Some Notes on Inductive and Co-Inductive Techniques in the Semantics of Functional Programs

DRAFT VERSION

Andrew M. Pitts

BRICS Notes Series **NS-94-5**

ISSN 0909-3206 **December 1994**

See back inner page for a list of recent publications in the BRICS Notes Series. Copies may be obtained by contacting:

BRICS
Department of Computer Science
University of Aarhus
Ny Munkegade, building 540
DK - 8000 Aarhus C
Denmark

Telephone: +45 8942 3360
Telefax: +45 8942 3255
Internet: BRICS@brics.dk

BRICS publications are in general accessible through WWW and anonymous FTP:

`http://www.brics.dk/`
`ftp ftp.brics.dk (cd pub/BRICS)`

Some Notes on Inductive and Co-inductive Techniques in the Semantics of Functional Programs

Andrew M. Pitts
Cambridge University Computer Laboratory
Cambridge CB2 3QG, UK

DRAFT VERSION

These notes have not yet been extensively tested and are bound to contain errors. When you find some, please email
Andrew.Pitts@cl.cam.ac.uk

Bibliography

ABRAMSKY, S. (1990), The lazy lambda calculus, *in* "Research Topics in Functional Programming" (Turner, D., Ed.), pp. 65–116, Addison-Wesley, Inc.

BIRD, R. AND WADLER, P. (1988), "Introduction to Functional Programming", Prentice-Hall International.

FIORE, M. P. (1993), A coinduction principle for recursive datatypes based on bisimulation, *in* "Proc. 8th Annual Symposium on Logic in Computer Science, Montréal", pp. 110–119, IEEE Computer Society Press, Washington.

FIORE, M. P. AND PLOTKIN, G. D. (1994), An axiomatisation of computationally adequate models of FPC, *in* "Proc. 9th Annual Symposium on Logic in Computer Science, Paris", pp. 92–102, IEEE Computer Society Press, Washington.

FREYD, P. J. (1991), Algebraically complete categories, *in* "Proc. 1990 Como Category Theory Conference", Lec. Notes in Math., Vol. 1488 (A. Carboni, *et al*, Eds), pp. 95–104, Springer-Verlag, Berlin.

FREYD, P. J. (1992), Remarks on algebraically compact categories, *in* "Applications of Categories in Computer Science" (Fourman, M. P., Johnstone, P. T., and Pitts, A. M., Eds), pp. 95–106, L.M.S. Lecture Note Series 177, Cambridge University Press.

GORDON, A. D. (1994), "Functional Programming and Input/Output", Distinguished Dissertations in Computer Science Series, Cambridge University Press.

GORDON, A. D. (1995), A Tutorial on co-induction and functional programming, *in* 1994 Glasgow Workshop on Functional Programming, Workshops in Computing, Springer-Verlag, *to appear.*

GUNTER, C. A. (1992), "Semantics of Programming Languages. Structures and Techniques", MIT Press.

HOWE, D. J. (1989), Equality in lazy computation systems, *in* "Proc. 4th Annual Symposium on Logic in Computer Science, Asilomar CA", pp. 198–203, IEEE Computer Society Press, Washington.

MEYER, A. R., AND COSMODAKIS, S. S. (1988), Semantical paradigms: notes for an invited lecture, *in* "Proc. 3rd Annual Symposium on Logic in Computer Science, Edinburgh", pp. 236–253, IEEE Computer Society Press, Washington.

MILNER, R. (1977), Fully abstract models of typed lambda-calculi, *Theoretical Computer Science* **4**, 1–23.

MILNER, R. (1989) "Communication and Concurrency", Prentice-Hall International.

MILNER, R., TOFTE, M., AND HARPER, R. (1990), "The Definition of Standard ML", MIT Press.

MOGGI, E. (1989), Notions of computations and monads, *Theoretical Computer Science* **93**, 55–92.

ONG, C.-H. L. (1993), Non-determinism in a functional setting, *in* "Proc. 8th Annual Symposium on Logic in Computer Science, Montréal", pp. 275–286, IEEE Computer Society Press, Washington.

PITTS, A. M. (1993), Computational adequacy via 'mixed' inductive definitions, *in* "Mathematical Foundations of Programming Language Semantics, Proc. 9th Int. Conf., New Orleans, LA, USA, April 1993", Lecture Notes in Computer Science, Vol. 802, pp. 72–82, Springer-Verlag, Berlin.

PITTS, A. M. (1993a) Relational properties of domains, Cambridge Univ. Computer Laboratory Technical Report No. 321.

PITTS, A. M. (1994), A co-induction principle for recursively defined domains, *Theoretical Computer Science* **124**, 195–219.

PLOTKIN, G. D. (1985), "Lectures on Predomains and Partial Functions". Notes for a course at CSLI, Stanford University.

REYNOLDS, J. C. (1974), On the relation between direct and continuation semantics, *in* "2nd Int. Colloq. on Automata, Languages and Programming" (Loeckx, J., Ed.), Lecture Notes in Computer Science, Vol. 14, pp. 141–156, Springer-Verlag, Berlin.

RITTER, E. AND PITTS, A. M. (1995), A fully abstract translation between a λ-calculus with reference types and Standard ML, *in* Proc. Conf. on Typed Lambda Calculus and Applications, Edinburgh, April

1995, Lecture Notes in Computer Science, Springer-Verlag, Berlin, *to appear.*

RUTTEN, J. J. M. M. (1993), A structural co-induction theorem,*in* "Mathematical Foundations of Programming Language Semantics, Proc. 9th Int. Conf., New Orleans, LA, USA, April 1993", Lecture Notes in Computer Science, Vol. 802, pp. 83–102, Springer-Verlag, Berlin.

SMITH, F. S. (1991), From operational to denotational semantics, *in* "Mathematical Foundations of Programming Language Semantics, Proc. 7th Int. Conf., Pittsburgh, USA, April 1991", Lecture Notes in Computer Science, Vol. 598, pp. 54–76, Springer-Verlag, Berlin.

SMYTH, M. B., AND PLOTKIN, G. D. (1982), The category-theoretic solution of recursive domain equations, *SIAM J. Computing* **11**, 761–783.

WADLER, P. (1992), Comprehending monads, *Mathematical Structures in Computer Science* **2**, 461–493.

WINSKEL, G. (1993), "The Formal Semantics of Programming Languages. An Introduction", MIT Press.

— CONTENTS —

— HIGHLIGHTS —

Applicative bisimilarity (5·1)
- proof of congruence properties via Howe's method (sect. 6)
- operational extensionality theorem: contextual equivalence = applicative bisimilarity (5·14)

Recursively defined domains
- "minimal invariant" property (9·4) ...
- ... and its application to proving computational adequacy (sect. 11)

Rationality of fixpoints with respect to contextual equivalence / applicative bisimilarity (10·9).

– A NOTE ON TERMINOLOGY –

The various types of program ordering and equivalence discussed in these notes are variously named in the literature.

Terminology of these Notes	Other common terminology
Applicative refinement	Applicative similarity
Applicative equivalence	Applicative bisimilarity
Observational refinement	Contextual preorder
Observational equivalence	Contextual equivalence

0. INTRODUCTION

The course will introduce some of the mathematical and logical methods which have been developed over the last 30 or so years to

formally specify the meaning (or "semantics") of various programming language constructs and to reason about their properties

Underlying theme: development of mathematical tools for establishing the behavioural equivalence of programs written in a given programming language.

Observational refinement relation

$E_1 \lesssim E_2$ holds iff for all complete programs $P[E_1]$ involving phrase E_1, any observable result of executing $P[E_1]$ is also an observable result of executing $P[E_2]$

E_1, E_2: two program phrases

$P[E_2]$: P, with occurrences of phrase E_1 replaced by phrase E_2.

Observational equivalence

$E_1 \simeq E_2$ iff $E_1 \lesssim E_2$ and $E_2 \lesssim E_1$

Example of $\simeq$ (in a functional language)

$$fact \simeq f(1,1)$$

where

$$\begin{cases} fact(x) \stackrel{def}{=} (\underline{if}\ x = 0\ \underline{then}\ 1\ \underline{else}\ x * fact(x-1)) \\ f(x,y) \stackrel{def}{=} \lambda z.\ \underline{if}\ y > z\ \underline{then}\ x\ \underline{else}\ f(x*y, y+1)z \end{cases}$$

Example of $\simeq$ (in a higher order imperative language with block structure)

[com = type of commands
  int = " " integers]

```
fun F(C: com) = .... : com
begin
    new x : int ;
    x := 0 ;
    F(x := x+1)
end
```

$\simeq$

```
fun F(C: com) = ... : com
F(skip)
```

To make these notions precise for a particular programming language, have to specify it

- Syntax : - what are the legal program phrases
  - what constitutes a complete (executable) program

- Operational semantics : a formal specification of how complete programs are executed.
  [Abstract machines ; Plotkin's SOS]

- what are the observable results of program execution.

In this course we will only deal with the case of functional programming languages

- not procedural ("imperative"), but rather "declaritive" style of programming : a program typically consists of some definitions ("declarations") (eg recursively defined functions) + an expression to evaluate which uses those definitions
- functions can be values (results of evaluation)

- operational semantics can be given via an evaluation relation
  $e \Downarrow v$ "expression e evaluates to value v"

and observational refinement, $e_1 \lesssim e_2$, defined by

$$e_1 \lesssim e_2 \iff \forall e[-].\ \forall v_1\ \ e[e_1] \Downarrow v_1 \Rightarrow \exists v_2\ \left(e[e_2] \Downarrow v_2\ \&\ obs(v_1) = obs(v_2) \right)$$

Where $obs(v)$ = observable form of value v

(eg. "v is the number 42"
"v is a pair"
"v is a function abstraction")

Example

$$fact \simeq f(1,1)$$

Where

$$fact(x) \stackrel{\text{def}}{=} \text{if } x = 0 \text{ then } 1 \text{ else } x * fact(x-1)$$

$$f(x,y) \stackrel{\text{def}}{=} \lambda z.\ \text{if } y > z \text{ then } x \text{ else } (f(x*y, y+1))(z)$$

<u>Problem</u> : it is very difficult to establish properties of $\simeq$ (such as the above examples) working directly from the definition (because of the quantification over all possible ways of using an expression).

We will look at two ways of tackling this problem :

(1) <u>Applicative (bi)simulation</u> - a useful co-inductive characterization of $\sqsubseteq$ (and $\simeq$) derived directly from the operational semantics of (functional) prog. langs.

(2) <u>Denotational semantics</u>

General idea : meaning of expressions in a prog. lang. given by a semantic function mapping expressions e to elements $[\![e]\!]$ of a suitable mathematical structure, D. Basic requirements :

(<u>D for "domain"</u>)

<u>compositionality</u> : the denotation $[\![e]\!]$ of compound expression e is built up from the denotations of its subexpressions by applying operations on D corresponding to the various expression forming constructs of the language.

<u>computational adequacy</u>

$$[\![e_1]\!] = [\![e_2]\!] \implies e_1 \simeq e_2$$

(hence can use the den. sem. to establish instances of obs. equiv.).

More generally, the mathematical structure D will carry a partial order $\sqsubseteq$, and we'll require

$$[\![e_1]\!] \sqsubseteq [\![e_2]\!] \implies e_1 \lesssim e_2$$

What kind of posets are suitable for giving such a denotational semantics?

There is a large body of work – <u>domain theory</u> – giving answers to this question.

A key technical difficulty that has to be overcome is that various PL features force one to find solutions D to "domain equations"

$$D = \Phi(D)$$

which (for cardinality reasons) have no solution in the category of sets & functions or the category of posets & monotone functions.

We will develop the theory of <u>recursively defined domains</u> in a category of (ω-)chain complete posets & continuous functions.

The "full abstraction" problem

Can give computationally adequate den. sem. to very many prog. lang. features using domain theory

BUT

for prog. langs with higher order features, it has turned out to be extremely difficult to get the reverse implication

$$e_1 \simeq e_2 \Rightarrow [\![e_1]\!] = [\![e_2]\!]$$

i.e. there can be obs. equiv. expressions with unequal denotations — bad! (eg lessens the usefulness of $[\![-]\!]$ for establishing conditional equational props. of $\simeq$).

We will demonstrate this "lack of full abstraction" of the standard domain theoretic approach to den. Sem. (~~&~~ would review some of the ways round the problem, if there were time...).

Reading material

Text books on prog. lang. semantics:

C. Gunter, "Semantics of Programming Languages. Structures & Techniques", MIT Press, 1992

G. Winskel, "The Formal Semantics of Programming Languages. An Introduction", MIT Press, 1993.

Background on functional programming:

H. Abelson & G. J. Sussman, "Structure & Interpretation of Computer Programs", MIT Press, 1985

L.C. Paulson, "ML for the Working Programmer", CUP, 1991.

R. Bird & P. Wadler, "Introduction to Functional Programming", Prentice-Hall, 1988.

1. PRELIMINARIES ON INDUCTIVE DEFINITIONS

Notation: given a set X,

$$\mathcal{P}(X) \stackrel{\text{def}}{=} \{S \mid S \subseteq X\} \quad \text{powerset of } X$$

1.1 Definitions

A monotone operator on $\mathcal{P}(X)$ is a function $\Phi : \mathcal{P}(X) \to \mathcal{P}(X)$ satisfying

$$S \subseteq S' \subseteq X \;\Rightarrow\; \Phi(S) \subseteq \Phi(S')$$

$\mu\Phi \in \mathcal{P}(X)$ is a least prefixed point for Φ if it satisfies

(1.1.1) $\quad \Phi(\mu\Phi) \subseteq \mu\Phi \qquad$ ("$\mu\Phi$ is a prefixed point of Φ")

(1.1.2) $\quad \forall S \in \mathcal{P}(X).\ \Phi(S) \subseteq S \;\Rightarrow\; \mu\Phi \subseteq S \qquad$ ("and it is the least such")

Note:

- $\exists$ at most one subset $\mu\Phi$ satisfying (i) + (ii)
- $\Phi(\mu\Phi) = \mu\Phi$ (Because

  $\Phi(\Phi(\mu\Phi)) \subseteq \Phi(\mu\Phi) \qquad$ (i) + monotonicity

  so taking $S = \Phi(\mu\Phi)$ in (ii), get $\mu\Phi \subseteq \Phi(\mu\Phi)$.)

So $\mu\Phi$ is also the least fixed point (lfp) for Φ.

1.1 Theorem (Tarski-Knaster Fixed Point Theorem, for $\mathcal{P}(X)$)

Every monotone operator $\Phi : \mathcal{P}(X) \to \mathcal{P}(X)$ possesses a least prefixed point, $\mu\Phi$.

Proof

Consider $\mu\Phi \stackrel{\text{def}}{=} \bigcap \{ S \in \mathcal{P}(X) \mid \Phi(S) \subseteq S \}$

Clearly this satisfies (1.1.2).

To verify (1.1.1), suffices to show

$$\forall S.\ \Phi(S) \subseteq S \Rightarrow \Phi(\mu\Phi) \subseteq S$$

But if $\Phi(S) \subseteq S$, then $\mu\Phi \subseteq S$, so $\Phi(\mu\Phi) \subseteq \Phi(S) \subseteq S$.

□

1.2 Definitions

A set of <u>rules</u> on X is a subset

$$\mathcal{R} \subseteq \mathcal{P}(X) \times X \ (= \{ (S, x) \mid S \subseteq X \ \& \ x \in X \})$$

Each such $\mathcal{R}$ determines a monotone operator

$$\Phi_R : \mathcal{P}(X) \to \mathcal{P}(X)$$

where $\Phi_{\mathcal{R}}(S) \stackrel{\text{def}}{=} \{ x \in X \mid \exists (S', x) \in \mathcal{R}.\ S' \subseteq S \}$

(check: Φ_R is monotone).

$\mu\Phi_R \subseteq X$ is called <u>the subset of X inductively defined by the rules R</u>

(<u>Exercise</u>: show that <u>any</u> monotone operator $\mathcal{P}(X) \to \mathcal{P}(X)$ is of the form Φ_R for some rule set $\mathcal{R}$ on X.)

Note: $C \in \mathcal{P}(X)$ is a prefixed point of Φ_R (ie $\Phi_R(C) \subseteq C$) iff C is closed under the rules in R (or R-closed), meaning

for each rule $(S, x) \in R$, if the "hypothesis" of the rule is contained in C $(S \subseteq C)$, then the "conclusion" of the rule is an element of C $(x \in C)$.

Thus $\mu \Phi_R$ is the least subset of X that is closed under the rules of R. Hence we have

1.3 Lemma (Principle of Rule Induction)

Suppose $S \subseteq X$ is the subset inductively defined by a rule set R on X. For any $S' \subseteq S$, to prove $S' = S$ it suffices to show that S' is closed under the rules in R.

□

1.4 Notation for finitary rules

Rule set R is finitary if $(S, x) \in R \Rightarrow S$ finite

If $S \subseteq X$ is inductively defined by finitary rule set R, we often write a rule $(\{x_1, \ldots, x_n\}, x) \in R$ as

$$\frac{x_1 \in S \quad \cdots \quad x_n \in S}{x \in S}$$

<u>1.4 Example</u>: reflexive-transitive closure $\to^* \subseteq A \times A$ of a binary relation $\to \subseteq A \times A$ is $\mu \Phi_R$ where R is

$$\{(\emptyset, (a, a')) \mid (a, a') \in \to\}$$
$$\cup \{(\emptyset, (a, a)) \mid a \in A\}$$
$$\cup \{(\{(a_1, a_2), (a_2, a_3)\}, (a_1, a_3)) \mid a_1, a_2, a_3 \in A\}$$

Using infix notation and the above convention for writing rules, R consists of all rules matching the following patterns:

empty hypothesis

$$\frac{}{a \to^* a'} \quad (a \to a')$$

rule holds only if <u>side-condition</u> is satisfied

$$\frac{}{a \to^* a}$$

$$\frac{a_1 \to^* a_2 \qquad a_2 \to^* a_3}{a_1 \to^* a_3}$$

Thus $S \subseteq A \times A$ is closed under the rules iff

$$\to \subseteq S$$
$$\Delta_A \stackrel{\text{def}}{=} \{(a, a) \mid a \in A\} \subseteq S$$
$$S \circ S = \{(a_1, a_3) \mid \exists a_2 \in A . (a_1, a_2) \in S \ \& \ (a_2, a_3) \in S\} \subseteq S$$

and $\to^*$ is the smallest such S.

<u>Claim</u>: $\to^* = \{(a, a') \mid \exists n \geq 1, \exists a_1, \ldots, a_n \,.\, a = a_1 \to a_2 \to \cdots \to a_n = a'\}$
Proof: $\subseteq$ by rule induction; $\supseteq$ by mathematical induction.

[Plenty of other egs of inductive def^{ns} later in course.]

1.5 Proposition

Suppose R is finitary (i.e. $\forall (S,x) \in R.\ S$ finite).
Define $\mu^{(0)}\Phi_R, \mu^{(1)}\Phi_R, \ldots \subseteq X$ by

$$\begin{cases} \mu^{(0)}\Phi_R = \emptyset \\ \mu^{(n+1)}\Phi_R = \Phi_R(\mu^{(n)}\Phi_R) \end{cases}$$

Then $\mu\Phi_R = \bigcup_{n<\omega} \mu^{(n)}\Phi_R$.

Proof

Let $\mu^{(\omega)}\Phi_R \stackrel{\text{def}}{=} \bigcup_{n<\omega} \mu^{(n)}\Phi_R$.

If $x \in \Phi(\mu^{(\omega)}\Phi_R)$, then $\exists (S,x) \in R$ with $S \subseteq \mu^{(\omega)}\Phi_R$.

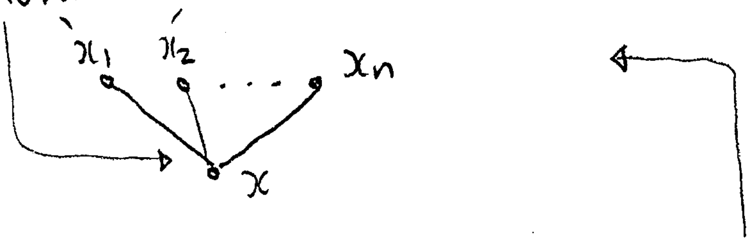
Note that $\mu^{(0)}\Phi_R \subseteq \mu^{(1)}\Phi_R \subseteq \ldots$
(prove $\forall n\ \mu^{(n)}\Phi_R \subseteq \mu^{(n+1)}\Phi_R$ by induction on n)

Since S is finite, we therefore have $S \subseteq \mu^{(N)}\Phi_R$ for some N. Hence $x \in \Phi(\mu^{(N)}\Phi_R) = \mu^{(N+1)}\Phi_R \subseteq \mu^{(\omega)}\Phi_R$.
Thus $\Phi(\mu^{(\omega)}\Phi_R) \subseteq \mu^{(\omega)}\Phi_R$.
So $\mu\Phi_R \subseteq \mu^{(\omega)}\Phi_R$.

Conversely can prove $\forall n\ (\mu^{(n)}\Phi_R \subseteq \mu\Phi_R)$ by induction on n, and hence $\mu^{(\omega)}\Phi_R \subseteq \mu\Phi_R$. $\square$

1.6 Remark

A <u>proof</u> that $x \in \mu\Phi_R$ (R finitary) is a finite rooted tree whose nodes are labelled with elements of X with the property that at each node



the finite set $\{x_1, \dots x_n\}$ of labels of children is such that $(\{x_1,\dots,x_n\}, x) \in R$.

It is not hard to prove (by induction on n) that

$$\mu^{(n)}\Phi_R = \{x \in X \mid \exists \text{ proof that } x \in \mu\Phi_R \text{ of height} < n\}$$

Hence

$$\mu\Phi_R = \{x \in X \mid \exists \text{ proof that } x \in \mu\Phi_R\}.$$

<u>Eg</u>: in 1.4 take $A = \mathbb{Z}$, $\to = \{(n, n+1) \mid n \in \mathbb{Z}\}$.
Then $\to^* = \{(m,n) \mid m < n\}$.
Here's a proof that $3 \to^* 6$:

$$\text{NB redundant}\left\{\quad \dfrac{\dfrac{\overline{3\to^*3} \qquad \overline{3\to^*4}}{3\to^*4} \qquad \dfrac{\overline{4\to^*5} \qquad \overline{5\to^*6}}{4\to^*6}}{3\to^*6}\right.$$

(In practice, will label the proof trees with names of rules (schemas).)

Generalization from powersets to complete lattices

1.7 Definitions

A partial order $\leq$ on a set P is a binary relation ($\leq\ \subseteq P \times P$) which is

reflexive: $\forall p \in P\ (p \leq p)$

transitive: $\forall p, p', p'' \in P\ (p \leq p' \ \&\ p' \leq p'' \Rightarrow p \leq p'')$

anti-symmetric: $\forall p, p' \in P\ (p \leq p' \ \&\ p' \leq p \Rightarrow p = p')$

A partially ordered set (or poset) is a set P equipped with a partial order $\leq_P$ (usually just written $\leq$).

A function $f : P \to Q$ between posets is monotone iff $\forall p, p' \in P\ (p \leq_P p' \Rightarrow f(p) \leq_Q f(p'))$

An upper bound for a subset $S \subseteq P$ of a poset is an element p satisfying $\forall s \in S\ (s \leq p)$.

A least upper bound (or lub, or sup, or join) for S is an element $\bigvee S \in P$ satisfying

- $\forall s \in S\ (s \leq \bigvee S)$ "$\bigvee S$ is an upper bound"
- $\forall p \in P\ (\forall s \in S\ (s \leq p) \Rightarrow \bigvee S \leq p)$ "and it is the least such".

Note $\bigvee S$ is unique if it exists.

The <u>opposite</u>, P^{op}, of a poset P is the poset with the same set of elements, but with partial order defined by

$$p \leqslant_{P^{op}} p' \iff p' \leqslant_P p .$$

Given $S \subseteq P$, $\bigwedge S$ is the <u>greatest lower bound</u> (or <u>glb</u>, or <u>inf</u>, or <u>meet</u>) of S iff it is the join of S in P^{op}.

1.8 Lemma

A poset possesses joins for all subsets iff it possesses meets for them. In this case we call the poset a <u>complete lattice</u>. In particular P is a complete lattice iff P^{op} is.

<u>Proof</u>

It is easy to check that a meet $\bigwedge S$ can be expressed as a join

$$\bigwedge S = \bigvee \{ p \in P \mid \forall s \in S (p \leq s) \}$$

(and so, dually

$$\bigvee S = \bigwedge \{ p \in P \mid \forall s \in S (s \leq p) \} \ ). \quad \square$$

<u>Note</u>: $(\mathcal{P}(X), \subseteq)$ is a complete lattice with $\bigwedge = \bigcap$ and $\bigvee = \bigcup$; hence its opposite, viz $(\mathcal{P}(X), \supseteq)$ is also a complete lattice.

1.9 Theorem (Tarski-Knaster Fixed Point Theorem for a complete lattice)

Let $f : P \to P$ be a monotone function from a complete lattice to itself. Then f possesses a least prefixed point, i.e. an element $\mu(f) \in P$ satisfying

- $f(\mu(f)) \leqslant \mu(f)$
- $\forall p \in P \; (f(p) \leqslant p \Rightarrow \mu(f) \leqslant p)$

Proof

Define $\mu(f) \stackrel{\text{def}}{=} \bigwedge \{p \in P \mid f(p) \leqslant p\}$. The proof that this works is just as in 1.1. $\square$

1.10 Corollary

With $f : P \to P$ as in 1.9, there is a greatest post-fixed point for f, i.e. $\nu(f) \in P$ such that

- $\nu(f) \leqslant f(\nu(f))$
- $\forall p \in P \; (p \leqslant f(p) \Rightarrow p \leqslant \nu(f))$

Proof

Apply 1.9 to the complete lattice P^{op}.
(Thus $\nu(f) = \bigvee \{p \in P \mid p \leqslant f(p)\}$.) $\square$

1.11 Definition

Given a set of rules R on a set X (ie. $R \subseteq P(X) \times X$), the <u>subset of X co-inductively defined by R</u> is $\nu(\Phi_R)$

(where $\Phi_R : P(X) \to P(X)$ is the monotone function on the complete lattice $P(X)$ associated with R as in 1.2).

<u>Note</u>: $D \in P(X)$ is a post-fixed point of Φ_R (ie $D \subseteq \Phi_R(D)$) iff D is <u>R-dense</u>, meaning

for each $x \in D$, there is some rule $(S, x) \in R$ with $S \subseteq D$.

Thus $\nu(\Phi_R)$ is the <u>biggest R-dense subset of X</u>. Hence we have

1.12 Lemma (Principle of Rule Co-Induction)

Suppose $S \subseteq X$ is the subset co-inductively defined by a rule set R on X. For any $x \in X$ to prove $x \in S$ it suffices to find some R-dense subset $D \subseteq X$ with $x \in D$.

$\square$

1.13 Example (cf. Example 1.4)

Given a binary relation $\rightarrow \subseteq A \times A$, consider the rule set
$R = \{ (\{a'\}, a) \mid a \rightarrow a' \}$ on A.
Thus $D \subseteq A$ is R-dense iff

$$a \in D \Rightarrow \exists a' \in D \, (a \rightarrow a')$$

From this, it's not hard to see that

$$\nu(\Phi_R) = \{ a \in A \mid \underbrace{\exists a_0, a_1, a_2, \ldots \, (a = a_0 \rightarrow a_1 \rightarrow a_2 \rightarrow \cdots)}_{\text{i.e. } \exists \alpha \in A^{\mathbb{N}} (\alpha(0) = a \,\&\, \forall n \in \mathbb{N}.\, \alpha(n) \rightarrow \alpha(n+1))} \}$$

[Further egs of co-inductively defined sets will occur in the work on applicative bisimulation.]

Further reading

On inductive definitions:

P. Aczel, "An Introduction to Inductive Definitions". In J. Barwise (ed.), "Handbook of Mathematical Logic" (North-Holland, 1977), pp 739–782.

On ordered structures:

B. A. Davey & H. A. Priestley, "Introduction to Lattices and Order", (CUP, 1990).

2. SYNTAX

of a simple, functional programming language, $\mathbb{L}$.

- Based on untyped λ-calculus & evaluation to canonical form.
- Chosen to be very rudimentary, in order to help see the wood from the trees in the theoretical development, so...
- inconveniently simple for writing programs, but...
- Turing powerful (i.e. can program all recursive partial functions)

The following (mutually disjoint) sets of symbols will be fixed throughout:

Var : a countably infinite set of <u>variables</u>, written $x, x', x'', \ldots, y, y', y'', \ldots$

Const $\stackrel{\text{def}}{=} \{\text{true}, \text{false}\} \cup \{\underline{n} \mid n \in \mathbb{Z}\}$: boolean and integer <u>constants</u>

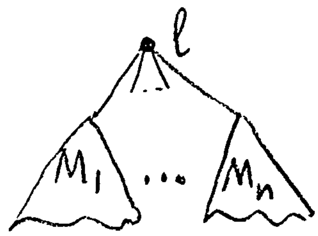
Op $\stackrel{\text{def}}{=} \{=, \leq, \geq, <, >, +, -, *, \ldots\}$: binary operator symbols (boolean- & integer-valued)

2.1 Abstract syntax of $\mathbb{L}$

The terms of $\mathbb{L}$ are a certain inductively defined subset of the set of all finite trees whose nodes are labelled by elements of the set

$$Var \cup Const \cup Op \cup \{ \text{if}, \text{pair}, \text{split}, \lambda, \text{app}, \text{rec} \}$$

Notation: given trees $M_1, \ldots, M_n$ and label ℓ, $\ell(M_1, \ldots, M_n)$ denotes the tree



Rules inductively defining the set Term of $\mathbb{L}$ terms :

Rules inductively defining the set Term of $\mathbb{L}$-terms.

$$\frac{}{x \in \mathrm{Term}}\ (x \in \mathrm{Var}) \qquad \frac{}{c \in \mathrm{Term}}\ (c \in \mathrm{Const})$$

$$\frac{M_1 \in \mathrm{Term} \qquad M_2 \in \mathrm{Term}}{op(M_1, M_2) \in \mathrm{Term}}\ (op \in \mathrm{Op})$$

$$\frac{M_1 \in \mathrm{Term} \qquad M_2 \in \mathrm{Term} \qquad M_3 \in \mathrm{Term}}{\mathrm{if}(M_1, M_2, M_3) \in \mathrm{Term}}$$

$$\frac{M_1 \in \mathrm{Term} \qquad M_2 \in \mathrm{Term}}{\mathrm{pair}(M_1, M_2) \in \mathrm{Term}}$$

$$\frac{M_1 \in \mathrm{Term} \qquad M_2 \in \mathrm{Term}}{\mathrm{split}(M_1, x, x', M_2) \in \mathrm{Term}}\ (x, x' \in \mathrm{Var})$$

$$\frac{M \in \mathrm{Term}}{\lambda(x, M) \in \mathrm{Term}}\ (x \in \mathrm{Var})$$

$$\frac{M_1 \in \mathrm{Term} \qquad M_2 \in \mathrm{Term}}{\mathrm{app}(M_1, M_2) \in \mathrm{Term}}$$

$$\frac{M \in \mathrm{Term}}{\mathrm{rec}(x, M) \in \mathrm{Term}}\ (x \in \mathrm{Var})$$

2.2 Concrete syntax of $\mathbb{L}$

To make things easier to read we will use a linear syntax (i.e. strings of symbols) disambiguated with punctuation & various binding conventions to refer to the terms of $\mathbb{L}$:

(i) Infix notation for binary operators:

M_1 op M_2 means op(M_1, M_2)

(ii) Conditional expressions:

if M_1 then M_2 else M_3 means if(M_1, M_2, M_3)

(iii) Pairing: pair(M_1, M_2) will be written (M_1, M_2)

split(M_1, x, x', M_2) will be written

split M_1 as (x, x') in M_2

(iv) Function abstraction: $\lambda(x, M)$ will be written

$\lambda x. M$ [Intended meaning: "the function $x \mapsto M$"]

(v) Function application: app(M_1, M_2) will be written just as $M_1\, M_2$

(vi) Recursively defined terms: rec(x, M) will be written rec $x. M$

[Intended meaning: anonymous notation for "the element (recursively) defined by $x = M$".]

Summary of the concrete syntax of $\mathbb{L}$-terms, M

$$
\begin{array}{lll}
M ::= & x & \text{variable} \\
| & c & \text{constant} \\
| & M \; op \; M & \text{binary operator} \\
| & \text{if } M \text{ then } M \text{ else } M & \text{conditional} \\
| & (M, M) & \text{pair} \\
| & \text{split } M \text{ as } (x, x) \text{ in } M & \text{pair-destructor} \\
| & \lambda x . M & \text{function abstraction} \\
| & M M & \text{function application} \\
| & \text{rec } x . M & \text{recursively defined term}
\end{array}
$$

where $x \in Var$

$c \in Const = \{true, false\} \cup \{\underline{n} \mid n \in \mathbb{Z}\}$

$op \in \{=, \leq, \geq, <, >, +, -, *, \ldots\}$

<u>Conventions</u> to disambiguate strings of symbols into syntax trees:

- Scope of $\lambda x.-$ and $\text{rec}\, x.-$ extends as far to the right as possible.
  Eg $\lambda x. MN$ means $\lambda x.(MN)$ not $(\lambda x. M)N$

- Function application associates to the left.
  Eg MNP means $(MN)P$ not $M(NP)$.

- Usual binding precedences for arithmetic & boolean operators; function application binds more tightly
  Eg $x * f(x - \underline{1})$ means $x * (f(x-\underline{1}))$ not $(x*f)(x-\underline{1})$.

2.2 Defined notation

$(\text{let } x = M \text{ in } N) \stackrel{\text{def}}{=} (\lambda x. N)\, M$

$(\text{letrec } f(x) = M \text{ in } N) \stackrel{\text{def}}{=} \text{let } f = (\text{rec} f. \lambda x. M) \text{ in } N$

$\text{fst}(M) \stackrel{\text{def}}{=} \text{split } M \text{ as } (x, y) \text{ in } x$

$\text{snd}(M) \stackrel{\text{def}}{=} \text{split } M \text{ as } (x, y) \text{ in } y$

2.3 Examples of terms

(i) Factorial function.

$\text{rec} f. \lambda x. \text{ if } x \leq \underline{0} \text{ then } \underline{1} \text{ else } x * f(x - \underline{1})$

[Cf. $\begin{cases} \text{fact}(0) = 1 \\ \text{fact}(x+1) = (x+1) * \text{fact}(x). \end{cases}$]

(ii) Ackermann's function.

$$\begin{aligned} &\text{rec} f. \lambda z. \text{ split } z \text{ as } (x, y) \text{ in} \\ &\quad \text{if } x = \underline{0} \text{ then } y + \underline{1} \text{ else} \\ &\quad\quad \text{if } y = \underline{0} \text{ then } f(x - \underline{1}, \underline{1}) \text{ else} \\ &\quad\quad\quad f(x - \underline{1}, f(x, y - \underline{1})) \end{aligned}$$

[Cf. $\begin{cases} \text{ack}(0, y) = y + 1 \\ \text{ack}(x+1, 0) = \text{ack}(x, 1) \\ \text{ack}(x+1, y+1) = \text{ack}(x, \text{ack}(x+1, y)) \end{cases}$

(iii) The infinite list $(0,(1,(2,(3,(\cdots)))))$.

$$\text{letrec } f(x) = (x, f(x+\underline{1})) \text{ in } f\underline{0}$$

(iv) A term with a type error

$$\underline{1} + (\underline{0}, \underline{0})$$

(v) Terms can involve self-application!

$$\lambda f.\ (\lambda x.\ f(x\,x))(\lambda x.\ f(x\,x))$$

2.4. Free & bound variable occurrences

(i) A <u>binding occurrence</u> of $x \in Var$ in $M \in Term$ is an occurrence of x in the syntax tree of M in a subterm (subtree) of the form

$$\begin{array}{ll} & split(M_1, x, x', M_2) \\ \underline{or} & split(M_1, x', x, M_2) \\ \underline{or} & \lambda(x, M_2) \\ \underline{or} & rec(x, M_2) \end{array}$$

The <u>scope</u> of the occurrence is the subterm M_2.

(ii) A <u>bound occurrence</u> of x in M is a non-binding occurrence of x within the scope of some binding occurrence of x.

(iii) A <u>free occurrence</u> of x in M is one which is neither binding nor bound.

Eg $\quad split\ \overset{(iii)}{x}\ as\ (\overset{(i)}{y}, \overset{(i)}{z})\ in\ \lambda \overset{(i)}{y}.\ \overset{(ii)}{z}\ \overset{(iii)}{x}\ \overset{(ii)}{y}$

(ie. $split(x, y, z, \lambda(y, app(app(z, x), y)))$.)

2.5 Definition: (Substitution)

Given $\begin{cases} \text{terms} & N, M_1, \ldots, M_n \\ \text{variables} & x_1, \ldots, x_n \end{cases}$ $(n \geq 1)$ <u>all distinct</u>

the term

$$N[M_1/x_1, \ldots, M_n/x_n]$$

("the result of simultaneously substituting M_i for all free occurrences of x_i in M")

is defined (inductively) according to the structure of M:

(i) case N is $y \in Var$:

$$N[\vec{M}/\vec{x}] \text{ is } \begin{cases} M_i & \text{if } y = x_i, \text{ some } i \\ y & \text{otherwise} \end{cases}$$

(ii) case N is $c \in Const$:

$$N[\vec{M}/\vec{x}] \text{ is } c$$

(iii) Case N is $op(N_1, N_2)$:

$$N[\vec{M}/\vec{x}] \text{ is } op(N_1[\vec{M}/\vec{x}], N_2[\vec{M}/\vec{x}])$$

(iv) Cases N is $if(N_1, N_2, N_3)$, $pair(N_1, N_2)$, $app(N_1, N_2)$; are similar to case (iii).

(v) Case N is $split(N_1, y, y', N_2)$:

2.5 Notation (Substitution)

Given

a list $\vec{M} = M_1, \dots, M_n$ of terms

a list $\vec{x} = x_1, \dots, x_n$ of distinct variables

a term N

Let

$$N[\vec{M}/\vec{x}] \quad \left(\text{also written } N[M_1/x_1, \dots, M_n/x_n]\right)$$

denote the term obtained by simultaneously replacing each free occurrence of x_i by the term M_i in N.

Egs:

(i) If $N = \text{split } x \text{ as } (y,z) \text{ in } \lambda y. z x y$

then

$$N[\underline{4}/x, \underline{3}/y] = \text{split } \underline{4} \text{ as } (y,z) \text{ in } \lambda y. z \underline{4} y$$

(ii) If $N = \lambda x. y$

then

$$N[x/y] = \lambda x. x$$

NB (ii) is an example of capture of a free variable in one of the M_i by a binding occurrence of the same variable in N.

We wish to avoid this happening since it can violate the intended meaning of terms. Eg in (ii)

N is "the function with constant value y

so we expect

$N[^x/_y]$ to be "the function with constant value x", but $N[^x/_y] = \lambda x.x$ is "the identity function" – semantically different.

We can avoid this capture of free variables by variable binders so long as we only ever form a substitution

$$N[\vec{M}/\vec{x}]$$

when the free variables of $\vec{M}$ are disjoint from the binding variables in N

A nice way of enforcing this is via the following, inductively defined relation.

2.6 Definition

The relation $\Gamma \vdash M$

where $\begin{cases} \Gamma \subseteq_{fin} Var & \text{is a finite set of variables} \\ M \text{ is a term} \end{cases}$

is inductively defined by the following rules

[where Γ, x means $\Gamma \cup \{x\}$ with $x \notin \Gamma$

Γ, x, y " $\Gamma \cup \{x, y\}$ with $\begin{cases} x, y \notin \Gamma \\ x \neq y \end{cases}$

etc.]

Rules for $\Gamma \vdash M$:

$$\frac{}{\Gamma, x \vdash x} \qquad \frac{}{\Gamma \vdash c}$$

$$\frac{\Gamma \vdash M_1 \quad \Gamma \vdash M_2}{\Gamma \vdash op(M_1, M_2)}$$

$$\frac{\Gamma \vdash M_1 \quad \Gamma \vdash M_2 \quad \Gamma \vdash M_3}{\Gamma \vdash if(M_1, M_2, M_3)}$$

$$\frac{\Gamma \vdash M_1 \quad \Gamma \vdash M_2}{\Gamma \vdash pair(M_1, M_2)}$$

$$\frac{\Gamma \vdash M_1 \quad \Gamma, x, y \vdash M_2}{\Gamma \vdash split(M_1, x, y, M_2)}$$

$$\frac{\Gamma, x \vdash M}{\Gamma \vdash \lambda(x, M)}$$

$$\frac{\Gamma \vdash M_1 \quad \Gamma \vdash M_2}{\Gamma \vdash app(M_1, M_2)}$$

$$\frac{\Gamma, x \vdash M}{\Gamma \vdash rec(x, M)}$$

2.7 Lemma

(i) If $\Gamma \vdash M$ then Γ contains all variables with free occurrences in M and does not contain any variable with binding or bound occurrences in M.

(ii) If $\Gamma \vdash M$ and $x \notin \Gamma \cup \{\text{binding \& bound vars of } M\}$, then $\Gamma, x \vdash M$.

Proof

by induction on the proof of $\Gamma \vdash M$ (i.e. by Rule Induction (1.3) for $\Gamma \vdash M$). $\square$

Thus if $\Gamma, x_1, \ldots, x_n \vdash N$

and $\Gamma \vdash M_1, \ldots, \Gamma \vdash M_n$

then the substitution $N[\vec{M}/\vec{x}]$ will not involve free variables in $\vec{M}$ (which are $\subseteq \Gamma$) being captured by binding variables in N. In this case, it is not hard to see that $\Gamma \vdash N[\vec{M}/\vec{x}]$ holds.

2.8 Remark

Here is a formal, inductive definition of the relation of substitution

$$\Gamma \vdash N[\vec{M}/\vec{x}] = N'$$

where $\Gamma, \vec{x} \vdash N$ and $\Gamma \vdash M_1, \ldots, \Gamma \vdash M_n$, $\Gamma \vdash N'$:

Inductive definition of substitution.

$$\frac{}{\Gamma \vdash y[\vec{M}/\vec{x}] = y}\ (y \in \Gamma) \qquad \frac{}{\Gamma \vdash x_i[\vec{M}/\vec{x}] = M_i}$$

$$\frac{}{\Gamma \vdash c[\vec{M}/\vec{x}] = c}$$

$$\frac{\Gamma \vdash N_1[\vec{M}/\vec{x}] = N_1' \quad \Gamma \vdash N_2[\vec{M}/\vec{x}] = N_2'}{\Gamma \vdash op(N_1, N_2)[\vec{M}/\vec{x}] = op(N_1', N_2')}$$

– similar rules for $N = if(N_1, N_2, N_3), pair(N_1, N_2), app(N_1, N_2)$.

$$\frac{\Gamma \vdash N_1[\vec{M}/\vec{x}] = N_1' \quad \Gamma, y, z \vdash N_2[\vec{M}/\vec{x}] = N_2'}{\Gamma \vdash split(N_1, y, z, N_2)[\vec{M}/\vec{x}] = split(N_1', y, z, N_2')}$$

$$\frac{\Gamma, y \vdash N_1[\vec{M}/\vec{x}] = N_1'}{\Gamma \vdash \lambda(y, N_1)[\vec{M}/\vec{x}] = \lambda(y, N_1')}$$

– similar rule for $N = rec(y, N_1)$

<u>Note</u>: it is not hard to prove (by Rule Induction) that

$\Gamma, \vec{x} \vdash N$ & $\Gamma \vdash M_1, \ldots, \Gamma \vdash M_n$

$\Rightarrow \exists! N' \quad \Gamma \vdash N[\vec{M}/\vec{x}] = N'$.

"exists a unique"...

Here are some simple properties of substitution, which can be proved by rule induction using 2·8.

2·9 Lemma

(i) $\Gamma \vdash N[x/x] = N$

(ii) If $\Gamma \vdash M$, $\Gamma \vdash N$ and $x \notin \Gamma \cup \{$bound & binding vars of $M\}$ (so that $\Gamma, x \vdash M$ by 2·7(ii)), then

$$\Gamma \vdash N[M/x] = N$$

(iii) If $\Gamma \vdash M$, $\Gamma, x \vdash N$, $\Gamma, x, y \vdash P$,

$\Gamma \vdash N[M/x] = N'$, $\Gamma, x \vdash P[N/y] = P'$,

and $\Gamma \vdash P'[M/x] = P''$,

then

$$\Gamma \vdash P[M/x, N'/y] = P''.$$

(I.e. " $(P[N/y])[M/x] = P[M/x, N[M/x]/y]$ ".)

2.10 Alpha Conversion

General idea: the abstract syntax of terms is still too concrete, in that terms differing only in the names of their bound variables will always be given the same meaning and so should be identified

(Eg. $\lambda x.(x+y)$ vs. $\lambda z.(z+y)$.)

or split M as (y,z) in $(y+z)$

vs split M as (x,y) in $(x+y)$.)

Formal definition

The relation of α-convertibility

$$\Gamma \vdash M \sim_\alpha M'$$

where

$$\Gamma \vdash M \ \& \ \Gamma \vdash M'$$

is defined inductively by the following rules:

Inductive definition of α-conversion

- reflexivity: $$\frac{}{\Gamma \vdash M \sim_\alpha M}$$

- symmetry: $$\frac{\Gamma \vdash M \sim_\alpha M'}{\Gamma \vdash M' \sim_\alpha M}$$

- transitivity: $$\frac{\Gamma \vdash M \sim_\alpha M' \qquad \Gamma \vdash M' \sim_\alpha M''}{\Gamma \vdash M \sim_\alpha M''}$$

- α-conversion axioms:

$$\frac{}{\Gamma \vdash \mathrm{split}(M_1, x, y, M_2) \sim_\alpha \mathrm{split}(M_1, x', y', M_2[x'/x, y'/y])}$$

$$\frac{}{\Gamma \vdash \lambda(x, M) \sim_\alpha \lambda(x', M[x'/x])}$$

$$\frac{}{\Gamma \vdash \mathrm{rec}(x, M) \sim_\alpha \mathrm{rec}(x', M[x'/x])}$$

- congruence rules:

$$\frac{\Gamma \vdash M_1 \sim_\alpha M_1' \qquad \Gamma \vdash M_2 \sim_\alpha M_2'}{\Gamma \vdash \mathrm{op}(M_1, M_2) \sim_\alpha \mathrm{op}(M_1', M_2')}$$

– similar rules for if, pair and app.

$$\frac{\Gamma \vdash M_1 \sim_\alpha M_1' \qquad \Gamma, x, y \vdash M_2 \sim_\alpha M_2'}{\Gamma \vdash \mathrm{split}(M_1, x, y, M_2) \sim_\alpha \mathrm{split}(M_1', x, y, M_2')}$$

– similar rules for λ and rec.

2.11 Definition of $\mathbb{L}$-expressions

Note that $\{(M, M') \mid \Gamma \vdash M \sim_\alpha M'\}$ is an equivalence relation on $\{M \mid \Gamma \vdash M\}$.

Let
$$\mathrm{Exp}(\Gamma) \stackrel{\text{def}}{=} \{M \mid \Gamma \vdash M\} / \sim_\alpha$$
denote the set of equivalence classes.

The elements of $\mathrm{Exp}(\Gamma)$ are called the expressions of $\mathbb{L}$ with free variables $\subseteq \Gamma$.

Convention: we will make no notational distinction between a term and the $\sim_\alpha$-equivalence class it determines. In practice this should not cause confusion,

BUT when defining notions involving $\mathbb{L}$-expressions via terms which represent them, we have to make sure that the definition/operation/etc. on terms is invariant under $\sim_\alpha$.

An example of this is substitution, which is well-defined on expressions because of the following lemma (proved by induction on the proof of $\Gamma, \vec{x} \vdash N \sim_\alpha N'$):

2.12 Lemma

If $\Gamma, \vec{x} \vdash N \sim_\alpha N'$ and $\Gamma' \vdash M_i \sim_\alpha M_i'$, then $\Gamma' \vdash N[\vec{M}/\vec{x}] \sim_\alpha N'[\vec{M'}/\vec{x}]$. $\square$

3. OPERATIONAL SEMANTICS

for the programming language $\mathbb{L}$.

3.1 Definitions

An $\mathbb{L}$ program is a closed $\mathbb{L}$ expression, i.e. an $\mathbb{L}$ expression with no free variables, i.e. an element of $Exp(\Gamma)$ in case $\Gamma = \emptyset$.

We write

$$Prog \stackrel{\text{def}}{=} Exp(\emptyset)$$

for the set of programs.

An $\mathbb{L}$ value, or canonical form is a program represented by a term in one of the following syntactic forms

- constants, c
- pairs, (P, Q) (where $\emptyset \vdash P$ and $\emptyset \vdash Q$)
- function abstractions, $\lambda x. M$ (where $\{x\} \vdash M$).

We write

$$Val$$

for the set of $\mathbb{L}$-values.

3.2 Definition

The evaluation relation

$$P \Downarrow V \qquad (P \in Prog, V \in Val)$$

is inductively defined by the following rules:

Rules for $\Downarrow$

$$(\Downarrow\text{VAL}) \qquad \frac{}{V \Downarrow V}$$

$$(\Downarrow\text{OP}) \qquad \frac{P_1 \Downarrow \underline{n_1} \qquad P_2 \Downarrow \underline{n_2}}{P_1 \text{ op } P_2 \Downarrow c} \quad (c = \text{value of } n_1 \text{ op } n_2)$$

$$(\Downarrow\text{IF}_1) \qquad \frac{B \Downarrow \text{true} \qquad P \Downarrow V}{(\text{if } B \text{ then } P \text{ else } Q) \Downarrow V}$$

$$(\Downarrow\text{IF}_2) \qquad \frac{B \Downarrow \text{false} \qquad Q \Downarrow V}{(\text{if } B \text{ then } P \text{ else } Q) \Downarrow V}$$

$$(\Downarrow\text{SPLIT}) \qquad \frac{P \Downarrow (P_1, P_2) \qquad M[P_1/x, P_2/y] \Downarrow V}{(\text{split } P \text{ as } (x,y) \text{ in } M) \Downarrow V}$$

$$(\Downarrow\text{APP}) \qquad \frac{P \Downarrow \lambda x.M \qquad M[Q/x] \Downarrow V}{P\,Q \Downarrow V}$$

$$(\Downarrow\text{REC}) \qquad \frac{M[\text{rec}\, x.M / x] \Downarrow V}{\text{rec}\, x.M \Downarrow V}$$

3.3 Example

Consider $F = \mathrm{rec} f.\ \lambda x.M$

where $M = \text{if } x \leq \underline{0} \text{ then } \underline{1} \text{ else } x * f(x - \underline{1})$

Claim that for all $n \geq 0$ $\quad F\underline{n} \Downarrow \underline{n!}$.

For this it suffices to show for all $n \geq 0$ that

(3.3.1) $\quad \forall P \in \mathrm{Prog}.\ (P \Downarrow \underline{n} \Rightarrow FP \Downarrow \underline{n!})$

and we can do this by induction on n.

First note that

$(\lambda x.M)[F/f] \Downarrow \lambda x.M[F/f]$ by $(\Downarrow \mathrm{VAL})$

so $F \Downarrow \lambda x.M[F/f]$ by $(\Downarrow \mathrm{REC})$

so by $(\Downarrow \mathrm{APP})$

(3.3.2) $\quad FP \Downarrow \underline{m}$ if $M[F/f, P/x] \Downarrow \underline{m}$

Then if $P \Downarrow \underline{0}$, $\quad P \leq \underline{0} \Downarrow \mathrm{true}$ by $(\Downarrow \mathrm{OP})$

so $M[F/f, P/x] \Downarrow \underline{1}$ by $(\Downarrow \mathrm{VAL})$ & $(\Downarrow \mathrm{IF}_1)$.

so $FP \Downarrow \underline{0!}$ by (3.3.2), as required for case $n=0$ of 3.3.1.

Suppose 3.3.1 holds for $n \geq 0$ and that $P \Downarrow \underline{n+1}$.

Then $P - \underline{1} \Downarrow \underline{n}$, so by hypothesis $F(P-\underline{1}) \Downarrow \underline{n!}$

so $P * F(P-1) \Downarrow \underline{(n+1)!}$. But also $P \leq \underline{0} \Downarrow \mathrm{false}$,

so all in all

$$M[F/f, P/x] \Downarrow \underline{(n+1)!}$$

as required.

Remark

The definitions of the evaluation relation $\Downarrow$ (and the transition relation $\rightarrow$ in 3.5, below) is an example of of <u>structural operational semantics</u> (SOS) in the sense of Plotkin: in any proof of $P \Downarrow V$ (or of $P \rightarrow P'$) the last rule used is determined by the syntactic structure of P.

<u>Further reading</u> on SOS

M. Hennessy, "The Semantics of Programming Languages: An Elementary Introduction using Structural Operational Semantics", Wiley, 1990

G. Kahn, "Natural Semantics". In K. Fuchi & M. Nivat (eds), "Programming of Future Generation Computers", North-Holland, 1988, pp 237-258.

G.D. Plotkin, "A structural approach to operational semantics", Report DAIMI FN-19, Aarhus Univ., 1981.

A full-scale example:

R. Milner, M. Tofte & R. Harper, "The Definition of Standard ML", MIT Press, 1990.

R. Milner & M. Tofte, "Commentary on Standard ML", MIT Press, 1991.

3.4 Proposition (Determinacy of evaluation)

If $P \Downarrow V$ and $P \Downarrow V'$, then $V = V'$.

Proof

Use Rule Induction for $\Downarrow$: check that
$\{(P, V) \mid P \Downarrow V \,\&\, \forall V' (P \Downarrow V' \Rightarrow V = V')\}$
is closed under the rules in 3.2.

$\square$

Thus $\{(P, V) \mid P \Downarrow V\}$ is a partial function from programs to values.
[Note: it is definitely not a total function
Eg $\nexists V$ ($\mathrm{rec}\,x.x \Downarrow V$). Why?]

The evaluation relation provides a convenient formulation for reasoning about general properties of this partial function, but it is less convenient for calculating the value of the partial function (if any) at particular programs. We will give a more concrete description of evaluation in terms of iterated steps of computation.

3.5 Definition

The transition relation

$$P \rightarrow Q \qquad (P, Q \in \mathrm{Prog})$$

is inductively defined by the following rules:

<u>Rules for $\rightarrow$</u>

$$(\rightarrow op_1)\ \frac{P_1 \rightarrow P_1'}{P_1 \ op\ P_2 \rightarrow P_1' \ op\ P_2}$$

$$(\rightarrow op_2)\ \frac{P_2 \rightarrow P_2'}{\underline{n}_1 \ op\ P_2 \rightarrow \underline{n}_1 \ op\ P_2'}$$

$$(\rightarrow op_3)\ \frac{}{\underline{n}_1 \ op\ \underline{n}_2 \rightarrow c} \quad (c = \text{value of } n_1 \ op\ n_2)$$

$$(\rightarrow IF_1)\ \frac{B \rightarrow B'}{(\text{if } B \text{ then } P \text{ else } Q) \rightarrow (\text{if } B' \text{ then } P \text{ else } Q)}$$

$$(\rightarrow IF_2)\ \frac{}{(\text{if true then } P \text{ else } Q) \rightarrow P}$$

$$(\rightarrow IF_3)\ \frac{}{(\text{if false then } P \text{ else } Q) \rightarrow Q}$$

$$(\rightarrow SPLIT_1)\ \frac{P \rightarrow P'}{(\text{split } P \text{ as } (x,y) \text{ in } M) \rightarrow (\text{split } P' \text{ as } (x,y) \text{ in } M)}$$

$$(\rightarrow SPLIT_2)\ \frac{}{(\text{split } (P_1, P_2) \text{ as } (x,y) \text{ in } M) \rightarrow M[P_1/x, P_2/y]}$$

$$(\rightarrow APP_1)\ \frac{P \rightarrow P'}{PQ \rightarrow P'Q}$$

$$(\rightarrow APP_2)\ \frac{}{(\lambda x. M)Q \rightarrow M[Q/x]}$$

$$(\rightarrow REC)\ \frac{}{\text{rec}\, x. M \rightarrow M[\text{rec}\, x. M/x]}$$

3.6 Proposition (Determinacy of $\to$)

If $P \to P'$ and $P \to P''$, then $P' = P''$.

Proof

Rule induction for $\to$: cf proof of Proposition 3.4. $\square$

3.7 Proposition

For all $P \in Prog$ and $V \in Val$,

$$P \Downarrow V \iff P \to^* V$$

(Recall from Example 1.4 that $\to^*$ denotes the reflexive-transitive closure of the relation $\to$.)

Proof (outline)

(i) Show that $\{(P, V) \mid P \to^* V\}$ is closed under the rules defining $\Downarrow$. Hence by Rule Induction

$$P \Downarrow V \Rightarrow P \to^* V.$$

(ii) Show that $\{(P, P') \mid \forall V (P' \Downarrow V \Rightarrow P \Downarrow V)\}$ is closed under the rules defining $\to$. Hence by Rule Induction

$$P \to P' \Rightarrow \forall V (P' \Downarrow V \Rightarrow P \Downarrow V)$$

Since $\{(P, P') \mid \forall V (P' \Downarrow V \Rightarrow P \Downarrow V)\}$ is a reflexive & transitive relation, it follows that

$$P \to^* P' \Rightarrow \forall V (P' \Downarrow V \Rightarrow P \Downarrow V)$$

Taking $P' = V$, for which we have $V \Downarrow V$, we get

$$P \to^* V \Rightarrow P \Downarrow V.$$

$\square$

3.8 Divergence

The transition relation $\rightarrow$ allows us to analyse the set $\{P \in Prog \mid \nexists V \in Val\,(P \Downarrow V)\}$ of programs which do not evaluate to canonical form.

For, by Proposition 3.6, every program P determines a unique computation sequence

$$P \stackrel{def}{=} P_0 \rightarrow P_1 \rightarrow P_2 \rightarrow \cdots$$

of maximal length. If the length is ω, we say P diverges, and write $P \rightarrow^{\omega}$. If it is finite, say

$$P = P_0 \rightarrow P_1 \rightarrow \cdots \rightarrow P_n \qquad (n \geq 0)$$

then P_n is terminal, ie $\nexists P'(P_n \rightarrow P')$.

Note that any $V \in Val$ is terminal. However there are "stuck" programs, ie. P which are terminal but not in canonical form

Eg of a stuck program : $\underline{0} + true$

Eg of a divergent program : $rec\ x.\ x$

Remark It is in fact possible to define the set $\{P \in Prog \mid P \text{ diverges}\}$ just starting with $\Downarrow$. One can show (exercise) that it coincides with the set $\{P \in Prog \mid P \Uparrow\}$ co-inductively defined (cf. 1.11) by the following rules :

Rules co-inductively defining divergence, $\Uparrow$

$$-\ \frac{P_1 \Uparrow}{P_1 \, op \, P_2 \Uparrow} \qquad -\ \frac{P_2 \Uparrow}{P_1 \, op \, P_2 \Uparrow}\ (P_1 \Downarrow \underline{n}_1)$$

$$-\ \frac{B \Uparrow}{(\text{if } B \text{ then } P \text{ else } Q) \Uparrow}$$

$$-\ \frac{P \Uparrow}{(\text{if } B \text{ then } P \text{ else } Q) \Uparrow}\ (B \Downarrow \text{true}) \qquad -\ \frac{Q \Uparrow}{(\text{if } B \text{ then } P \text{ else } Q) \Uparrow}\ (B \Downarrow \text{false})$$

$$-\ \frac{P \Uparrow}{(\text{split } P \text{ as } (x,y) \text{ in } M) \Uparrow} \qquad -\ \frac{M[P_1/x, P_2/y] \Uparrow}{(\text{split } P \text{ as } (x,y) \text{ in } M) \Uparrow}\ (P \Downarrow (P_1, P_2))$$

$$-\ \frac{P \Uparrow}{P Q \Uparrow} \qquad -\ \frac{M[Q/x] \Uparrow}{P Q \Uparrow}\ (P \Downarrow \lambda x. M)$$

$$-\ \frac{M[\text{rec}\, x. M / x] \Uparrow}{\text{rec}\, x. M \Uparrow}$$

(Each rule is labelled with a "−" to remind you that the are being used to define a set co-inductively, i.e. we are interested in the greatest post-fixed point of the associated monotone operator.)

Thus the Principle of Rule Co-Induction (Lemma 1·12) in this case yields the following method for proving divergence:

To prove $P\Uparrow$, it suffices to show $P \in D$ for some $D \subseteq Prog$ satisfying:

- $V \notin D$ (any $V \in Val$)
- $(P_1 \ op\ P_2) \in D \Rightarrow P_1 \in D$ or $(\exists n (P_1 \Downarrow \underline{n})\ \&\ P_2 \in D)$
- $(\text{if } B \text{ then } P_1 \text{ else } P_2) \in D \Rightarrow B \in D$ or $(B \Downarrow true\ \&\ P \in D)$ or $(B \Downarrow false\ \&\ Q \in D)$
- $(\text{split } P \text{ as } (x,y) \text{ in } M) \in D \Rightarrow P \in D$ or $\exists P_1, P_2 (P \Downarrow (P_1, P_2)\ \&\ M[P_1/x, P_2/y] \in D)$
- $PQ \in D \Rightarrow P \in D$ or $\exists x, M (P \Downarrow \lambda x.M\ \&\ M[Q/x] \in D)$
- $rec\,x.M \in D \Rightarrow M[rec\,x.M/x] \in D$.

<u>NB</u> In particular cases D might be quite small.
<u>Eg</u> $D = \{rec\,x.x\}$ suffices to witness that $(rec\,x.x)\Uparrow$.

3.9 Remark

The rules for $\Downarrow$ (and for $\to$) embody certain choices about how to evaluate function application and pair-destructors: both rules are non-strict (or "call-by-name") in that arguments are substituted for parameters without being evaluated.

The strict (or "call-by-value") versions would be

$$(3.9.1)\quad \frac{P \Downarrow \lambda x.M \qquad Q \Downarrow V \qquad M[V/x] \Downarrow V'}{P\,Q \Downarrow V'}$$

$$(3.9.2)\quad \frac{P \Downarrow (V_1, V_2) \qquad M[V_1/x, V_2/y] \Downarrow V}{\text{split } P \text{ as } (x,y) \text{ in } M \Downarrow V}$$

plus the definition of value is changed so that (P_1, P_2) is a value iff (inductively) P_1 & P_2 are values — hence we also need the following rule:

$$(3.9.3)\quad \frac{P_1 \Downarrow V_1 \qquad P_2 \Downarrow V_2}{\langle P_1, P_2\rangle \Downarrow \langle V_1, V_2\rangle}.$$

Eg With $\Downarrow$ as in Definition 3.2, we have

$$(\lambda x.\,\mathrm{true})(\mathrm{rec}\,x.x) \Downarrow \mathrm{true}$$

whereas with the strict rule (3.9.1) we get

$$\nexists V\left((\lambda x.\,\mathrm{true})(\mathrm{rec}\,x.x) \Downarrow V \right)$$

(because $\nexists V(\mathrm{rec}\,x.x \Downarrow V)$).

4. OBSERVATIONAL REFINEMENT & EQUIVALENCE for the programming language $\mathbb{L}$.

Recall the general idea of $\lesssim$ and $\simeq$ from the Introduction: for $\mathbb{L}$ expressions M, M',

$M \lesssim M'$ if for all programs $P[M]$ involving occurrences of M (in the syntax tree of a term representing the program up to α-equivalence), any observable result of evaluating $P[M]$ is an observable result of evaluating $P[M']$ — where the latter is $\boxed{\text{"}P[M] \text{ with occurrences of } M \text{ replaced by } M'\text{"}}$

<u>Technical problem</u>: this notion is not well-defined on α-equivalence classes of terms, so goes against the Convention in 2.11.

<u>Eg</u>: the "context" $P[-] \stackrel{\text{def}}{=} \lambda x. -$ should be the same as (α-convertible with) $\lambda y. -$.
But replacing $-$ by x yields different results up to α-equivalence ($\lambda x. x$ in the first case and $\lambda y. x$ in the second).

<u>Technical solution</u>: we will maintain the convention of working up-to-α-equivalence by introducing <u>function variables</u> and <u>substitution of meta-abstractions for function variables.</u>

Terminology: in the literature "terms with holes" like $\lambda x. -$ are called contexts. Hence observational equivalence is often called contextual equivalence.

4.1 Definitions

Fix a countably infinite set FVar of function variables $\xi, \xi', \ldots$ (disjoint from $\mathrm{Var} \cup \mathrm{Const} \cup \mathrm{Op}$), and a function

$$ar : \mathrm{FVar} \to \mathbb{N}$$

assigning to each $\xi \in \mathrm{FVar}$ its arity $ar(\xi) \geq 0$. (We assume $\{\xi \in \mathrm{FVar} \mid ar(\xi) = n\}$ is countably infinite, for each $n \in \mathbb{N}$.)

The set Term^* of extended $\mathbb{L}$-terms is inductively defined by rules as in 2.1 plus the rule

$$\frac{M_1 \in \mathrm{Term}^* \quad \cdots \quad M_n \in \mathrm{Term}^*}{\xi(M_1, \ldots, M_n) \in \mathrm{Term}^*} \quad (ar(\xi) = n)$$

The relation $\Gamma \vdash^* M$

where $\begin{cases} \Gamma \subseteq_{fin} \mathrm{Var} \cup \mathrm{FVar} \\ M \in \mathrm{Term}^* \end{cases}$

is inductively defined by the rules in 2.6 plus the rule

$$\frac{\Gamma, \xi \vdash^* M_1 \quad \cdots \quad \Gamma, \xi \vdash^* M_n}{\Gamma, \xi \vdash^* \xi(M_1, \ldots, M_n)} \; (ar(\xi) = n)$$

Note: Term $\subseteq$ Term* and

$$\Gamma \vdash M \Rightarrow \Gamma \vdash^* M \qquad (\Gamma \subseteq_{fin} Var,\ M \in Term).$$

The formal, inductive definition of substitution of extended terms for variables:

$$\Gamma \vdash^* N[\vec{M}/\vec{x}] = N'$$

is just as in 2.8, except that it is defined for those $\Gamma \subseteq Var \cup FVar$, $\vec{M}, N, N' \in Term^*$ satisfying

$$\Gamma, \vec{x} \vdash^* N, \quad \Gamma \vdash^* M_i, \ldots, \text{ and } \quad \Gamma \vdash^* N'.$$

4.2 Definition (Substitution of meta-abstractions for fn. vars.)

If $\Gamma, \vec{x} \vdash^* M$ where $\vec{x} = x_1, \ldots, x_n$ is a list of distinct variables, then

$$(\vec{x})M$$

is a meta-abstraction (with free variables & function variables $\subseteq \Gamma$). We wish to define the result of substituting a meta-abstraction $(\vec{x})M$ for a function variable ξ in an extended term N, denoted $N[{}^{(\vec{x})M}/_{\xi}]$.

The key case is when N is of the form $\xi(M_1,\ldots,M_n)$, where we take $N[(\vec{x})M/\xi]$ to be $M[\vec{M}/\vec{x}]$.

Formally, we inductively define a relation

$$\Gamma \vdash^* N[(\vec{x})M/\xi] = N'$$

where $\begin{cases} \Gamma, \vec{x} \vdash^* M \\ \Gamma, \xi \vdash^* N \\ \Gamma \vdash^* N' \end{cases}$

by rules like those in 2.8 for the cases that N is a constant, operator-form, conditional, pair, application, pair-destructor or recursively defined (extended) term, plus the following rules for the cases that N is a variable or a function variable applied to extended terms:

$$\frac{}{\Gamma \vdash^* y[(\vec{x})M/\xi] = y} \quad (y \in \Gamma)$$

$$\frac{\Gamma \vdash^* N_1[(\vec{x})M/\xi] = N_1' \quad \cdots \quad \Gamma \vdash^* N_m[(\vec{x})M/\xi] = N_m'}{\Gamma \vdash^* \xi'(N_1,\ldots,N_m)[(\vec{x})M/\xi] = \xi'(N_1',\ldots,N_m')} \quad (\xi' \in \Gamma)$$

$$\frac{\Gamma \vdash^* N_1[(\vec{x})M/\xi] = N_1' \quad \cdots \quad \Gamma \vdash^* N_n[(\vec{x})M/\xi] = N_n'}{\Gamma \vdash^* \xi(N_1,\ldots,N_n)[(\vec{x})M/\xi] = M[N_1'/x_1,\ldots,N_n'/x_n]}$$

(<u>NB</u>: in the second rule, since $\xi' \in \Gamma$, necessarily $\xi' \neq \xi$.)

4.3 Remarks

(i) The relation $\Gamma \vdash^* N[(\vec{x})M/\xi] = N'$ is the graph of a function, i.e.

$$\Gamma, \vec{x} \vdash^* M \ \& \ \Gamma, \xi \vdash^* N \Rightarrow \exists!\, N' (\Gamma \vdash^* N[(\vec{x})M/\xi] = N')$$

(ii) It is possible to define <u>simultaneous</u> substitution of meta-abstractions for function variables, but we won't bother to do so here.

4.4 Lemma (cf 2.9(iii))

If

$$\Gamma, \vec{x} \vdash^* M \qquad \Gamma, \xi, \vec{y} \vdash^* N \qquad \Gamma, \xi' \vdash^* P \qquad (\xi \neq \xi')$$

$$\Gamma, \vec{y} \vdash^* N[(\vec{x})M/\xi] = N'$$

$$\Gamma, \xi \vdash^* P[(\vec{y})N/\xi'] = P'$$

$$\Gamma \vdash^* P'[(\vec{x})M/\xi] = P''$$

then

$$\Gamma \vdash^* P[(\vec{y})N'/\xi'] = P''.$$

(I.e. "$(P[(\vec{y})N/\xi'])[(\vec{x})M/\xi] = P[(\vec{y})N[(\vec{x})M/\xi]/\xi']$")

Proof

By induction on the proof of $\Gamma, \xi' \vdash^* P$. $\square$

4.5 Definition : α-conversion

The equivalence relation of <u>α-convertibility</u>

$$\Gamma \vdash^* M \sim_\alpha M' \qquad (\Gamma \vdash^* M \ , \Gamma \vdash^* M')$$

is defined for extended terms as in 2·10 (with a "congruence" rule for each $\xi \in FVar$).

For each $\Gamma \subseteq_{fin} Var \cup FVar$, let

$$Exp^*(\Gamma) = \{ M \in Term^* \mid \Gamma \vdash^* M \} / \sim_\alpha$$

denote the set of equivalence classes. The elements of $Exp^*(\Gamma)$ are called the <u>extended $\mathbb{L}$ expressions</u> with free variables and function variables $\subseteq \Gamma$.

<u>Note</u> : When $\Gamma \subseteq_{fin} Var$, $Exp(\Gamma) = Exp^*(\Gamma)$.

Thus $\mathbb{L}$ expressions are the special case of extended $\mathbb{L}$ expressions containing no function variables.

The analogue of Lemma 2·12 holds, and so substitution of meta-abstractions for function variables determines a well-defined function

$$\left.\begin{array}{l} M \in Exp^*(\Gamma, \vec{x}) \\ N \in Exp^*(\Gamma, \xi) \end{array}\right\} \mapsto N[(\vec{x})M/\xi] \in Exp^*(\Gamma)$$

(<u>NB</u> if $\Gamma \subseteq Var$, then necessarily M and $N[(\vec{x})M/\xi]$ are expressions rather than extended expressions.)

4.6 Example

When $\begin{cases} N = \lambda x.\,\xi(x) \in \mathrm{Exp}^*(\xi) \\ M = x \in \mathrm{Exp}(x) \end{cases}$

then

$$N[{}^{(x)M}/_{\xi}] = \lambda x.x \quad (= \lambda y.y\ ,\text{etc.})$$

$\lambda x.\,\xi(x)$ is our version of the "context" $\lambda x.-$.
Note that we cannot get by interpreting "holes", $-$, as variables: there is no substitution we can make for y in $\lambda x.y$ which yields $\lambda x.x$ up to α-conversion.

With these preliminaries about function variables out of the way, we get back to defining the notions of observational refinement and observational equivalence for $\mathbb{L}$ expressions ...

4.6 Definitions

Let $\mathrm{Obs} = \mathrm{Const} \cup \{\mathrm{pair}, \lambda\}$. The <u>observable form</u> $\mathrm{obs}(V) \in \mathrm{Obs}$ of a value $V \in \mathrm{Val}$ is inductively defined by

$$\begin{cases} \mathrm{obs}(c) = c & (c \in \mathrm{Const}) \\ \mathrm{obs}((P,Q)) = \mathrm{pair} & \\ \mathrm{obs}(\lambda x.M) = \lambda & \end{cases}$$

For each $\Gamma = \{x_1, \ldots, x_n\} \subseteq_{fin} \mathrm{Var}$, the relation of <u>observational refinement</u>

$$\Gamma \vdash M \lesssim M' \qquad (M, M' \in \mathrm{Exp}(\Gamma))$$

is defined to hold iff

for all $P \in \mathrm{Exp}^*(\xi)$ (where $\xi \in \mathrm{FVar}$ & $ar(\xi) = n$), and all $V \in \mathrm{Val}$, if

$$P[(x_1, \ldots, x_n)M/\xi] \Downarrow V$$

then

$$P[(x_1, \ldots, x_n)M'/\xi] \Downarrow V'$$

for some $V' \in \mathrm{Val}$ with $\mathrm{obs}(V) = \mathrm{obs}(V')$.

The relation of <u>observational equivalence</u>

$$\Gamma \vdash M \cong M' \qquad (M, M' \in \mathrm{Exp}(\Gamma))$$

is defined to hold iff

$$\Gamma \vdash M \lesssim M' \text{ and } \Gamma \vdash M' \lesssim M$$

4.7 Lemma

Observational refinement is a preorder, i.e. it is reflexive

$$\Gamma \vdash M \lesssim M \qquad (\text{all } M \in Exp(\Gamma))$$

and transitive

$$\Gamma \vdash M \lesssim M' \ \& \ \Gamma \vdash M' \lesssim M'' \Rightarrow \Gamma \vdash M \lesssim M''.$$

Observational equivalence is an equivalence relation.

Proof

Immediate from the definitions. $\square$

4.8 Proposition

If $\Gamma, \vec{x} \vdash M \lesssim M'$ and $\Gamma, \xi \overset{*}{\vdash} N$, then $\Gamma \vdash N[(\vec{x})M/\xi] \lesssim N[(\vec{x})M'/\xi]$.

Proof Suppose $\vec{x} = x_1, \ldots, x_n$ and $\Gamma = \{\vec{y}\}$ where $\vec{y} = y_1, \ldots, y_m$. For any $P \in Exp^*(\xi')$ where $ar(\xi') = m$, choosing $\xi'' \in FVar$ with $ar(\xi'') = m+n$, define

$$N' \overset{\text{def}}{=} N[(\vec{x})\ \xi''(\vec{y}, \vec{x})/\xi] \in Exp^*(\vec{y}, \xi'')$$

$$P' \overset{\text{def}}{=} P[(\vec{y})N'/\xi'] \in Exp^*(\xi'')$$

Then by Lemma 4.4

$$N'[(\vec{y}\vec{x})M/\xi''] = N[(\vec{x})\ \xi''(\vec{y}, \vec{x})[(\vec{y}\vec{x})M/\xi'']/\xi]$$

$$= N[(\vec{x})M/\xi]$$

and hence

$$P'[(\vec{y}\vec{x})M/\xi''] = P[(\vec{y})N'[(\vec{y}\vec{x})M/\xi'']/\xi']$$

$$= P[(\vec{y})N[(\vec{x})M/\xi]/\xi']$$

and similarly with M' in place of M. Hence if

$$P[(\vec{y})N[(\vec{x})M/\xi]/\xi'] \Downarrow V$$

then

$$P'[(\vec{y}\vec{x})M/\xi''] \Downarrow V$$

So since $\vec{y}\vec{x} \vdash M \lesssim M'$, there is some V' with $obs(V') = obs(V)$ and

$$P'[(\vec{y}\vec{x})M'/\xi''] \Downarrow V'$$

i.e. with

$$P[(\vec{y})N[(\vec{x})M'/\xi]/\xi'] \Downarrow V'.$$

Since this holds for any $P \in Exp^*(\xi')$, we have $\Gamma \vdash N[(\vec{x})M/\xi] \lesssim N[(\vec{x})M'/\xi]$.

$\square$

<u>4.9 Corollary</u> (Congruence properties of $\lesssim$)

(i) If $\Gamma \vdash M \lesssim M'$ & $\Gamma \vdash N \lesssim N'$, then $\Gamma \vdash (M \text{ op } N) \lesssim (M' \text{ op } N')$

(ii) If $\Gamma \vdash B \lesssim B'$, $\Gamma \vdash M \lesssim M'$ and $\Gamma \vdash N \lesssim N'$, then $\Gamma \vdash (\text{if } B \text{ then } M \text{ else } N) \lesssim (\text{if } B' \text{ then } M' \text{ else } N')$

(iii) If $\Gamma \vdash M \lesssim M'$ and $\Gamma \vdash N \lesssim N'$, then $\Gamma \vdash (M, N) \lesssim (M', N')$.

(iv) If $\Gamma \vdash M \lesssim M'$ and $\Gamma, x, y \vdash N \lesssim N'$, then
$\Gamma \vdash (\text{split } M \text{ as } (x,y) \text{ in } N) \lesssim (\text{split } M' \text{ as } (x,y) \text{ in } N')$.

(v) If $\Gamma, x \vdash M \lesssim M'$, then $\Gamma \vdash \lambda x. M \lesssim \lambda x. M'$.

(vi) If $\Gamma \vdash M \lesssim M'$ and $\Gamma \vdash N \lesssim N'$, then $\Gamma \vdash MN \lesssim M'N'$.

(vii) If $\Gamma, x \vdash M \lesssim M'$, then $\Gamma \vdash \text{rec } x. M \lesssim \text{rec } x. M'$.

(viii) If $\Gamma \vdash M \lesssim M'$ and $\Gamma, x \vdash N \lesssim N'$, then $\Gamma \vdash N[M/x] \lesssim N'[M'/x]$.

Proof

Each of (i)-(vii) follows from 4.8 by choosing a suitable extended expression. Eg for (v) use $\Gamma, \xi \overset{*}{\vdash} \lambda x. \xi(x)$. In the cases of multiple arguments, we change one at a time and apply transitivity (4.7). Eg for (i) we first prove

$$\Gamma \vdash M \lesssim M' \Rightarrow \Gamma \vdash M \text{ op } N \lesssim M' \text{ op } N$$
$$\Gamma \vdash N \lesssim N' \Rightarrow \Gamma \vdash M \text{ op } N \lesssim M \text{ op } N'$$

(by considering $\Gamma, \xi \overset{*}{\vdash} \xi() \text{ op } N$ and $\Gamma, \xi \overset{*}{\vdash} M \text{ op } \xi()$ respectively) and then use transitivity to get (i). Similarly for (ii), (iii), (iv) and (vi).

Finally for (viii), using transitivity, we can split it into proving

$$\Gamma \vdash M \lesssim M' \;\&\; \Gamma, x \vdash N \Rightarrow \Gamma \vdash N[M/x] \lesssim N[M'/x]$$

and $\Gamma \vdash M \;\&\; \Gamma, x \vdash N \lesssim N' \Rightarrow \Gamma \vdash N[M/x] \lesssim N'[M/x]$.

The first implication follows from (i)–(vii) by induction on the proof of $\Gamma, x \vdash N$. The second follows directly from 4.8, using $\Gamma, \xi \overset{*}{\vdash} \xi(M)$. $\square$

Notation.

When $P, Q \in Prog$, we just write $P \lesssim Q$ and $P \simeq Q$ for $\emptyset \vdash P \lesssim Q$ and $\emptyset \vdash P \simeq Q$.

4.10 FACTS about $\lesssim$ and $\simeq$.

(i) If $P \Downarrow V$ then $P \simeq V$. [NB this fact reflects the deterministic nature of evaluation in $\mathbb{L}$.]

(ii) β-conversions:

(a) $(\lambda x. M)\, Q \simeq M[Q/x]$

(b) $(\text{split } (P, Q) \text{ as } (x, y) \text{ in } M) \simeq M[P/x, Q/y]$

(c) $(\text{if true then } P \text{ else } Q) \simeq P$

(d) $(\text{if false then } P \text{ else } Q) \simeq Q$

(iii) Conditional η-conversions:

(a) If $P \Downarrow \lambda x. M$, then $P \simeq \lambda y. P y$

(b) If $P \Downarrow (P_1, P_2)$, then
$(\text{split } P \text{ as } (x, y) \text{ in } M[(x,y)/z]) \simeq M[P/z]$

(c) If $B \Downarrow \text{true}$ or $B \Downarrow \text{false}$, then
$(\text{if } B \text{ then } M[\text{true}/x] \text{ else } M[\text{false}/x]) \simeq M[B/x]$

(iv) extensionality properties:

(a) $\lambda x. M \lesssim \lambda x. M' \iff \forall P\, (M[P/x] \lesssim M'[P/x])$

(b) $(P_1, P_2) \lesssim (Q_1, Q_2) \iff P_1 \lesssim Q_1 \ \& \ P_2 \lesssim Q_2$

(c) $c \lesssim c' \iff c = c'$

(v) least prefixed point property:

(a) $\mathrm{rec}\, x.M \simeq M[\mathrm{rec}\, x.M / x]$,

(b) $M[P/x] \sqsubseteq P \Rightarrow \mathrm{rec}\, x.M \sqsubseteq P$.

(vi) properties of $\Omega \stackrel{\text{def}}{=} \mathrm{rec}\, x.x$:

(a) $\Omega \sqsubseteq P$

(b) $P \simeq \Omega \Leftrightarrow \nexists V (P \Downarrow V)$

(vii) definable lubs and continuity :

Put $\begin{cases} \mathrm{rec}^{(0)} x.M \stackrel{\text{def}}{=} \Omega \\ \mathrm{rec}^{(n+1)} x.M \stackrel{\text{def}}{=} M[\mathrm{rec}^{(n)} x.M / x] \end{cases}$

Then $\mathrm{rec}^{(0)} x.M \sqsubseteq \mathrm{rec}^{(1)} x.M \sqsubseteq \cdots \sqsubseteq \mathrm{rec}\, x.M$,

and $\forall n (\mathrm{rec}^{(n)} x.M \sqsubseteq P) \Rightarrow \mathrm{rec}\, x.M \sqsubseteq P$

More generally, for any $N \in Exp(y)$,

$$N[\mathrm{rec}\, x.M / y] \sqsubseteq P \Leftrightarrow \forall n (N[\mathrm{rec}^{(n)} x.M / y] \sqsubseteq P).$$

These FACTS provide a useful basis for reasoning about observational refinement/equivalence of $\mathbb{L}$ expressions. The problem is that it is difficult to prove these FACTS directly from the definition of $\sqsubseteq$ and $\simeq$, because of the quantification over all "contexts" ($P \in Exp^*(\xi)$) in the definition. In subsequent Sections we introduce various operational and denotational tools for solving this problem.

5. APPLICATIVE (BI)SIMULATIONS

AIM: to show that observational refinement can be characterized as the largest relation between $\mathbb{L}$-programs satisfying

$$
\begin{aligned}
P \sqsubseteq P' \Leftrightarrow\ & \forall c \in Const\ (P \Downarrow c \Rightarrow P' \Downarrow c) \\
& \& \\
& \forall P_1, P_2\ (P \Downarrow (P_1, P_2) \Rightarrow \exists P_1', P_2' \\
& \qquad P' \Downarrow (P_1', P_2')\ \&\ P_1 \sqsubseteq P_1'\ \&\ P_2 \sqsubseteq P_2'\ ) \\
& \& \\
& \forall \lambda x.M\ (P \Downarrow \lambda x.M \Rightarrow \exists \lambda x'.M' \\
& \qquad P' \Downarrow \lambda x'.M'\ \&\ \forall Q (\, M[Q/x] \sqsubseteq M'[Q/x']))
\end{aligned}
$$

Note: that $\sqsubseteq$ satisfies the above biconditional and enjoys the congruence properties in 4.9, is sufficient to establish several of the facts listed in 4.10, namely (i), (ii), (iii), (iv), (v)(a) and (vi).

5.1 Definitions

Given a binary relation $\mathcal{S} \subseteq Prog \times Prog$, let $[\mathcal{S}] \subseteq Prog \times Prog$ be defined by

$$
\begin{aligned}
P\,[\mathcal{S}]\,P' \Leftrightarrow\ & \forall c\,(P \Downarrow c \Rightarrow P' \Downarrow c) \\
& \& \\
& \forall P_1, P_2\,(P \Downarrow (P_1, P_2) \Rightarrow \exists P_1', P_2' \\
& \qquad P' \Downarrow (P_1', P_2')\ \&\ P_1\,\mathcal{S}\,P_1'\ \&\ P_2\,\mathcal{S}\,P_2') \\
& \& \\
& \forall \lambda x.M\,(P \Downarrow \lambda x.M \Rightarrow \exists \lambda x'.M' \\
& \qquad P' \Downarrow \lambda x'.M'\ \&\ \forall Q\,(M[Q/x]\,\mathcal{S}\,M'[Q/x']))
\end{aligned}
$$

<u>Note</u> that $\mathcal{S} \mapsto [\mathcal{S}]$ is a monotone operator on $\mathcal{P}(Prog \times Prog)$ (i.e. $\mathcal{S} \subseteq \mathcal{S}' \Rightarrow [\mathcal{S}] \subseteq [\mathcal{S}']$).

The relation $\lesssim\ \subseteq Prog \times Prog$ of <u>applicative refinement</u> is defined to be the greatest post-fixed point of $\mathcal{S} \mapsto [\mathcal{S}]$ (cf. 1.10).

The relation $\sim\ \subseteq Prog \times Prog$ of <u>applicative equivalence</u> is defined by:

$$P \sim Q \iff P \lesssim Q\ \&\ Q \lesssim P.$$

We extend $\lesssim$ and $\sim$ to all $\mathbb{L}$ expressions (rather than just the closed ones) as follows:

given $\Gamma \subseteq_{fin} Var$, say $\Gamma = \{x_1, \ldots, x_n\}$, and $M, N \in Exp(\Gamma)$, define

$$\Gamma \vdash M \lesssim N \iff \forall P_1,\ldots,P_n \in Prog.\ M[\vec{P}/\vec{x}] \lesssim N[\vec{P}/\vec{x}]$$

$$\Gamma \vdash M \sim N \iff \forall P_1,\ldots,P_n \in Prog.\ M[\vec{P}/\vec{x}] \sim N[\vec{P}/\vec{x}].$$

We aim to prove

Theorem. Applicative refinement (resp. equivalence) coincides with observational refinement (resp. equivalence), ie

$$\begin{cases} \Gamma \vdash M \lesssim N \iff \Gamma \vdash M \eqsim N \\ \Gamma \vdash M \sim N \iff \Gamma \vdash M \simeq N \end{cases}$$

5.2 Lemma

$\lesssim$ is a preoder and (hence) $\sim$ is an equivalence relation.

Proof

It is not hard to check that $\mathcal{S} \mapsto [\mathcal{S}]$ satisfies

(5.2.1) $\quad I \subseteq [I]$

(5.2.2) $\quad [\mathcal{S}'] \circ [\mathcal{S}] \subseteq [\mathcal{S}' \circ \mathcal{S}]$

where $I = \{(P,P) \mid P \in Prog\}$ is the identity binary relation on $Prog$, and

$$\mathcal{S}' \circ \mathcal{S} = \{(P,R) \mid \exists Q (P \mathcal{S} Q \ \& \ Q \mathcal{S}' R)\}$$

is the composition of relations.

Since $\lesssim$ is the greatest post-fixed point of $\mathcal{S} \mapsto [\mathcal{S}]$, (5.2.1) gives $I \subseteq \lesssim$, i.e. $\lesssim$ is relfexive. And (5.2.2) gives $\lesssim \circ \lesssim = [\lesssim] \circ [\lesssim] \subseteq [\lesssim \circ \lesssim]$, so $\lesssim \circ \lesssim \subseteq \lesssim$, ie. $\lesssim$ is transitive. $\square$

5.3 Remark

Since $\lesssim$ is reflexive by 5.2, and since $\lesssim = [\lesssim]$, it follows that $P \lesssim P'$ holds if

$$\forall V \in Val (\, P \Downarrow V \Rightarrow P' \Downarrow V).$$

This serves to establish that FACTS 4.10 (i), (ii), v(a) & (vi) hold for applicative equivalence.

Since $\lesssim$ is defined as a greatest fixed point of a monotone operator, we can formulate a co-induction principle for it (cf lemma 1.12):

5.4 Proposition (Applicative simulations)

$\mathcal{S} \subseteq Prog \times Prog$ is an <u>applicative simulation</u> if it satisfies that for all $P, P' \in Prog$, $P \,\mathcal{S}\, P'$ implies:

- $P \Downarrow c \Rightarrow P' \Downarrow c \qquad (c \in Const)$
- $P \Downarrow (P_1, P_2) \Rightarrow \exists P'_1, P'_2 \,(P' \Downarrow (P'_1, P'_2) \;\&\; P_1 \mathcal{S} P'_1 \;\&\; P_2 \mathcal{S} P'_2)$
- $P \Downarrow \lambda x.M \Rightarrow \exists \lambda x'.M' (\, P' \Downarrow \lambda x'.M \;\&$
  $\forall Q (\, M[Q/x] \,\mathcal{S}\, M'[Q/x'])\,).$

Then for any $P, P' \in Prog$, to prove $P \lesssim P'$ it suffices to show $P \,\mathcal{S}\, P'$ for some applicative simulation $\mathcal{S}$. $\square$

Proof.

Just note that $\mathcal{S}$ is an applicative simulation iff it is a post-fixed point of $[\cdot]$, ie. iff $\mathcal{S} \subseteq [\mathcal{S}]$. So for any such $\mathcal{S}$, $\mathcal{S} \subseteq \lesssim$. $\square$

5.5 Remark (Applicative bisimulations)

To prove $P \sim P'$ clearly it suffices to find applicative simulations $\mathcal{S}$ and $\mathcal{S}'$ with $P \mathcal{S} P'$ (hence $P \lesssim P'$) and $P \mathcal{S}' P'$ (hence $P' \lesssim P$). However $\sim$ can be characterized directly as the greatest post-fixed point of $\mathcal{S} \mapsto \langle \mathcal{S} \rangle$ where

$$\langle \mathcal{S} \rangle \stackrel{\text{def}}{=} [\mathcal{S}] \cap [\mathcal{S}^{\circ}]^{\circ}$$

where $\mathcal{S}^{\circ} = \{(P', P) \mid P \mathcal{S} P'\}$ denotes the opposite of a binary relation $\mathcal{S}$.

(Exercise: prove $\sim$ is the greatest post-fixed point of $\mathcal{S} \mapsto \langle \mathcal{S} \rangle$.)

Thus to prove $P \sim P'$ it suffices to show $P \mathcal{S} P'$ for some $\mathcal{S} \subseteq \langle \mathcal{S} \rangle$. Such $\mathcal{S}$ are called applicative bisimulations. Clearly $\mathcal{S}$ is such a relation iff for all $P, P' \in Prog$, $P \mathcal{S} P'$ implies

- $P \Downarrow c \iff P' \Downarrow c \qquad (c \in Const)$
- $P \Downarrow (P_1, P_2) \Rightarrow \exists P_1', P_2' (P' \Downarrow (P_1', P_2') \;\&\; P_1 \mathcal{S} P_1' \;\&\; P_2 \mathcal{S} P_2')$
- $P' \Downarrow (P_1', P_2') \Rightarrow \exists P_1, P_2 (P \Downarrow (P_1, P_2) \;\&\; P_1 \mathcal{S} P_1' \;\&\; P_2 \mathcal{S} P_2')$

- $P \Downarrow \lambda x.M \Rightarrow \exists \lambda x'.M' \,(P' \Downarrow \lambda x'.M' \;\&$
  $\forall Q (M[Q/x] \mathrel{\mathcal{S}} M'[Q'/x']))$
- $P' \Downarrow \lambda x'.M' \Rightarrow \exists \lambda x.M \,(P \Downarrow \lambda x.M \;\&$
  $\forall Q (M[Q/x] \mathrel{\mathcal{S}} M'[Q'/x']))$.

Here is an example to illustrate the use of applicative bisimulations for establishing applicative equivalences.

<u>5.6 Example</u> (Two descriptions of "$(0,(1,(2,(\cdots))))$")

Let

$$\mathrm{msuc} \stackrel{\mathrm{def}}{=} \mathrm{rec}\, f.\, \lambda x.\, \mathrm{split}\; x \;\mathrm{as}\; (y,z) \;\mathrm{in}\; (y+\underline{1}, f z)$$

$$\mathrm{from} \stackrel{\mathrm{def}}{=} \mathrm{rec}\, f.\, \lambda x.\, (x, f(x+\underline{1}))$$

$$\mathrm{nats} \stackrel{\mathrm{def}}{=} \mathrm{rec}\, x.\, (\underline{0}, \mathrm{msuc}\, x)$$

Prove : $\mathrm{nats} \sim \mathrm{from}\, \underline{0}$

Proof :

For each $n \in \mathbb{N}$, define a program $\mathrm{msuc}^n\, \mathrm{nats}$ as follows :

$$\begin{cases} \mathrm{msuc}^0\, \mathrm{nats} \stackrel{\mathrm{def}}{=} \mathrm{nats} \\ \mathrm{msuc}^{n+1}\, \mathrm{nats} \stackrel{\mathrm{def}}{=} \mathrm{msuc}(\mathrm{msuc}^n\, \mathrm{nats}) \end{cases}$$

Then to see that $\mathrm{nats} \sim \mathrm{from}\, \underline{0}$, it suffices to check that

$$\mathcal{S} \stackrel{\mathrm{def}}{=} \{(\mathrm{msuc}^n\, \mathrm{nats}, \mathrm{from}\, P') \mid n \in \mathbb{N} \;\&\; P' \sim \underline{n}\}$$
$$\cup \{(P, P') \mid \exists n \in \mathbb{N}.\; P \sim \underline{n} \sim P'\}$$

is an applicative bisimulation. This follows from the following easily established facts :

- $P \sim \underline{n} \iff P \Downarrow \underline{n}$
- $P \sim \underline{n} \implies P + \underline{1} \sim \underline{n+1}$
- $\text{from}\ P \Downarrow (P, \text{from}(P+\underline{1}))$
- $\forall n \geq 0 . \exists P . (\ \text{msuc}^n\ \text{nats} \Downarrow (P, \text{msuc}^{n+1}\ \text{nats}) \ \&\ P \sim \underline{n})$

(the last can be proved by induction on $n \in \mathbb{N}$).

We turn now to the proof that applicative refinement coincides with observational refinement. The main step is to establish the congruence properties of $\lesssim$ (cf. Corollary 4.9).

<u>5.7 Proposition</u> (Congruence properties of $\lesssim$)

(i) $\Gamma \vdash M \lesssim M' \ \&\ \Gamma \vdash N \lesssim N' \implies \Gamma \vdash M \ op\ N \lesssim M' \ op\ N'$.

(ii) $\Gamma \vdash B \lesssim B' \ \&\ \Gamma \vdash M \lesssim M' \ \&\ \Gamma \vdash N \lesssim N'$
$\implies \Gamma \vdash (\text{if } B \text{ then } M \text{ else } N) \lesssim (\text{if } B' \text{ then } M' \text{ else } N')$.

(iii) $\Gamma \vdash M \lesssim M' \ \&\ \Gamma \vdash N \lesssim N' \implies \Gamma \vdash (M, N) \lesssim (M', N')$.

(iv) $\Gamma \vdash M \lesssim M' \ \&\ \Gamma, x, y \vdash N \lesssim N'$
$\implies \Gamma \vdash (\text{split } M \text{ as } (x,y) \text{ in } N) \lesssim (\text{split } M' \text{ as } (x,y) \text{ in } N')$.

(v) $\Gamma, x \vdash M \lesssim M' \implies \Gamma \vdash \lambda x . M \lesssim \lambda x . M'$.

(vi) $\Gamma \vdash M \lesssim M' \ \&\ \Gamma \vdash N \lesssim N' \implies \Gamma \vdash M N \lesssim M' N'$.

(vii) $\Gamma, x \vdash M \lesssim M' \implies \Gamma \vdash \text{rec}\, x . M \lesssim \text{rec}\, x . M'$.

The proof of this proposition will be given in the next section.

5.8 Corollary

(i) $\Gamma \vdash M \precsim M'$ & $\Gamma, x \vdash N \Rightarrow \Gamma \vdash N[M/x] \precsim N[M'/x]$.

(ii) $\Gamma \vdash M \precsim M'$ & $\Gamma, x \vdash N \precsim N' \Rightarrow \Gamma \vdash N[M/x] \precsim N'[M'/x]$.

(iii) $\Gamma, \vec{x} \vdash M \precsim M'$ & $\Gamma, \xi \vdash^{*} N \Rightarrow \Gamma \vdash N[(\vec{x})M/\xi] \precsim N[(\vec{x})M'/\xi]$.

Proof

(i) is proved by induction on the proof of $\Gamma, x \vdash N$, using 5.7.

For (ii), first note that by definition of $\precsim$ on open expressions we have

(5.8.1) $\Gamma, x \vdash N \precsim N' \Rightarrow \Gamma \vdash N[M/x] \precsim N'[M/x]$

for all $M \in Exp(\Gamma)$; for if $\Gamma, x \vdash N \precsim N'$ and $\Gamma = \{\vec{x}\}$ say, then

$$\begin{aligned} N[M/x][\vec{P}/\vec{x}] &= N[\vec{P}/\vec{x}, M[\vec{P}/\vec{x}]/x] && \text{by 2-9(iii)} \\ &\precsim N'[\vec{P}/\vec{x}, M[\vec{P}/\vec{x}]/x] \\ &= N'[M/x][\vec{P}/\vec{x}] \end{aligned}$$

so that $\Gamma \vdash N[M/x] \precsim N'[M/x]$. Then (ii) follows from (i) + (5.8.1) + transitivity of $\precsim$.

Finally (iii) is proved by induction on the proof of $\Gamma, \xi \vdash^{*} N$ using 5.7 and using (ii) for the case that N is of the form $\xi(N_1, \dots, N_n)$. $\square$

Recalling the definition (4.6) of the observable form $obs(V)$ of a value V, the fact that $\lesssim = [\lesssim]$ immediately implies

5.9 Lemma

$$P \lesssim P' \;\&\; P \Downarrow V \Rightarrow \exists V' (P' \Downarrow V' \;\&\; obs(V) = obs(V'))$$

$\square$

5.10 Proposition

Applicative refinement entails observational refinement:

$$\Gamma \vdash M \lesssim M' \Rightarrow \Gamma \vdash M \sqsubseteq\!\!\!\sim M'.$$

Proof

Suppose $\Gamma \vdash M \lesssim M'$ and $\Gamma = \{x_1, \ldots, x_n\}$, say. For all $P \in Exp^*(\xi)$ with $ar(\xi) = n$, by 5.8(iii) we have $P[(\vec{x})M/\xi] \lesssim P[(\vec{x})M'/\xi]$. So if $P[(\vec{x})M/\xi] \Downarrow V$, then by 5.9, $P[(\vec{x})M'/\xi] \Downarrow V'$ for some V' with $obs(V) = obs(V')$.

So by definition 4.6, $\Gamma \vdash M \sqsubseteq\!\!\!\sim M'$.

$\square$

5.11 Corollary

4.10 (i), (ii), v(a) and (vi) are all valid.

Proof

Combine Remark 5.3 with Proposition 5.10.

$\square$

5.12 Lemma (cf 4.10(iv) (a) & (b))

(i) $\lambda x.M \sqsubseteq \lambda x'.M' \Rightarrow \forall P \;\; M[P/x] \sqsubseteq M'[P/x']$

(ii) $(P_1, P_2) \sqsubseteq (P_2, Q_2) \Rightarrow \;\; P_1 \sqsubseteq P_2 \;\&\; Q_1 \sqsubseteq Q_2$

Proof

(i) If $\lambda x.M \sqsubseteq \lambda x'.M'$, then for any $P \in Prog$ $(\lambda x.M)P \sqsubseteq (\lambda x'.M')P$, by 4.9(vi). But by 4.10(ii)(a) (proved in 5.11),

$$(\lambda x.M)P \simeq M[P/x] \;\;\&\;\; (\lambda x'.M')P \simeq M'[P/x'].$$

Hence $M[P/x] \simeq (\lambda x.M)P \sqsubseteq (\lambda x'.M')P \simeq M'[P/x']$.

(ii) is similar to (i), but using 4.9(iv) and

$$\begin{cases} (\text{split } (P,Q) \text{ as } (x,y) \text{ in } x) \simeq P \\ (\text{split } (P,Q) \text{ as } (x,y) \text{ in } y) \simeq Q \end{cases}$$

which are instances of 4.10(ii)(b).

□

5.13 Proposition

$\{(Q,Q') \mid Q \sqsubseteq Q'\}$ is an applicative simulation (cf. 5.4).

Proof

Suppose $Q \sqsubseteq Q'$.

If $Q \Downarrow V$; then putting $P = \xi() \in Exp^*(\xi)$, we have $P[()Q/\xi] = Q \Downarrow V$, so

$$Q' = P[()Q'/\xi] \Downarrow V'$$

for some V' with $obs(V) = obs(V')$. Moreover by 4.10(i), $V \simeq Q \sqsubseteq Q' \simeq V'$!

We thus have

$$Q \sqsubseteq Q' \;\&\; Q \Downarrow V \Rightarrow \exists V' (Q' \Downarrow V' \;\&\; obs(V) = obs(V') \;\&\; V \sqsubseteq V').$$

Combining this with Lemma 5.12 gives that $\sqsubseteq$ is an applicative simulation.

$\square$

Putting everything together, we have:

5.14 Theorem (Applicative & Observational refinement coincide)

For all $\Gamma \subseteq_{fin} Var$ and $M, M' \in Exp(\Gamma)$,

$$\Gamma \vdash M \lesssim M' \Leftrightarrow \Gamma \vdash M \sqsubseteq M'.$$

Proof

$\Rightarrow$ is Proposition 5.10.

For $\Leftarrow$, note that by Proposition 5.13

$$Q \sqsubseteq Q' \Rightarrow Q \lesssim Q'$$

for all $Q, Q' \in Prog$. Hence if $\Gamma \vdash M \sqsubseteq M'$ with $\Gamma = \{x_1, \ldots, x_n\}$ say, then by (repeated use of) 4.9 (viii)

$$M[\vec{P}/\vec{x}] \sqsubseteq M'[\vec{P}/\vec{x}]$$

so

$$M[\vec{P}/\vec{x}] \lesssim M'[\vec{P}/\vec{x}]$$

for any $P_1, \ldots, P_n \in Prog$. Hence by Def^n 5.1

$$\Gamma \vdash M \lesssim M'.$$

$\square$

Of the FACTS mentioned in 4.10 we have now proved (i), (ii), (v)(a) and (vi). The conditional η-conversions (iii) follow from (ii) + the congruence properties 4.9. The extensionality properties (iv) follow from Thm 5.14 since $\lesssim$ has these properties almost by definition.

That leaves FACTS (v)(b) and (vii). These properties of rec x.– will follow from the denotational semantics of $\mathbb{L}$, to be given in section 8.

There is one piece of unfinished business – we have not yet given the proof of the congruence properties of applicative refinement, Proposition 5.7...

6: CONGRUENCE PROPERTY OF APPLICATIVE REFINEMENT / EQUIVALENCE

We wish to prove (Proposition 5.7):

(i) $\Gamma \vdash M \lesssim M'$ & $\Gamma \vdash N \lesssim N'$ $\Rightarrow$ $\Gamma \vdash M \; op \; N \lesssim M' \; op \; N'$

(ii) $\Gamma \vdash B \lesssim B'$ & $\Gamma \vdash M \lesssim M'$ & $\Gamma \vdash N \lesssim N'$ $\Rightarrow$

$\Gamma \vdash (\text{if } B \text{ then } M \text{ else } N) \lesssim (\text{if } B' \text{ then } M' \text{ else } N')$

(iii) $\Gamma \vdash M \lesssim M'$ & $\Gamma \vdash N \lesssim N'$ $\Rightarrow$ $\Gamma \vdash (M, N) \lesssim (M', N')$

(iv) $\Gamma \vdash M \lesssim M'$ & $\Gamma, x, y \vdash N \lesssim N'$ $\Rightarrow$

$\Gamma \vdash (\text{split } M \text{ as } (x,y) \text{ in } N) \lesssim \text{split } M' \text{ as } (x,y) \text{ in } N'$

(v) $\Gamma, x \vdash M \lesssim M'$ $\Rightarrow$ $\Gamma \vdash \lambda x. M \lesssim \lambda x. M'$

(vi) $\Gamma \vdash M \lesssim M'$ & $\Gamma \vdash N \lesssim N'$ $\Rightarrow$ $\Gamma \vdash M N \lesssim M' N'$

(vii) $\Gamma, x \vdash M \lesssim M'$ $\Rightarrow$ $\Gamma \vdash \text{rec } x. M \lesssim \text{rec } x. M'$

We will employ a proof method introduced by D. Howe and which appears to be quite general-purpose (i.e. it has been applied successfully to other types of programming language feature).

Reference:

D. Howe, "Equality in Lazy Computation Systems", in Proc. 4th Ann. Symp. Logic in Computer Science, Asilomar (Comp. Soc. Press, Washington, 1989), pp 198-203.

6.1 Definition

A relation

$$\Gamma \vdash M \lesssim^* N \qquad (\Gamma \subseteq_{fin} Var,\ M, N \in Exp(\Gamma))$$

is inductively defined by the following rules:

Rules defining $\lesssim^*$

- $$\frac{}{\Gamma, x \vdash x \lesssim^* N} \; (\Gamma, x \vdash x \lesssim N)$$

- $$\frac{}{\Gamma \vdash c \lesssim^* N} \; (\Gamma \vdash c \lesssim N)$$

- $$\frac{\Gamma \vdash M_1 \lesssim^* M_1' \quad \Gamma \vdash M_2 \lesssim^* M_2'}{\Gamma \vdash (M_1 \, op \, M_2) \lesssim^* N} \; (\Gamma \vdash (M_1' \, op \, M_2') \lesssim N)$$

- $$\frac{\Gamma \vdash B \lesssim^* B' \quad \Gamma \vdash M_1 \lesssim^* M_1' \quad \Gamma \vdash M_2 \lesssim^* M_2'}{\Gamma \vdash (\text{if } B \text{ then } M_1 \text{ else } M_2) \lesssim^* N} \; (\Gamma \vdash (\text{if } B' \text{ then } M_1' \text{ else } M_2') \lesssim N)$$

- $$\frac{\Gamma \vdash M_1 \lesssim^* M_1' \quad \Gamma \vdash M_2 \lesssim^* M_2'}{\Gamma \vdash (M_1, M_2) \lesssim^* N} \; (\Gamma \vdash (M_1, M_2) \lesssim N)$$

- $$\frac{\Gamma \vdash M_1 \lesssim^* M_1' \quad \Gamma, x, y \vdash M_2 \lesssim^* M_2'}{\Gamma \vdash (\text{split } M_1 \text{ as } (x, y) \text{ in } M_2) \lesssim^* N} \; (\Gamma \vdash (\text{split } M_1' \text{ as } (x, y) \text{ in } M_2') \lesssim N)$$

- $$\frac{\Gamma, x \vdash M \lesssim^* M'}{\Gamma \vdash \lambda x. M \lesssim^* N} \; (\Gamma \vdash \lambda x. M' \lesssim N)$$

- $$\frac{\Gamma \vdash M_1 \lesssim^* M_1' \quad \Gamma \vdash M_2 \lesssim^* M_2'}{\Gamma \vdash M_1 M_2 \lesssim^* N} \; (\Gamma \vdash M_1' M_2' \lesssim N)$$

- $$\frac{\Gamma, x \vdash M \lesssim^* M'}{\Gamma \vdash \text{rec } x. M \lesssim^* N} \; (\Gamma \vdash \text{rec } x. M' \lesssim N)$$

6.2 Lemma

(i) $\Gamma \vdash M \lesssim^* N \;\&\; \Gamma \vdash N \lesssim N' \Rightarrow \Gamma \vdash M \lesssim^* N'$

(ii) $\Gamma \vdash M \Rightarrow \Gamma \vdash M \lesssim^* M$

(iii) $\Gamma \vdash M \lesssim N \Rightarrow \Gamma \vdash M \lesssim^* N$

Proof

(i) can be proved by induction on the proof of $\Gamma \vdash M \lesssim^* N$; (ii) by induction on the proof of $\Gamma \vdash M$ (using reflexivity of $\lesssim$). Then (iii) follows from (i) + (ii).

$\square$

6.3 Lemma

$\lesssim^*$ has the properties (i)–(vii) stated for $\lesssim$ in Proposition 5.7.

Proof

This follows immediately from the definition of $\lesssim^*$ (together with reflexivity of $\lesssim$).

$\square$

6.4 Lemma

$$\Gamma \vdash M \lesssim^* M' \;\&\; \Gamma, x \vdash N \lesssim^* N' \Rightarrow \Gamma \vdash N[M/x] \lesssim^* N'[M'/x].$$

Proof

By induction on the proof of $\Gamma, x \vdash N \lesssim^* N'$, using 6.2(iii) and the fact that by definition of $\lesssim$ on open expressions (5.1),

$\Gamma \vdash M \;\&\; \Gamma, x \vdash N \lesssim N' \Rightarrow \Gamma \vdash N[M/x] \lesssim N'[M/x]$.

$\square$

Notation: $P \lesssim^* Q$ means $\emptyset \vdash P \lesssim^* Q$.

6.5 Lemma

For all $c \in Const$, $P_1, P_2, Q \in Prog$ and $M \in Exp(x)$

(i) $c \lesssim^* Q \Rightarrow Q \Downarrow c$

(ii) $(P_1, P_2) \lesssim^* Q \Rightarrow \exists Q_1, Q_2 \,(Q \Downarrow (Q_1, Q_2)$ & $P_1 \lesssim^* Q_1$ & $P_2 \lesssim^* Q_2 \,)$

(iii) $\lambda x.M \lesssim^* Q \Rightarrow \exists \lambda y.N \,(Q \Downarrow \lambda y.N$ & $\forall P.\ M[P/x] \lesssim^* N[P/y] \,)$.

Proof

(i) $c \lesssim^* Q$ must have been deduced from $c \lesssim Q$, in which case $Q \Downarrow c$.

(ii) $(P_1, P_2) \lesssim^* Q$ must have been deduced from

(6.5.1) $P_1 \lesssim^* P_1'$ & $P_2 \lesssim^* P_2'$

for some P_1', P_2' with

$$(P_1', P_2') \lesssim Q$$

Which implies $Q \Downarrow (Q_1, Q_2)$ for some Q_1, Q_2 with

(6.5.2) $P_1' \lesssim Q_1$ & $P_2' \lesssim Q_2$

Applying 6.2(i) to (6.5.1)+(6.5.2) yields $P_1 \lesssim^* Q_1$ & $P_2 \lesssim^* Q_2$, as required.

(iii) $\lambda x.M \lesssim^* Q$ must have been deduced from

(6.5.3) $x \vdash M \lesssim^* M'$

for some M' with $\lambda x.M' \lesssim Q$: hence $Q \Downarrow \lambda y.N$

for some $\lambda y.N$ with

(6.5.4) $\quad \forall P.\ M'[P/x] \lesssim N[P/y]$

Applying 6.4 to (6.5.3) and $\emptyset \vdash P \lesssim^* P$ (which holds by 6.2(ii)), we get

(6.5.5) $\quad \forall P\ \ M[P/x] \lesssim^* M'[P/x]$

and then applying 6.2(i) to (6.5.4) + (6.5.5) gives

$$\forall P\ \ M[P/x] \lesssim^* N[P/y]$$

as required.

$\square$

6.6 Proposition

$$P \Downarrow V \Rightarrow \forall Q\,(P \lesssim^* Q \Rightarrow \exists W\,(Q \Downarrow W \;\&\; V \lesssim^* W))$$

Proof

It suffices to check that

$$\mathcal{E} \stackrel{\text{def}}{=} \{(P,V) \mid \forall Q.\ P \lesssim^* Q \Rightarrow \exists W\,(Q \Downarrow W \;\&\; V \lesssim^* W)\}$$

is closed under the rules in 3.2 inductively defining $\Downarrow$.

Case ($\Downarrow$VAL):

Subcase $V = c \in Const$: that $(c,c) \in \mathcal{E}$ is just 6.5(i).

Subcase $V = (P_1, P_2)$: if $(P_1,P_2) \lesssim^* Q$, then by 6.5(ii) $Q \Downarrow (Q_1, Q_2)$ with $P_1 \lesssim^* Q_1$ & $P_2 \lesssim^* Q_2$ and hence $(P_1,P_2) \lesssim^* (Q_1, Q_2)$ by 6.3. Thus $((P_1,P_2),(P_1,P_2)) \in \mathcal{E}$.

subcase $V = \lambda x.M$: if $\lambda x.M \lesssim^* Q$, this must have been deduced from

(6.6.1) $\qquad x \vdash M \lesssim^* M'$

for some $\lambda x.M'$ satisfying

(6.6.2) $\qquad \lambda x.M' \lesssim Q$

Then (6.6.2) implies $Q \Downarrow \lambda x''.M''$ for some $\lambda x.M''$ with

$$\forall P.\ M'[P/x] \lesssim M''[P/x'']$$

and hence

(6.6.3) $\qquad \lambda x.M' \lesssim \lambda x''.M''$

Applying 6.3 to (6.6.1) yields $\lambda x.M \lesssim^* \lambda x.M'$ and applying 6.2(i) to this + (6.6.3) yields $\lambda x.M \lesssim^* \lambda x''.M''$. Thus $(V,V) \in \mathcal{E}$ when $V = \lambda x.M$.

Case ($\Downarrow$ OP):

Suppose $(P_i, \underline{n}_i) \in \mathcal{E}$ $(i=1,2)$, and that c = value of n_1 op n_2.

If P_1 op $P_2 \lesssim^* Q$, must have

(6.6.4) $\qquad P_1 \lesssim^* P_1'$ & $P_2 \lesssim^* P_2'$

for some P_1', P_2' with

(6.6.5) $\qquad P_1'$ op $P_2' \lesssim Q$

Since $(P_i, \underline{n}_i) \in \mathcal{E}$, (6.6.4) implies $P_i' \Downarrow V_i'$ for some V_i' with $\underline{n}_i \lesssim^* V_i'$, hence with $\underline{n}_i \lesssim V_i'$ and therefore $V_i' = \underline{n}_i$. So $P_i' \Downarrow \underline{n}_i$ $(i=1,2)$, and hence

$$P_1' \text{ op } P_2' \Downarrow c$$

and so from (6.6.5), $Q \Downarrow c$. Thus $(P_1 \text{ op } P_2, c) \in \mathcal{E}$.

Case ($\Downarrow$ IF$_1$) :

Suppose $(B, \text{true}) \in \mathcal{E}$ and $(P_1, V_1) \in \mathcal{E}$.

If $(\text{if } B \text{ then } P_1 \text{ else } P_2) \lesssim^* Q$, must have

(6.6.6) $\quad B \lesssim^* B'$ & $P_1 \lesssim^* P_1'$ & $P_2 \lesssim^* P_2'$

for some B', P_1', P_2' with

(6.6.7) $\quad (\text{if } B' \text{ then } P_1' \text{ else } P_2') \lesssim Q$

Since $(B, \text{true}) \in \mathcal{E}$, from (6.6.6) we get $B' \Downarrow V'$ for some V' with $\text{true} \lesssim^* V'$, hence $\text{true} \lesssim V'$, hence $V' = \text{true}$. Thus

(6.6.8) $\quad B' \Downarrow \text{true}$

Since $(P_1, V_1) \in \mathcal{E}$, from (6.6.6) we get

(6.6.9) $\quad P_1' \Downarrow V_1'$

for some V_1' with

(6.6.10) $\quad V_1 \lesssim^* V_1'$

Applying ($\Downarrow$ IF$_1$) to (6.6.8)+(6.6.9) yields

$$(\text{if } B' \text{ then } P_1' \text{ else } P_2') \Downarrow V_1'$$

and then (6.6.7) implies

$$Q \Downarrow V_1''$$

for some V_1'' with

$$V_1' \lesssim V_1''$$

and hence (by 6.2(i) & (6.6.10)) with $V_1 \lesssim^* V_1''$.

Thus $(\text{if } B \text{ then } P_1 \text{ else } P_2, V_1) \in \mathcal{E}$.

Case $(\Downarrow IF_2)$: – like that for $(\Downarrow IF_1)$.

Case $[\Downarrow SPLIT]$:

Suppose $(P, (P_1, P_2)) \in \mathcal{E}$ and $(M[P_1/x, P_2/y], V) \in \mathcal{E}$.
If $(\text{split } P \text{ as } (x,y) \text{ in } M) \lesssim^* Q$, must have

(6.6.11) $\quad P \lesssim^* P'$ & $\quad x, y \vdash M \lesssim^* M'$

for some P', M' with

(6.6.12) $\quad (\text{split } P' \text{ as } (x,y) \text{ in } M') \lesssim Q$

Since $(P, (P_1, P_2)) \in \mathcal{E}$, (6.6.11) yields $P' \Downarrow V'$ for some V' with $(P_1, P_2) \lesssim^* V'$; so by 6.5 (ii)

(6.6.13) $\quad P_1 \lesssim^* P_1'$ & $P_2 \lesssim^* P_2'$

for some P_1', P_2' with $(P_1', P_2') = V'$.

Thus

(6.6.14) $\quad P' \Downarrow (P_1', P_2')$

and applying 6.4 to (6.6.11)+(6.6.13) twice yields

(6.6.15) $\quad M[P_1/x, P_2/y] \lesssim^* M'[P_1'/x, P_2'/y]$.

Then since $(M[P_1/x, P_2/y], V) \in \mathcal{E}$, (6.6.15) yields

(6.6.16) $\quad M'[P_1'/x, P_2'/y] \Downarrow V'$

for some V' with

(6.6.17) $\quad V \lesssim^* V'$

Applying ($\Downarrow$SPLIT) to (6.6.14) + (6.6.16) yields

$$(\text{split } P' \text{ as } (x,y) \text{ in } M') \Downarrow V'$$

and hence from (6.6.12) we have $Q \Downarrow V''$ for some V'' with $V' \lesssim V''$ and hence (by (6.6.17) + 6.2(i)) $V \lesssim^* V''$.

Thus $(\text{split } P \text{ as } (x,y) \text{ in } M,\ V) \in \mathcal{E}$.

Cases ($\Downarrow$APP) & ($\Downarrow$REC): are similar to that for ($\Downarrow$SPLIT) and are left as *exercises*.

$\square$

6.7 Corollary

$$P \lesssim^* Q \Rightarrow P \lesssim Q$$

Proof

It suffices to show that $\{(P,Q) \mid P \lesssim^* Q\}$ is an applicative simulation (cf 5.4). But this follows immediately from 6.6 plus 6.5.

$\square$

6.8 Proposition

$$\Gamma \vdash M \lesssim^* N \iff \Gamma \vdash M \lesssim N.$$

Proof

$\Leftarrow$ is Lemma 6.2(iii). Conversely, if $\Gamma \vdash M \lesssim^* N$ with $\Gamma = \{x_1, \ldots, x_n\}$ say, then by repeated use of 6.4, we have $M[\vec{P}/\vec{x}] \lesssim^* N[\vec{P}/\vec{x}]$ for any $P_1, \ldots, P_n \in Prog$. Then by 6.7, $M[\vec{P}/\vec{x}] \lesssim N[\vec{P}/\vec{x}]$. Since this holds for all $\vec{P}$, we have $\Gamma' \vdash M \lesssim N$, by definition.

$\square$

Since $\lesssim^*$ coincides with $\lesssim$, Proposition 5.7 follows immediately from Lemma 6.3.

7. LEAST FIXED POINTS IN CPPOS

Motivation: Fact 4.10 (v) (yet to be proved) shows that $\mathrm{rec}\, x. M$ is the least pre-fixed point of the monotone function $P \mapsto M[P/x]$ on the preordered set $(\mathrm{Prog}, \sqsubseteq)$. Furthermore, 4.10(vii) shows that this least pre-fixed point is the lub (cf 1.7) of the chain built up starting with the least element Ω of $(\mathrm{Prog}, \sqsubseteq)$ (by 4.10(vi)) and iterating $P \mapsto M[P/x]$. Moreover, any $\mathbb{L}$-definable monotone function $P \mapsto N[P/x]$ preserves these lubs.

It turns out that these FACTS can be established by considering a mathematical idealization of $(\mathrm{Prog}, \sqsubseteq)$, viz. preordered sets (in fact partially ordered sets will suffice) possessing lubs of all countable chains whatsoever. Such structures will provide a denotational semantics for $\mathbb{L}$ satisfying the general requirements of compositionality and computational adequacy mentioned in the Introduction.

7.1 Definitions

(Recall the definitions of poset, monotone function and lub from 1.7.)

An <u>ω-chain complete poset</u> (or <u>cpo</u>, for short) is a poset $(D, \sqsubseteq)$ possessing lubs for all countable ascending chains

$$d_0 \sqsubseteq d_1 \sqsubseteq d_2 \sqsubseteq \cdots \qquad (d_i \in D).$$

The lub of such a chain, written $\bigsqcup_{i<\omega} d_i$, is uniquely determined by the property

$$\forall d \in D \left(\bigsqcup_{i<\omega} d_i \sqsubseteq d \iff \forall i<\omega \, (d_i \sqsubseteq d) \right)$$

A <u>pointed cpo</u> (or <u>cppo</u>, for short) is a cpo D possessing a least element, $\perp$:

$$\forall d \in D \, (\perp \sqsubseteq d)$$

A function $f : D \to E$ between cpos is <u>continuous</u> if it is monotone ($d \sqsubseteq d' \Rightarrow f(d) \sqsubseteq f(d')$) and preserves the lubs of countable ascending chains, i.e. given $d_0 \sqsubseteq d_1 \sqsubseteq \cdots$ in D, then

$$f\left(\bigsqcup_{i<\omega} d_i\right) = \bigsqcup_{i<\omega} f(d_i).$$

<u>Note</u>: $(f(d_i) \mid i<\omega)$ <u>is</u> a (countable) ascending chain in E because f is monotone. Any monotone f satisfies

$$\bigsqcup_{i<\omega} f(d_i) \sqsubseteq f\left(\bigsqcup_{i<\omega} d_i\right)$$

(because $\forall j \left(f(d_j) \sqsubseteq f\left(\bigsqcup_{i<\omega} d_i\right) \right)$), so to check montone

f is continuous, it suffices to show

$$f\left(\bigsqcup_{i<\omega} d_i\right) \sqsubseteq \bigsqcup_{i<\omega} f(d_i)$$

for all ω-chains $(d_i \mid i<\omega)$ in D.

<u>7.2 Theorem</u> (Tarski Fixed Point Theorem for cppos)

Any continuous function $f : D \to D$ on a cppo possesses a least pre-fixed point, i.e. an element $\mu(f) \in D$ satisfying

(i) $f(\mu(f)) \sqsubseteq \mu(f)$

(ii) $\forall d \in D.\ f(d) \sqsubseteq d \Rightarrow \mu(f) \sqsubseteq d$

<u>Proof</u>

Define $\begin{cases} f^0(\bot) = \bot \\ f^{n+1}(\bot) = f(f^n(\bot)) \end{cases}$ It is easy to check, by induction on n, that $\forall n\ (f^n(\bot) \sqsubseteq f^{n+1}(\bot))$. Then define

$$\mu(f) = \bigsqcup_{n<\omega} f^n(\bot).$$

By continuity of f,

$$\begin{aligned} f(\mu(f)) &= \bigsqcup_{n<\omega} f(f^n(\bot)) \\ &= \bigsqcup_{n<\omega} f^{n+1}(\bot) \\ &= \bigsqcup_{n<\omega} f^n(\bot) \\ &= \mu(f) \end{aligned}$$

so that $\mu(f)$ satisfies (i), indeed, is a fixed point of f. Furthermore, if $f(d) \sqsubseteq d$, then it is easy to prove

$$\forall n\,(f^n(\bot) \sqsubseteq d)$$

by induction on n; hence $\mu(f) \sqsubseteq d$, as required for (ii).

□

7.3 Examples

(i) Any complete lattice (cf. lemma 1.8) is in particular a cppo. When R is a finitary rule set on X (cf. 1.4), then $\Phi_R : \mathcal{P}(X) \to \mathcal{P}(X)$ (defined in 1.2) is continuous, and $\mu(\Phi_R)$ as calculated in the proof of Theorem 7.2 coincides with the construction of $\mu(\Phi_R)$ given in Proposition 1.5.

(ii) Given sets X, Y, the set

$$Pfn(X,Y) = \{F \subseteq X \times Y \mid \forall x \in X.\ \forall y, y' \in Y.\ (x,y) \in F \ \&\ (x,y') \in F \Rightarrow y = y'\}$$

of (graphs of) partial functions from X to Y, partially ordered by $\subseteq$, is a cppo. The least element is $\emptyset$; and the lub of $F_0 \subseteq F_1 \subseteq F_2 \subseteq \ldots$ is $\bigcup_{n<\omega} F_n$ (which is a partial function).

(Note that in general $Pfn(X,Y)$ does not possess lubs for all subsets, i.e. is not in general a complete lattice. It does however possess all non-empty glb's, given by intersection.)

Taking $X = Y = \mathbb{N}$, consider

$$f : Pfn(\mathbb{N}, \mathbb{N}) \to Pfn(\mathbb{N}, \mathbb{N})$$

given by

$$f(F) = \{(0,1)\} \cup \{(m+1, (m+1)n) \mid (m,n) \in F\}$$

i.e. $f(F)$ is the partial function mapping m to

$$\begin{cases} 1 & \text{if } m = 0 \\ mF(m-1) & \text{if } m > 0 \ \& \ F(m-1) \text{ defined} \\ \text{undefined} & \text{otherwise} \end{cases}$$

Then f is continuous (exercise: check this) and

$$\mu(f) = \{(m, !m) \mid m \in \mathbb{N}\}$$

is the (graph of the) factorial function.
(Proof: it suffices to show (by induction on n) that for all $n \geq 0$

$$f^n(F) = \{(m, !m) \mid m < n\}. \quad)$$

7.4 Definition

A subset $S \subseteq D$ of a cppo D is called <u>admissible</u> if

(7.4.1) it contains $\perp$

(7.4.2) it is closed under lubs of ω-chains in D, i.e. given $d_0 \sqsubseteq d_1 \sqsubseteq d_2 \sqsubseteq \cdots$ in D,

$$\forall n (d_n \in S) \Rightarrow (\bigsqcup_{n<\omega} d_n) \in S.$$

7.5 Proposition (Scott's Fixed Point Induction Principle)

Let $f : D \to D$ be a continuous function on a cppo D. For any admissible $S \subseteq D$, to prove $\mu(f) \in S$ it suffices to prove

(7.5.1) $\forall d \in D (\, d \in S \Rightarrow f(d) \in S)$.

<u>Proof</u>

From (7.4.1) + (7.5.1) it follows by induction on $n \in \mathbb{N}$ that $\forall n \geq 0 (\, f^n(\perp) \in S)$. Hence by (7.4.2)

$$\mu(f) = \bigsqcup_{n<\omega} f^n(\perp) \in S.$$

$\square$

7.6 Example:

Suppose f, g, h are continuous functions $D \to D$ (D a cppo). If $h \circ f = f \circ h$, $g \circ f = f \circ g$ and $g(\perp) = h(\perp)$, then $g(\mu(f)) = h(\mu(f))$.

Proof

$S = \{ d \in D \mid g(d) = h(d) \}$ is admissible ($\perp \in S$ by assumption; S ω-chain closed 'cos g & h are continuous); and if $d \in S$ then
$$g(f(d)) = f(g(d)) = f(h(d)) = h(f(d)),$$
so $f(d) \in S$. Hence by Proposition 7.5, $\mu(f) \in S$ as required. $\square$

7.7 Definition

A function $f : D \to E$ between cppos is called <u>strict</u> if $f(\perp_D) = \perp_E$.

The notation $f : D \multimap E$ will be used to indicate that f is strict and continuous.

7.8 Proposition (Plotkin's characterization of μ in terms of a "uniformity" property)

The family of operations

$$f \mapsto \mu(f) \qquad (D \text{ a cppo, } f: D \to D \text{ continuous})$$

is uniquely determined by the following two properties

(i) $\quad f(\mu(f)) = \mu(f)$

(ii) for any commutative square (s strict & cts)

$$\begin{array}{ccc} D & \xrightarrow{s} & D' \\ f\downarrow & & \downarrow f' \\ D & \xrightarrow[s]{} & D' \end{array}$$

$$\mu(f') = s(\mu(f))$$

Proof

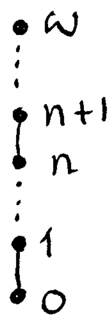
Clearly μ satisfies (i), and we can use Scott induction (7.5) to verify that it satisfies (ii). First note that

$$f'(s(\mu(f))) = s(f(\mu(f))) = s(\mu(f)) \quad \text{(by (i))}$$

So $s(\mu(f))$ is a fixed point for f' and hence $\mu(f') \sqsubseteq s(\mu(f))$. So it suffices to show that $s(\mu(f)) \sqsubseteq \mu(f')$, i.e. that $\mu(f) \in S$, where $S = \{d \in D \mid s(d) \sqsubseteq \mu(f')\}$. Clearly S is admissible ($\perp \in S$ because s strict; S ω-chain closed because s continuous). So by 7.5 it suffices to check $d \in S \Rightarrow f(d) \in S$. But if $d \in S$ then $s(f(d)) = f'(s(d)) \sqsubseteq f'(\mu(f')) = \mu(f')$, so $f(d) \in S$.

It remains to show that μ is unique with properties (i) & (ii). Suppose m is another such operation.

Let Ω be the ordinal $\omega+1$.



Hasse diagram for Ω

Clearly Ω is a cppo, and the function $\sigma : \Omega \to \Omega$ given by

$$\sigma(n) = \begin{cases} n+1 & n<\omega \\ \omega & n=\omega \end{cases}$$

is continuous. Furthermore, given any ω-chain $d_0 \sqsubseteq d_1 \sqsubseteq \cdots$ in a cpo D, there is a unique continuous function $\hat{d} : \Omega \to D$ with

$$\hat{d}(n) = d_n \qquad (n<\omega)$$

and hence

$$\hat{d}(\omega) = \bigsqcup_{n<\omega} d_n$$

Given any cppo D and continuous $f : D \to D$, let $\hat{d} : \Omega \to D$ correspond to the ω-chain with $d_n = f^n(\bot)$ $(n<\omega)$. Thus we have $\hat{d}(\omega) = \mu(f)$ and

$$\begin{CD} \Omega @>{\hat{d}}>> D \\ @V{\sigma}VV @VV{f}V \\ \Omega @>>{\hat{d}}> D \end{CD}$$

commutes. Then since m satisfies (ii),

$m(f) = \hat{d}(\cdot m(\sigma))$.

But m also satisfies (i), so $m(\sigma) = \sigma(m(\sigma))$ and hence $m(\sigma) = \omega$ (since σ has exactly one fixed point, viz. ω). Therefore

$$m(f) = \hat{d}(m(\sigma)) = \hat{d}(\omega) = \mu(f)$$

as required. $\square$

8. FUNCTORIAL CONSTRUCTIONS ON DOMAINS

TERMINOLOGY For the purposes of this course the most complicated type of semantic domain (for denotations of programming language expressions) we will need is a pointed cpo – i.e. an ω-chain complete poset with a least element. Henceforward we will refer to pointed cpos (cppos) as domains. [Many more special types of domain – Scott-domains, DI-domains, ... – occur in the literature.]

In this section we give various constructions of cpos and domains that will be needed for the denotational semantics of $\mathbb{L}$. In each case we give the underlying set and the partial order, leaving the reader to check that lubs of ω-chains do indeed exist (and that a least element exists in the case of a construction of a domain).

Then we look at various associated constructions on continuous functions (using a modicum of category theory).

8.1 Definitions

(i) Lifting

The lift $X_\perp$ of a cpo X is the domain obtained by adjoining a new element

$$X_\perp \stackrel{\text{def}}{=} X \cup \{\perp\} \qquad \text{where } \perp \notin X$$

and extending the partial order $\sqsubseteq_X$ to $X_\perp$ by making $\perp$ least:

$$u \sqsubseteq_{X_\perp} u' \Leftrightarrow \big(u = \perp \text{ or } (u, u' \in X \;\&\; u \sqsubseteq_X u')\big)$$

(NB If X already has a least element $\perp_X$ (ie. is a domain), then $\perp_X$ is no longer least in $X_\perp$.)

Notation: given a domain D, its "unlifting" is the cpo

$$D_\downarrow \stackrel{\text{def}}{=} \{d \in D \mid d \neq \perp_D\}$$

(ii) Discrete cpos & flat domains

Each set X gives rise to a discrete cpo via the partial order: $x \sqsubseteq_X x' \Leftrightarrow x = x'$.

A domain is flat if it is of the form $X_\perp$ for some discrete cpo X.

(iii) Cartesian & smash products

Given cpos X & Y, their cartesian product is the cpo

$$X \times Y \stackrel{\text{def}}{=} \{ (x,y) \mid x \in X \;\&\; y \in Y \}$$

$$(x,y) \sqsubseteq_{X \times Y} (x',y') \Leftrightarrow (x \sqsubseteq_X x' \;\&\; y \sqsubseteq_Y y') .$$

NB $(x_0,y_0),(x_1,y_1),\ldots$ is an ω-chain in $X \times Y$ iff $x_0,x_1,\ldots$ and $y_0,y_1,\ldots$ are ω-chains in X & Y respectively; and hence

$$\bigsqcup_{i<\omega} (x_i,y_i) = \left(\bigsqcup_{i<\omega} x_i , \bigsqcup_{j<\omega} y_j \right)$$

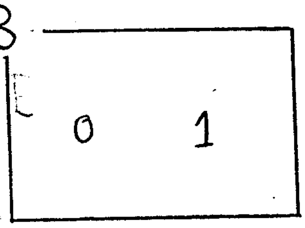
NB If X & Y are domains, so is $X \times Y$ – its least element being $(\perp_X, \perp_Y)$.

The smash product $D \otimes E$ of two domains D & E is the domain $D \otimes E \stackrel{\text{def}}{=} (D_\downarrow \times E_\downarrow)_\perp$

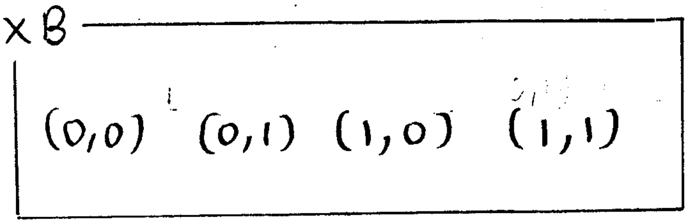
$$= \{ (d,e) \in D \times E \mid d \neq \perp_D \;\&\; e = \perp_E \} \cup \{ \perp \}$$

Some pictures, in case $B=\{0,1\}$, discrete cpo

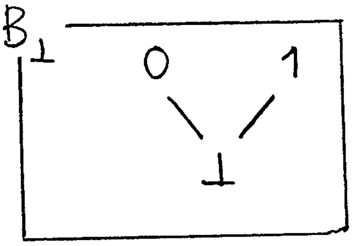
B



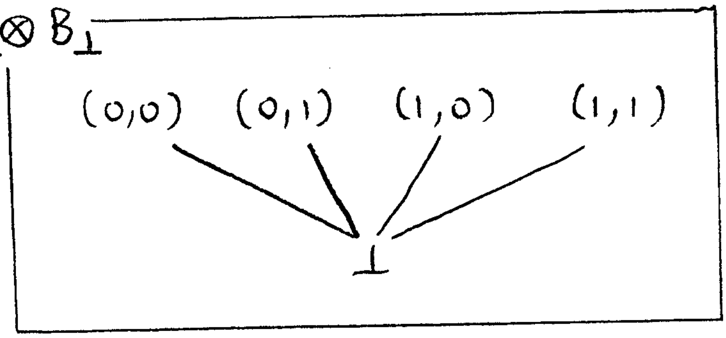
$B \times B$



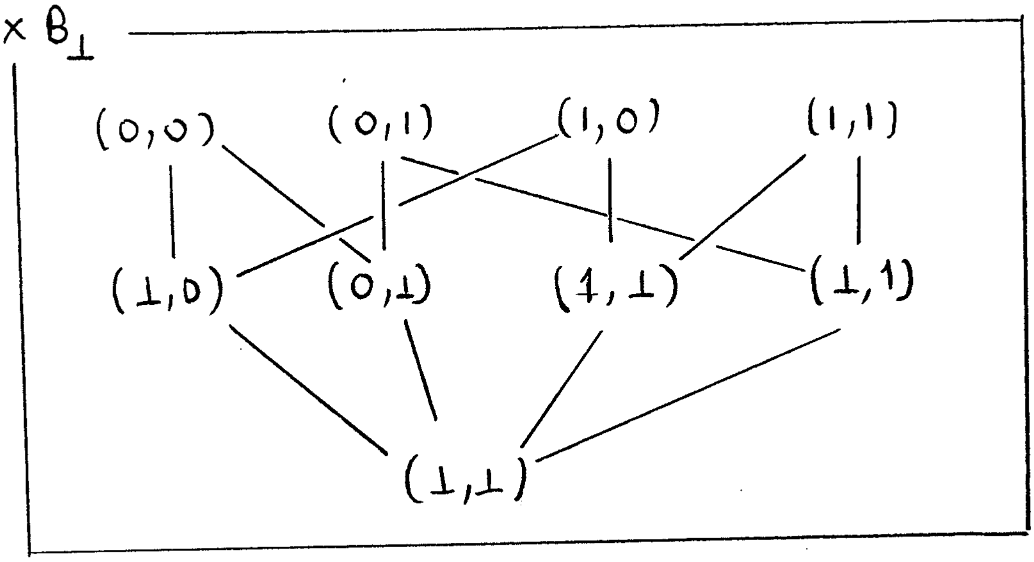
$B_\perp$



$B_\perp \otimes B_\perp$



$B_\perp \times B_\perp$



(iv) Disjoint union & coalesced sum

Given cpos X & Y, their <u>disjoint union</u> is the cpo

$$X + Y \stackrel{\text{def}}{=} \{ inl(x) \mid x \in X \} \cup \{ inr(y) \mid y \in Y \}$$

$$u \sqsubseteq_{X+Y} u' \Leftrightarrow \exists x, x' \in X \; (u = inl(x) \;\&\; u' = inr(x') \;\&\; x \sqsubseteq_X x')$$
$$\vee \; \exists y, y' \in Y \; (u = inr(y) \;\&\; u' = inr(y') \;\&\; y \sqsubseteq_Y y')$$

where $x \mapsto inl(x)$, $y \mapsto inr(y)$ are injective functions with disjoint images (e.g. for definiteness, could take $inl(x) \stackrel{\text{def}}{=} (0, x)$, $inr(y) \stackrel{\text{def}}{=} (1, y)$).

<u>NB</u> if $u_0, u_1, \ldots$ is an ω-chain in $X+Y$, then
either $u_i = inl(x_i)$ with $x_0, x_1, \ldots$ an ω-chain in X
or $u_i = inr(y_i)$ with $y_0, y_1, \ldots$ " " " " Y
In the first case $\bigsqcup_{i<\omega} u_i = inl(\bigsqcup_{i<\omega} x_i)$, in the second case $\bigsqcup_{i<\omega} u_i = inr(\bigsqcup_{i<\omega} y_i)$.

The <u>coalesced sum</u> $D \oplus E$ of domains D & E is the domain

$$D \oplus E \stackrel{\text{def}}{=} (D_\downarrow + E_\downarrow)_\perp$$

$$= \{ inl(d) \mid \perp_D \neq d \in D \} \cup \{ inr(e) \mid \perp_E \neq e \in E \} \cup \{\perp\}.$$

(v) Function spaces

Given cpos X & Y, their continuous function space is the cpo: $X \to Y \stackrel{\text{def}}{=} \{ f : X \to Y \mid f \text{ is continuous} \}$

$$f \sqsubseteq_{X\to Y} f' \iff \forall x \in X \, (f(x) \sqsubseteq_Y f'(x))$$

NB If $f_0, f_1, f_2, \ldots$ is an ω-chain in $X \to Y$, then

- for each $x \in X$, $f_0(x), f_1(x), \ldots$ is an ω-chain in Y
- the function $\lambda x \in X. \bigsqcup_{i<\omega} f_i(x)$ is continuous

  and is the lub of $f_0, f_1, \ldots$ in $X \to Y$.

NB If Y has a least element, so does $X \to Y$, viz. $\lambda x \in X. \perp_Y$.

The strict continuous function space $D \multimap E$ of two domains D & E is the domain

$$D \multimap E \stackrel{\text{def}}{=} \{ f \in (D \to E) \mid f(\perp_D) = \perp_E \}$$

with partial order inherited from $D \to E$.

Recall that $f : D \to E$ is said to be strict if $f(\perp_D) = \perp_E$.

Notation: $f : D \multimap E$ indicates f is a strict continuous function from D to E (i.e. $f \in (D \multimap E)$).

NB $D \multimap E$ is a domain because $\perp_{D\to E}$ is strict and $\bigsqcup_{i<\omega} f_i$ is strict when $f_0, f_1, \ldots$ is an ω-chain of strict continuous functions.

8.2 Definition

A cpo-enriched category $\mathbb{C}$ is a category in which for each pair X, X' of objects, the collection $\mathbb{C}(X,X')$ of morphisms $X \to X'$ is endowed with the structure of a cpo, in such a way that each composition function

$$\begin{aligned} \mathbb{C}(X,X') \times \mathbb{C}(X',X'') &\longrightarrow \mathbb{C}(X,X'') \\ (f\ ,\ g) &\longmapsto g \circ f \end{aligned}$$

is continuous.

Note that if $\mathbb{C}$ is cpo-enriched, the opposite category $\mathbb{C}^{op}$ is as well — just take the cpo-structure on $\mathbb{C}^{op}(X,X') \stackrel{\text{def}}{=} \mathbb{C}(X',X)$ to be that given by $\mathbb{C}$ for $\mathbb{C}(X',X)$.

Note also that if $\mathbb{C}$ & $\mathbb{D}$ are both cpo-enriched, we can use 8.1 (iii) to make the product category $\mathbb{C} \times \mathbb{D}$ cpo-enriched — just take the cpo structure on $(\mathbb{C}\times\mathbb{D})((X,Y),(X',Y')) \stackrel{\text{def}}{=} \mathbb{C}(X,X') \times \mathbb{D}(Y,Y')$ to be as in 8.1 (iii) [exercise: check that composition in $\mathbb{C}\times\mathbb{D}$ is continuous given that it is in $\mathbb{C}$ & $\mathbb{D}$.]

A (cpo-enriched or) locally continuous functor $F : \mathbb{C} \to \mathbb{D}$ between cpo-enriched categories $\mathbb{C}$ & $\mathbb{D}$ is a functor for which the action on morphisms

$$\begin{aligned} \mathbb{C}(X,X') &\longrightarrow \mathbb{D}(F(X),F(X')) \\ f &\longmapsto F(f) \end{aligned}$$

is continuous (for each pair of $\mathbb{C}$-objects X, X').

8.3 Examples

The categories

$\mathbf{Cpo}$ = cpos & continuous functions

$\mathbf{Dom}$ = domains & continuous functions

$\mathbf{Cpo}_\perp$ = domains & strict continuous functions

(with composition & identity morphisms inherited from the underlying category of sets & functions)

are all cpo-enriched via 8.1(v):

$$\mathbf{Cpo}(X,Y) = X \to Y$$
$$\mathbf{Dom}(D,E) = D \to E$$
$$\mathbf{Cpo}_\perp(D,E) = D \multimap E$$

<u>Exercise</u>: check that composition

$$\begin{aligned} (X\to Y)\times(Y\to Z) &\longrightarrow (X\to Z) \\ (f,g) &\longmapsto g\circ f \stackrel{\text{def}}{=} \lambda x \in X.\, g(f(x)) \end{aligned}$$

is a continuous function. Use the following:

8.4 Lemma

Given a cpo X and a family of elements $(x_{ij} \mid i<\omega, j<\omega)$ satisfying

$$i \leq i' \ \&\ j \leq j' \Rightarrow x_{ij} \sqsubseteq x_{i'j'}$$

then

$$\bigsqcup_{i<\omega}\Big(\bigsqcup_{j<\omega} x_{ij}\Big) = \bigsqcup_{k<\omega} x_{kk} = \bigsqcup_{j<\omega}\Big(\bigsqcup_{i<\omega} x_{ij}\Big)$$

$\square$

8.5 Corollary

Given cpos X, Y, Z, a function $f: X \times Y \to Z$ is continuous iff

$\forall x \in X \quad f(x,-): Y \to Z$ is continuous

and

$\forall y \in Y \quad f(-,y): X \to Z$ is continuous

$\square$

8.6 Proposition

Lifting extends to a locally continuous functor $\mathbf{Cpo}_\perp \to \mathbf{Cpo}_\perp$.

Cartesian product, smash product and coalesced sum extend to locally continuous functors $\mathbf{Cpo}_\perp \times \mathbf{Cpo}_\perp \longrightarrow \mathbf{Cpo}_\perp$.

Continuous & strict continuous function space constructs extend to locally continuous functors $\mathbf{Cpo}_\perp^{op} \times \mathbf{Cpo}_\perp \to \mathbf{Cpo}_\perp$.

Proof

We just give the definition of the action of these various domain constructors on strict continuous functions and leave the reader to check that these actions are well-defined (in particular, that they yield strict continuous functions) and continuous.

Lifting: given $f : D \multimap E$, $f_\perp : D_\perp \multimap E_\perp$ is defined by

$$f_\perp(u) \stackrel{\text{def}}{=} \begin{cases} f(d) & \text{if } u = d \in D \\ \perp & \text{if } u = \perp \end{cases}$$

Product: given $f_i : D_i \multimap E_i$ $(i = 1,2)$, $f_1 \times f_2 : D_1 \times D_2 \multimap E_1 \times E_2$ is defined by

$$(f_1 \times f_2)(d_1, d_2) \stackrel{\text{def}}{=} (f_1(d_1), f_2(d_2))$$

Smash product: given $f_i : D_i \multimap E_i$ $(i=1,2)$ $f_1 \otimes f_2 : D_1 \otimes D_2 \multimap E_1 \otimes E_2$ is defined by

$$(f_1 \otimes f_2)(u) = \begin{cases} (f_1(d_1), f_2(d_2)) & \text{if } u = (d_1, d_2) \in (D_1)_\downarrow \times (D_2)_\downarrow \\ & \text{and } f_i(d_i) \neq \perp \ (i=1,2) \\ \perp & \text{otherwise} \end{cases}$$

Coalesced sum: given $f_i : D_i \multimap E_i$ $(i=1,2)$ $f_1 \oplus f_2 : D_1 \oplus D_2 \multimap E_1 \oplus E_2$ is defined by

$$(f_1 \oplus f_2)(u) = \begin{cases} inl(f_1(d_1)) & \text{if } u = inl(d_1), d_1 \in (D_1)_\downarrow \ \& \ f_1(d_1) \neq \perp \\ inr(f_2(d_2)) & \text{if } u = inr(d_2), d_2 \in (D_2)_\downarrow \ \& \ f_2(d_2) \neq \perp \\ \perp & \text{otherwise} \end{cases}$$

Continuous & strict continuous function spaces: given $f_1 : E_1 \multimap D_1$ (NB!), $f_2 : D_2 \multimap E_2$, then $(f_1 \to f_2) : (D_1 \to D_2) \multimap (E_1 \to E_2)$ & $(f_1 \multimap f_2) : (D_1 \multimap D_2) \multimap (E_1 \multimap E_2)$ are both given by $g \mapsto f_2 \circ g \circ f_1$. $\square$

9. RECURSIVELY DEFINED DOMAINS

In the next section we will give $\mathbb{L}$ programs $P \in Prog$ denotations $[\![P]\!] \in D$, where D is a particular domain satisfying

$$(*) \quad D \cong (C + (D \times D) + D \to D)_\perp$$

where $C \stackrel{\text{def}}{=} \{true, false\} \cup \mathbb{Z}$, regarded as a discrete cpo and $\cong$ indicates an isomorphism in the category **Dom** of domains.

[Note: to specify an isomorphism $i : X \cong Y$ in **Cpo**, **Dom** or $\mathbf{Cpo}_\perp$, it suffices to give a bijection between the underlying sets of the cpos with the property
$\forall x, x' \in X (\; x \sqsubseteq_X x' \Leftrightarrow i(x) \sqsubseteq_Y i(x'))$.
This property suffices to ensure that both i and its inverse $i^{-1} : Y \cong X$ are continuous functions, and preserve $\perp$ if it exists.
Exercise: check this.]

$(*)$ is a typical example of a recursive domain equation. The existence of D satisfying $(*)$ is a non-trivial problem, initially solved by Dana Scott in 1969. The general form of recursive domain equation we will consider is $D \cong F(D, D)$ where $F : \mathbf{Cpo}_\perp^{op} \times \mathbf{Cpo}_\perp \to \mathbf{Cpo}_\perp$ is a locally continuous functor.

9.1 Definitions

Let $F : \mathbf{Cpo}_\perp^{op} \times \mathbf{Cpo}_\perp \to \mathbf{Cpo}$ be a (locally continuous) functor.

An *invariant* for F is a pair (D, i), where D is a domain and $i : F(D,D) \cong D$ is an isomorphism (in $\mathbf{Cpo}_\perp$).

9.2 Example

Note that $(C + (D \times D) + (D \to D))_\perp = C_\perp \oplus (D \times D)_\perp \oplus (D \to D)_\perp$.
So by virtue of Prop.ⁿ 8.6, solutions to (*) are the same thing as invariants for the functor $F : \mathbf{Cpo}_\perp^{op} \times \mathbf{Cpo}_\perp \to \mathbf{Cpo}$ whose action on objects sends a pair of domains D^-, D^+ to

$$(C + (D^+ \times D^+) + (D^- \to D^+))_\perp$$

[<u>NB</u>: it is the fact that the function space constructors $\to$ & $\multimap$ are contravariant in their left-hand arguments (for strict continuous functions) which necessitates considering functors of shape $\mathbf{Cpo}_\perp^{op} \times \mathbf{Cpo}_\perp \to \mathbf{Cpo}$ rather than just $\mathbf{Cpo}_\perp \to \mathbf{Cpo}_\perp$.]

For the denotational semantics of $\mathbb{L}$ to have good properties (specifically, for it to be "computationally adequate") we cannot just use any domain satisfying $(*)$ — we have to use one which is suitably minimal, in the following sense.

9.3 Definition

Let (D, i) be an invariant for a locally continuous functor $F : \mathbf{Cpo}_{\perp}^{op} \times \mathbf{Cpo} \to \mathbf{Cpo}$. By virtue of local continuity of F, we get a continuous function

$$\delta : (D \multimap D) \to (D \multimap D)$$

given by $e \longmapsto i \circ F(e,e) \circ i^{-1}$.

We say (D, i) is a minimal invariant for F if the least prefixed point, $\mu(\delta)$, of δ is id_D, the identity function on D, i.e. if for all $d \in D$

$$d = \bigsqcup_{n < \omega} \pi_n(d)$$

Where $\pi_n : D \multimap D$ $(n < \omega)$ are defined by

$$\begin{cases} \pi_0(d) = \perp_D \\ \pi_{n+1}(d) = i(F(\pi_n, \pi_n)(i^{-1}(d))) \end{cases}$$

9.4 Theorem (Existence of minimal invariants)

Any locally continuous functor $F: \mathbf{Cpo}_\perp^{op} \times \mathbf{Cpo}_\perp \to \mathbf{Cpo}$ possesses a minimal invariant, $i: F(D,D) \cong D$.

Proof

(i) Construction of D (as the limit of an ω^{op}-chain).

Let $(D_n \mid n<\omega)$ be the family of domains defined by

$$\begin{cases} D_0 \stackrel{\text{def}}{=} \emptyset_\perp = \{\perp\} \\ D_{n+1} \stackrel{\text{def}}{=} F(D_n, D_n) \end{cases}$$

We define strict continuous functions

$$\begin{cases} D_0 \xrightarrow{i_0} D_1 \xrightarrow{i_1} D_2 \longrightarrow \cdots \\ D_0 \xleftarrow{r_0} D_1 \xleftarrow{r_1} D_2 \longleftarrow \cdots \end{cases}$$

by

$$\begin{cases} i_0 \stackrel{\text{def}}{=} \perp_{(D_0 \multimap D_1)} , \quad r_0 \stackrel{\text{def}}{=} \perp_{(D_1 \multimap D_0)} \\ i_{n+1} \stackrel{\text{def}}{=} F(r_n, i_n), \quad r_{n+1} = F(i_n, r_n) \end{cases}$$

Now let

$$D \stackrel{\text{def}}{=} \left\{ (d_n \mid n<\omega) \in \prod_{n<\omega} D_n \;\middle|\; \forall n<\omega \left(r_n(d_{n+1}) = d_n \right) \right\}$$

with partial order

$$(d_n \mid n<\omega) \sqsubseteq (d'_n \mid n<\omega) \Leftrightarrow \forall n<\omega \left(d_n \sqsubseteq_{D_n} d'_n \right)$$

Because the P_n are strict continuous, D is a domain, with lubs of ω-chains given component-wise and with least element $\perp = (\perp_{D_n} \mid n<\omega)$. Furthermore, the projection functions $(d_n \mid n<\omega) \mapsto d_n$ determine

(i) strict continuous functions
$$p_n : D \multimap D_n .$$
satisfying $r_n \circ p_{n+1} = p_n$.

(ii) Lemma

For all $n<\omega$, $\begin{cases} r_n \circ i_n = id_{D_n} \\ i_n \circ r_n \sqsubseteq id_{D_{n+1}} \end{cases}$

Proof: by induction on n, from the definition of i_n & r_n plus the fact that F preserves $\circ$ & $\sqsubseteq$. $\square$ of (ii)

(iii) Lemma

For each $n, m<\omega$ and each $x \in D_n$, define
$$(e_n(x))_m \stackrel{def}{=} \begin{cases} r_{nm}(x) & \text{if} \quad m<n \\ x & \text{if} \quad m=n \\ i_{nm}(x) & \text{if} \quad m>n \end{cases}$$

Where for $m<n$
$$\begin{cases} r_{nm} \stackrel{def}{=} (D_n \xrightarrow{r_{n-1}} D_{n-1} \multimap \cdots \xrightarrow{r_m} D_m) \\ i_{mn} \stackrel{def}{=} (D_m \xrightarrow{i_m} D_{m+1} \multimap \cdots \xrightarrow{i_{n-1}} D_n \end{cases}$$

Then

(a) $e_n(x) \stackrel{def}{=} ((e_n(x))_m \mid m<\omega) \in D$

(b) $e_n : D_n \multimap D$

(c) $e_n = e_{n+1} \circ i_n$ & $e_n \circ r_n \sqsubseteq e_{n+1}$

(d) $p_n \circ e_n = id_{D_n}$

(e) $e_0 \circ p_0 \sqsubseteq e_1 \circ p_1 \sqsubseteq \dots$ and $\bigsqcup_{n<\omega} e_n \circ p_n = id_D$

Proof

(a) follows from (ii) and the definition of D

(b) follows from the fact that all the i_n & r_n are continuous & strict.

(c) & (d) follow from (i) + defintion of e_n.

For (e), note first that

$$
\begin{aligned}
e_n \circ p_n &= (e_{n+1} \circ i_n) \circ (r_n \circ p_{n+1}) && \text{by (i) \& (c)} \\
&= e_{n+1} \circ (i_n \circ r_n) \circ p_{n+1} \\
&\sqsubseteq e_{n+1} \circ p_{n+1} && \text{by (i)}
\end{aligned}
$$

Moreover

$$((e_n \circ p_n)(d))_m = (e_n(d_n))_m = d_m \quad \text{for } m \leq n$$

Thus for any $d \in D$ and any $m \geq 0$

$$
\begin{aligned}
((\bigsqcup_{n<\omega} e_n \circ p_n)(d))_m &= \bigsqcup_{n<\omega} ((e_n \circ p_n)(d))_m \\
&= \bigsqcup_{n \geq m} ((e_n \circ p_n)(d))_m && \left(\begin{array}{l}\text{lub of a chain} \\ = \text{lub of any cofinal} \\ \text{segment}\end{array}\right) \\
&= d_m
\end{aligned}
$$

Hence $\bigsqcup_{n<\omega} e_n \circ p_n = id_D$.

$\square$ of (iii)

(iv) Construction of $i : F(D,D) \multimap D$ and its inverse

Note that by (i) + (iii)(c) + functoriality of F:

$$
\begin{aligned}
F(e_n, p_n) = F(e_{n+1} i_n, r_n p_{n+1}) &= F(i_n, r_n) F(e_{n+1}, p_{n+1}) \\
&= r_{n+1} F(e_{n+1}, p_{n+1})
\end{aligned}
$$

So that $e_{n+1} F(e_n, p_n) = e_{n+1} r_{n+1} F(e_{n+1}, p_{n+1}) \underset{\text{by (iii)(c).}}{\sqsubseteq} e_{n+2} F(e_{n+1}, p_{n+1})$.

Thus $(e_{n+1}\circ F(e_n, p_n) : F(D,D) \multimap D \mid n<\omega)$ forms an ω-chain in $(F(D,D) \multimap D)$. Define

$$i \stackrel{\text{def}}{=} \bigsqcup_{n<\omega} e_{n+1}\circ F(e_n, p_n) \quad : \; F(D,D) \multimap D.$$

Similarly, $(F(p_n, e_n)\cdot p_{n+1} : D \multimap F(D,D) \mid n<\omega)$ is an ω-chain and we can define

$$j \stackrel{\text{def}}{=} \bigsqcup_{n<\omega} F(p_n, e_n)\circ p_{n+1} \quad : \; D \multimap F(D,D).$$

Then

$$\begin{aligned}
j\circ i &= \Big(\bigsqcup_{m<\omega} F(p_m, e_m) p_{m+1}\Big)\Big(\bigsqcup_{n<\omega} e_{n+1} F(e_n, p_n)\Big) \\
&= \bigsqcup_{k<\omega} F(p_k, e_k) p_{k+1} e_{k+1} F(e_k, p_k) && \text{by lemma 8.4} \\
&= \bigsqcup_{k<\omega} F(p_k, e_k) F(e_k, p_k) && \text{by (iii)(d)} \\
&= \bigsqcup_{k<\omega} F(e_k\circ p_k, e_k\circ p_k) \\
&= F\Big(\bigsqcup_{m<\omega} e_m p_m, \bigsqcup_{n<\omega} e_n p_n\Big) && \text{by Lemma 8.4} \\
&= F(id, id) && \text{by (iii)(e)} \\
&= id
\end{aligned}$$

and

$$\begin{aligned}
i\circ j &= \Big(\bigsqcup_{n<\omega} e_{n+1} F(e_n, p_n)\Big)\Big(\bigsqcup_{m<\omega} F(p_m, e_m) p_{m+1}\Big) \\
&= \bigsqcup_{k<\omega} e_{k+1} F(e_k, p_k) F(p_k, e_k) p_{k+1} && \text{by lemma 8.4} \\
&= \bigsqcup_{k<\omega} e_{k+1} F(p_k e_k, p_k e_k) p_{k+1} \\
&= \bigsqcup_{k<\omega} e_{k+1} p_{k+1} && \text{by (iii)(d)} \\
&= id && \text{by (iii)(e).}
\end{aligned}$$

(v) Verification of minimal invariant property

For each $n \geq 0$, note that

$$
\begin{aligned}
i \circ F(p_n, e_n) &= \Big(\bigsqcup_{m<\omega} e_{m+1} F(e_m, p_m)\Big) F(p_n, e_n) \\
&= \bigsqcup_{m>n} e_{m+1} F(p_n e_m, p_m e_n) \\
&= \bigsqcup_{m>n} e_{m+1} F(r_{mn}, i_{nm}) \qquad \text{by def}^n \text{ of } e \\
&= \bigsqcup_{m>n} e_{m+1} F(r_{m-1}, i_{m-1}) F(r_{m-2}, i_{m-2}) \cdots F(r_n, i_n) \\
&= \bigsqcup_{m>n} e_{m+1}\, i_m\, i_{m-1} \cdots i_{n+1} \\
&= \cdots = \bigsqcup_{m>n} e_{n+1} \\
&= e_{n+1}
\end{aligned}
$$

Similarly $F(e_n, p_n) \circ j = \cdots = p_{n+1}$.

Hence

$$
\begin{aligned}
e_{n+1} p_{n+1} &= i F(p_n, e_n) F(e_n, p_n) j \\
&= i F(e_n p_n, e_n p_n) j \\
&= \delta(e_n p_n)
\end{aligned}
$$

where δ is as in Definition 9.3. Since $e_0 p_0 = \perp_{D \to D}$, it follows (by induction on n) that $\forall n < \omega\, (\delta^n(\perp) = e_n p_n)$. Hence $\mu(\delta) = \bigsqcup_{n<\omega} e_n p_n = id$, by (iii)(e).

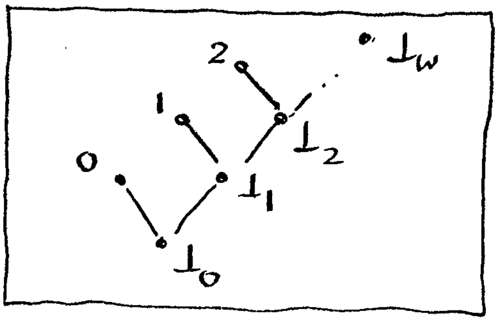
$\square$ Thm 9.4

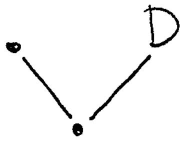
9.5 Example

Let $F : \mathbf{Cpo}_\perp^{op} \times \mathbf{Cpo}_\perp \to \mathbf{Cpo}_\perp$ be the locally continuous functor given on objects by

$$F(D^-, D^+) = (1 + D^+)_\perp$$

Let D be



Clearly $F(D,D) = (1+D)_\perp =$ 

$\cong D$

Check that this D is a minimal invariant for F.

9.6 Proposition

Let $i : F(D,D) \cong D$ be as in Definition 9.3. For any pair of morphisms

$$f : A \multimap F(B,A), \quad g : F(A,B) \multimap B$$

in $\mathbf{Cpo}_\perp$, there are unique morphisms

$$h : A \multimap D, \quad k : D \multimap B$$

making

$$\begin{tikzcd} D \arrow[r, "i^{-1}", "\cong"'] & F(D,D) \\ A \arrow[u, "h"] \arrow[r, "f"'] & F(B,A) \arrow[u, "F(k,h)"'] \end{tikzcd} \qquad \begin{tikzcd} F(D,D) \arrow[r, "i", "\cong"'] \arrow[d, "F(h,k)"'] & D \arrow[d, "k"] \\ F(A,B) \arrow[r, "g"'] & B \end{tikzcd}$$

commute in $\mathbf{Cpo}_\perp$.

Proof

Existence of h, k: let (h,k) be the least prefixed point of the continuous function

$$\varphi : (A \multimap D) \times (D \multimap B) \longrightarrow (A \multimap D) \times (D \multimap B)$$
$$(u,v) \mapsto (i F(v,u) f,\ g F(u,v) i^{-1})$$

Thus

$$\begin{cases} h = i F(k,h) f & \text{, ie } \quad i^{-1} h = F(k,h) f \\ k = g F(h,k) i^{-1} & \text{, ie } \quad k i = g F(h,k) \end{cases}$$

as required.

Uniqueness of h, k: suppose (h', k') are another such pair. Consider

$$S \stackrel{\text{def}}{=} \{ e \in (D \multimap D) \mid e h \sqsubseteq h' \ \&\ k e \sqsubseteq k' \}$$

Clearly S is an admissible subset of $D \multimap D$

Moreover, if $e \in S$, then

$$\begin{aligned} \delta(e) h &= i F(e,e) i^{-1} i F(k,h) f \\ &= i F(ke, eh) f \\ &\sqsubseteq i(k', h') f \\ &= h' \end{aligned}$$

and
$$\begin{aligned} k \delta(e) &= g F(h,k) i^{-1} i F(e,e) i^{-1} \\ &= g F(eh, ke) i^{-1} \\ &\sqsubseteq g(h', k') i^{-1} \\ &= k' \end{aligned}$$

so that $\delta(e) \in S$. Hence by Scott's Fixed Point Induction Principle (7.5), $\mu(\delta) \in S$. But by minimal invariant assumption, $id = \mu(\delta)$. So $id \in S$, i.e. $h \sqsubseteq h'$ & $k \sqsubseteq k'$.

A symmetric argument gives $h' \sqsubseteq h$ & $k' \sqsubseteq k$.

$\square$

9.7 Corollary

The minimal invariant for a locally continuous functor $F : Cpo_\perp^{op} \times Cpo_\perp \to Cpo_\perp$ is unique up to isomorphism

Proof

If (D, i) and (D', i') are both minimal invariants for F, they both satisfy the "universal property" in Proposition 9.6. Applying that property for (D, i) to $A = D' = B$, $f = (i')^{-1}$, $g = i'$ we get $h : D' \multimap D$, $k : D \multimap D'$ such that

$$i^{-1} h = F(k, h)(i')^{-1} \quad \& \quad i' F(h, k) = k i$$

Hence

$$i^{-1}(hk) = F(hk, hk) i^{-1} \quad \& \quad i F(hk, hk) = (hk) i$$

But $i^{-1} id = F(id, id) i^{-1}$ & $i F(id, id) = id\ i$,
so by the uniqueness part of the universal property $hk = id_D$. A symmetric argument gives $kh = id_{D'}$.
Thus $k : D \cong D'$.

$\square$

10. DENOTATIONAL SEMANTICS

of the programming language $\mathbb{L}$.

Let D be the minimal invariant domain for the locally continuous functor of Example 9.2:

$$F(D^-, D^+) \stackrel{\text{def}}{=} (C + (D^+ \times D^+) + (D^- \to D^+))_\perp$$

Where $C = \{\text{true}, \text{false}\} \cup \mathbb{Z}$ (discrete cpo).

Thus D comes equipped with an isomorphism

$$i : (C + (D \times D) + (D \to D))_\perp \cong D$$

satisfying the property in 9.3. Let

$$\begin{array}{lll} \text{const} & : & C \hookrightarrow (C + (D \times D) + (D \to D))_\perp \stackrel{i}{\cong} D \\ \text{pr} & : & (D \times D) \hookrightarrow (C + (D \times D) + (D \to D))_\perp \stackrel{i}{\cong} D \\ \text{fun} & : & (D \to D) \hookrightarrow (C + (D \times D) + (D \to D))_\perp \stackrel{i}{\cong} D \end{array}$$

be the restriction of i to the various summands.

Thus:

(A) const, pr, fun are continuous and order-reflecting ($\text{fun}(f) \sqsubseteq \text{fun}(f') \Rightarrow f \sqsubseteq f'$, etc.)

(B) The images of const, pr, fun are disjoint and their union is $D_\downarrow = \{d \in D \mid d \neq \perp\}$.

(C) $id : D \to D$ is the least prefixed point of the continuous function $\delta : (D \multimap D) \to (D \multimap D)$ sending $e \in (D \multimap D)$ to $\delta(e) \in (D \multimap D)$ where for all $d \in D$

$$\delta(e)(d) = \begin{cases} \perp & \text{if } d = \perp \\ \text{const}(c) & \text{if } d = \text{const}(c) \\ \text{pr}(e(d_1), e(d_2)) & \text{if } d = \text{pr}(d_1, d_2) \\ \text{fun}(e \circ f \circ e) & \text{if } d = \text{fun}(f) \end{cases}$$

We will interpret $\mathbb{L}$ programs $P \in Prog$ as elements $[\![P]\!]$ of the domain D. More generally if $\Gamma = \{x_1, \ldots, x_n\} \subseteq_{fin} Var$, open expressions $M \in Exp(\Gamma)$ will be interpreted as continuous functions $[\![(\vec{x})M]\!] : D^n \to D$ mapping a choice $\vec{d} \in D^n$ of elements of D for the variables $\vec{x}$ to an element $[\![(\vec{x})M]\!](\vec{d}) \in D$. [NB D^n is the n-fold Cartesian product $D \times \cdots \times D$]

The definition of $[\![(\vec{x})M]\!](\vec{d})$ is given by induction on the structure of M (more precisely, by induction on the proof of $\vec{x} \vdash M$). The definition makes use of certain constructions on continuous functions. The proof of the following Lemma is left as an exercise.

10.1 Lemma

(i) The insertion $X \hookrightarrow X_\perp$ of a cpo into its lift is a continuous function.

(ii) If $f : X \times Y \to D$ is continuous, where X, Y are cpos and D a domain, then so is the function $X \times Y_\perp \to D$ defined by

$$(x, u) \mapsto \begin{cases} f(x, u) & \text{if } u \in Y \\ \perp & \text{if } u = \perp \end{cases}$$

(iii) Given $X_1, X_2 \in Cpo$, the projection functions

$$X_1 \times X_2 \to X_i \qquad (i = 1, 2)$$
$$(x_1, x_2) \mapsto x_i$$

are continuous.

(iv) If $f : X \to Y$, $g : X \to Z$ are continuous fns between cpos, so is $\langle f, g\rangle : X \to Y \times Z$, where
$$\langle f, g\rangle(x) = (f(x), g(x)).$$

(v) The application function
$$\begin{aligned}(X \to Y) \times X &\longrightarrow Y \\ (f, x) &\longmapsto f(x)\end{aligned}$$
is continuous.

(vi) Given $f : X \times Y \to Z$ in $\mathbf{Cpo}$, the function
$$\begin{aligned}cur(f) : X &\longrightarrow (Y \to Z) \\ x &\longmapsto (\lambda y \in Y.\ f(x, y))\end{aligned}$$
is continuous

(vii) Given domains D, E, the functions
$$\begin{aligned}D &\longrightarrow D \oplus E \\ d &\longmapsto \begin{cases} inl(d) \text{ if } d \neq \bot \\ \bot \quad \text{ if } d = \bot\end{cases}\end{aligned} \qquad \begin{aligned}E &\longrightarrow D \oplus E \\ e &\longmapsto \begin{cases} inr(e) & e \neq \bot \\ \bot & e = \bot\end{cases}\end{aligned}$$
are strict & continuous.

(viii) Given $f : D \multimap F$, $g : E \multimap F$ in $\mathbf{Cpo}_\bot$, the function $[f, g] : D \oplus E \longrightarrow F$ defined by
$$[f, g](u) = \begin{cases} f(d) & \text{if } u = inl(d), \bot \neq d \in D \\ g(e) & \text{if } u = inr(e), \bot \neq e \in E \\ \bot & \text{if } u = \bot \end{cases}$$
is strict & cts.

(ix) For each domain D, the least prefixed point operation
$$\begin{aligned}\mu : (D \to D) &\longrightarrow D \\ f &\longmapsto \bigsqcup_{n<\omega} f^n(\bot)\end{aligned}$$
is continuous.

$\square$

10.2 Definition

For each $M \in Exp(x_1, \ldots, x_n)$, define

$$[\![(x_1, \ldots, x_n) M]\!] \in (D^n \to D)$$

by induction on the proof of $\vec{x} \vdash M$, as follows:

(i) $[\![(\vec{x}) x_i]\!] (\vec{d}) = d_i$ and $[\![(\vec{x}) c]\!] (\vec{d}) = const(c)$

(ii)
$$[\![(\vec{x})\, M_1 \text{ op } M_2]\!] (\vec{d}) = \begin{cases} const(c) & \text{if } [\![(\vec{x}) M_i]\!] (\vec{d}) = const(n_i) \\ & (i=1,2) \ \&\ c = n_1 \text{ op } n_2 \\ \perp & \text{otherwise} \end{cases}$$

(iii)
$$[\![(\vec{x}) \text{ if } B \text{ then } M \text{ else } N]\!] (\vec{d}) = \begin{cases} [\![(\vec{x}) M]\!] (\vec{d}) & \text{if } [\![(\vec{x}) B]\!] (\vec{d}) = const(true) \\ [\![(\vec{x}) N]\!] (\vec{d}) & \text{if } [\![(\vec{x}) B]\!] (\vec{d}) = const(false) \\ \perp & \text{otherwise} \end{cases}$$

(iv) $[\![(\vec{x}) (M_1, M_2)]\!] (\vec{d}) = pair([\![(\vec{x}) M_1]\!] (\vec{d}), [\![(\vec{x}) M_2]\!] (\vec{d}))$

(v) $[\![(\vec{x}) \text{ split } M \text{ as } (y, z) \text{ in } N]\!] (\vec{d}) =$

$$\begin{cases} [\![(\vec{x}, y, z) N]\!] (\vec{d}, d_1, d_2) & \text{if } [\![(\vec{x}) M]\!] (\vec{d}) = pair(d_1, d_2) \\ \perp & \text{otherwise} \end{cases}$$

(vi) $[\![(\vec{x})\,\lambda y.M]\!](\vec{d}) = \mathrm{fun}\left(\lambda d \in D.[\![(\vec{x},y)M]\!](\vec{d},d)\right)$

[NB $\lambda d \in D.[\![(\vec{x}y)M]\!](\vec{d},d)$ is in $(D \to D)$ because, by induction hypothesis, $[\![(\vec{x}y)M]\!]$ is a cts function.]

(vii) $$[\![(\vec{x})FM]\!](\vec{d}) = \begin{cases} f([\![(\vec{x})M]\!](\vec{d})) & \text{if } [\![(\vec{x})F]\!](\vec{d}) = \mathrm{fun}(f) \\ \bot & \text{otherwise} \end{cases}$$

(viii) $[\![(\vec{x})\,\mathrm{rec}\,y.M]\!](\vec{d}) = \mu\left(\lambda d \in D.[\![(\vec{x},y)M]\!](\vec{d},d)\right)$

[NB as in (vi)]

As well as defining the denotations of expressions, we can define the denotations of extended expressions (cf. §4): given $M \in Exp^*(\xi, x_1,\dots,x_n)$ where $ar(\xi) = m$ say, we get a continuous function

$$[\![(\xi, x_1,\dots,x_n)M]\!] : (D^m \to D) \times D^n \longrightarrow D$$

by induction on the proof of $\xi, \vec{x} \vdash^* M$, using clauses like (i) – (viii), plus in case $M = \xi(M_1,\dots,M_m)$

(ix) $$[\![(\xi,\vec{x})\,\xi(M_1,\dots,M_m)]\!](f,\vec{d}) = f\left([\![(\xi,\vec{x})M_1]\!](f,\vec{d}),\dots,[\![(\xi,\vec{x})M_m]\!](f,\vec{d})\right)$$

10.3 Notation

When $P \in \mathrm{Prog}\left(\stackrel{\mathrm{def}}{=} \mathrm{Exp}(\phi)\right)$, from 10.2 we get $[\![()P]\!] : D^{0} \to D$. We write $[\![()P]\!]()$ just as

$$[\![P]\!] \in D$$

and call it the denotation of the program P.

Similarly when $P \in \mathrm{Exp}^*(\xi)$, we get

$$[\![(\xi)P]\!] : (D^{m} \to D) \times D^{0} \to D$$

and write

$$[\![(\xi)P]\!] \in (D^{m} \to D) \to D \quad (ar(\xi) = m)$$

for $\lambda f \in D^{m} \to D.\ [\![(\xi)P]\!](f, ())$.

10.4 Lemma (Compositionality of $[\![-]\!]$)

(i) If $M \in \mathrm{Exp}(\vec{x})$ and $N \in \mathrm{Exp}(\vec{x}, y)$ (so that $N[M/y] \in \mathrm{Exp}(\vec{x})$), then

$$[\![(\vec{x})N[M/y]]\!](\vec{d}) = [\![(\vec{x}y)N]\!](\vec{d}, [\![(\vec{x})M]\!](\vec{d}))$$

In particular, if $P \in \mathrm{Prog}$ & $N \in \mathrm{Exp}(y)$, then

$$[\![N[P/y]]\!] = [\![(y)N]\!]([\![P]\!])$$

(ii) If $M \in \mathrm{Exp}(\vec{x}, \vec{y})$ and $N \in \mathrm{Exp}^*(\xi, \vec{x})$, (where $ar(\xi) = \mathrm{length}\ \vec{y}$; then $(N[(\vec{y})M/\xi] \in \mathrm{Exp}(\vec{x})$ and)

$$[\![(\vec{x})N[(\vec{y})M/\xi]]\!](\vec{d}) = [\![(\xi, \vec{x})N]\!]\left([\![(\vec{x}\vec{y})M]\!](\vec{d}, -), \vec{d}\right)$$

In particular, when $\{\vec{x}\} = \phi$

$$[\![N[(\vec{y})M/\xi]]\!] = [\![(\xi)N]\!]([\![(\vec{y})M]\!])$$

Proof

Both (i) & (ii) can be proved by induction on the proof of $\vec{x}, y \vdash N$ (resp. $\xi, \vec{x} \vdash^{*} N$).

$\square$

10.5 Proposition (Soundness of the denotational semantics)

Given $P \in Prog$ and $V \in Val$,

$$P \Downarrow V \Rightarrow [\![P]\!] = [\![V]\!]$$

Proof

One checks that $\{ (P, V) \mid [\![P]\!] = [\![V]\!] \}$ is closed under the rules in 3.2 defining $\Downarrow$. Lemma 10.4 (i) is needed for ($\Downarrow$ SPLIT), ($\Downarrow$ APP) and ($\Downarrow$ REC).

$\square$

10.6 Lemma

(i). $\forall V \in Val.\ [\![V]\!] \neq \perp$

(ii) $\forall V, V' \in Val.\ [\![V]\!] \sqsubseteq [\![V']\!] \Rightarrow obs(V) = obs(V')$

Proof

Both (i) & (ii) are immediate from the definition of $[\![-]\!]$ in 10.2.

$\square$

So far we have only used the fact that D is an invariant domain for

$$F(D^-, D^+) = (C + (D^+ \times D^+) + (D^- \to D^+))_\perp$$

(I.e. 10.2 – 10.6 all follow from the mere existence of an iso $F(D,D) \cong D$.) The next, key, result depends crucially upon the minimal invariant property of D (property (c) on page 10.1). We postpone its proof until we have drawn some consequences from it.

10.7 Proposition

For all $P \in Prog$, if $[\![P]\!] \neq \perp$ in D then $P \Downarrow V$ for some $V \in Val$.

(Proof – postponed.)

10.8 Corollary (Computational adequacy of $[\![-]\!]$)

Given $M, M' \in Exp(x_1, \ldots, x_n)$, if

$$[\![(\vec{x})M]\!] \sqsubseteq [\![(\vec{x})M']\!] \quad \text{in} \quad D^n \to D$$

then $\vec{x} \vdash M \lesssim M'$. Hence

$$[\![(\vec{x})M]\!] = [\![(\vec{x})M']\!] \Rightarrow \vec{x} \vdash M \simeq M'.$$

Proof

Suppose $[\![(\vec{x})M]\!] \sqsubseteq [\![(\vec{x})M']\!]$. Then for any $P \in Exp^*(\xi)$ (where $ar(\xi) = n$), by lemma 10.4(ii) we have

$$\begin{aligned} [\![P[(\vec{x})M/\xi]]\!] &= [\![(\xi)P]\!]\left([\![(\vec{x})M]\!]\right) \\ &\sqsubseteq [\![(\xi)P]\!]\left([\![(\vec{x})M']\!]\right) \\ &= [\![P[(\vec{x})M'/\xi]]\!] \end{aligned}$$

Thus if $P[(\vec{x})M/\xi] \Downarrow V$, by Proposition 10.5
$[\![V]\!] = [\![P[(\vec{x})M/\xi]]\!] \sqsubseteq [\![P[(\vec{x})M'/\xi]]\!]$.
Since by 10.6(i), $[\![V]\!] \neq \bot$, we must have
$\bot \neq [\![P[(\vec{x})M'/\xi]]\!]$, so by Proposition 10.7
$P[(\vec{x})M'/\xi] \Downarrow V'$ for some V'. Moreover,
since (by 10.5 again)

$$[\![V']\!] = [\![P[(\vec{x})M'/\xi]]\!] = \cdots \sqsupseteq [\![V]\!]$$

by 10.6(ii) $obs(V) = obs(V')$. Thus we have shown

$$\forall V.\ P[(\vec{x})M/\xi] \Downarrow V \Rightarrow \exists V'(P[(\vec{x})M'/\xi] \Downarrow V' \ \& \ obs(V) = obs(V'))$$

Since this holds for any P, by definition we have
$\vec{x} \vdash M \precsim M'$.

$\square$

<u>10.9 Proposition</u> (cf. 4.10(vii))

Suppose $M \in Exp(x)$, $N \in Exp(y)$. Then for any $P \in Prog$

$$N[rec\,x.M/y] \precsim P \Leftrightarrow \forall n<\omega\,(N[rec^{(n)}x.M/y] \precsim P)$$

where

$$\begin{cases} rec^{(0)}x.M \overset{def}{=} \Omega \overset{def}{=} rec\,x.x \\ rec^{(n+1)}x.M \overset{def}{=} M[rec^{(n)}x.M/x] \end{cases}$$

<u>Proof</u>

$\Rightarrow$: Using Theorem 5.14 ($\lesssim$ & $\precsim$ coincide) plus $\forall P(\Omega \lesssim P)$ and $M[rec\,x.M/x] \sim rec\,x.M$ (which hold by Remark 5.3), we have

$$\forall n<\omega\,(\mathrm{rec}^{(n)}x.M \sqsubseteq \mathrm{rec}\,x.M)$$

Hence (by 4.9(viii))

$$\forall n<\omega\,(N[\mathrm{rec}^{(n)}x.M/y] \sqsubseteq N[\mathrm{rec}\,x.M/y]).$$

which immediately gives $\Rightarrow$.

$\Leftarrow$: First note that

$$[\![\mathrm{rec}^{(0)}x.M]\!] = [\![\Omega]\!] = [\![\mathrm{rec}\,x.x]\!]$$

$$= \mu(\lambda d\in D.\,[\![(x)x]\!](d))$$

$$= \mu(\lambda d\in D.d)$$

$$= \bot_D$$

and $[\![\mathrm{rec}^{(n+1)}x.M]\!] = [\![M[\mathrm{rec}^{(n)}x.M/x]]\!]$

$$= [\![(x)M]\!]([\![\mathrm{rec}^{(n)}x.M]\!])$$

Thus

$$\bigsqcup_{n<\omega}[\![\mathrm{rec}^{(n)}x.M]\!] = \mu([\![(x)M]\!]) = [\![\mathrm{rec}\,x.M]\!]$$

and hence by continuity of $[\![(y)N]\!]$

$$[\![N[\mathrm{rec}\,x.M/y]]\!] = [\![(y)N]\!]([\![\mathrm{rec}\,x.M]\!])$$

$$= \bigsqcup_{n<\omega}[\![(y)N]\!]([\![\mathrm{rec}^{(n)}x.M]\!])$$

$$= \bigsqcup_{n<\omega}[\![N[\mathrm{rec}^{(n)}x.M/y]]\!]$$

Thus for any $Q\in Exp^*(\xi)$ $(ar(\xi)=0)$, if $Q[()N[\mathrm{rec}\,x.M/y]/\xi]\Downarrow V$, then by continuity of $[\![(\xi)Q]\!]$:

$$\bot \neq [\![V]\!] = [\![(\xi)Q]\!]([\![N[\mathrm{rec}\,x.M/y]]\!])$$

$$= \bigsqcup_{n<\omega}[\![(\xi)Q]\!]([\![N[\mathrm{rec}^{(n)}x.M/y]]\!])$$

$$= \bigsqcup_{n<\omega} [\![Q[{}^{()}N[rec^{(n)}x.M/y]/_{\xi}]]\!]$$

and hence for some $n<\omega$

$$[\![Q[{}^{()}N[rec^{(n)}x.M/y]/_{\xi}]]\!] \neq \perp$$

and thus by 10.7

$$\exists V'.\ Q[{}^{()}N[rec^{(n)}x.M/y]/_{\xi}] \Downarrow V'$$

Since by hypothesis $N[rec^{(n)}x.M/y] \sqsubseteq P$ & hence also $Q[{}^{()}N[rec^{(n)}x.M/y]/_{\xi}] \sqsubseteq Q[{}^{()}P/_{\xi}]$, it follows that

$$\exists V''.\ (Q[{}^{()}P/_{\xi}] \Downarrow V'' \ \&\ obs(V') = obs(V''))$$

Since $[\![V']\!] \sqsubseteq [\![V]\!]$ by construction, we also have $obs(V) = obs(V')$. Thus altogether, if $Q[{}^{()}N[rec\,x.M/y]/_{\xi}] \Downarrow V$, then $Q[{}^{()}P/_{\xi}] \Downarrow V''$ for some V'' with $obs(V) = obs(V'')$. So by definition

$$N[rec\,x.M/y] \sqsubseteq P$$

as required.

$\square$

11. PROOF OF COMPUTATIONAL ADEQUACY

We give the proof of Proposition 10.7, i.e. that

$$[\![P]\!] \neq \perp \Rightarrow \exists V (P \Downarrow V).$$

The strategy of the proof we give is due to Plotkin, with simplifications introduced recently by the author (see: A.M. Pitts, "Relational Properties of Domains", Univ. Camb. Computer Lab. Tech. Rpt. 321, Dec. 1993.)

11.1 Definition

Let D be the domain used in the previous section. A formal approximation relation is a binary relation $\lhd \subseteq D \times Prog$ satisfying:

(a) For all $P \in Prog$, $\{d \in D \mid d \lhd P\}$ is an admissible subset of D (cf. Definition 7.4).

(b) For all $d \in D$ & $P \in Prog$, $d \lhd P$ holds iff

$$\begin{aligned}
&d = \perp \\
\vee\ &\exists c \in Const\,(d = const(c)\ \&\ P \Downarrow c) \\
\vee\ &\exists d_1, d_2 \in D, P_1, P_2 \in Prog\,(d = pair(d_1, d_2)\ \& \\
&\qquad P \Downarrow (P_1, P_2)\ \&\ d_1 \lhd P_1\ \&\ d_2 \lhd P_2) \\
\vee\ &\exists f \in D \to D, \lambda x.M \in Val\,(d = fun(f)\ \& \\
&\qquad P \Downarrow \lambda x.M\ \&\ \forall d', P' (d' \lhd P' \Rightarrow f(d') \lhd M[P'/x]))
\end{aligned}$$

11.2 Remark

We can rephrase Definition 11.1 as a fixed point problem. Let

$$\mathcal{R} \stackrel{\text{def}}{=} \{ R \subseteq D \times Prog \mid \forall P \in Prog \; \{ d \mid (d,P) \in R \} \text{ admissible} \}$$

It is not hard to see that $\mathcal{R}$ is closed under arbitrary intersections in $\mathcal{P}(D \times Prog)$: hence $(\mathcal{R}, \subseteq)$ is a complete lattice (cf. 1.8). Define

$$\Phi : \mathcal{R} \to \mathcal{R}$$

by

$$\begin{aligned}\Phi(R) = \{ (d,P) \mid\; & d = \bot \;\vee\; \exists c\,(d = const(c) \;\&\; P \Downarrow c) \\ & \vee\; \exists d_1, d_2, P_1, P_2 \,(d = pair(d_1, d_2) \;\& \\ & \quad P \Downarrow (P_1, P_2) \;\&\; (d_1, P_1) \in R \;\&\; (d_2, P_2) \in R) \\ & \vee\; \exists f, \lambda x.M\,(d = fun(f) \;\&\; P \Downarrow \lambda x.M \\ & \quad , \forall (d', P') \in R.\; (f(d'), M[P'/x]) \in R) \}\end{aligned}$$

Then a formal approximation relation is precisely a fixed point, $\triangleleft = \Phi(\triangleleft)$, of Φ. But we cannot appeal to Theorem 1.9 (Tarski-Knaster Fixed Point Theorem for complete lattices) since unfortunately Φ is not monotone. (Why not? because of the negative occurence of R in $\Phi(R)$ at $\ldots \forall (d', P') \in R \ldots$)

However, if we can demonstrate the existence of a fixed point, 10.7 follows, because ...

11.2 Proposition

For all $M \in Exp(x_1, \ldots, x_n)$ if
$$d_1 \lhd P_1 \ \& \cdots \& \ d_n \lhd P_n$$
then
$$[\![(\vec{x}) M]\!](\vec{d}) \lhd M[\vec{P}/\vec{x}]$$

Proof

This can be proved by induction on the proof of $x_1, \ldots, x_n \vdash M$. For example, suppose M is $rec\, y. N$ and, inductively, the property holds for $N \in Exp(\vec{x}, y)$. Since
$$[\![(\vec{x})\, rec\, y. N]\!](\vec{d}) = \bigsqcup_{n<w} f^n(\bot)$$

where $f \stackrel{def}{=} \lambda d \in D. [\![(\vec{x} y) N]\!](\vec{d}, d)$,

by the admissibility property 11.1 (a) of $\lhd$, it suffices to show
$$\forall n<w.\ f^n(\bot) \lhd (rec\, y. N)[\vec{P}/\vec{x}]$$
This can be done by induction on n: case $n=0$ holds because $\forall P (\bot \lhd P)$; and if
$$f^n(\bot) \lhd (rec\, y. N)[\vec{P}/\vec{x}]$$
then by induction hypothesis on N
$$f^{n+1}(\bot) = [\![(\vec{x} y) N]\!](\vec{d}, f^n(\bot)) \lhd N[\vec{P}/\vec{x}, rec\, y. N[\vec{P}/\vec{x}]/y]$$
and hence
$$f^{n+1}(\bot) \lhd rec\, y. N[\vec{P}/\vec{x}]$$
(since it follows from 11.1(b), that : $d \lhd P \ \& \ \forall v (P \Downarrow v \Leftrightarrow Q \Downarrow v) \Rightarrow d \lhd Q$).

The proof for other cases of the structure of M are simpler, and are omitted. $\square$

11.3 Corollary

For all $P \in Prog$, $[\![P]\!] \lhd P$

Proof

This is just the $n=0$ case of 11.2. $\square$

Thus if $[\![P]\!] \neq \bot$, then from 11.1(b) and $\bot \neq [\![P]\!] \lhd P$, it follows immediately that $P \Downarrow V$ for some V, as required for 10.7.

So it just remains to show that a relation $\lhd$ as in 11.1 exists.

With $\mathcal{R}$ as in 11.2, given $R^-, R^+ \in \mathcal{R}$, define

$$\begin{aligned}\Psi(R^-, R^+) \overset{\text{def}}{=} \{(d,P) \mid\ & d = \bot \vee \exists c\,(d = const(c) \,\&\, P \Downarrow c) \\ & \vee \exists d_1, d_2, P_1, P_2\,(d = pair(d_1, d_2) \\ & \quad \& \, P \Downarrow (P_1, P_2) \,\&\, (d_1, P_1) \in R^+ \,\&\, (d_2, P_2) \in R^+) \\ & \vee \exists f, \lambda x.M\,(d = fun(f) \\ & \quad \& \, P \Downarrow \lambda x.M \\ & \quad \& \, \forall (d', P') \in R^-.\,(f(d'), M[P'/x]) \in R^+)\}\end{aligned}$$

The following properties are easy to check

- $\Phi(R) = \Psi(R, R)$

- Ψ determines a monotone function
$$\mathcal{R}^{op} \times \mathcal{R} \to \mathcal{R}$$
and hence $\Psi^{\S} : \mathcal{R}^{op} \times \mathcal{R} \to \mathcal{R}^{op} \times \mathcal{R}$
defined by
$$\Psi^{\S}(R^-, R^+) = (\Psi(R^+, R^-), \Psi(R^-, R^+))$$
is a monotone operator on the complete lattice $\mathcal{R}^{op} \times \mathcal{R}$.

So we can apply Theorem 1.9 to deduce the existence of a least pre-fixed point $(\lhd^-, \lhd^+)$ for $\Psi^{\S}$. Thus we have

(11.4) $\lhd^-, \lhd^+ \in \mathcal{R}$

(11.5) $\lhd^- = \Psi(\lhd^+, \lhd^-)$ & $\lhd^+ = \Psi(\lhd^-, \lhd^+)$

(11.6) for any $R^-, R^+ \in \mathcal{R}$, if
$$R^- \subseteq \Psi(R^+, R^-) \ \& \ \Psi(R^-, R^+) \subseteq R^+$$
then $R^- \subseteq \lhd^-$ & $\lhd^+ \subseteq R^+$.

Taking $R^- = \lhd^+$, $R^+ = \lhd^-$ in (11.6), from (11.5) we have $\lhd^+ \subseteq \lhd^-$. If we can show the reverse inclusion, and hence that $\lhd^- = \lhd^+$, then we can satisfy 11.1 with $\lhd \stackrel{\text{def}}{=} \lhd^- = \lhd^+$ (since then

$$\triangleleft = \triangleleft^{+} = \Psi(\triangleleft^{-}; \triangleleft^{+}) = \Psi(\triangleleft, \triangleleft) = \Phi(\triangleleft) \text{).}$$

So it just remains to show $\triangleleft^{-} \subseteq \triangleleft^{+}$. It is at this point that we appeal to the minimal invariant property of the domain D, ie. that $id_D = \mu(\delta)$ where $\delta : (D \multimap D) \to (D \multimap D)$ sends $e \in (D \multimap D)$ to $\delta(e) \in (D \multimap D)$, where

$$\delta(e)(d) = \begin{cases} \bot & \text{if } d = \bot \\ const(c) & \text{if } d = const(c) \\ pr(e(d_1), e(d_2)) & \text{if } d = pair(d_1, d_2) \\ fun(e \circ f \circ e) & \text{if } d = fun(f) \end{cases}$$

Given $R^{-}, R^{+} \in \mathcal{R}$, and $e \in (D \multimap D)$, write

$$e : R^{-} \subset R^{+}$$

to mean $\forall (d, P) \in R^{-}.\ (e(d), P) \in R^{+}$

It is straightforward to check from the definitions of δ and Ψ that

$$e : R^{-} \subset R^{+} \Rightarrow \delta(e) : \Psi(R^{+}, R^{-}) \subset \Psi(R^{-}; R^{+})$$

Thus by (11.5)

$$e : \triangleleft^{-} \subset \triangleleft^{+} \Rightarrow \delta(e) : \triangleleft^{-} \subset \triangleleft^{+}$$

Since $\{e \in (D \multimap D) \mid e : \triangleleft^{-} \subset \triangleleft^{+}\}$ is easily seen to be an admissible subset of $D \multimap D$, by Scott's Fixed Point Induction principle (7.5) we have

$\mu(\delta) \in \{e \in (D \multimap D) \mid e : \triangleleft^- \subset \triangleleft^+\}$. Thus $id_D = \mu(\delta) : \triangleleft^- \subset \triangleleft^+$, which means that $\triangleleft^- \subset \triangleleft^+$, as required.

So taking $\triangleleft$ to be $\triangleleft^- = \triangleleft^+$, we have constructed a relation as in 11·1 and hence completed the proof of Proposition 10·7.

12. FAILURE OF 'FULL ABSTRACTION'
for the domain-theoretic denotational semantics of $\mathbb{L}$

The converse of 10.8 (Computational Adequacy) fails: there are $P, Q \in Prog$ with $[\![P]\!] \neq [\![Q]\!] \in D$ but $P \simeq Q$.

12.1 Example (Plotkin's "parallel or" example: Ref: G.D. Plotkin, "LCF considered as a programming language", TCS 5(1977) 223-255.)

For $b \in \{true, false\}$, let $T_b \in Prog$ be

$$
\begin{aligned}
\lambda f.\ & \text{if } f(true, \Omega) = true \text{ then} \\
& \quad \text{if } f(\Omega, true) = true \text{ then} \\
& \quad\quad \text{if } f(false, false) = false \text{ then } b \text{ else } \Omega \\
& \quad \text{else } \Omega \\
& \text{else } \Omega
\end{aligned}
$$

Then $[\![T_{true}]\!] \neq [\![T_{false}]\!]$, but $T_{true} \simeq T_{false}$.

<u>$[\![T_{true}]\!] \neq [\![T_{false}]\!]$:</u>

First note that with $\mathbb{B} \stackrel{def}{=} \{true, false\}$, there is a (monotonic hence) continuous function

$$POR : \mathbb{B}_\perp \times \mathbb{B}_\perp \to \mathbb{B}_\perp$$

satisfying

$$\begin{cases} POR(true, -) = true \\ POR(-, true) = true \\ POR(false, false) = false \end{cases}$$

If D is the domain used for the denotational semantics of $\mathbb{L}$, then we can extend POR to a function $f : D \to D$ satisfying

$$f(pr(const(true), \perp)) = const(true)$$

etc.

Now $[\![T_b]\!] = fun(F_b)$ and

$$F_b(fun(f)) = b \qquad (b = true, false)$$

Hence $F_{true} \neq F_{false}$, so $[\![T_{true}]\!] \neq [\![T_{false}]\!]$.

<u>$T_{true} \cong T_{false}$</u> : By 4.10(iv)(a), it suffices to show $\forall P \in Prog$ that

$$M_{true}[P/f] \simeq M_{false}[P/f]$$

where we write $T_b = \lambda f. M_b$. In fact it is the case that $M_b[P/f] \Uparrow$ for all P, since

<u>FACT</u>: there is no $P \in \mathrm{Prog}$ such that

$$P(\mathrm{true}, \Omega) \Downarrow \mathrm{true}$$
$$P(\Omega, \mathrm{true}) \Downarrow \mathrm{true}$$
$$P(\mathrm{false}, \mathrm{false}) \Downarrow \mathrm{false}.$$

This fact can be established via the use of suitable "logical relations"....

[...but there is no time to do it!]

Recent Publications in the BRICS Notes Series

NS-94-5 **Andrew M. Pitts.** ***Some Notes on Inductive and Co-Inductive Techniques in the Semantics of Functional Programs*** **DRAFT VERSION. December 1994. vi+135 pp.**

NS-94-4 **Peter D. Mosses, editor.** ***Abstracts of the 6th Nordic Workshop on PROGRAMMING THEORY*** **(Aarhus, Denmark, 17–19 October, 1994), October 1994. v+52 pp.**

NS-94-3 **Sven Skyum, editor.** ***Complexity Theory: Present and Future*** **(Aarhus, Denmark, 15–18 August, 1994), September 1994. v+213 pp.**

NS-94-2 **David Basin.** ***Induction Based on Rippling and Proof Planning. Mini-Course*****. August 1994. 62 pp.**

NS-94-1 **Peter D. Mosses, editor.** ***Proc. 1st International Workshop on Action Semantics*** **(Edinburgh, 14 April, 1994), May 1994. 145 pp.**