Univ of Texas "Year of Programming"
Aug 1987 Inst. on Declarative Programming

Higher Order Functions and I/O in a Strict Functional Language

John Hughes, John Williams, Ed Wimmers
John Backus

Outline

1. FL's approach to I/O & permanent files
2. Defining higher order functions in a strict language

- Changes in FL since Preliminary Manual (IBM Res. Rep. RJ 5339)
- Basic issues, not FL syntax

(see last slide for definitions of combining forms)

○

Design goals of FL language

1. Broad scope Interactive, systems, data-base, number-crunching

2. Cost-effective programming
   - powerful combining forms ⇒ structure
   - off-the-shelf, comprehensible general programs

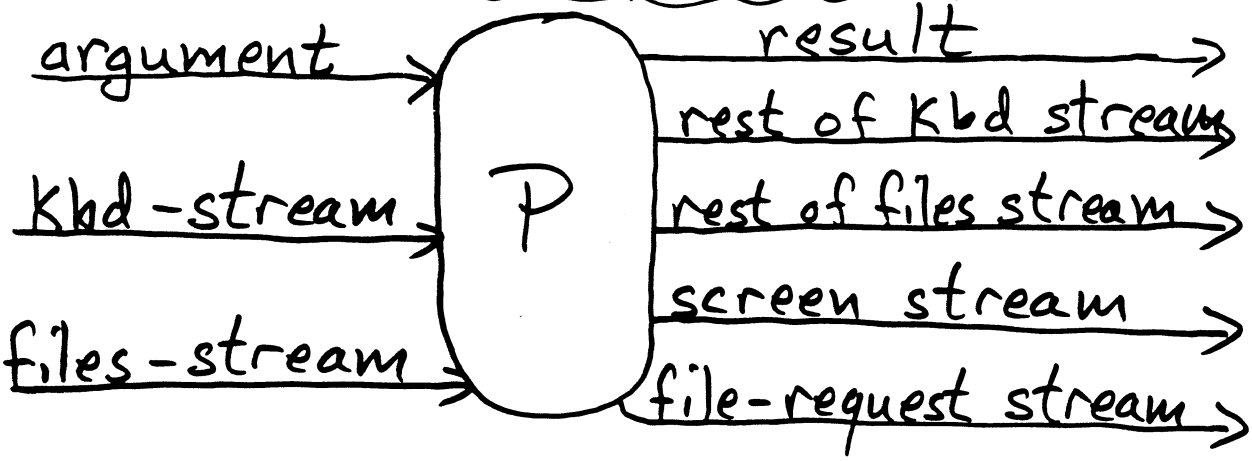
3. Fast-running programs (~ Fortran)
   - algebraic program transformation

I/O and permanent files in functional languages

Issues considered in desiging FL:

- Programming Interactive, File-Referencing (IFR) programs
  - Lazy-stream approach
    FP84 (Halpern, Williams, Wimmers)
  - Strict-history approach
    FL (IBM Res. Rep. RJ5339)
- Semantics of lazy vs strict languages
  - for "pure" functions (no I/O)
    strict → simpler semantic domain & operational semantics
  - for IFR programs
    strict → simpler ordering of I/O events

Functionality of Interactive, File-Referencing (IFR) programs in lazy-stream approach



where

Kbd-stream = all keyboard data

files-stream = all responses to read-file requests from this and all subsequent programs

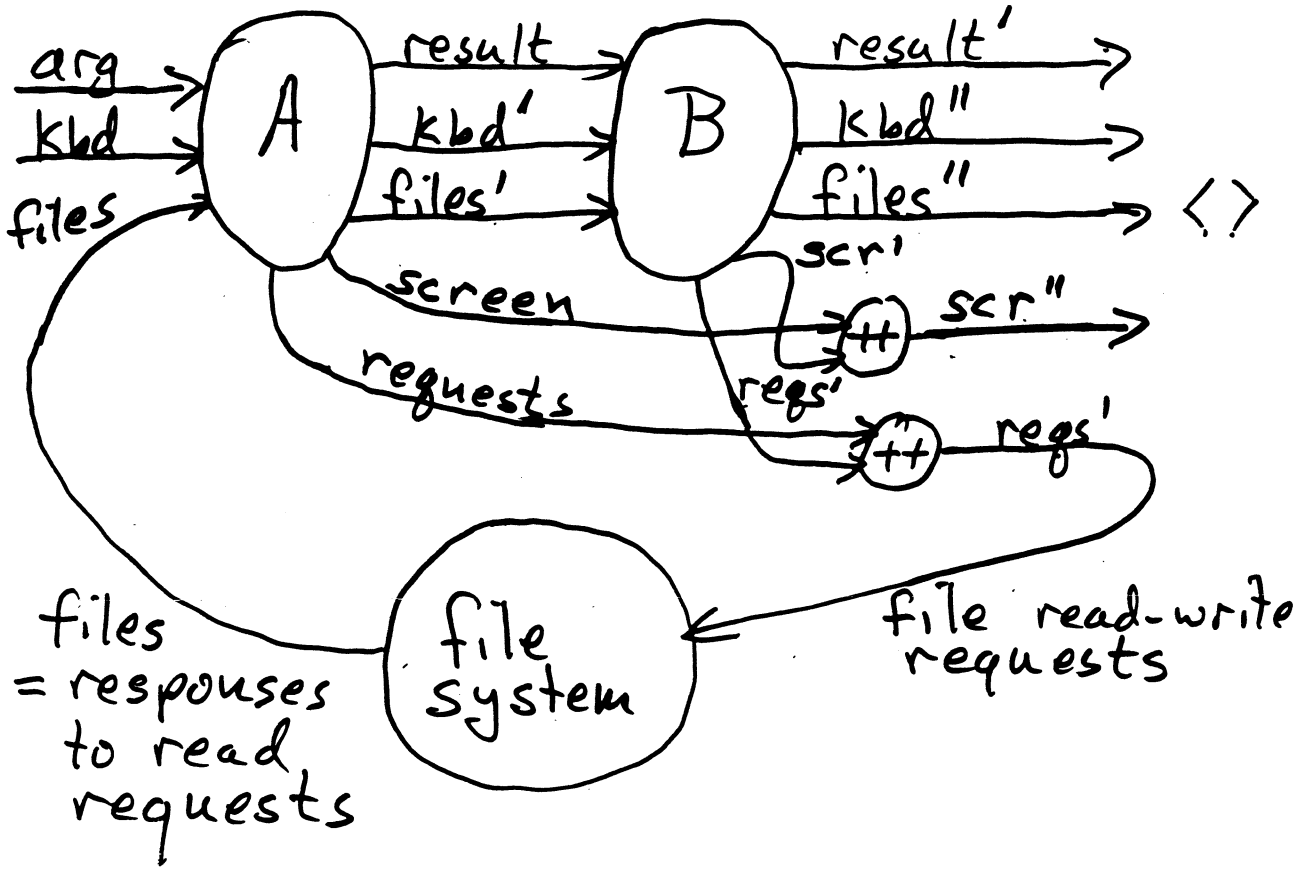
rest of Kbd = data not consumed by P

rest of files = files requested by progs following P

screen = data for display

file-request = read or write requests to the file system

Composition of IFR programs A,B connected to file system



Timing, ordering of events

Style: program from first principles, or use combining forms for IFR composition, conditon, construction, ...

Problems we found with lazy-stream approach

- Lazy-stream semantics difficult to understand when stream elements correspond to "events" - sequencing - debugging, unexpected ordering of events
- IFR programming from first principles is complex & difficult.

- Orderly IFR programming requires:
  - New set of IFR combining forms, in addition to regular ones

  - If any subprogram is IFR, all should be; convert "pure" subprograms to IFR by combining form; then build program only with IFR combining forms.

The FL strict-history approach to IFR programming

All FL programs P :

Formally: $P:(Value, History) = (V', H')$

Informally: $P: V = V'$

side effect: $H \longrightarrow H'$

Histories = finite or infinite chronological records of I/O & FR events

Examples

$tl:(\langle 1,2,3\rangle, h) = (\langle 2,3\rangle, h)$

$tl:\langle 1,2,3\rangle = \langle 2,3\rangle$ no side effect $h \to h$

$in:(dev\text{-}nm, h) = (input, h')$

$in: dev\text{-}nm = input$ $h \to h'$

side effect: next $in:dev\text{-}nm = {}^{next}_{input}$

$P:(x, h) = (\bot, h')$

$P: x = \bot$ P non-terminating, does I/O.

$\Rightarrow h'$ contains infinite record of I/O activity

FL strict-history approach, cont'd

Four primitives that change history:

in, out (I/O) get, put (files)

(get doesn't change h)

Evaluation of expressions =

leftmost-innermost

Example:

$([in, in]:dn, h_1) \rightarrow\rightarrow (\langle in:dn, in:dn\rangle, h_1)$

$\rightarrow\rightarrow (\langle input_1, in:dn\rangle, h_2)$

$\rightarrow\rightarrow (\langle input_1, input_2\rangle, h_3)$

Example:

Let prompt = in ∘ ~'kb' ∘ out ∘ [~'scr', id]

Then prompt:'input?'

$\rightarrow\rightarrow$ in:(~'kb':(out:⟨'scr', 'input?'⟩))

$\rightarrow\rightarrow$ in:(~'kb':⟨'scr', 'input?'⟩)

and 'input?' appears on scr

$\rightarrow\rightarrow$ in:'kb'

$\rightarrow\rightarrow$ input from keyboard in response to prompt 'input?'

Consequences of strict-history approach

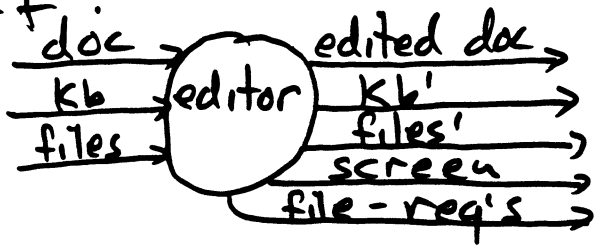
+1. Standard combining forms apply to all programs, "pure" & IFR

+2. FP laws apply for "pure" programs

+3. The same laws (weaker than FP) apply in strict-history & lazy-stream IFR approaches! Williams & Wimmers

+4. Informal strict-history functionality is simpler than for IFR lazy streams:

Instead of:



have:

interaction + files = side effects

+5. Simple ordering of I/O events

Consequences of strict-history approach, cont'd

- 6. A strict-history program tends to "wire in" the I/O devices it uses, whereas a lazy-stream program must have its I/O streams as explicit arguments, making it easier to replace a "device-stream" by a program-generated stream.
  - The strict-history approach trades generality for convenience

Defining higher order functions in a strict function level language

Issues:

- Making clear, readable definitions of higher order functions without proper variables or λ-abstraction.
- Defining the meaning of functional equations to yield proper fixpoints.

  Why the solution of

  $$g = \text{isnull} \rightarrow [\,]; \text{a!} \circ [f \circ 1, g \circ 1]$$

  is $g = \text{insert}(f)$, not $g = \perp$.

Patterns in function level definitions

$$f = [\![x., y.]\!] \rightarrow g \circ [y, x]$$

same as object level:

$$f:\langle x, y\rangle = g:\langle y, x\rangle$$

Predicate combining forms

$$[\![p_1, \ldots, p_n]\!]:x = \text{True} \text{ iff } x=\langle x_1,\ldots,x_n\rangle \;\&\; p_i:x_i = \text{true}$$

$$(p \mapsto q):x = \text{True} \text{ iff } p\circ s1 : x = \text{true} \;\&\; q\circ t1 : x = \text{true}$$

Patterns

$X.P \xrightarrow{pred} P$

$X.P \xrightarrow{def} X = id$

$X. \simeq X.T$

$[\![[\![x., y.]\!], z.]\!] \xrightarrow{P} [\![[\![T, T]\!], T]\!]$

$\xrightarrow{d} x = s1 \circ s1$
$y = s2 \circ s1$
$z = s2$

$y:\langle\langle 1,2\rangle, 3\rangle = 2$

$[\![x., y.]\!] \mapsto z.seqof:isint \xrightarrow{P} [\![T, T]\!] \mapsto seqof:isint$

$\xrightarrow{d} x = s1 \circ s1$
$y = s2 \circ s1$
$z = t1$

Patterns in function level definitions, cont'd

If P is a pattern, then

$$f = P \rightarrow E$$

is equivalent to

$$f = Pred(P) \rightarrow (E \text{ where' } Defs(P)); error$$

Function level definitions; Lifting

Object level: For all objects x, y:

$$f:\langle x, y\rangle = GCD:\langle x, add:\langle y, 1\rangle\rangle$$

Lifted function level: For all functions x, y:

$$f \circ [x, y] = GCD \circ [x, add \circ [y, \sim 1]]$$

Using patterns:

$$f = [\![x., y.]\!] \rightarrow GCD \circ [x, add \circ [y, \sim 1]]$$

Equivalently, pattern eliminated:

$$f = [\![T, T]\!] \rightarrow GCD \circ [s1, add \circ [s2, \sim 1]]$$

Where

$\sim 1 : x = 1$ all $x \neq \perp$

$T : x = true$ "

Defining higher order functions in a strict language

To define K

$$K:x:y = x \quad \text{if } y \neq \bot$$
$$= \bot \quad \text{o.w.}$$

Simple lifting rule doesn't apply

Instead, define K^* = uncurried K

for all values x, y (object level)

$$K^*:\langle x, y\rangle = x \quad \text{if } \langle x, y\rangle \neq \bot$$

Or, with patterns & functions x, y:

$$K^* = [\![x., y.]\!] \to x$$

Then

$$K = curry:([\![x., y.]\!] \to x)$$
$$= curry:S1$$

curried fn's always get pairs

where $curry:f:x:y = f:\langle x, y\rangle$

Defining higher order functions, cont'd

General strategy (for many functions)

$G:f:x = curry:G^*:f:x = G^*:\langle f, x\rangle$ (Hughes)

G:f can't rewrite until it has next arg.

Example

Define cons such that

$$cons:\langle f_1,\ldots,f_n\rangle:x = \langle f_1:x,\ldots,f_n:x\rangle$$

Define

$$cons = curry:(\alpha:apply \circ distr)$$

Then

$$\begin{aligned} cons:\langle f,g\rangle:x &= cons^*:\langle\langle f,g\rangle, x\rangle \\ &= \alpha:apply \circ distr:\langle\langle f,g\rangle, x\rangle \\ &= \alpha:apply:\langle\langle f,x\rangle,\langle g,x\rangle\rangle \\ &= \langle apply:\langle f,x\rangle, apply:\langle g,x\rangle\rangle \\ &= \langle f:x,\ g:x\rangle \end{aligned}$$

Recursive definitions & fixpoints in a strict language

Let

$$g = (x. \mapsto rest.) \longrightarrow$$
$$\quad isnull \circ rest \longrightarrow x;\ f \circ [x, g \circ rest]$$

then equation has fixpoints:

strict: $g = \bot$

Lazy: $g = /:f = insert(f)$

strict, $\geq nt$: $g = /:f$

where $nt:x = \bot$, all x

$g =_1 E(g) =_1$ "equal functions"

Thus $\bot =_1 nt$

Unexpected fixpoints exist in lazy languages

(Hughes)

Let

TC = transitive closure

Let

(1) $f = elim \circ (id +\!\!+ f \circ close)$

where

elim eliminates duplicate pairs

$+\!\!+$ is concatenate &

$\langle x_1, \ldots, x_n \rangle +\!\!+ \perp = \langle x_1, \ldots, x_n, * \rangle$

$*$ "zero or more other elements"

Least fixpoint of (1) is TC^*, where

$TC^* : \langle\langle a, b \rangle\rangle = \langle\langle a, b \rangle, * \rangle$

The expected fixpoint, TC, is larger:

TC is a fixpoint of (1)

$TC^* < TC$

Abstraction, preliminaries

Infix rule $(f\ r\ g) = r:\langle f, g\rangle$

$f \circ g = \circ:\langle f, g\rangle$

Prime $f':\langle x, y\rangle = f \circ [x, y]$

$$(f\ r'\ g):x = r':\langle f, g\rangle : x$$
$$= r \circ [f, g] : x \quad \text{(Infix rule)}$$
$$= r:\langle f:x,\ g:x\rangle$$
$$= (f:x\ \ r\ \ g:x)$$

Similarly

$$['f, g]:x = [f:x,\ g:x]$$

$$(p \to' f; g):x = p:x \to f:x; g:x$$

and

$$(f \circ' g):x = (f:x) \circ (g:x)$$

Abstraction, examples

(1) $Ins:f =_1 isnull \circ tl \to s1 ; f \circ [s1, Ins:f \circ tl]$

$Ins = {}^{\smile}(isnull \circ tl) \to' {}^{\smile}s1 ;$
$id \circ' [{}'^{\smile}s1, Ins \circ' {}^{\smile}tl]$

check that $Ins:f$ gives r.h.s. of (1)
$Ins > k:nt$, $=_2$ = 'equal functionals'

(2) $curry:f:x:y = f:\langle x, y\rangle$

$curry:f:x = f \circ [k:x, id]$

$curry:f = k:f \circ' [{}'k, k:id]$

$curry = K \circ'' [{}''K:K, K^2:id]$

director strings

(3) $lift:f:x = f \circ cons:x$
$= o:\langle f, cons:x\rangle$

$lift = curry:([\![f., x.]\!] \to o \circ [f, cons \circ x])$
$= curry:(o \circ [s1, cons \circ s2])$
$(o \circ [s1, cons \circ s2]):\langle f, x\rangle$
$= o:\langle f, cons:x\rangle$

FL combining forms (Reference)

Composition $f \circ g : x = \circ : \langle f, g \rangle : x = f : (g : x)$

Construction $[f_1, \ldots, f_n] : x = \langle f_1 : x, \ldots, f_n : x \rangle$

where $[f_1, \ldots, f_n] = cons : \langle f_1, \ldots, f_n \rangle$

Condition

$$(p \rightarrow f; g) : x = \begin{cases} f : x & \text{if } p : x \neq F, \perp \\ g : x & \text{if } p : x = F \\ \perp & \text{o.w.} \end{cases}$$

$(p \rightarrow f) = (p \rightarrow f; error)$

Apply-to-all $\alpha : f : \langle x_1, \ldots, x_n \rangle = \langle f : x_1, \ldots, f : x_n \rangle$

Constant $\overline{\ }x : y = K : x : y = x \quad (y \neq \perp)$

Curry$_n$ $curry_n : f : x_1 : \cdots : x_n = f : \langle x_1, \ldots, x_n \rangle$

$curry_n = K^{n-1} \delta^{(n)} [{}^{(n)}K : K^{n-1}, K^2 : K^{n-2}, \ldots, K^n : id]$

$curry = curry_2$

Lift $lift : f : x = f \circ cons : x$

Prime $f' = lift : f \qquad \delta^{(n)} = lift^n : \circ$

$[{}^{(n)}f_1 \ \ f_n] = lift^n : cons : \langle f_1, \ldots, f_n \rangle$

E-construction $[\![p_1, \ldots, p_n]\!] : x = (len : x = n \ \& \ p_i : x_i = true)$

↦ $(p \mapsto q) : x = p \circ s1 : x \ \ \& \ q \circ t1 : x$