\mathcal{L} \longrightarrow \mathcal{L}$ parametrized by variables x of $\mathcal{L}$: for each x, there is a map $a \mapsto \lambda\!\!\lambda x.a$, where $a \in \mathcal{L}$ and where $\lambda\!\!\lambda x.a$ is defined by recursion as:

$$\begin{aligned} \lambda\!\!\lambda x.x &\stackrel{\text{def}}{=} \mathbf{skk} \\ \lambda\!\!\lambda x.a &\stackrel{\text{def}}{=} \mathbf{k}a && \text{if } x \text{ does not occur free in } a \\ \lambda\!\!\lambda x.ab &\stackrel{\text{def}}{=} \mathbf{s}(\lambda\!\!\lambda x.a)(\lambda\!\!\lambda x.b). \end{aligned}$$

Note that $\lambda\!\!\lambda x.(\text{-})$ maps $\mathcal{L}$ to $\mathcal{L}$ – not to be confused with λ-abstraction. We shall often use $\mathbf{i}$ as a shorthand for $\mathbf{skk}$. Nested abstractions $\lambda\!\!\lambda x_1.(\cdots \lambda\!\!\lambda x_n.(\text{-}))$ shall be written as $\lambda\!\!\lambda x_1 \cdots x_n.(\text{-})$.

Proposition 3.2.1 (β-simulation) *For each $a \in \mathcal{L}$,*

(i) x does not occur free in $\lambda\!\!\lambda x.a$, and

(ii) $\mathbf{C}_0 \vdash \forall x.((\lambda\!\!\lambda x.a)x = a)$.

(iii) Hence it follows that $\mathbf{C}_0 \vdash (\lambda\!\!\lambda x.a)b = a[b/x]$ for all $b \in \mathcal{L}$.

(iv) (ii) extends to

$$\mathbf{C}_0 \vdash \forall x_1 \cdots x_n.((\lambda\!\!\lambda x_1 \cdots x_n.a)x_1 \cdots x_n \quad = \quad a)$$

with $\mathsf{fv}(a) \subseteq \{ x_1, \cdots, x_n \}$.

Proof (i) can be proved easily by an induction on the size of a. (ii) is proved by structural induction on a. Suppose $D \vdash \mathbf{C}_0$. Fix $a \in \mathcal{L}$. Take any $d \in D$. We write $[\![t]\!]_\rho$ for the interpretation of t in D given the variable valuation ρ. Now take ρ to be a valuation that maps x to d. We aim to prove $([\![\lambda\!\!\lambda x.a]\!]_\rho)d = [\![a]\!]_\rho$. Suppose $a \equiv x$. Then $l.h.s. = [\![\mathbf{i}]\!]_\rho d = d = r.h.s.$ Suppose $x \notin a$. Then $l.h.s. = [\![(\lambda\!\!\lambda x.a)]\!]_\rho d = [\![\mathbf{k}a]\!]_\rho d = [\![a]\!]_\rho = r.h.s.$ Finally suppose $a \equiv a_1 a_2$.

$$\begin{aligned}
[\![\lambda\!\!\lambda x.a_1 a_2]\!]_\rho d &= [\![\mathbf{s}(\lambda\!\!\lambda x.a_1)(\lambda\!\!\lambda x.a_2)]\!]_\rho d \\
&= [\![\lambda\!\!\lambda x.a_1]\!]_\rho d [\![\lambda\!\!\lambda x.a_2]\!]_\rho d \qquad \text{by the induction hypothesis} \\
&= [\![a_1]\!]_\rho [\![a_2]\!]_\rho = [\![a_1 a_2]\!]_\rho.
\end{aligned}$$

(iii) is an immediate consequence of (ii). We prove (iv) by induction on n. The base case is (ii). For the inductive case of $n = r+1$, note that by (i) variables occurring free in $\lambda\!\!\lambda x_2 \cdots x_{r+1}.a$ are contained in $\{ x_1 \}$. Hence by (ii) $\mathbf{C}_0 \vdash (\lambda\!\!\lambda x_1.(\lambda\!\!\lambda x_2 \cdots x_{r+1}.a))x_1 = \lambda\!\!\lambda x_2 \cdots x_{r+1}.a$; and so

$$\mathbf{C}_0 \vdash (\lambda\!\!\lambda x_1.(\lambda\!\!\lambda x_2 \cdots x_{r+1}.a))x_1 x_2 \cdots x_{r+1} = (\lambda\!\!\lambda x_2 \cdots x_{r+1}.a)x_2 \cdots x_{r+1}.$$

Hence by the induction hypothesis $\mathbf{C}_0 \vdash (\lambda\!\!\lambda x_1 \cdots x_{r+1}.a)x_1 \cdots x_{r+1} = a$.

□

Proposition 3.2.2 *All non-trivial combinatory algebras are infinite.*

Proof Fix n. Suppose there is a combinatory algebra A of size n. For each natural number i where $1 \leqslant i \leqslant n+1$, define a_i to be $\lambda\!\!\lambda x_1 \cdots x_{n+1}.x_i$. Then, w.l.o.g., say, $A \vDash a_1 = a_2$. So for any *distinct* b_1 and b_2 of A, we have $[\![a_1]\!] b_1 b_2 \cdots b_2 = [\![a_2]\!] b_1 b_2 \cdots b_2$, and so $b_1 = b_2$ – a contradiction. □

3.3 Combinatory completeness

An applicative structure A is said to be ***combinatory complete*** if for every term t of $\mathcal{L}_0$ with all free variables of t occurring in $\{ x_1, \cdots, x_n \}$ and constant parameters from A, there is an element f in A such that

$$\textbf{(cc)} \qquad A \quad \vDash \quad f x_1 \cdots x_n = t.$$

(This means that $f a_1 \cdots a_n = t[a_1/x_1, \cdots, a_n/x_n]$ for all $a_1, \cdots, a_n \in A$.) We say that f ***represents*** t. One may think of t as a polynomial over the set $\{ x_1, \cdots, x_n \}$ of variables and constant symbols from A.

Proposition 3.3.1 (Characterization) *An applicative structure A is combinatory complete if and only if A can be given the structure of a combinatory algebra.*

Proof "$\Leftarrow$": Given a term t of the required kind, take f to be $\lambda\!\!\lambda x_1 \cdots x_n.t$ where $\mathsf{fv}(t) \subseteq \{x_1, \cdots, x_n\}$. The result follows by an appeal to Proposition 3.2.1(iv). "$\Rightarrow$": Take $\mathbf{s}$ to be the element of A representing $x_1x_3(x_2x_3)$ with $n = 3$, and $\mathbf{k}$ to be the element of A representing x_1 with $n = 2$. $\square$

Note that (**cc**) is equivalent to the following axioms:

$$\begin{cases} \exists k.\forall xy.kxy = x \\ \exists s.\forall xyz.sxyz = xz(yz). \end{cases}$$

Let $\mathbf{e}$ denote the term $\lambda\!\!\lambda x.(\lambda\!\!\lambda y.xy)$. By Proposition 3.2.1(iv), we have $\mathbf{C}_0 \vdash \mathbf{e}xy = xy$.

Lemma 3.3.2 *Let t be an $\mathcal{L}$-term and suppose x does not occur free in t. Then*

$$\mathbf{C}_0 \quad \vdash \quad \lambda\!\!\lambda x.tx = \mathbf{e}t = \mathbf{s}(\mathbf{k}t)\mathbf{i}.$$

Proof Any combinatory algebra validates $\mathbf{e}t = \lambda\!\!\lambda y.ty = \mathbf{s}(\lambda\!\!\lambda y.t)\mathbf{i} = \mathbf{s}(\mathbf{k}t)\mathbf{i}$. $\square$

Now consider the axioms:

$$(\mathbf{C}_1) \qquad \begin{cases} \mathbf{k} = \lambda\!\!\lambda xy.\mathbf{k}xy \\ \mathbf{s} = \lambda\!\!\lambda xyz.\mathbf{s}xyz. \end{cases}$$

The following are consequences of $\mathbf{C}_0 + \mathbf{C}_1$.

$$\begin{aligned} \mathbf{k}x &= \lambda\!\!\lambda y.\mathbf{k}xy \\ \mathbf{s}xy &= \lambda\!\!\lambda z.\mathbf{s}xyz; \end{aligned}$$

and hence, by Lemma 3.3.2, so are

$$(\mathbf{C}_1^0) \qquad \begin{cases} \mathbf{e}(\mathbf{k}x) = \mathbf{k}x \\ \mathbf{e}(\mathbf{s}xy) = \mathbf{s}xy. \end{cases}$$

To summarize we have shown:

Lemma 3.3.3 $\mathbf{C}_0 + \mathbf{C}_1 \vdash \mathbf{C}_1^0$. $\square$

Proposition 3.3.4 (e-invariance) *The following are consequences of $\mathbf{C}_0 + \mathbf{C}_1$.*

(i) $\lambda\!\!\lambda x.t = \mathbf{e}(\lambda\!\!\lambda x.t) = \lambda\!\!\lambda x.(\lambda\!\!\lambda x.t)x$ *i.e. "$\mathbf{e}$ fixes any $\lambda\!\!\lambda$-abstraction"*

(ii) $\mathbf{ee} = \mathbf{e}; \mathbf{e}(\mathbf{e}x) = \mathbf{e}x$.

Proof (i) The second identity follows from Lemma 3.3.2 as x does not occur free in $(\lambda\!\!\lambda x.t)$. Observe that

$$\lambda\!\!\lambda x.t \quad = \quad \begin{cases} \mathbf{skk} & \text{or} \\ \mathbf{k}t & \text{or} \\ \mathbf{s}uv & \end{cases}$$

depending on the shape of t. Hence, by $\mathbf{C}_1^0$, $\mathbf{e}(\lambda\!\!\lambda x.t) = (\lambda\!\!\lambda x.t)$.

(ii) The first equation follows from (i) since $\mathbf{e} = \lambda\!\!\lambda x.(\lambda\!\!\lambda y.xy)$. By Lemma 3.3.2, $\mathbf{e}x = \lambda\!\!\lambda y.xy$; the second equation then follows from (i). □

3.4 Extensionality axioms

Weak extensionality scheme: for all $\mathcal{L}$-terms t, u

$$(\mathbf{WExt}) \qquad (\forall x.t = u) \to \lambda\!\!\lambda x.t = \lambda\!\!\lambda x.u.$$

By induction we get as a consequence the scheme

$$(\forall x_1 \cdots x_n.t = u) \to \lambda\!\!\lambda x_1 \cdots x_n.t = \lambda\!\!\lambda x_1 \cdots x_n.u.$$

Weak extensionality axiom (which is a formula):

$$(\mathbf{WExt}') \qquad \forall yz.[(\forall x.yx = zx) \to \mathbf{e}y = \mathbf{e}z].$$

Extensionality axiom is the formula:

$$(\mathbf{Ext}) \qquad \forall yz.\{\forall x[yx = zx] \to y = z\}.$$

The last axiom says that two elements are equal if and only if they are equal applicatively (or extensionally), i.e., they have the same behaviour as functions.

Proposition 3.4.1 **WExt** *and* $\mathbf{WExt}'$ *are equivalent modulo* $\mathbf{C}_0 + \mathbf{C}_1$ *(in fact the weaker axiom* $\mathbf{C}_0 + \mathbf{C}_1^0$ *suffices).* □

Definition 3.4.2 We denote by **CL** (***combinatory logic***) the system of axioms

$$\mathbf{CL} \quad \stackrel{\text{def}}{=} \quad \mathbf{C}_0 + \mathbf{C}_1 + \mathbf{WExt}$$

or equivalently $\mathbf{C}_0 + \mathbf{C}_1 + \mathbf{WExt}'$; and by **ECL** (***extensional combinatory logic***) the system of axioms

$$\mathbf{ECL} \quad \stackrel{\text{def}}{=} \quad \mathbf{C}_0 + \mathbf{Ext}$$

Note that in the literature ***combinatory logic*** is often the name associated with the weaker formal system $\mathbf{C}_0$, rather than **CL**.

3.5 Translation between $\lambda\beta$ and CL

λ-calculus and combinatory logic are very closely related. As formal theories, they are almost, but not quite, equivalent. The nature of their relationship deserves careful study. There are very natural translations between the two systems. A major question we shall investigate is the extent to which each translation preserves the theory.

First we assume that variables of the first-order language $\mathcal{L}$ coincide with variables of the λ-calculus. Define maps between λ-terms and combinatory logic terms

$$\boldsymbol{\Lambda} \underset{(\text{-})_{\lambda}}{\overset{(\text{-})_{\mathrm{cl}}}{\rightleftarrows}} \mathcal{L}$$

where $t \mapsto t_{\mathrm{cl}}$ is defined by recursion as follows:

$$\begin{cases} x_{\mathrm{cl}} & \stackrel{\mathrm{def}}{=} & x \\ (tu)_{\mathrm{cl}} & \stackrel{\mathrm{def}}{=} & t_{\mathrm{cl}} u_{\mathrm{cl}} \\ (\lambda x.t)_{\mathrm{cl}} & \stackrel{\mathrm{def}}{=} & \lambda\!\!\lambda x.(t_{\mathrm{cl}}) \end{cases}$$

and $a \mapsto a_\lambda$ by

$$\begin{cases} x_\lambda & \stackrel{\mathrm{def}}{=} & x \\ (ab)_\lambda & \stackrel{\mathrm{def}}{=} & a_\lambda b_\lambda \\ \mathbf{s}_\lambda & \stackrel{\mathrm{def}}{=} & \lambda xyz.xz(yz) \\ \mathbf{k}_\lambda & \stackrel{\mathrm{def}}{=} & \lambda xy.x. \end{cases}$$

Lemma 3.5.1 *For any terms a, b of $\mathcal{L}$,*

(i) if $\mathbf{CL} \vdash a = b$ *then* $\boldsymbol{\lambda\beta} \vdash a_\lambda = b_\lambda$

(ii) if $\mathbf{ECL} \vdash a = b$ *then* $\boldsymbol{\lambda\beta\eta} \vdash a_\lambda = b_\lambda$. □

Lemma 3.5.2 *(i) For every λ-term t,* $\boldsymbol{\lambda\beta} \vdash (t_{\mathrm{cl}})_\lambda = t$.

(ii) For every $\mathcal{L}$-term a, $\mathbf{CL} \vdash (a_\lambda)_{\mathrm{cl}} = a$.

Proof For (i) we prove by induction over the structure of t. We shall consider only the hardest case of $t \equiv \lambda x.u$. Then t_{cl} is $\lambda\!\!\lambda x.u_{\mathrm{cl}}$. By Proposition 3.2.1(iii), we have $\mathbf{CL} \vdash t_{\mathrm{cl}} x = u_{\mathrm{cl}}$. Thus by Lemma 3.5.1 $(t_{\mathrm{cl}})_\lambda x = (u_{\mathrm{cl}})_\lambda$. By induction hypothesis $(u_{\mathrm{cl}})_\lambda = u$. Hence $(t_{\mathrm{cl}})_\lambda x = u$, and so,

$$\lambda x.(t_{\mathrm{cl}})_\lambda x \quad = \quad \lambda x.u \quad = \quad t.$$

Now since $t_{\mathrm{cl}} \equiv \lambda\!\!\lambda x.u_{\mathrm{cl}}$, we have $\mathbf{CL} \vdash \mathbf{e} t_{\mathrm{cl}} = t_{\mathrm{cl}}$, by Proposition 3.3.4(i). Thus $(\mathbf{e}(t_{\mathrm{cl}}))_\lambda = (t_{\mathrm{cl}})_\lambda$. But by Proposition 3.3.4(i), $(\mathbf{e}(t_{\mathrm{cl}}))_\lambda = \lambda\!\!\lambda x.t_{\mathrm{cl}} x_\lambda = \mathbf{s}(\mathbf{k} t_{\mathrm{cl}})\mathbf{i}_\lambda = \mathbf{s}_\lambda(\mathbf{k}_\lambda t_{\mathrm{cl}\lambda})\mathbf{i}_\lambda = \lambda x.t_{\mathrm{cl}\lambda} x$; hence $(t_{\mathrm{cl}})_\lambda = \lambda x.(t_{\mathrm{cl}})_\lambda x = t$.

Next we prove (ii) by induction on the size of a. The base case of a being a variable is obvious. The inductive case of a being an application is easily checked. Suppose a is $\mathbf{s}$. We have $(\mathbf{C}_0)$: $\mathbf{s}xyz = xz(yz)$ for any x, y, z. By $(\mathbf{WExt})$ we have

$$\lambda\!\!\lambda xyz.\mathbf{s}xyz \quad = \quad \lambda\!\!\lambda xyz.xz(yz).$$

By $(\mathbf{C}_1)$, $\mathbf{s} = \lambda\!\!\lambda xyz.xz(yz)$, and so, by definition of $(\text{-})_\lambda$ and $(\text{-})_{\mathrm{cl}}$, $\mathbf{s} = (\mathbf{s}_\lambda)_{\mathrm{cl}}$. □

A main result of this section is that the encoding $(\text{-})_{\text{cl}} : \mathbf{\Lambda} \longrightarrow \mathcal{L}$ preserves equations in $\boldsymbol{\lambda\beta}$ (in the sense of Theorem 3.5.4). To prove it we need a substitution lemma.

Lemma 3.5.3 *For $u, t \in \mathbf{\Lambda}$, $\mathbf{CL} \vdash (u[t/x])_{\text{cl}} = u_{\text{cl}}[t_{\text{cl}}/x]$.*

Proof By induction on the size of u. The cases of u being a variable and application are immediate. We only consider the case of $u \equiv \lambda y.v$.

CLAIM: $\mathbf{CL} \vdash (u_{\text{cl}}[t_{\text{cl}}/x])y = (u[t/x])_{\text{cl}}y$.

$$\begin{array}{rcll} (u[t/x])_{\text{cl}}y & \equiv & (\lambda y.v[t/x])_{\text{cl}}y & \text{by definition of } (\text{-})_{\text{cl}} \\ & = & (\lambda\!\!\lambda y.(v[t/x])_{\text{cl}})y & \text{by Proposition 3.2.1} \\ & = & (v[t/x])_{\text{cl}} & \text{by induction hypothesis} \\ & = & v_{\text{cl}}[t_{\text{cl}}/x] & \\ & = & (u_{\text{cl}}y)[t_{\text{cl}}/x] & \\ & = & (u_{\text{cl}}[t_{\text{cl}}/x])y. & \end{array}$$

But by definition of $(\text{-})_{\text{cl}}$

$$\begin{array}{rcll} u_{\text{cl}}y & = & (\lambda\!\!\lambda y.v_{\text{cl}})y & \text{by Proposition 3.2.1} \\ & = & v_{\text{cl}}. & \end{array}$$

Hence the claim is proved.

By $(\mathbf{WExt}')$ from Claim, we have

$$\mathbf{CL} \quad \vdash \quad \mathbf{e}(u_{\text{cl}}[t_{\text{cl}}/x]) \quad = \quad \mathbf{e}(u[t/x])_{\text{cl}} \tag{1}$$

Now $u_{\text{cl}} \equiv \lambda\!\!\lambda y.(v_{\text{cl}})$ and $(u[t/x])_{\text{cl}} \equiv \lambda\!\!\lambda y.(v[t/x])_{\text{cl}}$. Hence by Proposition 3.3.4(i)

$$\begin{array}{lll} \mathbf{CL} & \vdash & \mathbf{e}u_{\text{cl}} = u_{\text{cl}}, \quad \text{and} \qquad (2) \\ \mathbf{CL} & \vdash & \mathbf{e}(u[t/x])_{\text{cl}} = (u[t/x])_{\text{cl}}. \qquad (3) \end{array}$$

From (2), $\mathbf{CL} \vdash u_{\text{cl}}[t_{\text{cl}}/x] = \mathbf{e}(u_{\text{cl}}[t_{\text{cl}}/x])$. Therefore, from (1) and (3), we get

$$\mathbf{CL} \quad \vdash \quad u_{\text{cl}}[t_{\text{cl}}/x] = (u[t/x])_{\text{cl}}.$$

□

Theorem 3.5.4 (Equivalence) *Let t and u be λ-terms. Then*

(i) $\boldsymbol{\lambda\beta} \vdash t = u$ *if and only if* $\mathbf{CL} \vdash t_{\text{cl}} = u_{\text{cl}}$

(ii) $\boldsymbol{\lambda\beta\eta} \vdash t = u$ *if and only if* $\mathbf{ECL} \vdash t_{\text{cl}} = u_{\text{cl}}$.

Proof We shall just prove (i), and leave (ii) as an exercise. "$\Leftarrow$": If $\mathbf{CL} \vdash t_{\mathrm{cl}} = u_{\mathrm{cl}}$ then $(t_{\mathrm{cl}})_\lambda = (u_{\mathrm{cl}})_\lambda$. Thus, by Lemma 3.5.2, $t = u$.

"$\Rightarrow$": It suffices to prove it for the case of $t \to_\beta u$ (one-step β-reduction). We then proceed by induction of the size of t. Clearly t cannot be a variable. There are two cases. Suppose $t \equiv \lambda x.t'$. Then $u \equiv \lambda x.u'$ and $t' \to_\beta u'$. By the induction hypothesis, $\mathbf{CL} \vdash t'_{\mathrm{cl}} = u'_{\mathrm{cl}}$. By **WExt**, $\mathbf{CL} \vdash \lambda\!\!\lambda x.(t'_{\mathrm{cl}}) = \lambda\!\!\lambda x.(u'_{\mathrm{cl}})$. Hence $\mathbf{CL} \vdash (\lambda x.t')_{\mathrm{cl}} = (\lambda x.u')_{\mathrm{cl}}$.

Now suppose $t \equiv pq$. There are three subcases.

- $u \equiv p'q$ and $p \to_\beta p'$: by the induction hypothesis, $\mathbf{CL} \vdash p_{\mathrm{cl}} = p'_{\mathrm{cl}}$, and so, $\mathbf{CL} \vdash p_{\mathrm{cl}}q_{\mathrm{cl}} = p'_{\mathrm{cl}}q_{\mathrm{cl}}$. Hence $\mathbf{CL} \vdash t_{\mathrm{cl}} = u_{\mathrm{cl}}$.
- $u \equiv pq'$ and $q \to_\beta q'$: similar to the previous case.
- $t \equiv (\lambda x.v)w$ and $u \equiv v[w/x]$: by Lemma 3.5.3, $\mathbf{CL} \vdash u_{\mathrm{cl}} = v_{\mathrm{cl}}[w_{\mathrm{cl}}/x]$; on the other hand, $t_{\mathrm{cl}} \stackrel{\mathrm{def}}{=} (\lambda\!\!\lambda x.v_{\mathrm{cl}})w_{\mathrm{cl}}$. Hence by Proposition 3.2.1, $\mathbf{CL} \vdash t_{\mathrm{cl}} = u_{\mathrm{cl}}$.

$\square$

Problems

3.1 Show that $\lambda\!\!\lambda xy.yx \equiv \mathbf{s}(\mathbf{k}(\mathbf{si}))(\mathbf{s}(\mathbf{kk})\mathbf{i})$. What is $\lambda\!\!\lambda xy.xy$?

3.2 Basis. Let $\mathcal{L}$ be a collection of λ-terms. The set $\mathcal{L}^+$ of terms generated by $\mathcal{L}$ is the least set $\mathcal{P}$ such that

- $\mathcal{L} \subseteq \mathcal{P}$
- if $s, t \in \mathcal{P}$ then $st \in \mathcal{P}$.

Let $\mathcal{Q} \subseteq \mathbf{\Lambda}$. $\mathcal{L} \subseteq \mathbf{\Lambda}$ is said to be a ***basis for*** $\mathcal{Q}$ just in case for every $q \in \mathcal{Q}$, there is some $t_q \in \mathcal{L}^+$ such that $\lambda\beta \vdash q = t_q$. $\mathcal{L}$ is a ***basis*** if $\mathcal{L}$ is a basis for $\mathbf{\Lambda}^o$ (the set of closed λ-terms).

Prove that $\{\mathbf{k}, \mathbf{s}\}$ is a basis. [Hint: Use Lemma 3.5.2]

3.3 Show that $\theta \equiv \lambda x.x\mathbf{ksk}$ is a singleton basis. [Hint: Calculate $\theta\theta\theta$ and $\theta(\theta\theta)$.]

3.4 (i) (Barendregt) Let $X \equiv \lambda x.x(x\mathbf{s}(\mathbf{kk}))\mathbf{k}$. Show that $\{X\}$ is a basis. [Hint: calculate XXX and $X\mathbf{k}$.]

(ii) (Rosser) Find a closed λ-term J such that $JJ = \mathbf{s}$ and $J\mathbf{s} = \mathbf{k}$.

3.5 Prove Proposition 3.7.

3.6 Prove Lemma 3.5.1.

3.7 Show that the set of closed λ-terms quotiented by β-equivalence is a model of **CL**. Hence or otherwise prove Proposition : **WExt** and **WExt**$'$ are equivalent modulo $\mathbf{C}_0 + \mathbf{C}_1$ (in fact the weaker axiom $\mathbf{C}_0 + \mathbf{C}_1^0$ suffices).

3.8 The ***weak combinatory logic*** notion of reduction is given by the union of the following binary relations (defined schematically): p, q and r range over combinatory logic terms

$$\langle \mathbf{k}pq, \quad p\rangle$$

$$\langle \mathbf{s}pqr, \quad pr(qr)\rangle$$

Show that the corresponding one-step weak reduction $\to_w$ is Church-Rosser.

[Hint (Rosser 1935): Define $s > t$ just in case there are disjoint weak-redexes $\Delta_1, \cdots, \Delta_n$ in s, and t is obtained by contracting them. For example $\mathbf{s}(\mathbf{k}pq)(\mathbf{s}pqr) > \mathbf{s}p(pr(qr))$. Show that

(i) $>$ satisfies the diamond property

(ii) $\twoheadrightarrow_w$ is the transitive closure of $>$.]

4 Böhm's Theorem

Böhm's theorem was proved in the late '60s and remains possibly the most significant discovery in the syntax of untyped λ-calculus. It gives rise to a powerful technique for obtaining separability results.

4.1 The theorem and its significance

Theorem 4.1.1 (Böhm) *Let s and t be closed normal λ-terms that are not $\beta\eta$-equivalent. Then there exist closed terms $u_1, \cdots, u_k$ such that*

$$\begin{cases} s\vec{u} &= \mathbf{f} \\ t\vec{u} &= \mathbf{t}. \end{cases}$$

where $\mathbf{t} \equiv \lambda xy.x$ and $\mathbf{f} \equiv \lambda xy.y$. □

Exercise 4.1.2 Show that $\mathbf{t}$ and $\mathbf{f}$ of the theorem can be replaced by any pair of closed β-normal forms that are not $\beta\eta$-equivalent.

Böhm's theorem is a classic result in the syntax of untyped λ-calculus. It is a powerful separability result.

An aside on λ-theories

A λ-theory is a consistent extension of $\lambda\beta$ that is closed under provability. A (closed) *equation* is a formula of the form $s = t$ where s and t are closed λ-terms. If $\mathcal{T}$ is a set of closed equations, then the theory $\lambda\beta + \mathcal{T}$ is obtained from $\lambda\beta$ by augmenting the axioms by $\mathcal{T}$.

Definition 4.1.3 Let $\mathcal{T}$ be a set of closed equations. $\mathcal{T}^+$ is the set of closed equations provable in $\lambda\beta + \mathcal{T}$. We say that $\mathcal{T}$ is a ***λ-theory*** just in case $\mathcal{T} = \mathcal{T}^+$ and $\mathcal{T}$ is *consistent* (i.e. there are terms s and t such that $s = t$ is not provable in $\mathcal{T}$).

Corollary 4.1.4 *Any λ-theory which identifies any two closed normal λ-terms that are not $\beta\eta$-equivalent is inconsistent.* □

Proof Take any λ-terms A and B. Write

$$D \quad \equiv \quad \lambda xyz.zyx.$$

Then we have

$$\begin{aligned} DAB\mathbf{f} &= A \\ DAB\mathbf{t} &= B. \end{aligned}$$

Hence if $\mathcal{L} \vdash s = t$ where s and t are any closed normal λ-terms that are not $\beta\eta$-equivalent, then for the $\vec{u}$ given by the theorem, we have $\mathcal{L} \vdash DAB(s\vec{u}) = DAB(t\vec{u})$, and so,

$$\mathcal{L} \vdash A = B.$$

□

The so-called "Böhm-out technique" is crucial to the proof of most ***local structure characterization theorems*** of λ-models.

4.2 Proof of the theorem

First some notations. The ***permutator of order*** n is defined to be the following term

$$\alpha_n \stackrel{\text{def}}{=} \lambda x_1 \cdots x_n x.xx_1 \cdots x_n.$$

Definition 4.2.1 We shall call ***Böhm transformation*** any function from $\mathbf{\Lambda}$ (the collection of λ-terms) to $\mathbf{\Lambda}$ defined by composing basic functions of the form $t \mapsto tu_0$ or $t \mapsto t[u_0/x]$ where u_0 and x are a given term and variable respectively.

We shall denote the functions as follows:

$$\begin{aligned} \mathbf{B}_{u_0} &: t \mapsto tu_0 \\ \mathbf{B}_{u_0,x} &: t \mapsto t[u_0/x]. \end{aligned}$$

Lemma 4.2.2 *For every Böhm transformation B, there are terms $u_1, \cdots, u_k$ such that $Bs = su_1 \cdots u_k$ for every closed term s.* □

Exercise 4.2.3 Prove the lemma.

Lemma 4.2.4 *Let s, t be two λ-terms. If one of the following*

$$\begin{array}{llll} (1) & s & \equiv xs_1 \cdots s_p & \\ & t & \equiv yt_1 \cdots t_q & \text{where } x \neq y \text{ or } p \neq q \\ (2) & s & \equiv \lambda x_1 \cdots x_m x.xs_1 \cdots s_p & \\ & t & \equiv \lambda x_1 \cdots x_n x.xt_1 \cdots t_q & \text{where } m \neq n \text{ or } p \neq q \end{array}$$

holds then

$$\begin{cases} Bs = \mathbf{f} \\ Bt = \mathbf{t} \end{cases}$$

for some Böhm transformation B.

Proof **Case (1)**:

(i) $x \neq y$, take $\sigma \equiv \lambda z_1 \cdots z_p.\mathbf{f}$ and $\tau \equiv \lambda z_1 \cdots z_q.\mathbf{t}$. Take B to be $\mathbf{B}_{\sigma,x} \circ \mathbf{B}_{\tau,y}$. Then

$$\begin{aligned} Bs &= \mathbf{f} \\ Bt &= \mathbf{t}. \end{aligned}$$

(ii) $x = y$ and $p < q$. Then

$$\begin{aligned} \mathbf{B}_{\alpha_q,x} s &= \alpha_q s_1^* \cdots s_p^* &= \lambda z_{p+1} \cdots z_q z.zs_1^* \cdots s_p^* z_{p+1} \cdots z_q \\ \mathbf{B}_{\alpha_q,x} t &= \alpha_q t_1^* \cdots t_q^* &= \lambda z.zt_1^* \cdots t_q^* \end{aligned}$$

where (-)* means (-)$[\alpha_q/x]$. This is case (2)(i).

Case (2):

(i) $m \neq n$, say $m < n$; take distinct variables $z_1, \cdots, z_n, z$ not occurring in s, t. Let

$$B \stackrel{\text{def}}{=} \mathbf{B}_z \circ \mathbf{B}_{z_n} \circ \cdots \circ \mathbf{B}_{z_1}.$$

Then

$$Bs = z_{m+1} s_1^* \cdots s_p^* z_{m+2} \cdots z_n z$$

where $(\text{-})^*$ is $(\text{-})[z_1/x_1, \cdots, z_m/x_m, z_{m+1}/x]$, and

$$Bt = z t_1^\dagger \cdots t_q^\dagger$$

where $(\text{-})^\dagger$ is $(\text{-})[z_1/x_1, \cdots, z_n/x_n, z/x]$. This is just case (1)(i).

(ii) $m = n$ and $p \neq q$; let $B \stackrel{\text{def}}{=} \mathbf{B}_x \circ \mathbf{B}_{x_m} \circ \cdots \circ \mathbf{B}_{x_1}$. We have

$$\begin{aligned} Bs &= x s_1 \cdots s_p \\ Bt &= x t_1 \cdots t_q \end{aligned}$$

This is just case (1)(ii).

Note: cases $(2)(ii) \longrightarrow (1)(ii) \longrightarrow (2)(i) \longrightarrow (1)(i)$. $\square$

Theorem 4.2.5 *Let s and t be non-$\beta\eta$-equivalent normal λ-terms, and $x_1, \cdots, x_k$ any distinct variables. Then for any $n_1, \cdots n_k$, provided they are large enough, there is a Böhm transformation B such that*

$$\begin{cases} B(s[\alpha_{n_1}/x_1, \cdots, \alpha_{n_k}/x_k]) = \mathbf{f} \\ B(t[\alpha_{n_1}/x_1, \cdots, \alpha_{n_k}/x_k]) = \mathbf{t}. \end{cases}$$

Proof The size $\mathsf{size}(s)$ of a term s is defined by recursion as follows:

$$\begin{aligned} \mathsf{size}(x) &\stackrel{\text{def}}{=} 1 \\ \mathsf{size}(st) &\stackrel{\text{def}}{=} \mathsf{size}(s) + \mathsf{size}(t) \\ \mathsf{size}(\lambda x.s) &\stackrel{\text{def}}{=} \mathsf{size}(s) + 2. \end{aligned}$$

We prove by induction on $\mathsf{size}(s) + \mathsf{size}(t)$.

Case analysis:

(1) s and t are both abstractions

(2) only one of s and t is an abstraction

(3) both are not abstractions.

Claim: It suffices to consider the last case.

Proof of Claim Take $y \neq x_1, \cdots, x_k$ with no occurrence in s and t, and let w_s and w_t be the normal form of sy and ty respectively. Now w_s is not $\beta\eta$-equivalent to w_t (why?). Suppose case (1), say $s \equiv \lambda x.u$ and $t \equiv \lambda x'.v$ then $w_s \equiv u[y/x]$, $w_t \equiv v[y/x']$ and

$$\mathsf{size}(w_s) + \mathsf{size}(w_t) \quad = \quad \mathsf{size}(s) + \mathsf{size}(t) - 4.$$

Suppose case (2), say, $s \equiv \lambda x.u$ and t is not an abstraction, then either t is a variable or $v_1 v_2$. Thus $w_s \equiv u[y/x]$ and $w_t \equiv ty$ and

$$\mathsf{size}(w_s) + \mathsf{size}(w_t) \quad = \quad \mathsf{size}(s) + \mathsf{size}(t) - 1.$$

Hence, in both cases, we can apply the induction hypothesis to w_s and w_t. Suppose for any $n_1, \cdots, n_k$ there exists B such that

$$\begin{cases} B(w_s[\alpha_{n_1}/x_1, \cdots, \alpha_{n_k}/x_k]) & = \quad \mathbf{f} \\ B(w_t[\alpha_{n_1}/x_1, \cdots, \alpha_{n_k}/x_k]) & = \quad \mathbf{t}. \end{cases}$$

Take the Böhm transformation $B \circ \mathbf{B}_y$ which works for s and t. □

We shall consider the case where both s and t are not abstractions, say

$$\begin{aligned} s &\equiv x s_1 \cdots s_p \\ t &\equiv y t_1 \cdots t_q \end{aligned}$$

where s_i, t_j are all normal forms.

Fix distinct numbers $n_1, \cdots, n_k$ and variables $x_1, \cdots, x_k$. We write

$$(\text{-})^* \quad \text{for} \quad (\text{-})[\alpha_{n_1}/x_1, \cdots, \alpha_{n_k}/x_k].$$

There are three subcases:

Case (i): $x, y \notin \{ x_1, \cdots, x_k \}$.

We have

$$\begin{aligned} s^* &\equiv x s_1^* \cdots s_p^* \\ t^* &\equiv y t_1^* \cdots t_q^*. \end{aligned}$$

If $x \neq y$ or $p \neq q$ then result follows from Lemma 4.2.4. If $x = y$ and $p = q$ take any number $n > p, n_1, \cdots, n_k$. Then take $B = \mathbf{B}_z \circ \mathbf{B}_{z_n} \circ \cdots \circ \mathbf{B}_{z_{p+1}} \circ \mathbf{B}_{\alpha_n, x}$. We have

$$\begin{aligned} Bs^* &= z s_1^\dagger \cdots s_p^\dagger z_{p+1} \cdots z_n \\ Bt^* &= z t_1^\dagger \cdots t_p^\dagger z_{p+1} \cdots z_n \end{aligned}$$

where $(\text{-})^\dagger$ is $(\text{-})[\alpha_{n_1}/x_1, \cdots, \alpha_{n_k}/x_k, \alpha_n/x]$.

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 $\mathbf{\Omega} \equiv (\lambda x.xx)(\lambda x.xx)$ be the standard unsolvable term. Then $\lambda x.\mathbf{\Omega} = \mathbf{\Omega}$ in the standard theory, since $\lambda x.\mathbf{\Omega}$ is also unsolvable; but $\lambda x.\mathbf{\Omega}$ is in weak head normal form, hence should be distinguished from $\mathbf{\Omega}$ 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:

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

Notation It is useful to fix some shorthand.

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

For example, $\mathbf{i}(\mathbf{ii}) \Downarrow \mathbf{i}$ and $\mathbf{k}(\mathbf{ii}) \Downarrow \lambda y.\mathbf{ii}$, and $\mathbf{\Omega}\Uparrow$. 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:

Stage 1 of experiment: $s \Downarrow \lambda x.p_1$;

environment "consumes" λ,

produces t_1 as input

Stage 2 of experiment: $p_1[t_1/x] \Downarrow \ldots$ $\quad s \not\Downarrow \Rightarrow s' \not\Downarrow$ no

$\vdots$ $\quad \underset{s}{(\lambda x. yx)} \precsim_0 \underset{s'}{(\lambda x.x)}$

Definition 5.3.1 We define a family of binary relations $\precsim_k$ ($k \in \omega$) over $\mathbf{\Lambda}^o$ as follows:

$(\lambda x.x) \precsim_0 (\lambda x y . \cancel{x})$ | $\Omega\Omega \precsim_1 y$ OK

- for any s and s', $s \precsim_0 s'$.
- $s \precsim_{k+1} s'$ provided $\forall \lambda x.p.[s \Downarrow \lambda x.p \implies \exists \lambda x.p'.[s' \Downarrow \lambda x.p' \;\;\&\;\; \forall r \in \mathbf{\Lambda}^o.p[r/x] \precsim_k p'[r/x]\,]]$.

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

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

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 $\mathbf{\Omega} \precsim x$ and $\mathbf{\Omega} x \precsim x$.

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

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

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 $\mathbf{\Lambda}$ 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. $\precsim$.

$\lambda xy.x$

(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 $n \leqslant m$. $\lambda x.x \not\precsim \lambda xy.x$

$\precsim_0$ ✓ $(\lambda x.x)(\Delta\Delta) \to \Delta\Delta$ ↺ $(\lambda xy.x)\Delta\Delta \to y$

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

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

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.* $\square$

We give a useful characterization of $\precsim$.

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

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 $r \in \mathbf{\Lambda}^o$, for any $n \geqslant 0$,*

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

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

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 $\precsim_{n+1}$. $\square$

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

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

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 $\precsim^{\mathrm{cxt}}$ over $\mathbf{\Lambda}^o$, called ***observational*** or ***contextual preorder*** is defined as

$$s \precsim^{\mathrm{cxt}} s' \quad \stackrel{\mathrm{def}}{=} \quad \forall C[X] \in \mathbf{\Lambda}^o . C[s]\Downarrow \implies C[s']\Downarrow.$$

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 $\lambda x.x\mathbf{\Omega} \precsim^{\mathrm{cxt}} \lambda x.xx$ or $\lambda x.xx \precsim^{\mathrm{cxt}} \lambda x.(\lambda y.xy)$. 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 $\mathbf{\Lambda}^o$.

$$s \precsim s' \quad \implies \quad \forall l \in \omega . \forall C[X] \in \mathbf{\Lambda}^o . C[s]\Downarrow_l \implies C[s']\Downarrow.$$

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 $s \precsim s'$, we have $s'P[s']Q[\vec{s'}]\Downarrow$, *i.e.* $C[s']\Downarrow$. $\square$

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 $\lesssim$ that the application operation in $\mathbf{\Lambda}^o$ is monotone in the *left* argument with respect to $\lesssim$. Operational extensionality is equivalent to the monotonicity of the application operation in the *right* argument, *i.e.*

$$s \lesssim s' \quad \Longrightarrow \quad \forall t \in \mathbf{\Lambda}^o . ts \lesssim ts';$$

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

$$s \lesssim s' \;\&\; t \lesssim t' \quad \Longrightarrow \quad st \lesssim s't'.$$

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.* $\square$

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

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

- for any s and s', $s \precapprox_0 s'$.
- $s \precapprox_{k+1} s'$ just in case whenever $s \Downarrow \lambda x.p$ then $s' \Downarrow \lambda x.p'$ and for every value v, $p[v/x] \precapprox_k p'[v/x]$.

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

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

(i) $s \precapprox t$

(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 $t\vec{v}\Downarrow$.

$\square$

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

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

Exercise 5.5.1 *Prove the following:*

(i) $\precapprox$ *is a preorder.*

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

$$s \precapprox t \quad \Longrightarrow \quad s[v/x] \precapprox t[v/x].$$

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 $R \in \mathcal{R}$

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

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.

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

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 $\mathrel{\underset{\sim}{\sqsubset}}$.*

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 $x \mathrel{\underset{\sim}{\sqsubset}} s$ then $x \leqslant s$
- if $s \leqslant s'$ and $t \leqslant t'$ and $s't' \mathrel{\underset{\sim}{\sqsubset}} r$ then $st \leqslant r$
- if $s \leqslant s'$ and $\lambda x.s' \mathrel{\underset{\sim}{\sqsubset}} r$ then $\lambda x.s \leqslant r$.

Exercise 5.5.5 *Prove the following:*

(i) *Whenever $s \leqslant t$ and $t \mathrel{\underset{\sim}{\sqsubset}} r$ 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 $\mathrel{\underset{\sim}{\sqsubset}}$ is contained in $\leqslant$.*

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

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

$\square$

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 $\mathrel{\underset{\sim}{\sqsubset}}$. Hence deduce the context lemma.*

[Hint: To prove that $\leqslant$ is contained in $\mathrel{\underset{\sim}{\sqsubset}}$, it suffices to show that $\leqslant$ is a pre-simulation (why?).]

Problems

Unless otherwise specified, assume $\Downarrow$ and $\mathrel{\underset{\sim}{\sqsubset}}$ 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 $\mathbf{\Omega} \equiv (\lambda x.xx)(\lambda x.xx)$ is a bottom element and **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 $s \in \mathbf{\Lambda}^o$

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

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.

6 (Very) Basic Recursion Theory

In this section we show the Turing completeness of the call-by-value λ-calculus (viewed as a minimal programming language) and the undecidability of β-convertibility.

In the following $s \Downarrow v$ shall mean the evaluation of program s to value v in the call-by-value λ-calculus; and $s \gg s'$ the reflexive, transitive closure of the one-step call-by-value reduction. Note that

$$s \Downarrow v \quad \Longleftrightarrow \quad s \gg v \;\; \& \;\; v \text{ is a value.}$$

6.1 Numerals

The salient feature of ***Scott numerals*** is the simplicity of the definition of predecessor. (Compare it with Church numerals.)

Definition 6.1.1

$$\begin{aligned}
\ulcorner 0 \urcorner &\stackrel{\text{def}}{=} \mathbf{k} \\
\ulcorner n+1 \urcorner &\stackrel{\text{def}}{=} \lambda xy.y\ulcorner n \urcorner \\
\text{succ} &\stackrel{\text{def}}{=} \lambda nxy.yn \\
\mathsf{pred} &\stackrel{\text{def}}{=} \lambda p.p\theta\mathbf{i} \\
\mathsf{case} &\stackrel{\text{def}}{=} \lambda xyz.xyz
\end{aligned}$$

where θ is any closed term and $\mathbf{i}$ the identity.

Note that for any values f and g

$$\mathsf{case}\ulcorner n \urcorner fg \quad \gg \quad \begin{cases} f & \text{if } n \text{ is } 0 \\ g(\ulcorner n-1 \urcorner) & \text{otherwise.} \end{cases}$$

6.2 Strong definability

A function can be defined by specifying its graph. We associate to every partial recursive function a CBV λ-calculus program that defines it i.e. the extensional behaviour of the program coincides with the graph of the function. Note that the program gives a way of *computing* it.

Definition 6.2.1 We say that a partial function $\phi : \mathbb{N}^m \rightharpoonup \mathbb{N}$ is ***strongly $\boldsymbol{\lambda}_{\mathbf{v}}$-definable*** by a program f just in case for every m-tuple $n_1, \cdots, n_m$ of natural numbers,

$$\begin{cases} \phi(\overrightarrow{n})\uparrow & \Longleftrightarrow \quad f\ulcorner n_1 \urcorner \cdots \ulcorner n_m \urcorner \Uparrow \\ \phi(\overrightarrow{n}) = l & \Longleftrightarrow \quad f\ulcorner n_1 \urcorner \cdots \ulcorner n_m \urcorner \Downarrow \ulcorner l \urcorner \end{cases}$$

where "$\cdots \uparrow$" means that "$\cdots$ is undefined".

Theorem 6.2.2 (Turing completeness) *A partial function $\mathbb{N}^m \rightharpoonup \mathbb{N}$ is partial recursive if and only if it is strongly $\boldsymbol{\lambda}_{\mathbf{v}}$-definable.*

Notation We write $\phi \rhd f$ to mean "ϕ is strongly $\boldsymbol{\lambda}_{\mathbf{v}}$-definable by f".

Lemma 6.2.3 *$st_1 \cdots t_n \Downarrow v$ if and only if for each i, $t_i \Downarrow u_i$ and $su_1 \cdots u_n \Downarrow v$.* $\square$

Exercise 6.2.4 (i) Prove the lemma.

(ii) The lemma is *not* true for CBN λ-calculus. Give a counterexample.

Proof of the theorem

It should be evident that a program of CBV λ-calculus defining a numeric function gives an algorithm for computing it. The direction "$\Leftarrow$" can be shown by appealing to Church's Thesis[1]. It then remains to prove "$\Rightarrow$".

Projection

$\ulcorner \mathsf{proj}_i^m \urcorner$ is $\lambda x_1 \cdots x_m.x_i$.

Composition

Suppose $\chi \rhd g$ and $\psi_i \rhd f_i$ and $\phi(\overrightarrow{n}) = \chi(\psi_1(\overrightarrow{n})), \cdots, \psi_m(\overrightarrow{n}))$. Now

$$\begin{array}{rll} \phi(\overrightarrow{n}) = p & \text{iff} & \text{for each } i,\ \psi_i(\overrightarrow{n}) = p_i \text{ and } \chi(p_1, \cdots, p_m) = p \\ & \text{iff} & f_i(\overrightarrow{\ulcorner n \urcorner}) \Downarrow p_i \text{ for each } i \text{ and } g\overrightarrow{\ulcorner p \urcorner} \Downarrow p \qquad \text{by Lemma 6.2.3} \\ & \text{iff} & (\lambda \overrightarrow{x}.g(f_1 \overrightarrow{x}) \cdots (f_m \overrightarrow{x}))\overrightarrow{\ulcorner n \urcorner} \Downarrow p \end{array}$$

Also

$$\begin{array}{rll} \phi(\overrightarrow{n}) \uparrow & \text{iff} & \text{for some } i,\ \psi_i(\overrightarrow{n})\uparrow \text{ or for each i, } \psi_i(\overrightarrow{n}) = p_i \text{ and } \chi(\overrightarrow{p}) \uparrow \\ & \text{iff} & \text{for some } i,\ f_i(\overrightarrow{\ulcorner n \urcorner}) \Uparrow \text{ or for each } i,\ f_i(\overrightarrow{\ulcorner n \urcorner}) \Downarrow p_i \text{ and } g\overrightarrow{\ulcorner p \urcorner} \Uparrow \\ & \text{iff} & (\lambda \overrightarrow{x}.g(f_1 \overrightarrow{x}) \cdots (f_m \overrightarrow{x}))\overrightarrow{\ulcorner n \urcorner} \Uparrow. \end{array}$$

Primitive recursion

Suppose

$$\begin{array}{rcl} \phi(0, \overrightarrow{y}) & \stackrel{\text{def}}{=} & \psi(\overrightarrow{y}) \\ \phi(k+1, \overrightarrow{y}) & \stackrel{\text{def}}{=} & \chi(\phi(k, \overrightarrow{y}), k, \overrightarrow{y}) \end{array}$$

where $\psi \rhd g$ and $\chi \rhd h$. Define $B \equiv \lambda xy.y(\lambda z.xxyz)$. Note for any value v

$$\begin{array}{rcl} BBv & > & (\lambda y.y(\lambda z.BByz))v \\ & > & v(\lambda z.BBvz). \end{array}$$

[1]By Church's Thesis, we shall mean the assertion that the *effectively computable* (partial) numeric functions are exactly the (partial) recursive functions.

Now set

$$\begin{aligned}
\Theta &\stackrel{\text{def}}{=} BB \\
v &\stackrel{\text{def}}{=} \lambda zx\overrightarrow{y}.\mathsf{case}\, x(g\overrightarrow{y})(\lambda\alpha.h(z\alpha\overrightarrow{y})\alpha\overrightarrow{y}) \\
\ulcorner\phi\urcorner &\stackrel{\text{def}}{=} \Theta v
\end{aligned}$$

Take $a, \overrightarrow{b}$ to be values, and set f to be Θv. Then

$$\begin{aligned}
fa\overrightarrow{b} &\gg v(\lambda z.\Theta vz)a\overrightarrow{b} \\
&\gg \mathsf{case}\, a(g\overrightarrow{b})(\lambda\alpha.h((\lambda z.\Theta vz)\alpha\overrightarrow{b})\alpha\overrightarrow{b})
\end{aligned}$$

- if a is $\ulcorner 0\urcorner$ then $f\ulcorner 0\urcorner\overrightarrow{b} \gg p$ provided $g\overrightarrow{b} \Downarrow p$
- if a is $\ulcorner n+1\urcorner$ then

$$\begin{aligned}
f\ulcorner n+1\urcorner\overrightarrow{b} &\gg (\lambda\alpha.h((\lambda z.\Theta vz)\alpha\overrightarrow{b})\alpha\overrightarrow{b})\ulcorner n\urcorner \\
&> h((\lambda z.\Theta vz)\ulcorner n\urcorner\overrightarrow{b})\ulcorner n\urcorner\overrightarrow{b} \\
&> h((\Theta v)\ulcorner n\urcorner\overrightarrow{b})\ulcorner n\urcorner\overrightarrow{b}.
\end{aligned}$$

Minimalization

Suppose $\psi \rhd g$. Define $\phi(\overrightarrow{y})$to be $\mu x.\psi(x, \overrightarrow{y})$. Set

$$\begin{aligned}
v &\stackrel{\text{def}}{=} \lambda zx\overrightarrow{y}.\mathsf{case}\,(gx\overrightarrow{y})x(\lambda\alpha.z(\mathrm{succ}\, x)\overrightarrow{y}) \\
h &\stackrel{\text{def}}{=} \Theta v.
\end{aligned}$$

Claim: For values $\overrightarrow{b}$,

$$h\ulcorner n\urcorner\overrightarrow{b} \quad\gg\quad \begin{cases} \ulcorner n\urcorner & \text{if } g\ulcorner n\urcorner\overrightarrow{b} \Downarrow \ulcorner 0\urcorner \\ h(\mathrm{succ}\ulcorner n\urcorner)\overrightarrow{b} & \text{otherwise if } g\ulcorner n\urcorner\overrightarrow{b} \Downarrow \ulcorner m+1\urcorner \text{ for some } m. \end{cases}$$

Now put $f \stackrel{\text{def}}{=} h\ulcorner 0\urcorner$.

□

Church numerals are defined as follows:

$$\underline{n} \stackrel{\text{def}}{=} \lambda fx.\underbrace{f(\cdots(f}_{n}\,x)\cdots)$$

Think of the Church numeral $\underline{n}$ as the procedure that takes a function-input and an argument-input, and applies the function n-times to the argument.

6.3 Undecidability of β-convertibility

Fix an effective *Gödel numbering* of λ-terms i.e. a (bijective) function $g : \mathbf{\Lambda} \longrightarrow \mathbb{N}$ that is computable. It should be clear that we have the following:

Fact 6.3.1 *(i) There is a total recursive function τ such that for any λ-terms s and t, $\tau(g(s), g(t)) = g(st)$.*

(ii) There is a total recursive function ν such that for any natural number n, $\nu(n) = g(\underline{n})$. □

Notation In the following we shall write

$$\lceil s \rceil \stackrel{\text{def}}{=} \underline{g(s)}$$

for each $s \in \mathbf{\Lambda}$ i.e. $\lceil s \rceil$ is the Church numeral of the Gödel numbering of s.

Lemma 6.3.2 *From the preceding fact it follows that there are λ-terms $\mathbf{p}$ and $\mathbf{q}$ such that*

$$\begin{aligned} \mathbf{p}\lceil s \rceil \lceil t \rceil &= \lceil st \rceil \\ \mathbf{q}\underline{n} &= \lceil \underline{n} \rceil \end{aligned}$$

for any λ-terms s and t, and for any $n \in \mathbb{N}$. □

Theorem 6.3.3 (Scott-Curry) *Let $\mathcal{A}$ and $\mathcal{B}$ be two collections of λ-terms that are closed under β-convertibility. There is no λ-term F such that for each n, $F\underline{n} = \underline{0}$ or $\underline{1}$, and satisfying*

$$F\lceil u \rceil = \begin{cases} \underline{1} & \text{if } u \in \mathcal{A} \\ \underline{0} & \text{if } u \in \mathcal{B} \end{cases}$$

Note that the $=$ is that of the formal system $\lambda\beta$.

Proof W.l.o.g. assume that $\mathcal{A}$ and $\mathcal{B}$ are disjoint. Suppose, for a contradiction, such an F exists.

Claim Fix some $A \in \mathcal{A}$ and $B \in \mathcal{B}$. There is a J such that

$$\begin{cases} F\lceil J \rceil = \underline{1} & \Longrightarrow \quad J = B \\ F\lceil J \rceil = \underline{0} & \Longrightarrow \quad J = A. \end{cases}$$

The Claim gives a contradiction. (Convince yourself that this is so.)

Construction of J: Let D be $\lambda xyz.z(\mathbf{k}y)x$. Then for any A and B, by a simple calculation, we see that

$$\begin{aligned} DAB\underline{1} &= B \\ DAB\underline{0} &= A. \end{aligned}$$

Let $H \equiv \lambda y.DAB(F(\mathbf{p}y(\mathbf{q}y)))$ and write $H\ulcorner H \urcorner$ as J. Now

$$\begin{array}{rcll} J & \equiv & H\ulcorner H\urcorner & \\ & = & DAB(F(\mathbf{p}\ulcorner H\urcorner(\mathbf{q}\ulcorner H\urcorner))) & \text{by Lemma 6.3.2} \\ & = & DAB(F(\mathbf{p}\ulcorner H\urcorner\ulcorner\ulcorner H\urcorner\urcorner)) & \text{by Lemma 6.3.2} \\ & = & DAB(F\ulcorner J\urcorner). & \end{array}$$

It follows that J thus defined satisfies the two implications in the Claim. □

"β is decidable" (by Church's Thesis) is the statement that there is λ-term G such that for any $s, t \in \mathbf{\Lambda}$,

$$G\ulcorner s\urcorner\ulcorner t\urcorner = \begin{cases} \underline{1} & \text{if } s = t \\ \underline{0} & \text{otherwise.} \end{cases}$$

Corollary 6.3.4 *$=_\beta$ is undecidable.*

Proof Suppose not. Take any λ-term, say, s. Let $\mathcal{A}$ be the $=_\beta$-equivalence class of s and $\mathcal{B}$ be $\mathbf{\Lambda} - \mathcal{A}$. Write $F = G\ulcorner s\urcorner$. Then F violates the theorem. □

Theorem 6.3.5 (Second Fixed-Point Theorem) *For any F, there exists an X such that*

$$F\ulcorner X\urcorner = X.$$

□

Problems

6.1 (i) Give the respective λ-terms that define the successor and predecessor (hard!) functions, and definition by cases (boolean conditional) for Church numerals.

(ii) Prove that CBV λ-calculus is Turing complete relative to Church numerals.

(iii) Prove that CBN λ-calculus is Turing complete.

6.2 Give an effective *Gödel numbering* of λ-terms i.e. a (bijective) function $g : \mathbf{\Lambda} \longrightarrow \mathbb{N}$ that is computable.

[Your encoding should be invariant over terms that are α-convertible. Hint: use ***de Bruiyn notation*** to represent λ-terms.]

6.3 Prove the following results:

(i) There is a total recursive function τ such that for any λ-terms s and t, $\tau(g(s), g(t)) = g(st)$.

(ii) There is a total recursive function ν such that for any natural number n, $\nu(n) = g(\underline{n})$.

6.4 (i) Why is the Claim in the proof of the Undecidability Theorem sufficient to force a contradiction?

(ii) Prove the Second Fixed-point Theorem. [Hint: Use the trick in the construction of J (in the proof of the undecidability Theorem) to construct the required X.]

6.5 Prove that $\{ s : s \text{ has a } \beta\text{-nf} \}$ is an r.e. set that is not recursive.

References

[AGM93] S. Abramsky, D. Gabbay, and T. S. E. Maibaum, editors. *Handbook of Logic in Computer Science, Vol 1.* Oxford University Press, 1993.

[AO93] S. Abramsky and C.-H. L. Ong. Full abstraction in the lazy lambda calculus. *Information and Computation*, 105:159–267, 1993.

[Aug84] L. Augustsson. A compiler for lazy ML. In *ACM Symp. on Lazy and Functional Programming*, pages 218–227, 1984.

[Bar77] J. Barwise, editor. *Handbook of Mathematical Logic.* North-Holland, 1977.

[Bar84] H. Barendregt. *The Lambda Calculus.* North-Holland, revised edition, 1984.

[Ber79] G. Berry. Modèles complètement adéquats et stables des lambda calculs typés. Technical report, Université Paris VII, 1979. Thèse de Doctorat d'Etat.

[Ber81] G. Berry. Some syntactic and categorical constructions of lambda calculus models. Papport de Recherche 80, Institute National de Recherche en Informatique et en Automatique (INRIA), 1981.

[Blo88] B. Bloom. Can LCF be topped? flat lattice models of typed lambda calculus. In *Proceedings of the third Symposium on LICS.* Computer Society Press, 1988.

[Chu40] A. Church. A formulation of the simple theory of types. *J. Symbolic Logic*, 5:56–68, 1940.

[Cur93a] P.-L. Curien. *Categorical Combinators, Sequential Algorithms, and Functional Programming.* Birkhäuser, second edition, 1993. Progress in Theoretical Computer Science Series.

[Cur93b] P.-L. Curien. Observable algorithms on concrete data structures. *Information and Computation*, 1993. To appear.

[DP90] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order.* Cambridge University Press, 1990. Cambridge Mathematical Textbooks.

[FW76] D. P. Friedman and D. S. Wise. Cons should not evaluate its arguments. In Michaelson and Milner, editors, *Automata, Languages and Programming.* Edinburgh University Press, 1976.

[Gir72] J.-Y. Girard. Interprétation fonctionnelle et élimination des coupures dans l'arithmétique d'ordre supérieur. Thèse de Doctorat d'Etat, Paris, 1972.

[GLT89] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types.* Cambridge University Press, 1989.

[Göd58] K. Gödel. Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes. *Dialectica*, pages 280–287, 1958.

[Göd90] K. Gödel. *Kurt Gödel Collected Works, Volumes I and II,* S. Feferman, *et al* (editors). Oxford Univ. Press, 1990.

[GS90] C. A. Gunter and D. S. Scott. Semantic domains. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol. B*, pages 635–674. Elsevier, 1990.

[Gun92] C. A. Gunter. *Semantics of Programming Languages: Structures and Techniques.* MIT Press, 1992.

[Ham88] A. G. Hamilton. *Logic for Mathematicians.* Cambridge University Press, revised edition, 1988.

[HM76] P. Henderson and J. H. Morris. A lazy evaluator. In *Third ACM Symposium on The Principles of Programming Languages, Atlanta, GA*, 1976.

[JM91] T. Jim and A. R. Meyer. Full abstraction and the context lemma. In Ito and Meyer, editors, *Proc. Int. Conf. Theoretical Aspects of Computer Software*, pages 131–151. Springer, 1991. LNCS Vol. 526.

[Kle59] S. C. Kleene. Recursive functionals and quantifiers of finite types I. *Trans. American Mathematical Society*, 91:1–52, 1959.

[Kle63] S. C. Kleene. Recursive functionals and quantifiers of finite types II. *Trans. American Mathematical Society*, 108:106–142, 1963.

[Koy82] C. P. J. Koymans. Models of the lambda calculus. *Information and Control*, 52:206–332, 1982.

[LS86] J. Lambek and P. J. Scott. *Introduction to Higher Order Categorical Logic.* Cambridge Studies in Advanced Mathematics No. 7. Cambridge University Press, 1986.

[MC88] A. R. Meyer and S. C. Cosmadakis. Semantical paradigms: notes for an invited lecture. In *Proc. 3rd Annual IEEE Symp. Logic in Computer Science.* Computer Society Press, 1988.

[Men87] E. Mendelson. *Introduction to Mathematical Logic.* Wadsworth, Inc., third edition, 1987.

[Mil77] R. Milner. Fully abstract models of typed lambda-calculus. *Theoretical Computer Science*, 4:1–22, 1977.

[Mil80] R. Milner. *A Calculus for Communicating Systems*, volume 92 of *Lecture Notes in Computer Science.* Springer-Verlag, Berlin, 1980.

[ML79] P. Martin-Löf. Constructive mathematics and computer programming. In *International Congress for Logic, Methodology and Philosophy of Science*, pages 538–571. North-Holland, 1979.

[Mos74] Y. Moschovakis. *Elementary Induction on Abstract Structures.* North Holland, 1974.

[MPW92] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I and II. *Information and Computation*, 100:1–77, 1992.

[Mul86] K. Mulmuley. Fully abstract submodels of typed lambda calculus. *Journal of Computer and System Sciences*, 33:2–46, 1986.

[Mul87] K. Mulmuley. *Full Abstraction and Semantic Equivalence.* MIT Press, 1987.

[Par80] D. M. Park. Concurrency on automata and infinite sequences. In P. Deussen, editor, *Conference on Theoretical Computer Science*, Berlin, 1980. Springer-Verlag. Lecture Notes in Computer Science Vol. 104.

[Pit94] A. M. Pitts. Computational adequacy via "mixed" inductive definitions. In *Proc. 18th Int. Symp. Mathematical Foundations of Computer Science, IX, New Orleans, 1993.* Springer, 1994. LNCS. To appear.

[PJ87] S. L. Peyton Jones. *The Implementation of Functional Programming Languages.* Prentice-Hall, 1987.

[Pla66] R. A. Platek. *Foundations of Recursion Theory.* PhD thesis, Standford University, 1966.

[Plo72] G. D. Plotkin. A set-theoretical definition of application. Technical Report MIP-R-95, School of A.I., Univ. of Edinburgh, 1972.

[Plo75] G. D. Plotkin. Call-by-name, call-by-value and the lambda calculus. *Theoretical Computer Science*, 1:125–159, 1975.

[Plo76] G. D. Plotkin. A powerdomain construction. *SIAM J. Computing*, 5(3):452–487, 1976.

[Plo77] G. D. Plotkin. LCF as a programming language. *Theoretical Computer Science*, 5:223–255, 1977.

[Plo81] G. D. Plotkin. Cpo's: Tools for making meanings. Post-Graduate Lecture Notes in Advanced Domain Theory, Dept. of Computer Science, Univ. of Edinburgh, 1981.

[Plo85] G. D. Plotkin. Types and partial functions. Post-Graduate Lecture Notes, Dept. of Computer Science, University of Edinburgh, 1985.

[Sco80] D. S. Scott. Relating theories of lambda calculus. In J. R. Hindley and J. P. Seldin, editors, *To H. B. Curry: Essays in Combinatory Logic, Lambda Calculus and Formalism*, pages 403–450. Academic Press, 1980.

[Sco93] D. S. Scott. A type-theoretical alternative to CUCH, ISWIM and OWHY. *Theoretical Computer Science*, 121:411–440, 1993.

[Sie92] K. Sieber. Reasoning about sequential functions via logical relations. In M. P. Fourman *et al*, editor, *Applications of Categories in Computer Science*, pages 66–94. Cambridge University Press, 1992.

[Spe62] C. Spector. Provably recursive functionals of analysis: a consistency proof of analysis by an extension of principles formulated in current intuitionistic mathematics. In *Recursive Function Theory,* Proc. Symposia in Pure Mathematics V, pages 1–27. American Mathematical Society, Providence, RI, 1962.

[Sta85] R. Statman. Logical relations and the typed λ-calculus. *Information and Control*, 65:85–97, 1985.

[Sto91a] A. Stoughton. Equationally fully abstract models of PCF. In *Proc. 5th Int. Conf. Math. Foundations of Programming Semantics*, pages 271–283. Springer, 1991. LNCS Vol. 442.

[Sto91b] A. Stoughton. Interdefinability of parallel operations in PCF. *Theoretical Computer Science*, 79:357–358, 1991.

[Sto91c] A. Stoughton. Parallel PCF has a unique extensional model. In *Proc. 6th IEEE Annual Symp. Logic in Computer Science*, pages 146–151. IEEE Computer Society Press, 1991.

[Tai67] W. W. Tait. Intensional interpretation of functionals of finite type i. *J. Symb. Logic*, 32:198–212, 1967.

[Tar55] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific J. Mathematics*, 5:285–309, 1955.

[Tur85] D. A. Turner. Miranda — a non-strict functional language with polymorphic types. In J. P. Jouannaud, editor, *Functional Programming Languages and Computer Architectures.* LNCS 201, Springer-Verlag, Berlin, 1985.

[vL90] J. van Leeuwen, editor. *Handbook of Theoretical Computer Science, Vol. B.* Elsevier, 1990.

[Wad85] P. Wadler. Introduction to orwell. Technical report, Oxford University Programming Research Group, 1985.

[Win93] G. Winskel. *The Formal Semantics of Programming Languages.* MIT Press, 1993. Foundations of Computing Series.

Index

A Class problems

Class 1

Syntax of the λ-calculus

Class 2

Reduction

Class 3

Combinatory logic

Class 4

Bohm's Theorem

Class 5

λ-calculus as a programming language

Class 6

Recursion theory

Class 7

Simply-typed λ-calculus, PCF, adequacy

B Sample examination questions

B.1 Prove that the λ-calculus is consistent.

[You should state what you mean by consistency carefully.]

B.2 (i) State and prove a result that relates the Church-Rosser property to the consistency of a formal theory of equations over a set of terms.

(iii) Prove that the formal theory $\lambda\beta$ is consistent.

(iii) Is the formal theory obtained from $\lambda\beta$ by augmenting it by $\mathbf{s} = \mathbf{k}$ consistent? Justify your result.

B.3 (i) State the Church-Rosser Theorem for β-reduction. Explain briefly why it is an important result in λ-calculus.

(ii) What is a fixed point combinator? Set $\mathbf{g} \equiv \lambda yf.f(yf)$. Prove that any λ-term s is a fixed point combinator if and only if s is a fixed point of $\mathbf{g}$.

(iii) Show that if $\mathbf{y}$ is a fixed point combinator, then so is $\mathbf{yg}$. Hence, or otherwise, show that there are infinitely many (β-inequivalent) fixed point combinators.

B.4 (i) Give a Gödel numbering on λ-terms i.e. an effectively given (computable) injective map $\#- : \mathbf{\Lambda} \longrightarrow \mathbb{N}$.

(ii) Use the Second Fixed Point Theorem to prove the Scott-Curry Theorem: Let $\mathcal{A}$ and $\mathcal{B}$ be two collections of λ-terms that are closed under β-convertibility. There is no λ-term p such that for each natural number n, $p\underline{n} = \underline{0}$ or $\underline{1}$, and

$$p\ulcorner u \urcorner \quad = \quad \begin{cases} \underline{1} & \text{if } u \in \mathcal{A} \\ \underline{0} & \text{if } u \in \mathcal{B} \end{cases}$$

where $=$ is the equational theory of the formal system $\lambda\beta$.

(iii) Hence, or otherwise, prove that the formal equational theory $\lambda\beta$ is undecidable.

B.5 "The λ-calculus is Turing complete." Discuss.

[You should state carefully any relevant definition and theorem, and give a proof of a major theorem in developing your argument.]

B.6 (i) Define Scott numerals and Church numerals.

(ii) Give a Gödel numbering on λ-terms i.e. an effectively given (computable) injective map $\#- : \mathbf{\Lambda} \longrightarrow \mathbb{N}$.

(iii) For each λ-term s, define $\ulcorner s \urcorner \stackrel{\text{def}}{=} \underline{\#s}$ where $\underline{n}$ is the n-th Church numeral. Prove the Second Fixed Point Theorem: for any $t \in \mathbf{\Lambda}$, there is a $u \in \mathbf{\Lambda}$ such that $t \ulcorner u \urcorner = u$.

B.7 (i) Define applicative simulation $\lessapprox$ for the call-by-name (CBN) λ-calculus.

(ii) Prove that applicative simulation is the maximal fixed point of a monotone function $F : \mathcal{R} \longrightarrow \mathcal{R}$ where $\mathcal{R}$ is the set of binary relations over the set $\mathbf{\Lambda}^o$ of closed λ-terms ordered by set inclusion.

(iii) State and prove a characterization result for $\lessapprox$.

(iv) Is it true that $\lambda x.sx \lessapprox s$ for all $s \in \mathbf{\Lambda}^o$?

B.8 Let s, s', t and t' range over closed λ-terms.

(i) Define applicative simulation $\lessapprox$ for the call-by-name (CBN) λ-calculus, and give (without proof) a characterization of it solely in terms of the convergence predicate $(-)\Downarrow$.

(ii) Prove that $\lambda\beta \vdash s = \lambda x.p$ if and only if $s\Downarrow$. Deduce that $s = s'$ and $s\Downarrow$ imply $s'\Downarrow$.

(iii) Hence, or otherwise, prove that if $\lambda\beta \vdash s = s'$ then $s \sim s'$ (i.e. $s \lessapprox s'$ and $s' \lessapprox s$)

(iv) Recall that a λ-theory is a consistent extension of $\lambda\beta$ that is closed under provability. Deduce that $\lambda\ell \stackrel{\text{def}}{=} \{\, s = t : s \sim t \,\}$ is a λ-theory.

B.9 (i) Define the call-by-name λ-calculus, and give its operational semantics in terms of a Martin-Löf style evaluation relation and a Plotkin-style transition relation. Show that the two are equivalent.

(ii) What is the Context Lemma? Give a careful proof in the case of the call-by-name λ-calculus.

B.10 Let $\mathcal{L}$ be the (first-order) language with constant symbols $\mathbf{k}, \mathbf{s}$ and a binary function symbol for application.

(i) Define an operation $\hat{\lambda}x.- : \mathcal{L} \longrightarrow \mathcal{L}$ parametrized by variables x of $\mathcal{L}$: for each x, there is a map $a \mapsto \hat{\lambda}x.a$, where $a \in \mathcal{L}$ and where $\hat{\lambda}x.a$ is defined by recursion as:

$$
\begin{array}{rcll}
\hat{\lambda}x.x & \stackrel{\text{def}}{=} & \mathbf{skk} & \\
\hat{\lambda}x.a & \stackrel{\text{def}}{=} & \mathbf{k}a & \text{if } a \text{ is a variable} \not\equiv x \text{ or } a \in \{\, \mathbf{s}, \mathbf{k} \,\} \\
\hat{\lambda}x.ab & \stackrel{\text{def}}{=} & \mathbf{s}(\hat{\lambda}x.a)(\hat{\lambda}x.b). & \text{if the previous cases do not apply}
\end{array}
$$

Prove that $\mathbf{C}_0 \vdash \forall x.(\hat{\lambda}x.a)x = a$. Hence prove that $\mathbf{C}_0 \vdash (\hat{\lambda}x.a)b = a[b/x]$ for all $b \in \mathcal{L}$.

(ii) What is a combinatory algebra? Define combinatory completeness (of an applicative structure).

(iii) Using (i), or some other abstraction algorithm, prove that an applicative structure is combinatory complete if and only if it can be given the structure of a combinatory algebra.

B.11 "The λ-calculus and Combinatory Logic are essentially equivalent." Discuss.

B.12 (i) Define the call-by-value (CBV) λ-calculus and give its operational semantics in terms of both the Martin-Löf style evaluation (big-step) relation $\Downarrow$ and Plotkin-style transition (small-step) relation $>$.

(ii) Prove that the big-step and small-step reduction relations are equivalent i.e. for any $s, v \in \mathbf{\Lambda}^o$

$$s \Downarrow v \quad \Longleftrightarrow \quad s \gg v \ \& \ v \not>$$

where $\gg$ is the reflexive transitive closure of $>$ and $v \not>$ means there is no $u \in \mathbf{\Lambda}$ for which $v > u$ holds.

(iii) Say that convergence testing is definable in a λ-calculus endowed with an evaluation relation $\Downarrow$ if there is a term $\mathbf{c} \in \mathbf{\Lambda}^o$ such that for any $s \in \mathbf{\Lambda}^o$

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

Is convergence testing definable in CBV λ-calculus?

B.13 (i) Define the syntax of Scott's language PCF and give its operational semantics in terms of either a small-step or a big-step reduction relation.

(ii) State and prove the Context Lemma for PCF.

B.14 State and prove the Weak Adequacy Theorem for Scott's language PCF.

C Lambda Calculus Mini-projects

University of Oxford, MSc (Maths & FoCS)

Lambda Calculus

Mini-project 1: Context lemma for the call-by-value λ-calculus

Michaelmas 1995

Instructions to candidates: *The following series of problems take you through a proof of the context lemma for the call-by-value λ-calculus. Your project should take the form of a mathematical report on your progress in solving the problems.*

We let p, r, s and t range over λ-terms. Programs of Plotkin's *call-by-value 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

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

We read $s \Downarrow v$ as "the program s converges or evaluates to value v", and write $s\Downarrow$ to mean $s \Downarrow v$ for some value v. Recall that $\Downarrow$ is deterministic i.e. $\Downarrow$ defines a partial function.

Definition 1 (Applicative simulation) (i) For closed s and t, s is said to *simulate t applicatively*, written $s \precsim t$, just in case for every finite (possibly empty) sequence of closed terms $r_1, \cdots, r_n$, if $s\vec{r}\Downarrow$ then $t\vec{r}\Downarrow$.

(ii) Applicative simulation can be extended to a relation over λ-terms in general: for any s and t, define $s \precsim t$ just in case $s_\sigma \precsim t_\sigma$ for every value substitution σ.

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

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

Problem 1 *Prove the following:*

(i) $\precsim$ is a preorder.

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

$$s \precsim t \quad \Longrightarrow \quad s[v/x] \precsim t[v/x].$$

$\square$

Definition 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 $R \in \mathcal{R}$

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

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.

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

Problem 2 *Prove the following:*

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

(ii) *$\lesssim$ is the same as $\underset{\sim}{\sqsubset}$.* □

Definition 3 (Observational preorder) For closed s and t, s is said to *approximate t observationally* just in case for any closed context $C[X]$ whenever $C[s]\Downarrow$ then $C[t]\Downarrow$.

Context lemma is said to be valid for call-by-value λ-calculus if applicative simulation (restricted to closed terms) coincides with observational preorder. Our aim is to prove the Context Lemma.

Definition 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 $x \underset{\sim}{\sqsubset} s$ then $x \leqslant s$
- if $s \leqslant s'$ and $t \leqslant t'$ and $s't' \underset{\sim}{\sqsubset} r$ then $st \leqslant r$
- if $s \leqslant s'$ and $\lambda x.s' \underset{\sim}{\sqsubset} r$ then $\lambda x.s \leqslant r$.

Problem 3 *Prove the following:*

(i) *Whenever $s \leqslant t$ and $t \underset{\sim}{\sqsubset} r$ 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 $\lambda x.s \leqslant \lambda x.s'$.* □

Problem 4 *Prove that $\leqslant$ is reflexive. Hence deduce that $\underset{\sim}{\sqsubset}$ is contained in $\leqslant$.* □

Problem 5 (Substitution Lemma) *Prove that whenever $s \leqslant s'$ and values $v \leqslant v'$ then*

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

□

Problem 6 *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.]

Problem 7 *Prove that $\leqslant$ coincides with $\underset{\sim}{\sqsubset}$. Hence deduce the context lemma.* □

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

University of Oxford, MSc (Maths & FoCS)

Lambda Calculus

Mini-project 2: Two exercises on PCF

Michaelmas 1995

Instructions to candidates: *Your project should take the form of a mathematical report on your progress in solving problems in both Parts I and II.*

I. An adequacy theorem

Let D be a CPO. A subset X of D is said to be *inductive* if it is downward closed and, for every ω-increasing chain $\langle d_i \rangle_{i \in \omega} \subseteq X$, the least upper bound (lub) $\bigsqcup_i d_i$ is an element of X.

Let r, s and t range over terms of PCF and u and v over values.

Definition 1 For each PCF-type A, for each d in the standard domain D_A of type A, and for each closed term s of type A, define $d \lhd_A s$ if

- $d = \perp$ or
- $s \Downarrow v$ and $d \lhd_A v$ where
  - $f \lhd_{B \Rightarrow C} u$ if for each $g \in D_B$ and for each closed term t of type B,

$$g \lhd_B t \quad \Longrightarrow \quad fg \lhd_C ut$$

  - $\mathsf{t} \lhd_o \mathsf{t}$, $\mathsf{f} \lhd_o \mathsf{f}$
  - $n \lhd_\iota n$.

Problem 1 *Prove that for each type A, and for each closed term s of type A, the set*

$$\{ d \in D_A : d \lhd_A s \}$$

is inductive.

Problem 2 *Suppose $x_1 : A_1, \cdots, x_n : A_n \vdash s : A$ for $n \geqslant 0$. For each $1 \leqslant i \leqslant n$, for each $d_i \in D_{A_i}$, and for each closed term t_i of type A_i such that $d_i \lhd_{A_i} t_i$, prove that*

$$[\![s]\!]_{[x_1 \mapsto d_1, \cdots, x_n \mapsto d_n]} \quad \lhd_A \quad s[t_1/x_1, \cdots, t_n/x_n].$$

Problem 3 (Adequacy theorem) *(i) Prove that for each closed term s of program type (i.e. o or ι), $[\![s]\!] \neq \perp$ if and only if $s \Downarrow v$ for some v.*

(ii) Is the result valid for closed terms of higher type? Justify your answer.

II. A combinatory logic version of PCF

The aim is to define a combinatory logic version of PCF called $\mathrm{PCF}^{\mathrm{cl}}$. The type structure and the constants (numerals, booleans, successor, predecessor, test-for-zero, conditional and fixed-point constants) should have the same sense as those of the standard PCF.

Problem 4 *Define the syntax of* $\mathrm{PCF}^{\mathrm{cl}}$ *and give the formal system that defines typing sequents of the form* $x_{A_1}, \cdots, x_{A_n} \vdash s : A$ *which means that the term* s *has type* A *in the context where (free) variables* $x_1, \cdots, x_n$ *have types* $A_1, \cdots, A_n$ *respectively.*

Problem 5 *Define either a "small-step" (Plotkin-style transition relation* $s \rightarrow s'$*) or a "big-step" (Martin-Löf style evaluation relation* $s \Downarrow v$*) call-by-name operational semantics for* $\mathrm{PCF}^{\mathrm{cl}}$*. What properties can you establish for the semantics?*

Problem 6 *Examine the relationship between* $\mathrm{PCF}^{\mathrm{cl}}$ *and* PCF*. To what extent do they agree?*

University of Oxford, MSc (Maths & FoCS)

Lambda Calculus

Mini-project 1: Call-by-name λ-calculus and convergence testing

Michaelmas 1996

Instructions to candidates: *Answer as many problems as you can.*

$\lambda\mathbf{c}$-terms, ranged over by s, t, etc., are defined as follows:

$$s \quad ::= \quad x \quad | \quad \mathbf{c} \quad | \quad (st) \quad | \quad (\lambda x.s)$$

where x ranges over a denumerable collection of variables, and $\mathbf{c}$ is a constant, known as ***convergence test***. Write $\mathbf{\Lambda}(\mathbf{c})$ (respectively $\mathbf{\Lambda}(\mathbf{c})^o$) for the collection of $\lambda\mathbf{c}$-terms (respectively closed $\lambda\mathbf{c}$-terms). *Programs* are closed terms; and *values*, ranged over by u, v, v', etc., are closed abstractions and $\mathbf{c}$. Evaluation is defined by induction over the following rules:

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

where $\mathbf{i} \equiv \lambda y.y$. 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.

Problem 1 (i) Give a Plotkin-style transition relation $> \subseteq \mathbf{\Lambda}(\mathbf{c})^o \times \mathbf{\Lambda}(\mathbf{c})^o$ that is equivalent to $\Downarrow$ in the sense that for any program s, and for any value v,

$$s \Downarrow v \quad \Longleftrightarrow \quad s \gg v \;\; \& \;\; v\not>$$

where $\gg$ is the reflexive, transitive closure of $>$, and $t\not>$ means $\neg[\exists t'.t > t']$.

(ii) A transition relation $\succ$ is said to be characterized by a set $\mathcal{E}$ of ***evaluation contexts*** (each of which must have exactly one "hole") and a set $\mathcal{R}$ of ***redex rules*** just in case for any s and s', $s \succ s'$ iff there is a unique $E \in \mathcal{E}$ such that $s \equiv E[\theta]$, $s' \equiv E[\theta']$ and $\theta \succ \theta'$ is an instance of a redex rule in $\mathcal{R}$.

Set $\mathcal{R}$ to be the following redex rules:

$$(\beta) \qquad (\lambda x.p)t \quad > \quad p[t/x]$$

$$(\mathbf{c}) \qquad \mathbf{c}(\lambda x.p) \quad > \quad \mathbf{i}.$$

Define the set $\mathcal{E}$ of evaluation contexts that, together with $\mathcal{R}$, characterize the transition relation $>$ in (i). Justify your answer.

Problem 2 Define the one-step reduction relation $\to$ (as a binary relation over $\mathbf{\Lambda}(\mathbf{c})$) by induction over the following rules:

$$(\lambda x.p)t \to p[t/x] \qquad \mathbf{c}(\lambda x.p) \to \mathbf{i} \qquad \frac{s \to s'}{ts \to ts'} \qquad \frac{s \to s'}{st \to s't} \qquad \frac{s \to s'}{\lambda x.s \to \lambda x.s'}.$$

Prove that $\to$ is Church-Rosser.

[Hint: Define an appropriate "parallel reduction" relation that satisfies the diamond property.]

Problem 3 (i) Show that convergence testing is not definable in call-by-name λ-calculus (as defined in section 5 of your notes). That is to say, writing $\Downarrow$ as the evaluation relation of call-by-name λ-calculus, show that there is no closed λ-term c such that $c\Downarrow$, and for any $s \in \mathbf{\Lambda}^o$

$$\begin{cases} s\Downarrow & \Longrightarrow \quad cs \Downarrow \mathbf{i} \\ s\Uparrow & \Longrightarrow \quad cs\Uparrow \end{cases}$$

where $s\Uparrow$ means $\neg[\exists v.s \Downarrow v]$.

(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 $p \sim q$ where $\sim$ is applicative bisimilarity i.e. $p \precsim q$ and $q \precsim p$.

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

(iv) Is it still the case that $p \sim q$ in λ-calculus with convergence testing (where $\sim$ is applicative bisimilarity of the augmented calculus)? No proof is required.

Michaelmas 1996, CHLO

University of Oxford, MSc (Maths & FoCS)

Lambda Calculus

Mini-project 2: Stable model for PCF

Michaelmas 1996

Instructions to candidates: *Answer as many problems as you can.*

Problem 1 Show that the following conditions on a CPO (i.e. a poset that has a least element and such that every directed subset has a LUB) are equivalent:

(1) Any two points that are bounded above have a LUB.

(2) Every subset that is bounded above has a LUB.

(3) Every non-empty subset has a GLB.

A CPO is said to be ***consistently complete*** just in case condition (2) (and hence, equivalently, (1) or (3)) is satisfied.

A consistently complete CPO D is said to be ***distributive*** just in case for any $x, y, z \in D$, if y and z are bounded above, then

$$x \wedge (y \vee z) \quad = \quad (x \wedge y) \vee (x \wedge z).$$

A ***Scott domain*** is a consistently complete, ω-algebraic CPO. A ***dI-domain*** is a distributive Scott domain that satisfies the following axiom:

(I): Every compact element dominates only finitely many elements.

An element x of a CPO D is a ***prime*** just in case for any subset $X \subseteq D$ that has a LUB,

$$x \leqslant \bigsqcup X \quad \Longrightarrow \quad \exists y \in X . x \leqslant y.$$

A CPO is ***prime algebraic*** if every element x is the LUB of the set of prime elements that are dominated by x.

Problem 2 (i) Prove that a Scott domain D is distributive if and only if for all $x, y, z \in D$, if $\{ x, y, z \}$ is bounded above then

$$x \wedge (y \vee z) \quad = \quad (x \wedge y) \vee (x \wedge z).$$

(ii) Suppose that a subset $X = X_1 \cup X_2$ of a Scott domain is bounded above then

$$\bigsqcup X \quad = \quad (\bigsqcup X_1) \vee (\bigsqcup X_2).$$

(iii) An element y of a CPO is said to ***cover*** x just in case $x < y$, and for any z, whenever $x \leqslant z \leqslant y$ then $z = y$ or $z = x$. Prove that in a dI-domain, prime elements are precisely those compact elements that cover exactly one element.

(iv) Using (i) to (iii), or otherwise, prove that in a Scott domain D that satisfies the axiom (I), distributivity is equivalent to prime algebraicity.

Let D and E be CPOs such that any two elements that are bounded above have a GLB. A function $f : D \longrightarrow E$ is said to be ***stable*** just in case it is continuous, and for any two elements x and y that are bounded above,

$$f(x \wedge y) \quad = \quad f(x) \wedge f(y).$$

Let f and g be functions from D to E. f is said to be less than g ***extensionally*** just in case

$$f \leqslant^{\mathrm{ext}} g \quad \stackrel{\mathrm{def}}{=} \quad \forall x \in D. f(x) \leqslant g(x).$$

f is said to be less than g according to the ***stable ordering*** just in case

$$f \leqslant^{\mathrm{s}} g \quad \stackrel{\mathrm{def}}{=} \quad \forall x, y \in D. x \leqslant y \implies f(x) = f(y) \wedge g(x).$$

Problem 3 (i) Show that with the extensional ordering on the set $D \Rightarrow E$ of stable functions from D to E, the application function $(D \Rightarrow E) \times D \longrightarrow E$ is not stable.

(ii) Prove that application is stable if and only if $D \Rightarrow E$ is ordered stably.

Problem 4 (i) Show (informally) that stable functions give a model of PCF.

(ii) Show that parallel-or is not definable in the model.

Michaelmas 1996, CHLO

D Overview lecture: copies of slides