NDPROG

A NONDETERMINISTIC PROGRAMMING LANGUAGE OF W.A. WOODS†

by

Ronald I. Becker
Department of Mathematics
University of Capetown
Cape
South Africa

and

Computational Speech and Language Processing Group
Electronics Research Laboratory
University of California at Berkeley
Berkeley, California 94720

†Research sponsored in part by Defense Advanced Research Project Agency of the Dept. of Defense and monitored by the U.S. Army Research Office under Grant DAHC04-75-G0088

TABLE OF CONTENTS

1. Foreword

This report is a description of NDPROG, a programming language for running nondeterministic programs. It was written by W.A. Woods in INTERLISP (see [1]) and has been rewritten with minor changes to run in LO UTEX LISP 1.5.9.1 (see [2]) on the CDC 6400. The program consists of a set of LISP routines which are listed in Section 6 below.

Our interest in NDPROG is primarily due to its simplicity. Most of the other nondeterministic programming languages are large and complex in their implementation. NDPROG is small and simple, which means that it is easily understood and easily implemented in various LISP systems. NDPROG is thus a good vehicle for experimenting with additional nondeterministic language features.

NDPROG is based on Woods' ATN parser (see [3]). ATN grammars can, in fact, be written as NDPROG programs. The example in 5.3 written by the author implements many of the essential features of this type of parser and provides an extensive use of the features of the program. NDPROG should provide a simple, flexible system for experimenting with various ATN parsing strategies. Another such parsing system is Ron Kaplan's GSP (see [4]).

We have had limited experience with running programs in the language and this must therefore be viewed as a preliminary report of this version. Needless to say, any faults in implementation and any faults due to changes should not be attributed to Dr. Woods.

Section 2 is an informal description and a brief introduction to nondeterministic programs. Section 3 provides a detailed description of the language and some insight into the workings of the program. Section 4 provides running instructions for the CDC 6400. Section 5 has some examples of programs, the first of which illustrates the workings of many of the functions in the packet. Section 6 has a listing of the program.

I would like to thank Michael O'Malley for suggesting this project and for his continuing interest in it.

2. Information Description of the Language

This section presents an informal description of some aspects of the language and a short introduction to nondeterministic programming. (For some references to nondeterministic programming see [5], Chapter 2.) It should be noted that certain remarks here concerning the program are only half-truths and Section 3 should be consulted for an exact description.

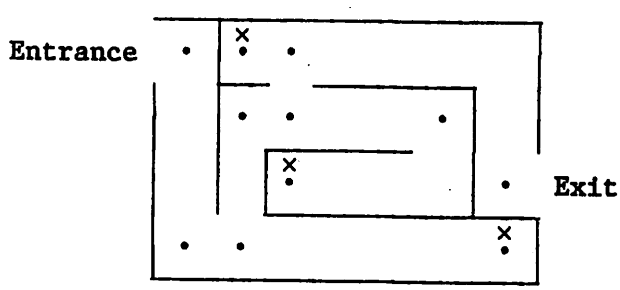
The body of a program is of the form:

```
(LABEL1  (A_1)                    .
         (A_2)                    .
           .                      .
           .                  (LABEL3  (Z_1)
           .                             .
         (A_i) )                         .
(LABEL2  (B_1)                           .
           .                           (Z_k) )
           .
           .
         (B_j) )
   .
   .
   .
```

where the (A_n) are LISP forms. The program starts by evaluting the forms of LABEL1 in turn. These may include transfers to other labels, in which case the forms of that label are evaluated in turn.

The program is nondeterministic in the following sense: There is a goal (e.g. to find the solution to a specific problem). The programmer must test at appropriate points whether the goal has been reached. If it has been attained, the program stops successfully. If the test is negative, the program continues. If a dead end is reached, the program backtracks to a point where a choice was made between several alternatives or where certain forms belonging to a node were stored for later execution. It then proceeds on the new course until success occurs or another backtrack is made. If no more alternatives remain, the program ends in failure.

<u>Example</u>. We give an example in informal language. The program will thread an arbitrary maze. We use the term "stoppoint" to denote a point in the maze at which there is either a "break" in a wall or a "dead end". (Stoppoints are marked with dots in the sketch and dead ends have a cross in addition.)



```
(INITIAL  (PASS THROUGH ENTRANCE TO FIRST STOPPOINT)
          (GO TO CHOOSE))
(CHOOSE   (IF THERE ARE BREAKS BEFORE YOU WHICH YOU HAVE NOT PASSED
          THROUGH, SELECT ONE, STORE THE REST, PASS THROUGH, WALK TO
          NEXT STOPPOINT, GO TO TEST)
          (IF THERE ARE PATHS BEFORE YOU WHICH YOU HAVE NOT TRAVERSED,
          SELECT ONE, STORE THE REST, WALK ALONG PATH TO FIRST STOPPOINT,
          GO TO TEST))
(TEST     (IF DEAD END, BACKTRACK ALONG PATH JUST TRAVERSED TO THE
          LAST STOPPOINT)
          (IF ONE OF THE BREAKS IS "EXIT" THEN SUCCESS)
          (ELSE GO TO CHOOSE))
```

Note that when backtracking occurs, the point returned to is governed by node "CHOOSE" and a choice is made according to the forms evaluated in CHOOSE. A further aid to visualization is to imagine oneself walking through the maze unravelling a ball of string.

3. Description of the Language

3.1 Remarks

We describe the format of nondeterministic programs and the various LISP functions provided for use by the programmer. The forms which the program evaluates will usually contain functions defined in the program packet. We state the arguments of each function and indicate whether they are evaluated when used in a form or not. When a program has been constructed, it may be run by using functions described in Section 3.11.

FEXPR Conventions. FEXPR's can have a list of arbitrary length as argument. However when for example only two elements a_1 and a_2 of such a list are used in the FEXPR, we adopt the convention of saying that the arguments are these two: a_1; a_2.

Calling the FEXPR for by using evalquote would require

$$FN((a_1\ a_2))\ .$$

Calling it by eval would require

$$(FN\ a_1\ a_2)\ .$$

If the program subsequently evaluates, for example, a_1 (yielding the same effect as if a_1 were evaluated and a_2 not) we will say that a_1 is evaluated and a_2 not. For a bad side effect of FEXPR use, see 3.10(b).

Notation for Functions. LISP functions will be written in a meta-notation, i.e. in small letters and underlined, e.g. fn. When used in code, we write FN (i.e. capitals).

Notation for Variables. In text, we use capital letters, e.g. VAR is used for a variable. In describing arguments of functions, we use small letters.

3.2 Program Syntax

A BNF form for the syntax of a program is:

```
<PROG> ::= (<IDENT>(LAMBDA<ARGS>(NDPROG<BODY>)))
<IDENT> ::= LISP identifier
<BODY> ::= &lt;SEGMENT&gt;+
&lt;SEGMENT&gt; ::= (<NODE><BRANCH>)
<BRANCH> ::= <EDGE>+
<NODE> ::= LISP identifier
<EDGE> ::= LISP form
<ARGS> ::= LISP list of identifiers|NIL
```

Here "+" means a string of one or more occurrences of the term. The various nodes should have distinct identifiers.

Notation. We use segment, branch, node, edge, etc. to denote the corresponding syntactic entities. (Observe, however, that in the coding, "body" is described by the variable BRANCHES.) We usually write edges as $(A_1), (A_2), \ldots, (B_1), \ldots$.

Any LISP function can be used in the LISP forms which constitute the edges. This includes SET and SETQ (which can normally be executed only in PROG's in LISP).

Example. The following is a program which illustrates the terminology. It involves no backtracking.

```
(FIVEH (LAMBDA NIL (NDPROG
     (GLOOP1  (SETR T1 0)
              (SETR T2 1)
              (TO GLOOP2))
     (GLOOP2  (SETR T1 (PLUS (GETRT1)(GETRT2)))
              (SETR T2 (ADD1 (GETRT1)))
              (TO GLOOP3))
     (GLOOP3  (IF (GREATERP (GETRT1) 500)(SUCCESS (GETRT2)))
              (SETR T2 (ADD1 (GETRT2)))
              (TO GLOOP2)))))
```

The program finds the least positive integer N such that $\sum_{n=1}^{N} n > 500$. Here, GLOOP2 is a node; ((SETR T1 0)(SETR T2 1)(TO GLOOP2)) is a branch; (TO GLOOP3) is an edge; (GLOOP1 (SETR T1 0)(SETR T2 1)(TO GLOOP2)) is a segment.

3.3 The Operation of NDPROG and STEP

(1) ndprog[segl;...;segn] FEXPR

segl,...,segn are segments (see 3.2). ndprog is the general overseer. It controls the start position; it decides which node or edge to work on next; it keeps a list of alternatives in the list ALTS; it decides when to stop computing. The start position is governed by the free variable SEQUENT (which must be given a value in a function that calls ndprog).

If SEQUENT = *T* (*T* is the value of the atom T), ndprog starts on the first edge in segl and evaluates the edges in turn. If a transfer to another node is made, it starts on the first edge of that node and continues. If a success edge is evaluated (see 3.7) the program will stop and return a value (unless, perhaps, parallel computation is underway, in which case it may continue for a while (see 3.9)). If the last edge of a segment is evaluated and it does not involve a transfer, the program backtracks by transferring to the "best" alternative on ALTS (see 3.5). This also occurs if abort (see 3.6) or suspend (see 3.5) is the function in an edge.

We define a configuration to be a list of the form (Branch Node Regs Prob Prev * T1 T2 T3 T4 T5) where Branch is a branch of the ndprog, Node is a node and the other variables will be explained in Section 3.5. (However PREV is not used in this implementation as yet.) A configuration may be thought of as representing the position of the program at some instant and has no information about past backtracking history, etc. In the coding, IC represents a configuration. If SEQUENT = a list of configurations, then ndprog starts computing from the first configuration on the list, i.e. from the first edge on Branch with the given values of Node, Regs, etc. The remaining configurations on the list are placed in ALTS.

The value returned by ndprog is important. It is a list L such that car L is the list of success values obtained and cdr L is the list of configurations remaining in ALTS. The next success value can be obtained by running ndprog with SEQUENT = cdr L. Sections 3.10 and 3.11 deal with these questions in more detail. If no success occurs, NIL is returned.

Value: List L such that car L = list of successes and cdr L = list of configurations remaining in ALTS. If there is no success, NIL is returned.

(2) step[config] EXPR

config is a configuration as described in (1). step is called by ndprog to evaluate the edges of a node. It is not usually used explicitly in user-defined programs.

Value: Returns *END*

3.4 Variables

We divide the variables that may be used by the programmer into two types and describe the use of each.

(a) Variables of Type 1. These variables are not destroyed when the program backtracks. We describe by example how to obtain and use two variables of type 1 (any number can be obtained similarly).

Example.

```
(DOSOMETHING (LAMBDA (VAR1 VAR2) (NDPROG
        (NOD (SETQ VAR1 2)
             (SETQ VAR2 VAR1))  )))
```

(This is an EXPR).

The function could be called (see 3.11) by for example

SEQUEVAL((DOSOMETHING NIL NIL) T) .

Then VAR1 and VAR2 would be NIL initially.

Note. Global variables could also be used, using CSET and CSETQ.

(b) Variables of Type 2. These variables are not preserved when the program chooses an alternative from ALTS. There are two sorts of type 2 variables:

(i) The list REGS can be used to store an unlimited number of variables and their values. Storage and retrieval are accomplished by the following functions:

(3) setr[reg;form] FEXPR

(form is evaluated, reg is not.) This adds the pair (reg. form1) to the front of the list REGS where form1 is the value of form.

Value: Returns the value of form.

(4) getr[reg] FEXPR

This gets the last value that reg was set to by setr (i.e., the CDR of the top pair in REGS whose left-hand member is reg). If reg was not previously set, the value NIL is returned.

Value: If reg was previously set by setr, the value is returned. If not, NIL is returned.

(ii) Six variables *, T1, T2, T3, T4, T5 are available for use. They may be set using SET and SETQ and their values retrieved like normal

LISP variables. The reason for having these is that REGS stores all previous settings of all its variables. If a variable is reset frequently this can use too much space. These six variables suffer from the disadvantage that it is difficult to give them mnemonic names (any method of doing so seems to involve a cost in use convenience).

Warning. Care should be taken not to use the system variables in user-defined (deterministic) subroutines. In particular, A$ - Z$ should not be used. TEMP, TEMPOR, etc. are dangerous.

3.5 The List ALTS

The variable ALTS contains a list of configurations which are the unused alternatives. (See 3.3(1) for a definition of configuration.) When the program backtracks it picks the "best" configuration in ALTS (in a sense described below) and restarts in this state.

(a) Weights. When an alternative configuration is put into ALTS, the position Prob is set to a real number. Normally the number is the current value of the variable PROB. (However see (c) and (d) below for methods of storing configurations with other weights.) Initially, PROB is set to 100. To change PROB use

(5) prob[N] EXPR

This gives PROB the value N.

Value: N

Note. When ndprog selects an alternative to backtrack to, it chooses that member of ALTS which is the "most recently set alternative of highest weight" (but see (d) below). This will be referred to as the "best alternative". Evaluating (DETOUR) does the selecting. Evaluating (ALTGEN) will place the unevaluated remainder of the current branch on ALTS with weight PROB. detour and altgen are not normally used by the programmer explicitly.

(b) Maximum and Minimum Weights. Ths variables MAXPR and MINPR store the maximum weight to which an alternative has been set during the program to date and the minimum weight, respectively. Initially, they are both set to 100. Thus MAXPR is never less than 100.

(c) Storing a Branch and then Proceeding.

(6) save[N;(A_1);...;(A_n)] FEXPR

N is evaluated if $N \neq T$; (A_1),...,(A_n) are not. This will place the branch (A_1),...,(A_n) on the altlist with Node equal to the current node, Prob equal to the value of N, if N is a real number, and remaining variables equal to their current values. If $N = T$, the weight is the current value of Prob. Execution proceeds by evaluating the form following the one with the save.

Value: The list of those alternatives to be stored during the current execution of STEP.

(d) Storing a Branch and Selecting an Alternative.

(7) suspend[N] EXPR

If a segment of the form

(SEG (A_1)$\cdots$(A_i)(SUSPEND N)(A_{i+1})$\cdots$(A_n))

is evaluated, the branch $((A_{i+1})\cdots(A_n))$ will be placed on the bottom of the ALTS list with Prob N and current values of the other variables. Then the program selects the best alternative on ALTS and starts at that configuration. Similarly for

$$(\text{IF TEST } (A_1)\cdots(A_i)(\text{SUSPEND N})(A_{i+1})\cdots(A_n))$$

and

$$(\text{TRY } \cdots) \text{ .}$$

(See 3. for <u>if</u> and <u>try</u>.)

<u>Value</u>: Returns *END.

3.6 <u>Transfer of Control</u>

(8) to[node] FEXPR

Node is the name of a node in the program. The program will next start evaluating the edges of node. It does <u>not</u> store the remaining edges of the branch in ALTS.

<u>Value</u>: *END

(8a) tol[node] EXPR

As in (8), but <u>tol</u> is an EXPR. This is useful for "computed GOTO's".

(9) abort[nil] EXPR

Transfers to the best configuration on ALTS. The remaining edges of the branch are <u>not</u> stored in ALTS.

<u>Value</u>: *END

Note. The edge (ABORT) will execute the function _abort_.

(10) resume[ic] EXPR

The argument is a list of the form

(Branch Node Regs Prob Prev *) .

When the next occasion arises for choosing a new alternative or using _to_, the program will instead resume execution at configuration ic. If a _to_ was encountered, the configuration to which _to_ transfers will be tackled after ic. Resume could be used to restart at a given Branch and Node with a different set of variables Regs, Prob. etc.

Value: Immaterial.

3.7 Successful Completion

(11) success[value] EXPR

This will cause the program to terminate when the current IC's have all had _step_ applied to them. The value of "value" will be part of the _car_ of the value returned by _ndprog_. (If there were parallel computations, this _car_ could be a list of successful values obtained.)

Value: *END

Note. A program may be capable of finding a number of successful values if allowed to use the remaining alternatives. To find all the values, _sequall_ can be used. To find the next value, _sequeval_ or _sequapply_ could be used (see 3.11).

3.8 Conditional Edges

(12) $\quad$ if[test;(A_1);...;(A_n)] $\quad$ FEXPR

The variable test is evaluated, the others not. Here test is a predicate and (A_1)···(A_n) are edges. If test does not evaluate to NIL, (A_1),...,(A_n) are evaluated and the remaining edges of the current node are stored on ALTS. The edges will normally involve some control transfer (e.g. to), for if not, the program will pick the best alternative on completing the evaluation of (A_n). If test evaluates to NIL, the next edge following *if* is evaluated, etc.

Value: ((A_1)···(A_n))

Note. A LISP *cond* can be used as well. If it is desired to execute several actions, a variant of *if* may be used as follows:

(13) $\quad$ try[test;(A_1);...;(A_n)] $\quad$ FEXPR

The variable test is evaluated, the others not. This works in the same way as *if* except that if (A_n) is evaluated and no transfer occurs, the next edge following *try* is evaluated, etc.

Value: ((A_1)···(A_n))

3.9 Parallel Computation

(14) $\quad$ split[b_1;...;b_n] $\quad$ FEXPR

Here $b_1,\ldots,b_n$ are branches. The program starts evaluating the edges on branch b_1 and continues with this path until either

(a) a <u>to</u> or <u>resume</u> edge is encountered

(b) a <u>success</u>, <u>abort</u> or <u>suspend</u> edge is encountered or a branch terminates without transfer.

The same is done for $b_2,\ldots,b_n$. If there are any branches ending as in (a), the remaining edges after the <u>split</u> edge are placed in ALTS and the computation continues as follows: The paths for those branches classified under (a) will continue in parallel until all end as in (b). If there have been any successes, these will be returned by the program and it will terminate. If not, the best alternative in ALTS is taken.

<u>Note</u>. Some parallel computation can be done using <u>resume</u> (3.6(10)) but we will not discuss this at all.

3.10 Subroutines

(a) <u>Deterministic subroutines</u> are best written in LISP, in the usual way, as functions. If the function is to be used as the function evaluated in an edge, the program can be made to take the best alternative on ALTS on completion of the edge by returning the value *END. Any other value returned will cause the program to continue with the next edge.

We devote our attention below to nondeterministic subroutines. We distinguish three types and then discuss passing variables to subroutines.

(b) <u>Subroutines Integrated Into a Calling Program</u>.

(15) ndsetr[reg;form] FEXPR

The variable form is evaluated, reg is not.

(15a) ndsetrl[reg;form;sequent] EXPR

All variables are evaluated.

Both of these are sometimes useful. (NDSETR FORM) is equivalent to (NDSETR1(QUOTE REG)FORM). However, in the latter, FORM is evaluated _before_ _ndsetr1_ is applied while in _ndsetr_, FORM is evaluated "inside" the function. (See warning at end of (b)). Suppose an edge in a program has the form

(NDSETR REG (NDFN ARG1···ARGN)) ,

where _ndfn_ is a nondeterministic function. The effect will be as follows: The first success values of _ndfn_ will be placed in register REG in the list REGS. If there are none, REG will have its previous value. Then the program places in ALTS the remaining unevaluated edges of the current branch, but headed by another _ndsetr_ edge which will start with the best alternative remaining in _ndfn_ and set REG to the next success values of _ndfn_ when its turn comes up. The edge after _ndsetr_ is then evaluated, etc. This process will continue if the alternative keeps being used, until all success values of _ndfn_ are used up. Hence the subroutine _ndfn_ is effectively integrated into the calling program. If _ndfn_ has no success values (or none remaining), the program picks the best alternative in its own ALTS and restarts there.

Value: A _list_ of first success values, if any. If not, *END.

Warning. Care must be taken to avoid the following type of error: If we define _rout_ _as a FEXPR_

```
(ROUT (LAMBDA(Z)
   (NDSETR REG (EVAL Z)) ))
```

then the edge (ROUT PROGR) will obtain the first success values of (PROGR) correctly, but it will place on the ALTS list to be executed an edge of the form

(NDSETR1 REG (EVAL Z)(restart configuration))

and if and when this is eventually evaluated, the subroutine will have exited from rout and Z will no longer have a binding. The correct effect can be obtained by using (NDSETR1 (QUOTE REG) Z T) in rout. The trouble with the first version is that eval[Z] is not evaluated before ndsetr, but internally to the latter.

(c) Non-integrated Subroutines. The user may wish to find one or several values of a nondeterministic subroutine and decide himself what to do with them and the remaining alternatives. This situation is dealt with in the next Section 3.11 (see the note there).

(d) The Case Where Form in (15) is Deterministic. Specifically, we suppose that we have a function fn which returns a list as value (e.g. the list could be a list of next states in a game). We can use (16) below to generate the states one at a time and place an ndsetr edge on the ALTS list with the remaining states. Thus

(NDSETR REG (SEQ (FN ARG1•••ARGN)))

will place the first element of the list in REG and store a generating function in ALTS. When detour picks this alternative, the next member of the list will be generated and placed in REG, etc.

(16) seq[list] EXPR

This allows the program to try a sequence of values one at a time as discussed above.

Value: Immaterial.

(e) Passing Variables to Nondeterministic Subroutines. To pass variables to nondeterministic subroutines, use:

(17) sendr[reg;form] FEXPR

Form is evaluated, reg is not. If used together with passr and initpass as explained below, sendr places the pair (reg. forml) (where forml is the value of form) in the list REGS of the next nondeterministic function evaluated by the current program.

Value: Returns the value of form.

(18) passr[ ] EXPR

The function has argument list NIL. passr should be the first edge evaluated by a nondeterministic routine to which one wishes to pass values. The subroutine should not evaluate this edge again. The list of values passed will contain all variables which have appeared in sendr edges since the last nondeterministic subroutine (if any) was evaluated. The list of values passed will appear as the initial value of REGS in the subroutine. If there were no sendr edges since the last nondeterministic subroutine, REGS will be NIL initially.

Value: The new value of REGS.

The variable SREGS is a global variable and care must be taken in backtracking, since the program will now store a copy of SREGS in ALTS. Thus there should be no backtracking between the sendr edges and the subroutine call. This is a departure from the nondeterministic philosophy and will be remedied in later versions. However, the situation is no worse than if arguments are passed by using a (LAMBDA NIL (argl,...)) in the subroutine. The present method is very flexible. SREGS must also be initialized. This can be done by calling

(19) initpass[ ] EXPR

The argument list is NIL. This initializes SREGS to NIL. Care must be taken that this instruction is not repeated an unwanted number of times. It is therefore best to include the call to evalquote

INITPASS NIL

before the nondeterministic programs are called.

Value: NIL

Example.

```
(SUBR (LAMBDA NIL (NDPROG
   (BEGIN  (PASSR)
           (TO N1))
   (N1     (SUCCESS (GETR NUMBER)))  )))
(MAINPR  (LAMBDA NIL (NDPROG
   (PASS (SENDR NUMBER 3)
         (NDSETR NUM (SUBR))
         (SUCCESS (CAR NUM)))  )))
```

Then

```
INITPASS NIL
SEQEVAL((MAINPR) *T*)
```

will have as value a list whose car is 3.

Note. The procedure can also pass variables back from subroutines, but this is perhaps best with ndsetr.

3.11 Running Nondeterministic Programs

The following functions supply a value for the free variables SEQUENT and are the analogue of LISP functions eval and apply for nondeterministic programs. (Remember that in LISP, *T* is the value of the atom T.)

(20) seqeval[form;sequent] EXPR

This evaluates form with the given value of sequent. Form should be a LISP form containing a nondeterministic program. Sequent can have value either *T* (in which case the function in form is evaluated starting at the first segment) or else sequent can be a list of configurations for the function (in which case the function is evaluated starting from the first configuration).

Value: A list L. car L is the first success value and cdr L is the list of alternative configurations after the first success.

(21) run[ndfn] EXPR

This function has a free variable SEQUENT. It causes the function ndfn to be applied to the null list (so ndfn must be a (LAMBDA NIL(··· function. It prints a list of the first successes, puts it into car SEQEUNT and puts the remaining alternatives into cdr SEQUENT.

Value: The first success of ndfn.

(22) seqall[form] FEXPR

Form should be a LISP form containing a nondeterministic program. It returns a list of all the success values the program can obtain.

Value: List of all success values.

Note. seqeval can return a list of alternatives in the cdr of its return values. These could be used in seqeval again to restart the program in an alternative configuration and look for another success. This corresponds to case (b) of (3.10).

Examples. Suppose for convenience that ndfn has a null argument list.

1) (CAR (SEQEVAL (QUOTE (NDFN)) T)) will evaluate to the first success values of ndfn.

2) (SETQ T1 (CDR (SEQEVAL (QUOTE (NDFN)) T)))
(CAR (SEQEVAL (QUOTE (NDFN)) T1))

would yield the next success, etc.

3) SEQEVAL ((RUN (QUOTE NDFN)) *T*)

3.12 Summary of System-Defined Variables and Functions

Function	Arguments	Type	Section	Number
ABORT	NIL	EXPR	3.6	9
GETR	REG	FEXPR	3.4	4
IF	TEST;(A_1);...;(A_n)	FEXPR	3.8	12
INITPASS	NIL	EXPR	3.10	19
NDPROG	SEG1;...;SEGN	FEXPR	3.3	1
NDSETR	REG;FORM†	FEXPR	3.10	15
NDSETR1	REG;FORM;SEQUENT	EXPR	3.10	15a
PASSR	NIL	EXPR	3.10	18
PROB	N	EXPR	3.5	5
RESUME	IC	EXPR	3.6	10
RUN	NDFN	EXPR	3.11	21
SAVE	N;(A_1);...;(A_n)	FEXPR	3.5	6
SEQ	LIST	EXPR	3.10	16
SEQALL	FORM	FEXPR	3.11	22
SEQEVAL	FORM;SEQUENT	EXPR	3.11	20
SENDR	REG;FORM†	FEXPR	3.10	17
SETR	REG;FORM†	FEXPR	3.4	3
SPLIT	B1;...;BN	FEXPR	3.9	14
STEP		EXPR	3.3	2
SUCCESS	VALUE	EXPR	3.7	11
SUSPEND	N	EXPR	3.5	7
TO	NODE	FEXPR	3.6	8
TO1	NODE	EXPR	3.6	8a
TRY	TEST;(A_1);...;(A_n)	FEXPR	3.8	13

† denotes evaluated. See 3.1 for conventions.

Variables	Section
BRANCH, EDGE, NODE	3.2
ALTS	3.3, 3.5
REGS	3.4
*, T1, T2, T3, T4, T5	3.4
PROB, MAXPR, MINPR	3.5
IC	3.3
SEQUENT	3.3, 3.11
SREGS	3.10

4. Running from Cards on the CDC 6400

The card sequence is as follows:

Job card (60K memory for grammar, 40K for NDPROG)
X,LISP
7-8-9
Blank card
Nondeterministic programming packet
Packet 1
Data cards
Packet 2
⋮
Packet n
Data cards
FIN
6-7-8-9

Here packet1,...,packetn are user-defined packets.

5. References

[1] Teitelman, Warren. INTERLISP Reference Manual. Xerox Palo Alto Research Center, Palo Alto, 1974.

[2] Greenwalt, E.M., Morris, James B. Jr., Singleton, Don J. The University of Texas 6400/6600 LISP 1.5. The University of Texas at Austin Computation Center.

[3] Woods, W.A. Transition network grammars for natural language analysis. CACM 13, 10 (1970) 591-606.

[4] Kaplan, R. A general syntactic processor, in Rustin, R. Natural Language Processing, Algorithmics Press, New York, 1973.

[5] Nilsson, Nils J. Problem Solving Methods in Artificial Intelligence. McGraw-Hill, New York, 1971.

[6] Woods, W.A., Kaplan, R.M. and Nash-Webber, B., The Lunar Sciences National Language Information Processing System: Final Report, BBN Report 2378. Bolt, Beranek and Newman Inc, Cambridge, Mass., June 1972.

6. Examples

6.1 Testnet

This a test program written by W.A. Woods which uses many of the functions in the program packet. If seqall is applied to testnet it returns the list of values

(TWO ONE THREE FOUR FIVE (SND.TWO)(SND.ONE)(SND.THREE)(SND.FOUR))

It is instructive to follow this program.

5.2 Queens

This program of W.A. Woods solves the 8 Queens problem. (See [5], Chapter 4, Section 6 for a discussion.) Its first solution is:

X							
						X	
				X			
							X
	X						
			X				
					X		
		X					

Time on the CDC 6400 was about 1 minute. The auxiliary function sdiff is needed. It takes the set difference between its first argument and its second.

5.3 ATN Grammars

A version of Wood's ATN parser has been programmed in the language. Only a few simple grammars have been tried and the system is still experimental and could do with some cleaning up. The main feature in the ATN grammars that has not been programmed is leftr. This can be done, but is not extensively used. We describe briefly how the system works, outlining only the different formalism from the report [6], which should be consulted for a full account of the operation. The listing following and the examples of a dictionary and grammar should be consulted. The grammar is taken from Woods [3]. No lexical analysis is done.

(a) Calling Sequence. The following sequence of calling functions should be employed.

```
INITPASS NIL
DEFINEV( dictionary entries )
     DEFINE((
(MNGRAM (LAMBDA NIL (NDPROG
(PASS (PASSR)
     (SETQ * (GETR LEX))
     (TO1 (GETR ENTPT))  )

     Grammar nodes and edges

     )))
      ))
SEQEVAL((RUN (QUOTE PARSER)) *T*)
(Data) TRACEDGE TRACEREGS
     (SENTENCE)
```

The last data item should be a single list which is the sentence to be parsed. It is read in by parser. For a discussion of TRACEDGE, TRACEREGS, see (f) below.

The sentence

(THE FOOTBALL WAS BELIEVED TO HAVE BEEN KICKED BY THE BOY)

is parsed as

```
(S DCL (NP (PRO SOMEONE))(TNS PAST)
   (VP (V (BELIEVE) (S DCL
      (NP DEF (NBOY)(NUSG))(TNS (PAST PERFECT))
         (VP (V KICK)(NP DEF (N FOOTBALL)(NUSG))))))
```

(b) Dictionary. As in [6].

(c) The grammar edges are of the following type:

$$(\text{IF FORM } (A_1)\cdots(A_n)(\text{TO LABEL2}))$$

We describe the various types of edges:

i) Cat Edges. Here FORM is a LISP form using the function

car[categ;tesv] FEXPR

tesv is evaluated, categ is not. If the current word has category categ and tesv is true it returns *T*, else NIL.

Example. (IF (CAT N T)(SETR SUBJ *)(ADVANCE)(TO NP/N))
(* and advance are explained below).

ii) Push Edges. Two functions are used here:

push[tesv] EXPR

This is used in the "tesv" position of the if statement and if tesv is non-null it allows computation of the rest of the if.

pushto[node] FEXPR

pushto regards mngram as a subroutine and makes a recursive call to it with entrypoint Node.

Example. (IF (PUSH (TRANS(GETRV)))(SENDR SUBJ *)(PUSHTO VP/)(JUMP)(TO VP/V))

Note. jump must be used with push edges.

iii) Pop Edges. The function used is

pop[tesv] EXPR

If test is true, the list in reg HOLD is essentially empty and this is an embedded computation. It returns *T*. If conditions are as above and the computation is not embedded but the sentence is at an end, *T* is returned. Else NIL is returned.

Example. (IF (POP T)(SUCCEED(NPBUILD NIL)))
A succeed form must appear on the edge.

iv) Wrd Edges.

wrd[word;tesv]   FEXPR

tesv is evaluated, word is not. If tesv is true and word = current word being scanned, it returns *T*; else NIL.

v) Mem Edges.

mem[sev;tesv]   FEXPR

If tesv is true and sev contains the current word being scanned, it returns *T*, else NIL.

vi) Vir Edges. The function used is

vir[categ;tesv]   FEXPR

tesv is evaluated, categ is not. If categ is car of an element on the list in HOLD, it returns true and deletes this element from the list HOLD. Else it returns NIL.

jump must be used with vir.

(d) Jump and Advance. Advancing the string and getting the next word must be done by hand. Immediately before a to form, either (JUMP) or (ADVANCE) must be included. The former leaves the string and word scanned as they are, the latter advances one word.

Note. push, jump and vir edges must use (JUMP). car, wrd and mem edges must use (ADVANCE). susp and sve edges could use either.

vii) Jump Edges. The function in the "tesv" position is

jmp[test] EXPR

It allows evaluation of the rest of the if edge if test is non-null.

Example. (IF (JMP (GETR))(SETR V *)(JUMP)(TO NP/))

Note. jump must be used with jmp.

viii) Suspend Edges. The function used in tesv position is

susp[N;tesv] EXPR

Suspends the rest of the unused edges corresponding to the current node with Prob the value of N, if tesv is non-null, then executes the rest of the if.

Example. (a) (IF (SUSP 80 TST))

(b) (IF (SUSP PROB TST)(SETR OBJ *)(JUMP)(TO Q1))

Note. Jump or advance could be used with susp, depending on what is done on the arc.

ix) SVE Edges. The function in tesv position is

sve[tesv] EXPR

If tesv is non-null, the rest of the *if* edge is evaluated,

Example. (IF (SVE T)$(A_1)\cdots(A_n)$(SAVE N $(A_{i+1})\cdots(A_n)$)$(A_{n+1})\cdots(A_m)$)

(e) The Variable *. This variable, of type 2, points to the current object being considered (e.g. word or phrase).

(f) The values of TRACEDGE and TRACEREGS determine whether or not tracing is done. If both are NIL, no tracing is done. If TRACEAGE is non-null, the current node and edge being scanned are tested and any jumps and advances noted. If TRACEREGS is non-null, the current value of REGS and of * are printed. If both are non-null, all of the above is printed. The first two s-expressions read in data must be the values of these variables.

(g) Other Functions. *buildq*, *getf*, etc. are the same as in ATN grammars. A number of useful functions have been omitted here (e.g. *addl*, *addr*). Most of these are easily transcribed to the current situation. *liftr*'s can be done but provide a little more trouble in programming.

```
DEFLIST((
  (IF
    (LAMBDA (A$)
      (COND
         ((EVAL (CAR A$ ))(PROG2 (ALTGEN)       (SETQ BRANCH (CDR A$))))
         (T NIL))   ))
  (GETR
    (LAMBDA (B$ )
      (ASSOC1 (CAR B$ ) REGS)))
  (NDPROG
    (LAMBDA (BRANCHES)
      (PROG (IC≠S NIC≠S ALTS NALTS IC VAL≠S MAXPR MINPR )
      (SETQ MAXPR (SETQ MINPR 100))
      (COND
         ((EQ SEQUENT T) (SETQ IC≠S
            (LIST (ICF (CDAR BRANCHES) (CAAR BRANCHES) NIL 100 NIL NIL
                       NIL NIL NIL NIL NIL      ))))
         ((SETQ ALTS (CDR SEQUENT)) (SETQ IC≠S (DETOUR)     ))
         (T (RETURN NIL)))
    LP
      (WHILE IC≠S (SETQ IC (CAR IC≠S)) (SETQ IC≠S (CDR IC≠S))
         (APPLY (FUNCTION STEP) IC))
      (SETQ ALTS (NCONC (REVERSE NALTS) ALTS))
      (SETQ NALTS NIL)
      (COND
         (NIC≠S (PROG2 (PROG2
            (SETQ IC≠S (REVERSE NIC≠S)) (SETQ NIC≠S NIL)) (GO LP)))
         (VAL≠S (RETURN (CONS VAL≠S ALTS)))
         (ALTS (PROG2 (SETQ IC≠S (DETOUR)     ) (GO LP)))
         (T (RETURN NIL)))    )))
  (NDSETR
    (LAMBDA (C$)
      (NDSETR1 (CAR C$) (CADR C$) T)))
  (NDSETR2
    (LAMBDA(D$)
      (NDSETR1 (CAR D$) (CADR D$) (CADDR D$))))
  (SAVE
    (LAMBDA (E$)
      (PROG (TEMP TEM)
      (COND
         ((EQ (CAR E$) (QUOTE T)) (SAVE1 (CAR E$)))
         ((SETQ TEMP (EVAL (CAR E$))) (PROG NIL
            (SETQ TEM PROB) (SETQ PROB TEMP) (SAVE1 (CDR E$))
            (MAXMIN PROB)
            (SETQ PROB TEM))))    )))
  (SENDR
    (LAMBDA (F$)
      (SENDR1 (CAR F$) (EVAL (CADR F$)))   ))
  (SEQALL
    (LAMBDA (G$)
      (SEQALL1 (CAR G$))  . ))
  (SETR
    (LAMBDA (H$)
      (SETR1 (CAR H$) (CADR H$))))
  (SPLIT
```

```
  (LAMBDA (I$)
    (PROG (TNIC≠S TVAL≠S)
    (SETQ TNIC≠S NIC≠S)
    (SETQ TVAL≠S VAL≠S)
    (MAPC I$ (FUNCTION (LAMBDA (X) (APPLY (FUNCTION STEP)
          (ICF X NODE REGS PROB PREV        #            T1 T2 T3 T4 T5
                 )))))
    (COND
       ((AND (EQUAL TNIC≠S NIC≠S) (EQUAL TVAL≠S VAL≠S)) T)
       (T (PROG2 (ALTGEN)     (RETURN (QUOTE *END))))))))
 (TO (LAMBDA (J$) (TO1 (CAR J$))  ))
 (TRY
   (LAMBDA (K$)
     (PROG (TEMP)
     (SETQ TEMP PROB)
     (PROB MAXPR)
     (COND ((EVAL (CAR K$)) (PROG2
           (ALTGEN)      (SETQ BRANCH (CDP K$)))))
     (PROB TEMP)    )))
 (WHILE
   (LAMBDA (L$)
     (PROG (TE$$ T$$)
     (SETQ TE$$ (CAR L$))
 L0  (COND
     ((NULL (EVAL TE$$)) (RETURN NIL)))
     (SETQ T$$ (CDR L$))
 L1  (COND
        ((NULL T$$) (GO L0)))
     (EVAL (CAR T$$))
     (SETQ T$$ (CDR T$$))
     (GO L1)    )))
     )FEXPR)
DEFINE((
  (ABORT
    (LAMBDA NIL  (PROG ( )
      (RETURN (QUOTE *END))    )))
  (ALTGEN
    (LAMBDA NIL
      (COND (BRANCH (STORALT (ICF BRANCH NODE REGS PROB PREV
                                     #          T1 T2 T3 T4 T5 ) NIL))

            (T NIL))  ))
  (ASSOC1
    (LAMBDA (A B) (COND
      ((NULL B) NIL)
      ((EQUAL A (CAAR B)) (CDAR B))
      (T (ASSOC1 A (CDR B)))    )))
  (DETOUR
    (LAMBDA NIL
      (PROG (LOC LOCW BEST BESTW VAL)
      L0
      (COND
         ((NULL ALTS) (RETURN NIL))

         ((NULL (CAR ALTS))(PROG2
            (SETQ ALTS (CDR ALTS))  (GO L0))))
      (SETQ BEST (SETQ LOC ALTS))
```

```
      (SETQ BESTW (IC/PROB (CAR ALTS)))
    L1
      (COND
         ((NULL LOC) (PROG2 (PROG2
            (SETQ VAL (CAR BEST)) (RPLACA BEST NIL))
            (RETURN (LIST VAL ))))
         ((NULL (CAR LOC)) NIL)
         ((GREATERP (SETQ LOCW (IC/PROB (CAR LOC)))  BESTW) (PROG2
            (SETQ BESTW LOCW) (SETQ BEST LOC))))
      (SETQ LOC (CDR LOC))
      (GO L1))))
  (GETNODE
    (LAMBDA (NODE)
      (ASSOC1 NODE BRANCHES)))
  (IC/PROB
    (LAMBDA(IC)
      (CADDDR IC)))
  (ICF
    (LAMBDA (BRANCH NODE REGS PROB PREV        *          T1 T2 T3
              T4 T5                            )
      (LIST  BRANCH NODE REGS PROB PREV        *          T1 T2 T3
              T4 T5                            )     ))
  (INITPASS
    (LAMBDA NIL
      (CSETQ SREGS NIL)   ))
  (MAPC
    (LAMBDA (X F)
      (COND
         ((NULL X) NIL)
         (T (PROG2 (F (CAR X)) (MAPC (CDR X) F))))))
  (MAXMIN
    (LAMBDA (N)
      (COND
         ((GREATERP N MAXPR) (SETQ MAXPR N))
         ((GREATERP N MINPR) T)
         (T (SETQ MINPR N)))   ))
  (NDSETR1
    (LAMBDA (REG FORM SEQUENT)
      (PROG ( )
      (SETQ SEQUENT (SEQEVAL FORM SEQUENT))
      (COND
         ((NULL SEQUENT) (RETURN (QUOTE *END)))
         ((CDR SEQUENT) (STORALT
            (ICF (CONS (LIST (QUOTE NDSETR2) REG FORM SEQUENT) BRANCH)
             NODE REGS PROB PREV        *         T1 T2 T3 T4 T5) NIL)))
      (RETURN (SETR1 REG (QUOTE (CAR SEQUENT))))   )))
  (PASSR
    (LAMBDA NIL
      (PROG NIL
      (SETQ REGS (NCONC SREGS REGS))
      (CSETQ SREGS NIL)
      (RETURN REGS)    )))
  (PROB
    (LAMBDA (N)
      (SETQ PROB N)))
  (RESUME
```

```
  (LAMBDA (IC)
    (PROG ( )
    (SETQ NIC≠S (CONS IC NIC≠S))
    (COND ((NULL BRANCH) (RETURN (QUOTE *END)))))))
(RUN
  (LAMBDA (NDFN)
    (PROG (SOLN TEMP)
    (SETQ TEMP (NDFN     ))
    (SETQ SOLN (CAR TEMP))
    (PRINT (QUOTE SOLUTIONS))    (TERPRI NIL)
    (PRINT SOLN)  )))
(SAVE1
  (LAMBDA (BRANCH)
    (ALTGEN)         ))
(SEQ
    (LAMBDA (LIST)
    (COND
       ((EQUAL SEQUENT 1) LIST)  (T (CDR SEQUENT)))))
(SENDR1
  (LAMBDA (REG FORM)
    (PROG (TEMP$)
    (CSETQ SREGS (CONS (CONS REG (SETQ TEMP$        FORM )) SREGS))
    (RETURN TEMP$)    )))
(SEQALL1
  (LAMBDA(FORM)
    (PROG(SEQUENT TEMP)
    (SETQ SEQUENT T)
    (WHILE (SETQ SEQUENT (SEQEVAL FORM SEQUENT))
                             (SETQ TEMP (NCONC TEMP (CAR SEQUENT))))
    (RETURN TEMP))))
(SEQEVAL
  (LAMBDA (FORM SEQUENT)
    (EVAL FORM)))
(SETR1
  (LAMBDA (REG FORM)  (PROG (TEM$)
    (SETQ REGS (CONS (CONS REG (SETQ TEM$ (EVAL FORM))) REGS))
    (RETURN TEM$))))
(STEP
  (LAMBDA (BRANCH NODE REGS PROB PREV
                 *           T1 T2 T3 T4 T5)
    (PROG (EDGE)
  LO
    (COND ((NULL BRANCH) (RETURN (QUOTE *END*))))
    (SETQ EDGE (CAR BRANCH))
    (SETQ BRANCH (CDR BRANCH))
    (COND
       ((EQUAL (EVAL EDGE) (QUOTE *END)) (GO END)))
    (GO LO)
END
    (RETURN (QUOTE *END*))  )))
(STORALT
  (LAMBDA (ALT NFLAG)
    (COND
       (NFLAG (SETQ ALTS (NCONC ALTS (LIST ALT ))))
       (T (SETQ NALTS (CONS ALT NALTS))))))
(SUCCESS
```

```
    (LAMBDA (VALUE)
      (PROG ( )
      (SETQ VAL≠S (NCONC VAL≠S (LIST VALUE)))
      (RETURN (QUOTE *END)))))
  (SUSPEND
    (LAMBDA (N)
      (PROG ( )
      (MAXMIN N)
      (STORALT (ICF BRANCH NODE REGS N PREV        *           T1 T2 T3
                  T4 T5                      ) T)
      (RETURN (QUOTE *END)) )))
  (TO1
    (LAMBDA (ND)
      (PROG ( )
       (SETQ NIC≠S (CONS (ICF (GETNODE (CAR ND)) (CAR ND) REGS PROB
               PREV      *          T1 T2 T3 T4 T5              ) NIC≠S))
      (RETURN (QUOTE *END)))))
      ))
STOP))))))))))))

DEFINE((
  (TESTNET
    (LAMBDA NIL
      (NDPROG
      (BEGIN    (SETR REGISTER (QUOTE ONE))
                (SAVE 90 (TO END))
                (SETR REGISTER (QUOTE TWO))
                (TRY T (TO END))
                (SUSPEND 80)
                (SPLIT ((SETR REGISTER (QUOTE THREE)) (TO END))
                       ((SETR REGISTER (QUOTE FOUR)) (TO END)))
                (IF T (SUCCESS (QUOTE FIVE))))
      (END      (IF T (SUSPEND 60) (SUCCESS (CONS (QUOTE SND)
                                       (GETR REGISTER))))
                (SUCCESS (GETR REGISTER))))))
      ))
STOP))))))))))))

DEFINE((
  (SDIFF
    (LAMBDA (A B)  (COND
      ((NULL A) NIL)
      ((MEMBER (CAR A) B) (SDIFF (CDR A) B))
      (T (CONS (CAR A) (SDIFF (CDR A) B))) )))
  (QUEENS
   (LAMBDA NIL
      (NDPROG
         (START (SETR COL 1)
                (TO GENERATE))
         (GENERATE (NDSETR ROW (SEQ (SDIFF (QUOTE (1 2 3 4 5 6 7 8))
                                                 (GETR ROWS))))
                   (TO CHECK))
         (CHECK (SETR D1 (PLUS (GETR COL) (GETR ROW) (MINUS 1)))
                (SETR D2 (PLUS (GETR COL) 8 (MINUS (GETR ROW))))
                (COND
                  ((OR (MEMBER (GETR D1) (GETR DIAG1))
```

```
                    (MEMBER (GETR D2) (GETR DIAG2)))
                  (ABORT)) (T T))
              (SETR ROWS (CONS (GETR ROW) (GETR ROWS)))
              (SETR DIAG1 (CONS (GETR D1) (GETR DIAG1)))
              (SETR DIAG2 (CONS (GETR D2) (GETR DIAG2)))
              (SETR SOLN (CONS (CONS (GETR COL) (GETR ROW))
                                        (GETR SOLN)))
              (COND
                 ((EQUAL (GETR COL) 8) (SUCCESS (GETR SOLN)))
                 (T (SETR COL (ADD1 (GETR COL)))))
              (TO GENERATE)))))
       ))
STOP))))))))))))

DEFLIST((
   (CAT (LAMBDA (N$) (CAT1 (CAR N$) (EVAL (CADR N$)))   ))
   (MEM (LAMBDA (R$) (MEM1 (CAR R$) (CADR R$))   ))
   (PUSHTO
     (LAMBDA (S$)
       (PROG NIL
       (SENDR STRING (GETR STRING))
       (SENDR LEX (GETR LEX))
       (SENDR EMBED T)
       (SENDR ENTPT (CAR S$))
       (COND ((EQ (NDSETR * (MNGRAM)) (QUOTE *END))
                               (RETURN (QUOTE *END))))
       (PASSR)
       (SETQ * (CAR (GETR *)))
       (RETURN *)    )))
   (VIR (LAMBDA (T$) (VIR1 (CAR T$) (EVAL (CADR T$)))   ))
   (WRD (LAMBDA (V$) (WRD1 (CAR V$) (EVAL (CADR V$)))   ))
       )FEXPR)
DEFINE((

   (ADVANCE
     (LAMBDA NIL
       (PROG (TE   )
       (COND (TRACEDGE (PROG2 (TERPRI NIL) (PRINT (QUOTE ADVANCING)))))
       (SETQ TE   (GETR STRING))
       (COND
          ((NULL TE) (RETURN (QUOTE *END)))
          ((NULL (CDR TE)) (SETQ * (SETR LEX NIL)))
          (T (SETQ * (SETR LEX (CADR TE)))))
       (SETR STRING (CDR TE  ))   )))
   (CAT1
     (LAMBDA (CAT TEST)
       (PROG (TEMP)
       (TRACES)
       (COND ((NULL TEST) (RETURN NIL)))
       (COND
          ((SETQ TEMP (DICTCHECK (GETR LEX) CAT)) (PROG2 (PROG NIL
                (SETQ * (CAAR TEMP)) (SETR FEATURES (CDAR TEMP)))
                (RETURN (CAR TEMP))))
          (T (RETURN NIL)))   )))
   (JMP
     (LAMBDA (TEST)
```

```
      (PROG NIL
      (TRACES)
      (COND (TEST (RETURN T)))    )))
  (JUMP
    (LAMBDA NIL
      (PROG NIL
      (COND (TRACEDGE (PROG2 (TERPRI NIL) (PRINT (QUOTE JUMPING)))))
      (SETQ * (GETR LEX))  )))
  (MEM1
    (LAMBDA (LIST TEST)
      (PROG2
      (TRACES)
      (COND
         ((NULL TEST) NIL)
         ((MEMBER * LIST) T)
         (T NIL))    )))
  (PARSER
    (LAMBDA NIL
      (NDPROG
(PARSE
      (CSETQ TRACEDGE (READ NIL))
      (CSETQ TRACEREGS (READ NIL))
      (SENDR LEX (CAR (SETR STRING (SENDR STRING (READ NIL)))))
      (SENDR ENTPT (QUOTE S/))
      (NDSETR PARSES (MNGRAM))
      (TERPRI NIL)
      (PRINT (QUOTE SENTENCE)) (TERPRI NIL)
      (PRINT (GETR STRING))
      (TERPRI NIL)  (TERPRI NIL)
      (PRINT (QUOTE (PARSES OF SENTENCE)))
      (TERPRI NIL)
      (PRINTLIST (CAR (GETR PARSES)))
      (TERPRI NIL) (TERPRI NIL)
      (SUCCESS (QUOTE DONE)))   )))
  (POP
    (LAMBDA (TEST)
      (PROG NIL
      (TRACES)
      (COND ((NULL TEST) (RETURN NIL)))
      (COND ((NULLIS (GETR HOLD)) (SETR HOLD NIL))
            (T (RETURN NIL)))
      (COND ((AND (NULL (GETR EMBED)) (GETR STRING)) (RETURN NIL)))
      (COND ((GETR EMBED) (PROG2
                (SENDR STRING (GETR STRING))
                (SENDR LEX (GETR LEX)))))
       (RETURN T)    )))
  (PUSH
    (LAMBDA (TEST)
      (PROG NIL
      (TRACES)
      (COND (TEST (RETURN T)))    )))
  (SVE
    (LAMBDA (TEST)
      (PROG NIL
      (TRACES)
      (COND (TEST (RETURN T)))    )))
```

```
  (SUSP
      (LAMBDA (N TEST)
      (PROG NIL
      (TRACES)
      (COND (TEST (PROG2 (SUSPEND N) (RETURN T))))   )))
  (VIR1
    (LAMBDA (CAT TST)
      (PROG (TM HLIST)
      (TRACES)
      (SETQ HLIST (GETR HOLD))
   L1
      (COND ((OR (NULL HLIST)(EQUAL HLIST (QUOTE (NIL)))) (RETURN NIL)))
      (COND ((OR (NULL (CAR HLIST)) (EQUAL (CAR HLIST) (QUOTE (NIL))))
             (PROG2 (SETQ HLIST (CDR HLIST)) (GO L1))))
      (SETQ TM (CAR HLIST))
      (COND
         ((AND TM (EQ (CAAR TM) CAT) TST) (PROG2 (PROG NIL
                (SETR HOLD (KILL TM (GETR HOLD)))
                (SETQ * (CAR TM))
                (SETR FEATURES (CDR TM))) (RETURN TM)))
         ((SETQ HLIST (CDR HLIST)) (GO L1))
         (T (RETURN NIL)))   )))
  (WRD1
    (LAMBDA (WORD TEST)
      (PROG2
      (TRACES)
      (COND
         ((NULL TEST) NIL)
         ((EQ * WORD) T)
         (T NIL))   )))
      ))
STOP))))))))))))

DEFLIST((
  (BUILDQ  (LAMBDA (M$) (BUILD  M$)))
  (DEFINEV
    (LAMBDA (P$)
      (MAP P$ (FUNCTION (LAMBDA (X) (CSET (CAAR X) (CDAR X)))))   ))
  (GETF (LAMBDA (Q$) (GETF1 (CAR Q$))))
      )FEXPR)
DEFINE((
  (APPEND1
    (LAMBDA (X Y)
      (COND ((NULL X) Y)
            (T (CONS (CAR X) (APPEND1 (CDR X) Y))))))
  (BUILD  (LAMBDA (ARGS) (PROG (X)
      (SETQ X (CAR ARGS))
      (SETQ ARGS (CDR ARGS))
      (RETURN(BUILD1 X)))))
  (BUILD1 (LAMBDA (X) (COND
      ((EQ X (QUOTE *) ) *)
      ((EQ X (QUOTE +))(PROG NIL
                        (SETQ X (CAR ARGS))
                        (SETQ ARGS (CDR ARGS))
                     (RETURN (ASSOC1 X REGS))))
      ((EQ X (QUOTE =))(PROG NIL
```

```
                          (SETQ X (CAR ARGS))
                          (SETQ ARGS (CDR ARGS))
                    (RETURN (EVAL X))))
    ( (NOT (LISTP X)) X)
    ((EQ (CAR X) (QUOTE $))
                (MAPCONC (CDR X) (FUNCTION (LAMBDA (Y)
                   (APPEND1 (BUILD1 Y) NIL)))))
    (T (BUILD2 X)))))
(BUILD2 (LAMBDA (X) (COND
    ((NULL X) NIL)
    ((NOT (LISTP X)) (BUILD1 X))
    (T
    (CONS (BUILD1 (CAR X)) (BUILD2 (CDR X) ))))))
(DICTCHECK
  (LAMBDA (LEX CAT)
    (PROG (DICTFORM)
    (COND
       ((NULL (SETQ DICTFORM (GET (EVAL LEX) CAT))) (RETURN NIL))
       ((ATOM DICTFORM) (GO L1))
       ((ATOM (CAR DICTFORM)) (RETURN (LIST DICTFORM)))
       (T (RETURN DICTFORM)))
 L1
    (COND
       ((EQ CAT (QUOTE N)) (RETURN (LIST
          (SELECT DICTFORM
             ((QUOTE REG)(CONS LEX (QUOTE ((NUMBER SG)))))
             ((QUOTE   ES    ) (CONS LEX      (QUOTE ((NUMBER SG)))))
             ((QUOTE   IES   ) (CONS LEX      (QUOTE ((NUMBER SG)))))
             ((QUOTE   IRR   ) (CONS LEX      (QUOTE ((NUMBER SG)))))
             ((QUOTE   MASS  ) (CONS LEX      (QUOTE ((NUMBER SG)))))
             ((QUOTE   S     ) (CONS LEX      (QUOTE ((NUMBER SG)))))
             (CONS DICTFORM (QUOTE ((NUMBER SG))))
            ))))
       ((EQ CAT (QUOTE V)) (RETURN (LIST (CONS LEX (QUOTE((TNS
                              PRESENT) (PNCODE X3SG) (UNTENSED)))))))
       ((EQ CAT (QUOTE ADJ)) (RETURN (LIST (LIST  LEX ))))
       ((EQ DICTFORM * ) (RETURN (LIST (LIST  LEX ))))
       (T (RETURN (LIST (LIST DICTFORM)))))    )))
(GET1 (LAMBDA (L P) (COND
    ((NULL L ) NIL)
    ((EQ (CAR L) P) (CDR L))
    (T (GET1 (CDR L) P))   )))
(GETF1
  (LAMBDA (FEATURE)
    (PROG (TEMP)
    (COND
       ((NULL (SETQ TEMP (CAR (ASSOC1 FEATURE (GETR FEATURES)))))
                       (RETURN NIL))
       (T (RETURN TEMP)))    )))
(HOLD
  (LAMBDA (FORMS$ FEATURES$)
    (SETR HOLD (CONS (CONS FORMS$ FEATURES$) (GETR HOLD)))))
(INTRANS(LAMBDA(V) (MEMBER (QUOTE INTRANS)
                                   (GET1 (EVAL V) (QUOTE FEATURES)))))
(KILL (LAMBDA (X Y) (COND
    ((NOT (LISTP Y)) Y)
```

```
      ((EQ (CAR Y)X) (CDR Y))
      (T (CONS (CAR Y) (KILL X (CDR Y)))))))
  (LISTP
    (LAMBDA (X)
      (AND (NOT (NUMBERP X)) (NOT (ATOM X)))    ))
  (MAPCONC (LAMBDA (X F) (COND
      ((NULL X) NIL) (T (NCONC (F (CAR X)) (MAPCONC (CDR X) F))) )))
  (MODAL (LAMBDA (A)
      (MEMBER * (QUOTE (DO WILL  MMODAL SHALL CAN MAY MUST)))))
  (NPBUILD
    (LAMBDA NIL
      (PROG (TEMP)
      (SETQ TEMP (BUILDQ          ($ (NP) = = (+ (NU +)) = =)
         (COND
            ((GETR DET) (LIST (GETR DET)))
            (T NIL))
         (REVERSE (GETR ADJS))
      N  NU  (REVERSE (GETR NMODS))
      (COND
         ((GETR NR) (BUILDQ         ($ (NR) = ) (REVERSE (GETR NR))))
         (T NIL))))
      (RETURN (COND ((GETR NEG) (BUILDQ         (NP + =) NEG TEMP))
                    (T TEMP))))))
  (NULLIS
      (LAMBDA (L)
      (COND
         ((NULL L) T)
         ((AND (CAR L) (NOT (EQUAL (CAR L) (QUOTE (NIL)))))   NIL)
         (T (NULLIS (CDR L))))))
  (PRINTLIST ,
      (LAMBDA (LIST)
      (COND
         ((NULL LIST) (PRINT NIL))
         ((NULL (CDR LIST)) (PRINT (CAR LIST)))
         (T (PROG2 (PRINT (CAR LIST)) (PRINTLIST (CDR LIST)))))    ))
  (S-TRANS (LAMBDA (V) (MEMBER (QUOTE S-TRANS) (GET1 (EVAL V)
                 (QUOTE FEATURES))    )))
  (TRANS(LAMBDA(X) (PROG (TEMP)
      (RETURN (OR (NOT (SETQ TEMP (GET1 (EVAL X) (QUOTE FEATURES))))
            (MEMBER (QUOTE TRANS) TEMP))))))
  (TRACES
    (LAMBDA NIL
      (PROG NIL
      (COND (TRACEDGE (PROG NIL  (TERPRI NIL) (PRINT (QUOTE TRYING))
            (TERPRI NIL) (PRIN1 (QUOTE *NODE)) (PRIN1 BLANK)
            (PRINT NODE) (TERPRI NIL) (PRINT EDGE))))
      (COND (TRACEREGS (PROG NIL (TERPRI NIL) (PRINT REGS) (TERPRI NIL)
            (PRIN1 (QUOTE ≠*)) (PRIN1 BLANK) (PRIN1 (QUOTE =))
             (PRIN1 BLANK) (PRINT *))))
      )))
      ))
STOP))))))))))))

DEFINE((
   (MNGRAM
     (LAMBDA NIL
```

```
        (NDPROG
 (PASS    (PASSR)
        (SETQ * (GETR LEX))
        (TO1 (GETR ENTPT))
        )
 (S/    (IF (CAT AUX T)
        (SETR V *) (SETR TNS (GETF TNS))
           (SETR TYPE (QUOTE Q)) (ADVANCE) (TO Q1))
        (IF (PUSH T)
           (PUSHTO NP/)
           (SETR SUBJ *) (SETR TYPE (QUOTE DCL))  (JUMP) (TO Q2)))
 (Q1    (IF (PUSH T)
           (PUSHTO NP/)
           (SETR SUBJ *)  (JUMP)   (TO Q3)))
 (Q2    (IF (CAT V T)
           (SETR V *) (SETR TNS (GETF TNS)) (ADVANCE) (TO Q3)))
 (Q3    (IF (CAT V (AND (GETF PASTPART) (EQ (GETR V) (QUOTE BE))))
           (HOLD (GETR SUBJ) NIL) (SETR SUBJ (BUILDQ (NP (PRO SOMEONE))))
           (SETR AGFLAG T) (SETR V *) (ADVANCE) (TO Q3))
        (IF (CAT V (AND (GETF PASTPART) (EQ (GETR V) (QUOTE HAVE))))
           (SETR TNS (LIST (GETR TNS) (QUOTE PERFECT)))
           (SETR V *) (ADVANCE) (TO Q3))
        (IF (PUSH (TRANS (GETR V)))
           (PUSHTO NP/)
           (SETR OBJ *)  (JUMP)   (TO Q4))
        (IF (VIR NP (TRANS (GETR V)))
           (SETR OBJ *)   (JUMP)  (TO Q4))
        (IF (POP (INTRANS (GETR V)))
           (SUCCESS (BUILDQ (S + + (TNS +) (VP (V +))) TYPE SUBJ TNS V))))
 (Q4    (IF (WRD BY (GETR AGFLAG))
           (SETR AGFLAG NIL) (ADVANCE) (TO Q7))
        (IF (WRD TO (S-TRANS (GETR V)))
           (ADVANCE) (TO Q5))
        (IF (POP T)
           (SUCCESS (BUILDQ (S + + (TNS +) (VP (V +) +)) TYPE SUBJ TNS V
                                          OBJ))))
 (Q5    (IF (PUSH T)
           (SENDR SUBJ (GETR OBJ)) (SENDR TNS (GETR TNS))
           (SENDR TYPE (QUOTE DCL))
           (PUSHTO VP/)
           (SETR OBJ *) (JUMP) (TO Q6)))
 (Q6    (IF (WRD BY (GETR AGFLAG))
           (SETR AGFLAG NIL) (ADVANCE) (TO Q7))
        (IF (POP T)
           (SUCCESS (BUILDQ (S + + (TNS +) (VP (V +) +)) TYPE SUBJ TNS V
                                          OBJ)))).
 (Q7    (IF (PUSH T)
           (PUSHTO NP/)
           (SETR SUBJ *)  (JUMP)   (TO Q6)))
 (VP/   (IF (CAT V (GETF UNTENSED)) (SETR V *) (ADVANCE) (TO Q3)))
 (NP/
        (IF (CAT DET T)
           (SETR DET *) (ADVANCE) (TO NP/DET))
        (IF (JMP T)
           (JUMP) (TO NP/DET))   )
 (NP/DET
```

```
      (IF (CAT N T)
         (SETR N (BUILDQ (N #)))
         (SETR NU (GETF NUMBER))
         (ADVANCE)
         (TO NP/N)))
(NP/N
      (IF (CAT N T)
         (SETR ADJS (BUILDQ (ADJ +) N))
         (SETR N (BUILDQ (N #)))
         (SETR NU (GETF NUMBER)) (ADVANCE) (TO S/POP))
      (IF T
         (SETR SUBJ (CADR (GETR N))) (JUMP) (TO S/POP)))
(S/POP
      (IF (POP T)
         (SUCCESS (NPBUILD))))
      )))
      ))
      DEFINEV(
(A          DET INDEF)
(BE V (BE (TNS PRES) (UNTENSED T)))
(BEEN V (BE  (PASTPART T)  (TNS PAST) (PNCODE ANY)))
(BELIEVE V FEATURES TRANS S-TRANS)
(BELIEVED V (BELIEVE (TNS PAST) (PASTPART T) (PNCODE ANY)) FEATURES
                          TRANS   S-TRANS)
(BOY        N S  (BOY (NU SG)))
(BOYS       N    (BOY (NU PL)))
(BY PREP)
(FOOTBALL    N S    (FOOTBALL   (NU SG)))
(FOOTBALLS   N      (FOOTBALL   (NU PL)))
(GREEN        ADJ           #)
(HAD        V       (HAVE (TNS PAST) (PASTPART T)  (PNCODE ANY)))
(HARD         ADJ           #)
 (HAS         V       (HAVE  (TNS PRES)              (PNCODE P3SG))
                    FEATURES (TRANS PASSIVE INTRANS))
(HAVE         V       (HAVE  (TNS PRES) (UNTENSED T) (PNCODE X3SG)))
(KICK       V        (KICK (TNS PRES) (UNTENSED T) (PNCODE X3SG))
            FEATURES TRANS          )
(KICKED     V        (KICK (TNS PAST) (PASTPART T) (PNCODE  ANY)))
(KICKS      V        (KICK (TNS PRES)              (PNCODE P3SG)))
(SOMETIMES  ADV IRR                                 )
(THE        DET   DEF)
(TO PREP)
(WAS V (BE (TNS PAST) (PASTPART T) (PNCODE ANY)))
(WILL       V       (WILL (TNS PRES)               (PNCODE ANY)))
(WOULD      V       (WILL (TNS PAST) (PASTPART T)  (PNCODE ANY)))
        )
STOP))))))))))
```