

20 Mar 74

user's state = (expr, defs, universe, modes, input, output)

<FETCH STATE done case>
<STORE STATE state>

Names

"programming generality" } Jack Dennis
"modularity"

"On the meaning of ..." - Sandy Fraser

(also, a recent SIGPLAN article?)

"Modularity" - Jay Euzley

Parnas

"Global Var Considered Harmful" Wulf & Shaw (SIGPLAN)

"Variables, Binding & Protection"

George & Sager, SIGPLAN, Dec 73

Dijkstra ...?

a z

A Z

o g

([<{?}>])

+ - x ÷

* /

· : ;

=

88 characters = 2+26+10+26

Leavenworth: "Transition Funs"
RC3459

Elgot, Eilenberg

$e = e_1 \rightarrow e_2 \rightarrow \dots \rightarrow e_n = v$

integers
strings
sequences

procedure (p₁, p₂, ..., p_n);
 local l₁, l₂, ..., l_m;
 <body>
end procedure

beg

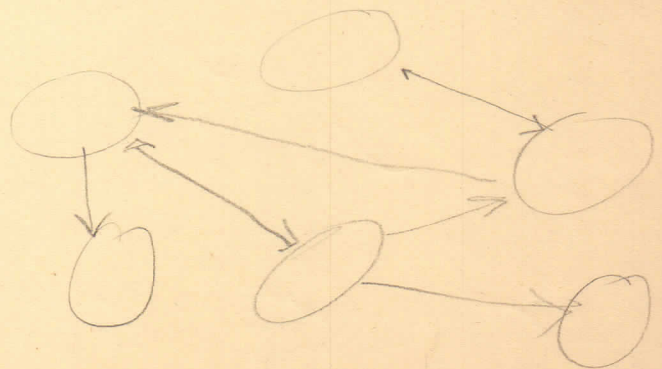
p₁;

p₂;

p₃;

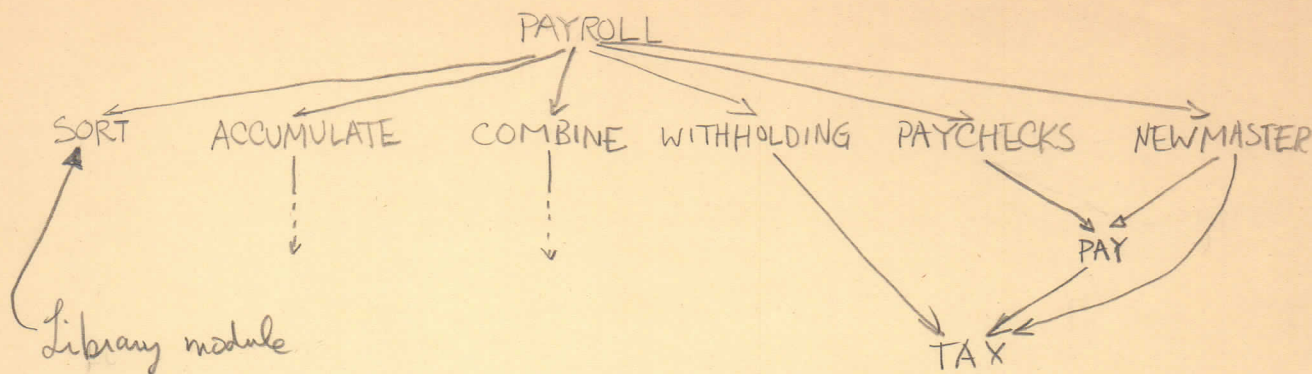
local v₁, v₂, v₃;

procedures & blocks → hierarchical, single flow of control.
processes → non hierarchical, multiple flow of control



25 Mar 74

Dependency Relation for Payroll Example



Algol version:

proc PAYROLL

begin proc SORT

proc ACCUMULATE

proc NEWMASTER

proc PAY

proc TAX

end

PAYROLL, {
 PAYROLL. {
 COMBINE
 WITHHOLDING
 PAYCHECKS
 NEWMASTER

This has several disadvantages:

1. As is well known, there is no way to incorporate a packaged SORT from library.
 2. SORT, ACCUMULATE, ..., TAX are all accessible to each other, although only the accessibilities graphed above are necessary, hence desirable.
- If a variable named PAY is accidentally undefined in TAX, it will reference the procedure PAY.

Rules:

1. Never evaluate any expression until you must ["call by name"]
2. Never evaluate an expression more than once.

Morris's example for deferring evaluation:

Testing for equality of frontiers of (possibly nonisomorphic) trees.

1. Can "flatten" both trees & check equality
2. Can have two coroutines each traversing a tree, returning leaves.
3. Try to implement 1 with efficiency of 2 (delay evaluation).

Properties a naming system should have:

1. Must always be possible to override global system definitions (e.g., for example, additions to system don't clobber existing programs).
2. If a global system definition has been overridden, it should still be possible to get at the overridden def.

Def $fl(t, l) = \text{atom } t \rightarrow \text{cons}(t, l)$
 $\quad \quad \quad | fl(\text{car } t, fl(\text{cdr } t, l))$

and $\text{equal}(t, u) = \text{atom } t \wedge \text{atom } u \rightarrow t = u$
 $\quad \quad \quad | \neg \text{atom } t \wedge \neg \text{atom } u \rightarrow \text{equal}(\text{car } t, \text{car } u) \wedge \text{equal}(\text{cdr } t, \text{cdr } u)$
 $\quad \quad \quad | \text{false}$

Now $\text{equal}(fl('A(BC))', NIL), fl('((A B) C)', NIL)$? [Answer: No!] [Use dot notation]

Try passing arguments "by name" (straight substitution), except primitives like, car, cdr, cons, atom, eq then evaluate them enough that they can give a result.

E.g. $\text{atom } fl[(A(BC)), NIL] = \dots = \text{atom } (A \cdot fl[(BC), NIL])$
 $\quad \quad \quad = \underline{\text{false}}$

the new $\text{car } (E \cdot I) = E$

$\text{cons}[E, I] = (E \cdot I)$ [no evaluation necessary here]

Definition (nonselfreferencing) of factorial function, say Z , in RED:

(($\emptyset$ (CN (ZEROP 2)
 (C 1)
 (TIMES [2 (APPLY (AR PRED))]))))

Notes:

a) $\langle \text{ZEROP } i \rangle \equiv \begin{cases} T & \text{if } i=0 \\ F & \text{otherwise} \end{cases}$

b) $\langle \text{PRED } i \rangle \equiv i-1$

c) $\langle \text{TIMES } (i \ j) \rangle \equiv i * j$

d) An example of the evaluation for the argument 2 is given. (In the following, every occurrence of ' Z ' should conceptually be replaced by the above definition before the example is read.)

$$\begin{aligned} \langle Z, 2 \rangle &\equiv \langle (\text{CN } (\text{ZEROP } 2) (\text{C } 1) (\text{TIMES } \dots)), (Z \ 2) \rangle \\ &\equiv \langle (\text{TIMES}, [2 (\text{APPLY } (\text{AR } \text{PRED}))]), (Z \ 2) \rangle \\ &\equiv \langle \text{TIMES}, (\langle 2, (Z \ 2) \rangle \langle \text{APPLY}, \langle (\text{AR } \text{PRED}), (Z \ 2) \rangle \rangle) \rangle \\ &\equiv \langle \text{TIMES}, (2 \langle \text{APPLY}, (Z \ \langle \text{PRED}, 2 \rangle) \rangle) \rangle \\ &\equiv \langle \text{TIMES}, (2 \langle \text{APPLY}, (Z \ 1) \rangle) \rangle \\ &\vdots \\ &\equiv \langle \text{TIMES}, (2, \langle \text{TIMES}, (1, \langle \text{APPLY}, (Z \ 0) \rangle) \rangle) \rangle \\ &\vdots \\ &\equiv \langle \text{TIMES}, (2 \ \langle \text{TIMES}, (1 \ 1) \rangle) \rangle \\ &\vdots \\ &\equiv 2 \end{aligned}$$

In λ -calc:

$\lambda n. f(f, n)$ where $f = \lambda(g, m). \text{ if } m=0 \text{ then } 1 \text{ else } m \times g(g, m-1)$

or:

$Y[\lambda f. \lambda n. \text{ if } n=0 \text{ then } 1 \text{ else } n \times f(n-1)]$ where $[Y = ZZ \text{ where } Z = \lambda t. \lambda u. u(ttu)]$

3 April 1974

Generalization of factorial example

Suppose we wish to define a function Z such that Z 's effect on an arbitrary object X ~~can be defined~~ is as follows:

$$\langle Z, X \rangle \equiv \begin{cases} \langle H, X \rangle & \text{if } \langle P, X \rangle \equiv T \\ \langle H, (X, \langle Z, \langle f, X \rangle \rangle) \rangle & \text{if } \langle P, X \rangle \equiv F \end{cases}$$

for previously defined functions P , D , H , and f .

Then the following schema gives the form of such an Z :

$$((\phi \text{ (CN } P \\ H \\ (H [Z \text{ (APPLY (AR } f))]))))$$

Debugging Want to handle traps (i.e. function breakpoints), interrupts (panic-attr)

Two approaches: "top down": entire program (with real applications mapped to pseudo-applications) is passed to handler function,

"bottom up":

Definitions: 1-level global naming

$\langle (\text{DEF } a) x \rangle \equiv x$, and puts $(a x)$ at head of DEF

$\langle (\text{PURGE } a) x \rangle \equiv x$, removes all but first cell named a

$\langle (\text{OFF } a) x \rangle \equiv x$, removes first cell named

$\langle (\text{FETCH } a) x \rangle = y$ where $(a y)$ is first cell named a

Global State: (expr, universe, exception program)

I/O : $\langle \text{PRINTLINE}, l \rangle$

$\langle \text{READLINE}, - \rangle$

Jim's suggestions:

loop: if interrupt then goto ...
advance prog
if error then goto ...
goto loop;

Can use "synthesized attributes" [Knuth] to indicate which expressions have non-constant subexpressions.

Naming Criteria:

1. should be able to override standard definition of a name (so for example additions to system don't affect existing programs)
2. Even though a standard definition has been overridden, would still like to get at the underlying overridden def.

Solution to 1&2:

- a) Name of form $\langle \text{prim}, n \rangle$ is permanent, nonoverridable (so satisfies 2)
- b) For all primitives of form $\langle \text{prim}, n \rangle$, a standard (but overridable, satisfying 1) definition with name simply n is made.

Summary of miscellaneous ideas

1. In pure RED, a self-referencing definition [e.g. $\text{SFACT} = (\dots \text{FACT} \dots)$] does not accomplish a recursive function definition. To solve this problem, we can try to write a LABEL meta-operator similar to the Lisp LABEL mechanism.

Thus

$$\langle (\text{LABEL}, \text{fn}, \text{body}), e \rangle \equiv \langle \text{body}', e \rangle$$

where fn ("function name") is an atom

body is a function composed of the standard primitives and fn

e is an operand expression

body' is body, with each occurrence of fn in a function context replaced by $(\text{LABEL } \text{fn } \text{body})$

The substitution process can easily be expressed in RED, except that in general it is impossible to determine whether an atom will (eventually) be used as a function!

For example, in

$((\text{CN}, p, \text{APPLY}, (\text{PRINT}, 1)) [(C, f), \phi])$ p to the

either f will be applied to the argument (if the application of the argument gives T) or f will be printed (as "data"). Of course the above program is functionally equivalent to

$(\text{CN}, p, f, (\text{PRINT}, (C, f)))$

and in this function the first occurrence of f is always a function. Presumably there are more pathological cases...

15 April 74

Dines Bjørner, Meta-IV

Lucas, Jones & Lucas

McCarthy & Painter

} compiler correctness

compare with
RED

{ ULD worked with transition functions on global state
Vienna people now believe in "small states" (factor out stacks, other well known manipulations)

Abstract objects: maps, classes, tuples, trees (heuristically chosen to clarify abstract description)
integers, ...

env: ENVIRONMENT (map identifiers to abstract storage locations)

σ: STORAGE (map abstract storage location to value)

Semantics will now be given by a function taking a statement, an environment, and a storage, and returning an (updated) storage. [State transformer].

In the λ -calculus, Y can be defined as

$$Y = (\lambda t. \lambda u. u(t t u)) (\lambda t. \lambda u. u(t t u))$$

If we try to do this in λ -Red, we have problems:

$$Y = \langle (\lambda t. \lambda u. 'u''t t' u'') (\lambda t. \lambda u. 'u''t t' u'') \rangle$$

But to use Y , i.e., to evaluate something like:

$$\langle \langle Y F \rangle X \rangle \quad [\text{where } F = (\lambda f. \dots)]$$

we must begin by evaluating Y itself, since it is an innermost application:

$$Y = \langle Z Z \rangle \quad [\text{where } Z = (\lambda t. \lambda u. 'u''t t' u'')]$$

$$= (\lambda u. 'u' \langle Z Z \rangle u'')$$

$$= (\lambda u. 'u' (\lambda u. 'u' \langle Z Z \rangle u'') u'')$$

$$\vdots$$

i.e., Y is undefined.

A 2-stack evaluator for Red

19 April 1974

$id(x) : x \Rightarrow x$ is an identifier

$k(x) : x \Rightarrow x$ is an application-free expression

$r(x) : x \Rightarrow psi(x)$ is false

$m(x) : x \Rightarrow psi(x)$ is true

} basic predicates

$app(rator, rand)$

$seq(head, tail)$

$nil()$

} form abstract Red prog

$mkseq()$

$apply()$

} 'markers' used by machine

ATOM: $S, id(x):C \Rightarrow x:S, C$

NIL: $S, nil():C \Rightarrow nil():S, C$

SEQ: $S, seq(head, tail):C \Rightarrow S, head:tail:mkseq():C$

$mkseq():C \Rightarrow seq(x, nil()):S, C$

MKSEQ: $seq(y):e(x):S, mkseq():C \Rightarrow seq(x, y):S, C$

APP: $S, app(rator, rand):C \Rightarrow S, rator:rand:apply():C$

APPLYATOM: $k(y):id(x):S, apply():C \Rightarrow S, rho x: y:C$

RC1: $k(y):seq(r(x), nil()):S, apply():C \Rightarrow y:x:S, C$

RCN: $k(y):seq(r(x), tail):S, apply():C \Rightarrow$

$y:tail:x:S, apply():apply():C$

MC: $k(y):seq(m(x), tail):S, apply():C \Rightarrow$

$seq(x, seq(seq(seq(x, tail), seq(y, nil())), nil())):$
 $S, apply():C$

Can delete now
since regular comp.
is a meta operator

$\rightarrow k(x) = id(x) \vee nil(x) \vee seq(x) \wedge k(head(x)) \wedge k(tail(x))$

[based on Lardin's SECD machine for evaluating λ -expressions]

Red evaluator using ISWIM

1. abstract syntax of Red

a) predicates: $\text{atom?}(x)$; $\text{seq?}(x)$; $\text{app?}(x)$ [Note: $\text{atom?}(\emptyset) = \text{true}$]b) constructors: $\text{mk-seq}(a, b)$; $\text{mk-app}(a, b)$ c) selectors: $\text{head}(s)$, $\text{tail}(s)$; $\text{rator}(a)$, $\text{rand}(a)$ 2. ρ , ψ :

3. rec meaning $x =$

$\text{atom? } x$	$\rightarrow x$
$\text{seq? } x$	$\rightarrow \text{mk-seq}(\text{meaning}(\text{head } x), \text{meaning}(\text{tail } x))$
$\text{app? } x$	$\rightarrow \text{apply}(\text{meaning}(\text{rator } x), \text{meaning}(\text{rand } x))$

and $\leftarrow$ ~~where~~ where rec $\text{apply}(f, a) =$

 $\text{atom? } f \rightarrow \text{meaning}(\rho f a)$ $\psi?(\text{head } f) \rightarrow$ $\text{apply}(\text{head } f, \text{mkseq}(f, \text{mkseq}(a, \emptyset)))$ $\text{tail } f = \emptyset \rightarrow \text{apply}(\text{head } f, a)$ else $\rightarrow \text{apply}(\text{head } f, \text{apply}(\text{tail } f, a))$ delete this since now
all "compositions" are meta $\rightarrow \left\{ \begin{array}{l} \text{tail } f = \emptyset \rightarrow \text{apply}(\text{head } f, a) \\ \text{else} \rightarrow \text{apply}(\text{head } f, \text{apply}(\text{tail } f, a)) \end{array} \right.$

PRIMAPPLY
CONCREV

ATOM: ((ATOM . :A) ::P) (:T) :S :D $\Rightarrow$ (::P) (:A :T) :S :D

FETCH: ((STAR) ::P) (:T) :S :D $\Rightarrow$ (::P) (:S :T) :S :D

SEQ: ((SEQ ::R) ::P) (:T) :S :D $\Rightarrow$ (::R(MKSEQ)) () :S (:P :T :S :D)

ROW: ((ROW ::R) ::P) (:T) :S :D $\Rightarrow$ (::R(MKROW)) () :S (:P :T :S :D)

APPL: ((APPL (:RATOR . :RAND)) ::P) (:T) :S :D $\Rightarrow$ (:RAND (RESTORE) :RATOR (AP)) () :S (:P :T :S :D)

MKSEQ: ((MKSEQ)) (:T) :S (:PP (:TT) :SS :DD) $\Rightarrow$:PP ((SEQ <<CONCREV :T>>) :TT) :SS :DD

MKROW: ((MKROW)) :T :S (:PP (:TT) :SS :DD) $\Rightarrow$:PP ((ROW <<CONCREV :T>>) :TT) :S :DD

RESTORE: ((RESTORE) ::P) :T :S (:PP (:TT) :SS :DD) $\Rightarrow$ (:PP) :T :SS (:PP :TT :SS :DD)

STORE: ((AP)) (** :RAND) :S (:PP (:TT) :SS :DD) $\Rightarrow$:PP ((ROW) :TT) :RAND :DD

DEFD: ((AP)) ((ATOM . :A) :RAND) :S (:PP (:TT) :SS :DD) $\Rightarrow$ ((AP)) (<LOOKUP :A :SS> :RAND) :S (:PP (:TT) :SS :DD)

PRIM: ((AP)) ((SEQ (ATOM . PRIM) :RATOR) :RAND) :S (:PP :TT :SS :DD) $\Rightarrow$ (<PRIMAPPLY :RATOR :RAND> :PP) :TT :SS :DD

MC: ((AP)) ^{ROW?} ((SEQ :RATOR :REST) :RAND) :S (:PP :TT :SS :DD) $\Rightarrow$ ((AP)) (:RATOR (ROW (SEQ (SEQ :RATOR :REST) :RAND) :SS)) :S (:PP :TT :SS :DD)

Maximal Strongly connected regions of an oriented graph.

Let G be a graph on the set of nodes N . Then G is a binary relation, $G = \{(a,b) \in N \times N \mid \text{an edge runs from node } a \text{ to node } b\}$.

Let G^* be the reflexive transitive closure of G [i.e. $G^* = \bigcup_{n=0}^{\infty} G^n$, where $G^0 = I_N = \{(a,a) \mid a \in N\}$ and $G^{n+1} = G^n \circ G$].

Let $E = G^* \cap \tilde{G}^*$, where $\tilde{G}^*$ is the inverse relation of G^* . Then $(a,b) \in E$ iff G contains an oriented path from a to b and back to a [or if $a=b$].

Clearly E is an equivalence relation. Thus it partitions N into a set of equivalence classes of strongly connected nodes.

The following Red expression gives the set of such classes:

$\langle (\text{EQUIV_CLASSES INTERSECTION } [\phi \text{ INVERSE}] \text{ R_T_CLOSURE}), G \rangle$

Using the facts that $(I \cup G)^k = I \cup G \cup \dots \cup G^k$ and $(I \cup G)^{k+1} = (I \cup G)^k \Rightarrow (I \cup G)^k = G^*$, we can define the Red function

$\text{SRTCLOSURE} = (2 \text{ (WHILE RT_STEP) } [\phi \text{ IDENTITY} \\ (\text{WHILE (NOT EQ } [2 \ 3]) \text{ RTCSTEP)} \\ [\phi \text{ IDENTITY (UNION [IDENTITY } \phi])])])$

where $\text{SRTCSTEP} = [1 \ 3 \text{ (UNION [(IDENTITY 1) (COMPOSE [1 2])])}]$

Strongly connected regions, continued

We define the Red function

$$\text{SEQUIV-CLASSES} = ((AA, \text{IMAGE}[1 \ (1 \ 2)]) \text{ DISTL } [\emptyset \ \emptyset])$$

In the above definitions, we assume a Red language geared for sets

have separate
operators:
MAPSET
MAPSEQ (i.e. AA)

$$\langle (AA, f), \{x_1, \dots, x_n\} \rangle = \{ \langle f, x_1 \rangle, \dots, \langle f, x_n \rangle \} \quad [\text{without repetitions}]$$

INTERSECTION } perform the obvious operation on arbitrary pairs of sets.
UNION }

INVERSE $\equiv (AA, [2 \ 1])$ when applied to a relation [set of pairs] produces the inverse relation,

IDENTITY $\equiv (\text{UNION } [(AA, [1 \ 1]) \ (AA, [2 \ 2])])$ when applied to a relation produces the corresponding identity relation.

COMPOSE when applied to a pair of relations, produces their composition.

Translating the above back into [pseudo-]SETL:

```

read graph;
reachability = rtclosure(graph);
connectedness = reachability  $\cap$  reachability-1; /* -1 denotes inverse of relation */
print { connectedness[x] | (x,y)  $\in$  connectedness };

define rtclosure(rel);
  id = { (x,x) | (x,y)  $\in$  rel or (y,x)  $\in$  rel };
  sofar = id; next = id  $\cup$  rel;
  while next  $\neq$  sofar do;
    sofar = next;
    next = id  $\cup$  next  $\circ$  rel; /*  $\circ$  denotes composition of relations */
  endwhile;
  return sofar;
end rtclosure;

```


24 April

1. Find out about type balls, and correspondence VS. abcd

ASCII 963 ?? APL

want 4 bracket pairs

3 . : ; ? !

() [] < > ' ' ° °

* / _ + - = 7

also (?) x → ← & | α λ

2. Meta red, with usual primitives + MU, FINDINNERMOST APPS,
 ↳ for ordinary applications ↳ for pseudo-apps TAU

3. ~~*****~~

$\langle (AT, i, f), (x_1, \dots, x_n) \rangle = (x_1, \dots, \langle f, x_i \rangle, \dots, x_n)$

$\langle (AT, (i_1, \dots, i_m), f), (x_1, \dots, x_n) \rangle = \langle x_{i_1}, \dots, \langle (AT, i_m, f), \dots \rangle \dots \rangle$ the obvious

~~*****~~ (*APP, (,)) ~~*****~~

(25 April 74)

Morris's "Flatten" Example
(see 27 March 74)

$$\begin{aligned}
 a = fl[(A B) C], NIL &= fl[(A B), fl[(C), NIL]] \\
 &= fl[A, fl[(B), fl[(C), NIL]]] \\
 &= cons[A, fl[(B), fl[(C), NIL]]] \\
 &= (A \cdot fl[(B), fl[(C), NIL]])
 \end{aligned}$$

$$\begin{aligned}
 b = fl[(A (B C)), NIL] &= fl[A, fl[(B C), NIL]] \\
 &= cons[A, fl[(B C), NIL]] \\
 &= (A \cdot fl[(B C), NIL])
 \end{aligned}$$

$$\begin{aligned}
 \therefore equal[a, b] &= equal[car a, car b] \wedge equal[cdr a, cdr b] \\
 &= \underline{true} \wedge equal[fl[(B), fl[(C), NIL]], fl[(B C), NIL]]
 \end{aligned}$$

$$\begin{aligned}
 c = fl[(B), fl[(C), NIL]] &= fl[B, fl[NIL, fl[(C), NIL]]] \\
 &= cons[B, fl[NIL, fl[(C), NIL]]] \\
 &= (B \cdot fl[NIL, fl[(C), NIL]])
 \end{aligned}$$

$$\begin{aligned}
 d = fl[(B C), NIL] &= fl[(B C), fl[NIL, NIL]] \\
 &= fl[B, fl[(C), fl[NIL, NIL]]] \\
 &= cons[B, fl[(C), fl[NIL, NIL]]] \\
 &= (B \cdot fl[(C), fl[NIL, NIL]])
 \end{aligned}$$

$$\begin{aligned}
 \therefore equal[c, d] &= equal[car c, car d] \wedge equal[cdr c, cdr d] \\
 &= \underline{true} \wedge equal[fl[NIL, fl[(C), NIL]], fl[(C), fl[NIL, NIL]]]
 \end{aligned}$$

$$e = fl[NIL, fl[(C), NIL]] = cons[NIL, fl[(C), NIL]] = (NIL \cdot fl[(C), NIL])$$

$$\begin{aligned}
 f = fl[(C), fl[NIL, NIL]] &= fl[C, fl[NIL, fl[NIL, NIL]]] \\
 &= cons[C, fl[NIL, fl[NIL, NIL]]] \\
 &= (C \cdot fl[NIL, fl[NIL, NIL]])
 \end{aligned}$$

$$\begin{aligned}
 \therefore equal[e, f] &= equal[car e, car f] \wedge equal[cdr e, cdr f] \\
 &= \underline{false}
 \end{aligned}$$

$$\therefore equal[c, d] = \underline{false} \quad \therefore equal[a, b] = \underline{false}$$

26 April 74

FLATTEN

$$\langle FL, (t, f) \rangle = \text{ATOMS OF } t \text{ 'PREFIXED TO' } f$$

$$SFL1 = (CN (ATOMP 1)$$

$$UN$$

$$(FL1 [(1 1) (FL1 [(TAIL 1) 2])]))$$

$$SFL2 = ((INSERT, CONCAT) (AA, (CN, ATOM$$

$$[\phi]$$

$$FL2)))$$

$$SCONCAT = (2, (WHILE, (NOT, NULL, 1),$$

$$[(TAIL, ROTR), (UN, [(1, ROTR), 2])]))$$

$$= (CN, (NULL 1), 2, (UN, [(1, 1), (CONCAT, [(REST, 1), 2])]))$$

$$SEQUAL = (CN (AND (AA, ATOM))$$

$$EQAT$$

$$(CN (EQAT (AA, LEN))$$

$$((INSERT, AND) (EE, EQUAL))$$

$$(SF)))$$

$$SLEN = ((INSERT, +), (AA, (C, 1)))$$

$$\langle (EQUAL [FL 1] [FL 2]), (T1, T2) \rangle =$$

$$\langle EQUAL \langle FL T1 \rangle \langle FL T2 \rangle \rangle$$

Red Project

1. Goals

a) explore "whole-entity programming"

- It is usually easier to perform operations on aggregates than on individuals in variable-free programming.
- Red seems to allow new operators & meta operators to be defined easily.
- Example: payroll, strongly connected region

b) Minimize the number of concepts in the basic language, while reserving the possibility to define (as a language ~~feed~~ user) new facilities

- Language definition is then short, more likely consistent
-

c) Want ability to do "algebraic transformations" on programs

Reduction property

$$(p \rightarrow f|g) \circ h \equiv p \circ h \rightarrow f \circ h | g \circ h$$

$$h \circ (p \rightarrow f|g) \equiv p \rightarrow h \circ f | h \circ g$$

$$(\text{WHILE } p \text{ } f) \equiv p \rightarrow (\text{WHILE } p \text{ } f) \circ f | I$$

$$\text{REV} \equiv \text{null} \circ \text{head} \circ \text{tail}^n \rightarrow$$

2. Implementation plans & problems

a) Prototype system using LISP

b) Add synonym facility for defining functions

c) "executive"

d) Optimized execution [problems are: ~~these~~ "copy" semantics, large intermediate values,]

- Compact representation by factoring type bits, structure templates
- Deferred evaluation?, Analysis of functions

$$[\text{head} \circ \text{tail}^{n-1}, \text{head} \circ \text{tail}^{n-2}, \dots, \text{head}]$$

$$\text{for } n = 1/2/3/\dots$$

Expression Transformation in LSP

def reverse[l] = rev[l, nil] where rev[l, m] = [null l → nil;
T → rev[t l, h l : m]]

$$\begin{aligned} \therefore \text{reverse}[l] &= \text{rev}[l, \text{NIL}] \\ &= [\text{null } l \rightarrow \text{NIL}; T \rightarrow \text{rev}[t\ l, h\ l : \text{NIL}]] \\ &= [\text{null } l \rightarrow \text{NIL}; \text{null}[\text{tail}[l]] \rightarrow \text{head } l : \text{NIL}; \\ &\quad \dots; \text{null}[\text{tail}^N[l]] \rightarrow \\ &\quad \text{head}[\text{tail}^{N-1} l] : \text{head}[\text{tail}^{N-2} l] : \dots : \text{head}[l] : \text{NIL} \end{aligned}$$

$$\alpha : (\text{DBL} : e) \Rightarrow \alpha : \langle \beta, \beta \rangle$$

$$\beta : e$$

$$(\text{DBL} : e) \rightarrow \langle e, e \rangle$$

$$(1 : \langle e_1, \dots, e_n \rangle) \rightarrow e_1$$

$$(\text{ROTL} : \langle e_1, \dots, e_n \rangle) \rightarrow \langle e_2, \dots, e_n, e_1 \rangle$$

$$(\text{ROTR} : \langle e_1, \dots, e_n \rangle) \rightarrow \langle e_n, e_1, \dots, e_{n-1} \rangle$$

$$(\text{SEP} : \langle e \rangle) \rightarrow \langle e, \text{NIL} \rangle$$

$$(\text{SEP} : \langle e_1, t_2, e_n \rangle) \rightarrow \langle e_1, \langle t_2, \dots, e_n \rangle \rangle$$

$$(\text{JN} : \langle e, \text{NIL} \rangle) \rightarrow \langle e \rangle$$

$$(\text{JN} : \langle e_1, \langle e_2, \dots, e_n \rangle \rangle) \rightarrow \langle e_1, \dots, e_n \rangle$$

$$(\text{COND} : \langle T, e_1, e_2 \rangle) \rightarrow e_1$$

$$(\text{COND} : \langle F, e_1, e_2 \rangle) \rightarrow e_2$$

~~$$(\text{EQAT} : \langle a, a \rangle) \rightarrow T$$~~

~~$$(\text{EQAT} : \langle a_1, a_2 \rangle) \rightarrow F \text{ if } a_1 \neq a_2$$~~

~~$$(\text{EQAT} : \langle a, a \rangle) \rightarrow T$$~~

~~$$(\text{EQAT} : e)$$~~

$$(\text{EQAT} : e) = \begin{cases} \text{error} & \text{if } e \in A \text{ or } e = \langle e_1, \dots, e_n \rangle \text{ and } n \neq 2 \\ T & \text{if } e = \langle a, a \rangle \text{ for some } a \in A \\ F & \text{if } e = \langle c_1, c_2 \rangle, \text{ and } c_1 \notin A \text{ or } c_2 \notin A \text{ or } c_1 \neq c_2 \end{cases}$$

$$(\text{COMP} : \langle \langle \text{COMP}, f_1, f_2, \dots, f_n \rangle, e \rangle)$$

$$\rightarrow (f_1 : (\langle \text{COMP}, f_2, \dots, f_n \rangle : e))$$

$$(\text{COMP} : \langle \langle \text{COMP}, f_1 \rangle, e \rangle)$$

$$\rightarrow (f_1 : e)$$

$$(\text{COMP} : \langle \langle \text{COMP} \rangle, e \rangle)$$

$$\rightarrow e$$

$\mu e = \underline{\text{if } e \in A \text{ then } e}$

$\underline{\text{else if } e = \langle e_1, \dots, e_n \rangle \text{ then } \langle \mu e_1, \dots, \mu e_n \rangle}$

$\underline{\text{else if } e = (e_1 : e_2) \text{ then}}$

$\underline{\text{if } e_1 \notin C \text{ then } \mu(\mu e_1 : e_2)}$

$\underline{\text{else if } e_1 = \langle c_1, \dots, c_n \rangle \text{ then}}$
 $\mu(c_1 : \langle \langle c_1, \dots, c_n \rangle, e_2 \rangle)$

$\underline{\text{else if } e_1 \in A \text{ then}}$

$\underline{\text{if } e_1 = \text{DBL} \text{ then } \langle e_2, e_2 \rangle}$

$\text{EQAT} \circ \text{DBL} = \text{any atom valued function} \equiv \langle C, T \rangle$

$(\text{EQAT} \circ \text{DBL} :$

$\text{atom}[\text{cons}[E, I]] \equiv \underline{\text{false}}$

1 May 1974

Mapping (Compiling) RED Expressions into ISWIM [Compare w/ 22 April 74]

We define a corresponding ISWIM expression E^* for every RED expression E :

$$(E_1, E_2, \dots, E_n)^* = E_1^*, E_2^*, \dots, E_n^* = E_1^* : E_2^* : \dots : E_n^* : ()$$

$$\langle E, F \rangle^* = \text{APP } E^* F^*$$

$$A^* = 'A' \quad (A \text{ an atom})$$

$$x: E_{\text{RED}} \rightarrow E_{\text{ISWIM}} \text{ such that}$$

$$c = \mu_{\text{RED}}^e \Rightarrow c^* = \mu_{\text{ISWIM}} c^*$$

$$\forall c \in C_{\text{RED}}, e \in E_{\text{RED}}$$

where $\text{APP} = \lambda e f. \text{not}(\text{islist } e) \rightarrow \text{RHO } e f$

else $\text{PSI}(\text{head } e) \rightarrow \text{APP}(\text{head } e)(e, f)$

$\text{isnull}(\text{tail } e) \rightarrow \text{APP}(\text{head } e) f$

else $\rightarrow \text{APP}(\text{head } e)(\text{APP}(\text{tail } e) f)$

$\text{PSI } c = \text{not}(\text{islist } c) \rightarrow \text{false}$

↳ take care of not primitives if any...

~~not~~ $(\text{head } c) = \emptyset^* \rightarrow \text{true}$

else $\rightarrow \text{false}$

$\text{RHO } e f = e = \text{DBL}^* \rightarrow f = \emptyset^* \rightarrow \emptyset^*$

else $\rightarrow f, f$

$e = 1^* \rightarrow \text{islist } f \rightarrow \text{head } f$

else $\rightarrow \emptyset^*$

... $e = \text{COND}^* \rightarrow \dots$

$e = \text{FST1}^* \rightarrow \text{not pair of pairs of specified values} \rightarrow \emptyset^*$
else $\rightarrow \text{APP}(\text{HTH } f)(\text{HTHT } f), \text{HTHT } f$

7 May 1974

atoms: identifiers

'string-of-char-reps'

char-rep: any char except ' ' or ' followed by ' ' or code for CR, LF, BS, TAB

have operations to convert quoted strings ^{or atom} to sequence of "characters"

Syntax: have shorthand for NF [f, ... f_n]

CN p → f | g

Use APL ball

C Γc [and change C to K?]

AA [and change AA to MAP?]

Provide ~~map~~ mapping
table for character code
independence.

regular comp: bracket pair? and 1-stroke name. e.g. *

use X:Y for <X,Y>?

use A.B.C for (A B C)?

let g and b be interchangeable separators?

use <> for tuples?

I/O: PRINT EXPR, READ EXPR, PRINT STRING ^{AS IS} { WITHOUT QUOTING }, READ STRING

Error handling: ~~error~~

John Backus:
Definition of RED

0. Let A be a fixed set of "atoms". We will construct I ("items") and R ("rows").

1. $A \subseteq I \subseteq R$

2. $x \in I, y \in R \Rightarrow \text{row}(x, y) \in R$

3. $x \in R \Rightarrow \text{seq}(x) \in I$

4. $x, y \in I \Rightarrow \text{ap}(x, y) \in I$

$\{f: \text{monotonic} \equiv x \leq y \Rightarrow f(x) \leq f(y)\}$

row, seq, ap must be monotonic [map $\perp$ to $\perp$]

5. The only elements of I, R are those determined by a finite number of applications of 1-4.

6. $\text{length}(x) = \begin{cases} 1 & \text{if } x \in I \\ 1 + \text{length}(z) & \text{if } x = \text{row}(y, z) \end{cases}$

7. $h(\text{seq}(x)) = \begin{cases} x & \text{if } x \in I \\ y & \text{if } x = \text{row}(y, z) \end{cases}$ [claim monotonic]

8. $\mu x = x \in A$ then x [claim continuous, so least fixed point exists]

$x = \text{row}(y, z)$ then $\text{row}(\mu y, \mu z)$

$x = \text{seq}(y)$ then $\text{seq}(\mu y)$

$x = \text{ap}(y, z)$ then

$\mu y \neq y$ or $\mu z \neq z$ then $\mu \text{ap}(\mu y, \mu z)$

$y \in A$ then $\mu(p(y)(z))$

else $\mu \text{ap}(h(y), \text{seq}(\text{row}(y, z)))$

how about if $y \in R$? impossible: see rule 4 above.

9. $p_y =$ a partial function from C to R , for all $y \in A$ [p_y must be monotonic]

primitives + definitions

Proposal for Quoted Expressions in RED.

Consider adding a new one-place constructor quote to Red. (Denote $\text{quote}[e]$ by $\{e\}$.) The semantics of quoted expressions are determined by some new cases for the definition of μ , and by some new primitives:

$$\mu\{e\} = \{e\}$$

$$\mu \text{ap}[\{e_1\}, e_2] = \phi$$

$$\text{predicates} \left\{ \begin{array}{l} \text{ISQATOM} : \phi \rightarrow \phi; \{a\} \rightarrow T; F \\ \text{ISQQUOTE} : \phi \rightarrow \phi; \{\{e\}\} \rightarrow T; F \\ \text{ISQSEQ} : \phi \rightarrow \phi; \{(e_1, \dots, e_N)\} \rightarrow T; F \\ \text{ISQAP} : \phi \rightarrow \phi; \{\langle e_1, e_2 \rangle\} \rightarrow T; F \end{array} \right.$$

Also need ISATOM, ISSEQ, ISQUOTE for expressions in general.

$$\text{selectors} \left\{ \begin{array}{l} \text{QHEAD} : \{(e_1, \dots, e_N)\} \rightarrow \{e_1\}; \phi \\ \text{QTAIL} : \{(e_1)\} \rightarrow \{\emptyset\}; \{(e_1, \dots, e_N)\} \rightarrow \{(e_2, \dots, e_N)\}; \phi \\ \text{QRATOR} : \{\langle e_1, e_2 \rangle\} \rightarrow \{e_1\}; \phi \\ \text{QRAND} : \{\langle e_1, e_2 \rangle\} \rightarrow \{e_2\}; \phi \end{array} \right.$$

$$\text{constructors} \left\{ \begin{array}{l} \text{QCONST} : \phi \rightarrow \phi; x \rightarrow \{x\} \\ \text{QSEQ [QUN]} : (\{e\}, \{\emptyset\}) \rightarrow \{(e)\}; (\{e\}, \{(e_2, \dots, e_N)\}) \rightarrow \{(e, e_2, \dots, e_N)\}; \phi \\ \text{QAP} : (\{e_1\}, \{e_2\}) \rightarrow \{\langle e_1, e_2 \rangle\}; \phi \end{array} \right. \quad [\text{can't have } \{\phi\}]$$

$$\text{UNQUOTE} : \{e\} \rightarrow e; \phi$$

The constructor quote violates AL3 ($\mu\{e\} \neq \{\mu e\}$), but is useful in implementing extended system features (e.g. trace) in Red itself. Note that these rules for quoted expressions preserve the idempotency of μ .

John's proposal for a quote-like feature (actually, "pseudo-applications")

There is a new (pseudo-) function named down, from E into C: ($\nabla \equiv$ down)

$$\nabla x \equiv x = AP/\bar{n} \Rightarrow AP/\bar{n}1 \quad [\text{e.g. } AP/1 \rightarrow AP/2]$$

$$x \in A \Rightarrow x$$

$$x = \text{row}(y, z) \Rightarrow \text{row}(\nabla y, \nabla z)$$

$$x = \text{seq}(y) \Rightarrow \text{seq}(\nabla y)$$

$$x = \text{ap}(y, z) \Rightarrow \text{seq}(\text{row}(AP/1, \text{seq}(\text{row}(\nabla y, \nabla z))))$$

Because ∇ is a pseudo-function, the definition of μ must be altered to handle it:

$$\mu x \equiv \dots$$

$$x = \text{ap}(y, z) \Rightarrow \dots$$

$$y = \nabla \Rightarrow \nabla z$$

Up, or Δ , is the "inverse function", and is a true RED function. Its effect is:

$$\Delta x \equiv x = AP/\bar{n}1 \Rightarrow AP/\bar{n}$$

$$x \in A \Rightarrow x$$

$$x = \text{seq}(\text{row}(AP/1, \text{seq}(\text{row}(y, z)))) \Rightarrow \langle \Delta y, \Delta z \rangle$$

$$x = \text{row}(y, z) \Rightarrow \text{row}(\Delta y, \Delta z)$$

$$x = \text{seq}(y) \Rightarrow \text{seq}(\Delta y)$$

$$x = \text{ap}() : \text{IMPOSSIBLE}$$

Comparison with a quote constructor [15 May 74]:

1. This scheme requires the introduction of the concept of "pseudo-function".
2. The expression $\langle \text{APPLY}(\langle F \ x \rangle \langle G \ y \rangle) \rangle$, where $\mu \langle F \ x \rangle \equiv \nabla$, does not quote $\langle G \ y \rangle$.
3. Δ & ∇ must make global changes to their arguments, while quote/unquote make a single [bounded] change.

John's definitions

~~John's~~

$$T x \equiv x = | y \rightarrow T \delta y$$

$$x = ap[y, z] \rightarrow y = | w \vee y = ap[p, q] \rightarrow Tap[Ty, z]$$

$$y = seq[low[z, w]] \rightarrow \text{meta comp}$$

$$y = [w] \rightarrow Tap[h[w], seq[low[$$

$$y \in A \rightarrow T \delta p y[z]$$

$$T \rightarrow x$$

$$\in E \rightarrow E$$

Top down, 1 step
[like μ_1 from 22 May 74]

$$\mu^* x = \text{if } x = | y \rightarrow \mu^* T y$$

$$x \in A \rightarrow x$$

$$x = y, z \rightarrow \mu^* y, \mu^* z$$

$$x = [y] \rightarrow [\mu^* y]$$

$$x = \langle y, z \rangle \rightarrow \mu^* T x$$

Top down, all the way

$$[p \text{ APPLY}] x = \tau x = (e_1, e_2) \rightarrow \langle e_1 e_2 \rangle$$

$$\mu x = x = | y \rightarrow \mu \delta y$$

$$x \in A \rightarrow x \quad \text{depth} = 1$$

$$x = y, z \rightarrow \mu y, \mu z \quad \text{depth} = d_y + d_z + 1$$

$$x = [y] \rightarrow [\mu y] \quad \text{depth} = d_y + 1$$

$$x = \langle y, z \rangle \rightarrow y = | w \rightarrow \mu \langle \delta w, z \rangle$$

$$y \in A \rightarrow \mu \delta y [\mu z] \quad \text{depth} = d_y + d_z + d_{pp}(\mu z)$$

$$y = [w] \rightarrow \mu \langle h[w], [y, z] \rangle$$

$$y = \langle p, q \rangle \rightarrow \mu \langle \mu y, z \rangle$$

Bottom up

Initial RED system

I. Character set: APL

[]	sequence brackets	()	} reserved
< >	string quotes		
< >	application ^{brackets} sign (infix, right associative)		
..	escape character (within strings)	;	

these are the system characters

II. Atoms

A. string: a sequence of string items, preceded by < and followed by > .

a string item is: any character except < > ..

or: a string

or: .. followed by one of < > R B T L

B. integer: sequence of digits, possibly preceded by + or -

[C. truth value: T F]

D. symbol: any (other) sequence of non system characters

III. Printing strings

A. First mode is literally, complete with outer quotes

B. Second mode removes outermost quote, maps <char> into appropriate character:

<char>	effect
..	print dieresis
<	print left string quote
>	print right string quote
R	carriage return
B	backspace
T	tab
L	linefeed
other	print <char>

} Question: what is low-level interface to terminal in VM?

IV. Miscellaneous

Empty sequence: NIL

• = RC

Unspecified: $\square$ I = identity funV. Syntax

1. Atom $\in I \in R$
2. $x \in I, y \in R \rightarrow x \overset{\text{row}[x,y]}{g} y \in R$
3. $x \in R \rightarrow [x]$
4. $x, y \in I \rightarrow \langle xgy \rangle \in I$

Convention blanks, commas are interchangeable
as sequence element separators.

Top-down vs. bottom up evaluation

↓
and "bar" for definitions

μ is as defined previously [e.g. 10 May 74]. But we also have μ_1 ["1-level evaluation"]

$$\mu_1 x \equiv x \in A \vee x = \text{row}[y, z] \vee x = \text{seq}[y] \rightarrow x$$

$$x = \text{ap}[y, z] \rightarrow \mu y \neq y \rightarrow \mu_1 \text{ap}[\mu_1 y, z]$$

$$y \in A \rightarrow \mu_1 \cdot \mu_2[y] z$$

$$y = \text{seq}[z] \rightarrow \mu_1 \text{ap}[\mu_2[y], \text{seq}[\text{row}[y, z]]]$$

$\mu_2[y]$ may have to apply μ or μ_1 to its argument z .

24 May 74

①

COMP360((

(READEXPR (LAMBDA () (PROG (TOK)

(SETQ TOK (READTOKEN))

(RETURN (SELECT (CAR TOK)

((QUOTE ATOM) TOK)

((QUOTE BAR) (READBAR))

((QUOTE LSEQ) (READSEQ)) — ((QUOTE RSEQ) (QUOTE (BADEXP EXTRA_RSEQ)))

← ((QUOTE LAP) (READAP)) — ((QUOTE RAP) (QUOTE (BADEXP EXTRA_RAP)))

← (CONS (QUOTE BADEXP) TOK))))))

(READBAR (LAMBDA () (PROG (TOK)

(SETQ TOK (READTOKEN))

(RETURN (COND ((EQ (CAR TOK) (QUOTE ATOM))

(CONS (QUOTE BAR) TOK))

(T (QUOTE (BADEXP BAR_USAGE))))))

(READSEQ (LAMBDA () (PROG (SEQ)

(SETQ SEQ (READSEQELTS))

(RETURN (COND ((NULL SEQ) (QUOTE (ATOM . NIL)))

((EQ (CAR SEQ) (QUOTE BADEXP)) SEQ)

(T (CONS (QUOTE SEQ) SEQ))))))

24 May 74

②

```
(READSEQELTS (LAMBDA () (PROG (HEAD TAIL)
  (SETQ HEAD (READEXPR))
  (COND ((EQ HEAD (QUOTE (BADEXP EXTRA_RSEQ))) (RETURN NIL))
        ((EQ (CAR HEAD) (QUOTE BADEXP)) (RETURN HEAD)) )
  (SETQ TAIL (READSEQELTS))
  (RETURN (COND ((EQ (CAR TAIL) (QUOTE BADEXP)) TAIL)
                (T (CONS HEAD TAIL)))))))
```

```
(READAP (LAMBDA () (PROG (RATOR RAND)
  (SETQ RATOR (READEXPR))
  (COND ((EQ (CAR RATOR) (QUOTE BADEXP)) (RETURN RATOR)))
  (SETQ RAND (READEXPR))
  (RETURN (COND ((EQ (CAR RAND) (QUOTE BADEXP)) RAND)
                ((EQ (CAR (READTOKEN)) (QUOTE RAP))
                 (CONS (QUOTE AP) (CONS RATOR RAND)))
                (T (QUOTE (BADEXP MISSING_RAP)))))))
```

28 May 74

(3)

```
(READTOKEN (LAMBDA () (PROG (CH)
  (SETQ CH (RDCHR))
  (RETURN (SELECT (CHARCLASS CH)
    ((QUOTE SEP) (READTOKEN))
    ((QUOTE DELIM) (DELIMTOKEN CH))
    ((QUOTE LSTR) (READSTRING))
    ((QUOTE SYM) (READSYMBOL CH))
    (CONS (QUOTE BADTOK) CH))))))
```

```
(READSTRING (LAMBDA () (PROG ()
  (CLEARBUFF)
  (PACK LSTR)
  (READSTR)
  (RETURN (CONS (QUOTE ATOM) (INTERNX))))))
```

```
(READSTR (LAMBDA () (PROG (CH CLASS)
  LP (SETQ CH (RDCHR))
    (SETQ CLASS (CHARCLASS CH))
    (COND ((EQ CLASS (QUOTE RSTR)) (RETURN (PACK CH)))
          ((EQ CLASS (QUOTE LSTR)) (PROG2 (PACK CH) (READSTR)))
          (T (PACK CH))))
  (GO LP))))
```


28 May 74

(4)

(READSYMBOL (LAMBDA (FIRST) (PROG (CH CLASS)

(CLEARBUFF)

(PACK FIRST)

LP (SETQ CH (RDCHR))

(SETQ CLASS (CHARCLASS CH))

(COND ((NOT (EQ CLASS (QUOTE SYM)))

(RETURN (PROG2 (BACKCHR) (CONS (QUOTE ATOM) (INTERNX))))))

(PACK CH)

(GO LP)))

(CHARCLASS (LAMBDA (CH)

(COND ((MEMBER CH SEPARATORS) (QUOTE SEP))

((MEMBER CH DELIMS) (QUOTE DELIM))

((EQ CH LSTR) (QUOTE LSTR))

((EQ CH RSTR) (QUOTE RSTR))

((EQ CH ESC) (QUOTE ESC))

(T (QUOTE SYM))))

28 May 74

(5)

must be first

(DELIMTOKEN (LAMBDA (CH)

(SELECT CH

(LSEQ (QUOTE (LSEQ)))

(RSEQ (QUOTE (RSEQ)))

(LAP (QUOTE (LAP)))

(RAP (QUOTE (RAP)))

(BAR (QUOTE (BAR)))

(CONS (QUOTE (BADTOK)) CH)))

))

SPECIAL((LSTR RSTR ESC ESCAPEDLITS

LSEQ RSEQ LAP RAP BAR DELIMS SEPARATORS))

CSET (LSTR 'C')

CSET (RSTR 'D')

CSET (ESC '...')

CSET (ESCAPEDLITS ('... 'C 'D'))

CSET (LSEQ '[')

CSET (RSEQ ']')

CSET (LAP '<')

CSET (RAP '>')

CSET (BAR '|')

CSET (DELIMS ('[' ']' '<' '>' '|'))

CSET (SEPARATORS (' ' ','))

28 May 74

6

```
(PRINTEXPR (LAMBDA (EXPR LITFLAG)
  (COND ((ATTYPE EXPR) (PRINTATOM (CDR EXPR) LITFLAG))
        ((BARTYPE EXPR) (PROG2 (PRINTCH BAR)
                                (PRINTATOM (CDDR EXPR) LITFLAG))))
        ((SEQTYPE EXPR) (PRINTSEQ (CDR EXPR) LITFLAG))
        ((APTTYPE EXPR) (PRINTAP (CDR EXPR) LITFLAG))))))
```

```
(PRINTATOM (LAMBDA (ATOM LITFLAG) (PROG (L)
  (COND (LITFLAG (RETURN (PRIN1 ATOM))))
  (SETQ L (EXPLODE ATOM))
LP (SETQ L (CDR L))
  (COND ((EQ L (LIST RSTR)) (RETURN))
        ((EQ (CAR L) ESC)
         (PROG2 (SETQ L (CDR L))
                 (COND ((MEMBER (CAR L) ESCAPEDLITS)
                        (PRINTCH (CAR L)))
                       (T (PROG2 (PRINTCH ESC) (PRINTCH (CAR L)))))))
        (T (PRINTCH (CAR L))))
  (GO LP))))
```

```
(PRINTSEQ (LAMBDA (SEQ LITFLAG) (PROG ()
  (PRINTCH LSEQ)
  (PRINTEXPR (CAR SEQ) LITFLAG)
  (PRINTTAIL (CDR SEQ) LITFLAG))))
```

29 May 74

⑦

```
(PRINTTAIL (LAMBDA (TAIL LITFLAG)
  (COND ((NULL TAIL) (PRINTCH RSEQ))
        (T (PROG() (PRINTCH BLANK)
                    (PRINTEXPR (CAR TAIL) LITFLAG)
                    (PRINTTAIL (CDR TAIL) LITFLAG) )))))
```

```
(PRINTAP (LAMBDA (AP LITFLAG) (PROG ()
  (PRINTCH LAP)
  (PRINTEXPR (CAR AP) LITFLAG)
  (PRINTCH BLANK)
  (PRINTEXPR (CDR AP) LITFLAG)
  (PRINTCH RAP) )))
```

```
(ATTYPE (LAMBDA (E) (EQ (CAR E) (QUOTE ATOM))))
(BARTYPE (LAMBDA (E) (EQ (CAR E) (QUOTE BAR))))
(SECTYPE (LAMBDA (E) (EQ (CAR E) (QUOTE SEQ))))
(APTYPE (LAMBDA (E) (EQ (CAR E) (QUOTE AP))))
```


Adding Side Effects to RED ...

assume we have added "pseudo-functions" to RED [i.e. functions with side-effects]

Thus $\langle \text{PRINTLINE } x \rangle$ prints a line containing a representation of x . Its value is, say, x .

We would like to add facilities like streams & locks, ~~for example~~ but it seems the basic RED sequencing mechanisms can't stand the strain. For example, suppose $\langle \text{LOCK } x \rangle$ halts the processor until lock x is free, then seizes lock x ; $\langle \text{UNLOCK } x \rangle$ performs the complementary function.

Suppose we want to define a pseudo function PRINTLONGMESSAGE :

$$\langle \text{PRINTLONGMESSAGE}, (l_1, \dots, l_N) \rangle = (l_1, \dots, l_N)$$

and has the sideeffect of printing l_i on successive lines. To avoid clashes with other users of the printer, we will associate a lock PRINTLOCK with the printer. Here is a definition:

$$\begin{aligned} \text{PRINTLONGMESSAGE} \equiv & (\text{UNLOCK } (C, \text{PRINTLOCK}) \\ & (\text{WHILE } (\text{NOT NULL}) \\ & \quad (\text{TAIL } (AL, \text{PRINTLINE}))) \\ & 1 [\neq (\text{LOCK } (C, \text{PRINTLOCK}))]) \end{aligned}$$

If this is executed strictly bottom-up, every thing is o.k. But if "claves" top-down methods were used, UNLOCK would be applied to PRINTLOCK , and then the computation would stop!

1 June 74

RED INTERPRETER

REDCAT ℓ is the identity function if no element of ℓ is a row;
otherwise it moves each row "up" into ℓ .

(MEANING (LAMBDA (E)

(COND ("ATOMTYPE E" E)

("SEQTYPE E")

(CONS (QUOTE SEQ)

(REDCAT (MAPLIST (CDR E) (FUNCTION MEANING))))

("APTYPE E") (REDAPPLY (MEANING (RATOR (CADR E))

(MEANING (RAND (CDDR E)))))

(T (ERROR2 (QUOTE (MALFORMED ^{REDU}EXPR)) E))))

(REDAPPLY (LAMBDA (F A)

(COND ("ATOMTYPE F" ~~F~~)

(COND ((NOT (EQUAL (DEFINITION F) F))

(REDAPPLY (DEFINITION F) A))

(T (MEANING (DENOTATION F A)))))

("SEQTYPE F")

(REDAPPLY (CADR F)

(CONS (QUOTE SEQ)

(LIST F A))))

(T (ERROR2 (QUOTE (MALFORMED RATOR)) F))))

(DEFINITION (LAMBDA (^FATOM)

(COND ((NULL (GET (CDR ATOM) (QUOTE REDDEF))) ^FATOM)

(T (GET (CDR ATOM) (QUOTE REDDEF)))))

(DENOTATION (LAMBDA (F A)

(COND ((NULL (GET (CDR ^FATOM) (QUOTE REDPRIM))) UNSPEC)

(T (APPLY (GET (CDR F) (QUOTE REDPRIM)) A NIL))))

1 June 74

RED INTERPRETER

```
('1 (LAMBDA (X)
      (COND ((SEQTYPE X) (CADR X) )
              (T UNSPEC)))))
```

```
('2 (LAMBDA (X) ('1 (TAIL X))))
```

```
('3 (LAMBDA (X) ('1 (TAIL (TAIL X))))))
```

```
(TAIL (LAMBDA (X)
      (COND ((SEQTYPE X)
              (COND ((NULL (CDDR X)) REDNIL)
                      (T (CONS (QUOTE SEQ)
                                (CDDR X)))))
              (T UNSPEC))))))
```

```
(EQ (LAMBDA (X)
      (COND ((NOT (SEQTYPE X)) UNSPEC)
              ((NOT (EQP (LENGTH (CDR X)) 2)) UNSPEC)
              ((ATYPE (CADR X))
               (COND ((EQUAL (CADR X) UNSPEC) UNSPEC)
                       (T REDNIL)))))
```

1 June 74

Some Macro-Extensions to LISP

$(\text{WHILE } b \ s1 \ s2 \ \dots \ sN) \Rightarrow (\text{PROG NIL}$
LP (COND ((NOT b) (RETURN)))
s1 s2 ... sN
(GO LP))

$(\text{LET } (v1 \ e1)(v2 \ e2) \dots (vN \ eN) \ e) \Rightarrow$
((LAMBDA (v1 v2 ... vN) e) e1 e2 ... eN)

$(\text{PROGN } s1 \ s2 \ \dots \ sN) \Rightarrow \text{~~(PROG NIL s1 s2 ... sN)~~}$
 $((\text{LAMBDA } (v1 \ v2 \ \dots \ vN) \ vN) \ s1 \ s2 \ \dots \ sN)$

Areas for Study of "Programming Languages:

1. Foundations or framework: basic semantic model (e.g. λ -calculus, RED, flowgraphs, dataflow, ...)
2. Design of "realistic language" (efficient implementability, ~~understandable~~ good human engineering)
3. Programming systems: integrated program development system: ^{structural vs. text} editor, interpreter, debugger, documentation aids, verifier. [Examples: JOSS, APL, INTERLISP]
 - + spelling correction, rectifying & undoing of previously input commands [Teitelman]
 - ↓ concordances, control flow diagrams, prettyprint

Notes:

1. Can develop whole entity programming techniques [a implementations thereof] in, say, LISP.
2. If we are interested in Programming Systems and frameworks, suggest finding a suitable framework for INTERLISP.

- ✓ [See Barry Rosen's notes]
1. Prove any bottom up evaluation seq gives same pr
 2. " top down evaluation is an extension
 3. Consider, say, an interactive text editing program. Give a formal semantics by considering each command's execution as recursive mapping from texts to texts, & somehow formalize the "terminal interactions".

Axioms for "side effects"

~ June 74

Talk 20 June 74?

See Engagement book

A complete language is a triple (E, C, μ)

with:

- 1) $C \subseteq E$
- 2) μ is a partial map from E onto C
- 3) C is the set of fixedpoints of μ

Example: the λ -calculus

E = the set of well-formed formulas

C = the set of principal normal forms

μ is defined for all $e \in E$ with a normal form,
in which case μe = the principal normal form of e

(the Church-Rosser theorem assures that μ is well defined)

Red

"SYNTAX"

E1. There is a set A of atoms such that $A \subset E$.

E2. If $e_1, e_2, \dots, e_n \in E$, then $(e_1, e_2, \dots, e_n) \in E$. (sequence)

E3. If $f, a \in E$, then $\langle f, a \rangle \in E$. (application)

E4. E is the smallest set satisfying E1-E3 above.

A constant (member of C) is an expression containing no subexpressions which are applications. Thus:

C1. $A \subset C$

C2. If $c_1, c_2, \dots, c_n \in C$, then $(c_1, c_2, \dots, c_n) \in C$.

C3. C is the smallest set satisfying C1 and C2.

Clearly, $C \subset E$.

RED SEMANTICS

Red is a family of languages:

All have same syntax (E, C)

All have same basic semantics (μ)

But each has different set of primitive operators

To specify a particular language, we provide a partial function ρ , with $\rho: A \rightarrow [C \rightarrow C]$.

If ρa is defined, its value is a primitive function $(a \in A)$.

Here is a recursive definition of μ :

$\mu e \stackrel{\text{def}}{=} \text{if } e \in A \text{ then } e$

elseif $e = (e_1, \dots, e_n)$ then $(\mu e_1, \dots, \mu e_n)$

elseif $e = \langle f, a \rangle$ then

if $f \in A$ then $\mu[\rho f] \mu a$

elseif $f = (f_1, \dots, f_n)$ then $\mu \langle f_1, ((f_1, \dots, f_n), a) \rangle$

Mathematical semantics

μ is a (partial) map from E onto C $\rightarrow$ least μ is uniquely defined as the "root" or fixedpoint of the defining equation given previously

Operational semantics (a set of transformation rules which, if repeatedly applied to a meaningful expression, will eventually produce the meaning of the expression).

If $\langle f, a \rangle$ is a subexpression of e and neither f nor a contain applications, then replace $\langle f, a \rangle$ in e by

$$\begin{aligned} [pf]a & \text{ if } f \text{ is an atom} \\ \langle f_i, (f, a) \rangle & \text{ if } f = (f_i, \dots, f_n) \end{aligned}$$

The "extended Church-Rosser property" for Red states that if μe is defined, then every sequence of reductions on e terminates, producing μe .

Some Red primitives ("regular")

$$\mu \langle \text{PLUS}, (i, j) \rangle = i + j \quad [\text{also TIMES, MINUS, etc.}]$$

$$\mu \langle 1, (c_1, \dots, c_n) \rangle = c_1 \quad [\text{also } 2, 3, \dots]$$

$$\mu \langle \text{TAIL}, (c_1, \dots, c_n) \rangle = \begin{cases} \text{NIL} & \text{if } n=1 \\ (c_2, \dots, c_n) & \text{otherwise} \end{cases}$$

$$\mu \langle \text{DISTL}, (x, (y_1, \dots, y_n)) \rangle = ((x, y_1), \dots, (x, y_n))$$

$$\mu \langle \text{DISTR}, ((x_1, \dots, x_n), y) \rangle = ((x_1, y), \dots, (x_n, y))$$

$$\mu \langle \text{SEP}, (x_1) \rangle = (x_1, \text{NIL})$$

$$\mu \langle \text{SEP}, (x_1, x_2, \dots, x_n) \rangle = (x_1, (x_2, \dots, x_n))$$

$$\mu \langle \text{UN}, (x_1, \text{NIL}) \rangle = (x_1)$$

$$\mu \langle \text{UN}, (x_1, (x_2, \dots, x_n)) \rangle = (x_1, x_2, \dots, x_n)$$

$$\begin{aligned} \mu \langle \text{TRANS}, ((x_{11}, \dots, x_{1n}), \dots, (x_{m1}, \dots, x_{mn})) \rangle \\ = \mu \langle (x_{11}, \dots, x_{m1}), \dots, (x_{1n}, \dots, x_{mn}) \rangle \end{aligned}$$

⋮

$$\mu \langle \text{I}, x \rangle = x$$

Some Red primitives ("meta")

$$\mu\langle \text{COMP}, f_1, \dots, f_n, x \rangle = \mu\langle f_1, \langle f_2, \dots \langle f_n, x \rangle \dots \rangle \rangle$$

$$\mu\langle \text{CN}, p, f, g, x \rangle = \begin{cases} \mu\langle f, x \rangle & \text{if } \mu\langle p, x \rangle = T \\ \mu\langle g, x \rangle & \text{if } \mu\langle p, x \rangle = F \end{cases}$$

$$\mu\langle \text{NF}, f_1, \dots, f_n, x \rangle = \mu(\langle f_1, x \rangle, \dots, \langle f_n, x \rangle)$$

$$\mu\langle \text{C}, x, y \rangle = x$$

$$\mu\langle \text{AA}, f, (x_1, \dots, x_n) \rangle = \mu(\langle f, x_1 \rangle, \dots, \langle f, x_n \rangle)$$

$$\mu\langle \text{INSERT}, f, (y_1) \rangle = \mu y_1$$

$$\mu\langle \text{INSERT}, f, (y_1, y_2, \dots, y_n) \rangle = \mu\langle f, (y_1, \langle f, (y_2, \dots \langle f, (y_{n-1}, y_n) \rangle \dots) \rangle) \rangle$$

$$\mu\langle \text{EE}, f, ((x_1, \dots, x_n), (y_1, \dots, y_n)) \rangle = \mu(\langle f, (x_1, y_1) \rangle, \dots, \langle f, (x_n, y_n) \rangle)$$

Some Red definitions

IP def (COMP, (INSERT, PLUS), (EE, TIMES))

[APL: $R \leftarrow +/A \times B$]

Example

$\langle IP, ((1,2,3), (4,5,6)) \rangle$

$= \langle (COMP, (INSERT, +), (EE, \times)), ((1,2,3), (4,5,6)) \rangle$

[def of IP]

$= \langle (INSERT, +), \langle (EE, \times), ((1,2,3), (4,5,6)) \rangle \rangle$

[def of COMP]

$= \langle (INSERT, +), (\langle \times, (1,4) \rangle, \langle \times, (2,5) \rangle, \langle \times, (3,6) \rangle) \rangle$

[def of EE]

$= \langle (INSERT, +), (4, 10, 18) \rangle$

[def of $\times$]

$= \langle + (4, \langle +, (10, 18) \rangle) \rangle$

[def of INSERT]

$\vdots$

$= 32$

[def of +]

Some Red definitions

$$\text{MM} \stackrel{\text{def}}{=} (\text{COMP}, (\text{AA}, (\text{AA}, \text{IP})), \\ (\text{AA}, \text{DISTL}), \\ \text{DISTR}, \\ \text{TRANS}, \\ (\text{NF}, 1, (\text{COMP}, \text{TRANS}, 2)))$$

Example

$$\text{let } a = ((a_{11}, \dots, a_{1n}), \dots, (a_{m1}, \dots, a_{mn}))$$

$$\text{let } b = ((b_{11}, \dots, b_{1p}), \dots, (b_{n1}, \dots, b_{np}))$$

$$\text{let } a_i = (a_{i1}, \dots, a_{in})$$

To compute $\langle \text{MM}, (a, b) \rangle$, we proceed as follows:

$$\langle (\text{NF}, 1, (\text{COMP}, \text{TRANS}, 2)), (a, b) \rangle = (a, b^T) \\ \text{where } b^T_{ij} \stackrel{\text{def}}{=} b_{ji}$$

$$\langle \text{DISTR}, (a, b^T) \rangle = ((a_1, b^T), \dots, (a_m, b^T))$$

$$\langle (\text{AA}, \text{DISTL}), ((a_1, b^T), \dots, (a_m, b^T)) \rangle \\ = (((a_1, b^T_1), \dots, (a_1, b^T_p)), \dots, ((a_m, b^T_1), \dots, (a_m, b^T_p)))$$

$$\langle (\text{AA}, (\text{AA}, \text{IP})), \dots \rangle = C \quad \text{where } C_{ij} = \langle \text{IP}, (a_i, b^T_j) \rangle$$

change /s1 [/s2 [/ [{ 1 | n | * } [G | *]]]

delete [1 | n | *]

replace [line]

insert [line]

find line

locate /string/

up [1 | n]

down [1 | n]

top
bottom

type [{ 1 | m | * } [n | *]]

A text file is a sequence of lines $\langle l_1, l_2, \dots, l_n \rangle$

A line l_i is a sequence of characters $\langle c_{i1}, c_{i2}, \dots, c_{in_i} \rangle$

The editor state is a pair $\langle I, F \rangle$ where $F = \langle l_1, l_2, \dots, l_n \rangle$ is the file
 $I \in \{0, 1, \dots, n\}$ is the cursor

top: $\langle I, F \rangle \rightarrow \langle 0, F \rangle$

bottom: $\langle I, F \rangle \rightarrow \langle n, F \rangle$ where $F = \langle l_1, \dots, l_n \rangle$

$\langle \text{up}, i \rangle: \langle I, F \rangle \rightarrow \langle I \ominus i, F \rangle$ } where $\ominus$ & $\oplus$ "hang" at end of file

$\langle \text{down}, i \rangle: \langle I, F \rangle \rightarrow \langle I \oplus i, F \rangle$ }

$\langle \text{find}, L \rangle: \langle I, F \rangle \rightarrow \langle J, F \rangle$ where $J = \begin{cases} \text{least integer} > I \text{ s.t. } L = \text{initial string of } F_J \\ n & \text{if no such int exists} \end{cases}$

$$1. f \circ (p \rightarrow g; h) \equiv (p \rightarrow f \circ g; f \circ h)$$

while $(p \rightarrow g; h) \circ f \equiv (p \circ f \rightarrow g \circ f; h \circ f)$ [See 3. below]

$$2. \alpha f \circ \alpha g \equiv \alpha(f \circ g)$$

$$\text{Pf. } (\alpha f \circ \alpha g) : x \equiv \alpha f : \alpha g : x$$

def of $\circ$

$$\equiv \alpha f : (x = \emptyset \rightarrow \emptyset;$$

def of α

$$x = \langle x_1, \dots, x_n \rangle \rightarrow \langle g : x_1, \dots, g : x_n \rangle;$$

$$\perp)$$

~~$$\alpha f : \emptyset = \emptyset \rightarrow \emptyset; \emptyset = \langle x_1, \dots, x_n \rangle \rightarrow \dots; \perp)$$~~

$$\equiv (x = \emptyset \rightarrow \alpha f : \emptyset;$$

$\perp$

$$x = \langle x_1, \dots, x_n \rangle \rightarrow \alpha f : \langle g : x_1, \dots, g : x_n \rangle;$$

$$\alpha f : \perp)$$

$$\equiv (x = \emptyset \rightarrow \emptyset; (\emptyset = \emptyset \rightarrow \emptyset;$$

def of α

$$\emptyset = \langle \dots \rangle \rightarrow \dots;$$

$$\perp);$$

$$x = \langle x_1, \dots, x_n \rangle \rightarrow (\langle g : x_1, \dots, g : x_n \rangle = \emptyset \rightarrow \emptyset;$$

$$\langle g : x_1, \dots, g : x_n \rangle = \langle y_1, \dots, y_n \rangle \rightarrow$$

$$\langle f : y_1, \dots, f : y_n \rangle;$$

$$\perp);$$

$$\perp) (\perp = \emptyset \rightarrow \emptyset; \perp = \langle \dots \rangle \rightarrow \dots; \perp))$$

$$\equiv (x = \emptyset \rightarrow \emptyset;$$

simplification

$$x = \langle x_1, \dots, x_n \rangle \rightarrow \langle f : g : x_1, \dots, f : g : x_n \rangle;$$

$$\perp)$$

$$\equiv \alpha f : (x = \emptyset \rightarrow \emptyset; x = \langle \dots \rangle \rightarrow \langle (f \circ g) : x_1, \dots, (f \circ g) : x_n \rangle; \perp)$$

def of $\circ$

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

def of α

QED

$$\text{Pf. d. 1 } f \circ (p \rightarrow g; h) : x \equiv f : (p \rightarrow g; h) : x \quad \text{def of } \circ$$
~~$$\equiv f : (p : x = T \rightarrow g : x; p : x = F \rightarrow h : x; \perp)$$~~

$$\equiv f : (\text{if } p : x = T \text{ then } g : x \\ \text{elseif } p : x = F \text{ then } h : x \\ \text{else } \perp) \quad \text{def of } \rightarrow$$

$$\equiv \text{if } p : x = T \text{ then } f : g : x \\ \text{elseif } p : x = F \text{ then } f : h : x \\ \text{else } f : \perp \quad \{ \text{which} = \perp \} \quad ?$$

$$\equiv \text{if } p : x = T \text{ then } f \circ g : x \\ \text{elseif } p : x = F \text{ then } f \circ h : x \\ \text{else } \perp \quad \text{def of } \circ$$

$$\equiv (p \rightarrow f \circ g; f \circ h) : x \quad \text{def of } \rightarrow$$

QED

$$3. (p \rightarrow f; g) \circ h \equiv (p \circ h \rightarrow f \circ h; g \circ h)$$

Pf. (similar to 1)

$$4. [f_1, \dots, f_n] \circ g \equiv [f_1 \circ g, \dots, f_n \circ g]$$

$$5. \alpha f \circ [g_1, \dots, g_n] \equiv [f \circ g_1, \dots, f \circ g_n]$$

$$\text{Pf. } \alpha f \circ [g_1, \dots, g_n] : X \equiv \alpha f : [\dots g_i \dots] : X$$

$$\equiv \alpha f : \langle \dots g_i : x \dots \rangle$$

def of $\cdot$
 def of α
 def of α

assume $g_i : x = \perp$, all is $\perp$

$$\left\{ \begin{array}{l} \equiv \langle \dots g_i : x \dots \rangle = \emptyset \rightarrow \emptyset; \\ \langle \dots g_i : x \dots \rangle = \langle \dots y_i \dots \rangle \rightarrow \langle \dots f : y_i \dots \rangle; \\ \perp \end{array} \right.$$

$$\equiv \langle \dots f : g_i : x \dots \rangle \equiv \langle \dots f \circ g_i : x \dots \rangle$$

$$\equiv [\dots f \circ g_i \dots] : x$$

6. Would like: $(\text{insert } f) \circ \alpha g \equiv (\text{insert } f \circ \alpha g)$
 [This implies $(\text{insert } f) : \langle x_1 \rangle \rightarrow f : \langle x_1, \text{rid } f \rangle$]

Some more identities

$$IP = (INSERT, PLUS) \circ (EE, TIMES)$$

$$(INSERT, f) \circ [g, ROW \circ h] = f \circ [g, (INSERT, f) \circ h]$$

$$(EE, f) \circ [[g, ROW \circ h], [j, ROW \circ k]] = [f \circ [g, h], ROW \circ (EE, f) \circ [h, k]]$$

$$IP \circ [1 \circ 1, (AA, 1) \circ 2]$$

$$= (INSERT, +) \circ (EE, x) \circ [[1 \circ 1 \circ 1, ROW \circ TAIL \circ 1 \circ 1], [1 \circ 1 \circ 2, ROW \circ (AA, 1) \circ TAIL \circ 2]]$$

$$= (INSERT, +) \circ [x \circ [1 \circ 1 \circ 1, 1 \circ 1 \circ 2], ROW \circ (EE, x) \circ [TAIL \circ 1 \circ 1, (AA, 1) \circ TAIL \circ 2]]$$

$$= + \circ [\underbrace{x \circ [1 \circ 1 \circ 1, 1 \circ 1 \circ 2]}_{\text{Multiply corresponding elements}}, \underbrace{(INSERT, +) \circ (EE, x) \circ [TAIL \circ 1 \circ 1, (AA, 1) \circ TAIL \circ 2]}_{\text{Scan across first row of left operand, Scan down first column of right operand}}]$$

Multiply corresponding elements

Scan across first row of left operand Scan down first column of right operand

Add the products

$$\begin{bmatrix} (EE, f) \circ (C, \emptyset) \\ (INSERT, f) \circ [g] \end{bmatrix}$$

$$= (C, \emptyset)$$

$$= g$$

{ provided g doesn't return empty row }
 ~~$g \neq (AA, ROW \circ f) \circ \emptyset$~~

also:

$$(INSERT, f) \circ [g, h, ROW \circ k] = (INSERT, f) \circ [f \circ [g, h], ROW \circ k]$$

$$(INSERT, f) \circ [g] = g \quad \{ \text{provided } g \text{ doesn't produce empty row} \}$$

or:

$$(INSERT, f, i) \circ [g, ROW \circ h] = \cancel{INSERT} f \circ [g, (INSERT, f, i) \circ h]$$

$$(INSERT, f, i) \circ (C, \emptyset) = (C, i)$$

$\underbrace{\quad}_{\text{identity for } f}$

30 Oct 74
John's stuff

Syntax

1. $A \subseteq E$
2. Let $R_i = E$, $i \geq 1$; then $e \in E \ \& \ r \in R_i \rightarrow e, r \in R_{i+1}$
3. $r \in R_i \rightarrow \langle r \rangle \in E$ (sequence)
4. $r \in R_i \rightarrow \llbracket r \rrbracket \in E$ (instruction)
5. $e, e' \in E \rightarrow \{e, e'\} \in E$ (world pair, e in environment e')
6. $e, e' \in E \rightarrow (e, e') \in E$ (application)

Semantics

$$\mu \{e, s\} \equiv \mu_s e \equiv e \in A \rightarrow e$$

$$e = \langle e_1, \dots, e_n \rangle \rightarrow \langle \mu_s e_1, \dots, \mu_s e_n \rangle$$

$$e = \llbracket e_1, \dots, e_n \rrbracket \rightarrow e$$

$$e = \{e' \ s' \} \rightarrow \mu \{e' \ \{s' \ s'\} \}$$

$$e = (x \ y) \rightarrow$$

$$x = 'x' \rightarrow \mu \{ (y, s), s \}$$

$$x \in A \rightarrow \mu \{ [p x] \mu_s y, s \}$$

$$x = \langle e_1, \dots, e_n \rangle \xrightarrow{(\text{meta comp})}$$

$$x = \llbracket \dots \rrbracket \rightarrow \sigma$$

$$T \rightarrow \mu_s (\mu_s x, y)$$

↙ "next state"
function

$$\Delta \{e, s\} \equiv \mu_s e = \langle \text{CTXT}, c \rangle \rightarrow c$$

$$\mu_s e = \llbracket e_1 \rrbracket \rightarrow \Delta \{e_1, s\}$$

$$\mu_s e = \llbracket e_1, \dots, e_n \rrbracket \rightarrow \Delta \{ \llbracket e_2 \dots e_n \rrbracket, \Delta \{e_1, s\} \}$$

$$T \rightarrow \Delta$$

↙ "output"
function

$$\sigma \{e, s\} \equiv \mu_s e = \llbracket e_1 \rrbracket \rightarrow \sigma \{e_1, s\}$$

$$\mu_s e = \llbracket e_1, \dots, e_n \rrbracket \rightarrow \text{concat} [\sigma \{e_1, s\}, \{ \llbracket e_2 \dots e_n \rrbracket, \Delta \{e_1, s\} \}]$$

$$T \rightarrow \mu \{e, s\}$$

10 DECEMBER 1974

I think this is the version of Red implemented in the Lisp program.

[It also has provisions for quoted expressions as in my 15 May 74 proposal]

RED WITH ROWS AND CONTEXTS

Syntax

- 1) $E_0 = \{\emptyset\}$
- 2) $A \cup \{*\} \subseteq E_1$
- 3) $r_i \in E_i \ \& \ r_j \in E_j \rightarrow r_i, r_j \in E_{i+j}$
- 4) $r \in E_i \rightarrow \langle r \rangle \in E_1$
- 5) $x \in E_i \ \& \ y \in E_j \rightarrow (x:y) \in E_1$

$a \in A$
$e \in E_i$
$r, s, \dots \in E_i$

Semantics

$$\mu_s \emptyset = \emptyset \quad (\text{the empty row})$$

$$\mu_s * = s \quad (\text{fetch current context})$$

$$\mu_s a = a \quad (\text{atom})$$

$$\mu_s e, r = \mu_s e, \mu_t r \text{ where } t = \Delta_s e \quad (\text{row})$$

$$\mu_s \langle r \rangle = \langle \mu_s r \rangle \quad (\text{sequence})$$

$$\mu_s (x:y) = \mu_s \lambda_s \overset{\mu_s \mu_s}{(x:y)} \quad (\text{application})$$

$$\Delta_s e, r = \Delta_t r \text{ where } t = \Delta_s e$$

$$\Delta_s (**:y) = \mu_s y \quad [\text{context changer}]$$

$$\Delta_s (x:y) = \Delta_s \lambda_s \overset{\mu_s \mu_s}{(x:y)}$$

$$\Delta_s e = s \quad [\text{i.e. } \emptyset, \text{ atom, sequence}]$$

$$\lambda_s (**:y) = \emptyset \quad [\text{context changer}]$$

$$\lambda_s (\langle \text{prim}, r \rangle : y) = \phi_r y \quad [\text{primitive}]$$

$$\lambda_s (a:y) = (\phi_{\text{fetch}} \langle a, s \rangle : y) \quad [\text{defined operator}]$$

$$\lambda_s (\langle e, r \rangle : y) = (e : \langle \langle e, r \rangle, y \rangle, s) \quad [\text{meta composition}]$$

$$\lambda_s (e, r : y) = \lambda_s (\langle e, r \rangle : y)$$

$$\phi_{\text{fetch}} e, \langle \text{cell}, e, y \rangle, r = y$$

$$\phi_{\text{fetch}} e, \langle x, r \rangle = \phi_{\text{fetch}} e, r$$

$$\phi_{\text{fetch}} e, \emptyset = \text{NOT-FOUND}$$

$$\phi_{\text{fetch}} x = \text{BAD-ARG}$$

How about $\lambda_s (\text{PRIM}, r : y)$?

[

Red Evaluators

1. Given $\mu = \mu_0$ as input, repeatedly set $\mu_{n+1} = \mu_n$ with an innermost application removed" until some μ_k ($k \geq 0$) is constant. μ_k is result.
(flag nonconstant exprs in input routine; flag exprs containing UNSPEC's)
2. Similar to 1, but the $\mu_1, \mu_2, \dots, \mu_{k-1}$ are never actually generated. A recursive program traverses μ_0 evaluating applications (and recursively traversing resulting ~~exprs~~ ^{exprs}) to produce a constant result.
3. Top down: pass over ~~program~~ expression translating it into a generator [of the value of the expr] then execute generator to produce stream of values.
So $\langle IP, (v, w) \rangle \equiv \langle (0 \text{ (INSERT, +)}, (EE, *)), (v, w) \rangle$
is translated to $\boxed{\text{result} := 0;}$
for $i := 1$ to $\text{length}(v)$ do
 $\text{result} := \text{result} + v(i) * w(i);$

17 December 1974

RED Interpreter

[Method used by Lisp-coded Red interpreter]
[see the EXEC function in the REDEV file]

1. Introduction

The method used here is to characterize the state of the interpreter as a tuple $\langle P, T, S, M \rangle$ whose components have the following purposes:

P ("Program")	A sequence of subexpressions to evaluate (also contains special markers).
T ("Temporaries")	A sequence of values to be used as operands by future actions.
S ("Store")	A sequence of stores, each corresponding to a different level of nesting.
M ("Error Mode")	A boolean value, <u>false</u> if no error has occurred in the evaluation.

The interpreter is initialized by setting P to the expression to be evaluated, T to Φ , S to the initial store, and M to false. The interpreter repeatedly transforms $\langle P, T, S, M \rangle$ using a set of rewriting rules given below. This process terminates when (if) P becomes Φ ; then S contains the new store.

2. Notation

$a \in \text{Atoms}$	} + variables	} literals	$\text{PRIM}, \text{FETCH}, ** \in \text{Atoms}$
$c, d \in \text{Constant Items (length=1)}$			$* \in E_1$
$T, S, t, u \in \text{Constant Rows (length} \geq 0)$			$\Phi \in C_0$
$e \in \text{General Items (length=1)}$			
$P, r \in \text{General Rows (length} \geq 0)$			$\text{mkseq}, \text{conc}, \text{ap}$ are special markers not in RED programs

3. Rewriting Rules

The general format is:

name: pattern $\Rightarrow$ result

17 December 1974 RED Interpreter: Rewriting Rules

[START]	[<u>r</u> is expression to evaluate; <u>c</u> is current context]	$\Rightarrow$	$\langle r \rangle$	$\emptyset$	$\langle c \rangle$		
NIL:	$\langle \emptyset \rangle, P$	T	S	$\Rightarrow$	P	$\langle \emptyset \rangle, T$	S
FETCH:	$\langle * \rangle, P$	T	c, S	$\Rightarrow$	P	$\langle c \rangle, T$	c, S
ATOM:	$\langle a \rangle, P$	T	S	$\Rightarrow$	P	$\langle a \rangle, T$	S
SEQ:	$\langle \langle r \rangle \rangle, P$	T	c, S	$\Rightarrow$	$\langle r \rangle, \text{mkseq}, P$	T	c, c, S
MKSEQ:	mkseq, P	$\langle t \rangle, T$	c, S	$\Rightarrow$	P	$\langle \langle t \rangle \rangle, T$	S
ROW:	$\langle e, r \rangle, P$	T	S	$\Rightarrow$	$\langle e \rangle, \langle r \rangle, \text{conc}, P$	T	S
CONC:	conc, P	$\langle t_1 \rangle, \langle t_2 \rangle, T$	S	$\Rightarrow$	P	$\langle t_2, t_1 \rangle, T$	S
APPL:	$\langle \langle r_1 : r_2 \rangle \rangle, P$	T	S	$\Rightarrow$	$\langle r_1 \rangle, \langle r_2 \rangle, \text{ap}, P$	T	S
STORE:	ap, P	$\langle c \rangle, \langle ** \rangle, T$	d, S	$\Rightarrow$	P	$\langle \emptyset \rangle, T$	c, S
PRIM:	ap, P	$\langle \langle t \rangle \rangle, \langle \langle \text{PRIM}, u \rangle \rangle, T$	S	$\Rightarrow$	$\langle \Phi_u t \rangle, P$	T	S
DEFD:	ap, P	$\langle \langle t \rangle \rangle, \langle \langle a \rangle \rangle, T$	c, S	$\Rightarrow$	ap, P	$\langle \langle t \rangle \rangle, \langle \langle \Phi_{\text{FETCH}} a, c \rangle \rangle, T$	c, S
[RC:	ap, P	$\langle \langle t \rangle \rangle, \langle \langle c, u \rangle \rangle, T$	S	$\Rightarrow$	ap, ap, P	$\langle \langle t \rangle \rangle, \langle \langle u \rangle \rangle, \langle \langle c \rangle \rangle, T$	S]
MC:	ap, P	$\langle \langle t \rangle \rangle, \langle \langle c, u \rangle \rangle, T$	d, S	$\Rightarrow$	ap, P	$\langle \langle \langle c, u \rangle, t \rangle, d \rangle, \langle \langle c \rangle \rangle, T$	d, S
[STOP]	$\emptyset$	$\langle t \rangle$	c	$\Rightarrow$	[<u>t</u> is result; <u>c</u> is new context]		

error mode = false

error mode = false

20 December 79

RED WITH CONTEXTS [BUT NO GENERAL ROWS]

Syntax

$$A \cup \{*\} \subset E_1$$

~~XXXXXX~~ $\{\phi \in A ? \text{ or is it special, like } * ?\}$

$$e \in E_1 \text{ \& } r \in E_i \rightarrow e, r \in E_{i+\delta} \text{ where } \delta = \begin{cases} 0 & \text{if } e = \phi \text{ or } r = \phi \\ 1 & \text{otherwise} \end{cases}$$

$$r \in E_i \rightarrow \langle r \rangle \in E_1$$

$$x, y \in E_1 \rightarrow (x:y) \in E_1$$

$$\begin{aligned} a &\in A \\ x, y, e &\in E_1 \\ r &\in E_i \end{aligned}$$

ϕ ["nil"] is the empty ~~sequence~~ row; all the nils in a row disappear (except one, if the row is all nils).
 $\langle \phi \rangle$ ["null"] is the empty sequence, and is a legitimate value (ϕ is returned by context changer, nothing else?)

Semantics

$$\mu_s \phi = \phi$$

$$\mu_s * = s$$

$$\mu_s a = a$$

$$\mu_s e, r = \mu_s e, \mu_t r \text{ where } t = \Delta_s e$$

$$\mu_s \langle r \rangle = \langle \mu_s r \rangle$$

$$\mu_s (x:y) = \mu_s \lambda_s (x: y)$$

$$\Delta_s (**:y) = \mu_s y$$

$$\Delta_s (***: \langle p \rangle) = \Delta_s p$$

$$(\Delta_s (***:(x:y))) = \Delta_s y$$

$$\Delta_s e, r = \Delta_t r \text{ where } t = \Delta_s e$$

$$\Delta_s (x:y) = \Delta_s \lambda_s (x: y)$$

$$\Delta_s e = s$$

$$\lambda_s (x:y) \equiv x \neq \mu_s x \rightarrow \lambda_s (\mu_s x : y)$$

$$x = ** \rightarrow \phi$$

$$x = *** \rightarrow \mu_s y$$

$$x \in A \rightarrow (\varphi_{\text{FETCH}} \langle x, s \rangle : y)$$

$$x = \langle \text{PRIM}, e \rangle \rightarrow \varphi_e y$$

$$x = \langle e, r \rangle \rightarrow (e: \langle x, y, s \rangle)$$

$$\varphi_{\text{FETCH}} \langle e, \langle \text{CELL}, e, y \rangle, r \rangle = y$$

$$\varphi_{\text{FETCH}} \langle e, \langle x, r \rangle \rangle = \varphi_{\text{FETCH}} \langle e, r \rangle$$

$$\varphi_{\text{FETCH}} \langle e, \langle \phi \rangle \rangle = \text{NOT-FOUND}$$

$$\varphi_{\text{FETCH}} e = \text{BAD ARG}$$

30 December 74

Proper Expressions

$$R_0 = \{\emptyset\}$$

$$A \cup \{*\} \subseteq I$$

$$e \in I \ \& \ r \in R_i \rightarrow e, r \in R_{i+1}$$

$$r \in R_i \rightarrow \langle r \rangle \in I$$

$$r \in R_i \cup I \ \& \ s \in R_j \cup I \rightarrow (r:s) \in I$$

$$E = I \cup \bigcup_{i \geq 0} R_i$$

$$(1:x) = x \in A \cup R_0 \rightarrow ?$$

$$x = e, r \rightarrow e$$

$$x = \langle \emptyset \rangle \rightarrow ?$$

$$x = \langle e, r \rangle \rightarrow e$$

$$(TAIL:x) = x \in A \cup R_0 \rightarrow ?$$

$$x = e, r \rightarrow r$$

$$x = \langle \emptyset \rangle \rightarrow ?$$

$$x = \langle e, r \rangle \rightarrow \langle r \rangle$$

$$(FETCH:r) = r \in I \cup R_0 \rightarrow ?$$

$$r = e, \emptyset \rightarrow ? \text{ NOT FOUND}$$

$$r = e, \langle \text{cell}, e, v \rangle, r' \rightarrow v$$

$$r = e, v \rightarrow (FETCH:e, (TAIL:v))$$

Extension to syntax

$$1. \ e \in I \rightarrow \langle e \rangle \equiv \langle e, \emptyset \rangle$$

$$2. \ \text{Drop final } \emptyset \text{ of rows longer than 1}$$

$$3. \ \text{ROW } r, s \equiv \begin{matrix} \text{if } r = \emptyset \text{ then } s \\ \text{else row head } r, (\text{row tail } r, s) \end{matrix}$$

$$r, s \equiv \text{row } r, s$$

~ 10 Jan 1974/5

Approximate "concrete"

[Syntax accepted by the
Lisp-coded Red interpreter]

expr : atom | quoted | seq | row | appl

atom : pseudo_char⁺ | digit⁺

quoted : ' expr '

seq : [expr { delim expr }^{*}]

row : expr { delim expr }⁺ | nil

appl : (appl : appl)

~~nil~~

delim : { blank | comma }⁺

pseudochar: (any character other than "e", "a", "<", ">", "(", ")", ":", ":", blank,
comma_{digit}) | ' (any character)

delims not necessary between "right-bracket left-bracket" pair.

↑ number ↑ "symbolic" atom
[123 ≠ '1'2'3]
 |
 1'2'3
 |
 1'2'3
 |
 1'2'3
 |
 etc.]

→ [actually ' is = and : is # in the EBCDIC implementation...]

Jan 29 '75

Syntax

- 1) $R_0 = \{\Delta\}$ *the nil row*
- 2) $e \in E \rightarrow (*GF, e) \in GF \subseteq E$
- 3) $r \in R_i \ \& \ s \in R_j \rightarrow r, s \in R_{i+j} \quad i, j \geq 0$
- 4) $e \in E \rightarrow \omega e \in O \subseteq R_1$
- 5) $r \in R_i \rightarrow \langle r \rangle \in E \quad i \geq 0$
- 6) $e \in E \ \& \ f \in E \rightarrow (e:f) \in AP \subseteq E$
- 7) $A \subseteq E \subseteq R_1$

Semantics[let $e, f, g \ \& \ h \in E$; $p, r \in R_i$]

$$\begin{aligned} \mu_g x &\equiv x \in A \cup GF \cup R_0 \rightarrow x \\ x = \Delta, r \quad \text{or} \quad x = r, \Delta &\rightarrow \mu_g r \\ x = p, g &\rightarrow \mu_g p, \mu_h g \quad \text{where } h = \Delta_g p \\ x = \langle r \rangle &\rightarrow \langle \mu_g r \rangle \\ x \in O \cup AP &\rightarrow \mu_g \lambda_g x \end{aligned}$$

$$\begin{aligned} \lambda_g x &\equiv x = \omega e \rightarrow \\ &\quad e \in A \cup GF \rightarrow ? \quad \text{the unspecified value} \\ &\quad e = \langle r \rangle \rightarrow r \\ &\quad e \in AP \rightarrow \omega \lambda_g e \\ x = (e:f) &\rightarrow \\ &\quad e \neq \mu_g e \rightarrow (\mu_g e : f) \\ &\quad e = \langle *prim, h \rangle \rightarrow \phi_h \mu_g f \\ &\quad e \in A \rightarrow (\phi_{\text{FETCH}} \langle e, g \rangle : f) \\ &\quad e = \langle h, r \rangle \rightarrow (h : \langle e, f, g \rangle) \\ &\quad e \in GF \rightarrow \phi_g \mu_g f \end{aligned}$$

$$\begin{aligned} \Delta_g x &\equiv x = (\langle *prim, ** \rangle : e) \rightarrow \mu_g e \quad P_{**} x = \Delta \text{ presumably} \\ &\quad x \in O \cup AP \rightarrow \Delta_g \lambda_g x \quad \text{[the only primitive returning a row]} \\ &\quad x = p, g \rightarrow \Delta_h g \quad \text{where } h = \Delta_g p \\ &\quad T \rightarrow g \end{aligned}$$

John:

1. φ_{xx} is only primitive returning a row

2. rule for $\mu g p, q$ doesn't eliminate

Δ produced by evaluation of non constant
 p or q (e.g. context changer)

~ February 1975

CRED

1. Syntax

$$E = \text{CONST} + \text{SEQ} + \text{AP}$$

$$\text{AP} = E \times E$$

$$\text{SEQ} = E^*$$

$$\text{CONST} = \{0, 1, T, F, \perp, \text{DBL}, \text{HEAD}, \text{ROTL}, \dots, \text{COMP}, \text{CN}, \text{M}, \dots\} \cup \text{ATOM}$$

2. Semantics

$$D = N + \text{Tr} + \text{At} + D^* + [D \rightarrow D]$$

N = natural numbers
 Tr = truth values
 At = a set of "symbolic" atoms

$$\mu: E \rightarrow D$$

[no environment or store necessary since Red is applicative and variable free]

$$\mu[a] = \hat{a} \text{ for all } [a] \in \text{ATOM} \quad \sim: \text{ATOM} \rightarrow \text{At} \quad \begin{matrix} \text{eD} \rightarrow \text{D}^2 \\ \text{eD} \end{matrix}$$

$$\mu[(e_1, e_2)] = \mu[e_1](\mu[e_2]) \text{ or rather } \text{apply}(\mu[e_1], \mu[e_2]) \in D^2$$

$$\mu[\langle e_1, \dots, e_n \rangle] = \langle \mu[e_1], \dots, \mu[e_n] \rangle$$

$$\mu[0] = 0; \mu[1] = 1; \mu[T] = \text{true}; \mu[F] = \text{false}; \mu[\perp] = \perp$$

$$\mu[\text{DBL}] = \lambda x. \langle x, x \rangle; \mu[\text{HEAD}] = \lambda \langle x_1, \dots, x_n \rangle. x_1; \mu[\text{ROTL}] = \lambda \langle x_1, x_2, \dots, x_n \rangle. \langle x_2, \dots, x_n, x_1 \rangle$$

...

$$\mu[\text{COMP}] = \lambda \langle f_1, \dots, f_n \rangle. \lambda x. f_1 \dots f_n x$$

$$\mu[\text{CN}] = \lambda \langle p, f, g \rangle. \lambda x. p x \rightarrow f x \mid g x$$

$$\mu[M] = \lambda m. \lambda f. \lambda p. \lambda x. f \langle m f, p, x \rangle$$

$$\mu[\text{NF}] = \lambda f l. \lambda x. \text{null } f l \rightarrow \text{nil} \mid h f l x: \text{NF}(t \ell) x$$

$$\text{so } M F P X \Rightarrow F \langle M F, P, X \rangle$$

$$\text{i.e. } F(: (M F)) (: P (: X \text{ NIL}))$$

Reduction property for CRED: If a subexpression e_i of e is replaced by an expression e_i' with $\mu[e_i] = \mu[e_i']$, then the meaning of e is not changed.

β for RED

$$\mu a = a$$

$$\mu \langle e_1, \dots, e_n \rangle = \langle \mu e_1, \dots, \mu e_n \rangle$$

$$\mu(e_1; e_2) = [\beta e_1] e_2 \quad \text{where } \beta: C \rightarrow [C \rightarrow C]$$

$$\beta 1 \langle x_1, \dots, x_n \rangle = x_1$$

$$\beta \text{ APPLY } \langle x_1, x_2 \rangle = [\beta x_1] x_2$$

$$\beta \langle \text{NF}, f_1, \dots, f_n \rangle = \lambda x. \langle [\beta f_1] x, \dots, [\beta f_n] x \rangle$$

$\vdots$

$$\beta \langle \text{COMP}, f_1, \dots, f_n \rangle = [\beta f_1] \circ \dots \circ [\beta f_n]$$

$\vdots$

$$\beta \langle x_1, x_2, \dots, x_n \rangle = \lambda y. [\beta x_1] \langle \langle x_1, x_2, \dots, x_n \rangle, y \rangle$$

$$\beta \text{ COMP } \langle \langle \text{COMP}, f_1, \dots, f_n \rangle x \rangle = ([\beta f_1] \circ \dots \circ [\beta f_n]) x$$

$\beta : \rho :: \tau : \mu$

$\mu: E \xrightarrow{\text{partial}} C$
C is fixed points of μ

$\tau: E \xrightarrow{\text{total}} E$
C is fixed points of τ

μ = "transitive closure" of τ =



$$\beta = \lambda c. \mu(\rho c) = \lambda c. \beta \mu(\rho c) = S(K\beta \mu))\rho$$

$$\text{ap}[\text{gd}] \rightarrow [\rho c]d$$

$$* \sigma \tau f x \rightarrow \sigma f(\tau f x)$$

Spring 1975

$$1. F = (\text{zerop} \rightarrow (\text{constant } 1); \\ (\text{onep} \rightarrow (\text{constant } 1); \\ + \circ [F \text{ decrease}^2, F \text{ decrease}]))$$

given

$$2. G = [F \text{ increase}, F]$$

definition

$$3. (G:0) = \langle (F:1), (F:0) \rangle = \langle 1, 1 \rangle$$

simplification

$$4. G \circ \text{increase} = [F \text{ increase}, F] \circ \text{increase} = [F \text{ increase}^2, F \text{ increase}]$$

right association [worry about $\perp$]

$$5. = [+ \circ [F \text{ decrease}^2, F \text{ decrease}] \circ \text{increase}^2, \\ + \circ [F \text{ decrease}^2, F \text{ decrease}] \circ \text{increase}]$$

$$= [+ \circ [F, F \text{ increase}], \text{~~the first part~~}, F \text{ increase}]$$

$$= [+ \circ [F, F \text{ increase}], \text{2nd}] \circ G$$

1st prn 4 Dec 2015

$$5. F \circ \text{increase}^2 = + \circ [F \text{ decrease}^2, F \text{ decrease}] \circ \text{increase}^2$$

$$= + \circ [F, F \circ \text{increase}] = + \circ G$$

$$F = (\text{zerop} \rightarrow (\text{constant } 1); \\ (\text{onep} \rightarrow (\text{constant } 1); \\ + \circ G \text{ decrease}^2)) .$$

$$G = (\text{zerop} \rightarrow \langle 1, 1 \rangle; \\ [+ \circ \text{2nd}] \circ G \text{ decrease} .$$

1st

$$[f, g] \circ h \stackrel{?}{=} [f \circ h, g \circ h] \neq$$

function transformations (without using ROW)

$$MM = \langle AA, \langle AA, IP \rangle \circ DISTL \rangle \circ DISTR \circ [1, TRANS \circ 2]$$

$$= \langle AA, \langle AA, IP \rangle \circ DISTL \rangle \circ DISTR \circ [UN \circ [1 \circ 1, TAIL \circ 1], TRANS \circ 2] \quad \begin{matrix} 1 = \phi 1; \phi = UN \circ [1, T] \\ [f, g] \circ h = [f \circ h, g \circ h] \end{matrix}$$

$$= \langle AA, \langle AA, IP \rangle \circ DISTL \rangle \circ UN \circ [[1 \circ 1, TRANS \circ 2], DISTR \circ [TAIL \circ 1, TRANS \circ 2]] \quad \begin{matrix} DISTR \\ DISTR \end{matrix}$$

$$= UN \circ [\langle AA, IP \rangle \circ DISTL \circ [1 \circ 1, TRANS \circ 2], \langle AA, \langle AA, IP \rangle \circ DISTL \rangle \circ DISTR \circ [TAIL \circ 1, TRANS \circ 2]] \quad AA$$

$$= UN \circ [\langle AA, IP \rangle \circ DISTL \circ [1 \circ 1, UN \circ [\langle AA, \overset{02}{1} \overset{02}{TRANS} \circ \langle AA, TAIL \rangle \overset{02}{1}]]], rest] \quad \begin{matrix} TRANS, assoc [] \end{matrix}$$

$$= UN \circ [\langle AA, IP \rangle \circ UN \circ [[1 \circ 1, \langle AA, 1 \rangle \circ 2], DISTL \circ [\overset{1 \circ 1}{AA}, TRANS \circ \langle AA, TAIL \rangle \circ 2]], rest] \quad \begin{matrix} DISTL \end{matrix}$$

$$\leftarrow \cancel{UN \circ [UN \circ [IP \circ [1 \circ 1, \langle AA, IP \rangle \circ \langle AA, 1 \rangle \circ 2], DISTL \circ [1 \circ 1, TRANS \circ \langle AA, TAIL \rangle \circ 2]], rest]} \quad \begin{matrix} AA \\ AA \end{matrix}$$

$$= UN \circ [UN \circ [IP \circ [1 \circ 1, \langle AA, 1 \rangle \circ 2], \langle AA, IP \rangle \circ DISTL \circ [1 \circ 1, TRANS \circ \langle AA, TAIL \rangle \circ 2]], rest] \quad \begin{matrix} AA \end{matrix}$$

⋮

$$= UN \circ [UN \circ [\overset{row}{IP} \circ [1 \circ 1, \langle AA, 1 \rangle \circ 2], \overset{column}{\langle AA, IP \rangle \circ DISTL \circ [1 \circ 1, TRANS \circ \langle AA, TAIL \rangle \circ 2]}], \langle AA, \langle AA, IP \rangle \circ DISTL \rangle \circ DISTR \circ [TAIL \circ 1, TRANS \circ 2]]$$

first elt of first row
rest of first row
remaining rows

$$(AB)_{ij} = \sum_k A_{ik} B_{kj}$$

Note that writing out the complete expression for a given NM would be an arduous task!

~Oct 76

programming language semantics

whole entity programming (like APL, [Lisp], Relational Algebra)

Red

elimination of regular composition as ~~the~~ basic transition rule

Church-Rosser property [mathematical semantics: fixed point theory]

Interpreters [CRED] . . . → order of evaluation . . . → reduction properties

Language extensions: quoted expressions, storage, i/o, "system" (debugging, . . .)

Data flow (Red + storage, i/o)

System R : RSS : recovery

"Software": attempted to convert CIT's PASCAL to VM; constructed interactive debuggers for Snobol 4

+ transformations on Red expressions & dataflow programs

"consultations" w/ D. Redell

B. Lindsay

S. Chernicoff

S. Ziller

[proved Church-Rosser property

fixed point theory
closed applicative languages, "reductions"

suggested eliminating regular composition from basic framework

[pointed out necessity for system aspects: i/o, storage] exceptions

criticized various later Red designs, proposed quoted expressions

wrote several Red interpreters in Lisp, McG

adapted John's metacomposition to McG combinator

reading list:

referenced two articles

familiarized self w/ various prog sys & lang

worked on Bayer's trans. closure algorithm

McCarthy

Burge

Leavenworth

Landin

Scott, Strachey

Reynolds

Morris

Manna (fixed point semantics)

Kosinski (DFA)

Dennis (DFPL, modularity)

Burstall (struct. ind.)

Given a functional programming system of sufficient expressibility,
what next?

proving properties about functions

fixed point induction

operator identities

+ op sys ?

Oct 78

PROGRAMMING LANGUAGE SEMANTICS

HYPOTHESIS: THE "SOFTWARE PROBLEM" (HIGH COSTS,
SLIPPED DEADLINES, POOR RELIABILITY) IS IN PART
CAUSED BY INADEQUACIES OF CURRENT
PROGRAMMING LANGUAGES:

HIGHLY COMPLEX

HARD TO LEARN

HARD TO IMPLEMENT

DEFINITIONS AMBIGUOUS, INCOMPLETE

"LOW LEVEL", MACHINE ORIENTATION

GO-TO

ASSIGNMENT

EXPOSURE OF ALTERNATE DATA REPRESENTATIONS

e.g. DEC vs. BIN

MULTIPLE STORAGE ALLOCATION POLICIES

e.g. AUTO, STATIC, BASED-ALLOCATE

EXPOSURE OF IMPLEMENTATION STORAGE MODEL

e.g. DEFINED

INADEQUACIES OF CURRENT LANGUAGES, CONT. :

LACKING IN DEFINITIONAL CAPABILITY (?)

FORTRAN : SUBPROGRAMS

PL/I : STRUCTURED DATA (ARRAY OF ARRAYS)
MACROS

CLU : EXTENDED DATA TYPES

CONCLUSION: CURRENT LANGUAGES SEEM MOSTLY
CONCERNED WITH HOW TO COMPUTE AN
ANSWER, RATHER THAN WHAT THAT ANSWER
SHOULD BE.

GOAL: TO DESIGN A LANGUAGE WITH THE
FOLLOWING PROPERTIES:

"HIGH LEVEL" / DESCRIPTIVE OF ENDS, NOT MEANS
SIMPLE, RIGOROUS SEMANTICS
NONTRIVIAL DEFINITIONAL CAPABILITY

IDEA: PURSUE THE NOTION OF APPLICATIVE
LANGUAGES, WHEREIN THE MEANING OF AN
EXPRESSION IS ALWAYS ITS VALUE, NEVER
A "SIDE-EFFECT."

EXAMPLE:

<pre>SUM=0; DO I=1 TO N; SUM=SUM+A(I); END;</pre>	<pre>SUM ← +/A</pre>
PL/I	APL