APPENDIX A

ATTRIBUTE GRAMMAR RULES AND TABLES

ORIGINAL ATTRIBUTE GRAMMAR RULES

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;     Context-free grammar productions and translation rules
;     in the original form.  In the file "setrules.lsp".
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(setf att_rules '#(
   ((*adj -> (VX))            (((tr 0) 1)))
   ((*and -> (VX))            (((tr 0) 1)))
   ((*arith_command -> (VX))  (((tr 0) 1)))
   ((*be_verb -> (VX))             (((tr 0) 1)))
   ((*between -> (VX))             (((tr 0) 1)))
   ((*command -> (VX))             (((tr 0) 1)))
   ((*comp -> (VX))           (((tr 0) 1)))
   ((*cred -> (VX))           (((tr 0) 1)))
   ((*cs -> (VX))             (((tr 0) 1)))
   ((*csp -> (VX))            (((tr 0) 1)))
   ((*det -> (VX))            (((tr 0) 1)))
   ((*hrs -> (VX))            (((tr 0) 1)))
   ((*no -> (VX))             (((tr 0) 1)))
   ((*ns -> (VX))             (((tr 0) 1)))
   ((*num -> (VX))            (((tr 0) 1)))
   ((*or -> (VX))             (((tr 0) 1)))
   ((*prep -> (VX))           (((tr 0) 1)))
   ((*than -> (VX))           (((tr 0) 1)))
   ((*vno -> (VX))            (((tr 0) 1)))
   ((*what -> (VX))           (((tr 0) 1)))

   ((ADJP -> (*ADJ))          (((TR 0) (TR 1))))
   ((ARITH_IMP -> (*ARITH_COMMAND NPS)) (((TR 0) (FIND_FUNC (TR 1) (TR 2)))))
   ((CN_SEQ -> (*CSP *VNO SEQUENCE))  (((TRA 0) (TRA 3))
                                       ((COA 0) (COA 3))
                                       ((TRB 0) (TRB 3))
                                       ((COB 0) (COB 3))
                                       ((CONJ 0) (CONJ 3))))
   ((CNPS -> (CSE *NUM))              (((TR 0) (TR 2))
                                       ((CO 0) '=)))
   ((CONJUNC -> (*AND))               (((TR 0) (TR 1))))
   ((CONJUNC -> (*OR))                (((TR 0) (TR 1))))
   ((CPP -> (*PREP CN_SEQ))           (((TRA 0) (TRA 2))
                                       ((COA 0) (COA 2))
                                       ((TRB 0) (TRB 2))
                                       ((COB 0) (COB 2))
                                       ((CONJ 0) (CONJ 2))))
   ((CPP -> (*PREP CNPS))             (((TRA 0) (TR 2))
                                       ((COA 0) (CO 2))
                                       ((CONJ 0) 'EMPTY)
                                       ((TRB 0) 'EMPTY)
```

```
                                        ((COB 0) 'EMPTY)))
    ((CSE -> (*CS *NO))                ((TR 0) CNO)))
    ((IMP -> (*COMMAND WHQNP))         ((TR 0) (TR 2))))
    ((NPS -> (*DET ADJP CSE CPP))      (((VAR 0) (GENSYM))
                                        ((TR 0) (NPSINTERP
                                                   (VAR 0)
                                                   (TR 1)
                                                   (TR 2)
                                                   (TR 3)
                                                   (COA 4)
                                                   (TRA 4)
                                                   (CONJ 4)
                                                   (COB 4)
                                                   (TRB 4)))))
    ((NPS -> (*DET NPS1 CPP))          (((VAR 0) (GENSYM))
                                        ((TR 0) (NPSINTERP
                                                   (VAR 0)
                                                   (TR 1)
                                                   NIL
                                                   (TR 2)
                                                   (COA 3)
                                                   (TRA 3)
                                                   (CONJ 3)
                                                   (COB 3)
                                                   (TRB 3)))))
    ((NPS1 -> (*CRED *HRS))            (((TR 0) CRED_HRS)))
    ((NPS1 -> (*NS))                   (((TR 0) (TR 1))))
    ((S -> (ARITH_IMP))                (((TR 0) (TR 1))))
    ((S -> (IMP))                      (((TR 0) (TR 1))))
    ((S -> (WHQ WHQNP))                (((TR 0) (TR 2))))
    ((SEQ -> (*COMP *THAN *NUM))       (((TR 0) (TR 3))
                                        ((CO 0) (COMPARE
                                                   (TR 1)
                                                   NIL) )))
    ((SEQ -> (*NUM *OR *COMP))         (((TR 0) (TR 1))
                                        ((CO 0) (COMPARE (TR 3) "=") )))
    ((SEQUENCE -> (*BETWEEN *NUM *AND *NUM))
                                       (((TRA 0) (LESSERNUM (TR 2) (TR 4)))
                                        ((COA 0) '>=)
                                        ((TRB 0) (GREATERNUM (TR 2) (TR 4)))
                                        ((COB 0) '<=)
                                        ((CONJ 0) (TR 3))))
    ((SEQUENCE -> (SEQ))               (((TRA 0) (TR 1))
                                        ((COA 0) (CO 1))
                                        ((TRB 0) 'EMPTY)
                                        ((COB 0) 'EMPTY)
                                        ((CONJ 0) 'EMPTY)))
    ((SEQUENCE -> (SEQ CONJUNC SEQ)) (((TRA 0) (TR 1))
                                       ((COA 0) (CO 1))
                                        ((TRB 0) (TR 3))
                                        ((COB 0) (CO 3))
                                        ((CONJ 0) (TR 2))))
    ((START -> (S))                    (((TR 0) (TR 1))))
    ((WHQ -> (*WHAT *BE_VERB))         (((TR 0) (TR 1))))
    ((WHQNP -> (NPS))                  (((TR 0) (TR 1))))
    ((WHQNP -> (*DET *ARITH_COMMAND *PREP NPS))
                                       (((TR 0) (FIND_FUNC (TR 2) (TR 4)))))
))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;     Attributes used in the semantic rules above.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(setf (get 'tr 'attribute) 't)
(setf (get 'tra 'attribute) 't)
(setf (get 'trb 'attribute) 't)
(setf (get 'var 'attribute) 't)
(setf (get 'co 'attribute) 't)
(setf (get 'coa 'attribute) 't)
(setf (get 'cob 'attribute) 't)
(setf (get 'conj 'attribute) 't)
```

CONTEXT-FREE GRAMMAR RULES

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;     Context-free grammar productions used by Tomita's
;     procedures, after the conversion from the original
;     attribute grammar rules by the procedure converttree.
;     This rules are bound to the array rules.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

0 (*adj -> (VX))
1 (*and -> (VX))
2 (*arith_command -> (VX))
3 (*be_verb -> (VX))
4 (*between -> (VX))
5 (*command -> (VX))
6 (*comp -> (VX))
7 (*cred -> (VX))
8 (*cs -> (VX))
9 (*csp -> (VX))
10 (*det -> (VX))
11 (*hrs -> (VX))
12 (*no -> (VX))
13 (*ns -> (VX))
14 (*num -> (VX))
15 (*or -> (VX))
16 (*prep -> (VX))
17 (*than -> (VX))
18 (*vno -> (VX))
19 (*what -> (VX))
20 (ADJP -> (*ADJ))
21 (ARITH_IMP -> (*ARITH_COMMAND NPS))
22 (CN_SEQ -> (*CSP *VNO SEQUENCE))
23 (CNPS -> (CSE *NUM))
24 (CONJUNC -> (*AND))
25 (CONJUNC -> (*OR))
26 (CPP -> (*PREP CN_SEQ))
27 (CPP -> (*PREP CNPS))
28 (CSE -> (*CS *NO))
29 (IMP -> (*COMMAND WHQNP))
30 (NPS -> (*DET ADJP CSE CPP))
31 (NPS -> (*DET NPS1 CPP))
32 (NPS1 -> (*CRED *HRS))
33 (NPS1 -> (*NS))
34 (S -> (ARITH_IMP))
35 (S -> (IMP))
36 (S -> (WHQ WHQNP))
37 (SEQ -> (*COMP *THAN *NUM))
38 (SEQ -> (*NUM *OR *COMP))
39 (SEQUENCE -> (*BETWEEN *NUM *AND *NUM))
40 (SEQUENCE -> (SEQ))
41 (SEQUENCE -> (SEQ CONJUNC SEQ))
42 (START -> (S))
43 (WHQ -> (*WHAT *BE_VERB))
44 (WHQNP -> (NPS))
45 (WHQNP -> (*DET *ARITH_COMMAND *PREP NPS))
```

LR PARSING ACTION TABLE

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;     LR parsing action table used by Tomita's parsing
;     procedures.  In the file "tbl1.lsp"
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(setf table1 '#(
        ((*ARITH_COMMAND (S 5)) (*COMMAND (S 6)) (*WHAT (S 7)))
        (($ (A)))
        ((@ALL (R 34)))
        ((@ALL (R 35)))
        ((*DET (S 10)))
        ((*DET (S 12)))
        ((*DET (S 10)))
        ((*BE_VERB (S 14)))
        ((@ALL (R 36)))
        ((@ALL (R 44)))
        ((*ARITH_COMMAND (S 15)) (*CRED (S 18)) (*NS (S 19))
                (*ADJ (S 20)))
        ((@ALL (R 21)))
        ((*CRED (S 18)) (*NS (S 19)) (*ADJ (S 20)))
        ((@ALL (R 29)))
        ((@ALL (R 43)))
        ((*PREP (S 21)))
        ((*PREP (S 23)))
        ((*CS (S 25)))
        ((*HRS (S 26)))
        ((@ALL (R 33)))
        ((@ALL (R 20)))
        ((*DET (S 12)))
        ((@ALL (R 31)))
        ((*CSP (S 30)) (*CS (S 25)))
        ((*PREP (S 23)))
        ((*NO (S 33)))
        ((@ALL (R 32)))
        ((@ALL (R 45)))
        ((@ALL (R 27)))
        ((@ALL (R 26)))
        ((*VNO (S 34)))
        ((*NUM (S 35)))
        ((@ALL (R 30)))
        ((@ALL (R 28)))
        ((*BETWEEN (S 37)) (*COMP (S 39)) (*NUM (S 40)))
        ((@ALL (R 23)))
        ((@ALL (R 22)))
        ((*NUM (S 41)))
        ((@ALL (R 40)) (*AND (S 43)) (*OR (S 44)))
        ((*THAN (S 45)))
        ((*OR (S 46)))
        ((*AND (S 47)))
        ((*COMP (S 39)) (*NUM (S 40)))
        ((@ALL (R 24)))
        ((@ALL (R 25)))
        ((*NUM (S 49)))
        ((*COMP (S 50)))
        ((*NUM (S 51)))
        ((@ALL (R 41)))
        ((@ALL (R 37)))
        ((@ALL (R 38)))
        ((@ALL (R 39)))
))
```

LR PARSING GOTO TABLE

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;     LR parsing goto table used by Tomita's parsing
;     procedures.  In the file "tbl2.lsp"
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(setf table2 '#(
        ((S 1) (ARITH_IMP 2) (IMP 3) (WHQ 4))
        NIL
        NIL
        NIL
        ((WHQNP 8) (NPS 9))
        ((NPS 11))
        ((WHQNP 13) (NPS 9))
        NIL
        NIL
        NIL
        ((NPS1 16) (ADJP 17))
        NIL
        ((NPS1 16) (ADJP 17))
        NIL
        NIL
        NIL
        ((CPP 22))
        ((CSE 24))
        NIL
        NIL
        NIL
        ((NPS 27))
        NIL
        ((CNPS 28) (CN_SEQ 29) (CSE 31))
        ((CPP 32))
        NIL
        NIL
        NIL
        NIL
        NIL
        NIL
        NIL
        NIL
        NIL
        ((SEQUENCE 36) (SEQ 38))
        NIL
        NIL
        NIL
        ((CONJUNC 42))
        NIL
        NIL
        NIL
        ((SEQ 48))
        NIL
        NIL
        NIL
        NIL
        NIL
        NIL
        NIL
        NIL
        NIL
))
```

LIST OF PRETERMINALS

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;     List of the possible preterminals according to the
;     context-free grammar rules.  In the file "preterm.lsp"
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(setq preterms
       '(*BE_VERB *WHAT *BETWEEN *THAN *COMP *NS *HRS *CRED
       *DET *COMMAND *NO *CS *PREP *OR *AND *NUM *VNO *CSP
         *ARITH_COMMAND *ADJ $)
)
```

TERMINAL LOOKUP TABLE

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;     Lookup table used in conjunction with the context-free
;     grammar rules.  In the file "lookup.lsp"
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(setf lookup '#(
      (a                      *det)
      (an                     *det)
      (and            *and)
      (are            *be_verb)
      (average        *arith_command)
      (between        *between)
      (course         *cs)
      (courses        *csp)
      (credit         *cred)
      (description    *ns)
      (descriptions   *ns)
      (for            *prep)
      (give           *command)
      (greater        *comp)
      (higher         *comp)
      (hours          *hrs)
      (is                     *be_verb)
      (less           *comp)
      (list           *command)
      (lower          *comp)
      (name           *ns)
      (names          *ns)
      (new            *adj)
      (number         *no)
      (numbered       *vno)
      (numbers        *no)
      (of                     *prep)
      (old            *adj)
      (or                     *or)
      (prerequisite           *ns)
      (prerequisites  *ns)
      (produce        *command)
      (sum            *arith_command)
      (than           *than)
      (the            *det)
      (what           *what)
      (1200           *num)
))
```

CONVERTED ATTRIBUTE GRAMMAR RULES

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;     Attribute grammar rules used by Petrick's procedures,
;     after the conversion from the original attribute
;     grammar rules.   This list bound to the variable
;     translationrules.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

((WHQNP *DET *ARITH_COMMAND *PREP NPS) (((TR 0)
                                (FIND_FUNC (TR 2) (TR 4))))
 (WHQNP NPS) (((TR 0) (TR 1))) (WHQ *WHAT *BE_VERB)
                                (((TR 0) (TR 1)))
 (START S) (((TR 0) (TR 1)))
 (SEQUENCE SEQ CONJUNC SEQ) (((TRA 0) (TR 1))
                                ((COA 0) (CO 1))
                                ((TRB 0) (TR 3))
                                ((COB 0) (CO 3))
                                ((CONJ 0) (TR 2)))
 (SEQUENCE SEQ) (((TRA 0) (TR 1))
                          ((COA 0) (CO 1))
                          ((TRB 0) 'EMPTY)
                          ((COB 0) 'EMPTY)
                          ((CONJ 0) 'EMPTY))
 (SEQUENCE *BETWEEN *NUM *AND *NUM)
                ((TRA 0) (LESSERNUM (TR 2)
                                                  (TR 4)))
                 ((COA 0) '>=)
                         ((TRB 0) (GREATERNUM (TR 2)
                                                       (TR 4)))
                 ((COB 0) '<=)
                 ((CONJ 0) (TR 3)))
 (SEQ *NUM *OR *COMP) (((TR 0) (TR 1))
                                    ((CO 0) (COMPARE (TR 3) "=")))
 (SEQ *COMP *THAN *NUM) (((TR 0) (TR 3))
                                      ((CO 0) (COMPARE (TR 1) NIL)))
 (S WHQ WHQNP) (((TR 0) (TR 2)))
 (S IMP) (((TR 0) (TR 1)))
 (S ARITH_IMP) (((TR 0) (TR 1)))
 (NPS1 *NS) (((TR 0) (TR 1)))
 (NPS1 *CRED *HRS) (((TR 0) CRED_HRS))
 (NPS *DET NPS1 CPP) (((VAR 0) (GENSYM))
                       ((TR 0) (NPSINTERP (VAR 0)
                                          (TR 1)
                                             NIL
                                             (TR 2)
                                             (COA 3)
                                             (TRA 3)
                                             (CONJ 3)
                                             (COB 3)
                                             (TRB 3))))
 (NPS *DET ADJP CSE CPP) (((VAR 0) (GENSYM))
                           ((TR 0) (NPSINTERP (VAR 0)
                                              (TR 1)
                                              (TR 2)
                                              (TR 3)
                                              (COA 4)
                                              (TRA 4)
                                              (CONJ 4)
                                              (COB 4)
                                              (TRB 4))))
 (IMP *COMMAND WHQNP) (((TR 0) (TR 2)))
 (CSE *CS *NO) (((TR 0) CNO))
 (CPP *PREP CNPS) (((TRA 0) (TR 2))
                           ((COA 0) (CO 2))
                           ((CONJ 0) 'EMPTY)
                           ((TRB 0) 'EMPTY)
                           ((COB 0) 'EMPTY))
 (CPP *PREP CN_SEQ) (((TRA 0) (TRA 2))
                                ((COA 0) (COA 2))
                                ((TRB 0) (TRB 2))
```

```
                                ((COB 0) (COB 2))
                                ((CONJ 0) (CONJ 2)))
 (CONJUNC *OR) (((TR 0) (TR 1)))
 (CONJUNC *AND) (((TR 0) (TR 1)))
 (CNPS CSE *NUM) (((TR 0) (TR 2)) ((CO 0) '=))
 (CN_SEQ *CSP *VNO SEQUENCE) (((TRA 0) (TRA 3))
                                              ((COA 0) (COA 3))
                                              ((TRB 0) (TRB 3))
                                              ((COB 0) (COB 3))
                                              ((CONJ 0) (CONJ 3)))
 (ARITH_IMP *ARITH_COMMAND NPS)
                         (((TR 0) (FIND_FUNC (TR 1) (TR 2))))
 (ADJP *ADJ) (((TR 0) (TR 1)))
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;; VX rules used to show that the different grammatical
;;;;; categories can include several grammatical terminals.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

 (*WHAT VX)                              (((TR 0) 1))
 (*VNO VX)                               (((TR 0) 1))
 (*THAN VX)                              (((TR 0) 1))
 (*PREP VX)                              (((TR 0) 1))
 (*OR VX)                                (((TR 0) 1))
 (*NUM VX)                               (((TR 0) 1))
 (*NS VX)                                (((TR 0) 1))
 (*NO VX)                                (((TR 0) 1))
 (*HRS VX)                               (((TR 0) 1))
 (*DET VX)                               (((TR 0) 1))
 (*CSP -> (VX))                  (((TR 0) 1))
 (*CS -> (VX))                   (((TR 0) 1))
 (*CRED -> (VX))                 (((TR 0) 1))
 (*COMP -> (VX))                 (((TR 0) 1))
 (*COMMAND -> (VX))              (((TR 0) 1))
 (*BETWEEN -> (VX))              (((TR 0) 1))
 (*BE_VERB -> (VX))              (((TR 0) 1))
 (*ARITH_COMMAND -> (VX))        (((TR 0) 1))
 (*AND -> (VX))                  (((TR 0) 1))
 (*ADJ -> (VX))                  (((TR 0) 1))
```

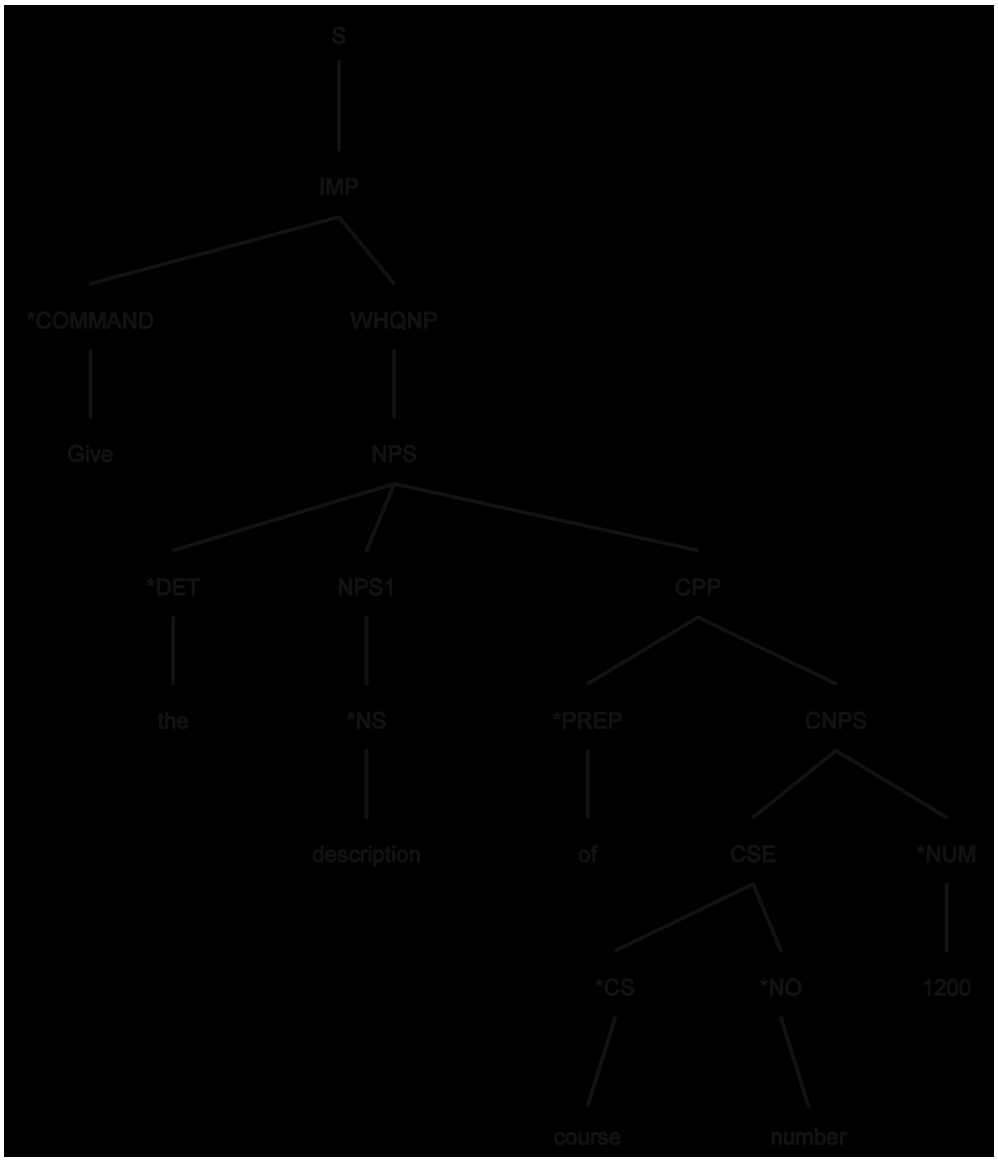
APPENDIX B

TRACE OF SYSTEM OPERATION

EXAMPLE QUERY 1

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
       Do you wish to use speech recognition input?
              Yes or No : no

       Speech input will not be recognized.

       --------------------------------------------------
       Do you wish the computer output to be spoken?
            Yes or No : no

       Output will not be spoken.

       --------------------------------------------------
       Do you wish the parsing to be timed?
            Yes or No : yes

       Parsing will be timed.

       --------------------------------------------------
       The sentence so far is empty.

       Input rest of sentence, one word at a time:
              Give
              the
              back

       --------------------------------------------------
       The sentence so far is: Give

       Input rest of sentence, one word at a time:
              clear

       --------------------------------------------------
       The sentence so far is empty.

       Input rest of sentence, one word at a time:
              Give
              a
              o

       --------------------------------------------------
       ERROR!!! Your last word was unrecognized.

       Possible choices are of the type:

         *adj (such as new), and *ns (such as description).
```

```
--------------------------------------------------
The sentence so far is: give a

Input rest of sentence, one word at a time:

       description
       of
       course
       number
       1200
       $

The sentence is accepted.

Time used for parsing: 0 minutes, 0 seconds.

--------------------------------------------------
This is the parse tree produced by Tomita's procedure and bound to fa:
       1 (*COMMAND give)
       2 (*DET a)
       3 (*NS description)
       4 (NPS1 (3))
       5 (*PREP of)
       6 (*CS course)
       7 (*NO number)
       8 (CSE (6 7))
       9 (*NUM 1200)
       10 (CNPS (8 9))
       11 (CPP (5 10))
       12 (NPS (2 4 11))
       13 (WHQNP (12))
       14 (IMP (1 13))
       15 (S (14))

--------------------------------------------------
This is the parse tree converted to Petrick's form and bound to parses:

       (((S)
          ((IMP)
           ((*COMMAND ((give TR))) ((T)))
                   ((WHQNP)
                    ((NPS)
                  ((*DET ((a TR))) ((T)))
                  ((NPS1) ((*NS ((description TR)))
                                                      ((T))))
                  ((CPP)
                     ((*PREP ((of TR))) ((T)))
                     ((CNPS)
                        ((CSE)
                           ((*CS ((course TR))) ((T)))
                           ((*NO ((number TR))) ((T))))
                        ((*NUM ((1200 TR))) ((T)))))))))

--------------------------------------------------

This is the parse tree in graphical form:
```



This is the parse tree converted to SETX notation and bound to **translation1:**

```
(SETX '#:G-173
   '(RELATION 'COURSES
                   '(DESCRIP NEWNO)
                   '(#:G-173 1200)
                   '(= =)))
```

This is the SETX notation converted to intermediate SQL notation and bound to **translation2:**

```
(WHQ SELECT\ DISTINCT\  ((DESCRIP A) (NEWNO A))
       (((COURSES A))
       ((#:G-173 (DESCRIP A)))
       (((NEWNO A) = 1200)) NIL NIL))
```

```
-------------------------------------------------
This is the SQL query produced from translation2 by the procedure printsql1:

        SELECT DISTINCT A.DESCRIP, A.NEWNO
        FROM COURSES A
        WHERE A.NEWNO = 1200

Time used for translation: 0 minutes, 1 seconds.

-------------------------------------------------
CHOOSE: continue or quit.
        continue

-------------------------------------------------
The sentence so far is empty.

Input rest of sentence, one word at a time:
        quit

Done.
```

EXAMPLE QUERY 2

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        Do you wish to use speech recognition input?
                Yes or No : no

        Speech input will not be recognized.

        --------------------------------------------------
        Do you wish the computer output to be spoken?
        Yes or No : no

        Output will not be spoken.

        --------------------------------------------------
        Do you wish the parsing to be timed?
        Yes or No : no

        Parsing will not be timed.

        --------------------------------------------------
        The sentence so far is empty.

        Input rest of sentence, one word at a time:
                Give
                the
                old
                course
                number
                of
                course
                number
                3100
                $

        The sentence is accepted.

        --------------------------------------------------
        SELECT DISTINCT A.OLDNO, A.NEWNO
        FROM COURSES A
        WHERE A.NEWNO = 3100

        --------------------------------------------------
        CHOOSE: continue or quit.
                quit

        Done.
```

EXAMPLE QUERY 3

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

       Do you wish to use speech recognition input?
              Yes or No : no

       Speech input will not be recognized.

       --------------------------------------------------
       Do you wish the computer output to be spoken?
       Yes or No : no

       Output will not be spoken.

       --------------------------------------------------
       Do you wish the parsing to be timed?
       Yes or No : no

       Parsing will not be timed.

       --------------------------------------------------
       The sentence so far is empty.

       Input rest of sentence, one word at a time:
              Give
              the
              new
              course
              number
              for
              course
              number
              301
              $

       The sentence is accepted.

       --------------------------------------------------
       SELECT DISTINCT A.OLDNO, A.NEWNO
       FROM COURSES A
       WHERE A.NEWNO = 301

       --------------------------------------------------
       CHOOSE: continue or quit.
              continue
```

EXAMPLE QUERY 4

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        The sentence so far is empty.

        Input rest of sentence, one word at a time:
                Give
                the
                name
                of
                course
                number
                3200
                $

        The sentence is accepted.

        --------------------------------------------------
        SELECT DISTINCT A.NAME, A.NEWNO
        FROM COURSES A
        WHERE A.NEWNO = 3200

        --------------------------------------------------
        CHOOSE: continue or quit.
                continue
```

EXAMPLE QUERY 5

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        The sentence so far is empty.

        Input rest of sentence, one word at a time:
                Give
                the
                prerequisites
                for
                course
                number
                5270
                $

        The sentence is accepted.

        --------------------------------------------------
        SELECT DISTINCT A.PREREQ, A.NEWNO
        FROM COURSES A
        WHERE A.NEWNO = 5270

        --------------------------------------------------
        CHOOSE: continue or quit.
                continue
```

EXAMPLE QUERY 6

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

       The sentence so far is empty.

       Input rest of sentence, one word at a time:
              What
              is
              the
              name
              of
              course
              number
              300
              $

       The sentence is accepted.

       --------------------------------------------------
       SELECT DISTINCT A.NAME, A.NEWNO
       FROM COURSES A
       WHERE A.NEWNO = 300

       --------------------------------------------------
       CHOOSE: continue or quit.
              continue
```

EXAMPLE QUERY 7

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

       The sentence so far is empty.

       Input rest of sentence, one word at a time:
              What
              is
              the
              sum
              of
              the
              credit
              hours
              for
              courses
              numbered
              higher
              than
              400
              $

       The sentence is accepted.

       --------------------------------------------------
       SELECT SUM(A.CRED_HRS)
       FROM COURSES A
       WHERE A.NEWNO > 400

       --------------------------------------------------
       CHOOSE: continue or quit.
              continue
```

EXAMPLE QUERY 8

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        The sentence so far is empty.

        Input rest of sentence, one word at a time:
                List
                the
                name
                of
                courses
                numbered
                between
                300
                and
                400
                $

        The sentence is accepted.

        --------------------------------------------------
        SELECT DISTINCT B.NAME, B.NEWNO, A.NEWNO
        FROM COURSES A, COURSES B
        WHERE A.NAME = B.NAME
        AND A.NEWNO >= 300
        AND B.NEWNO <= 400

        --------------------------------------------------
        CHOOSE: continue or quit.
                continue
```

EXAMPLE QUERY 9

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

       The sentence so far is empty.

       Input rest of sentence, one word at a time:
               List
               the
               old
               course
               numbers
               for
               courses
               numbered
               between
               1000
               and
               500
               $

       The sentence is accepted.

       --------------------------------------------------
       SELECT DISTINCT B.OLDNO, B.NEWNO, A.NEWNO
       FROM COURSES A, COURSES B
       WHERE A.OLDNO = B.OLDNO
       AND A.NEWNO >= 500
       AND B.NEWNO <= 1000

       --------------------------------------------------
       CHOOSE: continue or quit.
               continue
```

EXAMPLE QUERY 10

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

       The sentence so far is empty.

       Input rest of sentence, one word at a time:
              What
              are
              the
              new
              course
              numbers
              for
              courses
              numbered
              greater
              than
              400
              and
              lower
              than
              600
              $

       The sentence is accepted.

       --------------------------------------------------
       SELECT DISTINCT B.OLDNO, B.NEWNO, A.NEWNO
       FROM COURSES A, COURSES B
       WHERE A.OLDNO = B.OLDNO
       AND A.NEWNO > 400
       AND B.NEWNO < 600

       --------------------------------------------------
       CHOOSE: continue or quit.
              continue
```

EXAMPLE QUERY 11

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

       The sentence so far is empty.

       Input rest of sentence, one word at a time:
              Average
              the
              credit
              hours
              for
              courses
              numbered
              1500
              or
              less
              $

       The sentence is accepted.

       --------------------------------------------------
       SELECT AVG(A.CRED_HRS)
       FROM COURSES A
       WHERE A.NEWNO <= 1500

       --------------------------------------------------
       CHOOSE: continue or quit.
              quit

       Done.
```

APPENDIX C

PROGRAM LISTINGS

PARSING PROCEDURES

```
;;;;;;;;;;;;;Tomita's Parsing Algorithm;;;;;;;;;;;;;;;;;;;
;  John D. Hastings, 4/8/91
;  - The files term.lsp, tbl1.lsp, and tbl2.lsp are created
;    by calling functions in lr0.lsp
;  - setrules.lsp contains the definition of the attribute
;    grammar rules ATT_RULES which are sorted
;    alphabetically, and which must be loaded.
;  - preterm.lsp contains the list of the preterminals
;    PRETERMS, which must be loaded.
;  - tbl1.lsp contains the action table TABLE1, which must
;    be loaded.
;  - tbl2.lsp contains the goto table TABLE2, which must be
;    loaded.
;  - utility functions in PUTILS.LSP and RESPLIT.LSP must be
;    loaded.
;  - Call (PARSESENTENCE) to interactively parse a sentence
;    of words according to the grammar rules RULES, the
;    action table TABLE1, and the goto table TABLE2.  The
;    parsing is timed and computer feedback is spoken, if
;    the user requests it.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(LOAD "setrules.lsp")
(LOAD "preterm.lsp")
(LOAD "tbl1.lsp")
(LOAD "tbl2.lsp")
(LOAD "putils.lsp")
(LOAD "resplit.lsp")
(LOAD "contree.lsp")
(LOAD "congram.lsp")
(LOAD "lookup.lsp")
(LOAD "npsinter.lsp")
(LOAD "knuth.lsp")
(LOAD "lcp.lsp")
(LOAD "newtos.lsp")
(LOAD "printsql.lsp")
(setq end_symbol '$)
(setq num_type '*num)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;; convert translation rules from Tomita's to Petrick's
;;;;; form
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(setq translationrules (convert_trans att_rules))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;; convert Tomita translation rules to the expected
;;;;; Tomita grammar
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(setq rules (convert_gram att_rules))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;; initialize the system for Knuth translations
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(setlcp (getcfrules translationrules))

(DEFUN FINDARC (XLIST WSET)
   (COND ((NULL XLIST) NIL)
         ((EQUAL WSET (CDR (AREF GA (CAR XLIST)))) (CAR XLIST))
         (T (FINDARC (CDR XLIST) WSET))))

(DEFUN RESPLIT- (V)
   (declare (special n L))
   (RESPLIT (1- N) V L))

(DEFUN ADDSUBNODE (N L) (SETF (AREF FA N) (APPEND
                       (AREF FA N) (LIST L))))

(DEFUN SYMBOL (X) (CAR (AREF GA X)))

(DEFUN STATE (V) (CAR (AREF GA V)))

(DEFUN CREATEVERTEX (IN LABEL)
   (COND ((EQUAL IN 'G)
          (SETQ GP (1+ GP))
          (SETF (AREF GA GP) (LIST LABEL))
          GP)
         ((EQUAL IN 'F)
          (SETQ FP (1+ FP))
          (SETF (AREF FA FP) (LIST LABEL))
          FP)))

(DEFUN CREATEEDGE (V X) (SETF (AREF GA V)
               (APPEND (AREF GA V) (LIST X))))

(DEFUN REDUCE2 (SW)
   (DECLARE (SPECIAL L))
   (PROG (S WSET U Z M Q CAT ACT)
      (SETQ S (CAR SW))
      (SETQ WSET (CDR SW))
      (SETQ U (FIND-U-IN-USET S))
      (COND (U (SETQ Z (FINDARC (CDR (AREF GA U)) WSET))
               (COND (Z (ADDSUBNODE (SYMBOL Z) L))
                     (T (SETQ M (CREATEVERTEX 'F LEFTP))
                        (ADDSUBNODE M L)
                        (SETQ Z (CREATEVERTEX 'G M))
                        (CREATEEDGE U Z)
                        (SETF (AREF GA Z) (APPEND (AREF GA Z) WSET))
                        (COND ((NOT (MEMBER U ASET))
                               (DO ((C CATLIST (CDR C)))
                                   ((NULL C) NIL)
                                   (SETQ CAT (CAR C))
                                   (DO ((AC (ACTION (STATE U) CAT)
                                            (CDR AC)))
                                       ((NULL AC) NIL)
                                       (SETQ ACT (CAR AC))
                                       (COND ((EQUAL 'R (CAR ACT))
                                              (SETQ Q (CADR ACT))
                                              (COND ((AND (< 0. (LENG Q))
                                                          (NOT (MEMBER (LIST U
                                                                       Z
                                                                       Q :test
                                                                         'equal)
                                                               RSET)))
                                                    (push (list u z q)
                                                          RSET))
                                              ))))))))))
            (T (SETQ M (CREATEVERTEX 'F LEFTP))
               (ADDSUBNODE M L)
               (SETQ U (CREATEVERTEX 'G S))
```

```
              (SETQ Z (CREATEVERTEX 'G M))
              (CREATEEDGE U Z)
              (SETF (AREF GA Z) (APPEND (AREF GA Z) WSET))
              (push U ASET)
              (push U USET)))))

(DEFUN REDUCER NIL
   (DECLARE (SPECIAL LEFTP))
   (PROG (X P)
      (SETQ X (CADAR RSET))
      (SETQ P (CADDAR RSET))
      (pop RSET)
      (SETQ LEFTP (LEFT P))
      (MAPCAR 'REDUCE1 (RESPLIT (- (* 2.
                                      (LENG P)) 2.) X NIL))))

(DEFUN REDUCE1 (WSET-L)
   (DECLARE (SPECIAL L LEFTP))
   (PROG (WSET SW-TABLE W)
      (SETQ WSET (CAR WSET-L))
      (SETQ L (CADR WSET-L))
      (DO ((W- WSET (CDR W-)))
         ((NULL W-) NIL)
         (SETQ W (CAR W-))
         (SETQ SW-TABLE
            (PUTASSOC (LIST (GOTO (STATE W) LEFTP) W)
                      SW-TABLE)))
      (MAPCAR 'REDUCE2 SW-TABLE)))

(DEFUN LENG (P) (LENGTH (CADDR (AREF RULES P))))

(DEFUN GOTO (S A) (CADR (ASSOC A (AREF TABLE2 S))))

(DEFUN LEFT (P) (CAR (AREF RULES P)))

(DEFUN ACTION (S C)
   (OR (APPEND (CDR (ASSOC '@ALL (AREF TABLE1 S)))
               (CDR (ASSOC C (AREF TABLE1 S))))
       (CDR (ASSOC '*ELSE (AREF TABLE1 S)))))

(DEFUN FIND-U-IN-USET (S)
   (DO ((USET- USET (CDR USET-)))
      ((NULL USET-) NIL)
      (COND ((EQUAL S (CAR (AREF GA (CAR USET-))))
                     (RETURN (CAR USET-))))))

(DEFUN PARSEWORD (CATLIST)
   (DECLARE (SPECIAL CATLIST))
   (PROG NIL
         (SETQ ASET USET)
         (SETQ QSET NIL)
         (SETQ RSET NIL)
         (SETQ RESET NIL)
    LOOP (COND (ASET (ACTOR))
               (RSET (REDUCER))
               (RESET (E-REDUCER)))
         (COND ((OR ASET RSET RESET) (GO LOOP)))
         (SHIFTER)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;; The variables fpstack, gpstack, and usetstack each
;;;;; contain a stack of pointers used for backing up in the
;;;;; parse tree.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(DEFUN INITPARSE NIL
   (SETQ GP 0.)
   (SETQ FP 0.)
   (SETQ RESULT NIL)
   (SETQ GA (MAKE-ARRAY 6000))
   (SETQ FA (MAKE-ARRAY 4000))
   (SETF (AREF GA 0) '(0.))
   (SETQ USET '(0.))
```

```
   (setq fpstack '(0))
   (setq gpstack '(0))
   (setq usetstack '((0.))))

(DEFUN ACTOR NIL
   (declare (special catlist))
   (PROG (V C X P ACT)
      (SETQ V (pop aset))
      (DO ((C- CATLIST (CDR C-)))
          ((NULL C-) NIL)
          (SETQ C (CAR C-))
          (DO ((ACT- (ACTION (STATE V) C) (CDR ACT-)))
              ((NULL ACT-) NIL)
              (SETQ ACT (CAR ACT-))
              (COND ((EQ (CAR ACT) 'S)
                     (push (LIST V C (CADR ACT)) QSET))
                    ((EQ (CAR ACT) 'R)
                     (SETQ P (CADR ACT))
                     (COND ((ZEROP (LENG P))
                            (COND ((NOT (MEMBER (LIST V P) RESET
                                                :test 'equal))
                                   (push (LIST V P) RESET)) ))
                           (T (DO ((X- (CDR (AREF GA V)) (CDR X-)))
                                  ((NULL X-) NIL)
                                  (SETQ X (CAR X-))
                                  (COND ((NOT (MEMBER (LIST V X P) RSET
                                                      :test 'equal))
                                         (push (LIST V X P) RSET)))))))
                    ((EQ (CAR ACT) 'A)
                     (SETQ RESULT (SYMBOL (CADR
                                          (AREF GA V)))))))))))

(DEFUN SHIFTER NIL
   (PROG (SWSET CWZSET CMSET V C S W M Z)
      (SETQ USET NIL)
      (DO ((QSET- QSET (CDR QSET-)))
          ((NULL QSET-) NIL)
          (SETQ V (CAAR QSET-))
          (SETQ C (CADAR QSET-))
          (SETQ S (CADDAR QSET-))
          (SETQ W (CADR (ASSOC S SWSET)))
          (COND ((NULL W)
                 (SETQ W (CREATEVERTEX 'G S))
                 (push (LIST S W) SWSET)
                 (push W USET)))
          (SETQ Z (CADR (ASSOC (LIST C W) CWZSET)))
          (COND ((NULL Z)
                 (SETQ M (CADR (ASSOC C CMSET)))
                 (COND ((NULL M)
                        (SETQ M (CREATEVERTEX 'F C))
                        (ADDSUBNODE M term_symbol)
                        (push (LIST C M) CMSET)))
                 (SETQ Z (CREATEVERTEX 'G M))
                 (push (LIST (LIST C W) Z) CWZSET)
                 (CREATEEDGE W Z)))
          (CREATEEDGE Z V))))

(DEFUN E-REDUCER NIL
   (PROG (SWSET CWZSET CMSET V C S W M Z)
      (SETQ USET NIL)
      (DO ((RESET- RESET (CDR RESET-)))
          ((NULL RESET-) NIL)
          (SETQ V (CAAR RESET-))
          (SETQ C (LEFT (CADAR RESET-)))
          (SETQ S (GOTO (STATE V) C))
          (SETQ W (CADR (ASSOC S SWSET)))
          (COND ((NULL W)
                 (SETQ W (CREATEVERTEX 'G S))
                 (push (LIST S W) SWSET)
                 (push W USET)
                 (push W ASET)))
          (SETQ Z (CADR (ASSOC (LIST C W) CWZSET)))
```

```
          (COND ((NULL Z)
                 (SETQ M (CADR (ASSOC C CMSET)))
                 (COND ((NULL M)
                        (SETQ M (CREATEVERTEX 'F C))
                        (ADDSUBNODE M NIL)
                        (push (LIST C M) CMSET)))
                 (SETQ Z (CREATEVERTEX 'G M))
                 (push (LIST (LIST C W) Z) CWZSET)
                 (CREATEEDGE W Z)))
          (CREATEEDGE Z V))
       (SETQ RESET NIL)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;; Take a terminal or preterminal and convert it to a
;;;;; lower case string.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defun term_to_string (term)
   (cond ((numberp term) (num_to_string term))
         (t (string-downcase (string term)))
   ))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;; Print a list of the category preterminals along with
;;;;; example category terminals, which can be legally
;;;;; entered as the next word in the current sentence.  The
;;;;; legal words will be chosen from the set of
;;;;; preterminals minus the invalid preterminal.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defun find_next_possible ()
   (prog (possible)
      (setq possible "")
      (do ((termin (remove (car word) preterms)
                                   (cdr termin)))
          ((null termin) nil)
          (setq current_term (car termin))
          (setq fp (car fpstack))
          (setq gp (car gpstack))
          (setq uset (car usetstack))
          (setq result nil)
          (parseword (list current_term))
          (cond ( (or (> (length uset) '0)
                      (not (eq result nil)))
                  (setq possible (string-append
                                   (cond ((equal possible "") "and ")
                                         (t ""))
                                   (term_to_string current_term)
                                   (cond ((eq current_term end_symbol)
                                          " (end of sentence marker"
                                         )
                                         (t (string-append
                                              " (such as "
                                              (term_to_string
                                                 (find_example current_term)
                                              )
                                            )
                                         )
                                   )
                                   (cond (possible (string-append
                                                     "), " possible))
                                         (t ")")
                                   )
                                 )
                  )
                )
          )
      )
      (setq possible (string-left-trim "and " possible))
      (setq possible (string-right-trim ", " possible))
      (setq fp (car fpstack))
      (setq gp (car gpstack))
      (setq uset (car usetstack))
      (setq result nil)
```

```
      (outit "Possible choices are of the type: ")
      (princ (string-append "         " possible "."))
      (terpri)
      (princ
      "--------------------------------------------------")
      (terpri)
   ))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;; Start the stopwatch.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defun init_time ()
   (multiple-value-bind (s m) (get-decoded-time)
      (setq sec1 s) (setq min1 m)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;; Stop the stopwatch.  Add the result to the total time.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defun add_to_total_time ()
   (prog (sec2 min2)
   (multiple-value-bind (s m) (get-decoded-time)
      (setq sec2 s) (setq min2 m))
   (cond ((< sec2 sec1)(setq sec2 (+ sec2 '60))
         (setq min2 (1- min2)) (setq sec2 (- sec2 sec1)))
        (t (setq sec2 (- sec2 sec1))))
   (setq min2 (- min2 min1))
   (setq totmin (+ totmin min2))
   (setq totsec (+ totsec sec2))
   (cond ((> totsec '59)(setq totmin (1+ totmin))
                        (setq totsec (- totsec '60))))
))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;; Lookup SYMBOL (category terminal) in the lookup table
;;;;; and determine it's category preterminal.
;;;;; Example: the lookup of 'the is '*det.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defun look (symbol)
   (prog (index)
      (setq index (bisearcharray lookup symbol))
      (cond (index (return (cadr (aref lookup
                                 (car index)))))
            (t (return nil)))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;; Find any preterminal which is an example of the
;;;;; CATEGORY preterminal in the grammar.
;;;;; Example: an example of '*det is 'the
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defun find_example (category)
   (prog (example)
      (do ((n 0 (1+ n)))
        ((or example (= n (length lookup))) )
        (cond ((equal category (cadr (aref lookup n)))
               (setq example (car (aref lookup n))) ) ) )
      (return example)
   ))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;; Read in the next terminal.  Convert it to a
;;;;; preterminal (bound to WORD), and parse it to see if
;;;;; it's valid.  If the terminal is not valid prompt the
;;;;; user, and give the list of valid categories.
;;;;; Otherwise, if the word is valid, add the word terminal
;;;;; and preterminal to the end of their respective
;;;;; sentences, and update the valid parse variables.
;;;;; Time the procedure, if requested.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defun next_symbol_in ()
   (prog ()
      (setq term_symbol (read))
      (cond (timed (init_time)))
```

```
      (cond ((equal term_symbol end_symbol)
             (setq word term_symbol))
            ((numberp term_symbol) (setq word num_type))
            (t (setq word (look term_symbol))))
      (cond (word
              (setq word (list word))
              (PARSEWORD word)
              (cond ( (and (= (length uset) '0)
                           (eq result nil))
                 (princ
      "--------------------------------------------------")
                      (terpri)
                      (outit "ERROR!!!  Your last word was
                                          invalid.")
                      (setq errflag t)
                      (find_next_possible)
                      (setq word nil))
                    (t (setq sentence (append sentence
                                        (list word)))
                       (setq symbol_sentence
                               (append symbol_sentence
                                                (list term_symbol)))
                       (setq fpstack (cons fp fpstack))
                       (setq gpstack (cons gp gpstack))
                       (setq usetstack (cons uset usetstack)))))
            ( (equal term_symbol 'back)(setq errflag t)
                                (backup_one)
              (princ
      "--------------------------------------------------")
              (terpri))
            ( (equal term_symbol 'clear)(setq errflag t)
              (setq symbol_sentence nil)(setq sentence nil)
              (INITPARSE)(setq word nil)
              (princ
      "--------------------------------------------------")
              (terpri))
            ( (equal term_symbol 'quit) (return nil))
            (t (princ
        "--------------------------------------------------")
               (terpri)
               (outit "ERROR!!! Your last word was
                                     unrecognized.")
               (find_next_possible)(setq errflag t)))
      (cond (timed (add_to_total_time)))
   ))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;; Remove the most recent terminal from the terminal
;;;;; sentence (SYMBOL_SENTENCE) and the most recent
;;;;; preterminal from the preterminal sentence (SENTENCE),
;;;;; and reset the parse variables to the last proper
;;;;; values.  Time the procedure, if requested.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defun backup_one ()
   (cond ((null sentence) (terpri)
          (outit "ERROR.  Can't backup in an empty sentence.")
          (terpri))
         (t
            (setq sentence (remlast sentence))
            (setq symbol_sentence (remlast symbol_sentence))
            (setq fpstack (cdr fpstack))
            (setq fp (car fpstack))
            (setq gpstack (cdr gpstack))
            (setq gp (car gpstack))
            (setq usetstack (cdr usetstack))
            (setq uset (car usetstack))
         )))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;; Ask the user if he wants to continue or quit, and set
;;;;; the variable CONTINUE as the result.  If CONTINUE is
;;;;; invalid, make the user reenter the information.
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defun get_cont ()
   (prog ()
      (terpri) (terpri)
again (outit "CHOOSE: continue or quit.")
      (setq continue (read)) (terpri)
      (cond ((not (or (eq continue 'continue)
                              (eq continue 'quit)))
             (outit "Not a valid choice, try again.")
             (go again)))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;; If spoken output is requested, speak it.      Otherwise,
;;;;; print it.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defun outit (str)
   (cond (speakit (speak str))
         (t (princ str)(terpri)(terpri))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;; Convert a number to a string for speech purposes.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defun num_to_string (num)
   (format nil "~S" num))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;; Determine if the user wishes computer output to be
;;;;; spoken, and set the flag SPEAKIT accordingly.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defun get_speech_out ()
   (prog ()
      (terpri)
loop1 (princ "Do you wish the computer output to be
                              spoken? ")
        (terpri)
      (princ "      Yes or No : ")
      (setq speakit (read))
      (cond ((eq speakit 'Yes) (setq speakit 'T)
                    (load "speech")
            (speak " Output will be spoken.")
            (princ
        "--------------------------------------------------")
            (terpri)(terpri))
           ((eq speakit 'No) (setq speakit nil)
            (princ " Output will not be spoken.")(terpri)
            (princ
        "--------------------------------------------------")
            (terpri)(terpri))
           (t (princ "Not a valid choice, TRY AGAIN.")
              (go loop1)))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;; Determine if the user wants the speech recognition
;;;;; input system to be installed.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defun get_speech_in ()
   (prog ()
      (sys:dos "cls")
      (terpri)
loop1 (princ "Do you wish to use speech recognition
                     input? ")
        (terpri)
      (princ "      Yes or No : ")
      (setq speechin (read))
      (cond ((eq speechin 'Yes)
            (sys:dos "cd \\vmkey")
            (sys:dos "vmkey")
            (sys:dos "cd \\gw\\gwd")
            (terpri)
            (princ
        "--------------------------------------------------")
            (terpri))
```

```
            ((eq speechin 'No)
             (princ "Speech input will not be
                              recognized.")(terpri)
             (princ
          "--------------------------------------------------")
             (terpri))
            (t (princ "Not a valid choice, TRY AGAIN.")
               (go loop1)))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;; Determine if the user wishes the parsing to be timed,
;;;;; and set the flag TIMED accordingly.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defun get_timed ()
   (prog ()
loop2 (outit "Do you wish the parsing to be timed?")
      (princ "      Yes or No : ")
      (setq timed (read))
      (cond ((eq timed 'Yes) (setq timed 'T)
             (outit " Parsing will be timed.")
             (princ
          "--------------------------------------------------")
             (terpri)(terpri))
            ((eq timed 'No) (setq timed nil)
             (outit " Parsing will not be timed.")
             (princ
          "--------------------------------------------------")
             (terpri)(terpri))
            (t (outit "Not a valid choice.  TRY AGAIN.")
               (princ
          "--------------------------------------------------")
               (terpri)(terpri)
               (go loop2)))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;; Take a list of the form (N V N...) and convert it to a
;;;;; string of the form " n v n ...".
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defun sent_to_string (sent)
   (prog (tempsent printsent)
      (setq tempsent sent) (setq printsent "")
loop4 (cond ((not (null tempsent))
             (setq printsent (string-append printsent " "
                                    (term_to_string (car tempsent))))
             (setq tempsent (cdr tempsent))
             (go loop4)))
      (return printsent)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;; Convert a sentence of the form (N V N...) to a string
;;;;; using sent_to_string, and print it.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defun print_sent (sentence)
   (prog ()
      (terpri)
      (cond ((null sentence)(outit "The sentence so far is
                        empty."))
            (t (outit (string-append "The sentence so far is:"
                                     (sent_to_string sentence)
                                     "          "))))
      (outit "Input rest of sentence, one word at a
                                time:")))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;; Print an array (with the indexes to the left of each
;;;;; element) up to and including the array element indexed
;;;;; by maxindex.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defun print_array (array maxindex)
   (terpri)
   (do ((n 0 (1+ n)))
       ((or (equal n (length array))
```

```
                (equal n (1+ maxindex))) nil)
       (princ n)(princ " ")(princ (aref array n))(terpri)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;; 1. initialize the system
;;;;; 2. interactively parse the input sentence
;;;;;    - special inputs : clear (clear the entire
;;;;;                                          sentence)
;;;;;                       quit (quit the procedure)
;;;;;                       back (back up one symbol in the
;;;;;                                  sentence)
;;;;;    - SYMBOL_SENTENCE contains the terminal sentence
;;;;;    - SENTENCE contains the preterminal sentence
;;;;;    - FA contains the parse tree stack
;;;;;    - GA contains the state stack
;;;;; 3. convert the resulting Tomita parse tree to a parse
;;;;;    tree in Petrick's form
;;;;; 4. use KTRANSBT to find a preliminary setx translation
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(DEFUN PARSESENTENCE ()
   (prog ()
      (get_speech_in)
      (get_speech_out)
      (get_timed)
loop3 (SETQ PARSES NIL)
      (INITPARSE) (setq sentence nil) (setq word nil)
      (setq symbol_sentence nil)
      (setq totmin 0) (setq totsec 0) (setq errflag t)
      (do ((n 0 (1+ n)))
        ((equal word (list end_symbol)) (outit "The sentence
                       is accepted."))
        (cond (errflag
                 (print_sent symbol_sentence)
                 (setq errflag nil)))
        (next_symbol_in)
        (cond ((equal term_symbol 'quit) (return))) )
      (cond ((equal term_symbol 'quit) (return
                               (outit "Done."))))
      (cond (timed
               (outit (string-append "Time used for parsing: "
                                        (num_to_string totmin)
                                        " minutes, "
                                        (num_to_string totsec)
                                        " seconds."
                     ))))
      (COND (TIMED (SETQ TOTMIN 0)
                   (SETQ TOTSEC 0)
                   (INIT_TIME)
            )
      )
      (CONVERTTREE FA FP)
      (SETQ TRANSLATION1 (car (KTRANSBT NIL 'TR 'S 1)))
      (setq translation2 (lftosql1 translation1 nil))
      (printsql1 translation2 nil nil)
      (terpri)(terpri)
      (COND (TIMED (ADD_TO_TOTAL_TIME)
                   (OUTIT (STRING-APPEND "Time used for
                                         translation: "
                                         (num_to_string totmin)
                                         " minutes, "
                                         (num_to_string totsec)
                                         " seconds."
                          ))))
      (get_cont)
      (cond ((eq continue 'continue) (go loop3))
            ((eq continue 'quit) (outit "Done.") (return))
      )
   )
)
```

LR PARSING TABLE CONSTRUCTION PROCEDURES

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; LR(0) Parsing Table Constructor in the file "lr0.lsp"
;  - grammar rules in "setrules.lsp" and utility functions
;    in "putils.lsp" must be loaded
;  - call the function CONSTRUCT to put the action table
;    into file "tbl1.lsp", and the goto table into file
;    "tbl2.lsp"
;  - the function FIND_ALL_PRETERMS puts a list of all
;    preterminals (according to the grammar rules RULES)
;    into the file "preterm.lsp"
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(load "setrules")
(load "putils")
(LOAD "CONGRAM.LSP")

(SETQ RULES (CONVERT_GRAM ATT_RULES))

(DEFUN NEXTSYMBOL (ITEM)
   (COND ((= (LENGTH (CADDR (AREF RULES (CAR ITEM))))
                     (CADR ITEM)) NIL)
         (T (NTH (CADR ITEM) (CADDR (AREF RULES
                     (CAR ITEM)))))))

(DEFUN PUTITEMINORDER (X L)
   (COND ((NULL L) (LIST X))
         ((ITEM> X (CAR L))
          (CONS (CAR L) (PUTITEMINORDER X (CDR L))))
         (T (CONS X L))))

(DEFUN ITEM> (ITEM1 ITEM2)
   (COND ((= (CAR ITEM1) (CAR ITEM2))
          (> (CADR ITEM1) (CADR ITEM2)))
         (T (> (CAR ITEM1) (CAR ITEM2)))))

(DEFUN CLOSUREd (ITEMS)
   (SETQ DONE NIL)
   (DO ((N 0. (1+ N)))
      ((= N (LENGTH ITEMS)) ITEMS)
      (SETQ S (NEXTSYMBOL (NTH N ITEMS)))
      (COND ((AND S (NOT (TERM S)) (NOT (MEMBER S DONE)))
             (PUSH S DONE)
             (SETQ ITEMS
                   (APPEND ITEMS
                           (MAPCAR 'ATTACHO (BISEARCHARRAY
                                       RULES S))))
))))

(DEFUN GOTO- (ITEMS S)
   (COND ((NULL ITEMS) NIL)
         ((EQ (NEXTSYMBOL (CAR ITEMS)) S)
          (CONS (LIST (CAAR ITEMS) (1+ (CADAR ITEMS)))
                (GOTO- (CDR ITEMS) S)))
         (T (GOTO- (CDR ITEMS) S))))

(DEFUN CONSTRUCTSETSOFITEMS NIL
   (PROG NIL
      (TERPRI)
      (PRINTLINE '|Constructing sets of items...|)
      (SETQ SETSOFITEMS
            (LIST (LIST (LIST (CAR (BISEARCHARRAY
                                       RULES 'START))
                              0.))))
      (DO ((N 0. (1+ N)))
        ((= N (LENGTH SETSOFITEMS)) NIL)
        (PRINC (LIST N))
        (SETQ GOTOS NIL)
        (SETQ T1 NIL)
        (SETQ T2 NIL)
```

```
         (SETQ CLOSUREDITEMS (CLOSUREd (NTH N SETSOFITEMS)))
         (DO ((ITEMS- CLOSUREDITEMS (CDR ITEMS-)))
            ((NULL ITEMS-) NIL)
            (SETQ S (NEXTSYMBOL (CAR ITEMS-)))
            (COND ((NULL S)
                   (COND ((EQ 'start (CAR (AREF RULES
                                                (CAAR ITEMS-))))
                          (SETQ T1 (PUTASSOC '($ (A)) T1)))
                         (T (SETQ T1
                                  (PUTASSOC (LIST '@ALL
                                                  (LIST 'R
                                                        (CAAR ITEMS-)))
                                            T1)))))
                  (T (SETQ GOTOS
                           (PUTASSOC (LIST S
                                           (LIST (CAAR ITEMS-)
                                                 (1+ (CADAR ITEMS-))))
                                     GOTOS)))))
         (DO ((GOTOS- GOTOS (CDR GOTOS-)))
            ((NULL GOTOS-) NIL)
            (SETQ NEWITEMS (CDAR GOTOS-))
            (SETQ NEWITEMS (SORT NEWITEMS 'ITEM>))
            (SETQ M
                  (- (LENGTH SETSOFITEMS)
                     (LENGTH (MEMBER NEWITEMS SETSOFITEMS :test
                                     'equal))))
            (COND ((= M (LENGTH SETSOFITEMS))
                   (SETQ SETSOFITEMS
                         (APPEND SETSOFITEMS (LIST NEWITEMS)))))
            (SETQ S (CAAR GOTOS-))
            (COND ((TERM S)
                   (SETQ T1 (PUTASSOC (LIST S (LIST 'S M)) T1)))
                  (T (SETQ T2 (PUTASSOC (LIST S M) T2)))))
      (PRINC T1 F1)
      (TERPRI F1)
      (PRINC T2 F2)
      (TERPRI F2)
)))

(DEFUN ATTACHO (X) (LIST X 0.))

(DEFUN GOTOITEMS (ITEMS S) (GOTO- ITEMS S))

(DEFUN CONSTRUCT NIL
   (SETQ F1 (OPEN '|tbl1.lsp| :DIRECTION :OUTPUT))
   (SETQ F2 (OPEN '|tbl2.lsp| :DIRECTION :OUTPUT))
   (PRINC '|(setf table1 '#(| F1)
   (TERPRI F1)
   (PRINC '|(setf table2 '#(| F2)
   (TERPRI F2)
   (CONSTRUCTSETSOFITEMS)
   (PRINC '|))| F1)
   (TERPRI F1)
   (PRINC '|))| F2)
   (TERPRI F2)
   (CLOSE F1)
   (CLOSE F2)
   (find_all_preterms))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;; Save a list of all preterminals.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defun find_all_preterms ()
   (prog (preterms subel f1)
      (setq f1 (open '|preterm.lsp| :direction :output))
      (princ '|(setq preterms '| f1)
      (terpri f1)
      (do ((n 0 (1+ n)))
         ((>= n (length rules)))
         (do ((subelement (caddr (aref rules n))
                                  (cdr subelement)))
            ((null subelement) nil)
```

```
          (setq subel (car subelement))
          (cond ((and (term subel)
                      (not (member subel preterms)))
                 (push subel preterms))) ) )
    (setq preterms (append preterms '($)))
    (princ preterms f1)
    (terpri f1)
    (princ '|)| f1)
    (close f1)
))
```

UTILITY PROCEDURES

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Utility procedures required by the parsing procedures and
; the LR parsing table construction procedures
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;; resplit in the file "resplit.lsp";;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(DEFUN RESPLIT (N X L)
   (declare (special n L))
   (PROG NIL
      (COND ((EVENP N) (push (CAR (AREF GA X)) L)))
      (COND ((ZEROP N) (RETURN (LIST (LIST (CDR (AREF GA X))
                                           L))))
            (T (RETURN (APPLY 'APPEND
                              (MAPCAR 'RESPLIT-
                                  (CDR (AREF GA X)))))))))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;the following procedures in the file "putils.lsp";;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(DEFUN TERM (X) (EQ #\* (CHAR (STRING X) 0)))

(DEFUN PRINTLINE (X) (PRINC X) (TERPRI))

(DEFUN BISEARCHARRAY (A KEY)
   (BISEARCHARRAY- A KEY 0. (1- (LENGTH A))))

(DEFUN BISEARCHARRAY- (A KEY ST END)
   (PROG (MID MIDKEY)
      (COND ((OR (= ST END) (> ST END))
; error in original, ST was replaced by END
            (COND ((EQUAL KEY (CAR (AREF A END)))
; error in original, ST was replaced by END
                   (RETURN (LIST END)))
                  (T (RETURN NIL)))))
      (SETQ MID (ROUND (+ ST (/ (- END ST) 2.))))
      (SETQ MIDKEY (CAR (AREF A MID)))
      (COND ((EQUAL MIDKEY KEY)
             (DO ((I (1- MID) (1- I)))
                ((OR (< I 0.) (STRING< (string (CAR (AREF A I)))
                                       (string KEY)))
                 (SETQ ST (1+ I))))
             (DO ((I (1+ MID) (1+ I)))
                ((OR (= I (LENGTH A))
                     (STRING< (string KEY) (string (CAR
                                      (AREF A I)))))
                 (SETQ END (1- I))))
             (RETURN (LISTNM ST END))))
      (COND ((STRING< (string MIDKEY) (string KEY))
             (RETURN (BISEARCHARRAY- A KEY (1+ MID) END)))
            (T (RETURN (BISEARCHARRAY- A KEY ST (1- MID)))))
))

(DEFUN LISTNM (ST END)
   (DO ((L NIL (APPEND L (LIST I))) (I ST (1+ I)))
                               ((> I END) L)))

(DEFUN PUTASSOC (X L)
   (COND ((NULL L) (LIST X))
         ((EQUAL (CAAR L) (CAR X))
          (CONS (APPEND (CAR L) (CDR X)) (CDR L)))
         (T (CONS (CAR L) (PUTASSOC X (CDR L))))))

(DEFUN RMOVEDUPL (X Y)
   (COND ((NULL X) NIL)
         ((MEMBER (CAR X) Y) (REMOVEDUPL (CDR X) Y))
```

```
         (T (CONS (CAR X) (REMOVEDUPL (CDR X) Y)))))

;;;;;;;;;return list l minus it's last element;;;;;;;;;;;;;
(defun remlast (l)
   (cond ((null l) nil)
         ((null (cdr l)) nil)
         (t (cons (car l) (remlast (cdr l))))))
```

CONVERSION FUNCTIONS

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;; Convert the translation rules in Tomita's form to
;;;;; translation rules in Petrick's form.  These procedures
;;;;; are in the file "congram.lsp".
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defun convert_trans (rules)
   (prog (temprules)
      (do ((n 0 (1+ n))) ((= n (length rules)))
        (setq temprules (cons (cadr (aref rules n))
                              temprules))
        (setq temprules (cons (cons (caar (aref rules n))
                                    (caddar (aref rules n)))
                              temprules) ) )
      (return temprules) ))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;; Convert the translation rules in Tomita's form to
;;;;; grammar rules in Tomita's form
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defun convert_gram (rules)
   (prog (temprules)
      (setf temprules (make-array (length rules)))
      (do ((n 0 (1+ n))) ((= n (length rules)))
        (setf (aref temprules n) (car (aref rules n))) )
   (return temprules) ))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;; Convert a parse tree in Tomita's form to Petrick's
;;;;; form.  These procedures are in the file "contree.lsp".
;;;;; PARSES must be initialized to nil before calling the
;;;;; main procedure CONVERTTREE.  The resulting parse tree
;;;;; is stored in "parses".
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defun preprocess (tree root temproot)
   (prog (temptree done tempdone)
      (setq done nil)
      (cond ((eq done nil)
             (cond ((> (length (aref tree temproot)) 2)
                    (setf temptree (copy-seq tree))
                    (setq done 't)
                    (do ((subnodes (cdr (aref tree temproot)) (cdr
                                                subnodes)))
                        ((null subnodes) nil)
                        (setf (aref temptree temproot)
                              (cons (car (aref tree temproot))
                                    (list (car subnodes))  ))
                        (converttree temptree root)  ))
                   ((atom (cadr (aref tree temproot)))
                                          (return nil))
                   (t (do ((nextnodes (cadr (aref tree temproot))
                                          (cdr nextnodes)))
                          ((null nextnodes) nil)
                          (setq tempdone (preprocess tree root
                                 (car nextnodes)))
                           (cond (tempdone (setq done 't)))  ))
             )))
   (return done) ))

(defun converttree (tree root)
   (prog ()
      (cond ((eq (preprocess tree root root) nil)
             (setq parses (cons (contree tree root) parses))
            ))
   ))

(defun contree (tree root)
   (prog (nextnodes currentnodename)
      (setq nextnodes (cadr (aref tree root)))
      (setq currentnodename (car (aref tree root)))
```

```
      (cond ((atom nextnodes)
                (return (list (list currentnodename
                                    (LIST (LIST NEXTNODES 'TR))
                                )
                                (LIST (LIST 'T)))) )
            (t (return (cons (list currentnodename)
                             (mymapcar2 'contree tree nextnodes)
                       ))) )))

(defun mymapcar2 (fn arg1 arg2)
   (prog (result)
      (do ((arg2list arg2 (cdr arg2list)))
         ((null arg2list) nil)
         (setq result (append result
                              (list (funcall fn arg1
                                             (car arg2list)))
                       )))
      (return result)
   ))
```

TRANSLATION PROCEDURES

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;; Procedures for returning a SETX expression.
;;;;; In the file "npsinter.lsp".
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(DEFUN NPSINTERP (VAR DET ADJ NP1 compa NP2 conj compb NP2b)
   (COND ((AND (EQ NP1 'CNO)    ;ASKING FOR NEW OR OLD
               ;COURSE NO.
            (OR
               (EQ ADJ 'OLD)
               (EQ ADJ 'NEW) ) )
          (LIST 'SETX (LIST 'QUOTE VAR)
             (COND ((EQ ADJ 'OLD)
                    (cond ((not (equal conj 'empty))
                           (LIST 'QUOTE
                              (list conj
                                 (LIST 'RELATION ''COURSES
                                    ''(OLDNO NEWNO)
                                    (LIST 'QUOTE (LIST VAR NP2))
                                   (list 'quote (list '= compa)))
                                 (LIST 'RELATION ''COURSES
                                    ''(OLDNO NEWNO)
                                    (LIST 'QUOTE (LIST VAR NP2b))
                                   (list 'quote (list '= compb)))
                              ) ) )
                          (t
                           (LIST 'QUOTE
                              (LIST 'RELATION ''COURSES
                                 ''(OLDNO NEWNO)
                                 (LIST 'QUOTE (LIST VAR NP2))
                                 (list 'quote (list '= compa)))
                           ) ) ) )
                   (T
                    (cond ((not (equal conj 'empty))
                           (LIST 'QUOTE
                              (list conj
                                 (LIST 'RELATION ''COURSES
                                    ''(NEWNO oldno)
                                    (LIST 'QUOTE (LIST NP2 var))
                                   (list 'quote (list compa '=)))
                                 (LIST 'RELATION ''COURSES
                                    ''(NEWNO oldno)
                                    (LIST 'QUOTE (LIST NP2b var))
                                   (list 'quote (list compb '=)))
                              ) ) )
                          (t
                           (LIST 'QUOTE
                              (LIST 'RELATION ''COURSES
                                 ''(NEWNO oldno)
                                 (LIST 'QUOTE (LIST NP2 var))
                                 (list 'quote (list compa '=)))
                           ) ))))))
         ((AND       ;ASKING FOR A COURSE DESCRIPTION
             (NULL ADJ)
             ( or (EQ NP1 'DESCRIPTION) (eq np1 'descriptions)) )
          (LIST 'SETX (LIST 'QUOTE VAR)
             (cond ((not (equal conj 'empty))
                    (LIST 'QUOTE
                       (list conj
                          (LIST 'RELATION ''COURSES
                             ''(descrip NEWNO)
                             (LIST 'QUOTE (LIST VAR NP2))
                             (list 'quote (list '= compa)))
                          (LIST 'RELATION ''COURSES
                             ''(descrip NEWNO)
                             (LIST 'QUOTE (LIST VAR NP2b))
                             (list 'quote (list '= compb)))
                       ) ))
                   (t
```

```
           (LIST 'QUOTE
              (LIST 'RELATION ''COURSES
                 ''(descrip NEWNO)
                 (LIST 'QUOTE (LIST VAR NP2))
                 (list 'quote (list '= compa)))
           ) ))))
((AND
    (NULL ADJ)
    ( or (EQ NP1 'prerequisites)
         (eq np1 'prerequisite)))
 (LIST 'SETX (LIST 'QUOTE VAR)
    (cond ((not (equal conj 'empty))
           (LIST 'QUOTE
              (list conj
                 (LIST 'RELATION ''COURSES
                    ''(prereq NEWNO)
                    (LIST 'QUOTE (LIST VAR NP2))
                    (list 'quote (list '= compa)))
                 (LIST 'RELATION ''COURSES
                    ''(prereq NEWNO)
                    (LIST 'QUOTE (LIST VAR NP2b))
                    (list 'quote (list '= compb)))
              ) ))
          (t
           (LIST 'QUOTE
              (LIST 'RELATION ''COURSES
                 ''(prereq NEWNO)
                 (LIST 'QUOTE (LIST VAR NP2))
                 (list 'quote (list '= compa)))
           ) ))))
((AND
    (NULL ADJ)
    (EQ NP1 'cred_hrs))
 (LIST 'SETX (LIST 'QUOTE VAR)
    (cond ((not (equal conj 'empty))
           (LIST 'QUOTE
              (list conj
                 (LIST 'RELATION ''COURSES
                    ''(cred_hrs NEWNO)
                    (LIST 'QUOTE (LIST VAR NP2))
                    (list 'quote (list '= compa)))
                 (LIST 'RELATION ''COURSES
                    ''(cred_hrs NEWNO)
                    (LIST 'QUOTE (LIST VAR NP2b))
                    (list 'quote (list '= compb)))
              ) ))
          (t
           (LIST 'QUOTE
              (LIST 'RELATION ''COURSES
                 ''(cred_hrs NEWNO)
                 (LIST 'QUOTE (LIST VAR NP2))
                 (list 'quote (list '= compa)))
           ) ))))
((AND
    (NULL ADJ)
    ( or (EQ NP1 'name) (eq np1 'names)) )
 (LIST 'SETX (LIST 'QUOTE VAR)
    (cond ((not (equal conj 'empty))
           (LIST 'QUOTE
              (list conj
                 (LIST 'RELATION ''COURSES
                    ''(name NEWNO)
                    (LIST 'QUOTE (LIST VAR NP2))
                    (list 'quote (list '= compa)))
                 (LIST 'RELATION ''COURSES
                    ''(name NEWNO)
                    (LIST 'QUOTE (LIST VAR NP2b))
                    (list 'quote (list '= compb)))
              ) ))
          (t
           (LIST 'QUOTE
              (LIST 'RELATION ''COURSES
```

```
                        ''(name NEWNO)
                        (LIST 'QUOTE (LIST VAR NP2))
                        (list 'quote (list '= compa)))
                    ) ))))
      (T 'NOTYETCODED)
   )
)

(defun compare (comp eqal)
   (cond ((or (equal comp 'greater)
              (equal comp 'higher))
          (read-from-string (concatenate 'string ">" eqal)))
         ((or (equal comp 'lower)
              (equal comp 'less))
          (read-from-string (concatenate 'string "<" eqal)))
         (t nil)
   ))

(defun find_func (op exp)
   (prog (temp)
      (setq temp (gensym))
      (cond ((equal op 'sum)
       (return
       (list 'SETX (LIST 'QUOTE temp)
          (list 'quote
          (list 'total temp
             exp) ) ) ) )
      ((equal op 'average)
       (return
       (list 'SETX (LIST 'QUOTE temp)
          (list 'quote
          (list 'average temp
             exp) ) ) ) ) ) ) )

(defun lessernum (num1 num2)
   (cond ((< num1 num2) num1)
         ((>= num1 num2) num2)
   ))

(defun greaternum (num1 num2)
   (cond ((>= num1 num2) num1)
         ((< num1 num2) num2)
   ))
```

INITIALIZATION PROCEDURES FOR TRANSLATION

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Some of the procedures from the left corner parser as
; provided by Dr. Petrick.  Only needed for initialization
; of variables for translation procedures.  In the file
; "lcp.lsp"
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(DEFUN TRANSF (X Y)
    (PROG NIL
  A   (COND
        ( (NULL Y)
          (RETURN X) ) )
      (SETQ X (CONS (CAR Y) X))
      (SETQ Y (CDR Y))
      (GO A) ) )

(DEFMACRO GETRULES (L)
 `(GET ,L 'RLIST))

(DEFMACRO TERMIN (L)
  `(NULL (CDR (GET ,L 'PRECINF))))

(DEFMACRO NOTPREC (L LL)
  `(NULL (MEMBER ,L (GET ,LL 'PRECINF))))

(DEFUN BLDTREE (L)
    (DECLARE (SPECIAL LISTTAIL))
    (SETQ LISTTAIL (COPY-TREE (CDR L)))
    (PROG (LISTTAIL)
      (SETQ LISTTAIL (COPY-TREE (CDR L)))
      (RETURN (BLDTREE1)) ) )
      ; (RETURN (BLDTREE1)) ) )

(DEFUN BLDTREE1 ()
    (DECLARE (SPECIAL LISTTAIL))
    (PROG (CANDIDATES RULE SLOT RULETAIL)
A   (COND
        ( (NULL LISTTAIL)
          (RETURN NIL) ) )
      (COND
        ( (NULL (CAR LISTTAIL))
          (GO B) ) )
      (SETQ RULE (CAR LISTTAIL))
      (SETQ LISTTAIL (CDR LISTTAIL))
      (SETQ RULETAIL (CDR RULE))
      (COND
        ( (NULL RULETAIL)
          (GO R) )
        ( (CAR RULETAIL)
          (GO R) ) )
      (SETQ SLOT RULETAIL)
  C   (SETQ RULETAIL (CDR RULETAIL))
      (COND
        ( (NULL RULETAIL)
          (GO A) ) )
      (RPLACA RULETAIL (CAR CANDIDATES))
      (SETQ CANDIDATES (CDR CANDIDATES))
      (GO C)
  B   (SETQ LISTTAIL (CDR LISTTAIL))
      (COND
        ( SLOT
          (RPLACA SLOT (BLDTREE1)) )
        ( T
          (SETQ CANDIDATES (CONS (BLDTREE1) CANDIDATES)) ) )
      (COND
        ( (NULL CANDIDATES)
          (RETURN RULE) ) )
      (GO A)
  R   (COND
```

```
        ( (CDR RULE)
          (RETURN RULE) ) )
      (RETURN (CAR RULE)) ) )

(DEFUN ORDERLIST (X)
    (DECLARE (SPECIAL FLAG*))
    (PROG (K NEWX NEWESTX)
  A   (COND
        ( (NULL X)
          (GO D) )
        ( (SETQ K (GETPAIR (CAAR X) NEWX))
          (GO C) ) )
      (SETQ NEWX (CONS (CONS (CAAR X) (LIST (CADAR X))) NEWX))
  B   (SETQ X (CDR X))
      (GO A)
  C   (COND
        ( (MEMBER (CADAR X) K)
          (GO B) ) )
      (SETQ NEWX (COPY2 NEWX (CAAR X) (CONS (CAAR X)
                (CONS (CADAR X) K))))
      (GO B)
  D   (SETQ FLAG* NIL)
      (SETQ NEWESTX (CHECK1 NEWX NEWX))
      (COND
        ( (NULL FLAG*)
          (RETURN NEWX) ) )
      (SETQ NEWX NEWESTX)
      (GO D) ) )

(DEFUN CHECK1 (X Y)
    (DECLARE (SPECIAL FLAG*))
    (PROG (P1)
  A   (COND
        ( (NULL X)
          (RETURN P1) ) )
      (SETQ P1 (CONS (CONS (CAAR X) (CHECK2 (CDAR X) Y))
                                    P1))
      (SETQ X (CDR X))
      (GO A) ) )

(DEFUN CHECK2 (X Y)
    (DECLARE (SPECIAL FLAG*))
    (PROG (P1 P2)
  A   (COND
        ( (NULL X)
          (RETURN P1) )
        ( (SETQ P2 (GETPAIR (CAR X) Y))
          (GO E) ) )
  C   (SETQ P1 (CONS (CAR X) P1))
      (SETQ X (CDR X))
      (GO A)
  E   (COND
        ( (OR (MEMBER (CAR P2) X) (MEMBER (CAR P2) P1))
          (GO D) ) )
      (SETQ P1 (CONS (CAR P2) P1))
      (SETQ FLAG* 'T)
  D   (SETQ P2 (CDR P2))
      (COND
        ( (NULL P2)
          (GO C) ) )
      (GO E) ) )

(DEFUN COPY2 (X Y Z)
    (COND
      ( (NULL X)
        NIL )
      ( (EQ (CAAR X) Y)
        (CONS Z (CDR X)) )
      ( T
        (CONS (CAR X) (COPY2 (CDR X) Y Z)) ) ) )

(DEFUN RESETGR ()
```

```
    (DECLARE (SPECIAL RLIST PRECINF))
    (PROG ()
      (SETQ RLIST 'RLIST)
      (SETQ PRECINF 'PRECINF)
      (PRINT RLIST)
      (RETURN NIL) ) )

(DEFUN SETLCP (RULES)
    (SETCFGRAM 'RLIST 'PRECINF RULES))

(DEFUN SETGR1 (RULES)
    (SETCFGRAM 'RLIST1 'PRECINF1 RULES))

(DEFUN SETCFGRAM (RLIST PRECINF RULES)
    (DECLARE (SPECIAL RLIST PRECINF))
    (PROG (PRPAIRS NODES)
      (SETQ NODES (LISTNODES RULES))
      (SETQ PRPAIRS (ORDERLIST RULES))
      (FIXPLS NODES PRPAIRS)
      (MAPLIST  '(LAMBDA (J) (PROCESS (CAR J))) RULES)
      (RETURN NIL) ) )

(DEFUN FIXPLS (Y Z)
    (DECLARE (SPECIAL Z))
    (PROG NIL
      (MAPLIST  '(LAMBDA (J) (SETF
                       (GET (CAR J) RLIST) NIL)) Y)
      (MAPLIST  '(LAMBDA (J)
         (SETF (GET (CAR J) PRECINF) (CONS (CAR J)
                       (GETPAIR (CAR J) Z))))
        Y)
      (RETURN NIL) ) )

(DEFUN PROCESS (R)
    (PROG (NEWRULE ID N U X Y)
      (SETQ ID (CADR R))
      (COND
        ( (NOT (ATOM ID))
          (SETQ ID (CAR ID)) ) )
      (SETQ NEWRULE
        (CONS
          (COND
            ( (ATOM (CAR R))
              (LIST (CAR R)) )
            ( T
              (CAR R) ) )
          (SETQ U (CONS NIL NIL))) )
      (SETQ R (CDDR R))
  C   (COND
        ( (NULL R)
          (GO D) ) )
      (RPLACD U
        (CONS
          (COND
            ( (ATOM (CAR R))
              (LIST (CAR R)) )
            ( T
              (CAR R) ) )
          NIL) )
      (SETQ U (CDR U))
      (SETQ R (CDR R))
      (GO C)
  D   (SETQ N (LENGTH NEWRULE))
      (SETQ Y (GET ID RLIST))
      (COND
        ( (NULL Y)
          (RETURN (SETF (GET ID RLIST) (CONS NEWRULE NIL)))
                      ) )
      (SETQ X (CONS NIL Y))
  A   (COND
        ( (NULL Y)
          (GO B) ) )
```

```
      (COND
        ( (EQUAL NEWRULE (CAR Y))
          (GO RET) ) )
      (COND
        ( (< N (LENGTH (CAR Y)))
          (GO B) ) )
      (SETQ X Y)
      (SETQ Y (CDR Y))
      (GO A)
  B   (RPLACD X (CONS NEWRULE Y))
      (COND
        ( (NULL (CAR X))
          (SETF (GET ID RLIST) (CDR X)) ) )
  RET (RETURN NIL) ) )

(DEFUN GETPAIR (X Y)
    (PROG NIL
  A   (COND
        ( (NULL Y)
          (RETURN NIL) )
        ( (EQ (CAAR Y) X)
          (RETURN (CDAR Y)) ) )
      (SETQ Y (CDR Y))
      (GO A) ) )

(DEFUN LISTNODES (RULES)
    (PROG (X Y)
  D   (COND
        ( (NULL RULES)
          (RETURN Y) ) )
      (SETQ X (CAR RULES))
  C   (COND
        ( (NULL X)
          (GO A) ) )
      (COND
        ( (MEMBER (CAR X) Y)
          (GO B) ) )
      (SETQ Y (CONS (CAR X) Y))
  B   (SETQ X (CDR X))
      (GO C)
  A   (SETQ RULES (CDR RULES))
      (GO D) ) )
```

KNUTH TRANSLATION PROCEDURES

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;; Dr. Petrick's translation procedures modified to work
;;;;; with my parser.  In the file "knuth.lsp"
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(DEFUN PUTPROPP (ATOM ATTRIBUTE VALUE)
   (PROG NIL
      (COND
        ( (NULL PRTR)
          (GO A) ) )
      (PRINT ATOM)
      (PRIN1 SPACER)
      (PRIN1 ATTRIBUTE)
      (PRIN1 SPACER)
      (PPRINT VALUE)
  A (RETURN
      (SETF (GET ATOM ATTRIBUTE) VALUE))
 )         )


(DEFUN ADDSYNTACTICFEATS (SYMBOL FEATS)
    (PROG (COORDS NOCO)
      (SETQ COORDS (EXTRACTCOORDS FEATS))
      (SETQ NOCO (EXTRACT_NO_AND_CO FEATS))
  A   (COND
        ( (NULL FEATS)
          (RETURN SYMBOL) )
        ( (OR (NULL (CAR FEATS)) (NULL (GET (CADAR FEATS)
                          'ATTRIBUTE)))
          (GO B) )
        ( (EQ (CAAR FEATS) '=)
          (SETF (GET SYMBOL (CADAR FEATS))
                 (ORDECIDE (CAR FEATS) COORDS NOCO)) )
        ( T
          (SETF (GET SYMBOL (CADAR FEATS)) (CAAR FEATS)) )
        )
  B   (SETQ FEATS (CDR FEATS))
      (GO A) ) )

(DEFUN ORDECIDE (FEAT COORDS NOCO)
    (PROG (TEMP FEATVAL)
      (SETQ FEATVAL (CADDR FEAT))
      (COND
        ( (OR
            (ATOM FEATVAL)
            (NOT (EQ (CAR FEATVAL) 'ORA))
            (LISTWITHATOM (CDR FEATVAL)))
          (RETURN FEATVAL) ) )
      (SETQ TEMP (MAPCAR 'CADR (CDR FEATVAL)))
      (COND
        ( (AND (EQ (CADR FEAT) 'COLN) (CDDR FEATVAL))
          (SETQ ORACELL FEATVAL) ) )
      (RETURN (CADR FEATVAL)) ) )

(DEFUN MYEVAL (FORM)
    (COND
      ( (NUMBERP FORM)
        (NUMLOOK FORM TREES) )
      ( (EQ FORM 'RHS)
        (CDR TREES) )
      ( (ATOM FORM)
       FORM )
      ( (EQ (CAR FORM) 'QUOTE)
        (CADR FORM) )
      ( (GET (CAR FORM) 'ATTRIBUTE)
        (GET (CADR (NUMLOOK (CADR FORM) TREES)) (CAR FORM)) )
      ( T (EVAL
            (CONS (CAR FORM)
                (MAPCAR 'QUOTEMEMBER (MAPCAR 'MYEVAL
```

```
                        (CDR FORM)))))
         ) ) )

(DEFUN UPDATEATTRIBUTES (TREE)
    (DECLARE (SPECIAL TREES))
    (PROG (TRULES ATTS)
      (COND
       ( (MEMBER (CAR TREE) EXEMPTEDNODES)
         (RETURN TREE) ) )
      (SETQ TRULES (TRULELOOKUP (TOPSTRUCT TREE)))
      (SETQ TREES (CONS TREE (CDDR TREE)))
  A   (COND
       ( (NULL TRULES)
         (SETQ K* NIL)
         (RETURN TREE) ) )
      (COND
        ( (EQ (CADR (CAAR TRULES)) 'ALL_SONS)
         (SETQ TRULES
           (NCONC (ALL_SONS_MACRO (CAR TRULES) TREES)
                  (CDR TRULES)) ) ) )
      (SETQ ATTS (CADR (NUMLOOK (CADR (CAAR TRULES))
                                TREES)))
      (COND
       ( (EQ ATTS '*K*)
         (GO B) )
       ( (GET ATTS (CAAAR TRULES))
         (GO B) )
       ( (ALLATTRIBUTESEXIST (LIST (CADAR TRULES)) TREES)
         (PUTPROPP ATTS (CAAAR TRULES) (MYEVAL (CADAR
               TRULES))) ) )
  B   (SETQ TRULES (CDR TRULES))
      (GO A) ) )

(DEFUN ALL_SONS_MACRO (FORM TREE)
    (PROG (NO EXPFORM FEAT)
      (SETQ NO 1)
      (SETQ FEAT (CAAR FORM))
      (SETQ FORM (CDR FORM))
  A   (SETQ TREE (CDR TREE))
      (COND
       ( (NULL TREE)
         (RETURN EXPFORM) ) )
      (SETQ EXPFORM (CONS (CONS (LIST FEAT NO) FORM)
                                EXPFORM))
      (SETQ NO (1+ NO))
      (GO A) ) )

(SETQ EXEMPTEDNODES NIL)

(DEFUN TRULELOOKUP (RULE)
    (PROG (RL)
      (SETQ RL TRANSLATIONRULES)
  A   (COND
       ( (NULL RL)
         (ERROR (FORMAT NIL "~A" (APPEND
                     '(INCORRECT PRODUCTION RULE)
                                       RULE))) )
       ( (RULESTRUCTMATCH RULE (CAR RL))
         (RETURN (CADR RL)) ) )
      (SETQ RL (CDDR RL))
      (GO A) ) )

(DEFUN ALLATTRIBUTESEXIST (EXPRESSIONS TREES)
    (COND
      ( (NULL EXPRESSIONS)
       'T )
      ( (ATOM (CAR EXPRESSIONS))
       (ALLATTRIBUTESEXIST (CDR EXPRESSIONS) TREES) )
      ( (EQ (CAAR EXPRESSIONS) 'QUOTE)
       (ALLATTRIBUTESEXIST (CDR EXPRESSIONS) TREES) )
      ( (GET (CAAR EXPRESSIONS) 'ATTRIBUTE)
       (COND
```

```
          ( (GET (CADR (NUMLOOK (CADAR EXPRESSIONS) TREES))
                       (CAAR EXPRESSIONS))
            (ALLATTRIBUTESEXIST (CDR EXPRESSIONS) TREES) )
          ( T  NIL ) ) )
       ( T (AND
          (ALLATTRIBUTESEXIST (CDAR EXPRESSIONS) TREES)
          (ALLATTRIBUTESEXIST (CDR EXPRESSIONS) TREES)) ) ) )

(DEFUN QUOTEMEMBER (M)
   (CONS 'QUOTE (CONS M NIL)))

(DEFUN SYNTRAN (TREE FEATURES LIMIT)
    (PROG (TEMP FEATS)
  A   (COND
        ( (EQ LIMIT 0)
          (RETURN '(LIMIT EXCEEDED)) ) )
      (SETQ FEATS FEATURES)
      (SETQ TEMP NIL)
      (ONCEUPANDOWN TREE)
  B   (SETQ TEMP (CONS (GET (CADR TREE) (CAR FEATS)) TEMP))
      (COND
        ( (NULL (CAR TEMP))
          (SETQ LIMIT (1- LIMIT))
          (GO A) ) )
      (SETQ FEATS (CDR FEATS))
      (COND
        ( FEATS
          (GO B) ) )
      (RETURN (REVERSE TEMP)) ) )

(DEFUN RULESTRUCTMATCH (STRUCTURE RULE)
    (PROG (TEMP)
  A   (COND
        ( (AND RULE (EQ (CAR RULE) '*))
          (GO E) )
        ( (NULL STRUCTURE)
          (GO B) )
        ( (NULL RULE)
          (RETURN NIL) )
        ( (OR (EQUAL (CAR RULE) (CAR STRUCTURE))
                     (EQ (CAR RULE) 'VX))
          (GO C) ) )
      (COND
        ( (EQ (CAR RULE) 'XX)
          (RETURN 'T) ) )
      (COND
        ( (AND (CONSP (CAR RULE)) (EQ (CAAR RULE) 'OR))
          (GO D) ) )
      (RETURN NIL)
  D   (SETQ TEMP (CDAR RULE))
      (COND
        ( (MEMBER (CAR STRUCTURE) TEMP)
          (GO C) ) )
      (RETURN NIL)
  E   (COND
        ( (AND
            STRUCTURE
            (OR
              (EQUAL (CAR STRUCTURE) (CADR RULE))
              (AND (CONSP (CADR RULE)) (MEMBER (CAR STRUCTURE)
                       (CDADR RULE))))
            (RULESTRUCTMATCH (CDR STRUCTURE) RULE))
          (SETQ K* 'T)
          (RETURN 'T) ) )
      (SETQ RULE (CDDR RULE))
      (GO A)
  B   (COND
        ( (NULL RULE)
          (RETURN 'T) ) )
      (RETURN NIL)
  C   (SETQ STRUCTURE (CDR STRUCTURE))
      (SETQ RULE (CDR RULE))
```

```
      (GO A) ) )

(DEFUN NUMLOOK (NO TREES)
    (PROG (NBR L)
      (COND
        ( (NOT (NUMBERP NO))
          (ERROR (FORMAT NIL "~A" (CONS '(NUMLOOK ERROR)
                (LIST NO TREES)))) ) )
      (SETQ NBR NO)
      (SETQ L TREES)
  A   (COND
        ( (ZEROP NBR)
          (RETURN (CAR L)) ) )
      (SETQ L (CDR L))
      (SETQ NBR (1- NBR))
      (COND
        ( (AND (NULL L) K*)
          (RETURN '(*K* *K*)) )
        ( (NULL L)
          (ERROR (FORMAT NIL "~A" (CONS '(NUMLOOK ERROR)
                (LIST NO TREES)))) ) ) (GO A) ) )

(DEFUN ADDFEATNODES (TREE)
    (PROG (ID)
      (COND
        ( (NULL (CDR TREE))
          (RETURN TREE) ) )
      (SETQ ID
        (PROGN
            (SETQ NODENUMBER (1+ NODENUMBER))
        (read-from-string (concatenate 'string "n"
                 (format nil "~a" NODENUMBER) #'char-equal)) ) )
      (RETURN
       (CONS
         (CAAR TREE)
         (CONS
           (COND
             ( (CDAR TREE)
                (ADDSYNTACTICFEATS ID (CADAR TREE)) )
               ( T ID ) )
             (MAPCAR 'ADDFEATNODES (CDR TREE))))) ) )

  (defun addfeatnodes1 (tree) (prog (id)
    (cond ((atom tree) (return tree)))
    (setq id (progn
      (setq nodenumber (1+ nodenumber))
      (read-from-string (string-append "n" (format nil "~a"
        nodenumber))) ) )
    (return
      (cons
        (car tree)
        (cons id (mapcar 'addfeatnodes1 (cdr tree)))))))

(DEFUN EXTRACT_NO_AND_CO (FEATS)
    (PROG (NO CO)
  A   (COND
        ( (OR (NULL FEATS) (AND NO CO))
          (RETURN (CONS NO CO)) )
        ( (AND (EQ (CAAR FEATS) '=) (EQ (CADAR FEATS) 'NAMEOF))
          (SETQ NO (CADDAR FEATS)) )
        ( (AND (EQ (CAAR FEATS) '=) (EQ (CADAR FEATS) 'CLASSOF))
          (SETQ CO (CADDAR FEATS)) ) )
      (SETQ FEATS (CDR FEATS))
      (GO A) ) )

(DEFUN ONCEUPANDOWN (TREE)
    (PROG NIL
      (COND
        ( (OR (ATOM TREE) (NULL (CDR TREE)))
          (RETURN TREE) ) )
      (SETQ TREE (UPDATEATTRIBUTES TREE))
      (RETURN
```

```
        (UPDATEATTRIBUTES (CONS (CAR TREE)
                                (MAPCAR 'ONCEUPANDOWN (CDR TREE)))))
         ) )

(DEFUN TOPSTRUCT (TREE)
    (PROG (BRANCHES BNODES)
      (SETQ BRANCHES (CDDR TREE))
  A   (COND
        ( (NULL BRANCHES)
          (RETURN (CONS (CAR TREE) (REVERSE BNODES))) )
        ( (NULL (CDAR BRANCHES))
          (SETQ BNODES (CONS (CAAAR BRANCHES) BNODES)) )
        ( T (SETQ BNODES (CONS (CAAR BRANCHES) BNODES)) ) )
      (SETQ BRANCHES (CDR BRANCHES))
      (GO A) ) )

(SETQ SPACER '| -| )

(DEFUN RECLAIMGATOMS (TREES)
    (PROG (A)
  A   (COND
        ( (NULL TREES)
          (RETURN A) )
        ( (NULL (CDAR TREES))
          (PROG2 (SETQ TREES (CDR TREES)) (GO A)) ) )
      (SETQ A (CONS (CADAR TREES) A))
      ;(REMALLPROPS (CAR TREES))
      (REMPROPS (CADAR TREES) (SYMBOL-PLIST (CADAR TREES)))
      (SETQ TREES (NCONC (CDDAR TREES) (CDR TREES)))
      (GO A) ) )

(defun remprops (symbol proplist)
   (do ((prop proplist (cddr prop)))
       ((null prop) nil)
       (remprop symbol (car prop)) ))

(DEFUN LISTWITHATOM (L)
    (PROG NIL
  A   (COND
        ( (ATOM L)
          (RETURN NIL) )
        ( (ATOM (CAR L))
          (RETURN 'T) ) )
      (SETQ L (CDR L))
      (GO A) ) )

(DEFUN EXTRACTCOORDS (FEATS)
    (PROG NIL
  A   (COND
        ( (NULL FEATS)
          (RETURN NIL) )
        ( (NULL (CAR FEATS))
          (GO B) )
        ( (AND (EQ (CAAR FEATS) '=) (EQ (CADAR FEATS) 'COORDS))
          (RETURN (CADDAR FEATS)) ) )
  B   (SETQ FEATS (CDR FEATS))
      (GO A) ) )

(DEFUN KTRANSBT (STRING FEAT NODE LIMIT)
    (PROG (X TRANS)
       (SETQ NODENUMBER 0)
  A   (COND
        ( (NULL PARSES)
          (RETURN TRANS) ) )
      (SETQ X (ADDFEATNODES (CAR PARSES)))
    (COND
      (PRTR
         (PPRINT X)
       ) )
      (SETQ TRANS (CONS (CAR (SYNTRAN X (LIST FEAT) LIMIT))
                        TRANS))
      (RECLAIMGATOMS (CONS X NIL))
```

```
      (SETQ PARSES (CDR PARSES))
      (GO A) ) )

(DEFUN GETCFRULES (TRULES)
    (PROG (RULES)
  A   (COND
        ( (NULL TRULES)
          (RETURN (REVERSE RULES)) ) )
      (SETQ RULES (CONS (CAR TRULES) RULES))
      (SETQ TRULES (CDDR TRULES))
      (GO A) ) )

(defun ktranruzzo (string feat node limit)
        (prog (parses trans x)
   (setq parses (rparse string node))
   (setq nodenumber 0)
 a (cond ((null parses) (return trans)))
   (setq x (addfeatnodes (dressformat (car parses))))
   (cond (prtr (pprint x) (terpri)))
   (setq trans (cons (car (syntran x (list feat) limit))
                     trans))
   (reclaimgatoms (cons x nil))
   (setq parses (cdr parses))
   (go a) ) )

(defun dressformat (tree)
  (cond
    ((atom tree) (list (list tree)))
    (t (cons (list (car tree)) (mapcar 'dressformat
                    (cdr tree))))))
(setq prtr nil)
```

LOGICAL FORM TO PRE-SQL TRANSLATION PROCEDURES

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;; Procedures used in the conversion from SETX to
;;;;; Pre-SQL.  In the file "newtos.lsp"
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun SEPORS (FORM HIGHERVARS)
   (COND ((EQ (CAR FORM) 'OR)
          (MAPCAR2ARGS (CDR FORM) 'SEPARATERELATIONS
                       HIGHERVARS))
         (T (LIST (SEPARATERELATIONS FORM HIGHERVARS)))))

(SETQ SQLFCTPREDS '((QUANTITY COUNT) (TOTAL SUM)
               (AVERAGE AVG) (MAX MAX) (MIN MIN)))

(defun LESSTHAN (X Y) (< X Y))

(defun GREATERTHAN (X Y) (> X Y))

(defun LESSTHANEQ (X Y) (OR (EQUAL X Y) (< X Y)))

(defun GREATERTHANEQ (X Y) (OR (EQUAL X Y) (> X Y)))

(defun mapcar2args (fn l extraarg)
   (cond ((null l) nil)
         (t (cons (apply fn (list (car l) extraarg))
                  (mapcar2args fn (cdr l) extraarg)))))

(defun SEPARATERELATIONS (FORM HIGHERVARS)
   (PROG (RELATIONS EQPREDCONJUNCTS NULLSETS
                    NONNULLSETS SQLFCTS)
      (COND ((EQ (CAR FORM) 'RELATION) (SETQ FORM
                                (LIST FORM)))
            ((EQ (CAR FORM) 'AND) (SETQ FORM (CDR FORM)))
            (T (ERROR "SEPARATEERROR")))
A     (COND ((EQ (CAAR FORM) 'RELATION)
             (SETQ RELATIONS (CONSRELATION (CAR FORM)
                                           RELATIONS)))
            ((ASSOC (CAAR FORM) SQLFCTPREDS)
             (SETQ SQLFCTS (CONS (CAR FORM) SQLFCTS)))
            ((MEMBER (CAAR FORM) EQPREDS)
             (SETQ EQPREDCONJUNCTS (CONS (CAR FORM)
                                         EQPREDCONJUNCTS)))
            ((AND
                (EQ (CAAR FORM) 'NOT*)
                (EQ (CAADAR FORM) 'SETX))
             (SETQ NULLSETS (CONS (CADAR FORM) NULLSETS)) )
            ((EQ (CAAR FORM) 'SETX)
             (SETQ NONNULLSETS (CONS (CAR FORM)
                                     NONNULLSETS)))
            (T (ERROR "SEPARATEERROR")))
      (SETQ FORM (CDR FORM))
      (COND (FORM (GO A)))
B     (setq eqpredconjuncts (eqpredproc eqpredconjuncts
                       sqlfcts highervars))
      (SETQ FORM (ANDOFRELATIONSPROC (REVERSE RELATIONS) ))
      (SETQ HIGHERVARS (ABSORB (CADR FORM) highervars))
      (return
      (list
         (CAR FORM)
         highervars
         (COND ((CDDR FORM)
             (append (translateeqpreds
                     eqpredconjuncts
                     (ordunion (cadr form) HIGHERVARS))
                  (caddr form)))
            (T EQPREDCONJUNCTS))
         (MAPCAR2ARGS NULLSETS 'LFTOSQL1 HIGHERVARS)
         (MAPCAR2ARGS NONNULLSETS 'LFTOSQL1 HIGHERVARS))) ))
```

```
(defun CONSRELATION (REL RELS)
   (PROG (ANS RCOPY)
      (SETQ RCOPY RELS)
A     (COND ((NULL RELS) (RETURN (CONS REL (REVERSE ANS))))
            ((RELSUBSUME (CAR RELS) REL) (RETURN RCOPY))
            ((NOT (RELSUBSUME REL (CAR RELS)))
             (SETQ ANS (CONS (CAR RELS) ANS))))
      (SETQ RELS (CDR RELS))
      (GO A) ))

(defun RELSUBSUME (REL1 REL2)
   (PROG ()
      (COND ((NOT (EQ (CADADR REL1) (CADADR REL2)))
                      (RETURN NIL)))
      (SETQ REL1 (MAKETRIPLES (CADR (CADDR REL1))
                     (CADR (CADDDR REL1))
                     (CADAR (CDDDDR REL1))))
      (SETQ REL2 (MAKETRIPLES (CADR (CADDR REL2))
                     (CADR (CADDDR REL2))
                     (CADAR (CDDDDR REL2))))
 A    (COND ((NULL REL2) (RETURN T))
            ((MEMBER (CAR REL2) REL1)
             (SETQ REL2 (CDR REL2)) (GO A)))
      (RETURN NIL) ))

(defun SAMESQLFCTVARS (FORM)
   (COND ((AND
           FORM
           (CDR FORM)
           (EQ (CAAR FORM) (CAADR FORM))
           (EQ (CADAR FORM) (CADADR FORM)))
       FORM)
      (T NIL)))

(defun CHANGESQLVAR (FORM)
   (CONS
      (CONS (CAAR FORM) (CONS (MYGENSYM) (CDDAR FORM)))
      (CDR FORM)))

(defun ADDEQPRED (X)
   (LIST (QUOTE EQUAL) (CADAR X) (CADADR X)))

(defun INSERTINTO2 (TABL PRED)
   (COND ((ASSOC (CADR PRED) TABL)
       (ERROR "INSERTINTO2ERROR"))
      (T (CONS
            (LIST
            (CADR PRED)
            (CONS (CAR PRED)
               (VARSTOFIELDS (CDDR PRED) TABL)))
            TABL))))

(defun ABSORB (VARS1 VARS2)
   (PROG (X)
A     (COND ((NULL VARS1) (RETURN VARS2)))
      (SETQ X (CONS (CAAR VARS1) (REVERSE (CDAR VARS1))))
B     (SETQ VARS2 (UPDATEVARS (CAR X) (CADR X) VARS2))
      (COND ((CDDR X) (SETQ X (CONS (CAR X) (CDDR X)))
          (GO B)))
      (SETQ VARS1 (CDR VARS1))
      (GO A) ))

(defun ANDOFRELATIONSPROC (RELATIONS)
   (PROG (RELIST)
      (COND ((NULL RELATIONS) (RETURN (LIST NIL NIL ))))
      (SETQ RELIST (ADDTUPLEVARS (MAKEPAIRS RELATIONS)))
      (RETURN (CONS (REMDUPS RELIST)
              (PRODUCEVARPROPSANDCONSTRAINTS RELATIONS
                              RELIST))) ))

(defun GENEXTRAS (RELATIONS VARS)
   (PROG (P U V W)
```

```
      (declare (special extraoutatts))
      (SETQ P (SETQ U (FINDPRS RELATIONS VARS)))
A     (COND ((NULL U) (RETURN (REVERSE V)))
         ((AND
          (ATOM (CADAR U))
          (SETQ W (EQUALASSOC (CAAR U) EXTRAOUTATTS)))
          (SETQ V (NCONC (GETEXTRAS (CADAR U) W P) V))))
      (SETQ U (CDR U))
      (GO A) ))

(defun SIMPRELS (FORM) (PROG (TEMP FORM1)
   (SETQ FORM1 FORM)
   (SETQ TEMP (FINDEQREL FORM))
   (COND ((OR (NULL (CAR TEMP)) (NULL (CDR TEMP)))
                      (RETURN FORM))
      ((OR (AND (ATOM (CAAR TEMP)) (ATOM (CADAR TEMP))
        (> (IDTONUMBER (CAAR TEMP))
               (IDTONUMBER (CADAR TEMP))))
           (AND (ATOM (CAAR TEMP)) (NOT (ATOM
                      (CADAR TEMP)))))
       (RETURN (MUSTHAVEVAR (SUBSTOP (CADAR TEMP)
                      (CAAR TEMP)
           (CADDDR (CAR TEMP)) (CADDAR TEMP)
           (SIMPRELS (CDR TEMP))) FORM1)))
      (T (RETURN (MUSTHAVEVAR
          (SUBSTOP (CAAR TEMP) (CADAR TEMP) (CADDAR TEMP)
            (CADDDR (CAR TEMP)) (SIMPRELS
            (CDR TEMP))) FORM1))))))

(defun FINDEQREL (FORM)
   (PROG (TEMP L)
A     (COND ((NULL FORM) (RETURN (QUOTE (NIL NIL))))
         ((AND (EQ (CAAR FORM) 'RELATION)
          (SETQ TEMP (CADADR (CDAR FORM)))
          (CDR TEMP) (NULL (CDDR TEMP))
          (EQ (CAR TEMP) (CADR TEMP)))
          (RETURN (CONS (APPEND (CADADR (CDDAR FORM))
                     (CADADR (CDDDAR FORM)))
                  (APPEND (REVERSE L) (CDR FORM))))))
      (SETQ L (CONS (CAR FORM) L))
      (SETQ FORM (CDR FORM))
      (GO A) ))

(defun SUBSTOP (X Y OPX OPY Z)
   (PROG (L)
      (COND ((AND (EQ OPX (QUOTE =)) (EQ OPY (QUOTE =)))
          (RETURN (SUBST X Y Z))))
A     (COND ((NULL Z) (RETURN L))
         ((AND (EQ (CAAR Z) (QUOTE RELATION))
            (MEMBER Y (CADR (CADDDR (CAR Z)))))
          (MODPTRS (CDDDAR Z) X Y OPX OPY)))
      (SETQ L (CONS (CAR Z) L))
      (SETQ Z (CDR Z))
      (GO A) ))

 (defun MODPTRS (FORM X Y OPX OPY) (PROG (U V)
   (SETQ U (CADAR FORM))
   (SETQ V (CADADR FORM))
 A (COND ((OR (NULL U) (NULL V)) (RETURN NIL))
      ((AND (EQ (CAR U) Y) (EQ (CAR V) OPY))
       (RPLACA U X)
       (RPLACA V OPX)))
   (SETQ U (CDR U))
   (SETQ V (CDR V))
   (GO A) ))

(defun lftosql1 (form highervars)
   (prog (vars topstring partran temp temp1 operator gvars)
      (declare (special vars intstartno))
      (setq tvs '(a b c d e f g h i j k l m n o p q r s t
                     u v w x y z))
      (setq intstartno 100)
```

```
      (cond ((or (eq (car form) 'setx)
                 (eq (car form) 'bagx))
             (setq topstring 'whq)))
a     (cond ((or (eq (car form) 'setx)
                 (eq (car form) 'bagx))
             (cond ((and (setq temp (cadr (assoc (caadr
                                 (caddr form)) sqlfctpreds)))
                         (eq (cadadr form) (cadadr
                                           (caddr form))))
                    (setq operator temp)
                    (SETQ FORM (CADDR (CADR (CADDR FORM))))
                    (go a))
                   (t nil))
             (cond ((and (or (eq (caadr (caddr form)) 'setx)
                             (eq (caadr (caddr form))
                                                 'bagx))
                         (eq (caadr (caddr (cadr
                                           (caddr form)))) 'and)
                         (setq temp (cadr (assoc (caadr
                               (cadr (caddr (cadr (caddr
                               form))))) sqlfctpreds))))
                    (setq form (list (car form) (cadr form)
                               (list (caaddr form)
                                                   (cons 'and
                               (cdaddr form))))))
                   (t nil))
             (cond (highervars
                    (setq vars (cons (list (car form)
                               (cadadr form)) vars)))
                   (t (setq vars (cons (list
                                 (car form) (cadadr form)) vars))))
             (setq form (cadr (caddr form)))
             (go a)))
b     (setq partran (sepors form highervars))
      (return (assemblesqllist vars operator partran
                               topstring))))

(defun ADDTUPLEVARS (L) (PROG (INDEX IVALUE ANSLIST)
  A (SETQ IVALUE (EQUALASSOC (CAR L) INDEX))
    (COND (IVALUE (SETQ ANSLIST (CONS IVALUE ANSLIST)))
       (T (SETQ ANSLIST (CONS (LIST (CAR L) (CAR TVS))
                   ANSLIST))
          (SETQ TVS (CDR TVS))
          (SETQ INDEX (CONS (CAR ANSLIST) INDEX))))
    (SETQ L (CDR L))
    (COND (L (GO A)))
    (SETQ IVALUE NIL)
    (SETQ INDEX ANSLIST)
  B (SETQ IVALUE (CONS (LIST (CAAAR INDEX) (CADAR INDEX))
                               IVALUE))
    (SETQ INDEX (CDR INDEX))
    (COND (INDEX (GO B)))
    (RETURN IVALUE) ))

(setq tvs '(A B C D E F G H I J K))

(defun MAKEPAIRS (RELATIONS)
  (COND ((NULL RELATIONS) NIL)
     (T (CONS
           (LIST (CADADR (CAR RELATIONS))
              (EXTRACTKEY (CAR RELATIONS))
              (GETEQVARS (CAR RELATIONS)))
           (MAKEPAIRS (CDR RELATIONS))))) )

(defun GETEQVARS (RELATION) (PROG (X Y Z)
   (SETQ X (CADR (CADDR RELATION)))
   (SETQ Y (CADR (CADDDR RELATION)))
 A (COND ((NULL X) (RETURN (MAPCAR (QUOTE CDR) Z)))
       ((ASSOC (CAR X) Z)
        (SETQ Z (UPDATEZ (CAR X) (CAR Y) Z)))
       (T (SETQ Z (CONS (LIST (CAR X) (CAR Y)) Z))))
   (SETQ X (CDR X))
```

```
   (SETQ Y (CDR Y))
   (GO A) ))

(defun UPDATEZ (A B C)
  (COND ((NULL C) NIL)
     ((EQ A (CAAR C))
      (CONS (CONS A (CONS B (CDAR C))) (CDR C)))
     (T (CONS (CAR C) (UPDATEZ A B (CDR C))))))

(defun EQUALASSOC (X Y)
  (COND ((NULL Y) NIL)
     ((EQUAL X (CAAR Y)) (CAR Y))
     (T (EQUALASSOC X (CDR Y)))))

(defun EXTRACTKEY (RELATION)
   (PROG (RELNAME KEY KEYATTRIBUTES TEMP RELATTS RELARGS
                                RELOPS)
      (SETQ RELNAME (CADADR RELATION))
      (SETQ KEYATTRIBUTES (GET RELNAME (QUOTE KEY)))
      (SETQ RELATTS (CADR (CADDR RELATION)))
      (SETQ RELARGS (CADR (CADDDR RELATION)))
      (SETQ RELOPS (CADADR (CDDDR RELATION)))
      (COND ((AND (CDR RELATTS)
                  (EQ (CAR RELATTS) (CADR RELATTS))
                  (EQUAL RELOPS (QUOTE ('= '=)))
                  (NOT (ATOM (CAR RELARGS))))
             (SETQ RELARGS (CONS (CADR RELARGS)
                                 (CONS (CAR RELARGS)
                                       (CDDR RELARGS))))))
A     (SETQ TEMP (SEARCHFOR (CAR KEYATTRIBUTES) RELATTS
                        RELARGS RELOPS))
      (COND ((NULL TEMP) (RETURN (LIST (MYGENSYM)))))
      (SETQ KEY (CONS TEMP KEY))
      (SETQ KEYATTRIBUTES (CDR KEYATTRIBUTES))
      (COND (KEYATTRIBUTES (GO A)))
      (RETURN KEY) ))

(defun SEARCHFOR (KEYAT RELATTS RELARGS RELOPS)
   (COND ((NULL RELATTS) NIL)
         ((AND
             (EQ KEYAT (CAR RELATTS))
             (EQ (CAR RELOPS) (QUOTE $$.=.)))
      (CAR RELARGS))
   (T (SEARCHFOR KEYAT (CDR RELATTS) (CDR RELARGS)
                       (CDR RELOPS)))) )

(setq EQTABLE '(('< LESSTHAN) ('> GREATERTHAN)
                       ($$.<=. LESSTHANEQ)
      ($$.>=. GREATERTHANEQ) ('= EQUAL) ($$.^=. UNEQUAL)))

(defun REMDUPS (L) (PROG (ANS)
  A (COND ((NULL L) (RETURN (REVERSE ANS)))
       ((NOT (MEMBER (CAR L) ANS)) (SETQ ANS (CONS
                       (CAR L) ANS))))
    (SETQ L (CDR L))
    (GO A) ))

(defun PRODUCEVARPROPSANDCONSTRAINTS (RELATIONS TABLE)
   (declare (special table))
  (PROG (REL RELATTS RELARGS RELOPS VARPS CONSTRAINTS)
  A (SETQ RELATTS (CADR (CADDAR RELATIONS)))
    (SETQ RELARGS (CADR (CADDDR (CAR RELATIONS))))
    (SETQ RELOPS (CADR (CADDDR (CDAR RELATIONS))))
    (SETQ REL (CDAR TABLE))
    (COND ((NULL REL) (SETQ REL (CAR TABLE))))
B
    (cond ((and (atom (car relargs)) (not (numberp
                       (car relargs))))
        (SETQ VARPS (UPDATEVARS (CAR RELARGS)
     (INCORPORATE (CONS (CAR RELATTS) REL)
                       (CAR RELOPS)) VARPS)))
       (T (SETQ CONSTRAINTS (CONS (ADJUSTBETWEENARGS
```

```
                        (LIST (CONS (CAR RELATTS) REL)
          (CAR RELOPS)
          (CHANGEINTTOBETWEEN (CAR RELARGS))))
                       CONSTRAINTS))))
    (SETQ RELATTS (CDR RELATTS))
    (SETQ RELARGS (CDR RELARGS))
    (SETQ RELOPS (CDR RELOPS))
    (COND (RELATTS (GO B)))
    (SETQ RELATIONS (CDR RELATIONS))
    (SETQ TABLE (CDR TABLE))
    (COND (RELATIONS (GO A)))
    (RETURN (LIST VARPS CONSTRAINTS)) ))

(defun INCORPORATE (ATT RELOP)
  (COND ((EQ RELOP (QUOTE $$.^=.)) (CONS RELOP ATT))
     (T ATT)))

(defun CHANGEINTTOBETWEEN (CONDITION)
    (cond ((numberp condition) condition)
       ((EQ (CAR CONDITION) (QUOTE INT))
        (CONS (QUOTE BETWEEN) (CADR CONDITION)))
       (T (CADR CONDITION))))

(defun MAKETRIPLES (X Y Z)
  (COND ((NULL X) NIL)
     (T (CONS (LIST (CAR X) (CAR Y) (CAR Z))
           (MAKETRIPLES (CDR X) (CDR Y) (CDR Z))))))

(defun NOTSIMP* (X) (COND
  ((EQ X (QUOTE GREATERTHAN)) (QUOTE LESSTHANEQ))
  ((EQ X (QUOTE GREATERTHANEQ)) (QUOTE LESSTHAN))
  ((EQ X (QUOTE LESSTHAN)) (QUOTE GREATERTHANEQ))
  ((EQ X (QUOTE LESSTHANEQ)) (QUOTE GREATERTHAN))
  (T (QUOTE NOTSIMP*ERROR))))

(defun NOTSIMP** (X) (COND
  ((EQ X (QUOTE GREATERTHAN)) (QUOTE LESSTHAN))
  ((EQ X (QUOTE GREATERTHANEQ)) (QUOTE LESSTHANEQ))
  ((EQ X (QUOTE LESSTHAN)) (QUOTE GREATERTHAN))
  ((EQ X (QUOTE LESSTHANEQ)) (QUOTE GREATERTHANEQ))
  (T (QUOTE NOTSIMP**ERROR))))

(defun UPDATEVARS (VAR FIELDOFTUPLE VARPS)
  (COND ((NULL VARPS) (LIST (LIST VAR FIELDOFTUPLE)))
     ((EQ (CAAR VARPS) VAR)
      (COND ((MEMBER FIELDOFTUPLE (CDAR VARPS)) VARPS)
            (T
        (CONS (CONS VAR (CONS FIELDOFTUPLE (CDAR VARPS)))
           (CDR VARPS))) ))
     (T (CONS (CAR VARPS) (UPDATEVARS VAR FIELDOFTUPLE
                 (CDR VARPS))))) )

(defun TRANSLATEEQPREDS (EQPS VARTABLE)
    (COND ((NULL EQPS) NIL)
       (T (CONS (TRANEQPRED (CAR EQPS) VARTABLE)
             (TRANSLATEEQPREDS (CDR EQPS) VARTABLE)))) )

(defun BETWEENARGSNEEDADJUSTING (CONDITION)
  (AND
   (LISTP (CAR CONDITION))
   (LISTP (CADDR CONDITION))
   (EQ (CAADDR CONDITION) (QUOTE BETWEEN))
   (LOOKUPFCT (CAR CONDITION) TABLE FIELDINFO) ))

(defun ADJUSTBETWEENARGS (CONDITION) (PROG (TEMP)
  (COND ((NOT (BETWEENARGSNEEDADJUSTING CONDITION))
      (RETURN CONDITION)))
  (SETQ TEMP (MAPCAR2ARGS
           (CDADDR CONDITION)
           (QUOTE ADJUSTNUMBER)
           (LOOKUPFCT (CAR CONDITION) TABLE FIELDINFO)))
  (COND ((EQUAL (CAR TEMP) (CADR TEMP)) (SETQ TEMP
```

```
                        (CAR TEMP)))
     (T (SETQ TEMP (CONS (QUOTE BETWEEN) TEMP))))
  (RETURN (LIST (CAR CONDITION) (CADR CONDITION) TEMP)) ))

(defun ORDUNION (X Y) (PROG (TEMP)
  A (COND ((NULL Y) (RETURN X))
        ((NOT (MEMBER (CAR Y) X))
         (SETQ X (INSERTINORDER2 (CAR Y) X))))
    (SETQ Y (CDR Y))
    (GO A)))

(defun INSERTINORDER2 (ELEMENT L)
  (COND ((NULL L) (LIST ELEMENT))
     ((ATOM ELEMENT)
       (COND ((EQ (CADAR L) ELEMENT)
              (COND ((NULL (CDR L)) (LIST (CAR L) ELEMENT))
                 ((EQ (CADADR L) ELEMENT)
                 (CONS (CAR L) (INSERTINORDER2 ELEMENT
                               (CDR L))))
                 (T (CONS (CAR L) (CONS ELEMENT (CDR L))))))
          (T  L))); TST
     ((EQ (CADAR L) (CADR ELEMENT))
      (COND ((NULL (CDR L)) (LIST (CAR L) ELEMENT))
            ((EQ (CADADR L) (CADR ELEMENT))
          (CONS (CAR L) (INSERTINORDER2 ELEMENT (CDR L))))
            (T (CONS (CAR L) (CONS ELEMENT (CDR L))))))
     (T (CONS (CAR L) (INSERTINORDER2 ELEMENT (CDR L))))))

(defun EQPREDPROC (E S H) (PROG ()
    (COND ((AND (NULL E) (NULL S)) (RETURN NIL))
       ((OR (NULL E) (AND S (NOT (ATOMIN (CADAR S) E))))
              (SETQ E (CONS (LIST (QUOTE EQUAL)
                (CADAR S) (CADAR S)) E))))
    (SETQ E (INSERTINTO E S))
    (SETQ S NIL)
  A (COND ((AND            ; COMPARISON OF 2 SQL EXPRESSIONS
          (MEMBER (CAAR E) EQPREDS)
          (ATOM (CADAR E))
          (ATOM (CADDAR E))
          (CDR E)
          (ASSOC (CAADR E) SQLFCTPREDS)
          (CDDR E)
          (ASSOC (CAADDR E) SQLFCTPREDS))
        (SETQ S (CONS (LIST (CAAR E)
           (LFTOSQL1 (LIST (QUOTE SETX) (LIST (QUOTE QUOTE)
            (CADADR E)) (LIST (QUOTE QUOTE) (CADR E))) H)
           (LFTOSQL1 (LIST (QUOTE SETX) (LIST (QUOTE QUOTE)
            (CADR (CADDR E))) (LIST (QUOTE QUOTE)
                     (CADDR E))) H)) S))
        (SETQ E (CDDR E)) )
       ((AND        ; COMPARISON OF A CONST WITH A SQL EXPR
          (MEMBER (CAAR E) EQPREDS)
          (CDR E)
          (ASSOC (CAADR E) SQLFCTPREDS)
          (EQ (CADDAR E) (CADADR E)))
        (SETQ S (CONS (LIST (CAAR E) (CADAR E)
           (LFTOSQL1 (LIST (QUOTE SETX) (LIST (QUOTE QUOTE)
           (CADADR E)) (LIST (QUOTE QUOTE) (CADR E)))
           H)) S))
        (SETQ E (CDR E)))
       ((AND
          (MEMBER (CAAR E) EQPREDS)
          (CDR E)
          (ASSOC (CAADR E) SQLFCTPREDS)
          (EQ (CADAR E) (CADADR E)))
        (SETQ S (CONS (LIST (NOTSIMP** (CAAR E))
           (CADDAR E)
           (LFTOSQL1 (LIST (QUOTE SETX) (LIST (QUOTE QUOTE)
           (CADADR E)) (LIST (QUOTE QUOTE) (CADR E)))
           H)) S))
        (SETQ E (CDR E)))
       ((MEMBER (CAAR E) EQPREDS) (SETQ S (CONS (CAR E) S)))
```

```
      (T (RETURN (QUOTE EQPREDPROCERROR)) ) )
    (SETQ E (CDR E))
    (COND (E (GO A)))
    (RETURN S) ))

(setq EQPREDS '(EQUAL UNEQUAL GREATERTHAN GREATERTHANEQ LESSTHAN
   GREATER3ARGS LESSTHANEQ MEMBER NOTMEMBER PERCENT RATIO))

(setq EQPREDTABLE
  '((EQUAL $$.=.)
   (PERCENT $$.%.)
   (RATIO $$./.)
   (EQ $$.=.)
   (UNEQUAL $$.^=.)
   (NOTMEMBER $$.NOT IN.)
   (MEMBER IN)
   (GREATERTHAN $$.>.)
   (LESSTHAN $$.<.)
   (GREATERTHANEQ $$.>=.)
   (LESSTHANEQ $$.<=.)) )

(defun MAPCAR2ARGS (L FN EXTRAARG) (PROG (X Y Z)
    (SETQ X (SETQ Y (LIST NIL)))
  A (COND ((ATOM L) (RETURN (CDR X))))
    (RPLACD Y (SETQ Z (LIST (FuNcall fn (CAR L) EXTRAARG))))
    (SETQ Y Z)
    (SETQ L (CDR L))
    (GO A) ))

(defun ASSEMBLESQLLIST1 (V OP NEG POS TS)
  (PROG (FLAG)
  (COND ((AND (NULL NEG) (NULL POS))
      (RETURN (QUOTE ASSEMBLESQLLIST1_1ERROR)))
     (NEG (SETQ FLAG T) (SETQ POS NEG)))
  (COND ((AND (LISTP (CAAR POS)) (ASSOC
                       (CAAAAR POS) EQPREDTABLE))
      (RETURN (FINISHASSEMBLEEQPREDTYPE
            V OP (CAAAR POS) TS FLAG)))
     ((EQ (CAAR POS) (QUOTE WHQ))
      (RETURN (FINISHASSEMBLEWHQTYPE
        V OP (CAR POS) TS FLAG)))
     (T (RETURN (QUOTE ASSEMBLESQLLIST1_2ERROR)))) ))

(defun FINISHASSEMBLEEQPREDTYPE (V OP PREDFORM TS FLAG)
  (PROG (VLINE FILELETTER TEMP FILELINE)
  (COND ((NOT (EQ (CAADR PREDFORM) (QUOTE QUOTE)))
      (RETURN (QUOTE FINASSERROR_1))))
  (SETQ VLINE (ASSOC (CADAR V)
               (CADR (SETQ TEMP (CADDDR
                                   (CADDR PREDFORM))))))
  (COND ((NULL VLINE) (RETURN (QUOTE FINASSERROR_2))))
  (SETQ FILELETTER (CADADR VLINE))
  (SETQ FILELINE (ASSOC2 FILELETTER (CAR TEMP)))
  (SETQ FILELINE (LIST (CAR FILELINE) (CAR TVS)))
  (SETQ PREDFORM (SUBST
    (APPEND VLINE (LIST (LIST (CAADR VLINE) (CAR TVS))))
              VLINE
              PREDFORM))
  (SETQ VLINE (LIST (CAR VLINE) (LIST (CAADR VLINE)
                                (CAR TVS))))
  (SETQ TVS (CDR TVS))
  (COND (FLAG (SETQ TEMP (NOTSIMP* (CAR PREDFORM))))
     (T (SETQ TEMP (CAR PREDFORM))))
  (SETQ TEMP (CADR (ASSOC TEMP EQPREDTABLE)))
  (SETQ PREDFORM (CONS (CADR PREDFORM) (CONS TEMP (CDDR
    PREDFORM))))
  (RETURN (LIST
    TS
    (QUOTE |SELECT DISTINCT |)
    (LIST  (CADR VLINE) )
    (LIST
     (LIST FILELINE) (LIST VLINE) (LIST PREDFORM) NIL NIL))) ))
```

```
(defun FINISHASSEMBLEWHQTYPE (V OP WHQFORM TS FLAG)
  (PROG (VLINE FILELETTER TEMP FILELINE)
  (COND ((NOT (EQ (CAR WHQFORM) (QUOTE WHQ)))
      (RETURN (QUOTE FINASSERROR_3))))
  (SETQ VLINE (ASSOC
          (CADAR V)
          (CADR (SETQ TEMP (CADDDR WHQFORM)))))
  (COND ((NULL VLINE) (RETURN (QUOTE FINASSERROR_4))))
  (SETQ FILELETTER (CADADR VLINE))
  (SETQ FILELINE (ASSOC2 FILELETTER (CAR TEMP)))
  (SETQ FILELINE (LIST (CAR FILELINE) (CAR TVS)))
  (SETQ WHQFORM (LIST (SUBST
            (APPEND VLINE (LIST (LIST (CAADR VLINE)
                                (CAR TVS))))
            VLINE WHQFORM)))
  (SETQ VLINE (LIST (CAR VLINE) (LIST (CAADR VLINE)
                      (CAR TVS))))
  (SETQ TVS (CDR TVS))
  (COND (FLAG (SETQ TEMP (LIST WHQFORM NIL)))
     (T (SETQ TEMP (LIST NIL WHQFORM))))
  (RETURN (LIST TS (QUOTE |SELECT DISTINCT |)
     (LIST       (CADR VLINE) )
     (CONS (LIST FILELINE) (CONS (LIST VLINE) (CONS
     NIL TEMP))))) ))

(defun mygensym ()
 (prog ()
  (setq intstartno (+ intstartno 1))
  (return (intern (concatenate 'string "x"
               (format nil "~d" intstartno))))))

(defun mapconc (l fn) (mapcan fn l))

(defun VARSTOFIELDS (VARS TABL) (PROG (OUT TEMP)
  A (COND ((NULL VARS) (RETURN (REMDUPS (REVERSE OUT)))))
    (SETQ TEMP (ASSOC (CAR VARS) TABL))
    (COND (TEMP (SETQ OUT (CONS        (CADR TEMP ) OUT))))
    (SETQ VARS (CDR VARS))
    (GO A) ))

(defun ASSEMBLESQLLIST (V OP PART TS)
   (declare (special part))
  (PROG (SELSTRING OUTATTS ANS TEMP)
   (COND ((AND (EQ TS (QUOTE OPER))            ; 8/8/84 SJB
            (NULL (CDR V))  )
           (RETURN (LIST (QUOTE OPERATOR) OP (CAR V)
                         (CAAR PART))))
      ((AND (EQ TS (QUOTE OPER))          ; 8/8/84 SJB
            (CDR V) )
           (RETURN (LIST (QUOTE OPERATOR) OP V
                                  (CAAR PART))) )
      ((AND PART
            (NULL (CAAR PART))
            (NULL (CADAR PART))
            (NULL (CADDAR PART))
            (NULL (CADDDR (CAR PART)))
            (SETQ TEMP (CADDDR (CDAR PART)))
            (LISTP TEMP) (LISTP (CAR TEMP)) (LISTP
                          (CAAR TEMP))
            (LISTP (CAAAR TEMP))
            (MEMBER (CAAAAR TEMP) EQPREDS))
        (RETURN (CONS (QUOTE OPERATOR) (CAAAR TEMP))))
      ((AND
         (EQUAL (LENGTH V) 1)
         PART
         (CAR PART)
         (NULL (CAAR PART))
         (NULL (CADAR PART))
         (NULL (CADDAR PART)))
       (RETURN (ASSEMBLESQLLIST1
              V OP (CADDDR (CAR PART))
              (CADDDR (CDAR PART)) TS)))
```

```
      ((AND
         (EQUAL (LENGTH V) 1)
         PART
         (CAR PART)
         (NULL (CAAR PART))
         (NULL (ASSOC (CADAR V) (CADAR PART))) )
       (RETURN (INVIFCTR (CDDAR PART) TS))))
    (COND ((OR (EQ (CAAR V) (QUOTE BAGX)) OP)
        (SETQ SELSTRING (QUOTE |SELECT |)))
       (T (SETQ SELSTRING (QUOTE |SELECT DISTINCT |)))))
  A (COND ((EQ (CAAR V) (QUOTE BAGX))
       (COND ((EQ OP (QUOTE COUNT))
           (SETQ OUTATTS (QUOTE '*)))
          (OP
           (SETQ OUTATTS (CONS (QUOTE '*) (CONS OP
               (VARSTOFIELDS (CDAR V) (CADAR PART))))))
          (T (SETQ OUTATTS (VARSTOFIELDS (CDAR V)
                           (CADAR PART))))))
       ((EQ OP (QUOTE COUNT))
        (SETQ OUTATTS (CONS (QUOTE DISTINCT)
                       (VARSTOFIELDS (CDAR V)
                               (CADAR PART)))))
       (OP
        (SETQ OUTATTS (CONS (QUOTE |*|) (CONS OP
             (VARSTOFIELDS (CDAR V) (CADAR PART))))))
       (T (SETQ OUTATTS (ORDUNION
                  (VARSTOFIELDS (MAPCAR (QUOTE CADR) v)
                      (CADAR PART))
          (COND ((EQ DISCOURSE (QUOTE CHATTY))
          (REMDUPS
              (mapcar 'car
                   (appendargsfor (caddar part)))))
             ((EQ DISCOURSE (QUOTE LACONIC)) NIL)
              (T (REMDUPS
               (MAPCONC (APPENDARGSOFOR (CADDAR PART))
                (FUNCTION (lambda (Y) (PROG (TEMP)
                   (COND ((AND (EQ (CADR Y) (QUOTE '=))
                               (OR (EQ (CADDR Y) (QUOTE NOTNULL))
                                (EQ (CADDR Y) (QUOTE NULL))))
                         (RETURN (CONS (CAR Y) NIL)))
                        ((AND (EQ (CADR Y) (QUOTE '=))
                              (ATOM (CADDR Y))
                         (OR (NULL (SETQ TEMP (LOOKUPFCT
                           (CAR Y) (CAAR PART) FIELDINFO)))
                         (CDDR TEMP)
                         (NOT (OR
                            (EQ (CADR TEMP) (QUOTE IN))
                            (EQ (CADR TEMP) (QUOTE FIN))
                          (EQ (CADR TEMP) (QUOTE INANDFIN))))))
                (RETURN NIL))
                 (T (RETURN (CONS (CAR Y)  NIL)))))))))))
          ))) )
    (SETQ ANS (CONS (LIST TS SELSTRING
           (REMOVEEQATTS OUTATTS (CADAR PART))
            (CHANGEEXISTSTOCONDS (CAR PART))) ANS))
    (SETQ PART (CDR PART))
    (COND (PART (GO A))
       ((NULL (CDR ANS)) (RETURN (CAR ANS))))
    (RETURN (MAKESELECTATTSPARALLEL
           (CONS (QUOTE UNION) (REVERSE ANS)))) ))

(defun REMOVEEQATTS (OUTATTS EQS) (PROG (TEMP)
   (SETQ EQS (MAPCAR 'CDR EQS))
 A (COND ((NULL EQS) (RETURN OUTATTS)))
 B (SETQ TEMP (INTERSECTION OUTATTS         (CAR EQS) ))
   (COND ((> (LENGTH TEMP) 1)
       (SETQ OUTATTS (DELETE (CAR TEMP) OUTATTS))
       (GO B)))
   (SETQ EQS (CDR EQS))
   (GO A) ))

(defun INVIFCTR (FORM TS)
```

```
   (COND ((AND
        (CAR FORM) (CAAR FORM) (CAAAR FORM)
        (ASSOC (CAAAR FORM) EQPREDTABLE) (EQ TS
                                (QUOTE CTR)))
       (CONS (CONS (CONS (NOTSIMP* (CAAAR FORM))
                                (CDAAR FORM))
                (CADR FORM)) (CDR FORM)))
      (T FORM)))

(setq discourse 'normal)

(defun CHANGEEXISTSTOCONDS (FORM)
  (PROG (FIRST SECOND THIRD FOURTH FIFTH TEMP4 TEMP5 TEMP)
    (SETQ FIRST (CAR FORM))
    (SETQ SECOND (CADR FORM))
    (SETQ THIRD (CADDR FORM))
    (SETQ FOURTH (CADDDR FORM))
    (SETQ FIFTH (CAR (CDDDDR FORM)))
  A (COND (FOURTH (GO B)) (FIFTH (GO C)))
    (RETURN (LIST FIRST SECOND (REVIFNOTORLIST THIRD)
                                TEMP4 TEMP5))
  B (COND ((EQ (CAAR FOURTH) (QUOTE WHQ))
        (SETQ TEMP4 (CONS (CAR FOURTH) TEMP4)))
       ((AND
         (LISTP (CAAR FOURTH))
         (LISTP (CAAAR FOURTH))
         (EQ (CAADDR (SETQ TEMP (CAAAR FOURTH)))
                        (QUOTE WHQ)))
        (SETQ THIRD (CONSTOORLIST
           (LIST (CADR TEMP) (CADR (ASSOC (NOTSIMP*
                                (CAR TEMP))
           EQPREDTABLE)) (CADDR TEMP)) THIRD)))
       (T (ERROR "CHANGEEXISTSERROR")))
    (SETQ FOURTH (CDR FOURTH))
    (GO A)
  C (COND ((EQ (CAAR FIFTH) (QUOTE WHQ))
        (SETQ TEMP5 (CONS (CAR FIFTH) TEMP5)))
       ((AND
         (LISTP (CAAR FIFTH))
         (LISTP (CAAAR FIFTH))
         (EQ (CAADDR (SETQ TEMP (CAAAR FIFTH)))
                        (QUOTE WHQ)))
        (SETQ THIRD (CONSTOORLIST (LIST (CADR TEMP)
              (CADR (ASSOC
            (CAR TEMP) EQPREDTABLE)) (CADDR TEMP)) THIRD)))
       (T (ERROR "CHANGEEXISTSERROR")))
    (SETQ FIFTH (CDR FIFTH))
    (GO A) ))

 (defun CONSTOORLIST (X Y)
  (COND ((AND Y (EQ (CAR Y) (QUOTE OR)))
      (CONS (QUOTE OR)
            (MAPCAR2ARGS
           (CDR Y)
           (QUOTE (lambda (U V) (CONS V U)))
           X)))
      (T (CONS X Y))))

 (defun REVIFNOTORLIST (L)
  (COND ((NULL L) NIL)
     ((EQ (CAR L) (QUOTE OR)) L)
     (T (REVERSE L))))

 (defun APPENDARGSOFOR (X)
  (COND ((ATOM X) X)
     ((EQ (CAR X) (QUOTE OR)) (MAPCONC (CDR X)
                           (FUNCTION (lambda (Y) Y))))
     (T X)))
```

PRE-SQL TO SQL TRANSLATION PROCEDURES

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;; The following procedures finish printing in SQL, the
;;;;; Pre-SQL form generated by LFTOSQL1.  In the file
;;;;; "printsql1.lsp"
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun PRINTSQL1 (SQLLIST INITSEL EMBEDSEL)
   (PROG (TOPSTRING SELSTRING OUTATTS PARTRAN ANDFLAG)
      (terpri)
      (setq extraprintcolumns nil)
      (SETQ TOPSTRING (CAR SQLLIST))
      (SETQ SELSTRING (CADR SQLLIST))
      (SETQ OUTATTS (CADDR SQLLIST))
      (SETQ PARTRAN (CADDDR SQLLIST))
      (SETQ OUTATTS
      (REMOVEEQUIVS
      (ADDEXTRAOUTATTS OUTATTS (CAR PARTRAN))
      (MAPCAR 'CDR (CADR (CADDDR SQLLIST)))))
      (SETQ SQLLIST NIL)
      (COND (INITSEL (PRINT TOPSTRING)))
      (COND (EMBEDSEL (PRINc "(")))
      (COND ((AND EMBEDSEL (NOT (EQ (CAR OUTATTS) "*")))
             (SETQ OUTATTS
                (COND ((EQ (CAR OUTATTS) 'UNIQUE)
                       (LIST 'UNIQUE (CADR OUTATTS)))
                      (T (LIST (CAR OUTATTS)))))))
      (COND ((AND (EQ (CAR OUTATTS) 'UNIQUE)
                  (EQUAL PARTNER "TQASQL"))
             (SETQ OUTATTS (CONS 'DISTINCT (CDR OUTATTS)))))
      (COND ((AND (EQ SELSTRING "SELECT UNIQUE .")
                  (EQUAL PARTNER "TQASQL"))
             (SETQ SELSTRING "SELECT DISTINCT .")))
      (PRINc SELSTRING)
      (PRINCOMMALIST (MAPCAR 'PACK3 OUTATTS))
      (TERPRI)
      (PRINc "FROM ")
      (PRINFROMLIST (CAR PARTRAN))
      (TERPRI)
      (SETQ ANDFLAG (PRINCONDITIONS PARTRAN))
      (PRINNOTEXISTS (CADDDR PARTRAN))
      (COND (EMBEDSEL (PRINc ")")))
      (RETURN T) ))

(defun REMOVEEQUIVS (ATTS EQS)
   (PROG ()
A     (COND ((NULL EQS) (RETURN ATTS))
        ((< (LENGTH (CAR EQS)) 2)
         (SETQ EQS (CDR EQS))
         (GO A))
        ((AND (MEMBER (CAAR EQS) ATTS) (MEMBER (CADAR EQS)
                                ATTS))
         (SETQ ATTS (DELETE (CADAR EQS) ATTS))))
      (COND ((EQUAL (LENGTH (CAR EQS)) 2)
         (SETQ EQS (CDR EQS))
         (GO A)))
      (SETQ EQS
      (CONS (CONS (CAAR EQS) (CDDAR EQS))
         (CONS (CDAR EQS) (CDR EQS))))
      (GO A) ))

(defun ADDEXTRAOUTATTS (OUTATTS TUPLEVARS)
   (PROG (NEWOUTATTS EXTRAS TEMP)
      (COND ((EQ (CAR OUTATTS) '*) (RETURN OUTATTS))
      ((EQ (CAR OUTATTS) (QUOTE UNIQUE))
       (SETQ OUTATTS (CDR OUTATTS))
       (SETQ TEMP (QUOTE (UNIQUE)))))
      (SETQ TUPLEVARS (MAPCAR 'reverse TUPLEVARS))
A     (COND ((NULL OUTATTS)(RETURN
       (REMDUPS (APPEND TEMP (REVERSE NEWOUTATTS))))))
```

```
      (SETQ EXTRAS
      (NEWASSOC
      (CONS (CAAR OUTATTS)
         (CDR (ASSOC (CADAR OUTATTS) TUPLEVARS)))
      EXTRAPRINTCOLUMNS))
      (COND ((NULL EXTRAS)
         (SETQ NEWOUTATTS (CONS (CAR OUTATTS) NEWOUTATTS))
         (GO C)))
B     (SETQ NEWOUTATTS (CONS (CONS (CAR EXTRAS)
                     (CDAR OUTATTS)) NEWOUTATTS))
      (SETQ EXTRAS (CDR EXTRAS))
      (COND (EXTRAS (GO B)))
C     (SETQ OUTATTS (CDR OUTATTS))
      (GO A) ))

 (defun PRINCOMMALIST (L) (PROG ()
    (COND ((NULL L) (RETURN NIL))
      ((EQ L '*)
       (RETURN (PRINC "COUNT(*)")))
      ((EQ (CAR L) (QUOTE UNIQUE))
       (PRINC "COUNT(UNIQUE ")
       (PRINC (CADR L))
       (RETURN (PRINC ")")))
      ((EQ (CAR L) (QUOTE DISTINCT))
       (PRINT1 "COUNT(DISTINCT ")
       (PRINC (CADR L))
       (RETURN (PRINC ")")))
      ((EQ (CAR L) '*)
       (PRINc (CADR L))
       (PRINc "(")
       (PRINC (CADDR L))
       (RETURN (PRINC ")"))))
  A (PRINC (CAR L))
    (SETQ L (CDR L))
    (COND ((NULL L) (RETURN NIL)))
    (PRINC ", ")
    (GO A) ))

(defun prinfromlist (l)
      (do ((currlist l (cdr currlist)))
      ((null currlist) nil)
      (do ((currvars (car currlist) (cdr currvars)))
         ((null currvars) nil)
         (princ (car currvars))
         (cond ((cdr currvars) (princ " "))
            ((cdr currlist) (princ ", "))) )))

(defun PACK3 (L)
   (COND ((ATOM L) L)
      ((EQ (CAR L) (QUOTE QUOTE)) (CADR L))
      (T (STRING-APPEND (CADR L) "." (CAR L))) ))

(defun PRINCONDITIONS (PART)
   (PROG (FLAG R L C)
      (SETQ R (MAPCAR 'cadr (CAR PART)))
      (SETQ L (MAPCAR 'ordernotslast (CADR PART) ))
      (SETQ C (CADDR PART))
A     (COND ((NULL L) (GO B))
      ((NULL (CDDAR L))
       (SETQ L (CDR L))
       (GO A))
      ((OR
         (MEMBER (CADR (CADAR L)) R)
         (MEMBER (CADR (CADDAR L)) R))
       (COND (FLAG (PRINc "AND "))
          (T (PRINc "WHERE ")))
       (PRINJOIN (CADAR L) (CADDAR L))
       (TERPRI)
       (SETQ FLAG T) ))
      (SETQ L (CONS (CONS (CAAR L) (LOPOFF1 (CDAR L)))
                    (CDR L)))
      (GO A)
```

```
B     (COND ((NULL C) (RETURN FLAG))
      (FLAG (PRINc "AND "))
      (T (PRINc "WHERE ")))
      (COND ((EQ (CAR C) (QUOTE OR))
       (COND (FLAG (PRINc "(")
          (SETQ FLAG (QUOTE OR)))
          (T T))
       (SETQ L (CADR C))
       (SETQ C (CDDR C)))
      (T (SETQ L C)
         (SETQ C NIL)))
D     (PRINCONDITION (CAR L))
      (COND ((NULL FLAG) (SETQ FLAG T)))
      (SETQ L (CDR L))
      (COND (L (TERPRI)
         (PRINc "AND ")
         (GO D)))
      (COND (C (SETQ L (CAR C))
         (SETQ C (CDR C))
         (TERPRI)
         (PRINc "OR ")
         (GO D)))
      (COND ((EQ FLAG (QUOTE OR)) (PRINc ")")))
      (RETURN FLAG) ))

(defun PRINCONDITION (L)
  (declare (SPECIAL CASE SUPCHARS TABLE FIELDINFO))
  (PROG (W X Y Z)
     (COND ((NEEDSEXPANSION L) (GO B)))
     (SETQ X (CAR L))
     (SETQ Y (CADR L))
     (SETQ Z (CADDR L))
A    (PRINquote X)
     (PRINc " ")
     (COND ((AND (LISTP Z) (EQ (CAR Z) 'BETWEEN))
      (SETQ Z (CDR Z))))
         (PRINC Y)
         (PRINC " ")
         (PRINquote Z)
     (RETURN NIL)
B    (SETQ L (EXPANDLISTTOLIKES L))
     (PRINC "(")
C    (PRINCONDITION (CAR L))
     (SETQ L (CDR L))
     (COND (L (TERPRI)
         (PRINC "OR ")
         (GO C)))
     (PRINC ")")
     (RETURN NIL) ))

(defun PRINNOTEXISTS (SELECTS)
   (DECLARE (SPECIAL ANDFLAG))
   (PROG ()
A     (COND ((NULL SELECTS) (RETURN NIL)))
      (COND (ANDFLAG (TERPRI)))
      (COND (ANDFLAG (PRINC "AND NOT EXISTS") (TERPRI))
      (T (PRINC "WHERE NOT EXISTS") (TERPRI)
       (SETQ ANDFLAG T)))
      (PRINTSQL1 (CAR SELECTS) NIL T)
      (SETQ SELECTS (CDR SELECTS))
      (GO A) ))

(defun PRINQUOTE (L)
   (COND ((AND (LISTP L) (EQ (CAR L) (QUOTE WHQ)))
       (PRINTSQL1 L NIL T))
      ((AND (LISTP L) (EQ (CAR L) (QUOTE MARKER**)))
       (PRINQUOTECOMMALIST (CDR L)))
      ((LISTP L) (PRINc (pack3 L)))
      ((numberp l) (princ l))
      (T
       (PRINc "'")
       (PRINc L)
```

```
       (PRINc "'"))))

(defun PRINQUOTECOMMALIST (L)
   (PROG ()
      (PRINc "(")
      (COND ((NULL L) (RETURN NIL)))
A     (PRINQUOTE (CAR L))
      (SETQ L (CDR L))
      (COND ((NULL L)
          (PRINc ")")
          (RETURN NIL)))
      (PRINc ",")
      (GO A) ))

(defun newassoc (x l)
   (cond ((null l) nil)
      ((equal x (caar l)) (cadar l))
      ((and (listp x)(equal (reverse x)(caar l)))
                     (massage (cadar l)))
      (t (newassoc x (cdr l))) ))

(defun PRINJOIN (X Y)
   (PROG (TEMP)
      (SETQ TEMP 0)
      (COND ((EQ (CAR X) "$$.^=.")
       (SETQ X (CDR X))
       (SETQ TEMP 1)))
      (COND ((EQ (CAR Y) "$$.^=.")
       (SETQ Y (CDR Y))
       (SETQ TEMP (ADD1 TEMP))))
      (COND ((EQUAL TEMP 0)
       (PRINCONDITION (LIST X '= Y)))
      ((EQUAL TEMP 1)
       (PRINCONDITION (LIST X "$$.^=." Y))))
      (RETURN NIL) ))

(defun lopoff1 (form)
   (cond ((eq (caadr form) "$$.^=.")
       (cons (car form) (cddr form)))
      (t (cdr form))))

(defun NEEDSEXPANSION (CONDITION)
   (AND
      (LISTP (CAR CONDITION))
      (LISTP (CADDR CONDITION))
      (NOT (EQ (CAADDR CONDITION) 'BETWEEN))
      (OR
      (EQ (CADR CONDITION) (QUOTE IN)))
      (GET (CAAR CONDITION) (QUOTE FIELDINFO)) ))

(defun EXPANDLISTTOLIKES (CONDITION)
   (PROG (X Y Z)
      (SETQ X (CAR CONDITION))
      (SETQ Y (CADR CONDITION))
      (COND ((EQ Y (QUOTE IN)) (SETQ Y (QUOTE '=))))
      (SETQ CONDITION (CADDR CONDITION))
      (COND ((EQ (CAR CONDITION) (QUOTE MARKER**))
       (SETQ CONDITION (CDR CONDITION))))
A     (COND ((NULL CONDITION) (RETURN (REVERSE Z))))
      (SETQ Z (CONS (LIST X Y (CAR CONDITION)) Z))
      (SETQ CONDITION (CDR CONDITION))
      (GO A) ))

(defun ordernotslast (l)
   (prog (x y z)
      (setq x (car l))
a     (setq l (cdr l))
      (cond ((null l) (return (cons x (append y z))))
      ((eq (caar l) "^=")
       (setq z (cons (car l) z)))
      (t (setq y (cons (car l) y))))
      (go a) ))
```