Paul McJones

Date:
(Div. & location mail address):
Dept. & Bldg:
& Tel. Ext.:

August 29, 1972
T.J. Watson Rhsearch Center
Yorktown Heights, N.Y.
440 - 801
862 - 2464

IBM

965-IBM-04

Subject: Annotated Listings of RED Systems.

Reference:

To: File

The attached sheets are annotated listings of the complete RED system. The listings represent the information in the three symbolic files REDPRM, REDDEL, and REDMOD. REDPRM contains the LISP lambda expressions that define the primitive RED operators (pp. 61-64, RJ-1010). REDDEL contains the internal representations of the RED expressions that define the operators on pp. 77-79 of RJ-1010. REDMOD is the file that contains the LISP definitions for the functions needed to build the RED system.

A RED system may be built from these symbolic files by entering LISP and executing the following three LISP doublets.

```
RDF  (REDMOD LSTMOD)
RDF  (REDDEL LSTDEL)
RDF  (REDPRM LSTPRM)
```

This causes LISP to take as input the information in these three files. The output will be generated as the files LSTxxx. To produce the output directly on the printer NIL should be used as the second argument of RDF.

P. D. Summers.

PDS/cah

965-IBM-04

For Explanation see the memo
"Adding Primitives to the RED System"

```
COMP360 (((NPAIRP (LAMBDA (X) (NOT (EQUAL (LENGTH X) 3))))))

(NPAIRP
   (LAMBDA (X) (NOT (EQUAL (LENGTH X) 3))))
VALUE =  (NPAIRP)

MAKEPROP (REV MODE PRIM)
VALUE =  REV

MAKEPROP (REV DEFN (LAMBDA (X) (COND ((NPAIRP X) REDQUAD) (T (LIST (CAR X) (CADDR X) (CADR X))))))
VALUE =  REV

MAKEPROP (ASSOC MODE PRIM)
VALUE =  ASSOC

MAKEPROP (ASSOC DEFN (LAMBDA (X) (COND ((NPAIRP X) REDQUAD) ((NPAIRP (CADDR X)) REDQUAD) (T (LIST (CAR X) (LIST
(CAR (CADDR X)) (CADR X) (CADR (CADDR X))) (CADDR (CADDR X)))))))
VALUE =  ASSOC

MAKEPROP (CAR MODE PRIM)
VALUE =  CAR

MAKEPROP (CAR DEFN (LAMBDA (X) (COND ((NPAIRP X) REDQUAD) (T (CADR X)))))
VALUE =  CAR

MAKEPROP (APPLY MODE PRIM)
VALUE =  APPLY

MAKEPROP (APPLY DEFN (LAMBDA (X) (COND ((NPAIRP X) REDQUAD) ((NPAIRP (CADR X)) REDQUAD) (T (LIST (CAR X) (REDUCE
 (CONS (QUOTE I) (CONS (CADR (CADR X)) (CADDR (CADR X)))) (CADDR X))))))
VALUE =  APPLY

MAKEPROP (COND MODE PRIM)
VALUE =  COND

MAKEPROP (COND DEFN (LAMBDA (X) (COND ((NPAIRP X) REDQUAD) ((NPAIRP (CADDR X)) REDQUAD) ((EQ (QUOTE T) (CADR X)
) (CADR (CADDR X))) ((EQ (QUOTE F) (CADR X)) (CADDR (CADDR X))) (T REDQUAD)))))
VALUE =  COND

MAKEPROP (EQAT MODE PRIM)
VALUE =  EQAT

MAKEPROP (EQAT DEFN (LAMBDA (X) (COND ((NPAIRP X) REDQUAD) ((NOT (ATOM (CADR X))) (QUOTE F)) ((NOT (ATOM (CADDR
X))) (QUOTE F)) ((EQ REDQUAD (CADR X)) REDQUAD) ((EQ REDQUAD (CADDR X)) REDQUAD) ((EQ (CADR X) (CADDR X)) (QUOTE
 T)) (T (QUOTE F))))))
VALUE =  EQAT

MAKEPROP (UNION MODE PRIM)
VALUE =  UNION

MAKEPROP (UNION DEFN (LAMBDA (X) (COND ((NPAIRP X) REDQUAD) (T (CONS (CAR X) (CONS (CADR X) (CDR (CADDR X))))))
))
VALUE =  UNION
```

```
MAKEPROP (DBL MODE PRIM)
VALUE =  DBL

MAKEPROP (DBL DEFN (LAMBDA (X) (LIST NIL X X)))
VALUE =  DBL

MAKEPROP (SEP MODE PRIM)
VALUE =  SEP

MAKEPROP (SEP DEFN (LAMBDA (X) (COND ((ATOM X) REDQUAD) ((ATOM (CDR X)) REDQUAD) ((NULL (CDDR X)) (LIST (CAR X)
 (CADR X) NIL)) (T (LIST (CAR X) (CADR X) (CONS (CAR X) (CDDR X)))))))
VALUE =  SEP

MAKEPROP (NIL MODE PRIM)
VALUE =  NIL

MAKEPROP (NIL DEFN (LAMBDA (X) X))
VALUE =  NIL

MAKEPROP (QUAD MODE PRIM)
VALUE =  QUAD

MAKEPROP (QUAD DEFN (LAMBDA (X) (COND ((EQ X REDQUAD) (QUOTE T)) (T (QUOTE F)))))
VALUE =  QUAD

MAKEPROP (PREFIX MODE PRIM)
VALUE =  PREFIX

MAKEPROP (PREFIX DEFN (LAMBDA (X) (COND ((NPAIRP X) REDQUAD) (T (CONS (CADR X) (CDR (CADDR X)))))))
VALUE =  PREFIX

MAKEPROP (UNPREFIX MODE PRIM)
VALUE =  UNPREFIX

MAKEPROP (UNPREFIX DEFN (LAMBDA (X) (COND ((ATOM X) REDQUAD) (T (LIST NIL (CAR X) (CONS NIL (CDR X)))))))
VALUE =  UNPREFIX

MAKEPROP (CONDARG$ MODE PRIM)
VALUE =  CONDARG$

MAKEPROP (CONDARG$ DEFN (LAMBDA (X) (PROG (CLIST Z B) (SETQ CLIST (CDDR (CADR X))) (SETQ Z (CADDR X)) L (COND (
(LESSP (LENGTH CLIST) 2) (RETURN REDQUAD))) (SETQ B (REDUCE (CONS (QUOTE |) (CONS (CAR CLIST) Z)))) (COND ((EQ
B (QUOTE T)) (RETURN (CONS (QUOTE |) (CONS (CADR CLIST) Z)))))) (SETQ CLIST (CDDR CLIST)) (COND ((EQ B (QUOTE F)
) (GO L))) (RETURN REDQUAD))))
VALUE =  CONDARG$

MAKEPROP ($DEF MODE PRIM)
VALUE =  $DEF

MAKEPROP ($DEF DEFN (LAMBDA (X) (PROG (Y Z) (SETQ Y (PSI (CADDR X))) (SETQ Z (CADR X)) (COND ((EQ (CADDR X) Y)
(GO L1))) (MAKEPROP Z (QUOTE MODE) (QUOTE DELM)) (GO L2) L1 (MAKEPROP Z (QUOTE MODE) (QUOTE DELP)) L2 (MAKEPROP
Z (QUOTE DEFN) Y) (RETURN Z))))
VALUE =  $DEF
```

```
MAKEPROP (SFIN MODE PRIM)
VALUE =  SFIN

MAKEPROP (SFIN DEFN (LAMBDA (X) (PROG NIL (SETQ FINFLAG T) (RETURN X))))
VALUE =  SFIN
```

```
MAKEPROP (!1 MODE DELR)
VALUE =  !1

MAKEPROP (!1 DEFN (NIL CAR SEP))
VALUE =  !1

MAKEPROP (TL MODE DELR)
VALUE =  TL

MAKEPROP (TL DEFN (NIL CAR REV SEP))
VALUE =  TL

MAKEPROP (FST MODE DELM)
VALUE =  FST

MAKEPROP (FST DEFN (NIL APPLY ASSOC CAR ROT))
VALUE =  FST

MAKEPROP (ROT MODE DELR)
VALUE =  ROT

MAKEPROP (ROT DEFN (NIL ASSOC REV ASSOC REV))
VALUE =  ROT

MAKEPROP (ADJ MODE DELM)
VALUE =  ADJ

MAKEPROP (ADJ DEFN (NIL CAR ROT))
VALUE =  ADJ

MAKEPROP (SEC MODE DELM)
VALUE =  SEC

MAKEPROP (SEC DEFN (NIL REV APPLY ASSOC REV (NIL FST REV) REV CAR ROT))
VALUE =  SEC

MAKEPROP (C MODE DELM)
VALUE =  C

MAKEPROP (C DEFN (NIL CAR REV CAR))
VALUE =  C

MAKEPROP (NOT MODE DELR)
VALUE =  NOT

MAKEPROP (NOT DEFN (NIL COND REV (NIL ADJ (NIL F T))))
VALUE =  NOT

MAKEPROP (EQNIL MODE DELR)
VALUE =  EQNIL

MAKEPROP (EQNIL DEFN (NIL EQAT (NIL ADJ NIL)))
VALUE =  EQNIL
```

} ⟨$DEL (!1 (CAR SEP))⟩

} ⟨$DEL (TL (CAR REV SEP))⟩

etc.

Defined operators are entered as properties on the property list of the atom denoting the operator. Such an operator may have one of two MODE 's. The mode is DELM if the defining set has an * prefix and DELR otherwise. The defining set for the operator $\mathbb{X}$ is represented in internal notation as $\Psi\mathbb{Y}$, where $(\mathbb{X}, \mathbb{Y}) \in \nabla$, and appears as the property DEFN.

Note: Executing ⟨$DEL ($\mathbb{X}$ $\mathbb{Y}$)⟩ in RED sets up and executes the appropriate pairs of "make prop" 's in LISP to correctly define $\mathbb{X}$

965 · 18M · 04

```
MAKEPROP (ATOM MODE DELR)
VALUE =  ATOM

MAKEPROP (ATOM DEFN (NIL EQAT DBL))
VALUE =  ATOM

MAKEPROP (AND MODE DELR)
VALUE =  AND

MAKEPROP (AND DEFN (NIL COND COND REV (NIL FST (NIL TRPRS REV (NIL ADJ (NIL (NIL T F) (NIL F F))) DBL))))
VALUE =  AND

MAKEPROP (TRPRS MODE DELR)
VALUE =  TRPRS

MAKEPROP (TRPRS DEFN (NIL (NIL FST REV) REV ROT (NIL FST (NIL ASSOC REV)) ASSOC))
VALUE =  TRPRS

MAKEPROP (APP MODE DELR)
VALUE =  APP

MAKEPROP (APP DEFN (NIL CAR APPLY DBL))
VALUE =  APP

MAKEPROP (!2APP MODE DELR)
VALUE =  !2APP

MAKEPROP (!2APP DEFN (NIL REV APPLY REV APPLY))
VALUE =  !2APP

MAKEPROP (!2F MODE DELM)
VALUE =  !2F

MAKEPROP (!2F DEFN (NIL !2APP TRPRS (NIL SEC DBL) (NIL FST TL)))
VALUE =  !2F

MAKEPROP (!3F MODE DELM)
VALUE =  !3F

MAKEPROP (!3F DEFN (NIL (NIL SEC (NIL !2APP TRPRS)) APPLY TRPRS (NIL SEC (NIL CAR ROT DBL DBL)) (NIL FST (NIL SEC
 TL))))
VALUE =  !3F

MAKEPROP (!2 MODE DELR)
VALUE =  !2

MAKEPROP (!2 DEFN (NIL !1 TL))
VALUE =  !2

MAKEPROP (SETQUAD MODE DELR)
VALUE =  SETQUAD

MAKEPROP (SETQUAD DEFN (NIL CONC (NIL SEC (NIL ADJ ?)) (NIL FST (NIL QUAD (1)) DBL))
VALUE =  SETQUAD
```


965 - IBM - 04

```
MAKEPROP (DISTFUNC MODE DELR)
VALUE =  DISTFUNC

MAKEPROP (DISTFUNC DEFN (NIL TRPRS (NIL SEC DBL) (NIL FST (NIL SETQUAD (NIL !2F !2 F1)))))
VALUE =  DISTFUNC

MAKEPROP (CONDARG MODE DELM)
VALUE =  CONDARG

MAKEPROP (CONDARG DEFN (NIL APP COND APPLY (NIL SEC DISTFUNC) DISTFUNC))
VALUE =  CONDARG

MAKEPROP (F1 MODE DELR)
VALUE =  F1

MAKEPROP (F1 DEFN (NIL UNION (NIL SEC TL) SEP))
VALUE =  F1

MAKEPROP (S MODE DELR)
VALUE =  S

MAKEPROP (S DEFN (NIL UNION (NIL ADJ UNION) UNION (NIL ADJ (NIL ADJ SS)) (NIL ADJ ADJ)))
VALUE =  S

MAKEPROP (K MODE DELR)
VALUE =  K

MAKEPROP (K DEFN (NIL ADJ C))
VALUE =  K

MAKEPROP (I MODE DELR)
VALUE =  I

MAKEPROP (I DEFN NIL)
VALUE =  I

MAKEPROP (SS MODE DELM)
VALUE =  SS

MAKEPROP (SS DEFN (NIL APP !2APP TRPRS (NIL SEC DBL) (NIL FST TL)))
VALUE =  SS
```


965 - IBM - 04

```
SPECIAL ((CC))
VALUE =  ((CC))

SPECIAL ((REDIOPT FINFLAG REDQUAD))
VALUE =  ((REDIOPT) (FINFLAG) (REDQUAD))

SETQ (FINFLAG NIL)
VALUE =  NIL

SETQ (REDIOPT NIL)
VALUE =  NIL
```

Declaring and initializing of internal variables

CC : current column (used in R1RD)
REDIOPT: T if RED/1 F if RED/2
FINFLAG: T if <$FIN X> has been evaluated
REDQUAD: ? (system version of □)

```
COMP360 (((REDSYS (LAMBDA (X) (PROG NIL (SETQ FINFLAG NIL) (SETQ REDIOPT NIL) (COND ((EQ X 1) (SETQ REDIOPT T))
) L1 (R1PRT (RHO (R1RD))) (COND ((NOT FINFLAG) (GO L1))) (PRINT (QUOTE END_OF_RED_SYSTEM)) (PRINT (QUOTE RETURNING_TO_LI
SP)))))))

(REDSYS
   (LAMBDA (X) (PROG NIL
            (SETQ FINFLAG NIL)
            (SETQ REDIOPT NIL)
            (COND
               ((EQ X 1) (SETQ REDIOPT T)))
      L1    (R1PRT (RHO (R1RD)))
            (COND
               ((NOT FINFLAG) (GO L1)))
            (PRINT (QUOTE END_OF_RED_SYSTEM))
            (PRINT (QUOTE RETURNING_TO_LISP))))))
VALUE =  (REDSYS)
```

Supervisor for RED system

```
MACRO (((PREFIX (LAMBDA (X) (CONS (QUOTE CAR) (CDR X))))))
VALUE =  (PREFIX)

MACRO (((LBDY (LAMBDA (X) (CONS (QUOTE CDR) (CDR X))))))
VALUE =  (LBDY)

MACRO (((OPR (LAMBDA (X) (CONS (QUOTE CADR) (CDR X))))))
VALUE =  (OPR)

MACRO (((OPD (LAMBDA (X) (CONS (QUOTE CDDR) (CDR X))))))
VALUE =  (OPD)
```

macros for the functions to retrieve the prefix and body of lists and the operator and operand of an application

```
COMP360 (((R1PRT (LAMBDA (X) (PROG NIL (R1PRT$ X) (RETURN (TERPRI)))))))

(R1PRT
   (LAMBDA (X) (PROG NIL
            (R1PRT$ X)
            (RETURN (TERPRI)))))
VALUE =  (R1PRT)

COMP360 (((R1PRT$ (LAMBDA (X) (PROG (Y) (COND ((ATOM X) (RETURN (PRIN1 X)))) (COND ((APPLP X) (GO L5))) (COND (
(NULL (CAR X)) (GO L1))) (PPIN1 (CAR X)) (GO L2) L1 (PRIN1 BLANK) L2 (SETQ Y (CDR X)) (PRIN1 LPAR) L3 (COND ((NULL
 Y) (GO L4))) (R1PRT$ (CAR Y)) (SETQ Y (CDR Y)) (COND ((NULL Y) (GO L4))) (PRIN1 BLANK) (GO L3) L4 (RETURN (PRIN1
 RPAR)) L5 (PRIN1 LBRAC) (R1PRT$ (OPR X)) (PRIN1 BLANK) (R1PRT$ (OPD X)) (RETURN (PRIN1 RBRAC)))))))
```

```
(R1PRT$
   (LAMBDA (X) (PROG (Y)
            (COND
               ((ATOM X) (RETURN (PRIN1 X))))
            (COND
               ((APPLP X) (GO L5)))
            (COND
               ((NULL (CAR X)) (GO L1)))
            (PRIN1 (CAR X))
            (GO L2)
      L1    (PRIN1 BLANK)
      L2    (SETQ Y (CDR X))
            (PRIN1 LPAR)
      L3    (COND
               ((NULL Y) (GO L4)))
            (R1PRT$ (CAR Y))
            (SETQ Y (CDR Y))
            (COND
               ((NULL Y) (GO L4)))
            (PRIN1 BLANK)
            (GO L3)
      L4    (RETURN (PRIN1 RPAR))
      L5    (PRIN1 LBRAC)
            (R1PRT$ (OPR X))
            (PRIN1 BLANK)
            (R1PRT$ (OPD X))
            (RETURN (PRIN1 RBRAC)))))
VALUE =  (R1PRT$)
```

Routine to print RED expressions

```
COMP360 (((CONT| (LAMBDA (X) (COND ((ATOM X) (EQ X (QUOTE |))) (T (OR (CONT| (CAR X)) (CONT| (CDR X))))))))
```

```
(CONT|
   (LAMBDA (X) (COND
      ((ATOM X) (EQ X (QUOTE |)))
      (T (OR
         (CONT| (CAR X))
         (CONT| (CDR X)))))))
VALUE =  (CONT|)
```

Predicate that is T iff its argument contains the atom |

ie: if the argument is an RED expression then the predicate is T iff the argument is not a constant.

```
COMP360 (((RHO (LAMBDA (X) (COND (RED1OPT (REDUCE (SUBDEL X))) (T (REDUCE X))))))
```

```
(RHO
   (LAMBDA (X) (COND
      (RED1OPT (REDUCE (SUBDEL X)))
      (T (REDUCE X)))))
VALUE =  (RHO)
```

```
COMP360 (((SUBDEL (LAMBDA (X) (COND ((ATOM X) (COND ((RDEFNP X) (SUBDEL (GETRDEF X))) ((MDEFNP X) (SUBDEL (CONS
(QUOTE *) (CDR (GETRDEF X)))))) (T X))) (T (CONS (SUBDEL (CAR X)) (SUBDEL (CDR X))))))))
```

```
(SUBDEL
   (LAMBDA (X) (COND
      ((ATOM X) (COND
         ((RDEFNP X) (SUBDEL (GETRDEF X)))
```

if X is a RED expression subdel[x] substitutes the definition of all non primitive entities for those entities in X.

```
          ((MDEFNP X) (SUBDEL (CONS (QUOTE *) (CDR (GETRDEF X)))))
          (T X)))
       (T (CONS (SUBDEL (CAR X)) (SUBDEL (CDR X))))))
VALUE =  (SUBDEL)

COMP360 (((RDEFNP (LAMBDA (X) (EQ (CAR (PROP X (QUOTE MODE))) (QUOTE DELR))))))

(RDEFNP
   (LAMBDA (X) (EQ (CAR (PROP X (QUOTE MODE))) (QUOTE DELR))))
VALUE =  (RDEFNP)

COMP360 (((PSI (LAMBDA (X) (COND ((ATOM X) (COND ((MDEFNP X) (GETRDEF X)) (T X))) ((EQ (CAR X) (QUOTE *)) (CONS
NIL (CDR X))) (T X))))))

(PSI
   (LAMBDA (X) (COND
      ((ATOM X) (COND
         ((MDEFNP X) (GETRDEF X))
         (T X)))
      ((EQ (CAR X) (QUOTE *)) (CONS NIL (CDR X)))
      (T X))))
VALUE =  (PSI)

COMP360 (((PRIMP (LAMBDA (X) (EQ (CAR (PROP X (QUOTE MODE))) (QUOTE PRIM))))))

(PRIMP
   (LAMBDA (X) (EQ (CAR (PROP X (QUOTE MODE))) (QUOTE PRIM))))
VALUE =  (PRIMP)

COMP360 (((MDEFNP (LAMBDA (X) (EQ (CAR (PROP X (QUOTE MODE))) (QUOTE DELM))))))

(MDEFNP
   (LAMBDA (X) (EQ (CAR (PROP X (QUOTE MODE))) (QUOTE DELM))))
VALUE =  (MDEFNP)

COMP360 (((GETRDEF (LAMBDA (X) (COND ((NULL (PROP X (QUOTE DEFN))) REDQUAD) (T (CAR (PROP X (QUOTE DEFN)))))))))
)

(GETRDEF
   (LAMBDA (X) (COND
      ((NULL (PROP X (QUOTE DEFN))) REDQUAD)
      (T (CAR (PROP X (QUOTE DEFN)))))))
VALUE =  (GETRDEF)

SETQ (REDQUAD (QUOTE ?))
VALUE =  ?

COMP360 (((APPLP (LAMBDA (X) (AND (NOT (ATOM X)) (EQ (CAR X) (QUOTE |))))))))

(APPLP
   (LAMBDA (X) (AND
      (NOT (ATOM X))
      (EQ (CAR X) (QUOTE |)))))
VALUE =  (APPLP)
```

} rdefnp[x] = T iff $x \in \nabla$ and $\psi x = x$

} RED/2 ψ

} primp[x] = T iff x is a primitive operator

} mdefnp[x] = T iff $x \in \nabla$ and $\psi x \neq x$

} getrdef[x] = y if $(x, y) \in \nabla$
= $\square$ otherwise

} applp[x] = T iff x is an application

965 · IBM · 04

```
COMP360 (((MA (LAMBDA (X Y) (LIST (QUOTE |) (PSI X) NIL X Y)))))

(MA
   (LAMBDA (X Y) (LIST (QUOTE |) (PSI X) NIL X Y)))
VALUE =  (MA)

COMP360 (((RM (LAMBDA (X Y) (LIST (QUOTE |) (PSI (CAR (LBDY X))) NIL X Y)))))

(RM
   (LAMBDA (X Y) (LIST (QUOTE |) (PSI (CAR (LBDY X))) NIL X Y)))
VALUE =  (RM)

COMP360 (((EPSILON (LAMBDA (X Y) (COND ((NULL X) Y) ((EQ X (QUOTE DEF)) (COND ((RDEFNP Y) (GETRDEF Y)) (T Y)))
((EQ X (QUOTE PSI)) (PSI Y)) ((PRIMP X) (APPLX (GETRDEF X) (LIST Y))) ((RDEFNP X) (CONS (QUOTE |) (CONS (GETRDEF
 X) Y))) (T REDQUAD))))))

(EPSILON
   (LAMBDA (X Y) (COND
      ((NULL X) Y)
      ((EQ X (QUOTE DEF)) (COND
         ((RDEFNP Y) (GETRDEF Y))
         (T Y)))
      ((EQ X (QUOTE PSI)) (PSI Y))
      ((PRIMP X) (APPLX (GETRDEF X) (LIST Y)))
      ((RDEFNP X) (CONS (QUOTE |) (CONS (GETRDEF X) Y)))
      (T REDQUAD))))
VALUE =  (EPSILON)

COMP360 (((REDUCE (LAMBDA (E) (COND ((ATOM E) E) ((NOT (CONT| E)) E) ((APPLP E) (REDUCE (DELTA (REDUCE (OPR E))
 (REDUCE (OPD E))))) (T (CONS (REDUCE (CAR E)) (REDUCE (CDR E))))))))))

(REDUCE
   (LAMBDA (E) (COND
      ((ATOM E) E)
      ((NOT (CONT| E)) E)
      ((APPLP E) (REDUCE (DELTA (REDUCE (OPR E)) (REDUCE (OPD E)))))
      (T (CONS (REDUCE (CAR E)) (REDUCE (CDR E)))))))
VALUE =  (REDUCE)

COMP360 (((DELTA (LAMBDA (X Y) (COND ((ATOM X) (EPSILON X Y)) ((NOT (EQUAL (PSI X) X)) (MA X Y)) ((NOT (NULL (PREFIX
 X))) REDQUAD) ((ATOM (LBDY X)) REDQUAD) ((NOT (EQUAL (CAR (LBDY X)) (PSI (CAR (LBDY X))))) (RM X Y)) (T (RC X
Y)))))))

(DELTA
   (LAMBDA (X Y) (COND
      ((ATOM X) (EPSILON X Y))
      ((NOT (EQUAL (PSI X) X)) (MA X Y))
      ((NOT (NULL (PREFIX X))) REDQUAD)
      ((ATOM (LBDY X)) REDQUAD)
      ((NOT (EQUAL (CAR (LBDY X)) (PSI (CAR (LBDY X))))) (RM X Y))
      (T (RC X Y)))))
VALUE =  (DELTA)

COMP360 (((RC (LAMBDA (X Y) (PROG NIL (COND ((NULL (CDR (LBDY X))) (GO L1))) (RETURN (CONS (QUOTE |) (CONS (CAR
(LBDY X)) (REDUCE (CONS (QUOTE |) (CONS (CONS NIL (CDR (LBDY X))) Y)))))) L1 (RETURN (REDUCE (CONS (QUOTE |) (CONS
```

Rule MA for Δ

Rule MC for Δ

ϵ — primitive execution function
$\epsilon(X\ Y) = \rho X(Y)$, provided X is an atom

((SEQ? E) (MK-SEQ (REDUCE (HEAD E)) (REDUCE (TAIL E))))

top level meaning finder

Δ - restricted transition function
$\Delta(X\ Y) = \mu\langle X\ Y\rangle$, provided X, Y are constant

can delete

```
 (CAR (LBDY X)) Y)))))))))

(RC
   (LAMBDA (X Y) (PROG NIL
            (COND
               ((NULL (CDR (LBDY X))) (GO L1)))
            (RETURN (CONS (QUOTE |) (CONS (CAR (LBDY X)) (REDUCE (CONS (QUOTE |) (CONS (CONS NIL (CDR (LBDY X))
) Y))))))
      L1    (RETURN (REDUCE (CONS (QUOTE |) (CONS (CAR (LBDY X)) Y)))))))
VALUE =  (RC)

COMP360 (((RDCPR (LAMBDA NIL (PROG (X) (SETQ X (R1RD)) (COND ((EQ CC BLANK) (RETURN X))) (ERROR (QUOTE IMPROPER_APPLICAT
ION)))))))

(RDOPR
   (LAMBDA NIL (PROG (X)
            (SETQ X (R1RD))
            (COND
               ((EQ CC BLANK) (RETURN X)))
            (ERROR (QUOTE IMPROPER_APPLICATION)))))
VALUE =  (RDOPR)

COMP360 (((RDOPD (LAMBDA NIL (PROG (Z) (SETQ Z (R1RD)) (COND ((EQ CC RBRAC) (RETURN (CAR (CONS Z (SETQ CC (RDCHR
)))))))) (ERROR (QUOTE IMPROPER_APPLICATION)))))))

(RDOPD
   (LAMBDA NIL (PROG (Z)
            (SETQ Z (R1RD))
            (COND
               ((EQ CC RBRAC) (RETURN (CAR (CONS Z (SETQ CC (RDCHR))))))))
            (ERROR (QUOTE IMPROPER_APPLICATION)))))
VALUE =  (RDOPD)

DEFINE (((R1RD (LAMBDA NIL (PROG (PREFIX) (SETQ PREFIX NIL) L1 (SETQ CC (RDCHR)) L2 (SELECT CC (BLANK (GO L1))
(LBRAC (RETURN (CONS (QUOTE |) (CONS (RDOPR) (RDOPD))))) (LPAR (RETURN (CONS PREFIX (RDLST)))) (RPAR (ERROR (QUOTE
 )_OUT_OF_PLACE))) (RBRAC (GO L1)) (GC GTATM)) GTATM (CLEARBUFF) (COND ((DIGIT CC) (GO LA1))) LA0 (SELECT CC (LPAR
 (GO LA2)) (LBRAC (GC LA3)) (RBRAC (GO LA3)) (BLANK (GO LA3)) (RPAR (GO LA3)) (PACK CC)) (SETQ CC (RDCHR)) (GO
LA0) LA1 (PACK CC) (SETQ CC (RDCHR)) (COND ((DIGIT CC) (GO LA1))) (RETURN (NUMOB)) LA2 (SETQ PREFIX (INTERNX))
(GO L2) LA3 (RETURN (INTERNX)))))))
VALUE =  (R1RD)

COMP360 (((RDLST (LAMBDA NIL (SELECT CC (RPAR (EQ NIL (SETQ CC (RDCHR)))) (LBRAC (ERROR (QUOTE LIST_SYNTAX_ERROR
))) (RBRAC (ERROR (QUOTE LIST SYNTAX ERROR))) (CONS (R1RD) (RDLST)))))) FIN FIN    )

(RDLST
   (LAMBDA NIL (SELECT
      CC
      (RPAR (EQ NIL (SETQ CC (RDCHR))))
      (LBRAC (ERROR (QUOTE LIST_SYNTAX_ERROR)))
      (RBRAC (ERROR (QUOTE LIST SYNTAX ERROR)))
      (CONS (R1RD) (RDLST)))))
VALUE =  (RDLST)

(FIN IS MISSING)
```

Rule RC for Δ $\langle (x1,x2,\cdots,xN), y\rangle = \langle x1 \langle (x2,\cdots,xN), y\rangle\rangle$

Reads an operator from input

Reads an operand from input

Basic Read Routine (Reads one RED/1 expression)

Reads a list from input

965 - IBM - 04