A LISP Interpreter in SNOBOL4

by

Clement McGowan
and
Peter Wegner

Technical Report
No. 68-27
November 1968

Department of Computer Science
Cornell University
Ithaca, New York 14850
A LISP INTERPRETER IN SNOBOL4

by

Clement McGowan

and

Peter Wegner

TABLE OF CONTENTS

1. Introduction 1
2. CAR , CDR and CONS 1
3. ATOM and EQ 5
4. EQUAL , PAIRLIS and ASSOC 6
5. List Representation of Forms and Functions 9
6. LISP Interpretation 12
7. APPLY , EVAL , EVCON and EVLIS 18
Appendix: Function Trace of FF 20

* This work was supported by NSF grant GJ-108
A LISP Interpreter in SNOBOL4

1. Introduction
(1)
SNOBOL4 is a convenient language for specifying interpreters. It has a natural notation both for syntactic scanning and for specifying data transformations during execution. Flexible tracing facilities allow selective printout of intermediate steps of the interpretation process. The LISP interpreter described below was programmed in SNOBOL4 to gain (2)
insight into the structure of the LISP apply function, and is part of a wider study of techniques for writing interpreters.

In the discussion below successive features of the LISP language will be introduced, and their implementation in SNOBOL4 will be defined. The LISP functions required to implement the LISP apply function will be specified in LISP, and then implemented in SNOBOL4. Sufficient explanatory material is given so that the discussion can be understood by persons not familiar with LISP or SNOBOL4.

1. CAR, CDR and CONS

LISP has two selection primitives "car" and "cdr" which respectively select the first element and remaining elements of a list.
car[(A B C)] = A
cdr[(A B C)] = (B C)
car[cdr[(A B C)]] = cadr[(A B C)] = B
cdr[cdr[(A B C)]] = cddr[(A B C)] = (C)
car[cdr[cdr[(A B C)]]] = caddr[(A B C)] = C

The primitive cons[x;y] constructs a list whose first element is x and whose remaining elements are the elements of the list y.

LISP has two kinds of data objects referred to respectively as atoms and lists. The list (A B C) has as its elements a sequence of three atoms. The two element list ((A B)C) has the list (A B) as its first component and the atom C as its second component. The lists (A B C) and ((A B)C) may be represented as follows:

```
(A B C)  ((A B)C)
```

* The abbreviation "c followed by a sequence of a's and d's followed by x" is an abbreviation for a corresponding sequence of cars and cdrs. Note particularly that cadr selects the second list element and caddr selects the third list element.
The symbol NIL is a distinguished atom used to terminate lists. Each list element may be thought of as having two pointer fields, car[L] selects the object (atom or list) pointed to by the first field, while cdr[L] selects the object (list or null list) pointed to by the second field. cons[x;y] creates a new list element whose first field points to the object x and whose second field points to the list y.

In simulating list processing operations in a string manipulation language such as SNOBOL4, the first impulse is to represent lists as strings, and to perform list selection by pattern matching. However, a more esthetic and efficient form of list representation can be obtained using the SNOBOL4 data definition facility.

The SNOBOL4 command "DATA('ID(FIELD1,...,FIELDN)')" defines a new datatype of type ID with N fields called FIELD1,...,FIELDN. When a new datatype has been defined by execution of a "DATA" command, instances of the datatype can be created by commands such as the following:

\[
X = ID(X1,...,XN)
\]

Create an instance of the datatype ID with the N component fields initialized to X1,...,XN, and assign the newly created object as the value of X.

If X is an object of datatype ID, the components of X can be obtained using the field names in the data definition as selector functions.
Y = FIELD1(X)  
If X is of type ID, this statement selects the first field of X and assigns this field as the value of Y. The statement fails if X is not of the datatype ID.

In SNOBOL4 the datatype of an object is a component of the value. The system function DATATYPE(X) returns as its value the datatype of X.

Y = DATATYPE(X)  
If X is of the datatype ID, this statement assigns to Y the two character string "ID".

The above data definitions facility may be used to define a datatype CONS with fields CAR and CDR.

DATA('CONS(CAR, CDR)')

The above data definition allows instances of this datatype to be created, and allows CAR(X) and CDR(X) to be used to select the CAR and CDR fields of elements of the type CONS.

Z = CONS('X', NIL)  
Z becomes an element of type CONS with its CAR field having the value 'X' and its CDR field having the value NIL.

V = CAR(Z)  
Assign the CAR field of Z to V

W = CDR(Z)  
Assign the CDR field of Z to W
Z = CONS(CAR(Z), CDR(Z))  If Z is of the type CONS, this statement will leave the value of Z unchanged.

Z = CONS('X', CONS('Y', NIL))  This statement assigns 'X' as the CAR field of Z and an element of type CONS as the CDR field of Z.

3. ATOM and EQ

It is evident that in the above representation the datatype CONS corresponds precisely to the datatype LIST of LISP. The datatype ATOM of LISP is a subclass of the SNOBOL4 datatype STRING. In order to simplify the discussion we shall assume that any object of the datatype STRING is an atom.

LISP has a one-argument predicate atom[x] which yields the value T if x is an atom and the value F otherwise. This predicate can be specified in SNOBOL4 as follows.*

```
DEFINE('ATOM(X)')
ATOM IDENT(DATATYPE(X), 'STRING'):F(A1)
ATOM='T'                :(RETURN)ment. If the datatype
A1 ATOM='F'                :(RETURN)is 'STRING', the
                             value of the function is 'T'. Otherwise the
                             value is 'F'.
```

LISP has a two-argument predicate eq[x;y] which has the value 'T' if x and y denote the same atom, has the value 'F' if x and y denote different atoms, and has
the value 'UNDEFINED' if x or y or both are not atoms.

This predicate can be specified in SNOBOL4 as follows:

```
DEFINE('EQ(X,Y)')
EQ IDENT(DATATYPE(X),'STRING'):F(EQ2)
IDENT(DATATYPE(Y),'STRING'):F(EQ2)
IDENT(X,Y) :F(EQ1)
EQ='T' :(RETURN)
EQ1 EQ='F' :(RETURN)
EQ2 EQ='UNDEFINED' :(FRETURN)
```

This function returns the value 'UNDEFINED' unless X and Y have the datatype STRING. The value 'T' is returned if X and Y are identical strings. The value 'F' is returned if X and Y are different strings.

4. EQUAL, PAIRLIS and ASSOC

Having defined the basic LISP primitives, we shall next discuss the SNOBOL4 implementation of a number of LISP functions required in specifying a LISP interpreter.

The function EQUAL is a generalization of EQ which tests the equality of both atoms and lists. It is specified in LISP as follows:

* SNOBOL4 functions are introduced by the function declaration DEFINE. The function name and parameters are in quotes since the arguments of all SNOBOL functions are evaluated before being used. If no entry point is explicitly specified, the entry point is assumed to be the statement labelled with the function name. Conditional transfers S(L) (transfer to L on success) F(L) (transfer to L on failure) and (L) (unconditional transfer to L )can be specified following each statement. Transfer to the label RETURN terminates function execution successfully, while transfer to the label FRETURN returns control with a failure indicator that may be used to trigger a conditional transfer F(L).
equal[x;y] = [atom[x] ⇒ [atom[y] ⇒ eq[x;y]; T ⇒ F]; atom[y] ⇒ F; equal[car[x];car[y]] ⇒ equal[cdr[x];cdr[y]]; T ⇒ F]

If x is an atom, then if y is an atom the value is eq[x;y]. If x is an atom and y is not an atom, the value is false. If y is an atom and x is not an atom, the value is false. If both x and y is nonatomic then if car[x] = car[y], the value is equal[cdr[x];cdr[y]]. If car[x] ≠ car[y] the value is F.

The SNOBOL4 program for implementing EQUAL has a strong resemblance to the LISP program. However, LISP conditional expressions are replaced by SNOBOL4 conditional branching expressions.

DEFINE('EQUAL(X,Y)')
EQUAL IDENT(DATATYPE(X),'STRING') :F(EL2) If both arguments are of datatype IDENT(DATATYPE(Y),'STRING') :F(EL1) string the argu-
EQUAL=EQ(X,Y) :(RETURN) ments are atomic EL1 EQUAL='F' :(RETURN) and the value is
EL2 IDENT(DATATYPE(Y),'STRING') :F(EL1) EQ(X,Y). If only IDENT(EQUAL(CAR(X),CAR(X)),'T'):F(EL1) one argument is
EQUAL=EQUAL(CDR(X),CDR(Y)) :(RETURN) atomic, the value if 'F'. If both arguments are lists, the value is obtained by applying the EQUAL function recursively to both the CAR and CDR components.
The next function to be defined has as its arguments two lists \( X = (X_1, \ldots, X_N) \), \( Y = (Y_1, \ldots, Y_N) \) with the same number of elements and a list \( A \) whose elements are ordered pairs and effectively constitute symbol table entries. The function PAIRLIST forms elements \((X_I, Y_I) \) \( I = 1, \ldots, N \) and appends these to the front of the list \( A \).

Example: Let \( A = ((D)X)(E)Y) \) then

\[
\text{PAIRLIST}[(A\ B\ C),(U\ V\ W),A] \rightarrow ((A\ U)(B\ V)(C\ W)(D\ X)(E\ Y)).
\]

The function PAIRLIST is defined in LISP as follows:

\[
\text{PAIRLIST}[x;y;a]=\begin{cases} \text{null}[x] \rightarrow a;T \rightarrow \text{cons}[\text{cons}[\text{car}[x];\text{car}[y]]; \text{PAIRLIST}[\text{cdr}[x];\text{cdr}[y];a]) \end{cases}.
\]

The corresponding SNOBOL4 program is as follows:

\[
\begin{align*}
\text{DEFINE('PAIRLIST(X,Y,A)')} \\
\text{PAIRLIST IDENT(X,WIL) \rightarrow F(P1)} \\
\text{PAIRLIST=A} \rightarrow (\text{RETURN}) \\
P1 \text{ PAIRLIST } = \text{CONS[CONS(CAR(X),CAR(Y)),PAIRLIST(CDR(X),} \\
\text{ CDR(Y),A)):(RETURN)}
\end{align*}
\]

The function \( \text{assoc}[x;a] \) has an atom \( x \) and a pairlist \( a \) as its arguments, and produces as its value the first pair in \( a \) having \( x \) as its first element. The function assoc is defined in LISP as follows:

\[
\text{assoc}[x;a]=\begin{cases} \text{equal}[\text{car}[\text{car}[a]];x] \rightarrow \text{car}[a];T \rightarrow \text{assoc}[x;\text{cdr}[a]) \end{cases}.
\]

The corresponding SNOBOL4 program is as follows:
5. List Representation of Forms and Functions

The LISP interpreter assumes that both the program being interpreted and the data are list structures. Before defining the remaining functions required for LISP interpretation, the correspondence between LISP programs and lists will be examined. LISP programs specified in the previously given notation are sometimes referred to as M-expressions (meta expressions) while programs specified as list structures are sometimes referred to as S-expressions (symbolic expressions). The translation of M-expressions into S-expressions may be thought of as a compilation process. The process of LISP interpretation will be illustrated by a running example, using the function ff defined below which selects the first atomic element of a list.

\[ ff[x] = [atom[x] \rightarrow x; T \rightarrow ff[car[x]]] \]

The above expression is called a form and is defined only when the values of \( x \) and \( ff \) are specified.

The complete LISP program for the function \( ff \) requires a \( \lambda \) operator to identify the bound variable \( x \) and a label operator to indicate that the symbol \( ff \) denotes the function name.
\[ \text{label[ff, } \lambda [x]; \text{atom}[x] \rightarrow x; T \rightarrow ff[\text{car}[x]][] \] .

The rules for translating LISP programs in the above representation into list representation are given in the LISP manual.\(^1\) Using these rules the above function \( ff \) translates into the following representation:

\( \text{LABEL FF } (\text{LAMBDA}(X)(\text{COND}((\text{ATOM } X) X)((\text{QUOTE } T)(\text{FF}((\text{CAR } X))))) \)

The above program representation would be represented by the following list structure:
This list structure can be constructed in SNOBOL4 by the following sequence of data definitions.*

* It is assumed that NIL is represented by the null string. Variables in SNOBOL4 automatically have the null string as their initial value. However, NIL can be explicitly initialized to the null string by executing the statement "NIL = ".
P = CONS('X',NIL)
P = CONS('CAR',P)
P = CONS(P,NIL)
P = CONS('FF',P)
P = CONS(P,NIL)
Q = CONS('T',NIL)
Q = CONS('QUOTE',Q)
P = CONS(Q,P)
P = CONS(P,NIL)
Q = CONS('X',NIL)
Q = CONS('ATOM',Q)
R = CONS('X',NIL)
Q = CONS(Q,R)
P = CONS(Q,P)
P = CONS('COND',P)
P = CONS(P,NIL)
Q = CONS('X',NIL)
P = CONS(Q,P)
P = CONS('LAMBDA',P)
P = CONS(P,NIL)
P = CONS('FF',P)
P = CONS('LABEL',P)

Note that we need precisely twenty-two instances of the operation CONS corresponding to the twenty-two list cells in the list structure.

6. LISP Interpretation

In order to illustrate the operation of a LISP interpreter, we shall illustrate the successive steps during execution when the above function ff is applied to the argument ((A B)C).
During execution successive symbols pointed to by list cells of a LISP program are scanned and actions are performed depending on the symbol being scanned just as though the successive symbols were machine language instructions. We shall first indicate the overall flavor of the LISP interpretation rules, and then specify them precisely by LISP and SNOBOL4 programs.

The first symbol to be encountered is the symbol LABEL. This symbol causes the function name ff defined by the second list element to be paired with the function defined by the remaining part of the list, and causes control to be switched to the remaining part of the list, starting with the symbol LAMBDA.

Scanning of LAMBDA causes the bound variable list which constitutes the second list element to be paired with the list of arguments, and causes control to be transferred to the remaining part of the list starting with the symbol COND.

The list elements following COND each denote a predicate-expression pair. Scanning of COND results in evaluation of successive predicates, and delivers the value of the expression associated with the first true predicate.

The action associated with a given element of the list structure is defined in terms of other elements having a fixed relative position to the given element. Each action may transform the program string, the data string, and the pair list. Execution is completed when all actions triggered by the first list element have been completed.
The function which performs LISP interpretation will be defined first in LISP and then in SNOBOL4. The LISP interpreter is defined by a two-argument function `evalquote` which takes the program list and data list as its two arguments. The function `evalquote` is defined in terms of a three-argument function `apply`, which takes the program list, data list, and pair list as its three arguments. The pair list is initially assumed to be empty.

\[
\text{evalquote}[fn; x] = \text{apply}[fn; x; \text{NIL}].
\]

The function `apply` specifies an action when the first symbol is atomic and equal to `CAR`, `CDR`, `CONS`, `ATOM`, or `EQ`, or when car of the first argument is `LAMBDA` or `LABEL`.

\[
\text{apply}[fn; x; a] =
\begin{align*}
\text{[atom}[fn] & \rightarrow \text{[eq}[fn; \text{CAR}] \rightarrow \text{caar}[x]; \\
\text{eq}[fn; \text{CDR}] & \rightarrow \text{cdar}[x]; \\
\text{eq}[fn; \text{CONS}] & \rightarrow \text{cons}[\text{car}[x]; \text{cadr}[x]]; \\
\text{eq}[fn; \text{ATOM}] & \rightarrow \text{atom}[\text{car}[x]]; \\
\text{eq}[fn; \text{EQ}] & \rightarrow \text{eq}[\text{car}[x]; \text{cadr}[x]]; \\
T & \rightarrow \text{apply}[\text{eval}[fn; a]; x; a]; \\
\text{eq}[\text{car}[fn]; \text{LAMBDA}] & \rightarrow \text{eval}[\text{caddr}[fn]; \text{pairlis}[\text{cadr}[fn]; x; a]]; \\
\text{eq}[\text{car}[fn]; \text{LABEL}] & \rightarrow \text{apply}[\text{caddr}[fn]; x; \text{cons}[\text{cons}[\text{cadr}[fn]; \text{caddr}[fn]]; a]]
\end{align*}
\]

When this function is first called during the evaluation of `ff`, the predicate "eq[car[fn]; LABEL]" will be the first true predicate. The value will therefore be as follows:
apply[car[cdr[cdr[fn]]]];x;cons[cons[car[cdr[fn]]] 
   car[cdr[cdr[fn]]]];NIL] .

Now car[cdr[cdr[fn]]] is precisely the portion of the list starting with the cell pointing to LAMBDA. The argument list remains unchanged, and the pair list is augmented by the pair (car[cdr[fn]] , car[cdr[cdr[fn]]]) = (FF , (LAMBDA ~)).

When the new instance of apply is evaluated the first true predicate is eq[car[fn];LAMBDA], which yields the following value:

eval[car[cdr[cdr[fn]]]];pairlis[car[cdr[fn]]];x;a];

eval is a two-argument function whose first argument is a form and whose second argument is the pairlist in which the values of the bound variables in the form are defined. The function eval is defined as follows:

eval[e;a] = [atom[e] -> cdr[assoc[e;a]];  
   atom[car[e]] -> [eq[car[e];QUOTE] -> cdr[e];  
   eq[car[e];COND] -> evcon[cdr[e];a];  
   T -> apply[car[e];evlis[cdr[e];a];a];  
   T -> apply[car[e];evlis[cdr[e];a];a]] .

When eval is entered following execution of the "LAMBDA" command the first argument is the following list

(COND((ATOM X) X )((QUOTE T)(FF(CAR X))))
The second argument is the pairlist with values defined for FF and X. We will assume that the argument X has the initial value ((A B)C).

The first true predicate is "eq[car[e];COND];" yielding the value "evcon[cdr[e];a]". evcon is a function for evaluating successive predicate-expression pairs, and is defined as follows:

\[ evcon[c;a] = [eval[caar[c];a] \rightarrow eval[cadar[c];a]; T \rightarrow evcon[cdr[c];a]] \]

The first argument c is always a list of the form
\[ ((p_1 e_1)(p_2 e_2)\ldots(p_n e_n)) \]

\( \text{car(car(c))} \) is the first predicate \( p_1 \) and \( \text{eval[caar[c];a]} \) yields the value \( T \) if \( p_1 \) is true and \( F \) if \( p_1 \) is false. In the first case, \( e_1 \) is evaluated while in the second case evcon is reentered with the list \[ ((p_2 e_2)\ldots(p_n e_n)) \]

In our illustrative example \( p_1 \) is (ATOM X). Call of eval with this argument results in execution of
"apply[car[e];evlis[cdr[e];a];a]".

evlis is a function whose first argument is a list \( (x_1 x_2 \ldots x_n) \) and whose second argument is a pairlist a. The effect of evaluation is to produce the list of values \( (\text{val}(x_1), \ldots, \text{val}(x_n)) \) with respect to the pairlist a.
\[ \text{evlis}(\lambda; a) = [\text{null}(\lambda) \rightarrow \text{NIL}; \ T \rightarrow \text{cons}(\text{eval}[\text{car}(\lambda); a]; \text{evlis}(\text{cdr}(\lambda); a))] \]

In the present case \( \lambda \) is the one element list \( (X) \). and evaluation yields the argument \( ((A \ B)C) \). Thus the function "apply[\text{car}(e); \text{evlis}(\text{cdr}(e); a); a]" has "ATOM" as its first argument and \( ((A \ B)C) \) as its second argument. The predicates \text{atom}[fn] \) and \text{eq}[fn; \text{ATOM}] \) are satisfied yielding the value \( \text{atom}[\text{car}(((A \ B)C))] \) which is false. We, therefore, return from the function apply to the function \text{evcon} \), and try the alternative "T \rightarrow \text{evcon}[\text{cdr}(c); a]".

The new first argument of evcon is now 
(QUOTE T)(FF(CAR X)). eval is entered and the predicate \text{eq}[\text{car}(e); \text{QUOTE}] \) is satisfied yielding the value \( \text{cadr}(e) = T \). Control is returned to evcon and "eval[\text{cadar}(c); a]" is then executed where "cadar[c] = (FF(CAR X))".

Entry to eval with this first argument results in execution of "apply[\text{car}(e); \text{evlis}(\text{cdr}(e); a); a]" with \( \text{car}(e) = \text{FF} \) and \( \text{cdr}(e) = (\text{CAR} \ \text{X}) \). Evaluation of the second argument eventually yields \( (A \ B) \), while evaluation of the first argument yields the following list:

\[ (\text{LAMBDA}(X)(\text{COND}(((\text{ATOM} \ X)X)(\text{QUOTE} \ T)(\text{FF}(\text{CAR} \ X)))) \]

Further evaluation results in associating a new value \( (A \ B) \) with \( X \) and repeating the above procedure. \( \text{atom}[(A \ B)] \) is again false, yielding the above function with \( X = A \).
When the procedure is repeated a third time, \( \text{atom}(A) = T \) yielding the value \( X = A \).

7. APPLY, EVAL, EVCON and EVLIS

We have already defined the function \( \text{car}, \text{cdr}, \text{cons}, \text{eq}, \text{atom}, \text{equal}, \text{pairlis} \) and \( \text{assoc} \) in SNOBOL4. In order to complete our definition of a LISP interpreter we must define the functions \( \text{apply}, \text{eval}, \text{evcon}, \text{evlis} \) and \( \text{evalquote} \) in SNOBOL4.

The function \( \text{apply} \) can be defined by the following SNOBOL4 program:

```
DEFINE('APPLY(F,X,A)')
APPLY IDENTITY(DATATYPE(F),'STRING') :F(AP6) eq[F;ATOM]
   IDENTITY(F,'CAR') :F(AP1) eq[F;CAR]
   APPLY=CAR(CAR(X)) :(RETURN) return with data element
AP1 IDENTITY(F,'CDR') :F(AP2) eq[F;CDR]
   APPLY=CDR(CAR(X)) :(RETURN) return with data element
AP2 IDENTITY(F,'CONS') :F(AP3)
   APPLY=CONS(CAR(X),CAR(CDR(X))) :(RETURN)
AP3 IDENTITY(F,'ATOM') :F(AP4)
   APPLY=ATOM(CAR(X)) :(RETURN)
AP4 IDENTITY(F,'EQ') :F(AP5)
   APPLY=EQU(CAR(X),CAR(CDR(X))) :(RETURN)
AP5 APPLY=APPLY(EVAL(F,A),X,A) :(RETURN)
AP6 IDENTITY(CAR(F),'LAMBDA') :F(AP7)
   APPLY=EVAL(CAR(CDR(CDF(F))),PAIRLIS(CAR(CDR(F)),X,A)) :(RETURN)
AP7 IDENTITY(CAR(F),'LABEL') :F(AP8)
   ARG1=CAR(CDR(CDF(F)))
   ARG3=CONS(CONS(CAR(CDR(F)),ARG1),A)
   APPLY=APPLY(ARG1,X,ARG3)
AP8 OUTPUT='ERROR **** FUNCTION ILL FORMED' :(END)
```
The function eval may be defined as follows:

\[
\begin{align*}
\text{DEFINE('EVAL(E,A)\text{CARE}') } & \quad \text{CARE is a local variable} \\
\text{EVAL} & \quad \text{IDENT(DATATYPE(E),'STRING')} : F(EV1) \text{ atom[e]} \\
\text{EVAL} & \quad \text{=CDR(ASSOC(E,A))} : (\text{RETURN})\text{value in a-list} \\
\text{EV1} & \quad \text{CARE=} \text{CAR(E)} \\
\text{IDENT(DATATYPE(CARE),'STRING')} & \quad : F(EV3) \text{ atom[car[e]]} \\
\text{IDENT(CARE,'QUOTE')} & \quad : F(EV2) \text{ eq[car[e];QUOTE]} \\
\text{EVAL} & \quad \text{=CAR(CDR(E))} : (\text{RETURN})\text{return with quoted} \\
\text{EV2} & \quad \text{IDENT(CARE,'COND')} \\
\text{EVAL} & \quad \text{=EVCON(CDR(E),A)} : (\text{RETURN})\text{evaluate conditional} \\
\text{EV3} & \quad \text{EVAL=} \text{APPLY(CARE,EVLIS(CDR(E),A),A)} : (\text{RETURN})\text{evaluate list}
\end{align*}
\]

The functions evcon and evlis may be defined as follows:

\[
\begin{align*}
\text{DEFINE('EVCON(C,A)')} \\
\text{EVCON} & \quad \text{IDENT(EVAL(CAR(CAR(C)),A),'T')} : F(EC1) \\
\text{EVCON} & \quad =\text{EVAL(CAR(CAR(C)),A)} : (\text{RETURN}) \\
\text{EC1} & \quad \text{EVCON=} \text{EVCON(CDR(C),A)} : (\text{RETURN})
\end{align*}
\]

\[
\begin{align*}
\text{DEFINE('EVLIS(L,A)')} \\
\text{EVLIS} & \quad \text{IDENT(L,NIL)} \\
\text{EVLIS=} : (\text{RETURN}) \\
\text{ES1} & \quad \text{EVLIS=} \text{CONS(EVAL(CAR(L),A),EVLIS(CDR(L),A))} : (\text{RETURN})
\end{align*}
\]

The function evalquote may now be defined as follows:

\[
\begin{align*}
\text{DEFINE('EVALQUOTE(FCT,ARGS)')} \\
\text{EVALQUOTE} & \quad \text{EVALQUOTE=} \text{APPLY(FCT,ARGS,NIL)}
\end{align*}
\]

The above definitions complete the definition of the LISP interpreter. It is evident that once the functions CAR, CDR and CONS have been defined using the data definition facility, all other functions can be defined in SNOBOL4 almost by transliteration.
The above definitions would require the function and arguments to be supplied as lists whose elements are of data type CONS. If the input is to be specified in a conventional list notation, then a "compiler" would have to be written for translating lists represented as strings in a conventional notation into data structures of the type CONS.


APPENDIX: Function Trace of FF

The sequence of entry to and exit from functions during execution of FF for the argument ARG = ((A B)C) was traced using the SNOBOL4 function trace facility. Interpretation required execution of 91 function calls with a maximum nesting depth of 21 as indicated below.

The listing below gives the depth of dynamic function nesting (level number), a "call" or "return" indication, the function name, the type of function arguments, and the time in milliseconds to the nearest 60th of a second. The total time required to apply FF to its arguments was approximately 2 1/2 seconds.
LEVEL 0 CALL OF APPLY(CONS, CONS, '), TIME = 150
LEVEL 1 CALL OF APPLY(CONS, CONS, CONS), TIME = 167
LEVEL 2 CALL OF PAIRLIS(CONS, CONS, CONS), TIME = 183
LEVEL 3 CALL OF PAIRLIS(';', ';', CONS), TIME = 383
LEVEL 3 RETURN OF PAIRLIS = CONS, TIME = 383
LEVEL 2 RETURN OF PAIRLIS = CONS, TIME = 383
LEVEL 2 CALL OF EVAL(CONS, CONS), TIME = .866
LEVEL 3 CALL OF EVCON(CONS, CONS), TIME = .882
LEVEL 4 CALL OF EVAL(CONS, CONS), TIME = .882
LEVEL 5 CALL OF EVLIS(CONS, CONS), TIME = .899
LEVEL 6 CALL OF EVAL('X', CONS), TIME = .899
LEVEL 7 CALL OF ASSOC('X', CONS), TIME = .899
LEVEL 8 CALL OF EQUAL('X', 'X'), TIME = 916
LEVEL 9 CALL OF EQ ('X', 'X'), TIME = 916
LEVEL 9 RETURN OF EQ = 'T', TIME = 932
LEVEL 8 RETURN OF EQUAL = 'T', TIME = 932
LEVEL 7 RETURN OF ASSOC = CONS, TIME = 949
LEVEL 6 RETURN OF EVAL = CONS, TIME = 949
LEVEL 6 CALL OF EVLIS(';', CONS), TIME = 965
LEVEL 6 RETURN OF EVLIS = ';', TIME = 965
LEVEL 5 RETURN OF EVLIS = CONS, TIME = 982
LEVEL 5 CALL OF APPLY('ATOM', CONS, CONS), TIME = 1049
LEVEL 6 CALL OF ATOM(CONS), TIME = 1065
LEVEL 6 RETURN OF ATOM = 'F', TIME = 1065
LEVEL 5 RETURN OF APPLY = 'F', TIME = 1065
LEVEL 4 RETURN OF EVAL = 'F', TIME = 1082
LEVEL 4 CALL OF EVCON(CONS, CONS), TIME = 1082
LEVEL 5 CALL OF EVAL(CONS, CONS), TIME = 1099
LEVEL 5 RETURN OF EVAL = 'T', TIME = 1099
LEVEL 5 CALL OF EVAL(CONS, CONS), TIME = 1115
LEVEL 6 CALL OF EVLIS(CONS, CONS), TIME = 1132
LEVEL 7 CALL OF EVAL(CONS, CONS), TIME = 1148
LEVEL 8 CALL OF EVLIS(CONS, CONS), TIME = 1148
LEVEL 9 CALL OF EVAL('X', CONS), TIME = 1165
LEVEL 10 CALL OF ASSOC('X', CONS), TIME = 1165
LEVEL 11 CALL OF EQUAL('X', 'X'), TIME = 1215
LEVEL 12 CALL OF EQ ('X', 'X'), TIME = 1262
LEVEL 12 RETURN OF EQ = 'T', TIME = 1282
LEVEL 11 RETURN OF EQUAL = 'T', TIME = 1298
LEVEL 10 RETURN OF ASSOC = CONS, TIME = 1315
LEVEL 9 RETURN OF EVAL = CONS, TIME = 1315
LEVEL 9 CALL OF EVLIS(';', CONS), TIME = 1315
LEVEL 9 RETURN OF EVLIS = ';', TIME = 1332
LEVEL 8 RETURN OF EVLIS = CONS, TIME = 1348
LEVEL 8 CALL OF APPLY('CAR', CONS, CONS), TIME = 1348
LEVEL 8 RETURN OF APPLY = CONS, TIME = 1365
LEVEL 7 RETURN OF EVAL = CONS, TIME = 1365
LEVEL 7 CALL OF EVLIS(';', CONS), TIME = 1365
LEVEL 7 RETURN OF EVLIS = ';', TIME = 1381
LEVEL 6 RETURN OF EVLIS = CONS, TIME = 1381
LEVEL 6 CALL OF APPLY('FF', CONS, CONS), TIME = 1398
LEVEL 7 CALL OF EVAL('FF', CONS), TIME = 1398
LEVEL 8 CALL OF ASSOC('FF', CONS), TIME = 1431
LEVEL 9 CALL OF EQUAL('X','FF'), TIME = 1431
LEVEL 10 CALL OF EQ ('X','FF'), TIME = 1431
LEVEL 10 RETURN OF EQ = 'F', TIME = 1448
LEVEL 9 RETURN OF EQUAL = 'F', TIME = 1448
LEVEL 9 CALL OF ASSOC('FF',CONS), TIME = 1465
LEVEL 10 CALL OF EQUAL('FF','FF'), TIME = 1465
LEVEL 11 CALL OF EQ ('FF','FF'), TIME = 1481
LEVEL 11 RETURN OF EQ = 'T', TIME = 1481
LEVEL 10 RETURN OF EQUAL = 'T', TIME = 1481
LEVEL 9 RETURN OF ASSOC = CONS, TIME = 1498
LEVEL 8 RETURN OF ASSOC = CONS, TIME = 1515
LEVEL 7 RETURN OF EVAL = CONS, TIME = 1515
LEVEL 7 CALL OF APPLY(CONS,CONS,CONS), TIME = 1531
LEVEL 8 CALL OF PAIRLIS(CONS,CONS,CONS), TIME = 1548
LEVEL 9 CALL OF PAIRLIS('','''',CONS), TIME = 1548
LEVEL 9 RETURN OF PAIRLIS = CONS, TIME = 1548
LEVEL 8 RETURN OF PAIRLIS = CONS, TIME = 1564
LEVEL 8 CALL OF EVAL(CONS,CONS), TIME = 1564
LEVEL 9 CALL OF EVCON(CONS,CONS), TIME = 1581
LEVEL 10 CALL OF EVAL(CONS,CONS), TIME = 1581
LEVEL 11 CALL OF EVLIS(CONS,CONS), TIME = 1598
LEVEL 12 CALL OF EVAL('X',CONS), TIME = 1598
LEVEL 13 CALL OF ASSOC('X',CONS), TIME = 1614
LEVEL 14 CALL OF EQUAL('X','X'), TIME = 1631
LEVEL 15 CALL OF EQ ('X','X'), TIME = 1631
LEVEL 15 RETURN OF EQ = 'T', TIME = 1648
LEVEL 14 RETURN OF EQUAL = 'T', TIME = 1648
LEVEL 13 RETURN OF ASSOC = CONS, TIME = 1648
LEVEL 12 RETURN OF EVAL = CONS, TIME = 1664
LEVEL 12 CALL OF EVLIS('','''',CONS), TIME = 1664
LEVEL 12 RETURN OF EVLIS = '', TIME = 1681
LEVEL 11 RETURN OF EVLIS = CONS, TIME = 1698
LEVEL 11 CALL OF APPLY('ATOM',CONS,CONS), TIME = 1698
LEVEL 12 CALL OF ATOM(CONS), TIME = 1698
LEVEL 12 RETURN OF ATOM = 'F', TIME = 1714
LEVEL 11 RETURN OF APPLY = 'F', TIME = 1731
LEVEL 10 RETURN OF EVAL = 'F', TIME = 1731
LEVEL 10 CALL OF EVCON(CONS,CONS), TIME = 1748
LEVEL 11 CALL OF EVAL(CONS,CONS), TIME = 1748
LEVEL 11 RETURN OF EVAL = 'T', TIME = 1764
LEVEL 11 CALL OF EVAL(CONS,CONS), TIME = 1764
LEVEL 12 CALL OF EVLIS(CONS,CONS), TIME = 1781
LEVEL 13 CALL OF EVAL(CONS,CONS), TIME = 1781
LEVEL 14 CALL OF EVLIS(CONS,CONS), TIME = 1797
LEVEL 15 CALL OF EVAL('X',CONS), TIME = 1797
LEVEL 16 CALL OF ASSOC('X',CONS), TIME = 1814
LEVEL 17 CALL OF EQUAL('X','X'), TIME = 1814
LEVEL 18 CALL OF EQ ('X','X'), TIME = 1831
LEVEL 18 RETURN OF EQ = 'T', TIME = 1831
LEVEL 17 RETURN OF EQUAL = 'T', TIME = 1847
LEVEL 16 RETURN OF ASSOC = CONS, TIME = 1847
LEVEL 15 RETURN OF EVAL = CONS, TIME = 1864
LEVEL 15 CALL OF EVLIS('','''',CONS), TIME = 1864
LEVEL 15 RETURN OF EVLIS = ''', TIME = 1864
LEVEL 14 RETURN OF EVALIS = CONS, TIME = 1881
LEVEL 14 CALL OF APPLY ('CAR', CONS, CONS), TIME = 1881
LEVEL 14 RETURN OF APPLY = 'A', TIME = 1897
LEVEL 13 RETURN OF EVAL = 'A', TIME = 1897
LEVEL 13 CALL OF EVALIS('!', CONS), TIME = 1914
LEVEL 13 RETURN OF EVALIS = '!', TIME = 1914
LEVEL 12 RETURN OF EVALIS = CONS, TIME = 1931
LEVEL 12 CALL OF APPLY ('FF', CONS, CONS), TIME = 1951
LEVEL 13 CALL OF EVAL ('FF', CONS), TIME = 1947
LEVEL 14 CALL OF ASSOC ('FF', CONS), TIME = 1947
LEVEL 15 CALL OF EQUAL ('X', 'FF'), TIME = 1964
LEVEL 16 CALL OF EQ ('X', 'FF'), TIME = 1964
LEVEL 16 RETURN OF EQ = 'F', TIME = 1980
LEVEL 15 RETURN OF EQUAL = 'F', TIME = 1980
LEVEL 15 CALL OF ASSOC ('FF', CONS), TIME = 1980
LEVEL 16 CALL OF EQUAL ('X', 'FF'), TIME = 1997
LEVEL 17 CALL OF EQ ('X', 'FF'), TIME = 2014
LEVEL 17 RETURN OF EQ = 'F', TIME = 2014
LEVEL 16 RETURN OF EQUAL = 'F', TIME = 2030
LEVEL 16 CALL OF ASSOC ('FF', CONS), TIME = 2030
LEVEL 17 CALL OF EQUAL ('FF', 'FF'), TIME = 2047
LEVEL 18 CALL OF EQ ('FF', 'FF'), TIME = 2047
LEVEL 18 RETURN OF EQ = 'T', TIME = 2047
LEVEL 17 RETURN OF EQUAL = 'T', TIME = 2064
LEVEL 16 RETURN OF ASSOC = CONS, TIME = 2064
LEVEL 15 RETURN OF ASSOC = CONS, TIME = 2080
LEVEL 14 RETURN OF ASSOC = CONS, TIME = 2080
LEVEL 13 RETURN OF EVAL = CONS, TIME = 2080
LEVEL 13 CALL OF APPLY (CONS, CONS, CONS), TIME = 2097
LEVEL 14 CALL OF PAIRLIS (CONS, CONS, CONS), TIME = 2114
LEVEL 15 CALL OF PAIRLIS ('!', '!', CONS), TIME = 2130
LEVEL 15 RETURN OF PAIRLIS = CONS, TIME = 2130
LEVEL 14 RETURN OF PAIRLIS = CONS, TIME = 2130
LEVEL 14 CALL OF EVAL (CONS, CONS), TIME = 2147
LEVEL 15 CALL OF EVCON (CONS, CONS), TIME = 2147
LEVEL 16 CALL OF EVAL (CONS, CONS), TIME = 2164
LEVEL 17 CALL OF EVALIS (CONS, CONS), TIME = 2164
LEVEL 18 CALL OF EVAL ('X', CONS), TIME = 2180
LEVEL 19 CALL OF ASSOC ('X', CONS), TIME = 2197
LEVEL 20 CALL OF EQUAL ('X', 'X'), TIME = 2197
LEVEL 21 CALL OF EQ ('X', 'X'), TIME = 2213
LEVEL 21 RETURN OF EQ = 'T', TIME = 2213
LEVEL 20 RETURN OF EQUAL = 'T', TIME = 2230
LEVEL 19 RETURN OF ASSOC = CONS, TIME = 2230
LEVEL 18 RETURN OF EVAL = 'A', TIME = 2247
LEVEL 18 CALL OF EVALIS ('!', CONS), TIME = 2247
LEVEL 18 RETURN OF EVALIS = '!', TIME = 2247
LEVEL 17 RETURN OF EVALIS = CONS, TIME = 2263
LEVEL 17 CALL OF APPLY ('ATOM', CONS, CONS), TIME = 2263
LEVEL 18 CALL OF ATOM ('A'), TIME = 2280
LEVEL 18 RETURN OF ATOM = 'T', TIME = 2280
LEVEL 17 RETURN OF APPLY = 'T', TIME = 2280
LEVEL 16 RETURN OF EVAL = 'T', TIME = 2297
LEVEL 16 CALL OF EVAL('X',CONS), TIME = 2313
LEVEL 17 CALL OF ASSOC('X',CONS), TIME = 2313
LEVEL 18 CALL OF EQUAL('X','X'), TIME = 2330
LEVEL 19 CALL OF EQ ('X','X'), TIME = 2330
LEVEL 19 RETURN OF EQ = 'T', TIME = 2347
LEVEL 18 RETURN OF EQUAL = 'T', TIME = 2347
LEVEL 17 RETURN OF ASSOC = CONS, TIME = 2347
LEVEL 16 RETURN OF EVAL = 'A', TIME = 2363
LEVEL 15 RETURN OF EVCON = 'A', TIME = 2363
LEVEL 14 RETURN OF EVAL = 'A', TIME = 2380
LEVEL 13 RETURN OF APPLY = 'A', TIME = 2380
LEVEL 12 RETURN OF APPLY = 'A', TIME = 2396
LEVEL 11 RETURN OF EVAL = 'A', TIME = 2396
LEVEL 10 RETURN OF EVCON = 'A', TIME = 2413
LEVEL 9 RETURN OF EVCON = 'A', TIME = 2413
LEVEL 8 RETURN OF EVAL = 'A', TIME = 2413
LEVEL 7 RETURN OF APPLY = 'A', TIME = 2430
LEVEL 6 RETURN OF APPLY = 'A', TIME = 2430
LEVEL 5 RETURN OF EVAL = 'A', TIME = 2430
LEVEL 4 RETURN OF EVCON = 'A', TIME = 2446
LEVEL 3 RETURN OF EVCON = 'A', TIME = 2446
LEVEL 2 RETURN OF EVAL = 'A', TIME = 2446
LEVEL 1 RETURN OF APPLY = 'A', TIME = 2463
LEVEL 0 RETURN OF APPLY = 'A', TIME = 2480