

Slides
used for the lecture of
John H. Williams

IBM Research
K54-282
5600 Cottle Road
San Jose, California 95193

Reduction Languages

1 Programming languages without:

gotos
labels
assignments
variables

- based on the combinatory calculus of Moses Schönfinkel, 1924.
- roughly speaking, Red languages are to the combinatory calculus as Lisp 1.5 is to the lambda calculus.
- defined by John Backus of IBM, their development was begun ~ 1971.
- References:

- i) "Reduction languages and variable-free programming." IBM tech. report, 1972
- ii) "Programming language semantics and closed applicative languages", ACM Symp. on POPL, Boston, 1973
- iii) "Can programming be liberated from the von Neumann style? - A functional style and its algebra of programs."
(to appear - CACM)

Objects $\mathcal{O}$

- i) atoms $\in \mathcal{O}$ and $\perp \in \mathcal{O}$
- ii) $x_1, \dots, x_n \in \mathcal{O} \Rightarrow \langle x_1, \dots, x_n \rangle \in \mathcal{O}$

Functions F

- i) $f \in F \Rightarrow f: \mathcal{O} \rightarrow \mathcal{O}$
- ii) all functions are
 - a) primitive or
 - b) generated by a functional form

Functional Forms $\mathcal{F}$

- i) if $\Gamma_n \in \mathcal{F}$ and $f_1, \dots, f_n \in F$
then $\Gamma_n(f_1, \dots, f_n) \in F$
- ii) $\mathcal{F}$ cannot be extended by definitions within the system

Application :

$f : x$ = object obtained by
applying f to x

if $(f : x)$ is an application,

then $f \in F$, $x \in \mathcal{O}$, $f : x \in \mathcal{O}$

Primitive Functions (examples)

Selectors

$$1: X \equiv X = \langle X_1, \dots, X_n \rangle \rightarrow X_1; \perp$$

$$2: X \equiv X = \langle X_1, X_2, \dots, X_n \rangle \rightarrow X_2; \perp$$

$\vdots$

Tail

$$tl: x \equiv x = \langle X_1 \rangle \rightarrow \phi$$

$$x = \langle X_1, X_2, \dots, X_n \rangle \rightarrow \langle X_2, \dots, X_n \rangle; \perp$$

Null

$$\text{null}: x \equiv x = \phi \rightarrow T; x \neq \perp \rightarrow F; \perp$$

Distribute from left (right)

$$\text{distl}: \langle x, \langle y_1, \dots, y_n \rangle \rangle \equiv \langle \langle x, y_1 \rangle, \dots, \langle x, y_n \rangle \rangle$$

$$\text{distr}: \langle \langle x_1, \dots, x_n \rangle, y \rangle \equiv \langle \langle x_1, y \rangle, \dots, \langle x_n, y \rangle \rangle$$

Transpose

$$\text{trans}: \langle x_1, \dots, x_n \rangle \equiv \langle y_1, \dots, y_m \rangle$$

$$\text{if } x_i = \langle x_{i1}, \dots, x_{im} \rangle$$

and

$$y_i = \langle x_{1i}, \dots, x_{ni} \rangle$$

Inner Product

$$\text{IP}: \langle x, y \rangle \equiv \sum_{i=1}^n x_i y_i$$

Functional Forms

Composition $f_1 \circ f_2 \circ \dots \circ f_n$
 $f_1 \circ f_2 \circ \dots \circ f_n : x \equiv f_1 : (f_2 : (\dots (f_n : x) \dots))$

N Functions $[f_1, \dots, f_n]$
 $[f_1, \dots, f_n] : x \equiv \langle f_1 : x, \dots, f_n : x \rangle$

Apply to all αf
 $\alpha f : x \equiv$
 $x = \phi \rightarrow \phi$
 $x = \langle x_1, \dots, x_n \rangle \rightarrow \langle f : x_1, \dots, f : x_n \rangle$

Conditional $(p \rightarrow f; g)$
 $(p \rightarrow f; g) : x \equiv$
 $p : x = T \rightarrow f : x;$
 $p : x = F \rightarrow g : x;$
 $\perp$

while $(\text{while } p \text{ } f)$
 $(\text{while } p \text{ } f) : x \equiv$
 $p : x = T \rightarrow (\text{while } p \text{ } f) : f : x;$
 $p : x = F \rightarrow x;$
 $\perp$

Constant $(\text{constant } x) = \overline{x}$
 $(\text{constant } x) : \gamma \equiv x$

Defining new functions

def $f = \Gamma_n(f_1, \dots, f_n)$ where $\Gamma_n \in \mathcal{T}$

then $f: x$ is replaced by

$$\Gamma_n(f_1, \dots, f_n) \circ x$$

eg

def $\text{last} = (\text{null} \cdot \text{tl} \rightarrow 1; \text{last} \cdot \text{tl})$

$$\begin{aligned} \text{last} &: \langle a, b \rangle \\ &= (\text{null} \cdot \text{tl} \rightarrow 1; \text{last} \cdot \text{tl}) : \langle a, b \rangle \\ &= \text{last} \cdot \text{tl} : \langle a, b \rangle \quad \{\text{since } \text{null} \cdot \text{tl} : \langle a, b \rangle = F\} \\ &= \text{last} : \langle b \rangle \\ &= (\text{null} \cdot \text{tl} \rightarrow 1; \text{last} \cdot \text{tl}) : \langle b \rangle \\ &= 1 : \langle b \rangle \quad \{\text{since } \text{null} \cdot \text{tl} : \langle b \rangle = T\} \\ &= b \end{aligned}$$

(LABEL LAST ($\lambda(L) \mathcal{E}_{\text{LAST}}$))

$$\lambda(x) [(\lambda(\text{LAST})(\text{LAST } x))^\circ (\lambda(L) \mathcal{E}_{\text{LAST}})]$$

eg

$$\underline{\text{def}} \text{ MATMPY} = \lambda \lambda \text{ IP} . \lambda \text{ distl} . \text{distr} . [1, \text{trans} . 2]$$

$$\underline{\text{let}} \ a_{m \times n} * b_{n \times p}$$

$$a = \langle a_1, \dots, a_m \rangle \text{ where } a_i = \langle a_{i1}, \dots, a_{in} \rangle$$

$$b = \langle b_1, \dots, b_p \rangle \text{ where } b_i = \langle b_{i1}, \dots, b_{ni} \rangle$$

$$\begin{aligned} & \text{MATMPY} : \langle a, b \rangle \\ &= \lambda \lambda \text{ IP} . \lambda \text{ distl} . \text{distr} . [1, \text{trans} . 2] : \langle a, b \rangle \\ &= \lambda \lambda \text{ IP} . \lambda \text{ distl} . \text{distr} : \langle a, b \rangle \\ &= \lambda \lambda \text{ IP} . \lambda \text{ distl} : \langle \langle a_1, b \rangle, \dots, \langle a_m, b \rangle \rangle \\ &= \lambda \lambda \text{ IP} : \langle \text{distl} : \langle a_1, b \rangle, \dots, \text{distl} : \langle a_m, b \rangle \rangle \end{aligned}$$

$$\underline{\text{let}} \ \overline{\text{row}}_i = \langle \langle a_i, b_1 \rangle, \dots, \langle a_i, b_p \rangle \rangle$$

$$\begin{aligned} &= \lambda \lambda \text{ IP} : \langle \overline{\text{row}}_1, \dots, \overline{\text{row}}_m \rangle \\ &= \langle \lambda \text{ IP} : \overline{\text{row}}_1, \dots, \lambda \text{ IP} : \overline{\text{row}}_m \rangle \\ &= \text{the result} \end{aligned}$$

since

$$\lambda \text{ IP} : \overline{\text{row}}_i = \langle \text{IP} : \langle a_i, b_1 \rangle, \dots, \text{IP} : \langle a_i, b_p \rangle \rangle$$

$$= \text{the } i^{\text{th}} \text{ row of the product matrix.}$$

⑥

Other possible primitives and functional forms:

1HALF

$$1HALF : x \equiv x = \phi \rightarrow \phi ;$$

$$x = \langle y_1, \dots, y_n \rangle \rightarrow \langle y_1, \dots, y_{\lceil \frac{n}{2} \rceil} \rangle ; \perp$$

2HALF

$$2HALF : x \equiv x = \phi \rightarrow \phi ;$$

$$x = \langle y_1, \dots, y_n \rangle \rightarrow \langle y_{\lceil \frac{n}{2} \rceil + 1}, \dots, y_n \rangle ; \perp$$

TREE

$$(TREE F) : x \equiv x = \phi \rightarrow \phi ;$$

$$x = \langle y_1 \rangle \rightarrow \langle y_1 \rangle ;$$

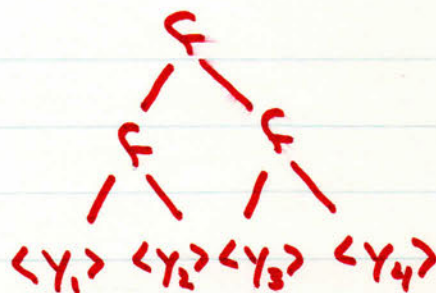
$$x = \langle y_1, \dots, y_n \rangle \quad \{n \geq 2\} \rightarrow$$

$$F \circ [(TREE F) \circ 1HALF, (TREE F) \circ 2HALF] ; x$$

$\perp$

so that:

$$(TREE F) : \langle y_1, y_2, y_3, y_4 \rangle \rightarrow$$



then if MERGE is a primitive which will merge two sorted sequences,

we can SORT the sequence $X = \langle y_1, \dots, y_n \rangle$ by evaluating:

(TREE MERGE) : X

This sort will be linear in the length of X if we fully exploit the possible parallelism since the time spent at each level is:

_____	n
_____	$\frac{1}{2}n$
_____	$\frac{1}{4}n$
⋮	⋮
-	<u>1</u>
	$< 2n$

Formal Specification of Red-1

Syntax

Constants C

$$1) \{ \emptyset, T, F \} \in \text{Atoms}$$

$$2) \text{Atoms} \subset C ; \perp \in C$$

$$3) x_1, \dots, x_n \in C \Rightarrow \langle x_1, \dots, x_n \rangle \in C$$

$$\text{but } \langle \dots, \perp, \dots \rangle = \perp$$

Expressions E

$$1) C \subset E$$

$$2) x_1, \dots, x_n \in E \Rightarrow \langle x_1, \dots, x_n \rangle \in E$$

$$3) x, y \in E \Rightarrow (x : y) \in E$$

Semantics

$$\mu : E \rightarrow C$$

Definition of μ

$$\mu x \equiv x = \perp \rightarrow \perp$$

$$x \in A \rightarrow x$$

$$x = \langle x_1, \dots, x_n \rangle \rightarrow \langle \mu x_1, \dots, \mu x_n \rangle$$

$$x = (\gamma : z) \rightarrow$$

$$\gamma = \perp \rightarrow \perp$$

$$\gamma \in A \rightarrow \mu [\rho \gamma] \cdot \mu z$$

$$\gamma = \langle \gamma_1, \dots, \gamma_n \rangle \rightarrow$$

$$\mu (\gamma_1 : \langle \gamma, z \rangle)$$

$$T \rightarrow \mu ((\mu \gamma) : z)$$

where ρ is the representation

function which maps constants to the functions they represent.

$$\rho : C \rightarrow (C \rightarrow E)$$

- $\rho \text{TAIL} = \text{tl}$
- $$\begin{aligned} & \mu(\text{APPLY} : \langle \text{TAIL}, \langle A, B \rangle \rangle) \\ &= \mu[\rho \text{APPLY}] \cdot \mu \langle \text{TAIL}, \langle A, B \rangle \rangle \\ &= \mu \text{ apply} \cdot \langle \text{TAIL}, \langle A, B \rangle \rangle \\ &= \mu(\text{TAIL} : \langle A, B \rangle) \\ &= \mu[\rho \text{TAIL}] \cdot \mu \langle A, B \rangle \\ &= \mu \text{ tl} \cdot \langle A, B \rangle \\ &= \mu \langle B \rangle = \langle \mu B \rangle = \langle B \rangle \end{aligned}$$
- "unused" atoms represent $\perp : x = \perp$
- innermost applications evaluated first

$$\mu(1 : (1 : \langle \langle A, B \rangle, C \rangle)) = A$$

- Metacomposition Rule

$$[\rho \langle x_1, \dots, x_n \rangle] : \gamma \equiv [\rho x_1] : \langle \langle x_1, \dots, x_n \rangle, \gamma \rangle$$

a) sequences also represent functions thus realizing functional forms.

- if $\rho \text{CONST} = 2.1$
then $\langle \text{CONST}, x \rangle$ represents the functional form, $(\text{CONST } x)$

$$\begin{aligned} [\rho \langle \text{CONST } x \rangle] : \gamma &= [\rho \text{CONST}] : \langle \langle \text{CONST}, x \rangle, \gamma \rangle \\ &= 2.1 : \langle \langle \text{CONST}, x \rangle, \gamma \rangle \\ &= 2 : \langle \text{CONST}, x \rangle = x \end{aligned}$$

- similarly, $\rho NF = \alpha \text{ apply} \cdot \text{tl} \cdot \text{distr}$

$$\begin{aligned}
 & [\rho \langle NF, f_1, \dots, f_n \rangle] : x \\
 &= [\rho NF] : \langle \langle NF, f_1, \dots, f_n \rangle x \rangle \\
 &= \alpha \text{ apply} \cdot \text{tl} \cdot \text{distr} : \langle \langle NF, f_1, \dots, f_n \rangle, x \rangle \\
 &= \alpha \text{ apply} \cdot \text{tl} : \langle \langle NF, x \rangle, \langle f_1, x \rangle, \dots, \langle f_n, x \rangle \rangle \\
 &= \alpha \text{ apply} : \langle \langle f_1, x \rangle, \dots, \langle f_n, x \rangle \rangle \\
 &= \langle \text{apply} : \langle f_1, x \rangle, \dots, \text{apply} : \langle f_n, x \rangle \rangle \\
 &= \langle (f_1 : x), \dots, (f_n : x) \rangle
 \end{aligned}$$

- likewise for other functional forms eg
 $\langle CN, p, f, g \rangle$ represents $(p \rightarrow f; g)$
 $\langle o, f_1, \dots, f_n \rangle$ " $f_1 \circ f_2 \circ \dots \circ f_n$
etc

- if a definition capability is added whereby a list of the form:

$$\begin{aligned}
 a_1 &= e_1 \\
 a_2 &= e_2 \\
 &\vdots
 \end{aligned}
 \qquad
 \begin{aligned}
 a_i &\in \text{Atoms} \\
 e_i &\in E
 \end{aligned}$$

is provided to augment ρ
 then this together with the
 metacomposition rule allows complete
 freedom in defining new functions
 and functional forms.

note: recursion is possible without a definition capability.

let LAST $\equiv \langle CN, p, f, g \rangle$

and $p \equiv \langle 0, NULL, TAIL, 2 \rangle$

$f \equiv 1 \cdot 2$

$g \equiv \langle 0, APPLY, \langle NF, 1, \langle 0, TAIL, 2 \rangle \rangle \rangle$

$$\begin{aligned} & \mu(\langle LAST : \langle a_1, a_2 \rangle \rangle) \\ &= \mu(LAST : \langle \langle LAST \rangle, \langle a_1, a_2 \rangle \rangle) \\ &= \mu(\langle CN, p, f, g \rangle : \langle \langle LAST \rangle, \langle a_1, a_2 \rangle \rangle) \end{aligned}$$

{ since $\mu(p : \langle \langle LAST \rangle, \langle a_1, a_2 \rangle \rangle)$
is F }

$$= \mu(\langle 0, APPLY, \langle NF, 1, \langle 0, TAIL, 2 \rangle \rangle \rangle : \langle \langle LAST \rangle, \langle a_1, a_2 \rangle \rangle)$$

$$= \mu(APPLY \cdot [1, TAIL \cdot 2] : \langle \langle LAST \rangle, \langle a_1, a_2 \rangle \rangle)$$

$$= \mu(APPLY : \langle \langle LAST \rangle, \langle a_1 \rangle \rangle)$$

$$= \mu(\langle LAST : \langle a_2 \rangle \rangle)$$

etc.

Algebraic properties of red language programs.

axioms, eg $h \circ (p \rightarrow f; g) \equiv p \rightarrow h \circ f; h \circ g$

$$\alpha(f \circ g) \equiv \alpha f \circ \alpha g$$

theorems, eg

"Iteration Theorem"

If $f \equiv p \rightarrow g; h \circ f \circ k$,
then $f \equiv p \rightarrow g; p \circ k \rightarrow h \circ g \circ k; \dots; p \circ k^n \rightarrow h \circ g \circ k^n$

We can use this together with other algebraic laws to show that $f \equiv g$ where:

$$f \equiv p \circ 1 \rightarrow 2; h \circ f \circ [k \circ 1, 2]$$

$$g \equiv p \circ 1 \rightarrow 2; g \circ [k \circ 1, h \circ 2]$$

Further work will involve proving expansion theorems for functions with more complex recursive structure, eg

$$f = \dots h \circ [\dots f \dots, \dots f \dots]$$

$$f = \dots f \circ [\dots f, \dots]$$

and so forth.

11.1

The 'fuse' function.

A recursive program in a possible red language

$$f \equiv E(f) = (\text{ego} \rightarrow \bar{0}; \text{eq}1 \rightarrow \bar{1}; \text{peven} \rightarrow f \cdot h; \\ + \cdot [f \cdot h \cdot s, f \cdot h \cdot a])$$

where h , s , and a are primitives that respectively halve, subtract 1 from, and add 1 to an integer argument.

Note that the running time is $\sim \log N$ because of the concurrent evaluation of $(f \cdot h \cdot s : x)$ and $(f \cdot h \cdot a : x)$.

A transliteration of Dijkstra's repetitive solution might be expressed as:

$$D = 3 \cdot (D0, \text{peven} \text{ ol} \rightarrow [h \text{ ol}, + \cdot [2, 3], 3], \\ \text{podd} \text{ ol} \rightarrow [h \text{ os} \text{ ol}, 2, + \cdot [2, 3]]) \cdot [id, \bar{1}, \bar{0}]$$

Again, this is a $\log N$ solution.

Exercise (!) Write a reduction language program for fuse with a running time of $\log \log N$. Hint (due to EWD): show that $F(N) = F(n) \cdot F(m+1) + F(n+1) \cdot F(m)$ where the binary representation of N has been arbitrarily split into 2 pieces, n being the number represented by the bits of the first piece, and m being the number represented by the mirror image of the bits of the second piece.

Further work

- definition facility
- "complete" language processor
I/O
history sensitive
a "naming" technique
- basis for semantic specification
of more complex languages

Paul McJones 1975

Gerald Jones 1977

- a practical, programming language?
inherent parallelism

Church-Rosser design

new LSI motivated architecture

Mego 1977