Paul McJones

Date: August 29, 1972
From (Div. & location): T. J. Watson Research Center
U.S. mail address): Yotktown Heights, N.Y.
Dept. & Bldg: 440 - 801
Mail Stop & Tel. Ext.: 862 - 1464

IBM

Subject: Summary of RED primitives and their definitions as lambda notations.

Reference: RJ-1010

To: File

The attached sheets summerize the primitives for the RED system (pp. 61-64 of RJ-1010). Each primitive is accompanied by its definition as a LISP lambda expression.

P. D. Summers.

PDS/cah

965 - IBM - 04

Used through out the definitions in the following summary is the function npairp[x] which is a predicate defined to be false iff its argument, x, is a list containing exactly two items.

ie.

npairp[x] = not[equal[3; length [x]]]

(For definition of internal notation for RED expressions see the memo "Adding primitives to the RED System".)

The variable REDQUAD is global and evaluates to the RED □ represented in this system by ? . All other functions are standard LISP functions.

1. ϵ ⟨REV S(X Y)⟩ = S (Y X) S may be empty

```
rev[x] = [npairp[x]   →  redquad;
          T  →  list [car[x]; caddr[x]; cadr[x]]
```

2. ϵ ⟨ASSOC S(X T(Y Z))⟩= S (T(X Y) Z) S, T may be empty.

```
assoc[x] = [npairp[x]   →  redquad;
            npairp [caddr[x]]  →  redquad;
            T  →  list [car[x];
                  list[caaddr[x]; cadr[x];
                      cadr[caddr[x]]];
                      caddr[x]]]]
```

3. ϵ ⟨APPLY S(T(X Y) Z))⟩ = S(X Y Z) S, T may be empty

```
apply[x] = [npairp[x] → redquad;
```

```
        npairp[cadr[x]]  →  redquad;
        T  →  list[car[x]; cons[ | ; cons[cadr[cadr[x]];
                                        caddr[cadr[x]]];
                                              caddr[x]]]]
```

4. ϵ ⟨CAR S (X Y)⟩ = X S may be empty

```
car[x]  =  [npairp [x]]  →  redquad;
           T  →  cadr[x]].
```

5. ϵ ⟨COND S (T T(X Y))⟩ = X S, T may be empty

   ϵ ⟨COND S(F T(X Y))⟩ = Y

```
cond[x]  =  [npairp[x] → redquad;
            npairp[caddr[x]]  →  redquad;
            eq[T;cadr[x]]  →  cadr[caddr[x]];
            eq[F;cadr[x]]  →  caddr[caddr[x]];
            T  →  redquad]
```

6. ϵ ⟨EQAT S(□ X)⟩ = □

   ϵ ⟨EQAT S(X □)⟩ = □ S may be empty

   ϵ ⟨EQAT S(A A)⟩ = T

   ϵ ⟨EQAT S(X Y)⟩ = F

```
eqat[x] = [npairp[x]  →  redquad;
           eq[redquad;cadr[x]]  →  redquad;
           eq[redquad;caddr[x]]  →  redquad;
           eq[cadr[x];caddr[x]]  →  T
           T  →  F]
```


965 - IBM - 04

7. $\epsilon \langle$ UNION $S(X\ T(Y1\ .\ .\ .Yn))\rangle = S\ (S\ Y1\ .\ .\ .\ Yn)$

S may be empty

```
union[x] = [npairp[x] → redquad;
            T → cons[car[x]; cons[cadr[x]; cdr[caddr[x]]]]]
```

8. $c\langle$ DBL $X\rangle = (X\ X)$

```
dbl[x] = list[NIL; x; x]
```

9. $\epsilon \langle$ SEP $S(X)\rangle = S(x\ \phi)$ S may be empty

$\epsilon \langle$ SEP $S(X1\ X2\ \ldots\ Xn)\rangle = S(X1\ S(SxX2\ldots Xn))$

```
sep[x] = [atom[x] → redquad;
          atom[cdr[x]] → redquad;
          null[cdr[x]] → list[car[x];cadr[x];NIL];
          T → list [car[x]; cadr[x]; cons[car[x];cddr[x]]]]
```

10. $\epsilon \langle$ QUAD $\square\rangle$ = T

$\epsilon \langle$ QUAD $X\rangle$ = F

```
quad[x] = [eq[redquad;x] → T;
           T → F]
```

11. $\epsilon \langle$ PREFIX $S(T\ U(X1\ .\ .\ .Xn))\rangle = T(X1\ .\ .\ .Xn)$

S, T, U may be empty

```
prefix[x] = [npairp[x] → redquad;
             atom[caddr[x]] → redquad;
             T → cons[cadr[x];cdr[caddr[x]]]]
```

12. ε ⟨UNPREFIX S(X1 . . .Xn))⟩ = T(X1 . . . Xn)

S may be empty

```
unprefix[x] = [atom[x] → redquad;
              T → list [NIL;car[x];cons[NIL;cdr[x]]]
```


965 - IBM - 04