Paul McJones

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

PERSONAL & CONFIDENTIAL

IBM

Subject: Adding primitives to the RED Systems.

Reference: RJ-1010

To: File

A. Introduction

This memo describes the procedure for adding new primitives to the RED system. One of the objectives of this particular implementations of the RED language was to provide a system that could be studied as to its appropriateness as an application language. As is descussed in the section on RED/3 languages, application languages are characterized as having sophisticated primitives. Since the set of such primitives is unknown, for a given application area, the task of adding primitives should be a simple one so that different sets might be tried.

The procedure for adding new primitives to the system is simple and is described below in detail. A knowledge of LISP is assumed in the discussion below.

B. Object Representations

RED objects (applications, lists, etc.) are represented as LISP S-expressions. Of the three types of RED entities a coder of a primitive need only be concerned with atoms and sets. This is because it is not meaningful for an operator to operate an applications. (This is a weaker statement than saying all operands must be constants) since, for example,

REV (<F X><G Y>) → (<G Y> <F X>)

is meaningful).

A RED atom is represented by a LISP atom. A RED list is a LISP list with the prefix of the RED list the first element of the LISP representation. A null prefix is denoted by the atom NIL, the * prefix by *, etc. An application is the S-expression (| . (operator . operand)), where operator and operand are appropriate S-expressions denoting the operator and operand of the application.

C. Entering a new primitive into the system.

Before calling the RED interputer (REDSYS) the following pair of LISP

965 - IBM - 04

doublets should be entered for each new primitieve, prim, to be entered.

MAKEPROP (prim, MODE, PRIM)

MAKEPROP (prim, DEFN, defn)

Where defn is the LISP lambda expression that defines the primitive. Since all RED operators, primitive or otherwise, have but one operand, the lambda expression should also have but one bound variable.

D. Example

To illustrate the how a primitive may be entered, the code to define the primitive REV will be given.
The RED definition for REV is as follows:

$\epsilon \langle$REV X Y$\rangle$ = (Y X)

$\epsilon \langle$REV S (X Y)$\rangle$ = S (Y X)

This is defined only if the operand is a list of two items. Available in the library of primitives is the LISP function npairp[x] which is false iff x is a RED list of two entities.

Using this function the following lambda expression will correctly define the operator REV.

```
(LAMBDA  (X)

   (COND  ((NPAIRP  X) REDQUAD)

          (T (LIST (CAR X) (CADDR X)
             (CADR X)))))
```

Where REDQUAD evaluates the RED constant quad implemented as "?'.

For further examples of how primitives may be defined see the memo "Annotated Listings of the RED System" and the memo "Summary of RED primitives and their definition as lambda expressions".

P. D. Summers

PDS/cah

965 - IBM - 04