Robs Parser

1) ROB - file that loads whole system
   Various 'FLS' file lists etc.
   Some of Rob's notes.

2) Dictionaries

3) Semantic check routines
   Agreement routines
   Semantic rules
   various other semantic facts

4) Packets (ie Syntax rules)

5) TOP level of system
   INIT
   BITIN1 initialisations

6) Support Code
   (In the order they are listed in the file ROB (section 1))).

7) Various additional material used by Chris.
   (needs incorporating)

```
SUBFILE: ROB, @20:56 10-APR-1981 <005> (515)
/* ROB : Consulting this file will load the complete parser
         from scratch. Type "ok." to set up a core image
         which you can then save.
       to save type: save rob.exe[400,444]

       use:    run mec:prolog for the latest version of Prolog

                                                    (Lawrence)
                                                    Updated: 31 March 81
*/

% FIXES
%
%  (31 March 81)
%
%       Updated filenames of the utilities to include util:
        Added mecdic to the list of dictionaries loaded.
-


:-          [
                opsr                    % Operator definitions
            ],

    compile([
                                        %% Utilities

                'utilr.pl',             % Selection of list utilities
                'util:flagro.pl',       % Handling flags
                'util:files.pl',        % Opening files with checks
                'util:test.pl',         % Testing compiled code

                                        %% Parser support routines

                'readin.rob',           % Sentence -> list of atoms
                'lookup.lpl',           % Dictionary lookup
                'morpho.lpl',           % Morphology
                'has.lpl',              % Check Node has Feature
                'packs.lpl',            % Operations on packets
                'node.lpl',             % Operations on nodes
                'featur.lpl',           % Operations on features
                'hacks.lpl',            % Funny feature operations
                'sem.lpl',              % Applying semantic rules
                'db.lpl',               % Managing the semantic database
                'enter.lpl',            % Tracing of (syntactic) rules
                'ptree.lpl',            % Pretty print parse tree
                'portr.lpl',            % Useful portray routines
                'load.lpl',             % Load dictionaries/rules etc
                'dbrep.lpl',            % Managing the (dict/rule) database
                'rulem.lpl',            % Main control of parser
                'switch.lpl',           % Switch table for (syntactic) rules

                                        %% Semantics

                'semsup.spl',           % Various support routines

                                        %% Packets of syntactic rules
```

```
            'sstart.pk',
            'cpool.pk',
            'npool.pk',
            'pdet.pk',
            'pap1.pk',
            'pap2.pk',
            'padj.pk',
            'pnoun.pk',
            'npcom.pk',
            'parpp.pk',
            'psubj.pk',
            'bldaux.pk',
            'paux.pk',
            'pvp.pk',
            'passiv.pk',
            'ssvp.pk',
            'obj.pk',
            'nosubj.pk',
            'thatc.pk',
            'infc.pk',
            'tlicom.pk',
            'tblcom.pk',
            'twobj.pk',
            'embsfi.pk',
            'bname.pk',
            'pconj.pk',
            'ssfin.pk'
        ]),

        [                               %% Utilities
            'util:edit.pl',             % Edit file (with FINE)
            'util:applic.pl',           % Application routines

            'top.lpl',                  % User interface
            'tflag.lpl',                % Trace flags
             init,                      % Various initialisations
             bitini,                    % Preassigning certain feature bits

             semchk,                    % Semantic checks
             agree                      % Syntactic agreement checks
        ],

load_dict([                             %% Dictionaries

            feats,                      % Specify features
            mordic,
            ndict,
            mdict,
            pdict,
            vdict,
            mecdic                      % Additional Mecho dictionary
        ]),

load_sem(   semrul      ).              % Semantic rules
```

\\\\\

SUBFILE: ROB.SUB @20:6 22-MAY-1981 <005> (122)
rob.sub
fls
flsdic
fl
pks
init
bitini
utilr.pl
top.lpl
tflag.lpl
rob
lookup.lpl
morpho.lpl
has.lpl
packs.lpl
node.lpl
featur.lpl
hacks.lpl
sem.lpl
 h.lpl
 nter.lpl
ptree.lpl
portr.lpl
load.lpl
dbrep.lpl
rulem.lpl
switch.lpl
semsup.spl
sstart.pk
cpool.pk
npool.pk
pdet.pk
pqp1.pk
pqp2.pk
padj.pk
pnoun.pk
npcom.pk
parpp.pk
 subj.pk
 daux.pk
paux.pk
pvp.pk
passiv.pk
ssvp.pk
obj.pk
nosubj.pk
thatc.pk
infc.pk
tlicom.pk
tblcom.pk
twobj.pk
embsfi.pk
bname.pk
pconj.pk
ssfin.pk
semchk
agree
feats
mordic

ndict
mdict
pdict
vdict
semrul
embsvp.pk

\\\\\

SUBFILE: FL. @20:56 23-APR-1981 <005> (102)
opsr
init
utilr.pl
top.lpl
tflag.lpl
lookup.lpl
morpho.lpl
has.lpl
packs.lpl
node.lpl
featur.lpl
hacks.lpl
sem.lpl
db.lpl
enter.lpl
ptree.lpl
portr.lpl
 ?ad.lpl
 _rep.lpl
rulem.lpl
switch.lpl
semsup.spl
semchk
agree
sstart.pk
cpool.pk
npool.pk
pdet.pk
pqp1.pk
pqp2.pk
padj.pk
pnoun.pk
npcom.pk
parpp.pk
psubj.pk
bldaux.pk
 ?ux.pk
 /p.pk
passiv.pk
ssvp.pk
obj.pk
nosubj.pk
thatc.pk
infc.pk
tlicom.pk
tblcom.pk
twobj.pk
embsfi.pk
bname.pk
pconj.pk
ssfin.pk

\\\\\

```
SUBFILE: FLS. @2:39 22-NOV-1980 <005> (74)
/noheader rob,-
top.lpl,-
tflas.lpl,-
utilr.pl,-
init,-
bitini,-
lookup.lpl,-
morpho.lpl,-
has.lpl,-
packs.lpl,-
node.lpl,-
featur.lpl,-
hacks.lpl,-
sem.lpl,-
db.lpl,-
enter.lpl,-
ptree.lpl,-
 ?rtr.lpl,-
_oad.lpl,-
dbrep.lpl,-
rulem.lpl,-
switch.lpl,-
semrul,-
agree,-
semchk,-
semsup.spl,-
feats,-
mordic,-
ndict,-
mdict,-
pdict,-
vdict
```

\\\\

```
SUBFILE: FLSDIC. @1:45 7-OCT-1980 <005> (13)
/noheader feats,-
mordic,-
ndict,-
mdict,-
pdict,-
vdict



\\\\\
```

```
SUBFILE: PKS. @20:57 23-APR-1981 <005> (66)
/noheader sstart.pk,-
cpool.pk,-
npool.pk,-
pdet.pk,-
pqp1.pk,-
pqp2.pk,-
padj.pk,-
pnoun.pk,-
npcom.pk,-
parpp.pk,-
psubj.pk,-
bldaux.pk,-
paux.pk,-
pvp.pk,-
passiv.pk,-
ssvp.pk,-
obj.pk,-
  \subj.pk,-
 .atc.pk,-
infc.pk,-
tlicom.pk,-
tblcom.pk,-
twobj.pk,-
embsfi.pk,-
bname.pk,-
pconj.pk,-
ssfin.pk



\\\\\
```

BUGS

Rob Milne
This file is all the bugs, errors and improvements for the parser.

THINGS TO DO

batch tester
AP stuff fixed
That stuff
ungrammatical more
as large as, and less than NP
tighter catagories
quantifier grammar
Have-Imperative and YNQ
New analysis of Imperative and Y_N_Q
Kissing aunts
He looked up the street
psuedo attachment - keeping ambiguity
proper adverbs
 uns semantic check
.erb case stuff verb typing
PP definition stuff
has feat inference and can have
reduced relative stuff - running diagnostic
Gazdar types and Verb typing made neccessary
Steedman's Wh method
comments added for semantics finalizing
nsstart and "block"
add description of proper semantics
Experiment to test delayed resolution prediction
write help and documentation

Implement:
        Bosureux semantics
        PP psudeo attachment
        Look up
        Fix number

 'AMMAR TO ADD
        when S S
        If S,S

BUGS

you so hit mary.
the same of so.
rob hit sue and pam left
the boy and the girl' mother.

1. Tom hit mary and Sue left, need one more lookahead to decide properly,
        move to stack or cheat ahead somehow.

2. TOM_MARY need looking at

```
        TYPE

        SS_START

5 -[binder]                                  if_what
10-[wh &(np#pp#ap)]                  wh_quest
10-[np][verb]                        major dcl s
10-[adverb][nsstart]                 adverb
10-[auxverb][nsstart]                aux_invert
10-[np][pp]                          np pp default
10-[np][fpunc]                       np utterance
10-[pp][fpunc]                       pp utterance
10-[tnsless]                         imperative
10-[pp]                              fronted_pp
15-[wh]                              wh_np ss-start

        CPOOL

5 -[X][conj][X]                      X and X          3
5- [poss_np],   agree_det            poss_det
 \-[compadv][that]                   so that
 J-[name, not(np)]                   propname
10-[propnoun]                        propnoun
10-[prep][nsstart]                   pp
10-[det]        agree_det            marked startnp
10-[nsstart&( not pronoun,det)]
10-[than_comp][np]                   than_np
10-[conj]                            and
10-[comp_s]                          comp to np
10-[pp]                              NP_PP
10-[pronoun]                         Pronoun
10-[vp]                              VP attach
15-[][possesive]                     poss_np

        NPOOL

10-[qp][conj][quant]                 qp and quant              3
10-[than][name]                      longer than
10-[qp][units]                       3 ft/sec
 \-[qp][adj]                         ft_long
 J-[quant]                           Noun_qp
10-[ap]                              ap_attach
10-[qp]                              qp attach

        PARSE_DET

10-[det]                             Determiner

        PARSE_QP_1

10-[how][adj]                        how_adj
10-[quant][adj#num]                  quant
10-[ord][noun,time]                  next week
10-[all][det&def]                    all_the
10-[quantifier]          quantifier
15-[]                                quant done

        PARSE_QP_2

10-[quant][adj#noun]                 det_quant
```

```
10-[ord]                              ordinal
15-[]                                 det_quant done

        PARSE_ADJ

10-[adj][adj#noun#dim]                Adj_group
10-[adj]                              adj np
15-[]                                 adjective

        PARSE_NOUN

10-[noun][variable]                   train_t
10-[noun][noun], agree_complex        complex noun
10-[noun,npl]                         nouns
10-[noun]                             noun agree
15-[]                                 np_built

        NP_COMPLETE

10-[qp][prep]                         qp_pp
 \-[prep][nsstart]                    prep_start
 J-[verb,ins]                         reduced relative
10-[verb,ed], agree_red_rel            "      "
10-[relative]                         rel attach
10-[relpron]                          rel_pron_np
10-[relpron_np]                       wh_relative clause
10-[pp]                               np_pp
10-[conj]                             And
10-[comma]                            Comma
15-[det]                              instert wh
15-[of][noun]                         of_pp
15-[]                                 np_done

        PARSE_PP

10-[prep]                             attach_prep
10-[np]                               pp_np
10-[wh]                               with which

        PARSE_SUBJ

10-[np][verb], agree_subj             unmarked order
10-[auxverb][np#nsstart]              Aux_inversion

        BUILD_AUX

10-[modal][tnsless]
10-[have][en]
10-[be][en]
10-[be][ins]
10-[do][tnsless]
10-[be][prep#adj]
10-[nes]
10-[adverb]
15-[]                                 aux_complete

        PARSE_AUX

10-[to][tnsless]                      To_inf
10-[verb]                             Start_aux
```

```
10-[aux]                          Aux_attach

        PARSE_VP

10-[pp*ap]                        Predp
10-[verb]                         Main_verb

        PASSIVE

5 -[]                             passive

        SS_VP

5 -[particle],  agree_particle part
10-[adverb][adverb]               adverb group
10-[adverb]                       adverb
10-[pp]                           PP_under_VP_1
15-[particle (prep)]
15-[]                             VP_done

        OBJECT

10-[np]          object

        NO_SUBJ

10-[to][tnsless]                  seems

        THAT_COMP

10-[that][nsstart]                that_s
5 -[np][verb]                     that s start

        INF_COMP

5 -[np][to][tnsless]              inf_s_start              3

        TO_LESS_INF_COMP

  `-[np][tnsless]

        TO_BE_LESS_INF_COMP

10-[np][en or adj]                insert to be
10-[en or adj]                    insert to be, 2

        SUBJ_LESS_INF_COMP

10-[to][tnsless]                  Create Delta subject

        EMBEDDED_S_FINAL

10-[pp]                           ppunder_s
15-[]                             s-done

        BUILD_NAME

10-[name]                         Name
15-[]                             end of name
```

```
        PARSE_CONJ

5 -[VP]                                 drop and
15-[]                                    **  **

        SS_FINAL

10-[PP]
10-[fpunc]                              S_done
10-[sent_subj]                          init_s_har
10-[comma][conj#binder] conjoined S
10-[comma]                              hypo_s


NP: 3
        [X][and][X]
        [np][to][tnsless]
        [qp][and][quant]
        110 rules
```

CASE

ROB MILNE

```
list of case frames and particles for all the verbs
this is only schetchy for now
SYMBOLS:        {oblisatory}, [necassary constraints], (preps
SENTENCE PREPS: time, manner, location
V3s - then intrans, else trans for (NP)
() = intrans                    all set the packet ss_vp


VERB: walk      V (NP) (PP)
SLOTS: {asent}  object  instrument      trajectory
NP/PP: np,pp            PARTICLES: on

VERB: run       V (NP) (PP)     object,intrans
SLOTS: {asent}  object2 trajectory
NP/PP: np,pp            PARTICLES: down, on,away

 RB: default    V (NP) (PP)     object,intrans
 OTS: {asent}   object  instrument      location        item
NP/PP: np,pp            PARTICLES:

VERB: sit,sat   V NP PP         V PP    object,pp_obj
SLOTS: {asent}  who     location(on,by
NP/PP: np,pp            PARTICLES:

VERB: go        V ()    gone-past       V toPP  V VP    V (toPP) VPbar
SLOTS: {asent}  object2 source  destination
NP/PP: pp               PARTICLES: on,in,ahead

VERB: went      V ()    V toPP  V(toPP) VPbar,intrans
SLOTS: {asent} destination(to   trajectory(around       instrument(in
NP/PP:pp                PARTICLES:

VERB: break     V (NP)  object,intrans
SLOTS: {asent}  {object}        instrument
NP/PP: pp               PARTICLES: on,ahead,by,in,down

 RB: broke      V (NP)  object,intrans
SLOTS: {asent} {object} instrument
NP/PP: np               PARTICLES

VERB: rape      V NP PP object
SLOTS: {asent}[human]   {object}[human] location
NP/PP: np               PARTICLES:

VERB: kiss      V NP    object
SLOTS: {asent}  {object}
NP/PP: np               PARTICLES:

VERB: shoot     V (NP)  object,intrans
SLOTS: {asent}  {object}        instrument      target
NP/PP: np               PARTICLES: up,out

VERB: shot      V (NP)  object,intrans
SLOTS: {asent} {object} instrument      location
NP/PP: np               PARTICLES:
```

```
VERB: leave      V NP NP,          V PP     V (NP)  two_obj,object,intrans,pp_obj
SLOTS: {agent}  location(from,np          instrument(in   object2
NP/PP: np,pp              PARTICLES: out,behind

VERB: left       V NP NP V PP     V (NP)  two_obj,object,intrans,pp_obj
SLOTS: {agent} source[item,loaction]
NP/PP:           PARTICLES:

VERB: attach     V NP toPP        object
SLOTS: {agent}  {object}          object(to,onto  instrument
NP/PP: np                 PARTICLES:

VERB: hit        V (NP)  V NP NP two_obj,object,intrans
SLOTS: {agent}  2-object {object}         instrument      target(on,in
NP/PP: np                 PARTICLES: on

VERB: connect    V NP toPP        object
SLOTS: {agent}  {object}          {object2}(to,and        instrument
NP/PP: np                 PARTICLES:

  RB: reach      V (NP)  V NP forPP       object,intrans
SLOTS: {agent}  {object}          location(on,in  instrument
NP/PP: np,pp              PARTICLES: out,in,around

VERB: show       V NP NP V NP     V NP toPP        V Sbar  two_obj,object
SLOTS: {agent}[person]  {object}[anim]  object3
NP/PP: np                 PARTICLES: up

VERB: determine           V NP     object
SLOTS: {agent}  NP        instrument
NP/PP: np                 PARTICLES:

VERB: find       V NP     V NP forPP       object
SLOTS: {agent}  NP or S instrument
NP/PP: np,pp              PARTICLES: out

VERB: locate     V NP     V NP NP V NP forPP       two_obj,object
SLOTS: {agent}  {object}          instrument      by-time location
NP/PP: np                 PARTICLES:

vcRB: support    V NP (PP)        object
SLOTS: {object} {object}          instrument
NP/PP: np                 PARTICLES:

VERB: suspend    V NP (fromPP)    object
SLOTS: {object} {object}          location(from   object2 instrument
NP/PP: np                 PARTICLES:

VERB: move       V NP     V (toPP) VPbar  object
SLOTS: {agent}  {object}          period  path    start   finish
NP/PP: np                 PARTICLES: on,out,over,in,about,up

VERB: place      V NP PP object
SLOTS: {agent}  {object}          {location}      instrument
NP/PP: np                 PARTICLES:

VERB: pass       V NP     V NP NP V NP toPP        two_obj,object
BOG:intrans
SLOTS: {agent}  {object}          destination(to  instrument
NP/PP: np                 PARTICLES: out
```

```
VERB: hang      V NP (PP)       V ()    object,intrans
SLOTS: {agent}  {object}        location(on,from        instrument
NP/PP: np               PARTICLES: on,out

VERB: drop      V NP    V NP NP two_obj,object
SLOTS: {agent}  {object}        source
NP/PP: np               PARTICLES: out,in,by,over

VERB: project   V NP PP object
SLOTS: {agent}  {object}        destination     instrument      trajectory
NP/PP: np               PARTICLES:

VERB: throw     V NP (PP)       object
SLOTS: {agent}  {object}        instrument      trajectory
NP/PP: np               PARTICLES: up,away,down

VERB: weigh     V NP    object
SLOTS: {agent}  {object}        instrument
NP/PP: np               PARTICLES:in

 _RB: lift      V NP    object
SLOTS: {agent}  {object}        source  instrument
NP/PP: np               PARTICLES: up

VERB: fall      V PP    pp_obj
VERB: fell      V NP    V PP    object,pp_obj
SLOTS: {agent}  source  destination
NP/PP: pp               PARTICLES: off

VERB: meet      V NP    object
SLOTS: {agent}  {object2}       location
NP/PP: np,pp            PARTICLES: up

VERB: remain    V PP    V ()    intrans,pp_obj
SLOTS: {agent}
NP/PP: pp               PARTICLES:

VERB: pull      V NP    object
  OTS: {agent}  {object}        instrument      source  destination
../PP: np               PARTICLES: out

VERB: block     V NP    object
SLOTS: {agent}  {object2}[location]     instrument
NP/PP: np               PARTICLES: out,up

VERB: carry     V NP    object
SLOTS: {agent}  {object}        instrument(in   source destination
NP/PP: np               PARTICLES:

VERB: travel    V PP    V ()    intrans,pp_obj
SLOTS: {agent}  instrument(in   source  destination
NP/PP: pp               PARTICLES:

VERB: take      V NP NP V NP toPP       V NP    two_obj,object
VERB: took
SLOTS: {agent}  {object}        destination     source  instrument(in
NP/PP: np               PARTICLES:up,out

VERB: come      V (NP)  object,intrans
```

```
SLOTS: {agent}  destination(to,home
NP/PP: pp               PARTICLES: in

VERB: arrive    V ()    V PP    intrans,pp_obj
SLOTS: {agent}  instrument(in
NP/PP: pp               PARTICLES:

VERB: give      V NP toPP       V NP NP two_obj,object
VERB: gave
SLOTS: {agent}[human]   2-object       {object2}[anim] {object1}
NP/PP: np               PARTICLES: up,in,out

VERB: attend    V NP    V ()    V toPP  object,intrans
SLOTS: {agent}[place,meeting,activity]  location
NP/PP: np               PARTICLES: to

VERB: deliver   V NP    V NP toPP       V NP NP two_obj,object
SLOTS: {agent}  {object}        instrument(in
NP/PP: np               PARTICLES:

VERB: change    V NP    object
SLOTS: {agent}  object(what
NP/PP: np               PARTICLES:

VERB: broke     (NP)
SLOTS: {agent}  {object}        instrument      location
NP/PP: np               PARTICLES:

VERB: know      V Sbar  V NP    object
VERB: known     A toPP
SLOTS: {agent}[person]  fact:NP or S
NP/PP: np               PARTICLES:

VERB: believe   V Sbar          V NP VPbar      V NP    V ()    object,intrans
SLOTS: {agent}[person]  fact:NP or S
NP/PP: np               PARTICLES:in

VERB: want      V VPbar         V NP VPbar      V NP    object
SLOTS: {agent}[anim]    object: NP or S
NP/PP: np               PARTICLES:

VERB: look      V PP    V A     ,pp_obj
SLOTS: {agent}[anim]    instrument      location(up,at
NP/PP: pp               PARTICLES: out,up,at

VERB: see       V NP    V Sbar  object
VERB: saw
BOG: deltaobj
SLOTS: {agent}[anim]    object  NP or S
NP/PP: np               PARTICLES:

VERB: seem      V AP            V (toPP) VPbar  V Sbar
SLOTS: {agent}  condition
NP/PP:          PARTICLES:

VERB: tell      V NP Sbar       V NP VPbar      V NP NP two_obj,object
BOG: thatcomp
SLOTS: {agent}  object[person]  item[story]     about[topic]
NP/PP: np               PARTICLES:
```

```
VERB: persuade  V NP Sbar                V NP VPbar      object
SLOTS: {agent}  object[anim]    action(to
NP/PP: np               PARTICLES:

VERB: promise   V NP Sbar        V (NP) VPbar    object,intrans
SLOTS: {agent}  NP and S         object[person]
NP/PP: np               PARTICLES:

VERB: ask       V NP VPbar       V NP NP         V VPbar V NP toPP       two_obj,d
BOG: transcomp
SLOTS: {agent}  object  item
NP/PP: np               PARTICLES:

PUT


SLOT LIST

SLOT            PREPS
agent           np
  ject          np
_bject2         with            %accompany
time            at,before,after,on
instrument      with,using,in
source          from
destination     until,to,into
cost            for
trajectory      over,along,by,up,around % also means where
target          in,on,to
location        in,at
during          from when to when
manner

Prepostions:
in,on,by,about,before,behind,ahead,up,out
above,below,of,if,
from,until,to,with,for,over,at,into

        GAZDAR'S VERB TYPES

_.      V               run,sing
7.      V NP            eat,sing
8.      V NP toPP       hand,give,sing,throw,ask,attach,connect
9.      V NP forPP      buy,cook,reserve
10.     V NP NP         spare,hand,give,buy
11.     V Sbar          know,believe
12.     V NP Sbar       promise,persuade,tell
14.     V VPbar         try,tend,happen,want,prefer,expect
17.     V NP VPbar      want,prefer,expect,believe,persuade,force,ask
20.     V NP VP         make
21.     V (NP) VPbar    promise
22.     V (toPP) VPbar  seem,appear
23.     V AP            be,seem,appear
25.     A               stupid,open,closed
26.     A toPP          known,attracted,drawn
27.     A (byPP)        unloved
28.     A VPbar         likely,eager
30.     A forPP VPbar   eager

        MY VERB TYPES
```

inf_comp	NP VPbar	[np][to][tnsless]	17.
that_comp	Sbar, not that	[np][verb]	
to_less_inf_obj	see,saw	NP or S	20.
to_be_less_inf_comp	seem	[en or adj]	
no_subj	seem	[to][tnsless]	
subj_less_inf_comp	want	[to][tnsless]	14.

NEED TO ADD
V NP NP
V VPbar
V S

Oct81

NB

Robs last addition of counters etc involved changes to:

TOP

RULEM.LPL

NODE.LPL

routines

increment(counter) counter ∈ { rules_checked, rules_run, aHash }

times_called(counter)

```
SUBFILE: FEATS. @11:57 18-JUN-1981 <005> (558)
/* FEATS : Features for rob's parser

                                                        Rob
                                                        Updated: 16 December 80
*/

%% Should be loaded using: load_dict(feats) %%

% This file defines all the features that can be used in dictionary
% entries, buffer matching etc. Each feature is assigned a bit in the
% feature bit-vector. This all happens in the file:
%
%                                                       DBREP.LPL



  ature(noun).              feature(nsstart).
.eature(n1p).               feature(n2p).    feature(n3p).
feature(det).               feature(def).    feature(indef).
feature(wh).                feature(ns).     feature(np1).
feature(tnsless).           feature(past).   feature(pres).
feature(future).            feature(modal).  feature(neg).
feature(en).                feature(ing).
feature(verb).              feature(auxverb).        feature(aux).
feature(v1s).               feature(v3s).    feature(v_3s).
feature(vsp1).              feature(vp1_2s).         feature(v13s).
feature(adj).               feature(prep).   feature(pronoun).
feature(relpron).           feature(ord).    feature(adverb).
feature(comma).             feature(poss).   feature(dim).
feature(np).                feature(pp).     feature(vp).
feature(s).                 feature(major).  feature(sec).
feature(qp).                feature(ap).     feature(binder).
feature(have).              feature(be).     feature(to).
feature(do).                feature(conj).   feature(how).
feature(for).
  ature(inf_comp).          feature(that_comp).      feature(no_subj).
feature(to_less_inf_comp).      feature(to_be_less_inf_comp).
feature(two_obj).

feature(name).              feature(propnoun).       feature(possesive).
feature(trace).             feature(compadv).        feature(time).
feature(variable).          feature(relative).       feature(quant).
feature(than).              feature(that).           feature(all).
feature(than_comp).         feature(comp_s).
feature(poss_np).           feature(relpron_np).
feature(andc).              feature(sent_subj).
feature(quantifier).        feature(unit).
feature(passive).           feature(decl).   feature(imperative).
feature(wh_quest).          feature(ynquest).
feature(np_utterance).  %       feature(pp_utterance).
feature(be_).
feature(comp).  feature(wh_comp).       feature(func).
%feature(inf).  feature(perf).
%feature(pros).             feature(copula).
feature(predp).             feature(part).   feature(of).

/* other feats that are nice to see printed     */
```

```
/*
poss_pronoun,comparative,modifiable,not_modifiable,
poss_det,ordRP,perf,inf,pros,copula,
np_presuposed,trace_np,prop_np,comp_np,pron_np,
quest,see_s,init_s_bar
                    */

/* multiple features sets disjoint groups:
noun
nsstart
n1p,n2p,n3p
def,indef
wh
ns
npl
tnsless
past
pres
future
  ,ing
/erb
auxverb
v1s,v3s,v_3s,v13s,vpl_2s,vspl,quant
adj,all,unit,dim,trace,compadv
prep,ord,adverb,pronoun,quantifier
relpron, passive
comma,fpunc,poss,conj,possesive
np,pp,vp,s,qp,propnoun
major,sec,andc,ppc,aux,ap
have,be,to,do,how,than,that
inf_comp
that_comp
no_subj
to_less_inf_comp
name,time,than_comp,that_s,det
comp_s,poss_np,relpron_np,f_s,
decl,imperative,wh_quest,ynquest,np_utterance,pp_utterance,relative
                              */



\\\\\
```

```
SUBFILE: MORDIC, @22:43 8-APR-1981 <005> (376)
/* MORDIC : Dictionary entries for morphology
                and transfer and coerce

                                                    Rob
                                                    Updated: 14 December 80 (R)
*/

%% Should be loaded using: load_dict(mordic) %%


%  DEEP
%
%     When to use deep holes for attach

deep(pp,np).
deep(pp,pp).
  ep(np,pp).



%  MORPH
%
%     Endins          Add                          Delete

morph(ing,       [pres,adj,ing],          [tnsless,past,noun]).
morph(ed,        [past,en,vspl],          [tnsless,pres,noun,v3s,v_3s]).
morph(en,        [past,en,adj,vspl],      [tnsless,pres,noun,v3s,v_3s]).
morph(er,        [],                      []).
morph(est,       [],                      []).
morph(s,         [npl,pres,v3s],          [tnsless,ns,v_3s]).
morph(es,        [npl,pres,v3s],          [tnsless,ns,v_3s]).
morph(ly,        [adverb,en,ing],         [noun,verb,nsstart]).
morph(ness,      [adverb],                []).
morph(ise,       [verb,tnsless,v3s],      []).

                        % add ism someday.


%  TRANSFER
%
%     Tarset node          Features to be transferred

transfer(np,      [n1p,n2p,n3p,wh,def,indef]).
transfer(vp,      [tnsless,past,pres,future,modal,neg,
                   v1s,v3s,v_3s,v13s,vpl_2s,vspl]).
                  % inf,perf,copula,pros                    (removed)

transfer(pp,      [wh,for]).
transfer(qp,      [n1p,n2p,n3p,ns,npl,wh]).
transfer(ap,      [n1p,n2p,n3p,ns,npl,wh]).
transfer(aux,    [modal,neg,v1s,v3s,v_3s,v13s,vpl_2s,vspl]).
transfer(auxi,    [tnsless,past,pres,future,modal,neg,
                   v1s,v3s,v_3s,v13s,vpl_2s,vspl]).
                  % inf,pros,pref,copula                    (removed)
```

```
%  COERCE
%
%     Node type            Features to be added

coerce(verb,      [verb,en,ing,tnsless,pres,past,
                   v3s,v_3s,vpl,v1s,vi3s,vpl_2s,passive,
                   inf_comp,that_comp,to_less_inf_comp,to_be_less_inf_comp,
                   no_subj,two_obj]).
                  % ins_obj,comp_obj,comp_2_obj,          (removed)
                  % two_obj_inf_obj,obj_binds_delta

coerce(noun,      [noun,ns,npl,n1p,n2p,n3p]).
coerce(adj,       [adj,ins,ns,n3p]).
coerce(modal,     [verb,auxverb,modal,future,pres,past,vspl]).
coerce(comp,      [comp,that]).
coerce(det,       [det,def,indef,ns,npl,n1p,n2p,n3p,wh]).



\\\\\
```

```
SUBFILE: NDICT, @13:42 10-APR-1981 <005> (865)
/* NDICT: Dictionary foor Rob's Parser

        Noun and Adjective definitions

                                                        Rob
                                                        Updated: 13 November 80
*/

%% Should be loaded using: load_dict(ndict) %%


/* adjectives, all have very simple properties */
feature(adjf,[adj,nsstart]).

def(red,adjf).          def(blue,adjf).         def(green,adjf).
def(yellow,adjf).       def(white,adjf).        def(little,adjf).
  f(big,adjf).          def(small,adjf).        def(wee,adjf).
def(fat,adjf).          def(thin,adjf).         def(old,adjf).
def(heavy,adjf).        def(light,adjf).        def(young,adjf).
def(tall,adjf).         def(short,adjf).        def(tiny,adjf).
def(stupid,adjf).       def(smart,adjf).        def(nice,adjf).
def(cute,adjf).         def(ugly,adjf).         def(tanned,adjf).
def(round,adjf).        def(wood,adjf).
def(bright,adjf).       def(wide,adjf).
def(thick,adjf).        def(fine,adjf).         def(rough,adjf).
def(smooth,adjf).       def(long,adjf).         def(fixed,adjf).
def(stationary,adjf).
def(good,adjf).         def(bad,adjf).
def(happy,adjf).        def(sad,adjf).          def(common,adjf).
def(upward,adjf,[adverb]).      def(downward,adjf,[adverb]).
def(here,adjf).
def(weightless,adjf).   def(frictionless,adjf).
def(initial,adjf,[ord]).        def(final,adjf).
def(high,adjf). def(straight,adjf).

   NOUNS        */

feature(nounf,[noun,nsstart,ns,n3p,verb,v3s,tnsless,pres]).
feature(nouni,[noun,nsstart,npl,n3p]).
feature(nouns,[noun,nsstart,ns,n3p]).
feature(nounname,[name,nsstart,ns,n3p,propnoun]).
feature(nounplace,[noun,nsstart,ns,n3p,propnoun]).

def(boy,nouns).         def(student,nouns).     def(exam,nounf).
def(man,nounf).         def(girl,nouns).        def(woman,nouns).
def(men,nouni).         def(wall,nounf).
def(aunt,nouns).        def(uncle,nouns).       def(brother,nouns).
def(elephant,nounf).    def(cat,nounf).         def(dog,nounf).
def(deer,nounf,[npl]).  def(fox,nounf).         def(worm,nounf).
def(cube,nounf).        def(pyramid,nounf).
def(particle,nounf).    def(string,nounf).      def(pulley,nounf).
def(table,nounf).       def(rock,nounf).        def(floor,nounf).
def(lollipop,nounf).    def(parser,nounf).      def(pencil,nounf).
def(jeep,nounf).        def(car,nounf).         def(lorry,nounf).
def(hat,nounf).         def(shoe,nounf).        def(leg,nounf).
def(robot,nounf).       def(point,nounf).       def(arm,nounf).
def(boat,nounf).        def(plane,nounf).       def(book,nounf).
```

```
def(rod,nounf).          def(pole,nounf).          def(peg,nounf).
def(rope,nounf).         def(bridge,nounf).        def(scaffold,nounf).
def(lever,nounf).        def(pulley,nounf).        def(train,nounf).
def(station,nounf).      def(track,nounf).         def(head,nounf).
def(week,nounf,[time]).  def(pipe,nounf).          def(statue,nounf).
def(map,nounf).          def(block,nounf).         def(peir,nounf).
def(end,nounf).          def(park,nounf).          def(ball,nounf).
def(cord,nounf).         def(door,nounf).          def(hinge,nounf).
def(pier,nounf).         def(system,nounf).        def(former,nounf).
def(latter,nounf).       def(upper,nounf).         def(lower,nounf).
def(cliff,nounf).        def(sea,nounf).           def(earth,nounf).
def(ground,nounf).       def(hook,nounf).          def(jump,nounf).
def(stand,nounf).        def(pin,nounf).
def(load,nounf).         def(motion,nounf).
def(tea,nounf).          def(sun,nounf).           def(trash,nounf).
def(mother,nounf).       def(street,nounf).        def(hill,nounf).
def(meeting,nounf,[ing,part]).  def(paper,nounf).        def(fact,nounf).
def(monday,nounf,[time]).       def(tuesday,nounf,[time]).
def(wednesday,nounf,[time]).    def(thursday,nounf,[time]).
  f(friday,nounf,[time]).       def(saturday,nounf,[time]).
def(top,nounf).          def(level,nounf,[adj]).
def(tower,nounf).        def(line,nounf).
def(toy,nounf).  def(journey,nounf).      def(bus,nounf).
def(stop,nounf).         def(start,nounf).         def(catch,nounf).
def(horse,nounf).        def(barn,nounf).          def(end,nounf).
def(vertical,nounf).     def(horizontal,nounf,[adj]).     def(uniform,nounf).
def(building,[noun,ns,n3p,verb,pres,ing,adj,v_3s]).

% unsure
def(rest,nounf).         def(force,nounf).         def(value,nounf).
def(direction,nounf).    def(magnitude,nounf).     def(distance,nounf).
def(age,nounf).          def(height,nounf).

def(rob,nounname).       def(tom,nounname).        def(steve,nounname).
def(sue,nounname).       def(mary,nounname).       def(pam,nounname).
def(al,nounname).        def(chris,nounname).      def(judy,nounname).
def(george,nounname).    def(alfred,nounname).     def(karen,nounname).
def(pat,nounname).       def(leslie,nounname).     def(jill,nounname).
  f(val,nounname).       def(john,nounname).       def(milne,nounname).
def(jack,nounname).

def(edinburgh,nounplace).        def(scotland,nounplace).
def(denver,nounplace).           def(boston,nounplace).
def(london,nounplace).           def(glasgow,nounplace).

def(there,nounplace).
```

\\\\\

```
SUBFILE: MDICT. @1:53 20-JUN-1981 <005> (590)
/* MDICT : Dictionary for Rob's parser

                                                    Rob
                                                    Updated: 6 December 80
*/

%% Should be loaded using: load_dict(mdict) %%



/* DETERMINERS  */

feature(detdef,[det,nsstart,def,n3pl]).
feature(detindef,[det,nsstart,indef,ns,n3pl]).

def(the,detdef,[ns,npl]).
  f(a,detindef,[variable]).      def(an,detindef).
def(every,detindef,[ns,quantifier,adverb]).
def(this,detdef,[ns,n1p,pronoun]).
def(these,detdef,[npl]).         def(those,detdef,[npl]).

/* CONJUNCTIONS, NOT, PUNCUATION          */

def(and,[conj]).          def(or,[conj]).

def(not,[neg,en,ing,tnsless]).
def(no,[neg,en,ing,tnsless]).

/* POSSESIVE    */
def('''',[possesive,poss]).      def('''s',[possesive,poss]).

/* QUANT only sort-of correct now         */
feature(quantifierf,[quantifier,nsstart,adverb]).

def(all,quantifierf,[all,indef,npl]).    def(each,quantifierf,[ns]).
  f(some,quantifierf,[npl]).             def(none,quantifierf,[ns,npl]).
def(both,quantifierf,[npl]).             def(another,quantifierf,[ns]).

def(.,[fpunc]).           def(!,[fpunc]).           def(?,[fpunc]).
def(',',[comma]).

/* PREPOSITIONS */

def(in,[prep,unit]).              def(by,[prep]).          def(on,[prep]).
def(about,[prep]).       def(before,[prep]).      def(behind,[prep]).
def(of,[prep,of]).                def(from,[prep]).        def(until,[prep]).
def(ahead,[prep]).       def(with,[prep]).        def(for,[prep,for]).
def(to,[prep,verb,pres,to]).
def(above,[prep]).       def(below,[prep]).
def(inside,[prep]).    def(over,[prep]).          def(out,[prep]).
def(down,[prep]).        def(up,[prep]).
def(through,[prep]).     def(between,[prep]).     def(along,[prep]).
def(past,[prep,adj]).

feature(ordf,[ord,nsstart]).
```

```
def(next,ordf).          def(last,ordf).          def(first,ordf).
def(other,ordf).         def(same,ordf).

/* SO and SUCH, not sure what to do      */
def(so,[compadv]).       def(such,[compadv]).


/* a couple quantifiers....      */
feature(quantity,[quant,nsstart,npl]).

def(one,[quant,nsstart,ns]).     def(two,quantity).       def(three,quantity).
def(twice,quantity).     def(zero,quantity).
%        intesers set picked up further down
feature(vari,[quant,nsstart,ns,npl,variable]).
def(x1,vari).    def(x2,vari).
def(x3,vari).
def(b,vari).     def(c,vari).
def(m,[vari,unit]).      def(v,vari).


   UNITS for the mechanics.... all semi-defined */
feature(unitf,[ns,unit]).

def(lb,unitf).           def(ft,unitf).   def(yd,unitf).
def(pound,unitf,[noun,verb,tnsless]).    def(stone,unitf,[verb,tnsless,v_3s,noun]).
def(mile,unitf).         def(sm,unitf).   def(metre,unitf).
def('ft/sec',unitf).     def(ton,unitf).
def(inch,unitf).         def(degree,unitf).       def(meter,unitf).
def(year,unitf,[noun,n3p]).      def(time,unitf,[noun,ns,verb,v_3s]).
def(sec,unitf). def(second,unitf,[time,noun,ord]).
def('ms-1',unitf).       def('ms-2',unitf).

/* DIMENSIONS measurable quantities...   */
feature(dimf,[noun,nsstart,ns,dim]).

def(weight,dimf).                def(mass,dimf).          def(velocity,dimf).
def(tension,dimf).               def(length,dimf).
def(acceleration,dimf).
def(masses,dimf,[npl]). def(speed,dimf).
```

\\\\\

```
SUBFILE: PDICT. @21:46 29-MAR-1981 <005> (281)

/* PDICT: Dictionary for Rob's Parser

                Pronouns and WH stuff

                                                        Rob
                                                        Updated: 13 November 80
*/

%% Should be loaded using: load_dict(pdict) %%


/* PRONOUNS all kinds o 'em      */

feature(pronounf,[pronoun,nsstart,n3p]).
feature(pronoun1,[pronoun,nsstart,npl,n1p]).
feature(pronoun2,[pronoun,nsstart,ns,n2p]).
  ature(pronounI,[pronoun,nsstart,ns,n1p]).
,eature(posspn,[pronoun,nsstart,poss,ns,npl]).

def(they,pronounf,[npl]).
def(it,pronounf,[ns]).              def(its,posspn,[n3p]).
def(she,pronounf,[ns]).             def(he,pronounf,[ns]).
def(you,pronoun2,[npl]).
def(we,pronoun1).
def(them,pronounf,[npl]).
def(i,pronounI).                    def(me,pronounI).
def(our,posspn,[n1p]).              def(your,posspn,[n2p]).
def(his,posspn,[n3p]).
def(my,posspn,[n1p]).
def(him,pronounf,[ns]).             def(her,posspn,[n3p]).
def(their,posspn,[n3p]).
def(himself,pronounf,[ns]).         def(herself,pronounf,[ns]).
def(theirselves,pronounf,[npl]).
def(mine,posspn,[n1p]).

  ature(wh_,[relpron,wh,n3p,ns,npl]).
,eature(whpronn,[relpron,n3p,wh]).

def(wh_,wh_).
def(when,whpronn,[ns]).
def(where,whpronn,[ns]).
def(who,whpronn,[npl,ns]).
def(whom,whpronn,[npl,ns]).
def(while,whpronn,[ns,npl,binder]).

def(what,[det,nsstart,ns,npl,n3p,indef,wh,relpron]).
def(which,whpronn,[det,ns,npl,indef,nsstart]).
def(how,whpronn,[how,nsstart]).
def(that,[det,nsstart,def,compp,pronoun,ns,that]).
def(than,[than]).



\\\\\
```

```
SUBFILE: VDICT, @2:17 20-JUN-1981 <005> (937)

/* VDICT: Dictionary for Rob's Parser

                Verb Definitions

                                                        Rob
                                                        Updated: 13 November 80
*/

%% Should be loaded using: load_dict(vdict) %%


/*      ADVERBS these are added for fun
        and not checked */
% this is odd, they should be adj's

feature(adverbf,[verb,tnsless,adj]).
  f(quick,adverbf).      def(slow,adverbf).
def(soft,adverbf).       def(loud,adverbf).
def(quiet,adverbf).      def(harsh,adverbf).
def(yesterday,adverbf,[adverb]).        def(hard,adverbf).
def(just,adverbf).
def(apart,adverbf).      def(tommorrow,adverbf,[adverb]).
def(away,adverbf,[prep]).       def(again,adverbf).

/* AUXVERBS and modals  */
feature(bef,[verb,auxverb,tnsless,be,be_]).
feature(amaux,[verb,auxverb,pres,v1s,be]).
feature(aux2,[verb,be,auxverb,vp1_2s]).
feature(aux3,[verb,auxverb,pres,v3s]).
feature(auxp,[verb,auxverb,past]).
feature(modalf,[verb,modal,auxverb,vsp1]).
feature(modal2,[verb,modal,auxverb]).
feature(auxhave,[verb,have,auxverb,inf_comp,to_less_inf_comp]).
feature(auxpres,[verb,auxverb,pres]).

  f(be,bef).                                    def(is,aux3,[be,sent_subj]).
def(am,amaux).                                  def(are,aux2,[pres]).
def(was,auxp,[v13s,be]).
def(could,modalf,[future]).                     def(can,modalf,[pres,noun,ns,verb]).
def(might,modal,[future]).
def(would,modalf,[past]).                       def(should,modalf,[past]).
def(will,modalf,[future,noun,ns]).              def(must,modalf).
def(were,aux2,[past]).
def(does,aux3,[do]).                            def(did,auxp,[vsp1,do]).
def(done,auxp,[en,do]).                         def(do,auxpres,[tnsless,v_3s,do]).
def(have,auxhave,[tnsless,pres,v_3s]).          def(has,auxhave,[pres,v3s]).
def(had,auxhave,[past,vsp1,ns]).
def(been,auxp,[en,be,be_]).
def(having,auxhave,[pres,ing]).


/* finally the VERBS ....ta da....        */

feature(verbf,[verb,tnsless,pres,v_3s,noun,ns,n3p]).
feature(verbp,[verb,past,en,vsp1]).
feature(verb1,[verb,en,tnsless,pres,vsp1]).
```

```
feature(verb2,[verb,inf_comp,to_less_inf_comp]).
feature(verb3,[verb,tnsless,pres,v_3s,inf_comp,that_comp,no_subj]).
feature(verb4,[verb,en,past,vspl,inf_comp,that_comp,no_subj]).
feature(verbonly,[verb,tnsless,pres,v_3s]).

def(walk,verbf).          def(run,verbf).
def(sit,verbf).           def(go,verbf,[inf_comp]).        def(break,verbf).
def(rape,verbf).          def(kiss,verbf).          def(shoot,verbf).
def(leave,verbf,[two_obj]).
def(attach,verbf).        def(connect,verbf).
def(reach,verbf).         def(show,verbf).          def(determine,verbf).
def(locate,verbf).        def(support,verbf).       def(suspend,verbf).
def(move,verbf).          def(place,verbf).         def(pass,verbf).
def(hang,verbf).          def(drop,verbf,[two_obj]).
def(project,verbf).
def(throw,verbf).         def(thrown,verbp,[en]).  def(weish,verbf).
def(lift,verbf).          def(fall,verbf).          def(meet,verbf).
def(remain,verbonly).     def(pull,verbf).          def(carry,verbf).
def(tapper,verbf).        def(extend,verbf).        def(add,verbf).
def(application,verbf).   def(travel,verbf).        def(build,verbf).
  f(bore,verbf).          def(fly,verbf).  def(destroy,verbf).
def(find,verbf,[inf_comp,that_comp,two_obj]).       def(gain,verbf).
def(take,verbonly,[that_comp,inf_comp,no_subj,two_obj]).                    def(eat,v
def(taken,verbp,[that_comp,inf_comp,no_subj,two_obj]).
def(fish,verbonly,[noun,ns,npl,nsstart]).           def(milk,verbonly,[noun,ns,npl]).
def(elapse,verbf).        def(release,verbf).       def(sleep,verbf).
def(accelerate,verbf).    def(maintain,verbf).      def(decelerate,verbf).
def(manage,verbf).        def(race,verbf).          def(cover,verbf).
def(surprise,verbf).      def(stir,verbf).

def(went,verbp).          def(came,verbp).
def(met,verbp).           def(hung,verbp).          def(left,verbp,[adj]).
def(shown,verb4).         def(found,verb4).

def(arrive,verbonly,[en]).        def(come,verbonly,[en]).
def(give,verbonly).               def(attend,verbonly).
def(analyze,verbonly).            def(deliver,verbonly).
def(change,verbonly,[inf_comp]).          def(schedule,verbf,[inf_comp]).

  f(gave,verbp,[two_obj]).        def(shot,verbp,[en]).
def(took,verbp,[inf_comp,two_obj]).       def(look,verbf).
def(went,verbp).                  def(fell,[verb,past,vspl]).       def(broke,verbp).
def(gone,verbonly,[en]).          def(ran,verbp).
def(known,verbonly,[inf_comp,that_comp]).
def(wonder,verbonly,[inf_comp,that_comp]).
def(likely,verbonly,[inf_comp]).

def(believe,verbonly,[that_comp,inf_comp]).
def(know,verbonly,[that_comp,inf_comp]).
def(knew,verbp,[inf_comp,that_comp]).
def(want,verbonly,[inf_comp,no_subj]).

def(hit,verb1,[two_obj]).         def(born,verbp).
def(see,verb2,[tnsless,v_3s]).    def(saw,verb2,[past,vspl]).
def(tell,verb3,[two_obj]).        def(say,verb3).
def(persuade,verb3).              def(ask,verb3).
def(invite,verb3,[noun]).         def(promise,verb3).

def(seen,verb4).                  def(said,verb4).
def(told,verb4,[two_obj]).
```

```
def(seem,verbonly,[that_comp,to_be_less_inf_comp,no_subj,inf_comp,sent_subj]).

% to_be_less_inf_comp,no_subj = seem only
% no_subj = take,taken before renaming
% to_less_inf_comp = saw,see,have



\\\\\
```

```
SUBFILE: SEMCHK. @1:58 20-JUN-1981 <005> (781)
/* Rob Milne              SEMCHK
                                                    Updated: 6 June 81

Semantic Checks and PP attachment        */

/* SEMANTIC_CHECK questions      */

/* PP_ATTACH see if head nouns are compatable
The NP_PP check is called in cpool when a [np][pp] is found, it does:
1. finds the number of the pp
2. Finds the number of the last Np of the 1st Buffer
3. sets the Np number for the PP
4. calls PP_CHECk with the two np numbers, it then decides
 can set [np][pp], [pp][pp], [np-qp][pp]  always attaches this last case

semantic_check(pp,PP,_,_,_,DB) :-        % of PP always attach
        set_label(PP,Num),
        find( is_prep(Num,of,NP), DB).

semantic_check(pp,PP,_,_,_,DB) :-        % if the PP has a qp, then attach
        set_label(PP,PPnum),
        find( qp_modify(PPnum,QP), DB).

semantic_check(pp,PP,B2,_,NP,DB) :-
        set_label(NP,NPnum),
        set_last_np(NPnum,LNP,DB),       % returns the number of the np
        set_label(PP,PP1),
        find(  is_prep(PP1,Prep,PPNP),DB), !,
        write('trying to attach '),write(PPNP),write(' to '), write(LNP),nl,
        pp_check(Prep,LNP,PPNP,DB).

semantic_check(pp,_,_,_,_,DB) :- !, fail.        % default to not attach

set_last_np(PP,LNP,DB) :-
        find(  pp_linked(PP,NP)&
        is_prep(NP,Prep,NP2),DB),
        set_last_np(NP2,LNP,DB), !.

set_last_np(NP,NP,DB) :- !.


/* PP_CHECK
this sets the NP number for the target of the attach,
and the NP number of the NP that does the PP     */
        % All the pp_check stuff was written by Keith MacKay for
        % an AI2 project.

%       default is false

pp_check(Prep,NP,PP,DB) :-
        find( headnoun(NP,NPword) &
          headnoun(PP,PPword), DB),
        pp_check(Prep,NPword,PPword).

pp_check(on,NP,PP,DB) :-
        find( headnoun(NP,NPword), DB),
        NPword = tension.
```

```
pp_check(Prep,NPword,PPword) :-
        has_property(NPword,PPword),
        !.

pp_check(Prep,NPword,PPword) :-
        has_property(PPword,NPword),
        !.

pp_check(Prep,NPword,PPword) :-
        person(Personlist),
        person_part(Partlist),
        member(NPword,Personlist),
        member(PPword,Partlist),
        !, fail.

pp_check(of,NPword,PPword) :-
        person(Personlist),
        person_part(Partlist),
        member(NPword,Partlist),
        member(PPword,Personlist),
        !.

pp_check(at,NPword,PPword) :-
        has_property(NPword,phys_obj),
        has_property(PPword,position),
        !.

pp_check(on,NPword,PPword) :-
        has_property(NPword,phys_obj),
        has_property(PPword,position),
        !.

pp_check(on,NPword,PPword) :-
        has_property(NPword,action),
        has_property(PPword,phys_obj),
        !.

        % a hack sort of for is 100 m above the sea.

pp_check(Prep,NP,PP,DB) :-
        find( qp_det(NP,QP), DB),
        not( find( headnoun(NP,Noun), DB) ),
        !.

pp_check(Prep,NP,PP,DB) :- !, fail.


has_property(Word,Property) :-
        semantic_def(Word,Semdef),
        member(Property,Semdef),
        !.

/* NOUNS          semantic check */
semantic_check(nouns,Nouns,Next,_,C,DB) :-
        (Next has auxverb # verb), !.

semantic_check(nouns,_,_,_,NP,DB) :-            % needs a headnoun
        set_label(NP,NPnum),
        not(find( headnoun(NPnum,Head),DB) ), !.
```

```
semantic_check(nouns,_,_,_,NP,DB) :-                    % det was singular
        set_label(NP,NPnum),
        find( num(NPnum,Number,Def), DB),
        not( Number = 1), !.

semantic_check(nouns,_,Next,_,_,DB) :-                  % 'ofPP' coming
        set_label(Next,of), !.

semantic_check(nouns,_,Next,_,_,DB) :-                  % statistical hueristic
        Next has nsstart#prep#adverb#pronoun, !, fail.          % don't attach

/* insert Jeep rocks semantic check here */
/* statistical results for nouns:
        attach if next is auxverb or definite verb (noun use)
        don't attach(verb) is next is nsstart,adverb,prep,verb,past,pronoun.
        */

/* REDUCED RELATIVE: uses a heuristic, 'must have a main verb'  */

semantic_check(red_rel,_,_,_,_,DB) :-
        find(curr_sent(S) &
        main_verb(S,_),DB), !.

                        % by explict listing
semantic_check(particle,B1,_,_,_,DB) :-
        set_label(B1,Prep),
        find(main_verb(S,Verb),DB),
        verb_particle(Verb,Prep), !.
```

\\\\\

```
SUBFILE: AGREE. @21:47 14-NOV-1980 <005> (279)

/* AGREE: grammar agreement routines
        check_agree, etc
                                                Rob
                                                14 Nov 80
                */

/* Number agreement checking routines   */


agree_all(num_type,B1,B2,B3) :-
        affix_agree(B1,B3),
        verb_noun_agree(B1,B2), !.

agree(affix_agree,B1,B2) :- affix_agree(B1,B2), !.

 :ree(det,Det,Noun) :-
        (Noun has adj # quant # ord);
        (Noun has noun, det_noun_agree(Det,Noun)), !.

agree(verb_noun,B1,B2) :- verb_noun_agree(B1,B2), !.

agree(det_noun,B1,B2) :- det_noun_agree(B1,B2), !.

agree_23(complex_noun,B2,B3) :-
        not(( B2 has pronoun # name);
        (B2 has modal, B3 has tnsless))), !.

agree_13(and_type,B1,B3) :- !, same_node_type(B1,B3,_).

agree(subj_verb,B1,B2) :- verb_noun_agree(B2,B1), !.

agree(A,_,_) :-  nl, write('** agree failed '),
        write(A),nl, !, fail.
agree_all(A,_,_,_) :- nl, write(' have agree rejected '),
        nl, !, fail.

/* part 2       */

affix_agree(Aux,Verb) :-
        (Aux has have, Verb has en);
        (Aux has be, Verb has en#ing);
        (Aux has modal, Verb has tnsless);
        (Aux has do), !.

verb_noun_agree(Verb,Noun) :-
        (Verb has v3s, Noun has ns & n3p);
        (Verb has v_3s, Noun has not(ns & n3p) );
        (Verb has v13s, (Noun has ns & not(n2p) ));
        (Verb has vspl);
        (Verb has v1s, Noun has n1p & ns);
        (Verb has vpl_2s, (Noun has npl # (n1p & ns))), !.

det_noun_agree(Det,Noun) :-
        (Noun has ns, Det has ns);
        (Noun has npl, Det has npl), !.
```

\\\\\

```
SUBFILE: SEMRUL, @13:58 3-JUN-1981 <005> (1545)
/* SEMRUL : Semantic rules

                                                    Rob
                                                    Updated: 16 December 80 (R)
                load using: load_sem(semrul).
*/



semantics(start, ([ Sentence ],
                        add( sentence(Sentence) ) )).

semantics(wh_quest, ([ Word ],
                        find( curr_sent(S) ),
                        add( wh_quest(S,Word) &
                        stype(S,wh_quest) ) )).

semantics(wh_np, ([ NP, WHword ],
                        find( num(NP,i,WHword) ),
                        add( headnoun(NP,WHword) ) )).

semantics(major_decl_s, ( [],
                        find( curr_sent(S) ),
                        add( stype(S,statement) ) )).

semantics(aux_invert, ( [],
                        find( curr_sent(S) ),
                        add( stype(S,yes_no_question) ) )).

semantics(imperative, ( [],
                        find( curr_sent(S) ),
                        add( stype(S,command) ) )).

semantics(utterance, ([ NP ],
                        find( curr_sent(S) ),
                        add( utterance(S,NP) ) )).

semantics(if_what, ([ S ],
        add( sentence(S) ) )).

semantics(that_s_start, ([ S ],
                        add( embedded_sent(S) ) )).

semantics(inf_s_start, ([ S, NP ],
                        add( embedded_sent(S) &
                             syn_subj(S,NP) ) )).

semantics(propname).

semantics(name, ([ Word, NP ],
                        find( num(NP,i,def) ),
                        add( name(NP,Word) ) )).

semantics(pronoun, ([ NP ],
                        find( num(NP,i,def) ) )).
```

```
semantics(poss_det, ([ DET, NP ],
                        add( poss_det(_,NP) ) )).

semantics(comp_to_np, ([ NP, S ],
                        find( num(NP,i,comp) ),
                        add( np_comp_s(NP,S) ) )).

semantics(vp_attach).

semantics(conj, ([ Num, NP1, NP2 ],
                        add( conj(Num,NP1,NP2) ) )).

semantics(np_complete).

semantics(how_many, ([ AP, Word ],
                        add( wh_trace(_,AP) &
                             intensifier(AP,how) &
                             headadj(AP,Word) ) )).

  mantics(so_that).

semantics(relpron_np, ([ WHword, NP ],
                        add( headnoun(NP,WHword) ) )).

semantics(start_np, ([ NP ],
                        add( num(NP,_,_) ) )).

semantics(det, ([ DET, NP ],
                        find( poss_det(NP,NPlower) &
                                num(NP,_,def) ) )

            or  ([ DET:wh, NP],
                        find( num(NP,_,DET) ))

            or  ([ DET:def, NP ],
                        find( num(NP,_,def) ))

            or ([ DET, NP ],
                        find( num(NP,_,indef) ) )).

semantics(det_qp).                                      % Needs thousht

semantics(quantifier, ([ Q, NP ],
                        add( quantifier(NP,Q) ) )).

semantics(adj, ([ Red, NP ],
                        find( sensym_label(ap,AP) ),
                        add( hasfeat(NP,AP) &
                             headadj(AP,Red) ) )).

semantics(adj_np, ([ Word, NP ],
                        find( num(NP,i,indef) ),
                        add( headadj(NP,Word) ) )).

semantics(pronoun, ([ Word:ns, NP ],
                        add( num(NP,1,pron) &
                             headnoun(NP,Word) ))

                or ([ Word, NP ],
```

```
                        add( num(NP,plur,pron) &
                             headnoun(NP,Word) ) )).

semantics(noun, ([ Word, NP ],
                        find( num(NP,i,_) ),
                        add( headnoun(NP,Word) ) )).

semantics(nouns, ([ Word, NP ],
                        find( num(NP,plur,_) ),
                        add( headnoun(NP,Word) ) )).

semantics(complex_noun, ([ Word, NP ],
                        find( num(NP,_,_) ),
                        add( headnoun(NP,Word) ) )).

semantics(train_t, ([ NP, Wvar ],
                        add( isa(Word,Wvar) ) )).

semantics(syn_subj, ([ NP ],
                        find( curr_sent(S) ),
                        add( syn_subj(S,NP) ) )).

semantics(syn_obj, ([ T:trace ])

               or  ([ PP:pp ])

               or  ([ NP ],
                        find( curr_sent(S) &
                              syn_obj(S,_) ),
                        add( np_object(S,NP) ))

               or  ([ NP ],
                        find( curr_sent(S) ),
                        add( syn_obj(S,NP) ) )).

semantics(qp_attach, ([ QP, NP ],
                        find( headnoun(NP,_) ),
                        add( qp_modify(NP,QP) ) )

                or      ([ QP, NP ],
                        add( qp_det(NP,QP) ) )).

semantics(qp_attach1, ([ QP ],
                        find( measure(QP,X,arbs) ))

                or      ([ QP ] )).

semantics(qp_units, ([ C, Word ],
                        find( measure(C,_,Word) ) )).

semantics(dim, ([ Word:det, NP ],
                        find( dim_var(Word,DV) &
                              num(NP,i,_) ),
                        add( headnoun(NP,Word) &
                             dim(NP,Word,DV) ) )

           or  ([ Word, NP ],
                        find( dim_var(Word,DV) &
                              num(NP,_,indef) ),
                        add( headnoun(NP,Word) &
```

```
                                dim(NP,Word,DV) ) )).

semantics(qp_pp, ([ QP, PP ],
                  add( qp_modify(PP,QP) ) )).

semantics(np_qp).

semantics(ft_long, ([ AP, QP, ADJ ],
                        add( qp_modify(AP,QP) &
                             headadj(AP,ADJ) ) )).

semantics(ap_attach, ([ NP, AP ],
                        add( hasfeat(NP,AP) ) )).

semantics(prep).

semantics(attach_prep, ([ Prep, PP ],
                        add( is_prep(PP,Prep,_) ) )).

semantics(pp_sets_np, ([ PP, NP ],
                        find( is_prep(PP,_,NP) ) )).

semantics(np_pp_default, ([ NP, PP ],                                   % Needs thousht
        add( pp_linked(NP,PP)  ) )).


semantics(conj_qp_1, ([ QP1, QP2 ],
                        add( conj(QP1,QP2,_) ) )).

semantics(conj_qp_2, ([ QP1, QP2 ],
                        find( conj(QP1,_,QP2) ) )).

semantics(quant, ([ QP, Word ],
                        add( measure(QP,Word,_) ) )).

semantics(ordinal, ([ Word, NP ],
                        add( headadj(NP,Word) ) )).

semantics(rel_attach, ([ S, NP ],
                        find( wh_trace(NP,_) ),
                        add( relc(NP,S) ) )).

semantics(wh_relative_clause, ([ S ],
                        add( embedded_sent(S) ) )).

semantics(np_pp, ([PP, NP ],
        add( pp_linked(NP,PP) ) )).

semantics(tom_mary).

semantics(poss_np).

semantics(comma).

semantics(np_done, ( [ NP ],
                        find( num(NP,Num,indef) ))

                or      ( [ NP ]) ).
```

```
semantics(than_comp).                                    % Needs thousht

semantics(to_infinitive).

semantics(start_aux).

semantics(aux_attach, ([ AUX:pres , C ],
                              add(  aux_verb(C,pres) ))
                or      ([ AUX:past , C ],
                              add(  aux_verb(C,past) ))
                or      ([ AUX:future , C ],
                              add(  aux_verb(C,future ) ))
                or      ([ AUX, C ],
                              add(  aux_verb(C,tnsless) )) ).

semantics(aux_adverb, ([ Word ],
                        find( curr_sent(S) ),
                        add( adverb(S,Word) ) )).

semantics(do_support).

semantics(simple_neg, ([],
                        find( curr_sent(S) ),
                        add( negative_sent(S) ) )).

semantics(be_pred).

semantics(part, ([ Part ],
                        find( main_verb(S,Verb) ),
                        add( particle(Verb,Part) ) )).

semantics(trace, ([ Trace:pp ],
                        find( is_prep(Trace,_,NP) &
                              num(NP,i,trace) ),
                        add( wh_trace(_,NP) ) )

                or ([ Trace ],
                        find( num(Trace,1,trace) ),
                        add( wh_trace(_,Trace) ) )).

semantics(bind_trace, ([ BIND ],
                        find( wh_trace(BIND,_) ) )).

semantics(drop_vp_trace, ([ Trace:pp ],
                find(curr_sent(S) ),
                add(pp_linked(S,Trace) ) )

                or  ([ Trace ],
                        find( curr_sent(S) ),
                        add( syn_obj(S,Trace) ) )).

semantics(passive, ([ Trace ],
                        find( curr_sent(S) &
                              syn_subj(S,Subj) ),
                        add( passive_sent(S) &
                             wh_trace(Subj,Trace) ) )).

semantics(passive_aux, ( [],
                        find( curr_sent(S) ),
                        add( passive_sent(S) ) )).
```

```
semantics(create_delta_subj, ([ Trace ],
                        find( curr_sent(S) &
                              syn_subj(S,Subj) ),
                        add( wh_trace(Subj,Trace) ) )).

semantics(main_verb, ([ VP ],
                        find( curr_sent(S) &
                              irreg_verb(VP,Root) ),
                        add( main_verb(S,Root) ) )

                  or  ([ VP ],
                        find( curr_sent(S) ),
                        add( main_verb(S,VP) ) )).

semantics(pp_under_x, ([ PP ],
                        find( curr_sent(S) ),
                        add( pp_linked(S,PP) ) )).

 emantics(adverb, ([ ADV ],
                        find( curr_sent(S) ),
                        add( adverb(S,ADV) ) )).

semantics(adverb_group, ([ NUM, ADV1, ADV2 ],
        add(  hasfeat(NUM,ADV2) &
              hasfeat(NUM,ADV1) ) )).

semantics(reduced_rel).

semantics(predp, ([ PP ],
                        find( curr_sent(S) &
                              is_prep(PP,_,NP) ),
                        add( syn_obj(S,NP) &
                                main_verb(S,be) ) )

              or  ([ AP ],
                        find( curr_sent(S) ),
                        add( syn_obj(S,AP) &
                                main_verb(S,be) ) )).

semantics(that_s_start_1, ([ NP, S ],
                        add( embedded_sent(S) &
                             syn_subj(S,NP) ) )).

semantics(inf_s_start_1, ([ NP, S ],
                        add( embedded_sent(S) &
                             syn_subj(S,NP) ) )).

semantics(insert_to, ([ S, NP ],
                        add( embedded_sent(S) &
                             syn_subj(S,NP) ) )).

semantics(obj_in_embedded_s, ([ NP ],
                        find( curr_sent(S) ),
                        add( syn_obj(S,NP) ) )).

semantics(vp_done).

semantics(embedded_s_done).
```

```
semantics(s_done).

semantics(init_s_bar).

semantics(hypo_s).


\\\\\
```

```
SUBFILE: SEMSUP.SPL @1:53 20-JUN-1981 <005> (783)
/* SEMSUP.SPL : Semantics Support routines
        contains word to number, semantic_def, irreg_verb and verb_particle

                                                    Rob
                                                    Updated: 20 June 81
*/


  :- public  word_to_num/2,
             semantic_def/2,
             person/1,
             person_part/1,
             irreg_verb/2,
             verb_particle/2.

  :- mode word_to_num(+,?),
          semantic_def(+,?),
          person(?),
          person_part(?),
          irreg_verb(+,?),
          verb_particle(+,+).


word_to_num(one,1) :- !.
word_to_num(two,2) :- !.
word_to_num(three,3) :- !.
word_to_num(four,4) :- !.
word_to_num(five,5) :- !.
word_to_num(X,X) :- !.

/* Table of properties of words .. Semantic dictionary. */
/* Intended for PP attachment                           */

semantic_def(particle,[mass,velocity,acceleration,phys_obj]).
semantic_def(block,[mass,length,height,velocity,acceleration,phys_obj]).
  antic_def(room,[length,width,height,wall,floor,ceiling,door,phys_obj]).
emantic_def(wall,[mass,length,height,point,phys_obj]).
semantic_def(ceiling,[width,length,height,point,phys_obj]).
semantic_def(ball,[mass,length,height,velocity,phys_obj]).
semantic_def(station,[length,height,phys_obj]).
semantic_def(train,[mass,length,height,velocity,acceleration,phys_obj]).
semantic_def(rod,[mass,length,phys_obj]).
semantic_def(cue,[mass,length,velocity,acceleration,phys_obj]).
semantic_def(jeep,[mass,velocity,acceleration,length,height,phys_obj]).
semantic_def(car,[mass,velocity,acceleration,length,width,height,phys_obj]).
semantic_def(lorry,[mass,velocity,acceleration,length,height,width,phys_obj]).
semantic_def(spring,[constant,tension,length,mass,elasticity,
                                                    extension,phys_obj]).
semantic_def(rope,[tension,length,end,phys_obj]).
semantic_def(string,[tension,length,end,phys_obj]).
semantic_def(pulley,[mass,diameter,phys_obj]).
semantic_def(man,[mass,height,phys_obj]).
semantic_def(boy,[mass,height,phys_obj]).
semantic_def(woman,[mass,height,phys_obj]).
semantic_def(girl,[mass,height,phys_obj]).
semantic_def(tom,[mass,height,phys_obj]).
semantic_def(mary,[mass,height,phys_obj]).
```

```
semantic_def(driver,[mass,height,phys_obj]).
semantic_def(painter,[mass,height,phys_obj]).
semantic_def(pair,[mass,length,phys_obj]).

semantic_def(edge,[position]).
semantic_def(corner,[position]).
semantic_def(end,[position]).
semantic_def(height,[position]).

semantic_def(force,[action]).
semantic_def(tension,[action]).

person([boy,girl,mother,father,men,women,woman,man]).
person_part([arm,leg,head,foot,body,teeth,hair,hand]).

/* irregular verb lists, used by semantics in Main_Verb assertions      */

irreg_verb(is,be).                irreg_verb(are,be).
irreg_verb(was,be).               irreg_verb(were,be).
  reg_verb(has,have).             irreg_verb(had,have).
irreg_verb(broke,break).          irreg_verb(thrown,throw).
irreg_verb(came,come).            irreg_verb(hung,hang).
irreg_verb(shown,show).           irreg_verb(shot,shoot).
irreg_verb(fell,fall).            irreg_verb(told,tell).
irreg_verb(found,find).           irreg_verb(knew,know).
irreg_verb(saw,see).              irreg_verb(seen,saw).
irreg_verb(did,do).               irreg_verb(done,do).
irreg_verb(does,do).              irreg_verb(born,bear).
irreg_verb(taken,take).

verb_particle(walk,on).           verb_particle(run,down).
verb_particle(run,away).          verb_particle(look,up).
verb_particle(go,on).             verb_particle(go,in).
verb_particle(break,in).          verb_particle(shoot,up).
verb_particle(shoot,out).
verb_particle(leave,out).         verb_particle(left,behind).
verb_particle(show,up).
verb_particle(find,out).          verb_particle(pass,out).
  rb_particle(hang,on).           verb_particle(drop,out).
 _rb_particle(throw,up).          verb_particle(weigh,in).
verb_particle(meet,up).           verb_particle(pull,out).
verb_particle(block,up).
verb_particle(come,in).           verb_particle(give,up). verb_particle(give,out).
verb_particle(attend,to).         verb_particle(shot,up). verb_particle(shot,at).
verb_particle(take,out).          verb_particle(stir,up).



\\\\\
```

```
SUBFILE: SSTART.PK @15:48 15-SEP-1981 <005> (1022)
/* SSTART.PK : Packet   SS_START
               assumes C is sentence start

                                                    Rob
                                                    Updated: 1 June 81 (R)
*/


  :- mode ss_start(+,-,-,-,-,?).
  :- mode ss_start(+,+,+,+,+,+,+,+,+,?).

/*-------------------------------------------------------------------------*/

/* rule IF_WHAT?: [binder] -> then make a hypothetical sentence */

ss_start(5, (binder), t, t, t, if_what).

  ss_start(if_what,B1,B2,B3,[C|TL],AS,[APacks|Packets],[U1|Unseen],DBold)  :-
        new_node(s,[binder],S1),
        push_sent(S1,DBold,DB),
        attach(B1,S1,binder,S2),
        semantics(if_what,DB,S1),
        !, rulematch(B2,B3,U1,[C,S2|TL],Rulematch,AS,
                        [APacks,[cpool,ss_start]|Packets],Unseen,DB).

/** rule WH_QUEST: [wh] -> attach 1st as wh_comp, wh_quest      */

ss_start(10, (wh & (np # pp # ap)), t, t, t, wh_quest).

  ss_start(wh_quest,B1,B2,B3,[C|TL],AS,[APacks|Packets],[U1|Unseen],DB):-
        addfeats(C,[major,wh_quest],C1),
        deactivate(ss_start,APacks,P1),
        activate(parse_subj,P1,P2),
        attach(B1,C1,wh_comp,C2),
        semantics(wh_quest,DB,B1),
        (B2 has verb & not(auxverb), !,
        (new_node(np,[trace],B11),
        semantics(start_np,DB,B11),
        semantics(trace,DB,B11),
        semantics(bind_trace,DB,B1),
        !, rulematch(B11,B2,B3,[C2|TL],Rulematch,AS,[P2|Packets],
                                                [U1|Unseen],DB));
        ( !,
        rulematch(B2,B3,U1,[C2|TL],Rulematch,[[wh_comp,B1]|AS],[P2|Packets],Unse

/** rule ADVERB: [adverb][nsstart] -> attach 1st as adverb.    */
               % added by Karen Archbold April 1981

ss_start(10,(adverb),(adverb # nsstart),t,t,adverb).

 ss_start(adverb,B1,B2,B3,[C|TL],AS,Packets,[U1|Unseen],DB):-
        semantics(adverb,DB,B1),
        attach(B1,C,adverb,C1),
        !, rulematch(B2,B3,U1,[C1|TL],Rulematch,AS,Packets,Unseen,DB).

/** rule MAJOR_DECL_S: [np][verb] -> label c decl, major,
               change packets  */
```

```
ss_start(10, (np), (verb), t, t, major_decl_s).

  ss_start(major_decl_s,B1,B2,B3,[C|TL],AS,[APacks|Packets],Unseen,DB) :-
        addfeats(C,[s,decl,major],C1),
        deactivate(ss_start,APacks,P1),
        activate(parse_subj,P1,P2),
        semantics(major_decl_s,DB),
        !, rulematch(B1,B2,B3,[C1|TL],Rulematch,AS,[P2|Packets],Unseen,DB).

/* rule AUX_INVERT: [auxverb][nsstart] -> push aux onto AS       */

ss_start(10, (auxverb), (nsstart), t, t, aux_invert).

  ss_start(aux_invert,B1,B2,B3,[C|TL],AS,[APacks|Packets],[U1|Unseen],DB) :-
        addfeats(C,[s,ynquest,major],C1),
        deactivate(ss_start,APacks,P1),
        activate(parse_subj,P1,P2),
        semantics(aux_invert,DB),
        !, rulematch(B2,B3,U1,[C1|TL],Rulematch,[[aux,B1]|AS],
                                        [P2|Packets],Unseen,DB).

/** rule NP_PP_DEFAULT: [np][pp] -> attach to np as PP  */
/* only when clause initial      */

ss_start(10, (np), (pp), t, t, np_pp_default).

  ss_start(np_pp_default,B1,B2,B3,C,AS,Packets,[U1|Unseen],DB):-
        attach(B2,B1,pp,B12),
        semantics(np_pp_default,DB,B1,B2),
        !, rulematch(B12,B3,U1,C,Rulematch,AS,Packets,Unseen,DB).

/** rule NP_UTTERANCE: [np][fpunc] -> done.      */

ss_start(10, (np), (fpunc), t, t, np_utterance).

  ss_start(np_utterance,B1,B2,B3,[C|TL],AS,Packets,Unseen,DBold):-
        addfeats(C,[major,np_utterance],C1),
        semantics(utterance,DBold,B1),
        attach(B1,C1,np,C2),
        attach(B2,C2,fpunc,C3),
        pop_sent(DBold,DB),
        alldone([C3|TL],DB).

/** rule PP_UTTERANCE: [pp][fpunc] -> done       */

ss_start(10, (pp), (fpunc), t, t, pp_utterance).

  ss_start(pp_utterance,B1,B2,B3,[C|TL],AS,Packets,Unseen,DBold) :-
        addfeats(C,[major,np_utterance],C1),              % was pp_utterance
        semantics(utterance,DBold,B1),
        attach(B1,C1,pp,C2),
        attach(B2,C2,fpunc,C3),
        pop_sent(DBold,DB),
        alldone([C3|TL],DB).


/** rule IMPERATIVE: [tnsless] -> insert you into the buffer     */
/* lexical ambiguity should also make 1st a verb         */
% doesn't work for have
```

```
ss_start(10, (tnsless), t, t, t, imperative).

  ss_start(imperative,B1,B2,B3,[C|TL],AS,[APacks|Packets],Unseen,DB) :-
        coerce(verb,B1,B11),
        addfeats(C,[s,imperative,major],C1),
        deactivate(ss_start,APacks,P1),
        activate(parse_subj,P1,P2),
        lookup(you,U2),
        semantics(imperative,DB),
        !, rulematch(U2,B11,B2,[C1|TL],Rulematch,AS,[P2|Packets],
                [B3|Unseen],DB).


/* rule FRONTED_PP: [pp] -> attach to C */
        % should set AS'd and recovered later or trace in

ss_start(10, (pp), t, t, t, fronted_pp).

  ss_start(fronted_pp,B1,B2,B3,[C|TL],AS,Packets,[U1|Unseen],DB) :-
        attach(B1,C,pp,C1),
        semantics(pp_under_x,DB,B1),
        !, rulematch(B2,B3,U1,[C1|TL],Rulematch,AS,Packets,Unseen,DB).



/* rule WH_NP: [wh] -> attach to c, for wh_quest        */

ss_start(15, (wh), t, t, t, wh_np).

  ss_start(wh_np,B1,B2,B3,C,AS,Packets,Unseen,DB) :-
        new_node(np,B11),
        semantics(start_np,DB,B11),
        attach(B1,B11,wh_comp,B12),
        semantics(wh_np,DB,B11,B1),
        !, rulematch(B12,B2,B3,C,Rulematch,AS,Packets,Unseen,DB).

/* rule Kissing_Aunts: [verb,ing,adj][noun,npl] -> np,vp        */
        % This rule is a HACK HACK HACK only

ss_start(10, (verb & ing & adj), (noun & npl), t, t, kissing).

  ss_start(kissing,B1,B2,B3,C,AS,Packets,[U1|Unseen],DB) :-
        new_node(np,[ap,ns,npl],B11),
        attach(B1,B11,verb,B12),
        attach(B2,B12,verb,B13),
        !, rulematch(B13,B3,U1,C,Rulematch,AS,Packets,Unseen,DB).



\\\\\
```

```
SUBFILE: CPOOL.PK @15:49 15-SEP-1981 <005> (1047)
/* CPOOL.PK : Packet    CPOOL
                assumes C is anything

                                                    Rob
                                                    Updated: 24 November 80 (R)
*/


  :- mode cpool(+,-,-,-,-,?).
  :- mode cpool(+,+,+,+,+,+,+,+,?).

/*-----------------------------------------------------------------------*/

/* rule X_AND_X: [x][conj][x] ->x conjoined      */

cpool(5, t, (conj), t, agree_13(and_type), x_and_x).

  cpool(x_and_x,B1,B2,B3,C,AS,Packets,[U1,U2|Unseen],DB) :-
        same_node_type(B1,B3,Feat),
        new_node(Feat,B11),
        attach(B1,B11,Feat,B12),
        attach(B2,B12,conj,B13),
        attach(B3,B13,Feat,B14),
        semantics(conj,DB,B11,B1,B3),
        !, rulematch(B14,U1,U2,C,Rulematch,AS,Packets,Unseen,DB).

/** rule POSS_DET: [poss_np] -> make a det and drop.     */

cpool(5, (poss_np), t, t, agree(det), poss_det).

  cpool(poss_det,B1,B2,B3,C,AS,Packets,[U1|Unseen],DB) :-
        new_node(det,[nsstart,ns,np|]),B31),      % should fix this number stuff
        semantics(poss_det,DB,B31,B1),
        attach(B1,B31,np,B32),
        !, rulematch(B32,B2,B3,C,Rulematch,AS,Packets,[U1|Unseen],DB).

  * rule SO_THAT: [so,such][that] -> that-as-a-comp      */

cpool(10, (compadv), (that), t, t, so_that).

  cpool(so_that,B1,B2,B3,C,AS,Packets,Unseen,DB) :-
        new_node(comp,[that,binder],B11),
        semantics(so_that,DB,B1,B2),
        attach(B1,B11,compadv,B12),
        attach(B2,B12,that,B13),
        lookup(',',U1),
        !, rulematch(U1,B13,B3,C,Rulematch,AS,Packets,Unseen,DB).

/** rule PROPNAME: [name, not np] -> new np node.        */

cpool(10, (name & not(np)), t, t, t, propname).

  cpool(propname,B1,B2,B3,C,AS,Packets,Unseen,DB):-
        new_node(np,[name],C1),
        semantics(start_np,DB,C1),
        semantics(propname,DB,C1,B1),
        !, rulematch(B1,B2,B3,[C1|C],Rulematch,AS,[[build_name]|
```

```
                                                    Packets],Unseen,DB).

/** rule PROPNOUN: [propnoun] -> np in 1st buffer          */

cpool(10, (propnoun ), t, t, t, propnoun).

  cpool(propnoun,B1,B2,B3,C,AS,Packets,Unseen,DB) :-
        new_node(np,B11),
        semantics(start_np,DB,B11),
        semantics(propnoun,DB,B11),
        attach(B1,B11,noun,B12),
        !, rulematch(B12,B2,B3,C,Rulematch,AS,Packets,Unseen,DB).

/** rule PP: [prep][nsstart]-> B1 <- PP,  attach 2nd to c as prep
                attach 3rd to c as np   cf left out      */

cpool(10, (prep), (nsstart), t, t, pp).

  cpool(pp,B1,B2,B3,C,AS,Packets,Unseen,DB) :-
        new_node(pp,B11),
        semantics(prep,DB,B11),
        !, rulematch(B1,B2,B3,[B11|C],Rulematch,AS,
                                [[parse_pp,cpool]|Packets],Unseen,DB).

/* rule MARKED_STARTNP: [det, agree_det] -> start a new np node */

cpool(10, (det), t, t, agree(det), marked_startnp).

  cpool(marked_startnp,B1,B2,B3,C,AS,Packets,Unseen,DB) :-
        new_node(np,C1),
        semantics(start_np,DB,C1),
        !, rulematch(B1,B2,B3,[C1|C],Rulematch,AS,
                [[parse_det,npool]|Packets],Unseen,DB).

/** rule STARTNP: [nsstart] -> start a new NP node       */

cpool(10, (nsstart & not(pronoun # det)), t, t, t, startnp).
        % the above pattern is needed for historical reasons

  cpool(startnp,B1,B2,B3,C,AS,Packets,Unseen,DB) :-
        new_node(np,C1),
        semantics(start_np,DB,C1),
        !, rulematch(B1,B2,B3,[C1|C],Rulematch,AS,
                [[parse_qp_1,npool]|Packets],Unseen,DB).


/* rule THAN_COMP: [than_comp][np] -> attach B2 to B1 as comparative    */

cpool(10, (than_comp), (np), t, t, than_comp).

 cpool(than_comp,B1,B2,B3,C,AS,PAckets,[U1|Unseen],DB) :-
        attach(B2,B1,than_comp,B11),
        semantics(than_comp,DB,B1,B2),
        !, rulematch(B11,B3,U1,C,Rulematch,AS,PAckets,Unseen,DB).

/** rule AND: [conj] -> stuff onto active stack */

cpool(10, (conj & not(andc)), t, t, t, and).

  cpool(and,B1,B2,B3,C,AS,Packets,[U1|Unseen],DB) :-
```

```
        addfeats(B1,andc,B11),
        !, rulematch(B2,B3,U1,[B11|C],Rulematch,
                        AS,[[cpool,parse_vp,parse_conj]|Packets],Unseen,DB).

/** rule COMP_TO_NP: [comp_s] -> make an np in B1          */

cpool(10, (comp_s), t, t, t, comp_to_np).

  cpool(comp_to_np,B1,B2,B3,C,AS,Packets,Unseen,DB) :-
        new_node(np,B11),                  % was labeled comp_np
        semantics(start_np,DB,B11),
        semantics(comp_to_np,DB,B11,B1),
        attach(B1,B11,s,B12),
        !, rulematch(B12,B2,B3,C,Rulematch,AS,Packets,Unseen,DB).

/* rule NP_PP [pp] -> consider to attach the pp to the np          */

cpool(10, (pp), t, t, sem_chk(pp), np_pp).

  cpool(np_pp,B1,B2,B3,[C|TL],AS,Packets,[U1|Unseen],DB) :-
        attach(B1,C,pp,C11,DB),
        semantics(np_pp,DB,B1,C),
        !, rulematch(B2,B3,U1,[C11|TL],Rulematch,AS,Packets,Unseen,DB).

/** rule PRONOUN: [pronoun] -> attach to c, fix feats    */

cpool(15, (pronoun), t, t, t, pronoun).

  cpool(pronoun,B1,B2,B3,C,AS,Packets,Unseen,DB):-
        new_node(np,B11),
        semantics(pronoun,DB,B1,B11),
        attach(B1,B11,pronoun,B12),
        (B1 has poss, !, addfeats(B12,poss_np,B13) ; B13 = B12),
        !, rulematch(B13,B2,B3,C,Rulematch,AS,Packets,Unseen,DB).

/** rule VP_ATTACH: [vp] -> attach to s */

cpool(10, (vp), t, t, t, vp_attach).

  cpool(vp_attach,B1,B2,B3,[C|TL],AS,Packets,[U1|Unseen],DB):-
        attach(B1,C,vp,C1),
        semantics(vp_attach,DB,B1,C),
        !, rulematch(B2,B3,U1,[C1|TL],Rulematch,AS,Packets,Unseen,DB).


/* rule POSS_NP: [poss] -> attach as poss          */

cpool(15, t, (possesive), t, t, poss_np).

  cpool(poss_np,B1,B2,B3,C,AS,Packets,[U1|Unseen],DB):-
        addfeats(B1,poss_np,B11),
        attach(B2,B11,poss,B12),
        semantics(poss_np,DB,B2,B1),
        !, rulematch(B12,B3,U1,C,Rulematch,AS,Packets,Unseen,DB).



\\\\\
```

```
SUBFILE: NPOOL.PK @15:51 15-SEP-1981 <005> (540)
/* NPOOL.PK : Packet    NPOOL
                assumes C is an NP being built

                                                    Rob
                                                    Updated: 6 December 80 (R)
*/


  :- mode npool(+,-,-,-,-,?).
  :- mode npool(+,+,+,+,+,+,+,+,?).

/*-------------------------------------------------------------------------*/

/** rule QP_AND_QUANT: [qp][and][quant] -> new qp node on c, attach B1 and B2   *

  pool(10, (qp), (conj), (quant), t, qp_and_quant).

  npool(qp_and_quant,B1,B2,B3,C,AS,Packets,[U1,U2|Unseen],DB) :-
        new_node(qp,C1),
        attach(B1,C1,qp,C2),
        attach(B2,C2,conj,C3),
        semantics(conj_qp_1,DB,C1,B1),
        !, rulematch(B3,U1,U2,[C3|C],Rulematch,AS,Packets,Unseen,DB).


/** rule LONGER_THAN: [than][name] -> make a comparative          */
        %       does only "ap than name"

npool(10, (than), (name), t, t, longer_than).

  npool(longer_than,B1,B2,B3,[C|TL],AS,Packets,[U1,U2|Unseen],DB) :-
        addfeats(C,than_comp,C11),
        attach(B1,C11,than,C12),
        attach(B2,C12,name,C13),
        semantics(than_comp,DB,C12,B2),
        !, rulematch(B3,U1,U2,[C13|TL],Rulematch,AS,Packets,Unseen,DB).


/** rule 3 FT/SEC: [qp][units] -> new qp in B1  */

npool(10, (qp), (unit), t, t, qp_units).

  npool(qp_units,B1,B2,B3,C,AS,Packets,[U1|Unseen],DB) :-
        semantics(qp_units,DB,B1,B2),
        addfeats(B1,[unit,ns],B11),
        attach(B2,B11,unit,B12),
        !, rulematch(B12,B3,U1,C,Rulematch,AS,Packets,Unseen,DB).

/** rule FT_LONG: [qp][adj] -> new ap, attach as adj    */

npool(10, (qp), (adj), t, t, ft_long).

  npool(ft_long,B1,B2,B3,C,AS,Packets,[U1|Unseen],DB) :-
        new_node(ap,AP1),
        attach(B1,AP1,qp,B11),
        attach(B2,B11,adj,B12),
```

```
        semantics(ft_long,DB,B11,B1,B2),
        !, rulematch(B12,B3,U1,C,ap_attach,AS,Packets,Unseen,DB).

/** rule NOUN_QP: [quant] -> new qp node          */

npool(10, (quant), t, t, t, noun_qp).

  npool(noun_qp,B1,B2,B3,C,AS,Packets,Unseen,DB) :-
        new_node(qp,B11),
        attach(B1,B11,quant,B12),
        semantics(quant,DB,B11,B1),
        semantics(np_qp,DB,B11,C),
        !, rulematch(B12,B2,B3,C,Rulematch,AS,Packets,Unseen,DB).

/** rule AP_ATTACH: [ap] -> attach to c as AP    */

npool(10, (ap), t, t, t, ap_attach).

  npool(ap_attach,B1,B2,B3,[C|TL],AS,Packets,[U1|Unseen],DB) :-
        attach(B1,C,ap,C1,DB),
        semantics(ap_attach,DB,C,B1),
        !, rulematch(B2,B3,U1,[C1|TL],Rulematch,AS,Packets,Unseen,DB).

/** rule QP_ATTACH: [qp] -> attach to c as qp    */

npool(10, (qp), t, t, t, qp_attach).

  npool(qp_attach,B1,B2,B3,[C|TL],AS,Packets,[U1|Unseen],DB):-
        attach(B1,C,qp,C1,DB),
        semantics(qp_attach,DB,B1,C),
        semantics(qp_attach1,DB,B1),              % adds the arbs
        (C has qp, !, semantics(conj_qp_2,DB,C,B1),
        !, rulematch(C1,B2,B3,TL,Rulematch,AS,Packets,[U1|Unseen],DB);
        !, rulematch(B2,B3,U1,[C1|TL],Rulematch,AS,Packets,Unseen,DB)).



  \\\
```

```
SUBFILE: PDET.PK @19:29 10-MAR-1981 <005> (125)
/* PDET.PK : Packet      PARSE_DET
                assumes C if a NP without det

                                                        Rob
                                                        Updated: 7 March 81 (R)
*/


  :- mode parse_det(+,-,-,-,-,?).
  :- mode parse_det(+,+,+,+,+,+,+,+,?).

/*-------------------------------------------------------------------------*/

/* rule DETERMINER: [det] -> attach      */

  rse_det(10, (det), t, t, t, determiner).

  parse_det(determiner,B1,B2,B3,[C|TL],AS,[APacks|Packets],[U1|Unseen],DB):-
        attach(B1,C,det,C1),
        deactivate(parse_det,APacks,P2),
        activate(parse_gp_2,P2,P3),
        semantics(det,DB,B1,C),
        !, rulematch(B2,B3,U1,[C1|TL],Rulematch,AS,[P3|Packets],Unseen,DB).



\\\\\
```

```
SUBFILE: PQP1.PK @15:54 15-SEP-1981 <005> (474)
/* PQP1.PK : Packet      PARSE_QP_1
                assumes C is a NP beins built without det

                                                    Rob
                                                    Updated: 10 March 81 (R)
*/



  :- mode parse_qp_1(+,-,-,-,-,?).
  :- mode parse_qp_1(+,+,+,+,+,+,+,+,?).

/*-------------------------------------------------------------------------*/


/* rule HOW_MANY:[how][adj] -> combine into how only     */

parse_qp_1(10, (how), (adj), t, t, how_many).

  parse_qp_1(how_many,B1,B2,B3,C,AS,Packets,[U1|Unseen],DB) :-
        new_node(qp,[how,relpron,pronoun],AP1),
        semantics(how_many,DB,AP1,B2),
        attach(B1,AP1,how,AP2),
        attach(B2,AP2,adj,B11),
        !, rulematch(B11,B3,U1,C,Rulematch,AS,Packets,Unseen,DB).

/** rule QUANT: [quant] -> new qp node in B1     */

parse_qp_1(10, (quant), (adj * noun), t, t, quant).

  parse_qp_1(quant,B1,B2,B3,C,AS,Packets,Unseen,DB):-
        new_node(qp,B13),
        attach(B1,B13,quant,B12),
        semantics(quant,DB,B13,B1),
        semantics(det_qp,DB,B12,C),
        !, rulematch(B12,B2,B3,C,Rulematch,AS,Packets,Unseen,DB).

/* rule NEXT_WEEK: [ord][noun,time] -> qp node and ord   */

parse_qp_1(10, (ord), (noun & time), t, t, next_week).

  parse_qp_1(next_week,B1,B2,B3,C,AS,[APacks|Packets],Unseen,DB) :-
        new_node(qp,B11),
        attach(B1,B11,ord,B12),
        semantics(ordinal,DB,B1,C),
        deactivate(parse_qp_1,APacks,P1),
        activate(parse_noun,P1,P2),
        !, rulematch(B12,B2,B3,C,Rulematch,AS,[P2|Packets],Unseen,DB).

/** rule ALL_THE: [all][det,def] -> insert "of" into B2 */

parse_qp_1(10, (all), (det & def), t, t, all_the).

  parse_qp_1(all_the,B1,B2,B3,C,AS,Packets,Unseen,DB) :-
        lookup(of,U2),
        !, rulematch(B1,U2,B2,C,Rulematch,AS,Packets,[B3|Unseen],DB).
```

```
/** rule QUANTIFIER: [quantifier] ->attach and transfer feats   */

parse_qp_1(10, (quantifier), t, t, t, quantifier).

  parse_qp_1(quantifier,B1,B2,B3,[C|TL],AS,[APacks|Packets],[U1|Unseen],DB) :-
        attach(B1,C,quantifier,C1),
        semantics(quantifier,DB,B1,C),
        (B2 has prep, !, (deactivate(parse_qp_1,APacks,P1),
                activate(np_complete,P1,P2)); P2=APacks),
        !, rulematch(B2,B3,U1,[C1|TL],Rulematch,AS,[P2|Packets],Unseen,DB).

/** rule QUANT_DONE: [t] -> change packets      */

parse_qp_1(15, t, t, t, t, quant_done).

  parse_qp_1(quant_done,B1,B2,B3,C,AS,[APacks|Packets],Unseen,DB):-
        deactivate(parse_qp_1,APacks,P1),
        activate(parse_adj,P1,P2),
        !, rulematch(B1,B2,B3,C,Rulematch,AS,[P2|Packets],Unseen,DB).



\\\\\
```

```
SUBFILE: PQP2.PK @15:54 15-SEP-1981 <005> (261)
/* PQP2.PK : Packet      PARSE_QP_2
                assumes C is a NP after a det is found

                                                    Rob
                                                    Updated: 10 March 81 (R)
*/



  :- mode parse_qp_2(+,-,-,-,-,?).
  :- mode parse_qp_2(+,+,+,+,+,+,+,+,?).

/*-------------------------------------------------------------------------*/

/** rule DET_QUANT: [quant,det or num] -> new qp node    */

  rse_qp_2(10, (quant), (adj # noun), t, t, det_quant).

  parse_qp_2(det_quant,B1,B2,B3,C,AS,Packets,Unseen,DB):-
        new_node(qp,B11),
        attach(B1,B11,quant,B12),
        semantics(det_qp,DB,B12,C),
        !, rulematch(B12,B2,B3,C,det_quant_done,AS,Packets,Unseen,DB).

/* rule ORDINAL: [ord] -> new ordqp node, ect    */

parse_qp_2(10, (ord), t, t, t, ordinal).

  parse_qp_2(ordinal,B1,B2,B3,C,AS,[APacks|Packets],[U1|Unseen],DB):-
        deactivate(parse_qp_2,APacks,P1),
        activate(parse_adj,P1,P2),
        new_node(ap,B11),
        attach(B1,B11,ord,B12),
        semantics(ordinal,DB,B1,C),
        !, rulematch(B12,B2,B3,C,Rulematch,AS,[P2|Packets],
                                                [U1|Unseen],DB).

/** rule DET_QUANT_DONE: [t] -> redo packets.    */

parse_qp_2(15, t, t, t, t, det_quant_done).

  parse_qp_2(det_quant_done,B1,B2,B3,C,AS,[APacks|Packets],Unseen,DB):-
        deactivate(parse_qp_2,APacks,P2),
        activate(parse_adj,P2,P3),
        !, rulematch(B1,B2,B3,C,Rulematch,AS,[P3|Packets],Unseen,DB).



\\\\\
```

```
SUBFILE: PADJ.PK @21:21 10-APR-1981 <005> (250)
/* PADJ.PK : Packet      PARSE_ADJ
                assumes C is an NP needing adj's

                                                    Rob
                                                    Updated: 7 November 80 (L)
*/



  :- mode parse_adj(+,-,-,-,-,?).
  :- mode parse_adj(+,+,+,+,+,+,+,+,?).

/*-------------------------------------------------------------------------*/

/* rule ADJ_GROUP [adj][adj # noun # dim] -> attach as adj       */

   se_adj(10, (adj), (adj # noun # dim), t, t, adj).

  parse_adj(adj,B1,B2,B3,[C|TL],AS,Packets,[U1|Unseen],DB) :-
        attach(B1,C,adj,C1),
        semantics(adj,DB,B1,C),
        !, rulematch(B2,B3,U1,[C1|TL],Rulematch,AS,Packets,Unseen,DB).

/* rule ADJ_NP: [adj] -> attach as adj  */

parse_adj(10, (adj & not(noun)), t, t, t, adj_np).

 parse_adj(adj_np,B1,B2,B3,[C|TL],AS,[APacks|Packets],[U1|Unseen],DB) :-
        attach(B1,C,adj,C1),
        addfeats(C1,ap,C2),
        deactivate(parse_adj,APacks,P1),
        activate(np_complete,P1,P2),
        semantics(adj_np,DB,B1,C),
        !, rulematch(B2,B3,U1,[C2|TL],np_done,AS,[P2|Packets],Unseen,DB).

  * rule ADJ_DONE: [t] -> change packets        */

parse_adj(15, t, t, t, t, adj_done).

  parse_adj(adj_done,B1,B2,B3,C,AS,[APacks|Packets],Unseen,DB):-
        deactivate(parse_adj,APacks,P1),
        activate(parse_noun,P1,P2),
        !, rulematch(B1,B2,B3,C,Rulematch,AS,[P2|Packets],Unseen,DB).



\\\\\
```

```
SUBFILE: PNOUN.PK @13:25 3-JUN-1981 <005> (341)
/* PNOUN.PK : Packet     PARSE_NOUN
                assumes C is an NP needing a noun

                                                    Rob
                                                    Updated: 16 December 80 (R)
*/


  :- mode parse_noun(+,-,-,-,-,?).
  :- mode parse_noun(+,+,+,+,+,+,+,+,?).

/*-----------------------------------------------------------------------*/


/* rule COMPLEX_NOUN: [noun][noun] -> attach 1st to c    */

parse_noun(10, (noun & not(np1)), (noun), t, agree_23(complex_noun), complex_noun

  parse_noun(complex_noun,B1,B2,B3,[C|TL],AS,Packets,[U1|Unseen],DB) :-
        attach(B1,C,nounns,C1,DB),
        semantics(complex_noun,DB,B1,C),
        !, rulematch(B2,B3,U1,[C1|TL],Rulematch,AS,Packets,Unseen,DB).

/* rule NOUNS: [noun, np1] -> attach to c if past test   */

parse_noun(10, (noun & np1), t, t, sem_chk(nouns), nouns).

  parse_noun(nouns,B1,B2,B3,[C|TL],AS,[APacks|Packets],[U1|Unseen],DB) :-
        deactivate(parse_noun,APacks,P1),
        activate(np_complete,P1,P2),
        attach(B1,C,noun,C1),
        semantics(nouns,DB,B1,C),
        !, rulematch(B2,B3,U1,[C1|TL],Rulematch,AS,[P2|Packets],Unseen,DB).

  * rule NOUN: [noun] -> attach to c as noun     */

parse_noun(10, (noun & not(np1)), t, t, t, noun).

  parse_noun(noun,B1,B2,B3,[C|TL],AS,[APacks|Packets],[U1|Unseen],DB):-
        attach(B1,C,noun,C1),
        deactivate(parse_noun,APacks,P1),
        activate(np_complete,P1,P2),
        semantics(noun,DB,B1,C),
        !, rulematch(B2,B3,U1,[C1|TL],Rulematch,AS,[P2|Packets],
              Unseen,DB).

/** rule NP_BUILT: [t] ->change packet to NP_complete    */

parse_noun(15, t, t, t, t, np_built).

  parse_noun(np_built,B1,B2,B3,C,AS,[APacks|Packets],Unseen,DB) :-
        deactivate(parse_noun,APacks,P1),
        activate(np_complete,P1,P2),
        !, rulematch(B1,B2,B3,C,Rulematch,AS,[P2|Packets],Unseen,DB).
```

\\\\\

```
SUBFILE: NPCOM.PK @15:55 15-SEP-1981 <005> (997)
/* NPCOM.PK : Packet    NP_COMPLETE
                assumes C is an NP with headnouns already

                                                    Rob
                                                    Updated: 23 April 81
*/



  :- mode np_complete(+,-,-,-,-,?).
  :- mode np_complete(+,+,+,+,+,+,+,+,+,?).

/*-------------------------------------------------------------------------*/

/* rule QP_PP: [qp][prep] -> start a pp */

  _complete(10, (qp), (prep), t, t, qp_pp).

  np_complete(qp_pp,B1,B2,B3,C,AS,Packets,[U1|Unseen],DB) :-
        new_node(pp,B11),
        semantics(prep,DB,B11),
        attach(B1,B11,qp,B12),
        semantics(qp_pp,DB,B1,B11),
        !, rulematch(B2,B3,U1,[B12|C],Rulematch,AS,
                                [[parse_pp,cpool]|Packets],Unseen,DB).

/** rule PP: [prep][nsstart]-> B1 <- PP,  attach 2nd to c as prep
                attach 3rd to c as np   cf left out     */

np_complete(10, (prep), (nsstart), t, t, pp).

  np_complete(pp,B1,B2,B3,C,AS,Packets,Unseen,DB) :-
        new_node(pp,B11),
        semantics(prep,DB,B11),
        !, rulematch(B1,B2,B3,[B11|C],Rulematch,AS,
                                [[parse_pp,cpool]|Packets],Unseen,DB).

/* rule REDUCED_RELATIVE: c is np, [verb,ing ] -> insert wh_ into B1    */
/* rule needs more. this is a garden path and semantics should decide */

np_complete(10, (verb & ing ), t, t, t, reduced_relative).

np_complete(10, (verb & en), t, t, sem_chk(red_rel), reduced_relative).

  np_complete(reduced_relative,B1,B2,B3,C,[[aux,B]|AS],Packets,Unseen,DB) :-
        !, rulematch(B1,B2,B3,C,np_done,[[aux,B]|AS],Packets,Unseen,DB).

  np_complete(reduced_relative,B1,B2,B3,C,AS,Packets,Unseen,DB) :-
        semantics(reduced_rel,DB,B1,C),
        lookup(wh_,U1),
        !, rulematch(U1,B1,B2,C,Rulematch,AS,Packets,[B3|Unseen],DB).

/** rule REL_ATTACH: [relative] -> attach to c,          */

np_complete(10, (relative), t, t, t, rel_attach).

  np_complete(rel_attach,B1,B2,B3,[C|TL],AS,Packets,[U1|Unseen],DB) :-
```

```
        semantics(rel_attach,DB,B1,C),
        attach(B1,C,relative,C2),
        !, rulematch(B2,B3,U1,[C2|TL],Rulematch,AS,Packets,Unseen,DB).


/* rule RELPRON_NP: [relpron] -> np in B1        */

np_complete(10, (relpron & wh), t, t, t, relpron_np).
np_complete(10, (that), (verb), t, t, relpron_np).

  np_complete(relpron_np,B1,B2,B3,C,AS,Packets,Unseen,DB) :-
        new_node(np,[relpron_np],C1),
        semantics(start_np,DB,C1),
        semantics(relpron_np,DB,B1,C1),
        attach(B1,C1,wh_comp,C2),
        !, rulematch(C2,B2,B3,C,Rulematch,AS,Packets,Unseen,DB).

/* rule WH_RELATIVE_CLAUSE: [relpron_np] ->
        embedded sentence,attach 1st as whcomp,trace if 2nd verb        */
        /* should not be labeled comp_s, but done to fit other stuff */

np_complete(10, (relpron_np), t, t, t, wh_relative_clause).

  np_complete(wh_relative_clause,B1,B2,B3,C,AS,Packets,[U1|Unseen],DBold) :-
        new_node(s,[sec,comp_s,relative],S1),
        push_sent(S1,DBold,DB),
        semantics(wh_relative_clause,DB,S1),
        attach(B1,S1,wh_comp,S2),
        (B2 has verb, !, (new_node(np,[trace],B11),
        /* set bindings to wh_comp        */
                semantics(start_np,DB,B11),
                semantics(trace,DB,B11),
        !, rulematch(B11,B2,B3,[S2|C],Rulematch,AS,
                        [[cpool,parse_subj]|Packets],[U1|Unseen],DB));
        ( semantics(trace,DB,B1),
        !, rulematch(B2,B3,U1,[S2|C],Rulematch,
        % push wh_comp onto AS stack
        [[wh_comp,B1]|AS],[[cpool,parse_subj]|Packets],Unseen,DB))).

/* rule NP_PP [pp] -> consider to attach the pp to the np        */

np_complete(10, (pp), t, t, sem_chk(pp), np_pp).

  np_complete(np_pp,B1,B2,B3,[C|TL],AS,Packets,[U1|Unseen],DB) :-
        attach(B1,C,pp,C11,DB),
        semantics(np_pp,DB,B1,C),
        !, rulematch(B2,B3,U1,[C11|TL],Rulematch,AS,Packets,Unseen,DB).

/* rule AND: [conj] -> stuff onto active stack  */

np_complete(10, (conj & not(andc)), t, t, t, and).

  np_complete(and,B1,B2,B3,C,AS,Packets,[U1|Unseen],DB) :-
        addfeats(B1,andc,B11),
        !, rulematch(B2,B3,U1,[B11|C],Rulematch,
                        AS,[[cpool,parse_vp,parse_conj]|Packets],Unseen,DB).

/* rule COMMA: [comma] -> run np_done next, for now     */

np_complete(10, (comma), t, t, t, comma).
```

```
  np_complete(comma,B1,B2,B3,C,AS,Packets,[U1|Unseen],DB) :-
        semantics(comma,DB),
        (B2 has conj # binder, !,
        ( !, rulematch(B1,B2,B3,C,np_done,AS,Packets,[U1|Unseen],DB) );
        ( !, rulematch(B2,B3,U1,C,np_done,AS,Packets,Unseen,DB) )).

/* [nstart] -> insert wh-
        for the boy tom saw
        the boy the girl saw, etc       */

/* rule TOM_MARY: [nsstart] -> insert wh_        */

np_complete(15, (det & not(that)), t, t, t, tom_mary).

  np_complete(tom_mary,B1,B2,B3,C,AS,Packets,Unseen,DB) :-
        lookup(wh_,U1),
        semantics(tom_mary,DB),
        !, rulematch(U1,B1,B2,C,Rulematch,AS,Packets,[B3|Unseen],DB).


/** rule NP_DONE: [t] -> drop c.        */

np_complete(15, t, t, t, t, np_done).

  np_complete(np_done,B1,B2,B3,[C|TL],AS,[APacks|Packets],Unseen,DB) :-
        semantics(np_done,DB,C),
        !, rulematch(C,B1,B2,TL,Rulematch,AS,Packets,[B3|Unseen],DB).

/* rule OF_PP: [of][noun] -> add nsstart to second to force PP */

np_complete(15, (of), (noun), t, t, of_pp).

  np_complete(of_pp,B1,B2,B3,C,AS,Packets,Unseen,DB) :-
        addfeats(B2,nsstart,B22),
        !, rulematch(B1,B22,B3,C,Rulematch,AS,Packets,Unseen,DB).



\\\\\
```

```
SUBFILE: PARPP.PK @20:52 2-MAR-1981 <005> (245)
/* PARPP.PK : Packet    PARSE_PP
              assumes C is partial PP

                                                    Rob
                                                    Updated: 1 March 81 (R)
*/


        :- mode parse_pp(+,-,-,-,-,?).
        :- mode parse_pp(+,+,+,+,+,+,+,+,?).

/*-------------------------------------------------------------------------*/

/* rule ATTACH_PREP: [prep] -> attach to C      */

parse_pp(10, (prep), t, t, t, attach_prep).

  parse_pp(attach_prep,B1,B2,B3,[C|TL],AS,Packets,[U1|Unseen],DB) :-
        attach(B1,C,prep,C1),
        semantics(attach_prep,DB,B1,C),
        !, rulematch(B2,B3,U1,[C1|TL],Rulematch,AS,Packets,Unseen,DB).

/* rule PP_NP: [np] -> attach to PP, drop PP    */

parse_pp(10, (np), t, t, t, pp_np).

  parse_pp(pp_np,B1,B2,B3,[C|TL],AS,[Apacks|Packets],Unseen,DB) :-
        attach(B1,C,np,C1),
        semantics(pp_sets_np,DB,C,B1),
        !, rulematch(C1,B2,B3,TL,Rulematch,AS,Packets,Unseen,DB).

/* rule WITH_WHICH: [wh] -> wh_np built, temp patch, not thousht out    */

parse_pp(15, (wh), t, t, t, with_which).

  parse_pp(with_which,B1,B2,B3,[C|TL],AS,Packets,Unseen,DB) :-
        new_node(np,B11),
        semantics(start_np,DB,B11),
        attach(B1,B11,wh_comp,B12),
        addfeats(C,relpron_np,C1),
        semantics(relpron_np,DB,B1,B11),
        !, rulematch(B12,B2,B3,[C1|TL],Rulematch,AS,Packets,Unseen,DB).



\\\\\
```

```
SUBFILE: PSUBJ.PK @19:31 10-MAR-1981 <005> (208)
/* PSUBJ.PK : Packet    PARSE_SUBJ
                assumes C is the S, needing a subj

                                                    Rob
                                                    Updated: 7 November 80 (L)
*/



  :- mode parse_subj(+,-,-,-,-,?).
  :- mode parse_subj(+,+,+,+,+,+,+,+,?).

/*-------------------------------------------------------------------------*/

/** rule UNMARKED_ORDER: [np][verb] -> attach 1st to s  */

 :rse_subj(10, (np), (verb), t, agree(subj_verb), unmarked_order).

  parse_subj(unmarked_order,B1,B2,B3,[C|TL],AS,[APacks|Packets],
                                                  [U1|Unseen],DB):-
        semantics(syn_subj,DB,B1),
        attach(B1,C,np,C1),
        deactivate(parse_subj,APacks,P2),
        activate(parse_aux,P2,P3),
        !, rulematch(B2,B3,U1,[C1|TL],Rulematch,AS,[P3|Packets],Unseen,DB).


/* rule AUX_INVERSION: [aux][np] -> attach B2   */

parse_subj(10, (auxverb), (np # ngstart), t, t, aux_inversion).

  parse_subj(aux_inversion,B1,B2,B3,C,AS,Packets,[U1|Unseen],DB) :-
        !, rulematch(B2,B3,U1,C,Rulematch,[[aux,B1]|AS],Packets,Unseen,DB).

                                                % relax agreement

parse_subj(15, (np), (verb), t, t, unmarked_order).



\\\\\
```

```
SUBFILE: BLDAUX.PK @16:56 2-JUN-1981 <005> (797)
/* BLDAUX.PK : Packet   BUILD_AUX
               assumes C is a partial AUX

                                                    Rob
                                                    Updated: 27 May 80 (R)
*/


  :- mode build_aux(+,-,-,-,-,?).
  :- mode build_aux(+,+,+,+,+,+,+,+,?).

/*-------------------------------------------------------------------------*/

/* insert can fish stuff here   */
/**  rule NEG: [neg][verb]  -> attach 1st to c as neg    */

build_aux(10, (not(be_)), (neg), t, t, neg).

   build_aux(neg,B1,B2,B3,[C|TL],AS,Packets,[U1|Unseen],DB) :-
        semantics(simple_neg,DB),
        attach(B2,C,neg,C2),
        !, rulematch(B1,B3,U1,[C2|TL],Rulematch,AS,Packets,Unseen,DB).

/** rule MODAL: [modal][tnsless] -> attach 1st to c as modal    */

build_aux(10, (modal), (tnsless), t, t, modal).

  build_aux(modal,B1,B2,B3,[C|TL],AS,Packets,[U1|Unseen],DB) :-
        attach(B1,C,modal,C2),
        !, rulematch(B2,B3,U1,[C2|TL],Rulematch,AS,Packets,Unseen,DB).

/** rule PERFECTIVE: [have][en] -> attach 1st to c as perf      */

build_aux(10, (have), (en), t, t, perfective).

  build_aux(perfective,B1,B2,B3,[C|TL],AS,Packets,[U1|Unseen],DB) :-
        attach(B1,C,auxverb,C2),                  % was perf
        !, rulematch(B2,B3,U1,[C2|TL],Rulematch,AS,Packets,Unseen,DB).

/** rule PASSIVE_AUX: [be][en] -> attach 1st to c as passive,
                label 2nd passive       */

build_aux(10, (be), (en), t, t, passive_aux).

  build_aux(passive_aux,B1,B2,B3,[C|TL],AS,Packets,[U1|Unseen],DB):-
        attach(B1,C,passive,C1),
        semantics(passive_aux,DB),
        addfeats(C1,passive,C2),
        !, rulematch(B2,B3,U1,[C2|TL],Rulematch,AS,Packets,Unseen,DB).

/** rule PROGRESSIVE: [be][ing] -> attach 1st as pros   */

build_aux(10, (be), (ing), t, t, progressive).

  build_aux(progressive,B1,B2,B3,[C|TL],AS,Packets,[U1|Unseen],DB) :-
        attach(B1,C,auxverb,C2),                  % was pros
```

```
        !, rulematch(B2,B3,U1,[C2|TL],Rulematch,AS,Packets,Unseen,DB).

/** rule DO_SUPPORT: [do][tnsless] -> attach 1st as do  */

build_aux(10, (do), (tnsless), t, t, do_support).

  build_aux(do_support,B1,B2,B3,[C|TL],AS,Packets,[U1|Unseen],DB) :-
        attach(B1,C,do,C1),
        semantics(do_support,DB,B1,C),
        !, rulematch(B2,B3,U1,[C1|TL],Rulematch,AS,Packets,Unseen,DB).

/* rule HAVE_TO: [have or be][to] -> attach 1st */

build_aux(10, (have#be), (to), t, t, have_to).

  build_aux(have_to,B1,B2,B3,[C|TL],AS,Packets,[U1|Unseen],DB) :-
        attach(B1,C,auxverb,C1),
        semantics(do_support,DB,B1,C),
        !, rulematch(B2,B3,U1,[C1|TL],Rulematch,AS,Packets,Unseen,DB).

/* rule TO_BE: [to][tnsless] -> attach to       */

build_aux(10, (to), (tnsless), t, t, to_be).

  build_aux(to_be,B1,B2,B3,[C|TL],AS,Packets,[U1|Unseen],DB) :-
        attach(B1,C,auxverb,C1),
        semantics(do_support,DB,B1,C),
        !, rulematch(B2,B3,U1,[C1|TL],Rulematch,AS,Packets,Unseen,DB).

/**  rule ADVERB: [adverb][verb] -> attach 1st to c as adverb   */
                % added by Karen Archbold April 1981

build_aux(10,((auxverb & not(be_)) * adverb),(adverb),t,t,adverb).

build_aux(adverb,B1,B2,B3,[C|TL],AS,Packets,[U1|Unseen],DB) :-
        attach(B2,C,adverb,C2),
        semantics(adverb,DB,B2),
        !, rulematch(B1,B3,U1,[C2|TL],Rulematch,AS,Packets,Unseen,DB).


/* rule AUX_ADVERB: [adverb] -> attach to aux   */

build_aux(10, (adverb), t, t, t, aux_adverb).

  build_aux(aux_adverb,B1,B2,B3,[C|TL],AS,Packets,[U1|Unseen],DB):-
        semantics(aux_adverb,DB,B1),
        attach(B1,C,adverb,C1),
        !, rulematch(B2,B3,U1,[C1|TL],Rulematch,AS,Packets,Unseen,DB).

/** rule AUX_COMPLETE: [t] -> drop c.   */

build_aux(15, t, t, t, t, aux_complete).

  build_aux(aux_complete,B1,B2,B3,[C|TL],AS,[APacks|Packets],Unseen,DB):-
        (B1 has noun & verb, !, coerce(verb,B1,B11);B11=B1),
        !, rulematch(C,B11,B2,TL,Rulematch,AS,Packets,[B3|Unseen],DB).

/** rule BE_PRED: [be][prep or adj] -> attach as copula */

build_aux(10, (be), (prep # adj), t, t, be_pred).
```

```
build_aux(be_pred,B1,B2,B3,[C|TL],AS,Packets,[U1|Unseen],DB):-
        semantics(be_pred,DB,B1,C),
        attach(B1,C,auxverb,C2),                    % was copula
        !, rulematch(B2,B3,U1,[C2|TL],Rulematch,AS,Packets,Unseen,DB).
```

\\\\\

```
SUBFILE: PAUX.PK @20:2 22-MAY-1981 <005> (329)
/* PAUX.PK : Packet      PARSE_AUX
                assumes C is the S, needing an aux

                                                    Rob
                                                    Updated: 7 November 80 (L)
*/



  :- mode parse_aux(+,-,-,-,-,?).
  :- mode parse_aux(+,+,+,+,+,+,+,+,?).

/*-----------------------------------------------------------------------*/

/** rule TO_INFINITIVE: [to,auxverb][tnsless] -> new aux node.  */

  rse_aux(10, (to), (tnsless), t, t, to_infinitive).

  parse_aux(to_infinitive,B1,B2,B3,C,AS,Packets,[U1|Unseen],DB):-
        semantics(to_infinitive,DB,B2,C),
        new_node(aux,C1),                  % did have [inf]
        attach(B1,C1,to,C2),
        !, rulematch(B2,B3,U1,[C2|C],Rulematch,AS,[[build_aux]|Packets],
                                                        Unseen,DB).

/** rule START_AUX: [verb] ->create new AUX node, etc   */
                % percolate sets tense

parse_aux(10, (verb # adverb), t, t, t, start_aux).

  parse_aux(start_aux,B1,B2,B3,C,[[aux,B11]|AS],Packets,Unseen,DB) :-
        semantics(start_aux,DB,B11,C),
        new_node(aux,C1),
        percolate(B11,C1,C2),
        !, rulematch(B11,B1,B2,[C2|C],Rulematch,AS,[[build_aux]|Packets],
                [B3|Unseen],DB).

  parse_aux(start_aux,B1,B2,B3,C,AS,Packets,Unseen,DB):-
        semantics(start_aux,DB,B1,C),
        new_node(aux,NewC),
        percolate(B1,NewC,C2),
        !, rulematch(B1,B2,B3,[C2|C],Rulematch,AS,[[build_aux]|Packets],
                Unseen,DB).

/** rule AUX_ATTACH: [aux] -> attach to s, change packets.      */

parse_aux(10, (aux), t, t, t, aux_attach).

  parse_aux(aux_attach,B1,B2,B3,[C|TL],AS,[APacks|Packets],[U1|Unseen],DB):-
        semantics(aux_attach,DB,B1,C),
        deactivate(parse_aux,APacks,P1),
        activate(parse_vp,P1,P2),
        attach(B1,C,aux,C1),
        !, rulematch(B2,B3,U1,[C1|TL],Rulematch,AS,[P2|Packets],Unseen,DB).
```

```
SUBFILE: PVP.PK @13:46 10-APR-1981 <005> (276)
/* PVP.PK : Packet       PARSE_VP
                assumes C is s, needing a vp

                                                        Rob
                                                        Updated:  23 April 81 (R)
*/



  :- mode parse_vp(+,-,-,-,-,?).
  :- mode parse_vp(+,+,+,+,+,+,+,+,+,?).

/*-------------------------------------------------------------------------*/


/** rule PREDP [pp] -> attach
  s predp, change packets        */

parse_vp(10, (pp # ap), t, t, t, predp).

  parse_vp(predp,B1,B2,B3,[C|TL],AS,[APacks|Packets],[U1|Unseen],DB):-
        deactivate(parse_vp,APacks,P1),
        addfeats(B1,predp,B11),
        semantics(predp,DB,B1),
        attach(B11,C,predp,C2),
        (C2 has major, !, activate(ss_final,P1,P2);
        (C2 has sec, !, activate(embedded_s_final, P1, P2);
        P2 = P1)),
        !, rulematch(B2,B3,U1,[C2|TL],Rulematch,AS,[P2|Packets],Unseen,DB).

/* rule MAIN_VERB: [verb] -> do everything.      */

parse_vp(10, (verb), t, t, t, main_verb).
        % agree with subject
  parse_vp(main_verb,B1,B2,B3,[C|TL],AS,[APacks|Packets],[U1|Unseen],DB):-
        deactivate(parse_vp,APacks,P1),
        (C has major, !, activate(ss_final,P1,P2);
        (C has sec, !, activate(embedded_s_final,P1,P2);
                P1=P2)),
        new_node(vp,C1),
        attach(B1,C1,verb,C2),
        semantics(main_verb,DB,B1),
        verb_types(B1,NewPackets),
        activate(ss_vp,NewPackets,P3),
        (B1 has two_obj, !, P3=P4 ; activate(object,P3,P4) ),
        activate(cpool,P4,P5),
        !, rulematch(B2,B3,U1,[C2,C|TL],Rulematch,AS,
                [P5,P2|Packets],Unseen,DB).



\\\\\
```

```
SUBFILE: PASSIV.PK @15:4 13-NOV-1980 <005> (139)
/* PASSIV.PK : Packet   PASSIVE
               assumes S is a VP, doesn't use it, always told to run

                                                    Rob
                                                    Updated: 7 November 80 (L)
*/



  :- mode passive(+,-,-,-,-,?).
  :- mode passive(+,+,+,+,+,+,+,+,+,?).

/*-------------------------------------------------------------------------*/

/* trace bound to syn_subj, or no_subj  */

  /* rule PASSIVE: [t] -> create trace   */

passive(S, t, t, t, t, passive).

  passive(passive,B1,B2,B3,[C|TL],AS,[APacks|Packets],Unseen,DB):-
        new_node(np,[trace],B11),
        semantics(start_np,DB,B11),
        semantics(passive,DB,B11),
        deactivate(passive,APacks,P1),
        !, rulematch(B11,B1,B2,[C|TL],Rulematch,AS,[P1|Packets],
                [B3|Unseen],DB).




\\\\\
```

```
SUBFILE: SSVP.PK @12:36 28-MAY-1981 <005> (485)
/* SSVP.PK : Packet      SS_VP
                assumes C is a VP, major

                                                    Rob
                                                    Updated: 4 June 81 (R)
*/


  :- mode ss_vp(+,-,-,-,-,?).
  :- mode ss_vp(+,+,+,+,+,+,+,+,?).

/*-------------------------------------------------------------------------*/

/* rule ADVERB_GROUP: [adverb][adverb] -> compound adverb        */

  _vp(5, (adverb), (adverb), t, t, adverb_group).

  ss_vp(adverb_group,B1,B2,B3,C,AS,Packets,[U1|Unseen],DB) :-
        new_node(adverb,A1),
        attach(B1,A1,adverb,A2),
        attach(B2,A2,adverb,A3),
        semantics(adverb_group,DB,A1,B1,B2),
        !, rulematch(A3,B3,U1,C,Rulematch,AS,Packets,Unseen,DB).

/* rule ADVERB: [adverb] -> attach as adverb     */
        /* For karen Archbold, add the patters:
                (adverb # prep# func) to the second buffer
                I will use the looser form       */

ss_vp(10, (adverb), t, t, t, adverb).

  ss_vp(adverb,B1,B2,B3,[CC|TL],AS,Packets,[U1|Unseen],DB):-
        attach(B1,C,adverb,C1),
        semantics(adverb,DB,B1),
        !, rulematch(B2,B3,U1,[CC1|TL],Rulematch,AS,Packets,Unseen,DB).



/* rule PART: [particle] -> attach to verb       */

ss_vp(5, (prep), t, t, sem_chk(particle), part).

  ss_vp(part,B1,B2,B3,[CC|TL],AS,Packets,[U1|Unseen],DB) :-
        attach(B1,C,part,C1),
        semantics(part,DB,B1),
        !, rulematch(B2,B3,U1,[CC1|TL],Rulematch,AS,Packets,Unseen,DB).

/** rule PP_UNDER_VP_1: [pp] -> attach to c
automatocally attaches to the vp, rule in cpool decides for the np
semantics: checks the can have, if true then it attaches to the np,
        else it attaches to the vp.*/

ss_vp(10, (pp), t, t, t, pp_under_vp_1).

  ss_vp(pp_under_vp_1,B1,B2,B3,[CC|TL],AS,Packets,[U1|Unseen],DB):-
        attach(B1,C,pp,C1),
```

```
        semantics(pp_under_x,DB,B1),
        !, rulematch(B2,B3,U1,[C1|TL],Rulematch,AS,Packets,Unseen,DB).

/* rule PART: [particle] -> attach to verb       */

ss_vp(15, (prep), t, t, t, part).

  ss_vp(part,B1,B2,B3,[C|TL],AS,Packets,[U1|Unseen],DB) :-
        attach(B1,C,part,C1),
        semantics(part,DB,B1),
        !, rulematch(B2,B3,U1,[C1|TL],Rulematch,AS,Packets,Unseen,DB).

/** rule VP_DONE: [t] -> drop c.          */

ss_vp(15, t, t, t, t, vp_done).


  ss_vp(vp_done,B1,B2,B3,[C|TL],[[wh_comp,B11]|AS],
                                [APacks|Packets],Unseen,DB):-
        attach(B11,C,trace,C1),
        semantics(drop_vp_trace,DB,B11),
        !, rulematch(C1,B1,B2,TL,Rulematch,AS,Packets,[B3|Unseen],DB).

  ss_vp(vp_done,B1,B2,B3,[C|TL],AS,[APacks|Packets],Unseen,DB):-
        !, rulematch(C,B1,B2,TL,Rulematch,AS,Packets,[B3|Unseen],DB).



\\\\\
```

```
SUBFILE: OBJ.PK @20:47 10-APR-1981 <005> (120)
/* OBJ.PK : Packet       OBJECT
                assumes C is a VP, needs one object

                                                        Rob
                                                        Updated: 23 April 81 (R)
*/


  :- mode object(+,-,-,-,-,?).
  :- mode object(+,+,+,+,+,+,+,+,+,?).

/*-------------------------------------------------------------------------------*/

/* rule OBJECT: [to,auxverb][tnsless] -> attach object  */

  ject(10, (np), t, t, t, object).

  object(object,B1,B2,B3,[C|TL],AS,[APacks|Packets],[U1|Unseen],DB) :-
        attach(B1,C,np,C1),
        semantics(syn_obj,DB,B1),
        deactivate(object,APacks,P1),
        !, rulematch(B2,B3,U1,[C1|TL],Rulematch,AS,[P1|Packets],Unseen,DB).



\\\\\
```

```
SUBFILE: NOSUBJ.PK @16:36 4-MAR-1981 <005> (139)
/* NOSUBJ.PK : Packet   NO_SUBJ
               assumes C is a CP, verb is want

                                                    Rob
                                                    Updated: 6 March 81 (R)
*/


  :- mode no_subj(+,-,-,-,-,?),
  :- mode no_subj(+,+,+,+,+,+,+,+,?),

/*-------------------------------------------------------------------------*/

/* rule CREATE_DELTA_SUBJECT: [to,auxverb][tnsless] ->
                          trace into B1   */

no_subj(10, (to), (tnsless), t, t, create_delta_subj),

  no_subj(create_delta_subj,B1,B2,B3,C,AS,[APacks|Packets],
                                                    Unseen,DB) :-
        new_node(np,[trace],B11),
        semantics(start_np,DB,B11),
        semantics(create_delta_subj,DB,B11),
        deactivate(no_subj,APacks,P1),
        !, rulematch(B11,B1,B2,C,Rulematch,AS,[CP1|Packets],[DB3|Unseen],DB).



\\\\\
```

```
SUBFILE: THATC.PK @21:25 29-MAR-1981 <005> (209)
/* THATC.PK : Packet    THAT_COMP
                assumes C is a VP, verb is that_comp

                                                    Rob
                                                    Updated: 16 January 81
*/


  :- mode that_comp(+,-,-,-,-,?).
  :- mode that_comp(+,+,+,+,+,+,+,+,?).

/*-------------------------------------------------------------------------*/

/* rule THAT_S_START_1: [np][verb] -> embedded sentence */

  at_comp(5, (np), (verb), t, t, that_s_start_1).

  that_comp(that_s_start_1,B1,B2,B3,C,AS,Packets,[U1|Unseen],DBold) :-
        new_node(s,[sec,comp_s],C1),
        push_sent(C1,DBold,DB),
        semantics(that_s_start_1,DB,B1,C1),
        attach(B1,C1,np,C2),
        !, rulematch(B2,B3,U1,[C2|C1],Rulematch,AS,[[cpool,parse_aux]|
                Packets],Unseen,DB).

/* rule THAT_S: [that] -> start an S bar          */

that_comp(10, (that), (nsstart), t, t, that_s).

  that_comp(that_s,B1,B2,B3,C,AS,Packets,[U1|Unseen],DBold) :-
        new_node(s,[sec,s,comp_s],C1),
        attach(B1,C1,comp,C2),
        push_sent(C2,DBold,DB),
        semantics(that_s_start,DB,C1),
        !, rulematch(B2,B3,U1,[C2|C1],Rulematch,AS,
                [[cpool,parse_subj]|Packets],Unseen,DB).



\\\\\
```

```
SUBFILE: INFC.PK @21:2 6-APR-1981 <005> (229)
/* INFC.PK : Packet      INF_COMP
                assumes C is a VP, verb is inf_comp

                                                        Rob
                                                        Updated: 7 November 80 (L)
*/



  :- mode inf_comp(+,-,-,-,-,?).
  :- mode inf_comp(+,+,+,+,+,+,+,+,?).

/*-------------------------------------------------------------------------*/

/* change to VPbar here */

   rule INF_S_START1: [np][to,auxverb][tnsless] ->
                new s node, attach ist to it as np       */
        /* make 3rd a verb ??   */

inf_comp(5, (np), (to), (tnsless), t, inf_s_start1).

  inf_comp(inf_s_start1,B1,B2,B3,C,AS,Packets,[U1|Unseen],DBold) :-
        new_node(s,[sec,comp_s],C1),
        push_sent(C1,DBold,DB),
        semantics(inf_s_start_1,DB,Bi,C1),
        attach(B1,C1,np,C2),
        !, rulematch(B2,B3,U1,[C2|C1],Rulematch,AS,[[cpool,parse_aux]|Packets],
                                                        Unseen,DB).

/* rule FOR_S_BAR: [for,pp][to] -> Sbar */

inf_comp(5, (for & pp), (to), t, t, for_s_bar).

  inf_comp(for_s_bar,B1,B2,B3,C,AS,Packets,[U1|Unseen],DBold) :-
        new_node(s,[sec,comp_s,for],C1),
        push_sent(C1,DBold,DB),
        semantics(inf_s_start_1,DB,Bi,C1),
        attach(B1,C1,np,C2),
        !, rulematch(B2,B3,U1,[C2|C1],Rulematch,AS,[[cpool,parse_aux]|
                                Packets],Unseen,DB).



\\\\\
```

```
SUBFILE: TLICOM.PK @16:21 6-JUN-1981 <005> (159)
/* TLICOM.PK : Packet   TO_LESS_INF_COMP
                assumes C is a VP, verb is see, saw

                                                    Rob
                                                    Updated: 7 November 80 (L)
*/



  :- mode to_less_inf_comp(+,-,-,-,-,?).
  :- mode to_less_inf_comp(+,+,+,+,+,+,+,+,?).

/*-------------------------------------------------------------------------*/

/* This packet does see and saw
        I saw tom.
        I saw tom hit her       */

/* rule UNMARKED_SUBJ: [np][tnsless] ->  for see, embedded sentence */

to_less_inf_comp(5, (np), (tnsless), t, t, unmarked_subj).

  to_less_inf_comp(unmarked_subj,B1,B2,B3,C,AS,Packets,[U1|Unseen],DBold):-
        new_node(s,[sec,comp_s],C1),
        push_sent(C1,DBold,DB),
        semantics(insert_to,DB,C1,B1),
        attach(B1,C1,np,C2),
        !, rulematch(B2,B3,U1,[C2|C],Rulematch,AS,[[cpool,parse_aux]|Packets],
                Unseen,DB).



\\\\\
```

```
SUBFILE: TBLCOM.PK @15:10 17-JAN-1981 <005> (199)
/* TBLCOM.PK : Packet   TO_BE_LESS_INF_COMP
               assumes C is VP, verb is seem

                                                    Rob
                                                    Updated: 7 November 80 (L)
*/


  :- mode to_be_less_inf_comp(+,-,-,-,-,?).
  :- mode to_be_less_inf_comp(+,+,+,+,+,+,+,+,+,?).

/*-----------------------------------------------------------------------*/

/* you seem happy -> you seem to be happy        */

   rule INSERT_TO_BE: [np][en or adj] -> insert to be into the buffer */

to_be_less_inf_comp(10, (np), (en # adj), t, t, insert_to_be).

%  to_be_less_inf_comp(insert_to_be,B1,B2,B3,C,AS,Packets,Unseen,DB):-
%        lookup(to,U2),
%        lookup(be,U3),
%        !, rulematch(B1,U2,U3,C,Rulematch,AS,Packets,[B2,B3|Unseen],DB).

/* rule INSERT_TO_BE_1: [en or adj] -> insert to be into the buffer */

to_be_less_inf_comp(10, (en # adj), t, t, t, insert_to_be).

  to_be_less_inf_comp(insert_to_be,B1,B2,B3,C,AS,Packets,Unseen,DB):-
        lookup(to,U2),
        lookup(be,U3),
        !, rulematch(U2,U3,B1,C,Rulematch,AS,Packets,
                [B2,B3|Unseen],DB).



\\\\\
```

```
SUBFILE: TWOBJ.PK @21:12 10-APR-1981 <005> (131)
/* TWOBJ.PK : Packet    TWO_OBJ
                assumes C is a VP, needs two objects

                                                    Rob
                                                    Updated: 6 March 81 (R)
*/


  :- mode two_obj(+,-,-,-,-,?).
  :- mode two_obj(+,+,+,+,+,+,+,+,+,?).

/*-------------------------------------------------------------------------*/

/* rule FIRST_OBJ: [to,auxverb][tnsless] ->  attach first object */

  o_obj(10, (np), t, t, t, first_object).

  two_obj(first_object,B1,B2,B3,[C|TL],AS,[APacks|Packets],[U1|Unseen],DB) :-
        attach(B1,C,np,C1),
        semantics(syn_obj,DB,B1),
        deactivate(two_obj,APacks,P1),
        activate(object,P1,P2),
        !, rulematch(B2,B3,U1,[C1|TL],Rulematch,AS,[P2|Packets],Unseen,DB).



\\\\\
```

```
SUBFILE: EMBSFI.PK @19:5 4-MAR-1981 <005> (176)
/* EMBSFI.PK : Packet   EMBEDDED_S_FINAL
                assumes C is S, embedded

                                                        Rob
                                                        Updated: 7 March 81 (R)
*/


  :- mode embedded_s_final(+,-,-,-,-,?).
  :- mode embedded_s_final(+,+,+,+,+,+,+,+,+,?).

/*-----------------------------------------------------------------------------*/

/* rule PP_UNDER_S_2: [pp] -> attach     */

  bedded_s_final(10, (pp), t, t, t, pp_under_s_2).

  embedded_s_final(pp_under_s_2,B1,B2,B3,[C|TL],AS,Packets,[U1|Unseen],DB):-
        attach(B1,C,pp,C1),
        semantics(pp_under_x,DB,B1),
        !, rulematch(B2,B3,U1,[C1|TL],Rulematch,AS,Packets,Unseen,DB).

/** rule EMBEDDED_S_DONE: [t] -> drop c.          */

embedded_s_final(15, t, t, t, t, embedded_s_done).

  embedded_s_final(embedded_s_done,B1,B2,B3,[C|TL],AS,[APacks|Packets],
                                                        Unseen,DBold):-
        pop_sent(DBold,DB),
        !, rulematch(C,B1,B2,TL,Rulematch,AS,Packets,[B3|Unseen],DB).



.\\\\
```

```
SUBFILE: BNAME.PK @15:58 15-SEP-1981 <005> (158)
/* BNAME.PK : Packet     BUILD_NAME
                assumes C is NP

                                                        Rob
                                                        Updated: 20 June 1981
*/


  :- mode build_name(+,-,-,-,-,?).
  :- mode build_name(+,+,+,+,+,+,+,+,+,?).

/*-------------------------------------------------------------------------*/

/** rule  NAME: [name] -> attach to c.  */

  ild_name(10, (name), t, t, t, name).

  build_name(name,B1,B2,B3,[C|TL],AS,Packets,[U1|Unseen],DB):-
        attach(B1,C,name,C1),
        semantics(name,DB,B1,C),
        !, rulematch(B2,B3,U1,[C1|TL],Rulematch,AS,Packets,Unseen,DB).

/** rule END_OF_NAME: [t] -> run np_done next   */

build_name(15, t, t, t, t, end_of_name).

  build_name(end_of_name,B1,B2,B3,C,AS,[P1|Packets],Unseen,DB) :-
        activate(np_complete,P1,P2),
        !, rulematch(B1,B2,B3,C,np_done,AS,[P2|Packets],Unseen,DB).



\\\\\
```

```
SUBFILE: PCONJ.PK @17:19 1-MAR-1981 <005> (112)
/* PCONJ.PK : Packet    PARSE_CONJ
                assumes C has and

                                                        Rob
                                                        Updated: 7 November 80 (L)
*/


        :- mode parse_conj(+,-,-,-,-,?).
        :- mode parse_conj(+,+,+,+,+,+,+,+,?).

/*-------------------------------------------------------------------------*/

/* rule DROP_AND: [c has and] -> drop Bi and "and" into buffers */

  rse_conj(5, (vp # pp), t, t, t, drop_and).

parse_conj(15, t, t, t, t, drop_and).

  parse_conj(drop_and,B1,B2,B3,[C|TL],AS,[APacks|Packets],Unseen,DB) :-
        rulematch(C,B1,B2,TL,Rulematch,AS,Packets,[B3|Unseen],DB).



\\\\\
```

```
SUBFILE: SSFIN.PK @15:59 15-SEP-1981 <005> (450)
/* SSFIN.PK : Packet    SS_FINAL
                assumes C is major S

                                                        Rob
                                                        Updated: 6 December 80 (R)
*/


  :- mode ss_final(+,-,-,-,-,?).
  :- mode ss_final(+,+,+,+,+,+,+,+,+,?).

/*-------------------------------------------------------------------------------*/

/** rule PP_UNDER_S_1: [PP] attach to c */

  _final(10, (PP), t, t, t, PP_under_s_1).

  ss_final(PP_under_s_1,B1,B2,B3,[C|TL],AS,Packets,[U1|Unseen],DB):-
        semantics(PP_under_x,DB,B1),
        attach(B1,C,PP,C1),
        !, rulematch(B2,B3,U1,[C1|TL],Rulematch,AS,Packets,Unseen,DB).

/** rule S_DONE: [finalPunc] -> attach and end. */

ss_final(10, (fPunc), t, t, t, s_done).

  ss_final(s_done,B1,B2,B3,[C|TL],AS,Packets,Unseen,DBold):-
        attach(B1,C,fPunc,C1),
        pop_sent(DBold,DB),
        alldone([C1|TL],DB).


/* rule here for what little fish eat, and garden path stuff     */
/* [be], drop and make NP        */

/* rule INIT_S_BAR: [verb] -> drop as a NP       */

ss_final(10, (sent_subj), t, t, t, init_s_bar).

  ss_final(init_s_bar,B1,B2,B3,[C|TL],AS,Packets,Unseen,DBold):-
        addfeats(C,[comp_s],C1),
        semantics(init_s_bar,DBold,C,B1),
        new_node(s,[major],S),
        semantics(start,DBold,S),
        pop_sent(DBold,DB1),
        push_sent(S,DB1,DB),
        !, rulematch(C1,B1,B2,[S|TL],Rulematch,AS,[[cpool,parse_subj]|Packets],
                                                        [B3|Unseen],DB).

/* CONJOINED_S: [comma][conj or binder] -> make into a conjoined S       */

ss_final(10, (comma), (conj # binder), t, t, conjoined_s).

  ss_final(conjoined_s,B1,B2,B3,[C|TL],AS,Packets,[U1,U2|Unseen],DBold) :-
        new_node(s,S1),
        attach(C,S1,s,S2),
```

```
        attach(B2,S2,conj,S3),
        pop_sent(DBold,DB2),
        push_sent(S3,DB2,DB1),
        new_node(s,Snew),
        push_sent(Snew,DB1,DB),
        !, rulematch(B3,U1,U2,[Snew,S3|TL],Rulematch,AS,
                [[cpool,ss_start],Packets],Unseen,DB).

/* HYPO_S: [comma] -> then an if, what sentence is assumed.
                attach the lowest node to the next up node and
                add a new s node and pray.                    */

ss_final(10, (comma), t, t, t, hypo_s).

  ss_final(hypo_s,B1,B2,B3,[S,IFS|TL],AS,[P1,Packets],[U1|Unseen],DBold) :-
        attach(S,IFS,s,S1),
        semantics(hypo_s,DB),
        new_node(s,Snew),
        pop_sent(DBold,DB1),
        push_sent(Snew,DB1,DB),
        !, rulematch(B2,B3,U1,[Snew,S1|TL],Rulematch,AS,
                                [[cpool,ss_start],Packets],Unseen,DB).



\\\\\
```

```
SUBFILE: EMBSVP.PK @19:8 4-MAR-1981 <005> (296)
/* EMBSVP.PK : Packet   EMBEDDED_S_VP
                assumes C is VP, embedded

                                                    Rob
                                                    Updated: 6 March 81 (R)
*/


  :- mode embedded_s_vp(+,-,-,-,-,?).
  :- mode embedded_s_vp(+,+,+,+,+,+,+,+,+,?).

/*-------------------------------------------------------------------------*/

/* rule OBJ_IN_EMBEDDED_S: [np] -> attach to c as np
        has semantic check that is left out      */

  bedded_s_vp(10, (np), t, t, t, obj_in_embedded_s).

  embedded_s_vp(obj_in_embedded_s,B1,B2,B3,[C|TL],AS,Packets,[U1|Unseen],DB):-
        attach(B1,C,np,C1),
        semantics(obj_in_embedded_s,DB,B1),
        !, rulematch(B2,B3,U1,[C1|TL],Rulematch,AS,Packets,Unseen,DB).

/* rule PP_UNDER_VP_2: [pp] -> attach, semantics left out        */

embedded_s_vp(10, (pp), t, t, t, pp_under_vp_2).

  embedded_s_vp(pp_under_vp_2,B1,B2,B3,[C|TL],AS,Packets,[U1|Unseen],DB):-
        attach(B1,C,pp,C1),
        semantics(pp_under_x,DB,B1),
        !, rulematch(B2,B3,U1,[C1|TL],Rulematch,AS,Packets,Unseen,DB).

/* rule EMBEDDED_VP_DONE: [t] -> drop c.         */

embedded_s_vp(15, t, t, t, t, embedded_vp_done).

  embedded_s_vp(embedded_vp_done,B1,B2,B3,[C|TL],[[wh_comp,B11]|AS],
                [APacks|Packets],Unseen,DB) :-
        attach(B11,C,trace,C1),
        semantics(drop_vp_trace,DB,B11),
        !, rulematch(C1,B1,B2,TL,Rulematch,AS,Packets,[B3|Unseen],DB).

  embedded_s_vp(embedded_vp_done,B1,B2,B3,[C|TL],AS,[APacks|Packets],
                                                        Unseen,DB):-
        semantics(vp_done,DB,B1,C),
        !, rulematch(C,B1,B2,TL,Rulematch,AS,Packets,[B3|Unseen],DB).



\\\\\
```

```
/* TOP.LPL: Top level of the parser etc

                                                    Lawrence
                                                    Updated: 21 November 80
*/

%% Run Interpreted %%


                        % Go from the terminal

go    :- ttynl, display('Sentence: '), ttyflush,
         read_in(Sentence),
         convert_wordlist(Sentence,Nodelist),
         asserta( last_sent(Sentence) ),
         do_sentence(Nodelist).


                        % Show all the sentences

show :- ttynl, display('Sentences:'), ttynl, ttynl,
         last_sent(Sentence),
         display(Sentence), ttynl,
         fail.

show.


                        % Redo last sentence

redo :- call( last_sent(Sentence) ),
         !,
         ttynl, display('Parsing: '), display(Sentence), ttynl,
         convert_wordlist(Sentence,Nodelist),
         do_sentence(Nodelist).


                        % Remove record of last sentence

oops :- retract( last_sent(_) ),
         display('(Ok, I''ve forgotten it!)'), ttynl,
         !.


                        % Leave the parser, showing all the sentences

bye   :- log,
         show, ttynl,
         display('Goodbye'), ttynl,
         halt.


                        % Parse a sentence
```

```
do_sentence(Nodelist)
     :-
        try_parse(Nodelist,Time,ANS),
        showtime(Time),
        set_tree(ANS,Tree),
        print_tree(Tree), ttynl,
        try_db(ANS),
        !.



                        % Error message if can't do it

try_parse(Nodelist,Time,ANS)
     :- parse(Nodelist,Time,ANS),
        !.

try_parse(_,_,_)
     :- ttynl, display('Sorry! I couldn''t parse that'), ttynl,
        fail.



                        % Ask if user wants to see database

try_db(DB)
     :- ttynl, display('Print Database? '), ttyflush,
        getyes,
        !,
        ttynl, display('Data Base:'), ttynl,
        print_db(DB).

try_db(_).


getyes
     :- repeat,
          get0(C),
          ( (C \/ 8'40) =:= 121
                    ;   /*C =:= 31, !, fail*/
                        ;   fail
          ),
        skip(31).



                        % Show the CPU time taken

showtime(Time)
     :- nl, write('Runtime = '), write(Time),
        write(' milliseconds.'), nl, nl.


                        % Input lots of sentences from a file

input(Problem)
     :- seeing(Old),
        fileerrors,
        log,
        tfast,
```

```
        repeat,
          flush_sensym,
          see(Problem),
          read_in(Sentence),
          see(Old),
          do(Sentence),
        close(Problem),
        !.


do([end,.]).

do(Sentence)
     :- ttynl, display('Parsing: '), display(Sentence), ttynl,
        convert_wordlist(Sentence,Nodelist),
        try_parse(Nodelist,Time,ANS),
        showtime(Time),
        times_called(rules_checked),
        times_called(rules_run),
        times_called(attach),
        set_tree(ANS,Tree),
        %print_tree(Tree), ttynl,
        %print_db(ANS),
        !,
        fail.

increment(Attach) :- counter(Attach,N),
        retract(counter(Attach,N)),
        N1 is N+1, !,
        assert(counter(Attach,N1)).

increment(Attach) :- assert(counter(Attach,1)).

times_called(Function) :-
        counter(Function,Times),
        write(Function), write(' was called '), write(Times),
        write(' times.'), nl,
        !, retract( counter(Function,Times) ).
```

← NB

```
ILE: TFLAG.LPL @22:8 10-APR-1981 <005> (101)
/* TFLAG.LPL : Setting the trace flag etc.

                                                    Lawrence
                                                    Updated: 23 April 81
*/

%% Run Interpreted %%


                        % Simple routines for setting the flag

tnice :- flag(trace_flag,_,tnice), t.


ton :- flag(trace_flag,_,ton), t.


  ff :- flag(trace_flag,_,toff), t.


tfast :- flag(trace_flag,_,tfast), t.

tcrash :- flag(trace_flag,_,tcrash), t.


                        % Show current flag

t     :- flag(trace_flag,TF,TF),
         ttynl,
         display('Tracing is set to: '),
         display(TF), ttynl.



  \\\
```

```
SUBFILE: INIT. @20:58 10-APR-1981 <005> (357)
/* INIT : Interpreter initialisations for Rob's parser

                                                    Lawrence
                                                    Updated: 23 April 81
*/

%% Consult this file:  [init]  %%


% FIXES
%
%  (31 March 81)
%
%       Changed ok so that it calls 'reinitialise' instead of '$reinit'
%       This is so that the new Prolog system (v 3.23 today) can be used
%


  :- tfast.              % Initial tracing setting


portray(X) :- pn1(X).    % Allows proper treatment of features for everything
                         % printed (eg during debugging).



                         % Mappings from packet names to file names

packet(P) :- filename(P,_).


filename(ss_start,'sstart.pk').
filename(cpool,'cpool.pk').
filename(npool,'npool.pk').
filename(parse_det,'pdet.pk').
filename(parse_qp_1,'pqp1.pk').
filename(parse_qp_2,'pqp2.pk').
filename(parse_adj,'padj.pk').
filename(parse_noun,'pnoun.pk').
filename(np_complete,'npcom.pk').
filename(parse_pp,'parpp.pk').
filename(parse_subj,'psubj.pk').
filename(no_subj,'nosubj.pk').
filename(parse_aux,'paux.pk').
filename(build_aux,'bldaux.pk').
filename(parse_vp,'pvp.pk').
filename(passive,'passiv.pk').
filename(ss_vp,'ssvp.pk').
filename(object,'obj.pk').
filename(no_subj,'nosubj.pk').
filename(that_comp,'thatc.pk').
filename(inf_comp,'infc.pk').
filename(to_less_inf_comp,'tlicom.pk').
filename(to_be_less_inf_comp,'tblcom.pk').
filename(two_obj,'twobj.pk').
filename(embedded_s_final,'embsfi.pk').
```

```
filename(build_name,'bname.pk').
filename(parse_conj,'pconj.pk').
filename(ss_final,'ssfin.pk').



                        % Try not to use this!
                        %   (if you want efficient compilation)

not(X) :- X, !, fail.

not(X).



                        % Set up a core image

ok :-   core_image, display('           [ ROBBIE the Parser ]'),
        ttynl, ttynl,
        reinitialise.



                        % Easy input (for file 'x')

i :- input(x).



\\\\\
```

```
SUBFILE: BITINI. @13:17 4-MAR-1981 <005> (188)
/* BITINI : Set aside certain bits in the feature repn

                                                        Lawrence
                                                        Updated: 21 November 80
*/

%% Consult this file:  [bitini]  %%


% Various funny feature operations assume that certain bits in the
% feature bit-vector have been set aside for certain features.
%
% This is set up here. See the files:
%
%                         HACKS.LPL       for the bit hackins
%                         DBREP.LPL       for how these entries work



:- recorda(#,#(14),_),                                  % Counter

   recorda(ns,bits(1),_),                               % NP number
   recorda(np1,bits(2),_),

   recorda(s,bits(3),_),                                % Node types
   recorda(np,bits(4),_),
   recorda(vp,bits(5),_),
   recorda(pp,bits(6),_),
   recorda(ap,bits(7),_),
   recorda(qp,bits(8),_),

   recorda(no_subj,bits(9),_),                          % Verb types
   recorda(that_comp,bits(10),_),
   recorda(inf_comp,bits(11),_),
   recorda(to_less_inf_comp,bits(12),_),
   recorda(to_be_less_inf_comp,bits(13),_),
   recorda(two_obj,bits(14),_).



\\\\\
```

```
SUBFILE: UTILR.PL @20:39 13-NOV-1980 <005> (67)
/* UTILR : Selected utilities for Rob's parser

                                                        Lawrence
                                                        Updated: 8 Oct 80
*/


  :- public member/2,
            append/3.


  :- mode member(?,?),
          append(?,?,?).



  mber(X,[X|_]).

member(X,[_|Rest]) :- member(X,Rest).


append([],L,L).

append([X|Rest],L,[X|Others]) :- append(Rest,L,Others).



\\\\\
```

```
SUBFILE: LOOKUP.LPL @17:11 13-NOV-1980 <005> (548)
/* LOOKUP.LPL : Dictionary lookup routines

                                                    Lawrence
                                                    Updated: 9 Oct 80
*/


   :- public convert_wordlist/2,
             lookup/2.


   :- mode convert_wordlist(+,?),
           lookup(+,?),
           lookup2(+,-),
           askuser(+,-),
           like(+,+,-),
           Jdone(+,+,+).



                        % Convert a list of words to a list of nodes
                        %  by looking up all the words
                        %  Stick a couple of null nodes on the end to make
                        %  sure all the buffers are always full of something

convert_wordlist([],[null_node,null_node]).

convert_wordlist([Word|Rest1],[Node|Rest2])
     :- lookup(Word,Node),
        convert_wordlist(Rest1,Rest2).



                        % Lookup the definition of a word
                        %  Easy if an integer
                        %  or try the dictionary
                        %  or try using the morphology
                        %  or, in desperation, ask the user

lookup(V,_)
     :- var(V),
        !,
        ttynl, display('** LOOKUP failure - attempt to lookup variable.'),
        ttynl,
        fail.

lookup(Node,Node) :- isnode(Node), !.           % Already done

lookup(Word,word_node(Word,Fr))
     :- lookup2(Word,Fr),
        !.

lookup(Word,_)
     :- display('** Unknown word: '),
        display(Word), ttynl,
        fail.
```

```
                        % Find the <FeatureRepn>

lookup2(I,Fr)
     :- integer(I),
        !,
        feature_repn(quantity,Fr).

lookup2(Word,Fr)
     :- get(def,Word,Fr).

lookup2(Word,Fr)
     :- morpho(Word,Fr),
        put(def,Word,Fr).               % Remember it!

lookup2(Word,Fr)
     :- askuser(Word,Fr).



                        % Ask the user for a definition
                        %  He must supply a similar word that is known

askuser(Word,Fr)
     :- ttynl, display('WARNING - unknown word: '), display(Word),
        ttynl, ttynl,
        display('Please give another word which it is like'), ttynl,
        display('(End with a period, Type ^Z to give up)'), ttynl,
        repeat,
           ( ttynl, display('> ')
                        ;  display('(Have another go)'), ttynl, fail ),
           ttyflush,
           read(Other),
           ( atomic(Other)  ; display('Please give one word in lower case'),
                              fail ),
           ( Other = end_of_file,
             seen,
             !,
             fail               ;  like(Other,Word,Fr)
        ),
        ttynl,
        !.



                        % Declare that some word is like an already known word
                        %  Look in dictionary
                        %  or use morphology

like(Oldword,Newword,Fr)
     :- get(def,Oldword,Fr),
        !,
        jdone(Oldword,Newword,Fr).

like(Oldword,Newword,Fr)
     :- morpho(Oldword,Fr),
        !,
        jdone(Oldword,Newword,Fr),
```

```
        ttytab(5), display('(morphology used)'), ttynl.

like(Oldword,_,Fr)
     :- display('Sorry, I don''t know '), display(Oldword),
        display(' either!!'), ttynl,
        fail.



                          % Now have a valid definition; enter with messase

Jdone(Oldword,Newword,Fr)
     :- put(def,Newword,Fr),
        display('OK - '), display(Newword),
        display(' is now like '), display(Oldword).



  \\\
```

```
SUBFILE: MORPHO.LPL @13:48 20-MAR-1981 <005> (393)
/* MORPHO.LPL : Morphology for Rob's parser

                                                    Lawrence
                                                    Updated: 21 March 81
*/


   :- public morpho/2.


   :- mode morpho(+,-),
           set_change(+,-,-),
           find_endings(+,-,-,-),
           further(+,+,-,-),
           endings(?,?),
           for(+,-),
           door(+,+,-).



                          % Try to find features of a word using
                          %  morphological rules

morpho(Word,Fr)
      :- find_endings(Word,Endings,FrE,Root),
         set_change(Endings,FrAdd,FrDel),
         or(FrE,FrAdd,Added),
         frsubtract(Added,FrDel,Fr),
         asserta( root(Word,Root) ),
         !.



                          % Find features to add and delete

   t_change(Endings,FrAdd,FrDel)
      :- set(morph,Endings,FrAdd-FrDel),
         !.

set_change(Endings,_,_)
      :- ttynl, display('** No morph mappins for: '),
         display(Endings), ttynl,
         fail.



                          % Try and pull an endins off the word

find_endings(Word,Endings,Fr,Root)
      :- name(Word,Chars),
         endings(Echars,Ftype),
         append(Rchars,Echars,Chars),
         !,
         further(Ftype,Rchars,Fr,Root),
         name(Endings,Echars).
```

```
further(Ftype,Rchars,Fr,Root)
     :- for(Ftype,Op),
        doop(Op,Rchars,Rchars2),
        name(Root,Rchars2),
        set(def,Root,Fr),
        !.


                        % Table of endings

ending("ing",further).
ending("ed",further).
ending("en",further).
ending("er",further).
ending("est",further).
ending("es",furthers).
ending("s",furthers).
  ding("ly",furtherly).
.nding("ness",further).
ending("ise", further).


                        % Further operations after removal of ending

for(_, null).

for(further, add("e")).
for(further, swap("i","y")).
for(further, swap([C,C],[C])).
for(further, swap("v","f")).

for(furthers, add("e")).
for(furthers, swap("ie","y")).
for(furthers, swap("ve","f")).

  r(furtherly, add("ile")).
.or(furtherly, swap("i","y")).


                        % Perform operations

doop(null,Rchars,Rchars).

doop(add(S),Rchars,Rchars2)
     :- append(Rchars,S,Rchars2).

doop(swap(S1,S2),Rchars,Rchars2)
     :- append(X,S1,Rchars),
        append(X,S2,Rchars2).



\\\\\
```

```
SUBFILE: HAS.LPL @13:18 4-MAR-1981 <005> (251)
/* HAS.LPL :  Routine for checking that a buffer satisfies
              certain feature constraints.

                                                    Lawrence
                                                    Updated: 8 March 81
*/


  :- public has/2.


  :- mode has(+,+),
        check_spec(+,+).



       Buffer has Spec

         where:

               Spec --> t
                     |  Spec & Spec
                     |  Spec # Spec
                     |  not Spec
                     |  Feature.

               Feature --> {some atom representing a feature}.

         and:

               Buffer is a <NodeStructure>

*/



                    % Check that some node satisfies the Feature Spec
                    %  note that Spec is given using literal features
                    %  ie atoms rather than Bit vectors


has(V,_)
     :- var(V),
        !,
        ttynl,
        display('** HAS error - Buffer is a variable.'), ttynl,
        fail.

has(_,t) :- !.                          % speedup

has(Node,Spec)
     :- set_feats(Node,Fr),
        check_spec(Spec,Fr).
```

```
                        % Do the actual check (decode the Spec)

check_spec(t,_) :- !.

check_spec(S&Srest,Fr)
     :- !,
        check_spec(S,Fr),
        check_spec(Srest,Fr).

check_spec(S#Srest,Fr)
     :- check_spec(S,Fr),
        !.

check_spec(S#Srest,Fr)
     :- !,
        check_spec(Srest,Fr).

check_spec(not(S),Fr)
     :- check_spec(S,Fr),
        !,
        fail.

check_spec(not(S),Fr) :- !.

check_spec(S,Fr)
     :- check_feature(S,Fr).




\\\\\
```

```
SUBFILE: PACKS.LPL @17:11 13-NOV-1980 <005> (118)
/* PACKS.LPL : Routines concerned with packets

                                                    Lawrence
                                                    Updated: 5 Oct 80
*/


   :- public activate/3,
             deactivate/3.


   :- mode activate(+,+,?),
           deactivate(+,+,?).



                          % Activate new packet on top of AP stack

activate(Packet,APlist,[Packet|APlist]).



                          % Deactivate packet from top of AP stack

deactivate(Packet,[Packet|APlist],APlist) :- !.

deactivate(Packet,[P|Rest1],[P|Rest2]) :- !, deactivate(Packet,Rest1,Rest2).

deactivate(Packet,[],[])
      :- display('** Unable to deactivate '),
         display(Packet), ttynl,
         fail.



\\\\\
```

```
SUBFILE: NODE.LPL @16:41 20-SEP-1981 <005> (1375)
/* NODE.LPL : Routines for handling parse tree nodes

                                                    Lawrence
                                                    Updated: 16 January 81
*/

  :- public isnode/1,
            new_node/2,
            new_node/3,
            attach/4,
            attach/5,
            percolate/3,
            addfeats/3,
            coerce/3,
            set_label/2,
            set_feats/2,
            change_feats/4,
            chlabel/2,
            closenode/2.


  :- mode isnode(+),
          new_node(+,-),
          new_node(+,+,-),
          attach(+,+,+,-),
          attach(+,+,+,-,+),
          atterr(+,+,+,+),
          percolate(+,+,-),
          addfeats(+,+,-),
          addf(+,+,-),
          coerce(+,+,-),
          coercefr(+,+,-),
          transfer(+,+,+,-),
          transfer2(+,+,+,-),
          np_number(+,+,+,-),
          select_hole(+,+,?,?,?),
          set_label(+,?),
          set_word(+,?),
          set_feats(+,-),
          change_feats(+,-,-,-),
          chlabel(+,-),
          closenode(+,-).


/*

  A <NodeStructure> can be one of the following types:

    null_node

    word_node(Word,Fr)

    closed_node(Label,Fr,Citems)

    open_node(Label,Fr,Oitems,Hole)
```

```
        where:
                Word is a word from the dictionary.
                Fr is a <FeatureRepn> structure.
                Label is of the form Type-N.
                        Type is a simple feature (an <atom>).
                        N is an identifying integer.
                Citems is either
                        [], an empty list;
                        [_|_], a list of closed nodes or word nodes.
                Oitems and Hole form a difference list of closed nodes
                        (ie Hole is the variable at the end of Oitems).

*/



                        % Check that some structure is a node

isnode(V) :- var(V), !, fail.

isnode(null_node).

isnode(word_node(_,_)).

isnode(closed_node(_,_,_)).

isnode(open_node(_,_,_,_)).



                        % Create a new open node (two versions)

new_node(Type,Node)
     :- new_node(Type,[],Node).


new_node(Type,Flist,open_node(Label,Fr,Hole,Hole))
     :- feature_repn([Type|Flist],Fr),
        gensym_label(Type,Label),
        !.

new_node(Type,Flist,_)
     :- display('** Cannot create new node: '),
        display(Type), display(' - '), display(Flist), ttynl,
        fail.



                        % Attach a node to another node as a Foo
                        %  This routine also moves various features
                        %  about, and makes decisions about which
                        %  hole to retain for future attaches

attach(   word_node(Word,FrL),
          open_node(Label,FrU,Items,
                [ closed_node(Foo-'*',Null,
                        [ word_node(Word,FrLnew) ] )
                | Hole ]),
          Foo,
          open_node(Label,FrUnew,Items,Hole) )
```

```
    :-
        increment(attach),
        coercefr(Foo,FrL,FrLnew),
        transfer(FrLnew,Label,FrU,FrUnew),
        nullfr(Null),
        !.

attach(   open_node(LabelL,FrL,ItemsL,HoleL),
          open_node(LabelU,FrU,ItemsU,
                [ closed_node(LabelL,FrLnew,ItemsL)
                | HoleU ]),
          Foo,
          open_node(LabelU,FrUnew,ItemsU,HoleX) )
    :-
        increment(attach),
        addf(Foo,FrL,FrLnew),
        transfer(FrLnew,LabelU,FrU,FrUnew),
        select_hole(LabelL,LabelU,HoleL,HoleU,HoleX),
        !.

attach(Node1,Node2,Foo,_)
    :- atterr(attach,Node1,Node2,Foo),
        fail.


                                %       Attach for after the headnoun
                                %       called only a few times
                                % hole is lower hole is has a headnoun

attach( open_node(LabelL,FrL,ItemsL,HoleL),
                open_node(np-NumU,FrU,ItemsU,
                        [closed_node(LabelL,FrLnew,ItemsL)
                         ]),
                Foo,
                open_node(np-NumU,FrUnew,ItemsU,HoleL),
                DB      )
        :-
        increment(attach),
                find( headnoun(np-NumU,_), DB ),
                addf(Foo,FrL,FrLnew),
                transfer(FrLnew,np-NumU,FrU,FrUnew),
                ! .

        %       This is a hack to do complex head nouns okay.
        %       It avoide usins noun_hack

attach( word_node(Word,FrL),
         open_node(Label,FrU,Items,
                [ closed_node(noun-'*',Null,
                        [ word_node(Word,FrLnew) ] )
                | Hole ]),
         nounns,
         open_node(Label,FrU,Items,Hole),
         DB )
   :-
        increment(attach),
        coercefr(noun,FrL,FrLnew),
        nullfr(Null),
        !.
```

```
attach(  NodeL,NodeU,Foo,Result,DB) :-
                attach(NodeL,NodeU,Foo,Result), !.


atterr(Type,X,Y,Z)
     :- ttynl,
        display('** Failed to '),
        display(Type), display('  (as '),
        display(Z), display('):'), ttynl,
        display('Lower: '), print(X), ttynl,
        display('Upper: '), print(Y), ttynl.



                        % Percolate features across
                        %  This is like attach except that nothing
                        %  is attached - only the movement of features
                        %  occurs

percolate(NodeL,NodeU,Newnode)
     :- set_feats(NodeL,FrL),
        change_feats(NodeU,FrU,FrNew,Newnode),
        set_label(NodeU,Label),
        chlabel(Label,Labelnew),
        transfer(FrL,Labelnew,FrU,FrNew),
        !.

percolate(Node1,Node2,Newnode)
     :- atterr(percolate,Node1,Node2,''),
        fail.

                                % chlabel is a hack for percolate

chlabel(aux-X,aux1-X) :- !.
chlabel(_,_).


                        % Add features to a node

addfeats(Node1,Flist,Node2)
     :- change_feats(Node1,Fr1,Fr2,Node2),
        addf(Flist,Fr1,Fr2),
        !.

addfeats(Node1,Flist,_)
     :- display('** Cannot add features: '),
        display(Flist), ttynl,
        display('   To: '), print(Node1), ttynl,
        fail.



addf(Flist,Fr1,Fr2)
     :- feature_repn(Flist,FrX),
        or(Fr1,FrX,Fr2).
```

```
                        % Coerce the features of a node in
                        %  accordance to a type

coerce(Type,Node,Newnode)
     :- change_feats(Node,Fr1,Fr2,Newnode),
        coercefr(Type,Fr1,Fr2),
        !.

coerce(Type,Node,_)
     :- display('** Cannot coerce (to '), display(Type),
        display('): '), print(Node), ttynl,
        fail.



                        % Coerce features to agree with Type
                        %  'coercefr' deals with feature reprs
                        %  'coerce' (above) deals with nodes

  ercefr(Type,Fr,Frnew)
     :- get(coerce,Type,FrX),
        !,
        and(Fr,FrX,Frnew).

coercefr(Type,Fr,Fr).



                        % Transfer features up

transfer(Frlower,Type-_,Frupper,Frnew)
     :- transfer2(Type,Frlower,Frupper,FrX),
        np_number(Type,Frlower,FrX,Frnew).


transfer2(Type,Frlower,Frupper,FrX)
     :- get(transfer,Type,FrT),
        !,
        and(Frlower,FrT,Y),
        or(Y,Frupper,FrX).

transfer2(_,_,Frupper,Frupper).


np_number(np,Frlower,Frupper,Frnew)
     :- !,
        noun_hack(Frlower,Frupper,Frnew).

np_number(_,_,Fr,Fr).



                        % Decide whether to use deep or normal hole
                        %  This is pretty unsophisticated at the moment

select_hole(X-_,Y-_,DeepHole,[],DeepHole) :- get(deep,X,Y), !.

select_hole(_,_,[],NormalHole,NormalHole).
```

```
                        % Access the label of a node

set_label(closed_node(Label,_,_),L) :- !, L=Label.

set_label(open_node(Label,_,_,_),L) :- !, L=Label.

set_label(word_node(Word,_),L) :- root(Word,L), !.

set_label(word_node(Word,_),L) :- !, L=Word.

set_label(Node,_)
     :- ttynl, display('** Cannot set label: '),
        print(Node), ttynl,
        fail.


                        % Select features from a node

set_feats(null_node,NFr) :- !, nullfr(NFr).

set_feats(Node,Fr) :- arg(2,Node,Fr).


                        % Handy routine for changing features

change_feats(word_node(W,Fr1),Fr1,Fr2,word_node(W,Fr2)).

change_feats(closed_node(L,Fr1,I),Fr1,Fr2,closed_node(L,Fr2,I)).

change_feats(open_node(L,Fr1,I,H),Fr1,Fr2,open_node(L,Fr2,I,H)).


                        % Close an open node

  osenode( open_node(Label,Fr,Items,[]),
           closed_node(Label,Fr,Items) ).



\\\\\
```

```
SUBFILE: FEATUR.LPL @17:4 22-MAR-1981 <005> (987)
/* FEATUR.LPL : Routines for handling feature representations

                                                        Lawrence
                                                        Updated: 13 March 81
*/


  :- public isfr/1,
            nullfr/1,
            feature_repn/2,
            bits_repn/2,
            check_feature/2,
            and/3,
            or/3,
            frsubtract/3,
            frtolist/2,
            frbits/6.


  :- mode isfr(?),
          nullfr(?),
          feature_repn(+,?),
          feature_repn(+,-,-,-,-,-,-),
          bits_repn(+,?),
          makerepn(+,-,-,-,-,-,-),
          check_feature(+,+),
          chkf(+,+),
          and(+,+,-),
          or(+,+,-),
          compl(+,-),
          frtolist(+,?),
          frvecs(+,+,?,?),
          frvec(+,+,?,?),
          frbits(+,+,+,+,?,?).



  A <FeatureRepn> has the following form:

    feature(N1,N2,N3,N4,N5,N6)

        where:
                N1,N2,N3,N4,N5,N6 are integers thus providing a 108 bit vector
                        (ie 6 x 18 bits).


*/


                        % Structure is a <FeatureRepn>

isfr(V) :- var(V), !, fail.

isfr(feature(_,_,_,_,_,_)).
```

```
                        % The null feature repn

nullfr(feature(0,0,0,0,0,0)).


%% Conversion from surface form %%


                        % Convert from literal repn to <FeatureRepn>

feature_repn(Flist,feature(N1,N2,N3,N4,N5,N6))
     :- feature_repn(Flist,N1,N2,N3,N4,N5,N6),
        !.

feature_repn(Flist,_)
     :- display('** Failed to convert feature list: '),
        display(Flist), ttynl,
        fail.


feature_repn([],0,0,0,0,0,0) :- !.

feature_repn([HD|TL],N1,N2,N3,N4,N5,N6)
     :- !,
        feature_repn(HD,Hn1,Hn2,Hn3,Hn4,Hn5,Hn6),
        feature_repn(TL,Tn1,Tn2,Tn3,Tn4,Tn5,Tn6),
        N1 is Hn1 \/ Tn1,
        N2 is Hn2 \/ Tn2,
        N3 is Hn3 \/ Tn3,
        N4 is Hn4 \/ Tn4,
        N5 is Hn5 \/ Tn5,
        N6 is Hn6 \/ Tn6.

feature_repn(A,N1,N2,N3,N4,N5,N6)
     :- atom(A),
        get(fr,A,X),
        makerepn(X,N1,N2,N3,N4,N5,N6),
        !.

feature_repn(X,_,_,_,_,_,_)
     :- ttynl,
        display('** Unknown feature: '),
        display(X), ttynl,
        fail.



                        % Database uses bits(....) rather then feature(....)
                        %  for defining the Feature tass themselves.

bits_repn(Flist,bits(N1,N2,N3,N4,N5,N6))
     :- feature_repn(Flist,N1,N2,N3,N4,N5,N6),
        !.

bits_repn(Flist,_)
     :- feature_repn(Flist,error).              % Hack! Forces error message
```

```
                        % Decode a bits database entry

makerepn(bits(N1,N2,N3,N4,N5,N6),N1,N2,N3,N4,N5,N6).

makerepn(bits(N),N1,0,0,0,0,0) :- N =< 18, !, N1 is 1 << (N-1).

makerepn(bits(N),0,N2,0,0,0,0) :- N =< 36, !, N2 is 1 << (N-19).

makerepn(bits(N),0,0,N3,0,0,0) :- N =< 54, !, N3 is 1 << (N-37).

makerepn(bits(N),0,0,0,N4,0,0) :- N =< 72, !, N4 is 1 << (N-55).

makerepn(bits(N),0,0,0,0,N5,0) :- N =< 90, N5 is 1 << (N-73).

makerepn(bits(N),0,0,0,0,0,N6) :- N =<108, N6 is 1 << (N-91).


%% Checking presence of features %%

                        % Check literal feature spec against <FeatureRepn>

check_feature(F,Fr)
     :- atom(F),
        set(fr,F,Br),
        !,
        chkf(Br,Fr).


chkf(bits(Bit),feature(N1,N2,N3,N4,N5,N6))
     :- Bit =< 18, !, N1 /\ (1 << (Bit-1)) > 0 ;
        Bit =< 36, !, N2 /\ (1 << (Bit-19)) > 0 ;
        Bit =< 54, !, N3 /\ (1 << (Bit-37)) > 0 ;
        Bit =< 72, !, N4 /\ (1 << (Bit-55)) > 0 ;
        Bit =< 90, !, N5 /\ (1 << (Bit-73)) > 0 ;
        Bit =<108, !, N6 /\ (1 << (Bit-91)) > 0.

chkf(bits(F1,F2,F3,F4,F5,F6),feature(N1,N2,N3,N4,N5,N6))
     :- F1 =:= F1 /\ N1,
        F2 =:= F2 /\ N2,
        F3 =:= F3 /\ N3,
        F4 =:= F4 /\ N4,
        F5 =:= F5 /\ N5,
        F6 =:= F6 /\ N6.


%% Operations on <featureRepn>'s %%

                        % And together two feature repns

and(feature(A1,A2,A3,A4,A5,A6),feature(B1,B2,B3,B4,B5,B6),
                                  feature(N1,N2,N3,N4,N5,N6))
     :- N1 is A1 /\ B1,
        N2 is A2 /\ B2,
        N3 is A3 /\ B3,
```

```
        N4 is A4 /\ B4,
        N5 is A5 /\ B5,
        N6 is A6 /\ B6.



                        % Or together two feature repns

or(feature(A1,A2,A3,A4,A5,A6),feature(B1,B2,B3,B4,B5,B6),
                                    feature(N1,N2,N3,N4,N5,N6))
     :- N1 is A1 \/ B1,
        N2 is A2 \/ B2,
        N3 is A3 \/ B3,
        N4 is A4 \/ B4,
        N5 is A5 \/ B5,
        N6 is A6 \/ B6.



                        % Subtract one feature repn from another

frsubtract(feature(A1,A2,A3,A4,A5,A6),feature(B1,B2,B3,B4,B5,B6),
                                      feature(N1,N2,N3,N4,N5,N6))
     :- N1 is A1 /\ \(B1),
        N2 is A2 /\ \(B2),
        N3 is A3 /\ \(B3),
        N4 is A4 /\ \(B4),
        N5 is A5 /\ \(B5),
        N6 is A6 /\ \(B6).



%% Conversion back to surface form %%


                        % Go the other way
                        %  Convert a <FeatureRepn> into a list of
                        %  all the features (as <atom>'s)

frtolist(Fr,List)
     :- isfr(Fr),
        frvecs(1,Fr,L,[]),
        List = L.



frvecs(7,_,Z,Z) :- !.           % 7 is muber if Ints plus 1

frvecs(N,Fr,List,Z)
     :- ars(N,Fr,Bits),
        frvec(Bits,N,List,Others),
        Next is N+1,
        frvecs(Next,Fr,Others,Z).



frvec(0,_,Z,Z) :- !.

frvec(Bits,Vec,List,Z)
     :- frbits(1,18,Vec,Bits,List,Z).
```

```
frbits(N,Max,_,_,Z,Z) :- N > Max, !.

frbits(N,Max,Vec,Bits,[F|Others],Z)
     :- Bits /\ (1 << (N-1)) > 0,
        !,
        Bit is (Vec-1)*18 + N,
        set(bit,Bit,F),
        Next is N+1,
        frbits(Next,Max,Vec,Bits,Others,Z).

frbits(N,Max,Vec,Bits,Others,Z)
     :- Next is N+1,
        frbits(Next,Max,Vec,Bits,Others,Z).



.\\\\
```

```
SUBFILE: HACKS.LPL @13:49 13-MAR-1981 <005> (370)
/* HACKS.LPL : Various funny feature operations

                                                    Lawrence
                                                    Updated: 13 March 81
*/


  :- public noun_hack/3,
            same_node_type/3,
            verb_types/2.


  :- mode noun_hack(+,+,-),
          nhck(+,+,-),
          same_node_type(+,+,?),
          verb_types(+,+,?).



% These routines rely on a knowledge of which bits are used by
% certain features:     (Bits in first word of the vector)
%                Noun Number      1-2
%                Node Type        3-8
%                Verb Type        9-14
%
% This state of affairs is set up in BITINI



                        % Mangle the noun features to force
                        %  number agreement

noun_hack(feature(Bits1,_,_,_,_,_), feature(Bits2,N2,N3,N4,N5,N6),
                        feature(Newbits,N2,N3,N4,N5,N6))
     :- Lower is Bits1 /\ 2'11,
        Upper is Bits2 /\ 2'11,
        nhck(Upper,Lower,New),
        Newbits is (Bits2 /\ \(2'11)) \/ New.



                        % The algorithm is:
                        %  If NP empty move number feats up
                        %  If NP has both intersect (by using lower feats)
                        %  Otherwise they must agree (leave NP)

nhck(2'00,Lower,Lower) :- !.

nhck(2'11,Lower,Lower) :- !.

nhck(Upper,2'11,Upper) :- !.

nhck(Upper,Lower,Upper) :- \(Upper) /\ Lower =:= 0, !.

nhck(_,_,_)
     :- display('** Unable to force Noun Number agreement'), ttynl,
        fail.
```

```
                        % Check two nodes for the same Node type
                        %  features, return the type found (an <atom>)
                        %  There should only be one such type

same_node_type(Node1,Node2,Type)
     :- isnode(Node1),
        isnode(Node2),
        set_feats(Node1,Fr1),
        set_feats(Node2,Fr2),
        ars(1,Fr1,Bits1),
        ars(1,Fr2,Bits2),
        Ans is 2'11111100 /\ Bits1 /\ Bits2,
        frbits(3,8,1,Ans,L,[]),
        L = [Type].



                        % Return a list (of <atom>'s) of all the
                        %  Verb type features on a node

verb_types(Node,List)
     :- isnode(Node),
        set_feats(Node,Fr),
        ars(1,Fr,Bits),
        Ans is 2'1111110000000 /\ Bits,
        frbits(9,14,1,Ans,List,[]).



\\\\\
```

```
SUBFILE: SEM.LPL @0:47 22-NOV-1980 <005> (361)
/* SEM.LPL : Definition of semantic rule application

                                                    Lawrence
                                                    Updated: 21 November 80
*/

  :- public semantics/2,
            semantics/3,
            semantics/4,
            semantics/5.


  :- mode semantics(+,+),
          semantics(+,+,+),
          semantics(+,+,+,+),
          semantics(+,+,+,+,+),
          dosem(+,+,+),
          apply_sem(+,+,+,+),
          apply_sem(+,+,+),
          match_sem(+,+),
          msem(?,+).



                          % Interface from packets

semantics(Type,DB) :-   dosem(Type,DB,[]).


semantics(Type,DB,A) :- dosem(Type,DB,[A]).


semantics(Type,DB,A,B) :- dosem(Type,DB,[A,B]).


semantics(Type,DB,A,B,C) :- dosem(Type,DB,[A,B,C]).



                          % Find a semantic rule and apply it

dosem(Type,DB,Args)
     :- atom(Type),
        get(semantics,Type,Rule),
        !,
        apply_sem(Type,Rule,Args,DB).

dosem(Type,_,_)
     :- semerr('Undefined Semantic operation: ',Type).



                          % Error message

semerr(Mess,Type)
     :- ttynl, display('** '), display(Mess),
```

```
        display(Type), ttynl,
        display('   (continuing)'), ttynl.



                        % Apply a rule

apply_sem(_,Rule,Args,DB)
     :- apply_sem2(Rule,Args,DB),
        !.

apply_sem(Type,_,_,_)
     :- semerr('Semantics rule failure: ',Type).



                        % How to apply the various forms of a semantic
                        %  rule body

  ply_sem2(null_rule,_,_).

apply_sem2(rule(Match,Finds,Adds),Args,DB)
     :- match_sem(Match,Args),
        find(Finds,DB),
        add(Adds,DB).

apply_sem2(Rule1 or Rule2,Args,DB)
     :- apply_sem2(Rule1,Args,DB) ;
        apply_sem2(Rule2,Args,DB).



                        % Matching the parameters against the arguments

match_sem([],[]).

match_sem([M|Mrest],[A|Arest])
     :- msem(M,A),
        match_sem(Mrest,Arest).


msem(M,Node)
     :- var(M),
        !,
        set_label(Node,M).

msem(M:Feature,Node)
     :- Node has Feature,
        set_label(Node,M).
```

\\\\\

```
SUBFILE: DB.LPL @21:0 8-APR-1981 <005> (1117)
/* DB.LPL : Handling the semantic database

                                                        Lawrence
                                                        Updated: 10 April 81
*/

% FIXES
%
%  (31 March 81)
%
%       print_db had some calls to ttynl in it rather than nl. This wasn't
%       nice when other files where used. This is now fixed.
%


  :- public init_db/2,
            push_sent/3,
            pop_sent/2,
            set_tree/2,
            dbfinish/1,
            find/2,
            add/2,
            print_db/2,
            print_db/1.


  :- mode init_db(+,-),
          push_sent(+,+,?),
          pop_sent(+,?),
          set_tree(+,?),
          dbfinish(+),
          find(+,+),
          find2(+,+),
          add(+,+),
          add2(+,+),
          list(+,?,?),
          hash(+,-),
          maxhash(?),
          print_db(+,+),
          print_db(+),
          prarray(+,+,+),
          prbucket(+),
          prbucket2(+).



% The semantic database is carried around as a term throughout the
% parse. It has the form:
%
%       database(ParseTree,CurrentSentStack,Array)
%
%               where:
%
%                       ParseTree is a closed node which is the final
%                               parse of the sentence.
%                       CurrentSentStack is a stack of node labels which
%                               gives the sentence embeddings.
```

```
%                       Array is a large term (functor '$') which is used
%                               as a hash array. It holds all the semantic
%                               assertions.
%
%       ParseTree is a variable for most of the parse, it gets instantiated
% right at the end. Array is initialised, but only at the top level. Each
% entry is a list ending with a variable - this gets continually expanded.
% CurrentSentStack is just a list. Items are pushed and popped by copying
% the 'database(_,_,_)' structure and changing this list. When this is done
% both ParseTree and Array just get unified across.



                        % Initialise the database structure

init_db(Snode,database(_,[S],Array))
     :- set_label(Snode,S),
        maxhash(N),
        functor(Array,'$',N).



                        % Push a new embedded sentence

push_sent(Snode, database(ParseTree,Slist,Array),
                 database(ParseTree,[S|Slist],Array) )
     :- set_label(Snode,S).



                        % Pop an embedded sentence

pop_sent(database(ParseTree,[S|Rest],Array), database(ParseTree,Rest,Array) ).



                        % Get the parse tree

get_tree(database(ParseTree,_,_),ParseTree).



                        % Check that there are no embedded sentences
                        %  at the end of the parse

dbfinish(database(_,[],_)) :- !.

dbfinish(database(_,Gash,_))
     :- ttynl,
        display('Semantic sentence stack not empty at end of parse!'),
        ttynl, portray_stack(write,Gash), ttynl.



                        % Find something in the Semantic database
                        %  Some special cases are handled as well to give
                        %  access to various other procedures.

find(V,_)
```

```
     :- var(V),
        !,
        ttynl, display('** Using FIND with a variable: '),
        display(V), ttynl,
        fail.

find(true,_) :- !.

find(A & B, DB)
     :- !,
        find(A,DB),
        find(B,DB).

find(curr_sent(X),database(_,L,_)) :- !, list(L,X,_).

find(irreg_verb(Type,Name),_) :- !, irreg_verb(Type,Name).

find(word_to_num(Word,Num),_) :- !, word_to_num(Word,Num).

  nd(sensym_label(Type,Label),_) :- !, sensym_label(Type,Label).

find(Ass,database(_,_,Array))
     :- hash(Ass,N),
        arg(N,Array,Bucket),
        find2(Bucket,Ass).



find2(V,_) :- var(V), !, fail.

find2([Ass|_],Ass) :- !.

find2([_|Rest],Ass)
     :- find2(Rest,Ass).



                        % Add a fact to the semantic database

_dd(V,_)
     :- var(V),
        !,
        ttynl, display('** Using ADD with a variable: '),
        display(V), ttynl,
        fail.

add(true,_) :- !.

add(A & B, DB)
     :- !,
        add(A,DB),
        add(B,DB).

add(Ass,database(_,_,Array))
     :- hash(Ass,N),
        arg(N,Array,Bucket),
        add2(Bucket,Ass).
```

```
add2(V,Ass) :- var(V), !, V=[Ass|More].

add2([_|Rest],Ass)
     :- add2(Rest,Ass).



                        % Break up a list
                        %  (Force variables to be local - ho hum)
                        %  This routine is used in find(_,_)

list([HD|TL],HD,TL).



                        % How to hash facts to array addresses

hash(num(_,_,_), 1) :- !.
hash(headnoun(_,_), 2) :- !.
  sh(is_prep(_,_,_), 3) :- !.
..sh(isa(_,_), 4) :- !.
hash(headadj(_,_), 5) :- !.
hash(intensifier(_,_), 6) :- !.
hash(main_verb(_,_), 7) :- !.
hash(curr_sent(_), 8) :- !.
hash(syn_subj(_,_), 9) :- !.
hash(syn_obj(_,_), 10) :- !.
hash(wh_trace(_,_), 11) :- !.
hash(np_comp_s(_,_), 12) :- !.
hash(passive_sent(_), 13) :- !.
hash(conj(_,_,_), 14) :- !.
hash(poss_det(_,_), 15) :- !.
hash(relc(_,_), 16) :- !.
hash(hasfeat(_,_), 17) :- !.
hash(dim(_,_,_), 18) :- !.
hash(measure(_,_,_), 19) :- !.
hash(ordinal(_,_), 20) :- !.
hash(np_object(_,_), 21) :- !.
  sh(aux_verb(_,_), 22) :- !.
. sh(name(_,_), 23) :- !.
hash(adverb(_,_), 24) :- !.
hash(utterance(_,_), 25) :- !.
hash(embedded_sent(_), 26) :- !.
hash(pp_linked(_,_), 27) :- !.
hash(sentence(_), 28) :- !.
hash(negative_sent(_), 29) :- !.
hash(qp_modify(_,_), 30) :- !.
hash(wh_quest(_,_), 31) :- !.
hash(comparative(_,_,_), 32) :- !.
hash(qp_det(_,_), 33) :- !.
hash(quantifier(_,_), 34) :- !.
hash(stype(_,_), 35) :- !.

hash(ELSE, 36).


                                % Current size of array

maxhash(36).
```

```
                                % Print out the database
                                %  Two versions - one for a file

print_db(File,DB)
     :- open(Old,File),
        print_db(DB),
        close(File),
        see(Old).


print_db(database(_,_,Array))
     :- maxhash(Max),
        prarray(1,Max,Array).


prarray(N,Max,_) :- N > Max, !.

  array(N,Max,Array)
     :- ars(N,Array,Bucket),
        prbucket(Bucket),
        Next is N+1,
        prarray(Next,Max,Array).


prbucket(V) :- var(V), !.

prbucket(List)
     :- nl,
        prbucket2(List).



prbucket2(V) :- var(V), !.

prbucket2([First|Rest])
     :- tab(8), write(First), put(','), nl,
        prbucket2(Rest).



\\\\\
```

```
SUBFILE: ENTER.LPL @22:26 10-APR-1981 <005> (433)
/* ENTER .LPL :  Tracing routines.

                                                    Lawrence
                                                    Updated: 23 April 81
*/


  :- public enter/8,
            trace/9,
            crash/9.


  :- mode enter(+,+,+,+,+,+,+,+,+),
          doenter(+,-),
          trace(+,+,+,+,+,+,+,+,+,+),
          crash(+,+,+,+,+,+,+,+,+,+).



                        % Possibly give entry message
                        %  Catch failure of any rule and give message

enter(_,_,_,_,_,_,_,_)
     :- flag(trace_flag,tfast,tfast),
        !.

enter(How,Packet,Rule,B1,B2,B3,Cstack,Pstack)
     :- flag(trace_flag,TF,TF),
        doenter(TF,Type),
        trace(Type,How,Packet,Rule,B1,B2,B3,Cstack,Pstack).

enter(How,Packet,Rule,B1,B2,B3,Cstack,Pstack)
     :- crash(failure,How,Packet,Rule,B1,B2,B3,Cstack,Pstack).



doenter(ton,pn1).
doenter(tnice,pn2).
doenter(toff,toff).
doenter(tcrash,toff).           % to get readable crash messages



                        % Trace of a packet call

trace(toff,_,_,_,_,_,_,_,_,_) :- !.

trace(Type,How,Packet,Rule,B1,B2,B3,Cstack,Pstack)
     :- ttynl,
        display('Packet: '), display(Packet),
        tab(4), ttyput("("), display(How), ttyput(")"),  ttynl,
        display('Rule about to run:   '), display(Rule), ttynl,
        display('Active Node Stack:'), ttynl, portray_stack(Type,Cstack),
        display('B1: '), portray(Type,B1), ttynl,
        display('B2: '), portray(Type,B2), ttynl,
```

```
        display('B3: '), portray(Type,B3), ttynl,
        display('Packet stack:'), ttynl, portray_stack(write,Pstack).


                        % Handle a crash of the parser
                        %  Only report one error - ie make sure we fail
                        %  back out past all the rule failure checks if
                        %  they are there.

crash(_,_,_,_,_,_,_,_,_)
     :- flag(crashing,yes,yes),
        !,
        fail.

crash(Crash,How,Packet,Rule,B1,B2,B3,Cstack,Pstack)
        :- flag(trace_flag,tcrash,tcrash),
           ttynl, crashmess(Crash), ttynl,
           trace(pn2,How,Packet,Rule,B1,B2,B3,Cstack,Pstack),
           flag(crashing,_,yes),
           fail.

crash(Crash,How,Packet,Rule,B1,B2,B3,Cstack,Pstack)
     :- ttynl, crashmess(Crash), ttynl,
        trace(pn1,How,Packet,Rule,B1,B2,B3,Cstack,Pstack),
        flag(crashing,_,yes),
        fail.


crashmess(failure) :- display('** Rule Failure:').

crashmess(nomatch) :- display('** Either you blew it, or I just Garden Pathed:')



\\\\\
```

```
SUBFILE: PTREE.LPL @23:29 21-NOV-1980 <005> (366)
/* PTREE.LPL : Print a parse tree - ie a <NodeStructure>

                                                    Lawrence
                                                    Updated: 21 November 80
*/


  :- public print_tree/1.


  :- mode print_tree(+),
         pnode(+,+),
         psub(+,+,+,+,+,-),
         psubnodes(+,+),
         plabel(+),
         incrindent(+,-),
         pupper(+),
         pupp(+).



                        % Top level routine

print_tree(Tree)
     :- nl, tab(2),
        pnode(Tree,2),
        fail.

print_tree(_).


   ~
                        % Kinds of node

  node(null_node,_)
     :- write('<null>').

pnode(word_node(W,Fr),Indent)
     :- pupper(W),
        tab(3), portray_feature(Fr).

pnode(closed_node(Label,Fr,Citems),Indent)
     :- psub(Label,'',Fr,Citems,Indent).

pnode(open_node(Label,Fr,Oitems,_),Indent)
     :- psub(Label,'<open>',Fr,Oitems,Indent).



                        % Recurse and print sub nodes

psub(Label,X,Fr,Subnodes,Indent)
     :- plabel(Label),
        write(X), tab(1),
        portray_feature(Fr),
        incrindent(Indent,NewIndent),
```

```
        psubnodes(Subnodes,NewIndent).


psubnodes(V,Indent)
     :- var(V),
        !,
        nl, tab(Indent), write('<hole>').

psubnodes([],_).

psubnodes([First|Rest],Indent)
     :- nl, tab(Indent),
        pnode(First,Indent),
        psubnodes(Rest,Indent).



                        % Print a label

   label(Name-'*')
     :- !,
        pupper(Name), tab(2).

plabel(Name-N)
     :- pupper(Name), put("-"), write(N), tab(2),
        !.

plabel(X) :- write(X), tab(2).



                        % Increment the indentation, catch overflows!

incrindent(N,N) :- N >= 60, !.

incrindent(N,N3) :- N3 is N + 3.



                        % Print in upper case if possible

pupper(Name)
     :- atom(Name),
        !,
        ( name(Name,Chars),
          pupp(Chars),
          fail          ;  true ).

pupper(Name) :- write(Name).


pupp([]) :- !.

pupp([Char|Rest])
     :- (  Char >= "a", Char =< "z",     C is Char /\ 2'1011111, put(C)
                                    ;    put(Char)
        ),
        !,
        pupp(Rest).
```

\\\\\

```
SUBFILE: PORTR.LPL @17:16 13-NOV-1980 <005> (508)
/* PORTR.LPL : Useful portray routines

                                                    Lawrence
                                                    Updated: 7 November 80
*/


  :- public portray_feature/1,
            portray_stack/2,
            portray/2,
            pn1/1,
            pn2/1.


  :- mode portray_feature(+),
          portray_stack(+,+),
          portray_stack(+,+,+),
          portray(+,+),
          pn1(+),
          pn2(+).



                        % Portray a feature repn

portray_feature(Fr)
     :- isfr(Fr),
        ( frtolist(Fr,List),
          write(List),
          fail
                ;  true ).



                        % Portray a stack with counter

portray_stack(Type,X)
     :- portray_stack(X,1,Type).


portray_stack([],_,_).

portray_stack([Top|Rest],N,Type)
     :- put(" "), write(N), write(': '),
        portray(Type,Top), nl,
        N1 is N + 1,
        portray_stack(Rest,N1,Type).



                        % Switch into various portrays

portray(print,X) :- print(X).

portray(write,X) :- write(X).
```

```
portray(pn1,X) :- pn1(X).

portray(pn2,X) :- pn2(X).



                        % Portray any structure
                        %  Show everything except for feature repn conversion
                        %  and the database (which is masked)
                        %  Thus this routine can be used for portraying any
                        %  structure whatever which contains <FeatureRepn>'s
                        %  that you want done nicely

pn1(Simple)
     :- ( var(Simple) ; atomic(Simple) ),
        !,
        write(Simple).

pn1(database(_,_,_))
     :- !,
        display('<database>').

pn1(Fr)
     :- portray_feature(Fr),
        !.

pn1([X|Rest])
     :- !,
        put("["), pnargs1([X|Rest]), put("]").

pn1(X-Y)                % To catch Type-N
     :- !,
        pn1(X), put("-"), pn1(Y).

pn1(Term)
     :- ( Term =.. [F|Args],
          write(F), put("("), pnargs1(Args), put(")"),
          fail  ;  true ).



pnargs1([X|Rest])
     :- pn1(X),
        pnrest1(Rest).



pnrest1(V)
     :- var(V),
        !,
        put("|"), write(V).

pnrest1([]) :- !.

pnrest1(List)
     :- put(","),
        pnargs1(List).
```

```
                        % Portray a node, showing only the type of
                        %  the node, and any contained words.
                        %  This gives a simple high level view

pn2(null_node) :- write('<null>').

pn2(word_node(Word,_)) :- write(Word).

pn2(closed_node(Type-_,_,Items))
     :- pupper(Type), tab(3),
        pnwords2(Items).

pn2(open_node(Type-_,_,Items,_))
     :- write('<open>  '),
        pupper(Type), tab(3),
        pnwords2(Items).



  words2(V) :- var(V), !.

pnwords2([]).

pnwords2([First|Rest])
     :- pnwords2(First),
        pnwords2(Rest).

pnwords2(word_node(Word,_))
     :- write(Word), put(" ").

pnwords2(closed_node(_,_,Items))
     :- pnwords2(Items).

pnwords2(open_node(_,_,Items,_))
     :- pnwords2(Items).



.\\\
```

```
SUBFILE: LOAD.LPL @17:17 13-NOV-1980 <005> (875)
/* LOAD.LPL :  Routines for loading packets to be interpreted,
                dictionaries, and semantic rules.

                                                    Lawrence
                                                    Updated: 13 November 80

*/


  :- public load/1,
            unload/1,
            load_dict/1,
            load_sem/1,
            loaded/0.


  :- mode load(+),
          unload(+),
          load_dict(+),
          load_sem(+),
          load(+,+,+),
          load2(+,+),
          load3(+),
          load4(+,+),
          lfile(+,+),
          remember(+,+,-),
          loaded.



                        % Load/Unload files in various ways
                        %  Loaded packets are run interpretively
                        %  Dictionary entries go into database
                        %  Semantics rules go into the database

 load(X) :- load(X,load,packet).


unload(X) :- load(X,unload,packet).


load_dict(X) :- load(X,load,dictionary).


load_sem(X) :- load(X,load,semantics).



                        % Run down the file list

load(V,_,Things)
     :- var(V),
        !,
        loadmess(Things,V).

load([],_,_) :- !.
```

```
load([HD|TL],Op,Thing)
     :- !,
        load(HD,Op,Thing),
        load(TL,Op,Thing).

load(Packet,load,packet)
     :- filename(Packet,File),
        !,
        load2(Packet,File).

load(Packet,unload,packet)
     :- packet(Packet),
        !,
        load3(Packet).

load(Dict,load,dictionary)
     :- atom(Dict),
        !,
        load4(dictionary,Dict).

load(Sem,load,semantics)
     :- atom(Sem),
        !,
        load4(semantics,Sem).

load(X,Op,Thing)
     :- loadmess(Thing,X).



                        % An error message

loadmess(Thing,X)
     :- display('** Illegal '), display(Thing),
        display(': '), display(X), ttynl.



                        % Load a packet

load2(Packet,File)
     :- check_exists(File),
        reconsult(File),
        flag(Packet,_,interpreted),
        display('Packet: '), display(Packet),
        display(' loaded into the interpreter.'), ttynl.

load2(Packet,_)
     :- display('** Cannot load packet: '),
        display(Packet), ttynl.



                        % Unload a packet

load3(Packet)
     :- flag(Packet,interpreted,compiled),
        abolish(Packet,6),
        abolish(Packet,9),
```

```
        revive(Packet,6),
        revive(Packet,9),
        display('Packet: '), display(Packet),
        display(' removed from the interpreter.'), ttynl.

load3(Packet)
     :- display('** Packet: '), display(Packet),
        display(' is not currently loaded.'), ttynl.



                % Load a Dictionary or Semantic rules
                %  Dictionaries contain a restricted set of different things
                %  which are stored in the 'recorded' data-base under keys
                %  Semantics rule forms are also stored in the database.

load4(Type,File)
     :- open(Old,File),
        statistics(heap,[Total1,Free1]),
        repeat,
           read(X),
           lfile(Type,X),
        !,
        statistics(heap,[Total2,Free2]),
        seen,
        see(Old),
        Diff is Total2-Free2-Total1+Free1,
        ttynl,
        display('Loaded '), display(Type), display(': '),
        display(File), tab(3), display(Diff),
        display(' words.'), ttynl,
        remember(Type,File,Remember),
        ( call(Remember)  ;  assertz(Remember) ),
        !.

load4(Type,File)
     :- display('** Unable to load '), display(Type), display(': '),
        display(File), ttynl.



                        % Action on individual entries in file

lfile(Type,V)
     :- var(V),
        !,
        display('** Variable ignored.'), ttynl,
        fail.

lfile(Type,end_of_file).

lfile(dictionary,X) :- load_item(X), !, fail.

lfile(semantics,X) :- load_rule(X), !, fail.

lfile(Type,X)
     :- display('** Entry ignored: '),
        display(X), ttynl,
        fail.
```

```
                        % How to remember whats what

remember(dictionary,File,dictionary(File)).

remember(semantics,File,semantics(File)).



                        % Provide information to user about loaded
                        %  packets, dictionaries and rules
                        %  Its a fine piece of Prolog, huh?

loaded
     :- ttynl,
        ( packet(P),
          flag(P,interpreted,interpreted),
          display('Packets loaded into the interpreter:'),
          ttynl, ttynl,
          ( packet(Packet),
            flag(Packet,interpreted,interpreted),
            ttytab(6), display(Packet), ttynl,
            fail
                  ;  true
          )
            ;  display('All packets compiled.'), ttynl
        ),
        ttynl,
        ( dictionary(D),
          display('Dictionaries loaded:'),
          ttynl, ttynl,
          ( dictionary(Dict),
            ttytab(6), display(Dict), ttynl,
            fail
                  ;  true
          )
            ;  display('No dictionaries loaded.'), ttynl
        ),
        ttynl,
        ( semantics(S),
          display('Semantics loaded:'),
          ttynl, ttynl,
          ( semantics(Sem),
            ttytab(6), display(Sem), ttynl,
            fail
                  ;  true
          )
            ;  display('No semantics loaded.'), ttynl
        ),
        ttynl,
        !.
```

\\\\\

```
SUBFILE: DBREP.LPL @13:56 13-MAR-1981 <005> (1061)
/* DBREP.LPL : Primitives for manipulating the data base

                                                    Lawrence
                                                    Updated: 13 March 81
*/


  :- public load_item/1,
            load_rule/1,
            put/3,
            set/3,
            sensym_label/2,
            flush_sensym/0.


  :- mode load_item(+),
          checkGP(+,+),
          chkGP(+,-),
          load_rule(+),
          fadecide(+,-,-),
          drop(+,+),
          put(+,+,+),
          set(+,+,?),
          newbit(+,-),
          sensym_label(+,?),
          flush_sensym.


/*
  The data base contains various things such as the dictionary,
  information concerning features etc.

  The following input forms are keyed as shown:

    feature(F)                    F ->- bits(N)
                                  N ->- feature(F)

    feature(F,Flist)              F ->- bits(N1,N2,N3,N4,N5,N6)

    def(Word,F)                   Word ->- feature(N1,N2,N3,N4,N5,N6)
    def(Word,F,Flist)

    coerce(Type,Flist)            Type ->- coerce(feature(N1,N2,N3,N4,N5,N6))

    morph(Ending,FlADD,FlDEL)     Ending ->- morph(feature(N1,N2,N3,N4,N5,N6),
                                                   feature(M1,M2,M3,M4,M5))

    transfer(Type,Flist)          Type ->- transfer(feature(N1,N2,N3,N4,N5,N6))

    deep(Lower,Upper)             Upper ->- deep(Lower,Upper)


  Then there are semantic rules:

    semantics(Name,.....)         Name ->- null_rule
                           or     Name ->- rule(Match,Finds,Adds)
                           or     Name ->- Rule or Rule
```

```
  Plus other items as follows:

    The bit count                  # ->- #(N)

    Gensym counts                  Type ->- gensym(N)

*/


                        % Placing input forms in data base

load_item(feature(F))
     :- checkGP(1,F),
        newbit(F,N),
        drop(F,bits(N)),
        drop(N,feature(F)).

load_item(feature(F,Flist))
     :- checkGP(2,F),
        bits_repn(Flist,Br),
        drop(F,Br).

load_item(def(Word,F))
     :- feature_repn(F,Fr),
        drop(Word,Fr).

load_item(def(Word,F,Flist))
     :- feature_repn([F|Flist],Fr),
        drop(Word,Fr).

load_item(coerce(Type,Flist))
     :- feature_repn(Flist,Fr),
        drop(Type,coerce(Fr)).

load_item(morph(Ending,FlistADD,FlistDEL))
     :- feature_repn(FlistADD,FrADD),
        feature_repn(FlistDEL,FrDEL),
        drop(Ending,morph(FrADD,FrDEL)).

load_item(transfer(Node,Flist))
     :- feature_repn(Flist,Fr),
        drop(Node,transfer(Fr)).

load_item(deep(Lower,Upper))                    % NB no uniqueness force
     :- recorda(Upper,deep(Lower,Upper),_).



                        % Check that a particular feature is not being
                        %  overwritten by a general feature, or vice-versa.
                        %  This is a likely error!

checkGP(N,F)
     :- chkGP(N,Brep),
        recorded(F,Brep,_),
        !,
        ttynl, display('** General/Particular feature name mixup: '),
```

```
        display(F), ttynl,
        fail.

checkGP(_,_).


chkGP(1,bits(_,_,_,_,_,_)).  % General not to be overwritten by Particular

chkGP(2,bits(_)).            % Particular not to be overwritten by General


                        % Decoding and placing semantic rule forms in
                        %  the database

load_rule(semantics(Name))
     :- drop(Name,null_rule).

  ad_rule(semantics(Name,Body))
     :- trbody(Body,Ruleform),
        drop(Name,Ruleform).


trbody( X or Y, RX or RY )
     :- !,
        trbody(X,RX),
        trbody(Y,RY).

trbody( (Matches,find(Finds),add(Adds)), rule(Matches,Finds,Adds) ) :- !.

trbody( (Matches,FAs), rule(Matches,Finds,Adds) )
     :- !,
        fadecide(FAs,Finds,Adds).

trbody( Matches, rule(Matches,true,true) ).


, decide(find(X),X,true).

fadecide(add(X),true,X).


                        % Drop an item into the data base
                        %  Guarantee uniqueness

drop(Key,Stuff)
     :- atomic(Key),
        functor(Stuff,F,N),
        functor(TP,F,N),
        ( recorded(Key,TP,ID),
          ( TP == Stuff  ;  erase(ID),
                            recorda(Key,Stuff,_)
          )
           ;  recorda(Key,Stuff,_)
        ),
        !.
```

```
drop(Key,_)
     :- display('** Database error, Key must be atom: '),
        display(Key), ttynl,
        fail.


                        % Put items into database (low level, def case only)

put(def,Word,Fr)
     :- isfr(Fr),
        drop(Word,Fr).


                        % Get items from the database (all cases)

set(def,Word,Fr) :- recorded(Word,Fr,_), isfr(Fr), !.

   .(coerce,Type,Fr) :- recorded(Type,coerce(Fr),_), !.

set(morph,Endins,FrA-FrD) :- recorded(Endins,morph(FrA,FrD),_), !.

set(transfer,Node,Fr) :- recorded(Node,transfer(Fr),_), !.

set(fr,Feature,Br) :- recorded(Feature,Br,_), functor(Br,bits,_), !.

set(bit,Bit,Feature) :- recorded(Bit,feature(Feature),_), !.

set(deep,Lower,Upper) :- recorded(Upper,deep(Lower,Upper),_), !.

set(semantics,Name,Rule)
     :- recorded(Name,Rule,_),
        ruleform(Rule),
        !.

    ruleform(null_rule).
    ruleform(rule(_,_,_)).
    ruleform(or(_,_)).


                        % Find a bit in the Feature repn vector

newbit(F,Bit)
     :- recorded(F,bits(Bit),_),
        !.

newbit(F,Bit)
     :- recorded(#,#(N),ID),
        N < 108,                                % Feature Repn size specific
        !,
        Bit is N+1,
        erase(ID),
        recorda(#,#(Bit),_).

newbit(F,_)
     :- ttynl,
        display('** Ran out of feature bits while entering: '),
        display(F), ttynl, ttynl,
```

```
        abort.


                        % Gensym up a new label

gensym_label(Name,Name-N)
     :- atom(Name),
        ( recorded(Name,gensym(X),ID),
          erase(ID)
                    ;  X = 0 ),
        X1 is X+1,
        recorda(Name,gensym(X1),_),
        !,
        N = X1.

gensym_label(Name,_)
     :- display('** Atom required to gensym label: '),
        display(Name), ttynl,
        fail.


                        % Flush out all the gensym counters

flush_gensym
     :- current_atom(Name),
        recorded(Name,gensym(_),ID),
        erase(ID),
        fail.

flush_gensym.



\\\\\
```

```
SUBFILE: RULEM.LPL @16:39 20-SEP-1981 <005> (839)
/* RULEM.LPL : Rulematch etc for Rob's parser.

                                                    Lawrence
                                                    Updated: 10 March 81
*/

  :- public parse/3,
            rulematch/9,
            alldone/2.


  :- mode parse(+,?,?),
          rulematch(+,+,+,+,+,+,+,+,+,?),
          set_precedence(-),
          set_packet(+,-),
          check_rule(+,+,+,+,+,+,+,+,+,-),
          docheck(+,+,+,+,+,+),
          rule_spec(+,+,-,-,-,-,+,-),
          apply_rule(+,+,+,+,+,+,+,+,+,+,+,?),
          alldone(+,-).



                        % Parse a list of nodes in some Time to
                        %  get an answer. This routine starts the parser
                        %  proper. There is now a flag "crashing" which
                        %  brings failures back out with only one message

parse([B1,B2,B3|Unseen],Time,DB)
     :- flag(crashing,_,no),
        statistics(runtime,[Start,_]),
        new_node(s,Snode),
        init_db(Snode,DB),
        semantics(start,DB,Snode),
        rulematch(B1,B2,B3,[Snode],_,[],[[ss_start,cpool]],Unseen,DB),
        statistics(runtime,[Finish,_]),
        Time is Finish-Start.

parse(_,_,_)
     :- flag(crashing,_,no),
        fail.



                        % The main control of the parser
                        %  Cycle through rule specs lookins for a
                        %  fireable rule, then call it

rulematch(B1,B2,B3,Cstack,Rulename,AS,Pstack,Unseen,DB)
     :- set_precedence(Prec),
        set_packet(Pstack,Packet),
        check_rule(Packet,Prec,B1,B2,B3,Cstack,DB,Rulename,How),
        !,
        enter(How,Packet,Rulename,B1,B2,B3,Cstack,Pstack),
        apply_rule(How,Packet,Rulename,B1,B2,B3,
```

```
                                        Cstack,AS,Pstack,Unseen,DB).

        % this is for aux-inversion and wh movement

rulematch(B1,B2,B3,Cstack,Rulename,[[Type,B11]|AS],Pstack,Unseen,DB)
        :- !,
        rulematch(B1,B11,B2,Cstack,Rulename,AS,Pstack,[B3|Unseen],DB).

rulematch(B1,B2,B3,Cstack,_,_,Pstack,_,_)
     :- crash(nomatch,'','','',B1,B2,B3,Cstack,Pstack).



                        % Possible precedences (in order)

set_precedence(5).
set_precedence(10).
set_precedence(15).



                        % Currently active packets (from top of Pstack)

set_packet([Packets|_],P) :- member(P,Packets).



                        % See if a packet contains a fireable rule
                        %  Return the name of the first rule found

check_rule(Packet,Prec,B1,B2,B3,[C|Crest],DB,Rulename,How)
     :-   increment(rules_checked),
        rule_spec(Packet,Prec,Spec1,Spec2,Spec3,SpecA,Rulename,How),
        B1 has Spec1,
        B2 has Spec2,
        B3 has Spec3,
        docheck(SpecA,B1,B2,B3,C,DB).



                        % How to decode the check specification which
                        %  can either be a syntactic agreement check
                        %  or a semantic check

docheck(t,_,_,_,_,_).

docheck(agree(Type),B1,B2,_,_,_)
     :- agree(Type,B1,B2).

docheck(agree_13(Type),B1,_,B3,_,_)
     :- agree_13(Type,B1,B3).

docheck(agree_23(Type),_,B2,B3,_,_)
     :- agree_23(Type,B2,B3).

docheck(agree_all(Type),B1,B2,B3,_,_)
     :- agree_all(Type,B1,B2,B3).

docheck(sem_chk(Type),B1,B2,B3,C,DB)
     :- semantic_check(Type,B1,B2,B3,C,DB).
```

```
                        % Find specs for rules

rule_spec(Packet,Prec,Spec1,Spec2,Spec3,SpecA,Rulename,interpreted)
     :- flag(Packet,interpreted,interpreted),
        !,
        R =.. [Packet,Prec,Spec1,Spec2,Spec3,SpecA,Rulename],
        call(R).

rule_spec(Packet,Prec,Spec1,Spec2,Spec3,SpecA,Rulename,compiled)
     :- switch1(Packet,Prec,Spec1,Spec2,Spec3,SpecA,Rulename).



                        % Fire the rule

apply_rule(interpreted,Packet,Rulename,B1,B2,B3,Cstack,AS,Pstack,Unseen,DB)
     :- increment(rules_run),
        R =.. [Packet,Rulename,B1,B2,B3,Cstack,AS,Pstack,Unseen,DB],
        call(R).

apply_rule(compiled,Packet,Rulename,B1,B2,B3,Cstack,AS,Pstack,Unseen,DB)
     :- increment(rules_run),
        switch2(Packet,Rulename,B1,B2,B3,Cstack,AS,Pstack,Unseen,DB).



                        % Return top of C stack at end of parse
                        %  This is called explicitly by the final
                        %  grammar rules rather then calling rulematch

alldone([N],DB)
     :- !,
        closenode(N,CN),
        set_tree(DB,CN),
        dbfinish(DB).

        %% this is a hack to do the if,what questions
        %% hack hack hack (rubbing of hands in background)
alldone([S1,S2],DB)
        :-  !,
        attach(S1,S2,s,S3),
        % semantics??
        closenode(S3,CN),
        set_tree(DB,CN),
        pop_sent(DB,DB1),
        dbfinish(DB1).

alldone([N|Others],DB)
     :- ttynl,
        display('C stack not empty at end of parse:'), ttynl,
        portray_stack(pn1,Others), ttynl,
        closenode(N,CN),
        set_tree(DB,CN),
        dbfinish(DB).
```

\\\\\

```
SUBFILE: SWITCH.LPL @21:2 10-APR-1981 <005> (1195)
/* SWITCH .LPL :  Compiled switch tables for packets.

                                                    Lawrence
                                                    Updated: 23 April 81
*/


  :- public switch1/7,
            switch2/10.


  :- mode switch1(+,+,-,-,-,-,?),
          switch2(+,+,+,+,+,+,+,+,+,?).



                          % Pick up spec clause for rule

switch1(ss_start,Prec,B1,B2,B3,A,Name)
     :- ss_start(Prec,B1,B2,B3,A,Name).

switch1(cpool,Prec,B1,B2,B3,A,Name)
     :- cpool(Prec,B1,B2,B3,A,Name).

switch1(npool,Prec,B1,B2,B3,A,Name)
     :- npool(Prec,B1,B2,B3,A,Name).

switch1(parse_det,Prec,B1,B2,B3,A,Name)
     :- parse_det(Prec,B1,B2,B3,A,Name).

switch1(parse_qp_1,Prec,B1,B2,B3,A,Name)
     :- parse_qp_1(Prec,B1,B2,B3,A,Name).

switch1(parse_qp_2,Prec,B1,B2,B3,A,Name)
     :- parse_qp_2(Prec,B1,B2,B3,A,Name).

switch1(parse_adj,Prec,B1,B2,B3,A,Name)
     :- parse_adj(Prec,B1,B2,B3,A,Name).

switch1(parse_noun,Prec,B1,B2,B3,A,Name)
     :- parse_noun(Prec,B1,B2,B3,A,Name).

switch1(np_complete,Prec,B1,B2,B3,A,Name)
     :- np_complete(Prec,B1,B2,B3,A,Name).

switch1(parse_pp,Prec,B1,B2,B3,A,Name)
     :- parse_pp(Prec,B1,B2,B3,A,Name).

switch1(parse_subj,Prec,B1,B2,B3,A,Name)
     :- parse_subj(Prec,B1,B2,B3,A,Name).

switch1(no_subj,Prec,B1,B2,B3,A,Name)
     :- no_subj(Prec,B1,B2,B3,A,Name).

switch1(parse_aux,Prec,B1,B2,B3,A,Name)
     :- parse_aux(Prec,B1,B2,B3,A,Name).
```

```
switch1(build_aux,Prec,B1,B2,B3,A,Name)
     :- build_aux(Prec,B1,B2,B3,A,Name).

switch1(parse_vp,Prec,B1,B2,B3,A,Name)
     :- parse_vp(Prec,B1,B2,B3,A,Name).

switch1(passive,Prec,B1,B2,B3,A,Name)
     :- passive(Prec,B1,B2,B3,A,Name).

switch1(ss_vp,Prec,B1,B2,B3,A,Name)
     :- ss_vp(Prec,B1,B2,B3,A,Name).

switch1(object,Prec,B1,B2,B3,A,Name)
        :- object(Prec,B1,B2,B3,A,Name).

switch1(no_subj,Prec,B1,B2,B3,A,Name)
     :- no_subj(Prec,B1,B2,B3,A,Name).

switch1(that_comp,Prec,B1,B2,B3,A,Name)
     :- that_comp(Prec,B1,B2,B3,A,Name).

switch1(inf_comp,Prec,B1,B2,B3,A,Name)
     :- inf_comp(Prec,B1,B2,B3,A,Name).

switch1(to_less_inf_comp,Prec,B1,B2,B3,A,Name)
     :- to_less_inf_comp(Prec,B1,B2,B3,A,Name).

switch1(to_be_less_inf_comp,Prec,B1,B2,B3,A,Name)
     :- to_be_less_inf_comp(Prec,B1,B2,B3,A,Name).

switch1(two_obj,Prec,B1,B2,B3,A,Name)
     :- two_obj(Prec,B1,B2,B3,A,Name).

switch1(embedded_s_final,Prec,B1,B2,B3,A,Name)
     :- embedded_s_final(Prec,B1,B2,B3,A,Name).

switch1(build_name,Prec,B1,B2,B3,A,Name)
     :- build_name(Prec,B1,B2,B3,A,Name).

..itch1(parse_conj,Prec,B1,B2,B3,A,Name)
     :- parse_conj(Prec,B1,B2,B3,A,Name).

switch1(ss_final,Prec,B1,B2,B3,A,Name)
     :- ss_final(Prec,B1,B2,B3,A,Name).



                        % Call, and run, the rule itself

switch2(ss_start,Name,B1,B2,B3,C,AS,Packets,Unseen,DB)
     :- ss_start(Name,B1,B2,B3,C,AS,Packets,Unseen,DB).

switch2(cpool,Name,B1,B2,B3,C,AS,Packets,Unseen,DB)
     :- cpool(Name,B1,B2,B3,C,AS,Packets,Unseen,DB).

switch2(npool,Name,B1,B2,B3,C,AS,Packets,Unseen,DB)
     :- npool(Name,B1,B2,B3,C,AS,Packets,Unseen,DB).

switch2(parse_det,Name,B1,B2,B3,C,AS,Packets,Unseen,DB)
```

```
     :- parse_det(Name,B1,B2,B3,C,AS,Packets,Unseen,DB).

switch2(parse_qp_1,Name,B1,B2,B3,C,AS,Packets,Unseen,DB)
     :- parse_qp_1(Name,B1,B2,B3,C,AS,Packets,Unseen,DB).

switch2(parse_qp_2,Name,B1,B2,B3,C,AS,Packets,Unseen,DB)
     :- parse_qp_2(Name,B1,B2,B3,C,AS,Packets,Unseen,DB).

switch2(parse_adj,Name,B1,B2,B3,C,AS,Packets,Unseen,DB)
     :- parse_adj(Name,B1,B2,B3,C,AS,Packets,Unseen,DB).

switch2(parse_noun,Name,B1,B2,B3,C,AS,Packets,Unseen,DB)
     :- parse_noun(Name,B1,B2,B3,C,AS,Packets,Unseen,DB).

switch2(np_complete,Name,B1,B2,B3,C,AS,Packets,Unseen,DB)
     :- np_complete(Name,B1,B2,B3,C,AS,Packets,Unseen,DB).

switch2(parse_pp,Name,B1,B2,B3,C,AS,Packets,Unseen,DB)
     :- parse_pp(Name,B1,B2,B3,C,AS,Packets,Unseen,DB).

switch2(parse_subj,Name,B1,B2,B3,C,AS,Packets,Unseen,DB)
     :- parse_subj(Name,B1,B2,B3,C,AS,Packets,Unseen,DB).

switch2(no_subj,Name,B1,B2,B3,C,AS,Packets,Unseen,DB)
     :- no_subj(Name,B1,B2,B3,C,AS,Packets,Unseen,DB).

switch2(parse_aux,Name,B1,B2,B3,C,AS,Packets,Unseen,DB)
     :- parse_aux(Name,B1,B2,B3,C,AS,Packets,Unseen,DB).

switch2(build_aux,Name,B1,B2,B3,C,AS,Packets,Unseen,DB)
     :- build_aux(Name,B1,B2,B3,C,AS,Packets,Unseen,DB).

switch2(parse_vp,Name,B1,B2,B3,C,AS,Packets,Unseen,DB)
     :- parse_vp(Name,B1,B2,B3,C,AS,Packets,Unseen,DB).

switch2(passive,Name,B1,B2,B3,C,AS,Packets,Unseen,DB)
     :- passive(Name,B1,B2,B3,C,AS,Packets,Unseen,DB).

   tch2(ss_vp,Name,B1,B2,B3,C,AS,Packets,Unseen,DB)
     :- ss_vp(Name,B1,B2,B3,C,AS,Packets,Unseen,DB).

switch2(object,Name,B1,B2,B3,C,AS,Packets,Unseen,DB)
        :- object(Name,B1,B2,B3,C,AS,Packets,Unseen,DB).

switch2(no_subj,Name,B1,B2,B3,C,AS,Packets,Unseen,DB)
     :- no_subj(Name,B1,B2,B3,C,AS,Packets,Unseen,DB).

switch2(that_comp,Name,B1,B2,B3,C,AS,Packets,Unseen,DB)
     :- that_comp(Name,B1,B2,B3,C,AS,Packets,Unseen,DB).

switch2(inf_comp,Name,B1,B2,B3,C,AS,Packets,Unseen,DB)
     :- inf_comp(Name,B1,B2,B3,C,AS,Packets,Unseen,DB).

switch2(to_less_inf_comp,Name,B1,B2,B3,C,AS,Packets,Unseen,DB)
     :- to_less_inf_comp(Name,B1,B2,B3,C,AS,Packets,Unseen,DB).

switch2(to_be_less_inf_comp,Name,B1,B2,B3,C,AS,Packets,Unseen,DB)
     :- to_be_less_inf_comp(Name,B1,B2,B3,C,AS,Packets,Unseen,DB).

switch2(two_obj,Name,B1,B2,B3,C,AS,Packets,Unseen,DB)
```

```
     :- two_obj(Name,B1,B2,B3,C,AS,Packets,Unseen,DB).

switch2(embedded_s_final,Name,B1,B2,B3,C,AS,Packets,Unseen,DB)
     :- embedded_s_final(Name,B1,B2,B3,C,AS,Packets,Unseen,DB).

switch2(build_name,Name,B1,B2,B3,C,AS,Packets,Unseen,DB)
     :- build_name(Name,B1,B2,B3,C,AS,Packets,Unseen,DB).

switch2(parse_conj,Name,B1,B2,B3,C,AS,PAckets,Unseen,DB)
     :- parse_conj(Name,B1,B2,B3,C,AS,PAckets,Unseen,DB).

switch2(ss_final,Name,B1,B2,B3,C,AS,Packets,Unseen,DB)
     :- ss_final(Name,B1,B2,B3,C,AS,Packets,Unseen,DB).



\\\\\
```

Chris' modifications

```
!
! Loading the parser
!
.mic set no parameters
.ru rob
*[-'sem.lpl'].
*[-'rulem.lpl'].
*[-'semsup.spl'].
*load([parse_vp,cpool,ss_vp,embedded_s_final,npool,
        ss_final,parse_qp_2,ss_start]).
*[-'util:files.pl'].
*[-padd].
*recorded(stretched,_,P), erase(P).      % Remove nasty definition
*load_dict(dicfix).                      % Load extra dictionary
*[-'semchk'].
*load_sem(semrul).
*:-los.
*:- core_image.
.save parse
.prot parse.exe<005>
.del parse.exe[400,445,parse]
  -n [400,445,parse]=parse.exe
```

```
/* PADD - Extra code for running the parser
   C.M., 15/4/81
*/

/* Do a problem */

run(X) :- input(X), go, output(X).

/* Do a problem from the terminal, outputting to file */

urun(X) :- input(user), go, output(X).

/* Input sentences from a file */

input(File) :- !,
   abolish(nextsentence,1),
   abolish(adatabase,1),
   input_file(File,File1),
   see(File1),
   repeat,
   read_in(S),
   assertz(nextsentence(S)),
   S=[end,.], !,
   seen.

   input_file(user,user) :- !.
   input_file(F,F2) :-
      name(F,Na1),
      conc12(Na1,".sen",Na2),
      name(F1,Na2),
      i_1(F1,F2).

   i_1(F,F) :- file_exists(F), !.
   i_1(F1,F2) :-
      name(F1,Na1),
      conc12("sen:",Na1,Na2),
      name(F2,Na2).

/* Process the sentences read in */

 r :- nextsentence(S), S \== [end,.],
   write('Parsing: '), write(S), nl, nl,
   convert_wordlist(S,Nodelist),
   try_parse(Nodelist,Time,ANS),
   showtime(Time),
   set_tree(ANS,Tree),
   print_tree(Tree), ttynl, ttynl, ttynl,
   assertz(adatabase(ANS)),
   fail.
go.

/* Output databases to a file */

output(File) :-
   output_file(File,File1),
   tell(File1),
   retract(adatabase(Db)),
   print_db(Db),
   fail.
output(_) :- nl, told.
```

```
   output_file(user,user) :- !.
   output_file(F1,F2) :-
      name(F1,Na1),
      conc12("syn:",Na1,Na2),
      conc12(Na2,".syn",Na3),
      name(F2,Na3).

/* "Append" */

conc12([],X,X) :- !.
conc12([A|B],C,[A|D]) :- !, append(B,C,D).
```

```
SSTART.PK :      Packet SS_START

rule IF_WHAT?:          [binder] -> then make a hypothetical sentence
rule WH_QUEST:          [wh] -> attach 1st as wh_comp, wh_quest
rule ADVERB:            [adverb][nsstart] -> attach 1st as adverb,
rule MAJOR_DECL_S:      [np][verb] -> label c decl, major,
rule AUX_INVERT:        [auxverb][nsstart] -> push aux onto AS
rule NP_PP_DEFAULT:     [np][pp] -> attach to np as PP
rule NP_UTTERANCE:      [np][fpunc] -> done,
rule PP_UTTERANCE:      [pp][fpunc] -> done
rule IMPERATIVE:        [tnsless] -> insert you into the buffer
rule FRONTED_PP:        [pp] -> attach to C
rule WH_NP:             [wh] -> attach to c, for wh_quest
rule Kissing_Aunts:     [verb,ing,adj][noun,npl] -> np,vp

CPOOL.PK :       Packet CPOOL

rule X_AND_X:           [x][conj][x] ->x conjoined
rule POSS_DET:          [poss_np] -> make a det and drop,
rule SO_THAT:           [so,such][that] -> that-as-a-comp
rule PROPNAME:          [name, not np] -> new np node,
rule PROPNOUN:          [propnoun] -> np in 1st buffer
rule PP:                [prep][nsstart]-> B1 <- PP,  attach 2nd to c as prep
rule MARKED_STARTNP:    [det, agree_det] -> start a new np node
rule STARTNP:           [nsstart] -> start a new NP node
rule THAN_COMP:         [than_comp][np] -> attach B2 to B1 as compartive
rule AND:               [conj] -> stuff onto active stack
rule COMP_TO_NP:        [comp_s] -> make an np in B1
rule NP_PP:             [pp] -> consider to attach the pp to the np
rule PRONOUN:           [pronoun] -> attach to c, fix feats
rule VP_ATTACH:         [vp] -> attach to s
rule POSS_NP:           [poss] -> attach as poss

NPOOL.PK :       Packet NPOOL

rule QP_AND_QUANT:      [qp][and][quant] -> new qp node on c, attach B1 and B2
rule LONGER_THAN:       [than][name] -> make a comparative
rule 3 FTSEC:           [qp][units] -> new qp in B1
rule FT_LONG:           [qp][adj] -> new ap, attach as adj
rule NOUN_QP:           [quant] -> new qp node
rule AP_ATTACH:         [ap] -> attach to c as AP
rule QP_ATTACH:         [qp] -> attach to c as qp

PDET.PK :        Packet PARSE_DET

rule DETERMINER:        [det] -> attach

PQP1.PK :        Packet PARSE_QP_1

rule HOW_MANY:          [how][adj] -> combine into how only
rule QUANT:             [quant] -> new qp node in B1
rule NEXT_WEEK:         [ord][noun,time] -> qp node and ord
rule ALL_THE:           [all][det,def] -> insert "of" into B2
rule QUANTIFIER:        [quantifier] ->attach and transfer feats
```

```
rule QUANT_DONE:          [t] -> change packets

-------------------------------------------------------------------------------
PQP2.PK :         Packet PARSE_QP_2

rule DET_QUANT:           [quant,det or num] -> new QP node
rule ORDINAL:             [ord] -> new ordQP node, ect
rule DET_QUANT_DONE:      [t] -> redo packets.

-------------------------------------------------------------------------------
PADJ.PK :         Packet PARSE_ADJ

rule ADJ_GROUP:           [adj][adj # noun # dim] -> attach as adj
rule ADJ_NP:              [adj] -> attach as adj
rule ADJ_DONE:            [t] -> change packets

-------------------------------------------------------------------------------
PNOUN.PK :        Packet PARSE_NOUN

rule COMPLEX_NOUN:        [noun][noun] -> attach 1st to c
rule NOUNS:               [noun, np1] -> attach to c if past test
rule NOUN:                [noun] -> attach to c as noun
rule NP_BUILT:            [t] ->change packet to NP_complete

-------------------------------------------------------------------------------
NPCOM.PK :        Packet NP_COMPLETE

rule QP_PP:               [QP][Prep] -> start a PP
rule PP:                  [Prep][nsstart]-> B1 <- PP,  attach 2nd to c as Prep
rule REDUCED_RELATIVE:    c is NP, [verb,ing ] -> insert wh_ into B1
rule REL_ATTACH:          [relative] -> attach to c,
rule RELPRON_NP:          [relpron] -> np in B1
rule WH_RELATIVE_CLAUSE:  [relpron_np] ->
rule NP_PP:               [PP] -> consider to attach the PP to the NP
rule AND:                 [conj] -> stuff onto active stack
rule COMMA:               [comma] -> run np_done next, for now
rule TOM_MARY:            [nsstart] -> insert wh_
rule NP_DONE:             [t] -> drop c,
rule OF_PP:               [of][noun] -> add nsstart to second to force PP

-------------------------------------------------------------------------------
ARPP.PK :         Packet PARSE_PP

rule ATTACH_PREP:         [Prep] -> attach to C
rule PP_NP:               [np] -> attach to PP, drop PP
rule WITH_WHICH:          [wh] -> wh_np built, temp patch, not thought out

-------------------------------------------------------------------------------
PSUBJ.PK :        Packet PARSE_SUBJ

rule UNMARKED_ORDER:      [np][verb] -> attach 1st to s
rule AUX_INVERSION:       [aux][np] -> attach B2

-------------------------------------------------------------------------------
BLDAUX.PK :       Packet BUILD_AUX

rule NEG:                 [neg][verb] -> attach 1st to c as neg
rule MODAL:               [modal][tnsless] -> attach 1st to c as modal
rule PERFECTIVE:          [have][en] -> attach 1st to c as perf
rule PASSIVE_AUX:         [be][en] -> attach 1st to c as passive,
```

```
rule PROGRESSIVE:       [be][ing] -> attach 1st as prog
rule DO_SUPPORT:        [do][tnsless] -> attach 1st as do
rule HAVE_TO:           [have or be][to] -> attach 1st
rule TO_BE:             [to][tnsless] -> attach to
rule AUX_ADVERB:        [adverb] -> attach to aux
rule AUX_COMPLETE:      [t] -> drop c.
rule BE_PRED:           [be][prep or adj] -> attach as copula

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

PAUX.PK :       Packet PARSE_AUX

rule TO_INFINITIVE:     [to,auxverb][tnsless] -> new aux node.
rule START_AUX:         [verb] ->create new AUX node, etc
rule AUX_ATTACH:        [aux] -> attach to s, change packets.

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

PVP.PK :        Packet PARSE_VP

rule PREDP:             [pp] -> attach
rule MAIN_VERB:         [verb] -> do everything.

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

'ASSIV.PK :     Packet PASSIVE

rule PASSIVE:           [t] -> create trace

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

SSVP.PK :       Packet SS_VP

rule ADVERB_GROUP:      [adverb][adverb] -> compound adverb
rule ADVERB:            [adverb] -> attach as adverb
rule PART:              [particle] -> attach to verb
rule PP_UNDER_VP_1:     [pp] -> attach to c
rule PART:              [particle] -> attach to verb
rule VP_DONE:           [t] -> drop c.

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

OBJ.PK :        Packet OBJECT

rule OBJECT:            [to,auxverb][tnsless] -> attach object

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

NOSUBJ.PK :     Packet NO_SUBJ

rule CREATE_DELTA_SUBJECT: [to,auxverb][tnsless] ->

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

THATC.PK :      Packet THAT_COMP

rule THAT_S_START_1:    [np][verb] -> embedded sentence
rule THAT_S:            [that] -> start an S bar

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

INFC.PK :       Packet INF_COMP

rule INF_S_START1:      [np][to,auxverb][tnsless] ->
rule FOR_S_BAR:         [for,pp][to] -> Sbar

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

TLICOM.PK :     Packet TO_LESS_INF_COMP
```

```
rule UNMARKED_SUBJ:     [np][tnsless] ->  for see, embedded sentence
```

```
TBLCOM.PK :     Packet TO_BE_LESS_INF_COMP

rule INSERT_TO_BE:      [np][en or adj] -> insert to be into the buffer
rule INSERT_TO_BE_1:    [en or adj] -> insert to be into the buffer
```

```
TWOBJ.PK :      Packet TWO_OBJ

rule FIRST_OBJ:         [to,auxverb][tnsless] ->  attach first object
```

```
EMBSFI.PK :     Packet EMBEDDED_S_FINAL

rule PP_UNDER_S_2:      [pp] -> attach
rule EMBEDDED_S_DONE:   [t] -> drop c.
```

```
"NAME.PK :      Packet BUILD_NAME

rule  NAME:             [name] -> attach to c.
rule END_OF_NAME:       [t] -> run np_done next
```

```
PCONJ.PK :      Packet PARSE_CONJ

rule DROP_AND:          [c has and] -> drop B1 and "and" into buffers
```

```
SSFIN.PK :      Packet SS_FINAL

rule PP_UNDER_S_1:      [pp] attach to c
rule S_DONE:            [finalpunc] -> attach and end.
rule INIT_S_BAR:        [verb] -> drop as a NP
rule CONJOINED_S:       [comma][conj or binder] -> make into a conjoined S
rule HYPO_S:            [comma] -> then an if, what sentence is assumed.
```

```
.MBSVP.PK :     Packet EMBEDDED_S_VP

rule OBJ_IN_EMBEDDED_S: [np] -> attach to c as np
rule PP_UNDER_VP_2:     [pp] -> attach, semantics left out
rule EMBEDDED_VP_DONE:  [t] -> drop c.
```

```
/* Fixes to the parser dictionary */

def(other,adjf).
def(stretch,verbonly).
def(when,[conj]).
def(while,[conj]).
def(uniform,adjf).
```

```
/* DICT.SRT
   Sorted version of Rob's dictionary
   C.M., 10/9/81
   Otherwise unchanged
*/

def(!,[fpunc]).
def('''',[possesive,poss]).
def('''s',[possesive,poss]).
def(',',[comma]).
def(.,[fpunc]).
def(?,[fpunc]).
def('1/3',quantity).
def('1/5',quantity).
def('2e',vari).
def(a,detindef,[variable]).
def(about,[prep]).
def(above,[prep]).
def(accelerate,verbf).
def(acceleration,dimf).
def(act,verbf).
 -f(add,verbf).
  f(after,[prep]).
def(again,adverbf).
def(against,[prep]).
def(age,nounf).
def(ahead,[prep]).
def(al,nounname).
def(alfred,nounname).
def(all,quantifierf,[all,indef,npl]).
def(along,[prep]).
def(am,amaux).
def(an,detindef).
def(analyze,verbonly).
def(and,[conj]).
def(angle,nounf).
def(another,quantifierf,[ns]).
def(apart,adverbf).
def(apparent,adjf).
def(application,verbf).
 -f(apply,verbf).
  f(are,aux2,[pres]).
def(arm,nounf).
def(arrive,verbonly,[en]).
def(as,[prep]).
def(ask,verb3).
def(at,[prep]).
def(attach,verbf).
def(attend,verbonly).
def(aunt,nouns).
def(automatically,adverb).
def(automobile,nounf).
def(away,adverbf,[prep]).
def(b,vari).
def('b-c',vari).
def(bad,adjf).
def(balance,verbf).
def(ball,nounf).
def(bar,nounf).
def(barn,nounf).
```

```
def(be,bef).
def(been,auxp,[en,be,be_]).
def(before,[prep]).
def(behind,[prep]).
def(believe,verbonly,[that_comp,inf_comp]).
def(below,[prep]).
def(between,[prep]).
def(big,adjf).
def(block,nounf).
def(blue,adjf).
def(boat,nounf).
def(book,nounf).
def(bore,verbf).
def(born,verbp).
def(boston,nounplace).
def(both,quantifierf,[npl]).
def(boy,nouns).
def(break,verbf).
def(bridge,nounf).
def(bright,adjf).
def(broke,verbp).
  f(brother,nouns).
  f(brought,verbp).
def(build,verbf).
def(buildings,[noun,ns,n3p,verb,pres,ing,adj,v_3s]).
def(bus,nounf).
def(by,[prep]).
def(c,vari).
def(calculate,verbf).
def(came,verbp).
def(can,modalf,[pres,noun,ns,verb]).
def(car,nounf).
def(carry,verbf).
def(cat,nounf).
def(catch,nounf).
def(ceiling,nounf).
def(center,nounf).
def(change,verbonly,[inf_comp]).
def(chris,nounname).
def(cliff,nounf).
  f(cm,unitf).
  f(coefficient,nounf).
def(collide,verbf).
def(come,verbonly,[en]).
def(common,adjf).
def(compute,verbf).
def(connect,verbf).
def(constant,nounf).
def(cord,nounf).
def(could,modalf,[future]).
def(couple,verbf).
def(cover,verbf).
def(crane,nounf).
def(cube,nounf).
def(cute,adjf).
def(d,vari).
def(decelerate,verbf).
def(deer,nounf,[npl]).
def(degree,unitf).
def(deliver,verbonly).
```

```
def(denver,nounplace).
def(destroy,verbf).
def(determine,verbf).
def(did,auxp,[vspl,do]).
def(direction,nounf).
def(direct,adjf).
def(distance,nounf).
def(do,auxpres,[tnsless,v_3s,do]).
def(does,aux3,[do]).
def(dos,nounf).
def(done,auxp,[en,do]).
def(door,nounf).
def(down,[prep]).
def(downward,adjf,[adverb]).
def(driver,nounf).
def(drop,verbf,[two_obj]).
def(due,adjf).
def(e,vari).
def(each,quantifierf,[ns]).
def(earth,nounf).
def(east,nounf).
  f(eat,verbf).
  f(edge,nounf).
def(edinburgh,nounplace).
def(effective,adjf).
def(elapse,verbf).
def(elasticity,[noun,ns,ngstart,n3p]).
def(elastic,adjf).
def(elephant,nounf).
def(elesticity,adjf).
def(end,nounf).
def(end,nounf).
def(engine,nounf).
def(equal,adjf).
def(equilibrium,nounf).
def(every,detindef,[ns,quantifier,adverb]).
def(exam,nounf).
def(exert,verbonly,[inf_comp,no_subj]).
def(extend,verbf).
def(extension,nounf).
  f(fact,nounf).
  f(fall,verbf).
def(far,[prep]).
def(fat,adjf).
def(fell,[verb,past,vspl]).
def(final,adjf).
def(find,verbf,[inf_comp,that_comp,two_obj]).
def(fine,adjf).
def(first,ordf).
def(fish,verbonly,[noun,ns,npl,ngstart]).
def(fixed,adjf).
def(floor,nounf).
def(fly,verbf).
def(for,[prep,for]).
def(force,nounf).
def(former,nounf).
def(found,verb4).
def(fox,nounf).
def(free,adjf).
def(frictionless,adjf).
```

```
def(friction,nounf).
def(frictional,adjf).
def(friday,nounf,[time]).
def(from,[prep]).
def(ft,unitf).
def('ft/sec',unitf).
def(g,unitf).
def(gain,verbf).
def(gave,verbp,[two_obj]).
def(george,nounname).
def(girl,nouns).
def(give,verbonly).
def(glasgow,nounplace).
def(gm,unitf).
def(go,verbf,[inf_comp]).
def(gone,verbonly,[en]).
def(good,adjf).
def(gravity,nounf).
def(greatest,adjf).
def(green,adjf).
def(ground,nounf).
  f(gun,nounf).
   f(had,auxhave,[past,vspl,ns]).
def(half,vari).
def('half-way',quantity,[ns]).
def(hammer,nounf).
def(hang,verbf).
def(happy,adjf).
def(hard,adverbf).
def(harsh,adverbf).
def(has,auxhave,[pres,v3s]).
def(hat,nounf).
def(have,auxhave,[tnsless,pres,v_3s]).
def(having,auxhave,[pres,ing]).
def(he,pronounf,[ns]).
def(head,nounf).
def(heavy,adjf).
def(height,nounf).
def(her,posspn,[n3p]).
def(here,adjf).
  f(herself,pronounf,[ns]).
 ef(high,adjf).
def(hill,nounf).
def(him,pronounf,[ns]).
def(himself,pronounf,[ns]).
def(hinge,nounf).
def(his,posspn,[n3p]).
def(hit,verb1,[two_obj]).
def(hook,nounf).
def(horizontal,nounf,[adj]).
def(horse,nounf).
def(how,whpronn,[how,ngstart]).
def(hung,verbp).
def(i,pronounI).
def(if,[prep,binder]).
def(impact,nounf).
def(impulse,verbf).
def(in,[prep,unit]).
def(inch,unitf).
def(incline,nounf).
```

```
def(inextensible,adjf).
def(initial,adjf,[ord]).
def(into,[prep]).
def(invite,verb3,[noun]).
def(is,aux3,[be,sent_subj]).
def(it,pronounf,[ns]).
def(its,posspn,[n3p]).
def(jack,nounname).
def(jeep,nounf).
def(jill,nounname).
def(john,nounname).
def(joule,unitf).
def(journey,nounf).
def(judy,nounname).
def(jump,nounf).
def(just,adverbf).
def(karen,nounname).
def(kg,unitf).
def(kilowatt,unitf,[noun,ns,n3p,ngstart]).
def(kiss,verbf).
def('km/h',unitf).
 -f('kmh-1',unitf).
  f(knew,verbp,[inf_comp,that_comp]).
def(know,verbonly,[that_comp,inf_comp]).
def(known,verbonly,[inf_comp,that_comp]).
def(kw,unitf).
def(l,vari).
def(l1,vari).
def(l2,vari).
def(last,ordf).
def(latter,nounf).
def(lb,unitf).
def('lb/ft',unitf).
def(leave,verbf,[two_obj]).
def(left,verbp,[adj]).
def(leg,nounf).
def(length,dimf).
def(leslie,nounname).
def(level,nounf,[adj]).
def(lever,nounf).
 -f(lift,verbf).
  f(light,adjf).
def(likely,verbonly,[inf_comp]).
def(line,nounf).
def(little,adjf).
def(load,nounf).
def(locate,verbf).
def(lollipop,nounf).
def(london,nounplace).
def(long,adjf).
def(look,verbf).
def(lorry,nounf).
def(loud,adverbf).
def(lower,nounf).
def(m,[vari,unit]).
def(m1,vari).
def(m2,vari).
def(magnitude,nounf).
def(maintain,verbf).
def(make,verbf,[to_less_inf_comp]).
```

```
def(man,nounf).
def(manage,verbf).
def(map,nounf).
def(mary,nounname).
def(mass,dimf).
def(masses,dimf,[npl]).
def(maximum,adjf).
def(me,pronounI).
def(meet,verbf).
def(meeting,nounf,[ing,part]).
def(men,nouni).
def(met,verbp).
def(meter,unitf).
def(metre,unitf).
def(ms,unitf).
def(might,modal,[future]).
def(mile,unitf).
def(milk,verbonly,[noun,ns,npl]).
def(milne,nounname).
def(mine,posspn,[n1p]).
def(modulus,nounf).
def(monday,nounf,[time]).
  f(mother,nounf).
def(motion,nounf).
def(move,verbf).
def('ms-1',unitf).
def('ms-2',unitf).
def(much,adverbf).
def(must,modalf).
def(muzzle,nounf).
def(my,posspn,[n1p]).
def(n,unitf).
def(nail,nounf).
def(natural,adjf).
def(neglisible,adjf).
def(newton,unitf).
def(next,ordf).
def(nice,adjf).
def(no,[neg,en,ing,tnsless]).
def(none,quantifierf,[ns,npl]).
  f(north,nounf).
  f(not,[neg,en,ing,tnsless]).
def(nt,unitf).
def(object,nounf).
def(of,[prep,of]).
def(old,adjf).
def(on,[prep]).
def(one,[quant,nsstart,ns]).
def(or,[conj]).
def(other,ordf).
def(our,posspn,[n1p]).
def(out,[prep]).
def(over,[prep]).
def(p,vari).
def(painter,nounf).
def(pam,nounname).
def(paper,nounf).
def(park,nounf).
def(parser,nounf).
def(particle,nounf).
```

```
def(pass,verbf).
def(past,[prep,adj]).
def(pat,nounname).
def(pes,nounf).
def(peir,nounf).
def(pencil,nounf).
def(persuade,verb3).
def(pier,nounf).
def(pin,nounf).
def(pipe,nounf).
def(place,verbf).
def(plane,nounf).
def(point,nounf).
def(pole,nounf).
def(position,nounf).
def(pound,unitf,[noun,verb,tnsless]).
def(power,nounf).
def(projection,nounf).
def(project,verbf).
def(promise,verb3).
def(pryamid,nounf).
def(pull,verbf).
  f(pulley,nounf).
def(pulley,nounf).
def(quick,adverbf).
def(quiet,adverbf).
def(race,verbf).
def(radius,nounf).
def(raise,verbf).
def(ran,verbp).
def(ranse,nounf).
def(rape,verbf).
def(reach,verbf).
def(reach,verbf).
def(reache,verbf).
def(red,adjf).
def(release,verbf).
def(remain,verbonly).
def(resistance,nounf).
def(respective,adj,[adverb]).
def(rest,nounf).
  f(restitution,adjf).
def(right,adjf).
def(road,nounf).
def(rob,nounname).
def(robot,nounf).
def(rock,nounf).
def(rod,nounf).
def(room,nounf).
def(rope,nounf).
def(roush,adjf).
def(round,adjf).
def(run,verbf).
def(s,unitf).
def(sad,adjf).
def(said,verb4).
def(same,ordf).
def(saturday,nounf,[time]).
def(saw,verb2,[past,vspl]).
def(say,verb3).
```

```
def(scaffold,nounf).
def(schedule,verbf,[inf_comp]).
def(scotland,nounplace).
def(sea,nounf).
def(sec,unitf).
def(second,unitf,[time,noun,ord]).
def(see,verb2,[tnsless,v_3s]).
def(seem,verbonly,[that_comp,to_be_less_inf_comp,no_subj,inf_comp,sent_subj]).
def(seen,verb4).
def(seesaw,nounf).
def(she,pronounf,[ns]).
def(shell,nounf).
def(shoe,nounf).
def(shoot,verbf).
def(short,adjf).
def(shot,verbp,[en]).
def(should,modalf,[past]).
def(show,verbf).
def(shown,verb4).
def(sit,verbf).
def(sleep,verbf).
  f(slope,nounf).
  f(slow,adverbf).
def(small,adjf).
def(smart,adjf).
def(smooth,adjf).
def(so,[compadv]).
def(soft,adverbf).
def(some,quantifierf,[npl]).
def(speed,dimf).
def(sphere,nounf).
def(spring,nounf).
def(stand,nounf).
def(start,nounf).
def(stationary,adjf).
def(station,nounf).
def(statue,nounf).
def(steve,nounname).
def(stir,verbf).
def(stone,unitf,[verb,tnsless,v_3s,noun]).
  f(stop,nounf).
  f(straight,adjf).
def(street,nounf).
def(stretched,verbp,[en,adj]).
def(strike,verbf).
def(string,nounf).
def(student,nouns).
def(stupid,adjf).
def(succesive,adjf).
def(such,[compadv]).
def(sue,nounname).
def(sun,nounf).
def(support,verbf).
def(surface,nounf).
def(surprise,verbf).
def(suspend,verbf).
def(system,nounf).
def(table,nounf).
def(take,verbonly,[that_comp,inf_comp,no_subj,two_obj]).
def(taken,verbp,[that_comp,inf_comp,no_subj,two_obj]).
```

```
def(tall,adjf).
def(tanned,adjf).
def(tapper,verbf).
def(taunt,adjf).
def(tea,nounf).
def(tell,verb3,[two_obj]).
def(ten,vari).
def(tension,dimf).
def(term,nounf).
def(than,[than]).
def(that,[det,nsstart,def,comp,pronoun,ns,that]).
def(the,detdef,[ns,npl]).
def(their,posspn,[n3p]).
def(theirselves,pronounf,[npl]).
def(them,pronounf,[npl]).
def(there,nounplace).
def(these,detdef,[npl]).
def(they,pronounf,[npl]).
def(thick,adjf).
def(thin,adjf).
def(this,detdef,[ns,nlp,pronoun]).
def(those,detdef,[npl]).
  f(three,quantity).
def(through,[prep]).
def(throw,verbf).
def(thrown,verbp,[en]).
def(thursday,nounf,[time]).
def(time,unitf,[noun,ns,verb,v_3s]).
def(tiny,adjf).
def(to,[prep,verb,pres,to]).
def(together,adverb).
def(told,verb4,[two_obj]).
def(tom,nounname).
def(tommorrow,adverbf,[adverb]).
def(ton,unitf).
def(took,verbp,[inf_comp,two_obj]).
def(top,nounf).
def(tower,nounf).
def(toy,nounf).
def(track,nounf).
  f(train,nounf).
  f(trash,nounf).
def(travel,verbf).
def(truck,nounf).
def(tuesday,nounf,[time]).
def(twice,quantity).
def(two,quantity).
def(u,vari).
def(ugly,adjf).
def(uncle,nouns).
def(uniform,nounf).
def(unstretch,adjf).
def(until,[prep]).
def(up,[prep]).
def(upper,nounf).
def(upward,adjf,[adverb]).
def(v,vari).
def(val,nounname).
def(value,nounf).
def(velocity,dimf).
```

```
def(vertical,nounf).
def(w,vari).
def(walk,verbf).
def(wall,nounf).
def(want,verbonly,[inf_comp,no_subj]).
def(was,auxp,[v13s,be]).
def(watch,nounf).
def(we,pronoun1).
def(wednesday,nounf,[time]).
def(wee,adjf).
def(week,nounf,[time]).
def(weish,verbf).
def(weishtless,adjf).
def(weisht,dimf).
def(went,verbp).
def(went,verbp).
def(were,aux2,[past]).
def(wh_,wh_).
def(what,[det,nsstart,ns,npl,n3p,indef,wh,relpron]).
def(when,whpronn,[ns]).
def(where,whpronn,[ns]).
def(which,whpronn,[det,ns,npl,indef,nsstart]).
def(while,whpronn,[ns,npl,binder]).
def(white,adjf).
def(who,whpronn,[npl,ns]).
def(whom,whpronn,[npl,ns]).
def(wide,adjf).
def(will,modalf,[future,noun,ns]).
def(wish,verbf).
def(with,[prep]).
def(woman,nouns).
def(wonder,verbonly,[inf_comp,that_comp]).
def(wood,adjf).
def(work,verbf).
def(worm,nounf).
def(would,modalf,[past]).
def(x1,vari).
def(x2,vari).
def(x3,vari).
def(yd,unitf).
def(year,unitf,[noun,n3p]).
def(yellow,adjf).
def(yesterday,adverbf,[adverb]).
def(you,pronoun2,[npl]).
def(young,adjf).
def(your,posspn,[n2p]).
def(zero,quantity).
```

```
/* SSTART.PK : Packet   SS_START
                assumes C is sentence start

                                                    Rob
                                                    Updated: 1 June 81 (R)
*/


  :- mode ss_start(+,-,-,-,-,?).
  :- mode ss_start(+,+,+,+,+,+,+,+,?).

/*-----------------------------------------------------------------------*/

/* rule IF_WHAT?: [binder] -> then make a hypothetical sentence */

ss_start(5, (binder), t, t, t, if_what).

  ss_start(if_what,B1,B2,B3,[C|TL],AS,[APacks|Packets],[U1|Unseen],DBold)  :-
        new_node(s,[binder],S1),
        push_sent(S1,DBold,DB),
        attach(B1,S1,binder,S2),          % if_what semantics abolished - CSM
        !, rulematch(B2,B3,U1,[C,S2|TL],Rulematch,AS,
                        [APacks,[cpool,ss_start]|Packets],Unseen,DB).

/** rule WH_QUEST: [wh] -> attach 1st as wh_comp, wh_quest      */

ss_start(10, (wh & (np # pp # ap)), t, t, t, wh_quest).

  ss_start(wh_quest,B1,B2,B3,[C|TL],AS,[APacks|Packets],[U1|Unseen],DB):-
        addfeats(C,[major,wh_quest],C1),
        deactivate(ss_start,APacks,P1),
        activate(parse_subj,P1,P2),
        attach(B1,C1,wh_comp,C2),
        semantics(wh_quest,DB,B1),
        (B2 has verb & not(auxverb), !,
        (new_node(np,[trace],B11),
        semantics(start_np,DB,B11),
        semantics(trace,DB,B11),
        semantics(bind_trace,DB,B1),
        !, rulematch(B11,B2,B3,[C2|TL],Rulematch,AS,[P2|Packets],
                                                    [U1|Unseen],DB));
        ( !,
        rulematch(B2,B3,U1,[C2|TL],Rulematch,[[wh_comp,B1]|AS],[P2|Packets],Unsee

/** rule ADVERB: [adverb][nsstart] -> attach 1st as adverb.   */
                % added by Karen Archbold April 1981

ss_start(10,(adverb),(adverb # nsstart),t,t,adverb).

 ss_start(adverb,B1,B2,B3,[C|TL],AS,Packets,[U1|Unseen],DB):-
        semantics(adverb,DB,B1),
        attach(B1,C,adverb,C1),
        !, rulematch(B2,B3,U1,[C1|TL],Rulematch,AS,Packets,Unseen,DB).

/** rule MAJOR_DECL_S: [np][verb] -> label c decl, major,
                change packets  */

ss_start(10, (np), (verb), t, t, major_decl_s).

  ss_start(major_decl_s,B1,B2,B3,[C|TL],AS,[APacks|Packets],Unseen,DB) :-
```

```
        addfeats(C,[s,decl,major],C1),
        deactivate(ss_start,APacks,P1),
        activate(parse_subj,P1,P2),
        semantics(major_decl_s,DB),
        !, rulematch(B1,B2,B3,[C1|TL],Rulematch,AS,[P2|Packets],Unseen,DB).

/* rule AUX_INVERT: [auxverb][nsstart] -> push aux onto AS       */

ss_start(10, (auxverb), (nsstart), t, t, aux_invert).

  ss_start(aux_invert,B1,B2,B3,[C|TL],AS,[APacks|Packets],[U1|Unseen],DB) :-
        addfeats(C,[s,ynquest,major],C1),
        deactivate(ss_start,APacks,P1),
        activate(parse_subj,P1,P2),
        semantics(aux_invert,DB),
        !, rulematch(B2,B3,U1,[C1|TL],Rulematch,[[aux,B1]|AS],
                                        [P2|Packets],Unseen,DB).

/** rule NP_PP_DEFAULT: [np][pp] -> attach to np as PP  */
/* only when clause initial      */

   start(10, (np), (pp), t, t, np_pp_default).

  ss_start(np_pp_default,B1,B2,B3,C,AS,Packets,[U1|Unseen],DB):-
        attach(B2,B1,pp,B12),
        semantics(np_pp_default,DB,B1,B2),
        !, rulematch(B12,B3,U1,C,Rulematch,AS,Packets,Unseen,DB).

/** rule NP_UTTERANCE: [np][fpunc] -> done.      */

ss_start(10, (np), (fpunc), t, t, np_utterance).

  ss_start(np_utterance,B1,B2,B3,[C|TL],AS,Packets,Unseen,DBold):-
        addfeats(C,[major,np_utterance],C1),
        semantics(utterance,DBold,B1),
        attach(B1,C1,np,C2),
        attach(B2,C2,fpunc,C3),
        pop_sent(DBold,DB),
        alldone([C3|TL],DB).

  * rule PP_UTTERANCE: [pp][fpunc] -> done       */

ss_start(10, (pp), (fpunc), t, t, pp_utterance).

  ss_start(pp_utterance,B1,B2,B3,[C|TL],AS,Packets,Unseen,DBold) :-
        addfeats(C,[major,np_utterance],C1),            % was pp_utterance
        semantics(utterance,DBold,B1),
        attach(B1,C1,pp,C2),
        attach(B2,C2,fpunc,C3),
        pop_sent(DBold,DB),
        alldone([C3|TL],DB).


/** rule IMPERATIVE: [tnsless] -> insert you into the buffer     */
/* lexical ambiguity should also make ist a verb         */
% doesn't work for have
ss_start(10, (tnsless), t, t, t, imperative).

  ss_start(imperative,B1,B2,B3,[C|TL],AS,[APacks|Packets],Unseen,DB) :-
        coerce(verb,B1,B11),
```

```
        addfeats(C,[s,imperative,major],C1),
        deactivate(ss_start,APacks,P1),
        activate(parse_subj,P1,P2),
        lookup(you,U2),
        semantics(imperative,DB),
        !, rulematch(U2,B11,B2,[C1|TL],Rulematch,AS,[P2|Packets],
                [B3|Unseen],DB).


/* rule FRONTED_PP: [pp] -> attach to C */
        % should set AS'd and recovered later or trace in

ss_start(10, (pp), t, t, t, fronted_pp).

  ss_start(fronted_pp,B1,B2,B3,[C|TL],AS,Packets,[U1|Unseen],DB) :-
        attach(B1,C,pp,C1),
        semantics(pp_under_x,DB,B1),
        !, rulematch(B2,B3,U1,[C1|TL],Rulematch,AS,Packets,Unseen,DB).



   rule WH_NP: [wh] -> attach to c, for wh_quest        */

ss_start(15, (wh), t, t, t, wh_np).

  ss_start(wh_np,B1,B2,B3,[C|TL],AS,Packets,Unseen,DB) :-
        new_node(np,B11),
        semantics(start_np,DB,B11),
        attach(B1,B11,wh_comp,B12),
        semantics(wh_np,DB,B11,B1),
        !, rulematch(B12,B2,B3,[C|TL],Rulematch,AS,Packets,Unseen,DB).

/* rule Kissing_Aunts: [verb,ing,adj][noun,np]] -> np,vp         */
        % This rule is a HACK HACK HACK only

ss_start(10, (verb & ing & adj), (noun & np1), t, t, kissing).

  ss_start(kissing,B1,B2,B3,C,AS,Packets,[U1|Unseen],DB) :-
        new_node(np,[ap,ns,np1],B11),
        attach(B1,B11,verb,B12),
        attach(B2,B12,verb,B13),
        !, rulematch(B13,B3,U1,C,Rulematch,AS,Packets,Unseen,DB).
```

```
/* CPOOL.PK : Packet    CPOOL
              assumes C is anything

                                                    Rob
                                                    Updated: 24 November 80 (R)
*/


  :- mode cpool(+,-,-,-,-,?).
  :- mode cpool(+,+,+,+,+,+,+,+,?).

/*-----------------------------------------------------------------------*/

/* rule X_AND_X: [x][conj][x] ->x conjoined      */ % Sem_chk added by CSM

cpool(5, t, (conj), t, agree_13(and_type)&sem_chk(and), x_and_x).

  cpool(x_and_x,B1,B2,B3,C,AS,Packets,[U1,U2|Unseen],DB) :-
        same_node_type(B1,B3,Feat),
        new_node(Feat,B11),
        attach(B1,B11,Feat,B12),
        attach(B2,B12,conj,B13),
        attach(B3,B13,Feat,B14),
        semantics(conj,DB,B11,B1,B2,B3),
        !, rulematch(B14,U1,U2,C,Rulematch,AS,Packets,Unseen,DB).

/** rule POSS_DET: [poss_np] -> make a det and drop.     */

cpool(5, (poss_np), t, t, agree(det), poss_det).

  cpool(poss_det,B1,B2,B3,C,AS,Packets,[U1|Unseen],DB) :-
        new_node(det,[nsstart,ns,npl],B31),      % should fix this number stuff
        semantics(poss_det,DB,B31,B1),
        attach(B1,B31,np,B32),
        !, rulematch(B32,B2,B3,C,Rulematch,AS,Packets,[U1|Unseen],DB).

/** rule SO_THAT: [so,such][that] -> that-as-a-comp      */

cpool(10, (compadv), (that), t, t, so_that).

  cpool(so_that,B1,B2,B3,C,AS,Packets,Unseen,DB) :-
        new_node(comp,[that,binder],B11),
        semantics(so_that,DB,B1,B2),
        attach(B1,B11,compadv,B12),
        attach(B2,B12,that,B13),
        lookup(',',U1),
        !, rulematch(U1,B13,B3,C,Rulematch,AS,Packets,Unseen,DB).

/** rule PROPNAME: [name, not np] -> new np node.        */

cpool(10, (name & not(np)), t, t, t, propname).

  cpool(propname,B1,B2,B3,C,AS,Packets,Unseen,DB):-
        new_node(np,[name],C1),
        semantics(start_np,DB,C1),
        semantics(propname,DB,C1,B1),
        !, rulematch(B1,B2,B3,[C1|C],Rulematch,AS,[[build_name]|
                                                   Packets],Unseen,DB).

/** rule PROPNOUN: [propnoun] -> np in 1st buffer        */
```

```
cpool(10, (propnoun ), t, t, t, propnoun).

  cpool(propnoun,B1,B2,B3,C,AS,Packets,Unseen,DB) :-
        new_node(np,B4),
        semantics(start_np,DB,B4),
        semantics(propnoun,DB,B4),
        attach(B1,B4,noun,B33),
        !, rulematch(B33,B2,B3,C,Rulematch,AS,Packets,Unseen,DB).

/** rule PP: [prep][nsstart]-> B1 <- PP,  attach 2nd to c as prep
                attach 3rd to c as np   cf left out     */

cpool(10, (prep), (nsstart), t, t, pp).

  cpool(pp,B1,B2,B3,C,AS,Packets,Unseen,DB) :-
        new_node(pp,B11),
        semantics(prep,DB,B11),
        !, rulematch(B1,B2,B3,[B11|C],Rulematch,AS,
                                [[parse_pp,cpool]|Packets],Unseen,DB).

  rule MARKED_STARTNP: [det, agree_det] -> start a new np node */

cpool(10, (det), t, t, agree(det), marked_startnp).

  cpool(marked_startnp,B1,B2,B3,C,AS,Packets,Unseen,DB) :-
        new_node(np,C1),
        semantics(start_np,DB,C1),
        !, rulematch(B1,B2,B3,[C1|C],Rulematch,AS,
                [[parse_det,npool]|Packets],Unseen,DB).

/** rule STARTNP: [nsstart] -> start a new NP node      */

cpool(10, (nsstart & not(pronoun # det)), t, t, t, startnp).
        % the above pattern is needed for historical reasons

  cpool(startnp,B1,B2,B3,C,AS,Packets,Unseen,DB) :-
        new_node(np,C1),
        semantics(start_np,DB,C1),
        !, rulematch(B1,B2,B3,[C1|C],Rulematch,AS,
                [[parse_qp_1,npool]|Packets],Unseen,DB).


/* rule THAN_COMP: [than_comp][np] -> attach B2 to B1 as comparitive     */

cpool(10, (than_comp), (np), t, t, than_comp).

 cpool(than_comp,B1,B2,B3,C,AS,PAckets,[U1|Unseen],DB) :-
        attach(B2,B1,than_comp,B11),
        semantics(than_comp,DB,B1,B2),
        !, rulematch(B11,B3,U1,C,Rulematch,AS,PAckets,Unseen,DB).

/** rule AND: [conj] -> stuff onto active stack */

cpool(10, (conj & not(andc)), t, t, t, and).

  cpool(and,B1,B2,B3,C,AS,Packets,[U1|Unseen],DB) :-
        addfeats(B1,andc,B11),
        !, rulematch(B2,B3,U1,[B11|C],Rulematch,
                        AS,[[cpool,parse_vp,parse_conj]|Packets],Unseen,DB).
```

```
/** rule COMP_TO_NP: [comp_s] -> make an np in B1        */

cpool(10, (comp_s), t, t, t, comp_to_np).

  cpool(comp_to_np,B1,B2,B3,C,AS,Packets,Unseen,DB) :-
        new_node(np,B11),                % was labeled comp_np
        semantics(start_np,DB,B11),
        semantics(comp_to_np,DB,B11,B1),
        attach(B1,B11,s,B12),
        !, rulematch(B12,B2,B3,C,Rulematch,AS,Packets,Unseen,DB).

/* rule NP_PP [pp] -> consider to attach the pp to the np        */

cpool(10, (pp), t, t, sem_chk(pp), np_pp).

  cpool(np_pp,B1,B2,B3,[C|TL],AS,Packets,[U1|Unseen],DB) :-
        write('HEY - NP_PP in packet CPOOL is running!!'), nl,
        set_label(C,Lab), write('Attaching a PP to '), write(Lab), nl,
        attach(B1,C,pp,C11,DB),
        semantics(np_pp,DB,B1,C),
        !, rulematch(B2,B3,U1,[C11|TL],Rulematch,AS,Packets,Unseen,DB).

/** rule PRONOUN: [pronoun] -> attach to c, fix feats    */

cpool(15, (pronoun), t, t, t, pronoun).

  cpool(pronoun,B1,B2,B3,C,AS,Packets,Unseen,DB):-
        new_node(np,B11),
        semantics(pronoun,DB,B1,B11),
        attach(B1,B11,pronoun,B12),
        (B1 has poss, !, addfeats(B12,poss_np,B13) ; B13 = B12),
        !, rulematch(B13,B2,B3,C,Rulematch,AS,Packets,Unseen,DB).

/** rule VP_ATTACH: [vp] -> attach to s */

cpool(10, (vp), t, t, t, vp_attach).

  cpool(vp_attach,B1,B2,B3,[C|TL],AS,Packets,[U1|Unseen],DB):-
        attach(B1,C,vp,C1),
        semantics(vp_attach,DB,B1,C),
        !, rulematch(B2,B3,U1,[C1|TL],Rulematch,AS,Packets,Unseen,DB).


/* rule POSS_NP: [poss] -> attach as poss        */

cpool(15, t, (possesive), t, t, poss_np).

  cpool(poss_np,B1,B2,B3,[C|TL],AS,Packets,[U1|Unseen],DB):-
        addfeats(B1,poss_np,B11),
        attach(B2,B11,poss,B12),
        semantics(poss_np,DB,B2,B1),
        !, rulematch(B12,B3,U1,[C|TL],Rulematch,AS,Packets,Unseen,DB).
```

```
/* PVP.PK : Packet       PARSE_VP
                assumes C is s, needing a VP

                                                    Rob
                                                    Updated:  23 April 81 (R)
*/



  :- mode parse_vp(+,-,-,-,-,?).
  :- mode parse_vp(+,+,+,+,+,+,+,+,?).

/*-------------------------------------------------------------------------*/


/** rule PREDP [PP] -> attach
 as predp, change packets        */

parse_vp(10, (PP # AP), t, t, t, predp).

  parse_vp(predp,B1,B2,B3,[C|TL],AS,[APacks|Packets],[U1|Unseen],DB):-
        deactivate(parse_vp,APacks,P1),
        addfeats(B1,predp,B11),
        semantics(predp,DB,B1),
        attach(B11,C,predp,C2),
        (C2 has major, !, activate(ss_final,P1,P2);
        (C2 has sec, !, activate(embedded_s_final, P1, P2);
        P2 = P1)),
        !, rulematch(B2,B3,U1,[C2|TL],Rulematch,AS,[P2|Packets],Unseen,DB).

/* rule MAIN_VERB: [verb] -> do everything.     */

parse_vp(10, (verb), t, t, t, main_verb).
        % agree with subject
  parse_vp(main_verb,B1,B2,B3,[C|TL],AS,[APacks|Packets],[U1|Unseen],DB):-
        deactivate(parse_vp,APacks,P1),
        (C has major, !, activate(ss_final,P1,P2);
        (C has sec, !, activate(embedded_s_final,P1,P2);
                P1=P2)),
        new_node(VP,C1),
        attach(B1,C1,verb,C2),
        semantics(main_verb,DB,B1),
        verb_types(B1,NewPackets),
        activate(ss_vp,NewPackets,P3),
        (B1 has two_obj, !, P3=P4 ; activate(object,P3,P4) ),
        activate(cpool,P4,P5),
        !, rulematch(B2,B3,U1,[C2,C|TL],Rulematch,AS,
                [P5,P2|Packets],Unseen,DB).
```

```
/* SSVP.PK : Packet      SS_VP
                assumes C is a VP, major

                                                    Rob
                                                    Updated: 4 June 81 (R)
        Minor changes made by CSM
*/



  :- mode ss_vp(+,-,-,-,-,?).
  :- mode ss_vp(+,+,+,+,+,+,+,+,?).

/*-----------------------------------------------------------------------*/

/* rule ADVERB_GROUP: [adverb][adverb] -> compound adverb        */

ss_vp(5, (adverb), (adverb), t, t, adverb_group).

  ss_vp(adverb_group,B1,B2,B3,C,AS,Packets,[U1|Unseen],DB) :-
        new_node(adverb,A1),
        attach(B1,A1,adverb,A2),
        attach(B2,A2,adverb,A3),
        semantics(adverb_group,DB,A1,B1,B2),
        !, rulematch(A3,B3,U1,C,Rulematch,AS,Packets,Unseen,DB).

/* rule ADVERB: [adverb] -> attach as adverb     */
        /* For karen Archbold, add the patters:
                (adverb # prep# fpunc) to the second buffer
                I will use the looser form       */

ss_vp(10, (adverb), t, t, t, adverb).

  ss_vp(adverb,B1,B2,B3,[C|TL],AS,Packets,[U1|Unseen],DB):-
        attach(B1,C,adverb,C1),
        semantics(adverb,DB,B1),
        !, rulematch(B2,B3,U1,[C1|TL],Rulematch,AS,Packets,Unseen,DB).



  rule PART: [particle] -> attach to verb       */

ss_vp(5, (prep), t, t, sem_chk(particle), part).

  ss_vp(part,B1,B2,B3,[C|TL],AS,Packets,[U1|Unseen],DB) :-
        attach(B1,C,part,C1),
        semantics(part,DB,B1),
        !, rulematch(B2,B3,U1,[C1|TL],Rulematch,AS,Packets,Unseen,DB).

/** rule PP_UNDER_VP_1: [pp] -> attach to c
automatocally attaches to the vp, rule in cpool decides for the np
semantics: checks the can have, if true then it attaches to the np,
        else it attaches to the vp.*/

ss_vp(10, (pp), t, t, sem_chk(v_pp), pp_under_vp_1). % sem_chk added by CSM

  ss_vp(pp_under_vp_1,B1,B2,B3,[C|TL],AS,Packets,[U1|Unseen],DB):-
        attach(B1,C,pp,C1),
        semantics(pp_under_x,DB,B1),
        !, rulematch(B2,B3,U1,[C1|TL],Rulematch,AS,Packets,Unseen,DB).
```

```
/* rule PART: [particle] -> attach to verb       */

ss_vp(15, (prep), t, t, t, part).

  ss_vp(part,B1,B2,B3,[C|TL],AS,Packets,[U1|Unseen],DB) :-
        attach(B1,C,part,C1),
        semantics(part,DB,B1),
        !, rulematch(B2,B3,U1,[C1|TL],Rulematch,AS,Packets,Unseen,DB).

/** rule VP_DONE: [t] -> drop c.          */

ss_vp(15, t, t, t, t, vp_done).


  ss_vp(vp_done,B1,B2,B3,[C|TL],[[wh_comp,B11]|AS],
                                [APacks|Packets],Unseen,DB):-
        attach(B11,C,trace,C1),
        semantics(drop_vp_trace,DB,B11),
        !, rulematch(C1,B1,B2,TL,Rulematch,AS,Packets,[B3|Unseen],DB).

  ss_vp(vp_done,B1,B2,B3,[C|TL],AS,[APacks|Packets],Unseen,DB):-
        !, rulematch(C,B1,B2,TL,Rulematch,AS,Packets,[B3|Unseen],DB).
```

```
/* EMBSFI.PK : Packet   EMBEDDED_S_FINAL
               assumes C is S, embedded

                                                    Rob
                                                    Updated: 7 March 81 (R)
*/


  :- mode embedded_s_final(+,-,-,-,-,?).
  :- mode embedded_s_final(+,+,+,+,+,+,+,+,+,?).

/*-------------------------------------------------------------------------*/

/* rule PP_UNDER_S_2: [pp] -> attach    */

embedded_s_final(10, (pp), t, t, sem_chk(v_pp), pp_under_s_2).
               % Semantic check added by CSM

  embedded_s_final(pp_under_s_2,B1,B2,B3,[C|TL],AS,Packets,[Ui|Unseen],DB):-
        attach(B1,C,pp,C1),
        semantics(pp_under_x,DB,B1),
        !, rulematch(B2,B3,U1,[C1|TL],Rulematch,AS,Packets,Unseen,DB).

/** rule EMBEDDED_S_DONE: [t] -> drop c.        */

embedded_s_final(15, t, t, t, t, embedded_s_done).

  embedded_s_final(embedded_s_done,B1,B2,B3,[C|TL],AS,[APacks|Packets],
                                                        Unseen,DBold):-
        pop_sent(DBold,DB),
        !, rulematch(C,B1,B2,TL,Rulematch,AS,Packets,[B3|Unseen],DB).
```

```
/* NPOOL.PK : Packet    NPOOL
              assumes C is an NP being built

                                                    Rob
                                                    Updated: 6 December 80 (R)
*/


  :- mode npool(+,-,-,-,-,?).
  :- mode npool(+,+,+,+,+,+,+,+,+,?).

/*-------------------------------------------------------------------------*/

/** rule QP_AND_QUANT: [qp][and][quant] -> new qp node on c, attach B1 and B2   *

npool(10, (qp), (conj), (quant), t, qp_and_quant).

  npool(qp_and_quant,B1,B2,B3,C,AS,Packets,[U1,U2|Unseen],DB) :-
        new_node(qp,C1),
        attach(B1,C1,qp,C2),
        attach(B2,C2,conj,C3),
        semantics(conj_qp_1,DB,C1,B1),
        !, rulematch(B3,U1,U2,[C3|C],Rulematch,AS,Packets,Unseen,DB).


/** rule LONGER_THAN: [than][name] -> make a comparative          */
        %       does only "ap than name"

npool(10, (than), (name), t, t, longer_than).

  npool(longer_than,B1,B2,B3,[C|TL],AS,Packets,[U1,U2|Unseen],DB) :-
        addfeats(C,than_comp,C11),
        attach(B1,C11,than,C12),
        attach(B2,C12,name,C13),
        semantics(than_comp,DB,C12,B2),
        !, rulematch(B3,U1,U2,[C13|TL],Rulematch,AS,Packets,Unseen,DB).


/** rule 3 FT/SEC: [qp][units] -> new qp in B1  */

  ool(10, (qp), (unit), t, t, qp_units).

  npool(qp_units,B1,B2,B3,C,AS,Packets,[U1|Unseen],DB) :-
        semantics(qp_units,DB,B1,B2),
        addfeats(B1,[unit,ns],B13),
        attach(B2,B13,unit,B11),
        !, rulematch(B11,B3,U1,C,Rulematch,AS,Packets,Unseen,DB).

/** rule FT_LONG: [qp][adj] -> new ap, attach as adj    */

npool(10, (qp), (adj), t, sem_chk(dimadj), ft_long).
        % Semantic check added by CSM
        % It should really be a syntactic feature check

  npool(ft_long,B1,B2,B3,C,AS,Packets,[U1|Unseen],DB) :-
        new_node(ap,AP1),
        attach(B1,AP1,qp,B11),
        attach(B2,B11,adj,B12),
        semantics(ft_long,DB,B11,B1,B2),
```

```
        !, rulematch(B12,B3,U1,C,ap_attach,AS,Packets,Unseen,DB).

/** rule NOUN_QP: [quant] -> new qp node          */

npool(10, (quant), t, t, t, noun_qp).

  npool(noun_qp,B1,B2,B3,[C|TL],AS,Packets,Unseen,DB) :-
        new_node(qp,B11),
        attach(B1,B11,quant,B12),
        semantics(quant,DB,B11,B1),
        semantics(np_qp,DB,B11,C),
        !, rulematch(B12,B2,B3,[C|TL],Rulematch,AS,Packets,Unseen,DB).

/** rule AP_ATTACH: [ap] -> attach to c as AP    */

npool(10, (ap), t, t, t, ap_attach).

  npool(ap_attach,B1,B2,B3,[C|TL],AS,Packets,[U1|Unseen],DB) :-
        attach(B1,C,ap,C1,DB),
        semantics(ap_attach,DB,C,B1),
        !, rulematch(B2,B3,U1,[C1|TL],Rulematch,AS,Packets,Unseen,DB).

  * rule QP_ATTACH: [qp] -> attach to c as qp    */

npool(10, (qp), t, t, t, qp_attach).

  npool(qp_attach,B1,B2,B3,[C|TL],AS,Packets,[U1|Unseen],DB):-
        attach(B1,C,qp,C1,DB),
        semantics(qp_attach,DB,B1,C),
        semantics(qp_attach1,DB,B1),              % adds the arbs
        (C has qp, !, semantics(conJ_qp_2,DB,C,B1),
        !, rulematch(C1,B2,B3,TL,Rulematch,AS,Packets,[U1|Unseen],DB);
        !, rulematch(B2,B3,U1,[C1|TL],Rulematch,AS,Packets,Unseen,DB)).
```

```
/* PQP2.PK : Packet      PARSE_QP_2
                assumes C is a NP after a det is found

                                                    Rob
                                                    Updated: 10 March 81 (R)
*/


  :- mode parse_qp_2(+,-,-,-,-,?).
  :- mode parse_qp_2(+,+,+,+,+,+,+,+,+,?).

/*-----------------------------------------------------------------------*/

/** rule DET_QUANT: [quant,det or num] -> new qp node    */

parse_qp_2(10, (quant), (adj # noun), t, t, det_quant).

  parse_qp_2(det_quant,B1,B2,B3,[C|TL],AS,Packets,Unseen,DB):-
        new_node(qp,B11),
        attach(B1,B11,quant,B12),
        semantics(det_qp,DB,B12,C),
        !, rulematch(B12,B2,B3,[C|TL],det_quant_done,AS,Packets,Unseen,DB).

/* rule ORDINAL: [ord] -> new ordqp node, ect    */

parse_qp_2(10, (ord), t, t, t, ordinal).

  parse_qp_2(ordinal,B1,B2,B3,[C|TL],AS,[APacks|Packets],[U1|Unseen],DB):-
        deactivate(parse_qp_2,APacks,P1),
        activate(parse_adj,P1,P2),
        new_node(ap,B11),
        attach(B1,B11,ord,B12),
        semantics(ordinal,DB,B1,B11),    % Chansed C to B11 - CSM
        !, rulematch(B12,B2,B3,[C|TL],Rulematch,AS,[P2|Packets],
                                                    [U1|Unseen],DB).

/** rule DET_QUANT_DONE: [t] -> redo packets.    */

parse_qp_2(15, t, t, t, t, det_quant_done).

  parse_qp_2(det_quant_done,B1,B2,B3,C,AS,[APacks|Packets],Unseen,DB):-
        deactivate(parse_qp_2,APacks,P2),
        activate(parse_adj,P2,P3),
        !, rulematch(B1,B2,B3,C,Rulematch,AS,[P3|Packets],Unseen,DB).
```

```
/* SSFIN.PK : Packet    SS_FINAL
              assumes C is major S

                                                    Rob
                                                    Updated: 6 December 80 (R)
*/


  :- mode ss_final(+,-,-,-,-,?).
  :- mode ss_final(+,+,+,+,+,+,+,+,?).

/*-------------------------------------------------------------------------*/

/** rule PP_UNDER_S_1: [pp] attach to c */

ss_final(10, (pp), t, t, t, pp_under_s_1).

  ss_final(pp_under_s_1,B1,B2,B3,[C|TL],AS,Packets,[U1|Unseen],DB):-
        semantics(pp_under_x,DB,B1),
        attach(B1,C,pp,C1),
        !, rulematch(B2,B3,U1,[C1|TL],Rulematch,AS,Packets,Unseen,DB).

/** rule S_DONE: [finalpunc] -> attach and end. */

ss_final(10, (fpunc), t, t, t, s_done).

  ss_final(s_done,B1,B2,B3,[C|TL],AS,Packets,Unseen,DBold):-
        attach(B1,C,fpunc,C1),
        pop_sent(DBold,DB),
        alldone([C1|TL],DB).


/* rule here for what little fish eat, and garden path stuff     */
/* [be], drop and make np        */

/* rule INIT_S_BAR: [verb] -> drop as a NP       */

ss_final(10, (sent_subj), t, t, t, init_s_bar).
                        % should really be sent_subj feature

  ss_final(init_s_bar,B1,B2,B3,[C|TL],AS,Packets,Unseen,DBold):-
        addfeats(C,[comp_s],C1),
        semantics(init_s_bar,DBold,C,B1),
        new_node(s,[major],S),
        pop_sent(DBold,DB1),    % 'start' semantics abolished - CSM
        push_sent(S,DB1,DB),
        !, rulematch(C1,B1,B2,[S|TL],Rulematch,AS,[[cpool,parse_subj]|Packets],
                                                    [B3|Unseen],DB).

/* CONJOINED_S: [comma][conj or binder] -> make into a conjoined S       */

ss_final(10, (comma), (conj # binder), t, t, conjoined_s).

  ss_final(conjoined_s,B1,B2,B3,[C|TL],AS,Packets,[U1,U2|Unseen],DBold) :-
        new_node(s,S1),
        attach(C,S1,s,S2),
        attach(B2,S2,conj,S3),
        pop_sent(DBold,DB2),
        push_sent(S3,DB2,DB1),
```

```
        new_node(s,Snew),
        semantics(conJ,DB1,S3,C,B2,Snew),       % Added by CSM
        push_sent(Snew,DB1,DB),
        !, rulematch(B3,U1,U2,[Snew,S3|TL],Rulematch,AS,
                [[cpool,ss_start],Packets],Unseen,DB).

/* HYPO_S: [comma] -> then an if, what sentence is assumed.
                attach the lowest node to the next up node and
                add a new s node and pray.                  */

ss_final(10, (comma), t, t, t, hypo_s).

  ss_final(hypo_s,B1,B2,B3,[S,IFS|TL],AS,[P1,Packets],[U1|Unseen],DBold) :-
        attach(S,IFS,s,S1),
        new_node(s,Snew),
        semantics(hypo_s,DB,S1,S,Snew),         % Extended - CSM
        pop_sent(DBold,DB1),
        push_sent(Snew,DB1,DB),
        !, rulematch(B2,B3,U1,[Snew,Si|TL],Rulematch,AS,
                                [[cpool,ss_start],Packets],Unseen,DB).
```

```
/* Rob Milne              SEMCHK
                                                    Updated: 6 June 81

Semantic Checks and PP attachment        */

/* SEMANTIC_CHECK questions      */

/* PP_ATTACH see if head nouns are compatable
The NP_PP check is called in cpool when a [np][pp] is found. it does:
1. finds the number of the pp
2. Finds the number of the last Np of the 1st Buffer
3. sets the Np number for the PP
4. calls PP_CHECk with the two np numbers, it then decides
 can set [np][pp], [pp][pp], [np-qp][pp]  always attaches this last case          */

semantic_check(pp,PP,_,_,_,DB) :-        % of PP always attach
        set_label(PP,Num),
        find( is_prep(Num,of,NP), DB).

semantic_check(pp,PP,_,_,_,DB) :-        % if the PP has a qp, then attach
        set_label(PP,PPnum),
        find( qp_modify(PPnum,QP), DB).

semantic_check(pp,PP,B2,_,NP,DB) :-
        set_label(NP,NPnum),
        set_last_np(NPnum,LNP,DB),       % returns the number of the np
        set_label(PP,PP1),
        find(  is_prep(PP1,Prep,PPNP),DB), !,
        write('trying to attach '),write(PPNP),write(' to '), write(LNP),nl,
        pp_check(Prep,LNP,PPNP,DB).

semantic_check(pp,_,_,_,_,DB) :- !, fail.        % default to not attach

set_last_np(PP,LNP,DB) :-
        find(  pp_linked(PP,NP)&
        is_prep(NP,Prep,NP2),DB),
        set_last_np(NP2,LNP,DB), !.

set_last_np(NP,NP,DB) :- !.


   PP_CHECK
this sets the NP number for the target of the attach,
and the NP number of the NP that does the PP     */
        % All the pp_check stuff was written by Keith MacKay for
        % an AI2 project.

%       default is false

pp_check(Prep,NP,PP,DB) :-
        find( headnoun(NP,NPword) &
          headnoun(PP,PPword), DB),
        pp_check(Prep,NPword,PPword).

pp_check(on,NP,PP,DB) :-
        find( headnoun(NP,NPword), DB),
        NPword = tension.

pp_check(Prep,NPword,PPword) :-
        has_property(NPword,PPword),
```

```
        !.

pp_check(Prep,NPword,PPword) :-
        has_property(PPword,NPword),
        !.

pp_check(Prep,NPword,PPword) :-
        person(Personlist),
        person_part(Partlist),
        member(NPword,Personlist),
        member(PPword,Partlist),
        !, fail.

pp_check(of,NPword,PPword) :-
        person(Personlist),
        person_part(Partlist),
        member(NPword,Partlist),
        member(PPword,Personlist),
        !.

pp_check(at,NPword,PPword) :-
        has_property(NPword,phys_obj),
        has_property(PPword,position),
        !.

pp_check(on,NPword,PPword) :-
        has_property(NPword,phys_obj),
        has_property(PPword,position),
        !.

pp_check(on,NPword,PPword) :-
        has_property(NPword,action),
        has_property(PPword,phys_obj),
        !.

        % a hack sort of for is 100 m above the sea.

pp_check(Prep,NP,PP,DB) :-
        find( qp_det(NP,QP) & measure(QP,_,U), DB),
        not( find( headnoun(NP,Noun), DB) ),
        % These extra checks added by CSM:
        rel_prep(Prep), distance_unit(U),
        !.

pp_check(Prep,NP,PP,DB) :- !, fail.


has_property(Word,Property) :-
        semantic_def(Word,Semdef),
        member(Property,Semdef),
        !.

/* NOUNS          semantic check */
semantic_check(nouns,Nouns,Next,_,C,DB) :-
        (Next has auxverb # verb), !.

semantic_check(nouns,_,_,_,NP,DB) :-            % needs a headnoun
        set_label(NP,NPnum),
        not(find( headnoun(NPnum,Head),DB) ), !.
```

```
semantic_check(nouns,_,_,_,NP,DB) :-                    % det was singular
        set_label(NP,NPnum),
        find( num(NPnum,Number,Def), DB),
        not( Number = 1), !.

semantic_check(nouns,_,Next,_,_,DB) :-                  % "ofPP" coming
        set_label(Next,of), !.

semantic_check(nouns,_,Next,_,_,DB) :-                  % statistical hueristic
        Next has nsstart#prep#adverb#pronoun, !, fail.          % don't attach

/* insert Jeep rocks semantic check here */
/* statistical results for nouns:
        attach if next is auxverb or definite verb (noun use)
        don't attach(verb) is next is nsstart,adverb,prep,verb,past,pronoun.
        */

/* REDUCED RELATIVE: uses a heuristic, "must have a main verb"  */

semantic_check(red_rel,_,_,_,_,DB) :-
        find(curr_sent(S) &
        main_verb(S,_),DB), !.

                        % by explict listing
semantic_check(particle,B1,_,_,_,DB) :-
        set_label(B1,Prep),
        find(main_verb(S,Verb),DB),
        verb_particle(Verb,Prep), !.

/* These extra checks added by CSM */

% AND: for conjunction of two NPs, they should be compatible

semantic_check(and,B1,B2,B3,C,DB) :-
   set_label(B1,NP1),
   set_label(B3,NP2),
   find(headnoun(NP1,N1)&headnoun(NP2,N2),DB), !,
   semantic_def(N1,Def1),
   semantic_def(N2,Def2),
   member(X,Def1), member(X,Def2), !.
 semantic_check(and,_,_,_,_,_).

% V_PP: PP attached to a VP should have a preposition compatible
%        with the verb

semantic_check(v_pp,PP,_,_,_,DB) :-
   find( curr_sent(S) & main_verb(S,V), DB),
   set_label(PP,Lab),
   find( is_prep(Lab,P,_), DB),
   prepfor(V,P).

% DIMADJ: Can an adjective appear in an AP starting "QP ADJ ..."?
%       This should be a syntactic feature check

semantic_check(dimadj,_,A,_,_,_) :-
   set_label(A,Wd),
   dimadj(Wd,_,_).
```

```
/* SEMRUL : Semantic rules

                                                    Rob
                                                    Updated: 16 December 80 (R)
            load using: load_sem(semrul).
        Extra 'S' argument added to 'wh_trace' to aid
        correct trace bindins - CSM.
        Extra 'Conj' argument to 'conj' to distinguish
        what the conjunction is - CSM
        Semantic rules 'start' and 'if_what' removed in
        favour of 'sentence' (activated at end, not start).
        This should handle sentence conjunction properly - CSM
*/



semantics(sentence, ([ Sentence ],
                        add( sentence(Sentence) ) )).

semantics(wh_quest, ([ Word ],
                        find( curr_sent(S) ),
                        add( wh_quest(S,Word) &
                        stype(S,wh_quest) ) )).

semantics(wh_np, ([ NP, WHword ],
                        find( num(NP,1,WHword) ),
                        add( headnoun(NP,WHword) ) )).

semantics(major_decl_s, ( [],
                        find( curr_sent(S) ),
                        add( stype(S,statement) ) )).

semantics(aux_invert, ( [],
                        find( curr_sent(S) ),
                        add( stype(S,yes_no_question) ) )).

semantics(imperative, ( [],
                        find( curr_sent(S) ),
                        add( stype(S,command) ) )).

semantics(utterance, ([ NP ],
                        find( curr_sent(S) ),
                        add( utterance(S,NP) ) )).

semantics(that_s_start, ([ S ],
                        add( embedded_sent(S) ) )).

semantics(inf_s_start, ([ S, NP ],
                        add( embedded_sent(S) &
                             syn_subj(S,NP) ) )).

semantics(propname).

semantics(name, ([ Word, NP ],
                        find( num(NP,1,def) ),
                        add( name(NP,Word) ) )).

semantics(pronoun, ([ NP ],
```

```
                        find( num(NP,1,def) ) )).

semantics(poss_det, ([ DET, NP ],
                        add( poss_det(_,NP) ) )).

semantics(comp_to_np, ([ NP, S ],
                        find( num(NP,1,comp) ),
                        add( np_comp_s(NP,S) ) )).

semantics(vp_attach).

semantics(conj, ([ Num, NP1, Conj, NP2 ],
                        add( conj(Num,NP1,Conj,NP2) ) )).

semantics(np_complete).

semantics(how_many, ([ AP, Word ],
                        find( curr_sent(S) ),
                        add( wh_trace(_,AP,S) &
                             intensifier(AP,how) &
                             headadj(AP,Word) ) )).

  antics(so_that).

semantics(relpron_np, ([ WHword, NP ],
                        add( headnoun(NP,WHword) ) )).

semantics(start_np, ([ NP ],
                        add( num(NP,_,_) ) )).

semantics(det, ([ DET, NP ],
                        find( poss_det(NP,NPlower) &
                                num(NP,_,def) ) )

            or  ([ DET:wh, NP],
                        find( num(NP,_,DET) ))

            or  ([ DET:def, NP ],
                        find( num(NP,_,def) ))

            or ([ DET, NP ],
                        find( num(NP,_,indef) ) )).

semantics(det_qp).                                      % Needs thought

semantics(quantifier, ([ Q, NP ],
                        add( quantifier(NP,Q) ) )).

semantics(adj, ([ Red, NP ],
                        find( sensym_label(ap,AP) ),
                        add( hasfeat(NP,AP) &
                             headadj(AP,Red) ) )).

semantics(adj_np, ([ Word, NP ],
                        find( num(NP,1,indef) ),
                        add( headadj(NP,Word) ) )).

semantics(pronoun, ([ Word:ns, NP ],
                        add( num(NP,1,pron) &
                             headnoun(NP,Word) ))
```

```
                or ([ Word, NP ],
                        add( num(NP,plur,pron) &
                             headnoun(NP,Word) ) )).

semantics(noun, ([ Word, NP ],
                        find( num(NP,i,_) ),
                        add( headnoun(NP,Word) ) )).

semantics(nouns, ([ Word, NP ],
                        find( num(NP,plur,_) ),
                        add( headnoun(NP,Word) ) )).

semantics(complex_noun, ([ Word, NP ],
                        find( num(NP,_,_) ),
                        add( headnoun(NP,Word) ) )).

semantics(train_t, ([ NP, Wvar ],
                        add( isa(Word,Wvar) ) )).

semantics(syn_subj, ([ NP ],
                        find( curr_sent(S) ),
                        add( syn_subj(S,NP) ) )).

semantics(syn_obj, ([ T:trace ])

                or  ([ PP:pp ])

                or  ([ NP ],
                        find( curr_sent(S) &
                              syn_obj(S,_) ),
                        add( np_object(S,NP) ))

                or  ([ NP ],
                        find( curr_sent(S) ),
                        add( syn_obj(S,NP) ) )).

semantics(qp_attach, ([ QP, NP ],
                        find( headnoun(NP,_) ),
                        add( qp_modify(NP,QP) ) )

                 or     ([ QP, NP ],
                        add( qp_det(NP,QP) ) )).

semantics(qp_attach1, ([ QP ],
                        find( measure(QP,X,arbs) ))

                 or     ([ QP ] )).

semantics(qp_units, ([ C, Word ],
                        find( measure(C,_,Word) ) )).

semantics(dim, ([ Word:det, NP ],
                        find( dim_var(Word,DV) &
                              num(NP,1,_) ),
                        add( headnoun(NP,Word) &
                             dim(NP,Word,DV) ) )

            or  ([ Word, NP ],
                        find( dim_var(Word,DV) &
```

```
                              num(NP,_,indef) ),
                         add( headnoun(NP,Word) &
                              dim(NP,Word,DV) ) )).

semantics(qp_pp, ([ QP, PP ],
                 add( qp_modify(PP,QP) ) )).

semantics(np_qp).

semantics(ft_long, ([ AP, QP, ADJ ],
                         add( qp_modify(AP,QP) &
                              headadj(AP,ADJ) ) )).

semantics(ap_attach, ([ NP, AP ],
                         add( hasfeat(NP,AP) ) )).

semantics(prep).

semantics(attach_prep, ([ Prep, PP ],
                         add( is_prep(PP,Prep,_) ) )).

semantics(pp_gets_np, ([ PP, NP ],
                         find( is_prep(PP,_,NP) ) )).

semantics(np_pp_default, ([ NP, PP ],                          % Needs thought
        add( pp_linked(NP,PP)  ) )).


semantics(conj_qp_1, ([ QP1, QP2 ],
                         add( conj(QP1,QP2,and,_) ) )).

semantics(conj_qp_2, ([ QP1, QP2 ],
                         find( conj(QP1,_,and,QP2) ) )).

semantics(quant, ([ QP, Word ],
                         find( word_to_num(Word,Num) ),
                         add( measure(QP,Num,_) ) )).

semantics(ordinal, ([ Word, NP ],
                         add( headadj(NP,Word) ) )).

  antics(rel_attach, ([ S, NP ],
                         find( wh_trace(NP,_,S) ),
                         add( relc(NP,S) ) )).

semantics(wh_relative_clause, ([ S ],
                         add( embedded_sent(S) ) )).

semantics(np_pp, ([PP, NP ],
        add( pp_linked(NP,PP) ) )).

semantics(tom_mary).

semantics(poss_np).

semantics(comma).

semantics(np_done, ( [ NP ],
                         find( num(NP,Num,indef) ))
```

```
                or      ( [ NP ]) ).


semantics(than_comp).                                   % Needs thousht

semantics(to_infinitive).

semantics(start_aux).

semantics(aux_attach, ([ AUX:pres , C ],
                            add(  aux_verb(C,pres) ))
                or      ([ AUX:past , C ],
                            add(  aux_verb(C,past) ))
                or      ([ AUX:future , C ],
                            add(  aux_verb(C,future ) ))
                or      ([ AUX, C ],
                            add(  aux_verb(C,tnsless) )) ).

semantics(aux_adverb, ([ Word ],
                        find( curr_sent(S) ),
                        add( adverb(S,Word) ) )).

  antics(do_support),

semantics(simple_neg, ([],
                        find( curr_sent(S) ),
                        add( negative_sent(S) ) )).

semantics(be_pred).

semantics(part, ([ Part ],
                        find( main_verb(S,Verb) ),
                        add( particle(Verb,Part) ) )).

semantics(trace, ([ Trace:pp ],
                        find( is_prep(Trace,_,NP) & curr_sent(S) &
                              num(NP,1,trace) ),
                        add( wh_trace(_,NP,S) ) )

                or ([ Trace ],
                        find( num(Trace,1,trace) & curr_sent(S) ),
                        add( wh_trace(_,Trace,S) ) )).

semantics(bind_trace, ([ BIND ],
                        find( curr_sent(S) & wh_trace(BIND,_,S) ) )).

semantics(drop_vp_trace, ([ Trace:pp ],
                find(curr_sent(S) ),
                add(pp_linked(S,Trace) ) )

                or  ([ Trace ],
                        find( curr_sent(S) ),
                        add( syn_obj(S,Trace) ) )).

semantics(passive, ([ Trace ],
                        find( curr_sent(S) &
                              syn_subj(S,Subj) ),
                        add( passive_sent(S) &
                             wh_trace(Subj,Trace,S) ) )).
```

```
semantics(passive_aux, ( [],
                         find( curr_sent(S) ),
                         add( passive_sent(S) ) )).

semantics(create_delta_subj, ([ Trace ],
                         find( curr_sent(S) &
                               syn_subj(S,Subj) ),
                         add( wh_trace(Subj,Trace,S) ) )).

semantics(main_verb, ([ VP ],
                         find( curr_sent(S) &
                               irreg_verb(VP,Root) ),
                         add( main_verb(S,Root) ) )

                  or  ([ VP ],
                         find( curr_sent(S) ),
                         add( main_verb(S,VP) ) )).

semantics(pp_under_x, ([ PP ],
                         find( curr_sent(S) ),
                         add( pp_linked(S,PP) ) )).

  mantics(adverb, ([ ADV ],
                         find( curr_sent(S) ),
                         add( adverb(S,ADV) ) )).

semantics(adverb_group, ([ NUM, ADV1, ADV2 ],
        add(  hasfeat(NUM,ADV2) &
              hasfeat(NUM,ADV1) ) )).

semantics(reduced_rel).

semantics(predp, ([ PP ],
                         find( curr_sent(S) &
                               is_prep(PP,_,NP) ),
                         add( syn_obj(S,NP) &
                                 main_verb(S,be) ) )

              or  ([ AP ],
                         find( curr_sent(S) ),
                         add( syn_obj(S,AP) &
                                 main_verb(S,be) ) )).

semantics(that_s_start_1, ([ NP, S ],
                         add( embedded_sent(S) &
                              syn_subj(S,NP) ) )).

semantics(inf_s_start_1, ([ NP, S ],
                         add( embedded_sent(S) &
                              syn_subj(S,NP) ) )).

semantics(insert_to, ([ S, NP ],
                         add( embedded_sent(S) &
                              syn_subj(S,NP) ) )).

semantics(obj_in_embedded_s, ([ NP ],
                         find( curr_sent(S) ),
                         add( syn_obj(S,NP) ) )).

semantics(vp_done).
```

```
semantics(embedded_s_done).

semantics(s_done).

semantics(init_s_bar).

semantics(hypo_s,([S1,S11,S12],add(conJ(S1,S11,if,S12)))).      % CSM
```

```
/* SEMSUP.SPL : Semantics Support routines
        contains word to number, semantic_def, irreg_verb and verb_particle

                                                        Rob
                                                        Updated: 20 June 81
                                                        Additions by CSM
*/


   :- public  word_to_num/2,
              semantic_def/2,
              person/1,
              person_part/1,
              irreg_verb/2,
              verb_particle/2,
              prepfor/2,
              distance_unit/1,
              rel_prep/1,
              dimadj/3.

   :- mode word_to_num(+,?),
           semantic_def(+,?),
           person(?),
           person_part(?),
           irreg_verb(+,?),
           verb_particle(+,+).


word_to_num(one,1) :- !.
word_to_num(two,2) :- !.
word_to_num(three,3) :- !.
word_to_num(four,4) :- !.
word_to_num(five,5) :- !.
word_to_num(X,X) :- !.

/* Table of properties of words .. Semantic dictionary. */
/* Intended for PP attachment                           */

semantic_def(particle,[mass,velocity,acceleration,phys_obj]).
semantic_def(block,[mass,length,height,velocity,acceleration,phys_obj]).
  antic_def(room,[length,width,height,wall,floor,ceiling,door,phys_obj]).
  mantic_def(wall,[mass,length,height,point,phys_obj]).
semantic_def(ceiling,[width,length,height,point,phys_obj]).
semantic_def(ball,[mass,length,height,velocity,phys_obj]).
semantic_def(station,[length,height,phys_obj]).
semantic_def(train,[mass,length,height,velocity,acceleration,phys_obj]).
semantic_def(rod,[mass,length,phys_obj]).
semantic_def(cue,[mass,length,velocity,acceleration,phys_obj]).
semantic_def(jeep,[mass,velocity,acceleration,length,height,phys_obj]).
semantic_def(car,[mass,velocity,acceleration,length,width,height,phys_obj]).
semantic_def(lorry,[mass,velocity,acceleration,length,height,width,phys_obj]).
semantic_def(spring,[constant,tension,length,mass,elasticity,
                                                    extension,phys_obj]).
semantic_def(rope,[tension,length,end,phys_obj]).
semantic_def(string,[tension,length,end,phys_obj]).
semantic_def(pulley,[mass,diameter,phys_obj]).
semantic_def(man,[mass,height,phys_obj]).
semantic_def(boy,[mass,height,phys_obj]).
semantic_def(woman,[mass,height,phys_obj]).
semantic_def(girl,[mass,height,phys_obj]).
```

```
semantic_def(tom,[mass,height,phys_obj]).
semantic_def(mary,[mass,height,phys_obj]).
semantic_def(driver,[mass,height,phys_obj]).
semantic_def(painter,[mass,height,phys_obj]).
semantic_def(pier,[mass,length,phys_obj]).
semantic_def(mass,[mass,phys_obj]).
semantic_def(table,[mass,end,length,phys_obj]).

semantic_def(edge,[position]).
semantic_def(corner,[position]).
semantic_def(end,[position]).
semantic_def(height,[position]).

semantic_def(force,[action]).
semantic_def(tension,[action]).
semantic_def(acceleration,[action]).    % CSM

person([boy,girl,mother,father,men,women,woman,man]).
person_part([arm,leg,head,foot,body,teeth,hair,hand]).

/* irregular verb lists, used by semantics in Main_Verb assertions      */

  reg_verb(is,be).                irreg_verb(are,be).
irreg_verb(was,be).               irreg_verb(were,be).
irreg_verb(has,have).             irreg_verb(had,have).
irreg_verb(broke,break).          irreg_verb(thrown,throw).
irreg_verb(came,come).            irreg_verb(hung,hang).
irreg_verb(shown,show).           irreg_verb(shot,shoot).
irreg_verb(fell,fall).            irreg_verb(told,tell).
irreg_verb(found,find).           irreg_verb(knew,know).
irreg_verb(saw,see).              irreg_verb(seen,saw).
irreg_verb(did,do).               irreg_verb(done,do).
irreg_verb(does,do).              irreg_verb(born,bear).
irreg_verb(taken,take).

verb_particle(walk,on).           verb_particle(run,down).
verb_particle(run,away).          verb_particle(look,up).
verb_particle(go,on).             verb_particle(go,in).
verb_particle(break,in).          verb_particle(shoot,up).
verb_particle(shoot,out).
  rb_particle(leave,out).         verb_particle(left,behind).
  rb_particle(show,up).
verb_particle(find,out).          verb_particle(pass,out).
verb_particle(hang,on).           verb_particle(drop,out).
verb_particle(throw,up).          verb_particle(weigh,in).
verb_particle(meet,up).           verb_particle(pull,out).
verb_particle(block,up).
verb_particle(come,in).           verb_particle(give,up). verb_particle(give,out).
verb_particle(attend,to).         verb_particle(shot,up). verb_particle(shot,at).
verb_particle(take,out).          verb_particle(stir,up).

/* Extras added by CSM */

% Which prepositions allow QP's, and
% which units are compatible?

   rel_prep(from).
   rel_prep(above).
   rel_prep(below).
```

```
    distance_unit(m).
    distance_unit(ft).

% Which prepositions are compatible with which verbs?
% This is a pretty crude classification

prepfor(_,X) :- member(X,[in,on,from,by,with,over,under,against,up]).
prepfor(V,to) :- member(V,[incline,pin,attach,apply,connect,fix,throw]).
prepfor(V,at) :- member(V,[incline,pin,attach,fix,travel]).

% Which adjectives mark specific dimensions?

dimadj(old,age,+).
dimadj(young,age,-).
dimadj(long,length,+).
dimadj(short,length,-).
```

```
/* SEM.LPL : Definition of semantic rule application

                                                    Lawrence
                                                    Updated: 21 November 80
        semantics/6 added - CSM
*/

  :- public semantics/2,
            semantics/3,
            semantics/4,
            semantics/5,
            semantics/6.


  :- mode semantics(+,+),
          semantics(+,+,+),
          semantics(+,+,+,+),
          semantics(+,+,+,+,+),
          semantics(+,+,+,+,+,+),
          dosem(+,+,+),
          apply_sem(+,+,+,+),
          apply_sem(+,+,+),
          match_sem(+,+),
          msem(?,+).



                        % Interface from packets

semantics(Type,DB) :-   dosem(Type,DB,[]).


semantics(Type,DB,A) :- dosem(Type,DB,[A]).


semantics(Type,DB,A,B) :- dosem(Type,DB,[A,B]).


semantics(Type,DB,A,B,C) :- dosem(Type,DB,[A,B,C]).


semantics(Type,DB,A,B,C,D) :- dosem(Type,DB,[A,B,C,D]).


                        % Find a semantic rule and apply it

dosem(Type,DB,Args)
     :- atom(Type),
        set(semantics,Type,Rule),
        !,
        apply_sem(Type,Rule,Args,DB).

dosem(Type,_,_)
     :- semerr('Undefined Semantic operation: ',Type).



                        % Error message
```

```
semerr(Mess,Type)
     :- ttynl, display('** '), display(Mess),
        display(Type), ttynl,
        display('   (continuing)'), ttynl.



                        % Apply a rule

apply_sem(_,Rule,Args,DB)
     :- apply_sem2(Rule,Args,DB),
        !.

apply_sem(Type,_,_,_)
     :- semerr('Semantics rule failure: ',Type).



                        % How to apply the various forms of a semantic
                        %  rule body

   ly_sem2(null_rule,_,_).

apply_sem2(rule(Match,Finds,Adds),Args,DB)
     :- match_sem(Match,Args),
        find(Finds,DB),
        add(Adds,DB).

apply_sem2(Rule1 or Rule2,Args,DB)
     :- apply_sem2(Rule1,Args,DB) ;
        apply_sem2(Rule2,Args,DB).



                        % Matching the parameters against the arguments

match_sem([],[]).

match_sem([M|Mrest],[A|Arest])
     :- msem(M,A),
        match_sem(Mrest,Arest).


msem(M,Node)
     :- var(M),
        !,
        set_label(Node,M).

msem(M:Feature,Node)
     :- Node has Feature,
        set_label(Node,M).
```

```
/* RULEM.LPL : Rulematch etc for Rob's parser.

                                                    Lawrence
                                                    Updated: 10 March 81
*/


  :- public parse/3,
            rulematch/9,
            alldone/2.


  :- mode parse(+,?,?),
          rulematch(+,+,+,+,+,+,+,+,+,?),
          set_precedence(-),
          set_packet(+,-),
          check_rule(+,+,+,+,+,+,+,+,+,-),
          docheck(+,+,+,+,+,+),
          rule_spec(+,+,-,-,-,-,+,-),
          apply_rule(+,+,+,+,+,+,+,+,+,+,+,?),
          alldone(+,-).



                        % Parse a list of nodes in some Time to
                        %  get an answer. This routine starts the parser
                        %  proper. There is now a flag "crashing" which
                        %  brings failures back out with only one message

parse([B1,B2,B3|Unseen],Time,DB)
     :- flag(crashing,_,no),
        statistics(runtime,[Start,_]),
        new_node(s,Snode),
        init_db(Snode,DB),      % 'start' semantics abolished - CSM
        rulematch(B1,B2,B3,[Snode],_,[],[[ss_start,cpool]],Unseen,DB),
        statistics(runtime,[Finish,_]),
        Time is Finish-Start.

parse(_,_,_)
     :- flag(crashing,_,no),
        fail.



                        % The main control of the parser
                        %  Cycle through rule specs looking for a
                        %  fireable rule, then call it

rulematch(B1,B2,B3,Cstack,Rulename,AS,Pstack,Unseen,DB)
     :- set_precedence(Prec),
        set_packet(Pstack,Packet),
        check_rule(Packet,Prec,B1,B2,B3,Cstack,DB,Rulename,How),
        !,
        enter(How,Packet,Rulename,B1,B2,B3,Cstack,Pstack),
        apply_rule(How,Packet,Rulename,B1,B2,B3,
                                        Cstack,AS,Pstack,Unseen,DB).

        % this is for aux-inversion and wh movement
```

```
rulematch(B1,B2,B3,Cstack,Rulename,[[Type,B11]|AS],Pstack,Unseen,DB)
        :- !,
        rulematch(B1,B11,B2,Cstack,Rulename,AS,Pstack,[B3|Unseen],DB).

rulematch(B1,B2,B3,Cstack,_,_,Pstack,_,_)
     :- crash(nomatch,'','','',B1,B2,B3,Cstack,Pstack).



                        % Possible precedences (in order)

set_precedence(5).
set_precedence(10).
set_precedence(15).



                        % Currently active packets (from top of Pstack)

set_packet([Packets|_],P) :- member(P,Packets).



                        % See if a packet contains a fireable rule
                        %  Return the name of the first rule found

check_rule(Packet,Prec,B1,B2,B3,[C|Crest],DB,Rulename,How)
     :- rule_spec(Packet,Prec,Spec1,Spec2,Spec3,SpecA,Rulename,How),
        B1 has Spec1,
        B2 has Spec2,
        B3 has Spec3,
        docheck(SpecA,B1,B2,B3,C,DB).



                        % How to decode the check specification which
                        %  can either be a syntactic agreement check
                        %  or a semantic check

docheck(A&B,B1,B2,B3,C,DB) :- !, % This extra rule added by CSM
   docheck(A,B1,B2,B3,C,DB),
   docheck(B,B1,B2,B3,C,DB).

docheck(t,_,_,_,_,_).

docheck(agree(Type),B1,B2,_,_,_)
     :- agree(Type,B1,B2).

docheck(agree_13(Type),B1,_,B3,_,_)
     :- agree_13(Type,B1,B3).

docheck(agree_23(Type),_,B2,B3,_,_)
     :- agree_23(Type,B2,B3).

docheck(agree_all(Type),B1,B2,B3,_,_)
     :- agree_all(Type,B1,B2,B3).

docheck(sem_chk(Type),B1,B2,B3,C,DB)
     :- semantic_check(Type,B1,B2,B3,C,DB).
```

```
                        % Find specs for rules

rule_spec(Packet,Prec,Spec1,Spec2,Spec3,SpecA,Rulename,interpreted)
     :- flag(Packet,interpreted,interpreted),
        !,
        R =.. [Packet,Prec,Spec1,Spec2,Spec3,SpecA,Rulename],
        call(R).

rule_spec(Packet,Prec,Spec1,Spec2,Spec3,SpecA,Rulename,compiled)
     :- switch1(Packet,Prec,Spec1,Spec2,Spec3,SpecA,Rulename).



                        % Fire the rule

apply_rule(interpreted,Packet,Rulename,B1,B2,B3,Cstack,AS,Pstack,Unseen,DB)
     :- R =.. [Packet,Rulename,B1,B2,B3,Cstack,AS,Pstack,Unseen,DB],
        call(R).

   ly_rule(compiled,Packet,Rulename,B1,B2,B3,Cstack,AS,Pstack,Unseen,DB)
     :- switch2(Packet,Rulename,B1,B2,B3,Cstack,AS,Pstack,Unseen,DB).



                        % Return top of C stack at end of parse
                        %  This is called explicitly by the final
                        %  grammar rules rather then calling rulematch

alldone([N],DB)
     :- !,
        semantics(sentence,DB,N),       % CSM
        closenode(N,CN),
        set_tree(DB,CN),
        dbfinish(DB).

        %% this is a hack to do the if,what questions
        %% hack hack hack (rubbing of hands in background)
        %% But it should also do limited sentence conjunction - CSM
alldone([S1,S2],DB)
        :-  !,
        attach(S1,S2,s,S3),
        semantics(sentence,DB,S3),              % CSM
        closenode(S3,CN),
        set_tree(DB,CN),
        pop_sent(DB,DB1),
        dbfinish(DB1).

alldone([N|Others],DB)
     :- ttynl,
        display('C stack not empty at end of parse:'), ttynl,
        portray_stack(pn1,Others), ttynl,
        closenode(N,CN),
        set_tree(DB,CN),
        dbfinish(DB).
```