iSAQB Advanced DSL - Lambda Calculus

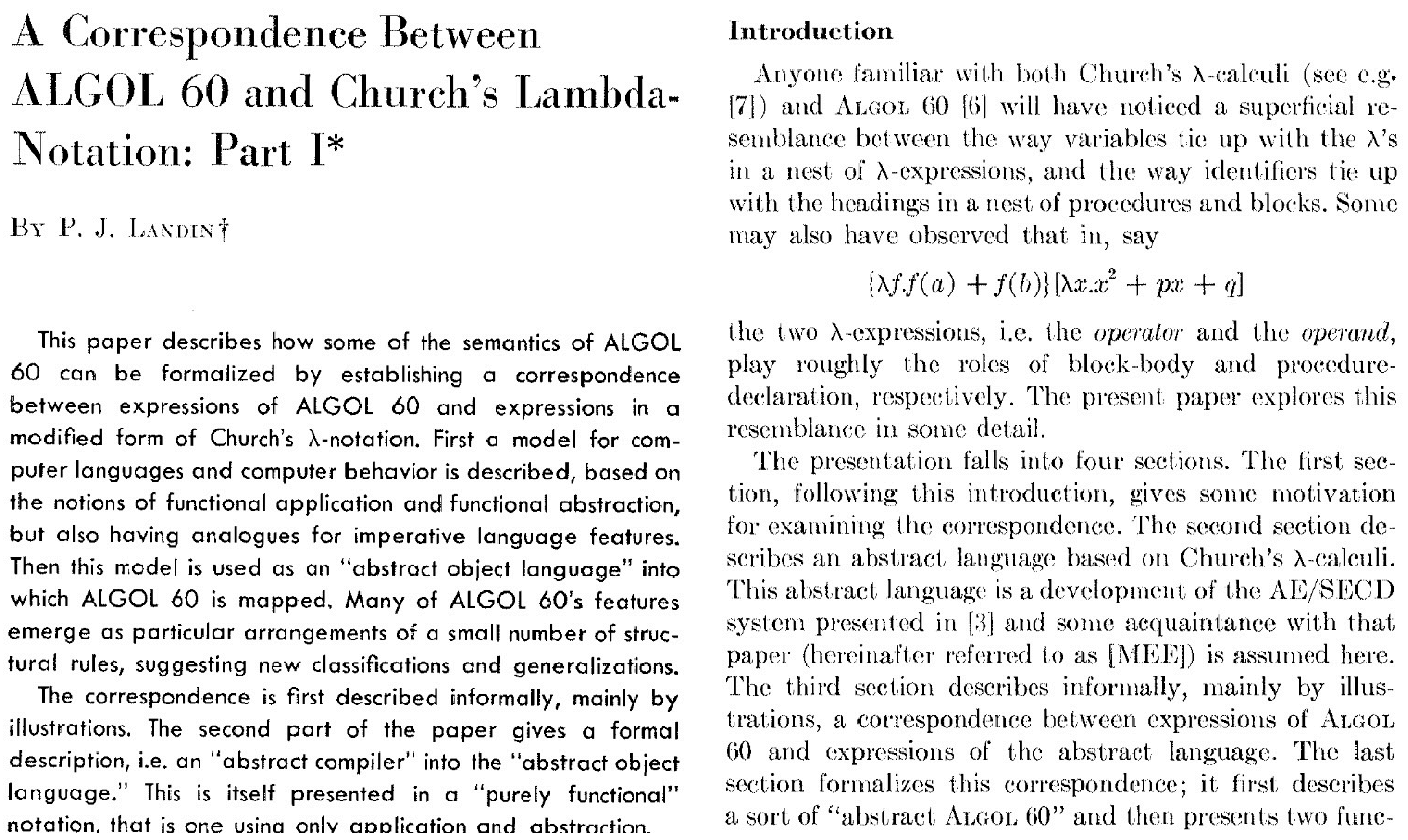
Michael Sperber

Created: 2024-07-03 Wed 17:19

Recurring Problems in Language Design

- naming
- semantics
- implementation
- "essence"

(LG 3-3) The Next 700 Programming Languages



A Correspondence Between ALGOL 60 and Church's Lambda-Notation: Part I*

By P. J. Landin†

This paper describes how some of the semantics of ALGOL 60 can be formalized by establishing a correspondence between expressions of ALGOL 60 and expressions in a modified form of Church's λ-notation. First a model for computer languages and computer behavior is described, based on the notions of functional application and functional abstraction, but also having analogues for imperative language features. Then this model is used as an "abstract object language" into which ALGOL 60 is mapped. Many of ALGOL 60's features emerge as particular arrangements of a small number of structural rules, suggesting new classifications and generalizations.

The correspondence is first described informally, mainly by illustrations. The second part of the paper gives a formal description, i.e. an "abstract compiler" into the "abstract object language." This is itself presented in a "purely functional" notation, that is one using only application and abstraction.

Introduction

Anyone familiar with both Church's λ-calculi (see e.g. [7]) and ALGOL 60 [6] will have noticed a superficial resemblance between the way variables tie up with the λ's in a nest of λ-expressions, and the way identifiers tie up with the headings in a nest of procedures and blocks. Some may also have observed that in, say

$$\{\lambda f.f(a) + f(b)\}[\lambda x.x^2 + px + q]$$

the two λ-expressions, i.e. the *operator* and the *operand*, play roughly the roles of block-body and procedure-declaration, respectively. The present paper explores this resemblance in some detail.

The presentation falls into four sections. The first section, following this introduction, gives some motivation for examining the correspondence. The second section describes an abstract language based on Church's λ-calculi. This abstract language is a development of the AE/SECD system presented in [3] and some acquaintance with that paper (hereinafter referred to as [MEE]) is assumed here. The third section describes informally, mainly by illustrations, a correspondence between expressions of ALGOL 60 and expressions of the abstract language. The last section formalizes this correspondence; it first describes a sort of "abstract ALGOL 60" and then presents two func-

Landin: Correspondence between ALGOL 60 and Church's Lambda-notation: part I (1965)

(LG 3-2) Lambda Calculus - Language

$$\begin{array}{rcl}\langle\mathcal{L}_\lambda\rangle \rightarrow & & \langle X\rangle \\ & | & (\langle\mathcal{L}_\lambda\rangle\,\langle\mathcal{L}_\lambda\rangle) \\ & | & (\lambda\,\langle V\rangle.\,\langle\mathcal{L}_\lambda\rangle)\end{array}$$

(Variable, Application, Abstraction)

Lambda Calculus - Examples

x	Variable
$(f\ x)$	Application
$((f\ x)\ (\lambda x.\, x))$	Application
$(\lambda x.\, (\lambda y.\, (x\ y)))$	Abstraktion
$(\lambda x.\, (f\ x))$	Abstraktion
$(\lambda x.\, (f\ (\lambda x\ x)))$	Abstraktion
$((\lambda x.\, x)\ (\lambda x.\, x))$	Application

Abbreviations

abbreviation	stands for
$\lambda \mathrm{f}.\, \lambda \mathrm{x}.\, \mathrm{f\ x}$	$(\lambda \mathrm{f}.\, (\lambda \mathrm{x}.\, (\mathrm{f\ x})))$
$\lambda x_1 \ldots x_n.\, e$	$\lambda x_1.\, (\lambda x_2.\, (\ldots \lambda x_n.\, e) \ldots)$
$e_0 \ldots e_n$	$(\ldots (e_0\ e_1)\ e_2) \ldots e_n)$

"Body of a λ extends to the right as far as possible."

$(\mathrm{f\ x})$	$\mathrm{f\ x}$
$(\mathrm{f\ x})\ \mathrm{y}$	$\mathrm{f\ x\ y}$
$(\lambda \mathrm{x}.\, (\mathrm{x}\ \ x))$	$\lambda \mathrm{x}.\, \mathrm{x\ x}$
$(\lambda \mathrm{x}.\, ((\mathrm{x\ y})\ \mathrm{z}))$	$\lambda \mathrm{x}.\, \mathrm{x\ y\ z}$
$(\lambda \mathrm{x}.\, (\lambda\ \mathrm{y}.\, (\mathrm{x\ y})))$	$\lambda \mathrm{xy}.\, \mathrm{x\ y}$

Exercise

Expand the following terms:

$$\mathrm{f\ x\ y\ z}$$

$$\lambda \mathrm{xyz.\ x\ y\ z}$$

$$((\lambda \mathrm{xy.\ x\ y})\ \mathrm{a\ b})$$

Free and Bound Variables

$$\mathrm{free}(e) := \begin{cases} \{v\} & \text{falls } e = v \\ \mathrm{free}(e_0) \cup \mathrm{free}(e_1) & \text{if } e = e_0\ e_1 \\ \mathrm{free}(e_0) \setminus \{v\} & \text{if } e = \lambda v.\, e_0 \end{cases}$$

$$\mathrm{bound}(e) := \begin{cases} \varnothing & \text{if } e = v \\ \mathrm{bound}(e_0) \cup \mathrm{bound}(e_1) & \text{if } e = e_0\ e_1 \\ \mathrm{bound}(e_0) \cup \{v\} & \text{if } e = \lambda v.\, e_0 \end{cases}$$

Exercise

$$\text{free}(x\ y)$$
$$\text{free}(\lambda x.\ x\ y)$$
$$\text{free}(\lambda xy.\ x\ y)$$
$$\text{free}((\lambda x.\ y)\ x)$$
$$\text{bound}(x\ y)$$
$$\text{bound}(\lambda x.\ x\ y)$$
$$\text{bound}(\lambda xy.\ x\ y)$$
$$\text{bound}((\lambda x.\ y)\ x)$$

Substitution

For $e, f \in \mathcal{L}_\lambda$:

$$e[x \mapsto f] := \begin{cases} f & \text{if } e = x \\ e & \text{if } e \text{ is a variable and } e \neq x \\ \lambda x.\, e_0 & \text{if } e = \lambda x.\, e_0 \\ \lambda y.\, (e_0[x \mapsto f]) & \text{if } e = \lambda y.\, e_0 \text{ and } y \neq x, y \notin \\ \lambda y'.\, (e_0[y \mapsto u'][x \mapsto f]) & \text{if } e = \lambda y.\, e_0 \\ & \text{and } y \neq x, y \in \text{free}(f) \\ & \text{and } y' \notin \text{free}(e_0) \cup \text{free}(f) \\ (e_0[x \mapsto f])\ (e_1[x \mapsto f]) & \text{if } e = e_0\ e_1 \end{cases}$$

Substitution Example

$$(\lambda \mathrm{x}.\, \lambda \mathrm{y}.\, \mathrm{x}\ (\lambda \mathrm{z}.\, \mathrm{z})\ \mathrm{z})[\mathrm{z} \mapsto \mathrm{x}\ \mathrm{y}]$$

$$\begin{aligned}
&= \lambda \mathrm{x}'.\, ((\lambda \mathrm{y}.\, \mathrm{x}\ (\lambda \mathrm{z}.\, \mathrm{z})\ \mathrm{z})[\mathrm{x} \mapsto \mathrm{x}'][\mathrm{z} \mapsto \mathrm{x}\ \mathrm{y}]) \\
&= \lambda \mathrm{x}'.\, ((\lambda \mathrm{y}.\, ((\mathrm{x}\ (\lambda \mathrm{z}.\, \mathrm{z})\ \mathrm{z})[\mathrm{x} \mapsto \mathrm{x}']))[\mathrm{z} \mapsto \mathrm{x}\ \mathrm{y}]) \\
&= \lambda \mathrm{x}'.\, ((\lambda \mathrm{y}.\, (\mathrm{x}[\mathrm{x} \mapsto \mathrm{x}']\ ((\lambda \mathrm{z}.\, \mathrm{z})[\mathrm{x} \mapsto \mathrm{x}'])\ \mathrm{z}[\mathrm{x} \mapsto \mathrm{x}']))[\mathrm{z} \mapsto \mathrm{x}\ \mathrm{y}]) \\
&= \lambda \mathrm{x}'.\, ((\lambda \mathrm{y}.\, (\mathrm{x}'\ (\lambda \mathrm{z}.\, \mathrm{z})\ \mathrm{z}))[\mathrm{z} \mapsto \mathrm{x}\ \mathrm{y}]) \\
&= \lambda \mathrm{x}'.\, \lambda \mathrm{y}'.\, ((\mathrm{x}'\ (\lambda \mathrm{z}.\, \mathrm{z})\ \mathrm{z})[\mathrm{y} \mapsto \mathrm{y}'][\mathrm{z} \mapsto \mathrm{x}\ \mathrm{y}]) \\
&= \lambda \mathrm{x}'.\, \lambda \mathrm{y}'.\, ((\mathrm{x}'[\mathrm{y} \mapsto \mathrm{y}']\ ((\lambda \mathrm{z}.\, \mathrm{z})[\mathrm{y} \mapsto \mathrm{y}'])\ \mathrm{z}[\mathrm{y} \mapsto \mathrm{y}'])[\mathrm{z} \mapsto \mathrm{x}\ \mathrm{y}]) \\
&= \lambda \mathrm{x}'.\, \lambda \mathrm{y}'.\, ((\mathrm{x}'\ (\lambda \mathrm{z}.\, \mathrm{z})\ \mathrm{z})[\mathrm{z} \mapsto \mathrm{x}\ \mathrm{y}]) \\
&= \lambda \mathrm{x}'.\, \lambda \mathrm{y}'.\, \mathrm{x}'[\mathrm{z} \mapsto \mathrm{x}\ \mathrm{y}]\ ((\lambda \mathrm{z}.\, \mathrm{z})[\mathrm{z} \mapsto \mathrm{x}\ \mathrm{y}])\ \mathrm{z}[\mathrm{z} \mapsto \mathrm{x}\ \mathrm{y}] \\
&= \lambda \mathrm{x}'.\, \lambda \mathrm{y}'.\, \mathrm{x}'\ (\lambda \mathrm{z}.\, \mathrm{z})\ (\mathrm{x}\ \mathrm{y})
\end{aligned}$$

Reduction

$$\begin{aligned} \lambda x.\, e &\rightarrow_\alpha \lambda y.\, (e[x \mapsto y]) \quad y \notin \mathrm{free}(e) \\ (\lambda x.\, e)\ f &\rightarrow_\beta e[x \mapsto f] \end{aligned}$$

A λ term has at most one normal form modulo α reduction.

Evaluation - Call-by-Name

- For shape $(\lambda x.\, e)\ f$, the whole term is the redex.
- For shape $e\ f$, and if e is **not** an abstraction, find the redex within e - on the **left**.
- For shape $e\ f$, and if e is **not** an abstraction and when there$'$s no redex in e, find the redex within f.

Evaluation - Call-by-Value

- For shape $e\ f$, find the redex within e - **leftmost-innermost**.
- For shape $e\ f$, and if $there's$ no redex in e, find redex within f.
- For shape $(\lambda x.\, e)\ f$, and if $there's$ no redex in f, the whole term is the redex.

(LG 3-2) Exercise: Expressiveness of the lambda calculus

- How could you represent binary conditionals in the lambda calculus?

(LG 3-2) Fixpoints

$$Y = \lambda \mathrm{f}.\,(\lambda \mathrm{x}.\,\mathrm{f}\;(\mathrm{x}\;\mathrm{x}))\;(\lambda \mathrm{x}.\,\mathrm{f}\;(\mathrm{x}\;\mathrm{x}))$$

$$\forall F \in \mathcal{L}_\lambda : F(Y\;F) \equiv Y\;F$$

(LG 3-2) Church-Turing Thesis

All equivalent:

- general recursive functions (Gödel)
- Lambda calculus
- Turing machines

Applied Lambda Calculus

$B := \{\texttt{#f}, \texttt{#t}, 0, 1, 2, \ldots\}$ is the set of **base values**.

For n a natural number, Σ^n is the set of n-ary **primitives**

For each $F \in \Sigma^n$ there is an n-ary function F^{op} - its **operation**.

$V := B \cup \{(\lambda x.\, e) \mid x \text{ variable}, e \text{ term}\}$ is the set of **values**.

$b \in B, v \in V$ implicitly

Applied Lambda Calculus

$$
\begin{array}{rcl}
\langle \mathcal{L}_{\lambda A} \rangle & \rightarrow & \langle V \rangle \\
& | & (\langle \mathcal{L}_{\lambda A} \rangle \; \langle \mathcal{L}_{\lambda A} \rangle) \\
& | & (\lambda \, \langle V \rangle . \, \langle \mathcal{L}_{\lambda A} \rangle) \\
& | & \langle B \rangle \\
& | & (\left\langle \Sigma^1 \right\rangle \; \langle \mathcal{L}_{\lambda A} \rangle) \\
& | & (\left\langle \Sigma^2 \right\rangle \; \langle \mathcal{L}_{\lambda A} \rangle \; \langle \mathcal{L}_{\lambda A} \rangle) \\
& \ldots & \\
& | & (\langle \Sigma^n \rangle \; \langle \mathcal{L}_{\lambda A} \rangle \; \ldots \; \langle \mathcal{L}_{\lambda A} \rangle) \quad (n \text{ times})
\end{array}
$$

$$(F \, b_1 \; \ldots \; b_n) \rightarrow_\delta F^{\mathrm{op}}(b_1, \ldots, b_n)$$

(LG 1-2, LG 3-3) Call-by-Value Operational Semantics

$$\frac{(\lambda x.\, e)\ v \rightarrow_\beta e[x \mapsto f]}{(\lambda x.\, e)\ v \rightarrow_{\text{cbv}} e[x \mapsto f]}$$

$$\frac{e_1 \rightarrow_{\text{cbv}} e_1'}{e_1\ e_2 \rightarrow_{\text{cbv}} e_1'\ e_2}$$

$$\frac{e_2 \rightarrow_{\text{cbv}} e_2'}{v\ e_2 \rightarrow_{\text{cbv}} v\ e_2'}$$

Call-by-Value Operational Semantics

$$\frac{F \in \Sigma^n \qquad F\, b_1\ \ldots\ b_n \rightarrow_\delta F^{\mathrm{op}}(b_1, \ldots, b_n)}{F\, v_1\ \ldots\ b_n \rightarrow_{\mathrm{cbv}} F^{\mathrm{op}}(b_1, \ldots, b_n)}$$

$$\frac{F \in \Sigma^n \qquad e_i \rightarrow_{\mathrm{cbv}} e_i'}{F\, b_1 \ldots b_{i-1}\ e_i\ e_{i+1}\ e_n \ldots \rightarrow_{\mathrm{cbv}} F\, b_1 \ldots b_{i-1}\ e_i'\ e_{i+1} \ldots e_n}$$

(LG 3-1) Call-by-Value "Big Step" Evaluation Semantics

$$\Gamma \vdash \lambda x.\, e \Downarrow \lambda x.\, e$$

$$\Gamma \vdash v \Downarrow v$$

$$\frac{\Gamma(x) = v}{\Gamma \vdash x \Downarrow v}$$

$$\frac{\Gamma \vdash e_1 \Downarrow \lambda x.\, e \qquad \Gamma \vdash e_2 \Downarrow v_2 \qquad \Gamma[x \mapsto v_2] \vdash e \Downarrow v}{\Gamma \vdash e_1\; e_2 \Downarrow v}$$

Call-by-Value Reduction Semantics

$$\begin{array}{rcl} C & \rightarrow & [\cdot] \qquad\qquad \textit{evaluation context} \\ & | & C\ \langle\mathcal{L}_{\lambda A}\rangle \\ & | & \langle V\rangle\ C \\ & | & \langle\Sigma^n\rangle\ \langle B\rangle\ \ldots\ \langle B\rangle\ C\ \langle\mathcal{L}_{\lambda A}\rangle\ \ldots\ \langle\mathcal{L}_{\lambda A}\rangle \end{array}$$

$$\begin{array}{rcl} C[(\lambda x.\, e)\ f] & \rightarrow_{\text{cbv}} & C[e[x \mapsto f]] \\ C[F\, b_1\ \ldots\ b_n] & \rightarrow_{\text{cbv}} & C[F^{\text{op}}(b_1, \ldots, b_n)] \end{array}$$

(LG 3-3) Denotational Semantics

- Function from expressions to mathematical object (*denotation*).
- Typically *continuous* function between *domains* (*complete partial orders*).

(LG 3-3) Denotational Semantics of the Lambda Calculus

$$
\begin{array}{rcl}
D_\infty & \overset{\phi}{\underset{\psi}{\rightleftarrows}} & [D_\infty \to D_\infty] \\[1em]
[\![-]\!] & \in & \mathcal{L}_\lambda \to [\mathit{Env} \to D_\infty] \\[1em]
[\![x]\!]\eta & = & \eta(v) \\
[\![\lambda x.\, e]\!]\eta & = & \psi(\lambda v \in D_\infty.\, [\![e]\!]\eta[x \mapsto v]) \\
[\![e_1\; e_2]\!]\eta & = & \phi([\![e_1]\!]\eta)\; ([\![e_2]\!]\eta)
\end{array}
$$

(LG 3-3) Exercise: Informal Semantics of Tim

Describe the semantics of Tim using appropriate notation and language!

(LG 5-2) PLT Redex: Lambda Calculus

```
(define-language Lambda
  (e ::=
     x
     (lambda (x_!_ ...) e)
     (e e ...))
  (x ::= variable-not-otherwise-mentioned))
```

PLT Redex: Evaluation Context

```
(define-extended-language Lambda-calculus Lambda
  (e ::= .... n)
  (v ::= n (lambda (x ...) e))
  (n ::= number)
  (C ::= hole (e ... C e ...) (lambda (x_!_ ...) C)))
```

PLT Redex: Reduction Relation

```
(define -->βv
  (reduction-relation
   Lambda-calculus
   (--> (in-hole C ((lambda (x_1 ..._n) e) v_1 ..._n))
        (in-hole C (subst ([v_1 x_1] ...) e))
        βv)))
```