Appendix A: The grammar

```
% *****************************************************************
%
% Author:   Detmar Meurers (dm@sfs.nphil.uni-tuebingen.de)
%
% Version:  June 1, 1994
% Project:  SFB 340/B4 (Hinrichs/Gerdemann)
%           Universit"at T"ubingen
%           Kleine Wilhelmstr. 113
%           72074 Tuebingen
%           Germany
% System:   ALE 1.0 (release 1 May 1993) under Sicstus 2.1 #9
%
%
% This file contains an ALE grammar covering the phenomena of aux-flip and
% partial verb-phrase topicalization in the three sentence types of German:
% verb first (V1), verb second (V2), and verb last (VL). Except for a few
% cases noted in the paper documenting this grammar, it closely follows
% the analyses proposed in:
%
%              Erhard Hinrichs and Tsuneko Nakazawa
%         ``Partial-VP and Split-NP Topicalization in German''
%                      ``- An HPSG Analysis -''
%
% and:
%           ``Linearizing Finite Aux in German Complex VPs''
%
% *****************************************************************
%
% You are welcome to try out the grammar. The code is available via
% anonymous ftp to "sns2.sfs.nphil.uni-tuebingen.de". The directory
% SFB/IMPLEMENTING contains the grammar including the test sentences,
% a file with some small extensions to ALE needed to run the tests,
% and a postscript version of the paper "On Implementing an HPSG theory"
% which documents the reasoning behind the grammar. If for some reason
% you are unable to retrieve the files, please contact the author at the
% address above.
%
% *****************************************************************
%
% Notational conventions:
%  a) variable names: only the first letter is capitalized
%  b) macros:  - macros without suffix are (generally) of type sign,
%              - suffix _y on a macro name is used for synsem macros
%              - suffix _lex on a macro name is used for sign macros
%                which are intended to abbreviate lexical entries
%              - suffix _arg is used on certain variables which are
%                place-holders for the arguments passed to the
%                macros. The problem is caused by the fact that these
%                argument place-holders in ALE are no real feature structure
%                variables. Several occurrences of this variable
%                in a macro definition are *not* interpreted as
%                reentrancy! These _arg "variables" therefore have
%                to be conjoined to a real variable in the body of
%                the macro, which then can be used to note
%                structure sharing.
%
% *****************************************************************
```



```
% *****************************************************************
%   Signature
% *****************************************************************

% ###################
% I. "Non-Linguistic"
% ###################


bot sub [bool, list, set, sign, synsem_or_nothing, loc, cat, val,
head, case, decl, pform, vform, marking, cont, index, per, num, gend,
qfpsoa, sem_det, conx, nonloc, nonloc1, const_struc, lr].


% ===========================
% I. "System Types"
% ===========================

bool sub [minus, plus].
   minus sub [].
   plus sub [].

list sub [ne_list, quant_list, synsem_list, sign_list, lr_list].
   ne_list sub [ne_quant_list,
                ne_synsem_list,
                ne_sign_list,
                ne_lr_list]
           intro [hd:bot,
                  tl:list].
   quant_list sub [e_list, ne_quant_list].
      e_list sub [].
      ne_quant_list sub []
                    intro [hd:quant,
                           tl:quant_list].
   synsem_list sub [e_list, ne_synsem_list].
      ne_synsem_list sub []
                     intro [hd:synsem,
                            tl:synsem_list].
   sign_list sub [ne_sign_list, n_sign_list, all_sign_list].
      ne_sign_list sub [ne_n_sign_list, ne_all_sign_list]
                   intro [hd:sign,
                          tl:sign_list].
      n_sign_list sub [e_list, ne_n_sign_list].
          ne_n_sign_list sub []
                         intro [hd:n_sign,
                                tl:n_sign_list].
      all_sign_list sub [e_list, ne_all_sign_list].
         ne_all_sign_list sub [].
   lr_list sub [e_list, ne_lr_list].
      ne_lr_list sub []
                 intro [hd:lr,
                        tl:lr_list].


set sub [neset, set_loc, set_npro, set_psoa, set_quant, set_ref, set_sign].
   eset sub [].
   neset sub [neset_loc, neset_npro, neset_psoa, neset_quant,
              neset_ref, neset_sign]
         intro [elt:bot, els:set].
   set_loc sub [eset, neset_loc].
      neset_loc sub []
                intro [elt:loc, els:set_loc].
```

```
   set_npro sub [eset, neset_npro].
      neset_npro sub []
                 intro [elt:npro, els:set_npro].
   set_psoa sub [eset, neset_psoa].
      neset_psoa sub []
                 intro [elt:psoa, els:set_psoa].
   set_quant sub [eset, neset_quant].
      neset_quant sub []
                  intro [elt:quant, els:set_quant].
   set_ref sub [eset, neset_ref].
      neset_ref sub []
                intro [elt:ref, els:set_ref].
   set_sign sub [eset, neset_sign].
      neset_sign sub []
                 intro [elt:sign, els:set_sign].


 % ################
 % II. "Linguistic"
 % ################

 %% BEGIN --- sign ---

 % sign bears an additional lr feature for debugging
 % to store the list of lexical rules applied

sign sub [lex_sign, phrase, n_sign, non_n_sign]
    intro [synsem:synsem,
           qstore:set_quant,
           retr:quant_list,
           lr:lr_list].
   lex_sign sub [word, lex_phrase, n_lex_sign, non_n_lex_sign]
            intro [retr: e_list].
      word sub [n_word, non_n_word].
         n_word sub [].
         non_n_word sub [].
      lex_phrase sub [n_lex_phrase, non_n_lex_phrase].
         n_lex_phrase sub [].
         non_n_lex_phrase sub [].
      n_lex_sign sub [n_word, n_lex_phrase].
      non_n_lex_sign sub [non_n_word, non_n_lex_phrase].
   phrase sub [lex_phrase, struc_phrase, n_phrase, non_n_phrase].
      struc_phrase sub [n_struc_phrase, non_n_struc_phrase]
                   intro [dtrs:const_struc].
         n_struc_phrase sub [].
         non_n_struc_phrase sub [].
      n_phrase sub [n_lex_phrase, n_struc_phrase].
      non_n_phrase sub [non_n_lex_phrase, non_n_struc_phrase].
   n_sign sub [n_lex_sign, n_phrase]
          intro [synsem:n_synsem].
   non_n_sign sub [non_n_lex_sign, non_n_phrase]
              intro [synsem:non_n_synsem].

 %%% BEGIN --- synsem ---

synsem_or_nothing sub [synsem, nothing].
   synsem sub [n_synsem, non_n_synsem]
          intro [loc:loc,
                 nonloc:nonloc].
      n_synsem sub []
               intro [loc:n_loc].
      non_n_synsem sub []
```



```
                intro [loc:non_n_loc].
   nothing sub [].

 %%%% BEGIN --- loc ---

loc sub [n_loc, non_n_loc]
    intro [cat:cat,
           conx:conx,
           cont:cont].
   n_loc sub []
         intro [cat:n_cat].
   non_n_loc sub []
             intro [cat:non_n_cat].


 %%%%% BEGIN --- cat ---

cat sub [n_cat, non_n_cat]
    intro [val:val,
           head:head,
           marking:marking,
           npcomp:bool].
   n_cat sub []
         intro [head:noun].
   non_n_cat sub []
             intro [head:non_noun].


 %%%%%% BEGIN --- val ---

val sub []
    intro [spr: sign_list,
           subj: sign_list,
           comps: sign_list].

 %%%%%% END --- val ---
 %%%%%% BEGIN --- head ---

 % Notes on the subtypes of head:
 % - adj_noun_det and adj_or_noun are included as type only to
 %   satisfy the feature intro condition
 %   and introduce case and decl for adj, noun, det
 % - the introduction of noun and non_noun
 %   needed for n_sign_list, which is used to constrain
 %   on the raisable arguments of auxiliaries, leads
 %   to quite an explosion of the hierarchy:
 %   beside the complication of the sign (!), synsem, and cat
 %   definitions, it necessitates the types adj_prep_verb
 %   adj_or_det, and non_noun to be introduced here (to
 %   get a lattice and satisfy the fic)
 % - the HPSGII subtype reltvzr is not needed in this grammar
 % - prepositions are not in the fragment either, but
 %   are included for easier extension. No pforms are specified.

head sub [func, subst, adj_noun_det, non_noun].
   func sub [det, marker]
        intro [spec:synsem].
      det sub [].
      marker sub [].
   subst sub [adj_or_noun, adj_prep_verb]
         intro [prd:bool,
                mod:synsem_or_nothing].
```

```
      adj_or_noun sub [adj,noun].
         adj sub []
             intro [mod:synsem].
         noun sub [].
      adj_prep_verb sub [adj,prep,verb].
         prep sub []
              intro [pform:pform].
         verb sub [fin_v,n_fin_v]
              intro [aux:bool,
                     inv:bool,
                     flip:bool,
                     vform:vform].
            fin_v sub []
                  intro [vform:fin].
            n_fin_v sub []
                    intro [inv:minus,
                           vform:n_fin].
   adj_noun_det sub [adj_or_noun, adj_or_det]
                intro [case:case,
                       decl:decl].
      adj_or_det sub [adj,det].
   non_noun sub [func, adj_or_det, adj_prep_verb].

case sub [nom, n_nom].
   nom sub [].
   n_nom sub [acc, dat, gen].
      acc sub [].
      dat sub [].
      gen sub [].

decl sub [wk,st].
   wk sub [].
   st sub [].

pform sub [].

vform sub [fin, n_fin].
   fin sub [].
   n_fin sub [bse, psp].
      bse sub [].
      psp sub [].

 %%%%% END --- head ---
 %%%%% BEGIN --- marking ---

 % no conjunctions are specified, since not in fragment

marking sub [marked, unmarked].
   marked sub [comp, conj].
      comp sub [dass, ob].
         dass sub [].
         ob sub [].
      conj sub [].
   unmarked sub [].

 %%%%% END --- marking ---
 %%%% END --- cat ---
 %%%% BEGIN --- cont ---

cont sub [nom_obj, psoa, quant].
   nom_obj sub [npro, pron]
           intro [index:index,
```



```
                       restr:set_psoa].
         npro sub [].
         pron sub [ana, ppro].
            ana sub [recp, refl].
               recp sub [].
               refl sub [].
            ppro sub [].
        psoa sub []
             intro [quants:quant_list,
                    nucl:qfpsoa].
        quant sub []
              intro [det:sem_det,
                     restind:npro].

 %%%%% BEGIN --- (nom-obj) index ---

index sub [ref,expl_es]
      intro [per:per,
             num:num,
             gend:gend].
   ref sub [].
   expl_es sub [].

per sub [first, second, third].
   first sub [].
   second sub [].
   third sub [].

num sub [plur, sing].
   plur sub [].
   sing sub [].

gend sub [fem, masc, neutr].
   fem sub [].
   masc sub [].
   neutr sub [].

 %%%%% END --- (nom-obj) index ---
 %%%%% BEGIN --- (psoa) qfpsoa ---

qfpsoa sub [prop, un_rel, bin_rel, tern_rel].
   prop sub [table, relative, man, book, poem, student, daughter, story,
             snake, father, water, letter, karl, anna, sarah, peter,
             good, green, small]
        intro [inst: ref].
      table sub [].
      relative sub [].
      man sub [].
      book sub [].
      poem sub [].
      student sub [].
      daughter sub [].
      story sub [].
      snake sub [].
      father sub [].
      water sub [].
      letter sub [].
      karl sub [].
      peter sub [].
      anna sub [].
      sarah sub [].
      good sub [].
```

```
      green sub [].
      small sub [].
   un_rel sub [laugh, auxiliary].
      laugh sub []
            intro [laugher:ref].
      auxiliary sub [have, will, can, must, want]
                intro [arg:psoa].
         have sub [].
         will sub [].
         can sub [].
         must sub [].
         want sub [].
   bin_rel sub [love, read, find].
      read sub []
           intro [reader:ref,
                  read:ref].
      love sub []
           intro [lover:ref,
                  loved:ref].
      find sub []
           intro [finder:ref,
                  found:ref].
   tern_rel sub [give,tell].
      give sub []
           intro [giver:ref,
                  receiver:ref,
                  given:ref].
      tell sub []
           intro [teller:ref,
                  tellee:ref,
                  told:ref].

 %%%%% END --- (psoa) qfpsoa ---
 %%%%% BEGIN --- (quant) sem_det ---

sem_det sub [neg_indef,def,indef].
   neg_indef sub [].
   def sub [].
   indef sub [].

 %%%%% END --- (quant) sem_det ---
 %%%% END --- cont ---
 %%%% BEGIN --- conx ---

 % Notes on conx:
 % - no context handling is done in the fragment
 % - name anchoring is done in content instead
 % - to include conx in the fragment
 %   c_inds has to be declared subtype of bot

conx sub [].

 %    intro [backgr: set_psoa,
 %           c_inds: c_inds].
 %
 % c_inds sub [].
 %        intro [addressee:ref,
 %               speaker:ref,
 %               utt_loc:ref].

 %%%% END --- conx ---
 %%% END --- loc ---
```



```
 %%% BEGIN --- nonloc ---

 % Notes on nonloc1:
 % - slash is the only relevant nonlocal feature for this grammar.
 %   Therefore que and rel are not specified as appropriate.

nonloc sub []
       intro [inherited:nonloc1,
              to_bind:nonloc1].


nonloc1 sub []
        intro [               % que:set_npro,
                              % rel:set_ref,
               slash:sign_list].

 %%% END --- nonloc ---
 %% END ---- synsem ---

 %% BEGIN --- const_struc ---

const_struc sub [coord_struc, headed_struc].
   coord_struc sub [].
   headed_struc sub [adjunct_head, comp_struc, filler_head, marker_head, spr_head]
                intro [head_dtr:sign].
      adjunct_head sub []
                   intro [adjunct_dtr:sign].
      comp_struc sub [head_comp, head_subj_comp]
                 intro [comp_dtrs:sign_list].
         head_comp sub [hci1,hci2,hci3,
                        hcni1,hcni2,hcni3,
                        vcf,vcnf].
            hci1 sub []. hci2 sub []. hci3 sub [].
            hcni1 sub []. hcni2 sub []. hcni3 sub [].
            vcf sub []. vcnf sub []. vci sub [].
         head_subj_comp sub [hsci1,hsci2,hsci3,hsci4,
                             hscni1,hscni2a,hscni2b,
                             hscni3a,hscni3b,hscni4a,hscni4b]
                        intro [subj_dtr:sign].
            hsci1 sub []. hsci2 sub []. hsci3 sub []. hsci4 sub [].
            hscni1 sub []. hscni2a sub []. hscni2b sub [].
            hscni3a sub []. hscni3b sub []. hscni4a sub []. hscni4b sub [].
      filler_head sub []
                  intro [filler_dtr:sign].
      marker_head sub []
                  intro [marker_dtr:sign].
      spr_head sub []
                  intro [spr_dtr:sign].

 %% END --- const_struc ---

 % lexical rule names as subtype of lr (for debugging)
lr sub [ein2kein, bse_to_3_sing, bse_to_3_plur, bse_to_psp,
        bse_to_subst_bse, celr, selr, pvpelr, obj_order_lr].

   ein2kein sub [].  bse_to_3_sing sub [].  bse_to_3_plur sub [].
   bse_to_psp sub [].  bse_to_subst_bse sub [].  celr sub [].
   selr sub [].  pvpelr sub [].  obj_order_lr sub [].

 % END ---- sign ---
```

```
% *****************************************************************
%    Macros
% *****************************************************************

% ###########################
% I. simple abbreviations
%   a) path:value
%   b) path:parameter passed
% ###########################

% ==========================
% 1. macros for system stuff
% ==========================

% abbreviation for singleton sets:
sset(X)        macro ((elt:  X),
                      (els: eset)).

% abbreviation for normal sets:
set(E,F)       macro ((elt:  E),
                      (els: F)).

% ==============================
% 2. macros for linguistic types
% ==============================

% ------------------------------
% macros of type < const_struc >
% ------------------------------

% dtrs filled:

adjunct_head_cs(Adj,Head)
               macro ((adjunct_dtr: Adj),
                      (head_dtr:    Head)).

filler_head_cs(Filler,Head)
               macro ((filler_dtr: Filler),
                      (head_dtr:   Head)).

marker_head_cs(Marker,Head)
               macro ((marker_dtr: Marker),
                      (head_dtr:   Head)).

spr_head_cs(Spr,Head)
               macro ((spr_dtr:  Spr),
                      (head_dtr: Head)).

head_comp_cs(Subtype,Head,Comps)
               macro (Subtype, (head_dtr:  Head),
                               (comp_dtrs: Comps)).

head_subj_comp_cs(Subtype,Head,Subj,Comps)
               macro (Subtype, (head_dtr: Head),
                               (subj_dtr: Subj),
                               (comp_dtrs: Comps)).

% for testing: macros with dtrs not filled
% adjunct_head_cs(_,_)            macro adjunct_head_cs.
% filler_head_cs(_,_)             macro filler_head_cs.
% marker_head_cs(_,_)             macro marker_head_cs.
% spr_head_cs(_,_)                macro spr_head_cs.
```



```
% head_comp_cs(Subtype,_,_)       macro (head_comp_cs, Subtype).
% head_subj_comp_cs(Subtype,_,_,_) macro (head_subj_comp_cs, Subtype).


% ------------------------------
% macros of type < sign >
% ------------------------------

% - - - - - - - - - -
% apply to all signs
% - - - - - - - - - -

head(X)        macro (synsem: @head_y(X)).
verb           macro (synsem: @verb_y).
noun           macro (synsem: @noun_y).
adj            macro (synsem: @adj_y).
det            macro (synsem: @det_y).
prep           macro (synsem: @prep_y).
marker         macro (synsem: @marker_y).

val(X)         macro (synsem: @val_y(X)).
val_subj(X)    macro (synsem: @val_subj_y(X)).
val_spr(X)     macro (synsem: @val_spr_y(X)).
val_comps(X)   macro (synsem: @val_comps_y(X)).

subj_sat       macro (synsem: @subj_sat_y).
spr_sat        macro (synsem: @spr_sat_y).
comps_sat      macro (synsem: @comps_sat_y).
sat            macro (synsem: @sat_y).

cont(X)        macro (synsem: @cont_y(X)).
conx(X)        macro (synsem: @conx_y(X)).

marking(X)     macro (synsem: @marking_y(X)).
unmarked       macro (synsem: @unmarked_y).

inherited(X)   macro (synsem: @inherited_y(X)).
to_bind(X)     macro (synsem: @to_bind_y(X)).

in_slash(X)    macro (synsem: @in_slash_y(X)).
to_slash(X)    macro (synsem: @to_slash_y(X)).

no_in_slash    macro (synsem: @no_in_slash_y).
no_to_slash    macro (synsem: @no_to_slash_y).
no_slash       macro (synsem: @no_slash_y).

npcomp(X)      macro (synsem: @npcomp_y(X)).
minus_npcomp   macro (synsem: @minus_npcomp_y).
plus_npcomp    macro (synsem: @plus_npcomp_y).

no_lr          macro (lr: []).

% - - - - - - - - - - - - - - - - - - - - -
% relevant for < noun >, < adj >, < det >
% - - - - - - - - - - - - - - - - - - - - -

decl(X)        macro (synsem: @decl_y(X)).
st             macro (synsem: @st_y).
wk             macro (synsem: @wk_y).
```

```
 % - - - - - - - - - - - - - - -
 % relevant for < func >
 % - - - - - - - - - - - - - - -

spec(X)       macro (synsem: @spec_y(X)).

 % - - - - - - - - - - - - - - -
 % relevant for < det >
 % - - - - - - - - - - - - - - -

restind(X)    macro (synsem: @restind_y(X)).

 % - - - - - - - - - - - - - - -
 % relevant for < subst >
 % - - - - - - - - - - - - - - -

mod(X)        macro (synsem: @mod_y(X)).
no_mod        macro (synsem: @no_mod_y).

prd(X)        macro (synsem: @prd_y(X)).
minus_prd     macro (synsem: @minus_prd_y).
plus_prd      macro (synsem: @plus_prd_y).

 % - - - - - - - - - - - - - - -
 % relevant for < noun >
 % - - - - - - - - - - - - - - -

index(X)      macro (synsem: @index_y(X)).
restr(X)      macro (synsem: @restr_y(X)).

per(X)        macro (synsem: @per_y(X)).
first         macro (synsem: @first_y).
second        macro (synsem: @second_y).
third         macro (synsem: @third_y).

num(X)        macro (synsem: @num_y(X)).
sing          macro (synsem: @sing_y).
plur          macro (synsem: @plur_y).

third_sing    macro (synsem: @third_sing_y).
third_plur    macro (synsem: @third_plur_y).

gend(X)       macro (synsem: @gend_y(X)).
masc          macro (synsem: @masc_y).
fem           macro (synsem: @fem_y).
neutr         macro (synsem: @neutr_y).

case(X)       macro (synsem: @case_y(X)).
nom           macro (synsem: @nom_y).
acc           macro (synsem: @acc_y).
dat           macro (synsem: @dat_y).
gen           macro (synsem: @gen_y).

 % - - - - - - - - - - - - - - -
 % relevant for < verbs >
 % - - - - - - - - - - - - - - -

vform(X)      macro (synsem: @vform_y(X)).
fin           macro (synsem: @fin_y).
bse           macro (synsem: @bse_y).
psp           macro (synsem: @psp_y).
n_fin         macro (synsem: @n_fin_y).
```



```
aux(X)        macro (synsem: @aux_y(X)).
minus_aux     macro (synsem: @minus_aux_y).
plus_aux      macro (synsem: @plus_aux_y).

inv(X)        macro (synsem: @inv_y(X)).
minus_inv     macro (synsem: @minus_inv_y).
plus_inv      macro (synsem: @plus_inv_y).

flip(X)       macro (synsem: @flip_y(X)).
minus_flip    macro (synsem: @minus_flip_y).
plus_flip     macro (synsem: @plus_flip_y).


 % -----------------------------
 % macros of type < synsem >
 % -----------------------------

 % - - - - - - - - - - -
 % apply to all synsems
 % - - - - - - - - - - -

head_y(X)          macro (loc:cat:head:X).
verb_y             macro @head_y(verb).
noun_y             macro @head_y(noun).
adj_y              macro @head_y(adj).
det_y              macro @head_y(det).
prep_y             macro @head_y(prep).
marker_y           macro @head_y(marker).

val_y(X)           macro (loc:cat:val:X).
val_subj_y(X)      macro @val_y(subj:X).
val_spr_y(X)       macro @val_y(spr:X).
val_comps_y(X)     macro @val_y(comps:X).

subj_sat_y         macro @val_subj_y([]).
spr_sat_y          macro @val_spr_y([]).
comps_sat_y        macro @val_comps_y([]).
sat_y              macro (@subj_sat_y, @spr_sat_y, @comps_sat_y).

cont_y(X)          macro (loc:cont:X).
conx_y(X)          macro (loc:conx:X).

marking_y(X)       macro (loc:cat:marking:X).
unmarked_y         macro @marking_y(unmarked).

inherited_y(X)     macro (nonloc:inherited: X).
to_bind_y(X)       macro (nonloc:to_bind: X).

in_slash_y(X)      macro @inherited_y(slash:X).
to_slash_y(X)      macro @to_bind_y(slash:X).

no_in_slash_y      macro @inherited_y(slash:[]).
no_to_slash_y      macro @to_bind_y(slash:[]).
no_slash_y         macro (@no_in_slash_y,
                          @no_to_slash_y).

npcomp_y(X)        macro (loc:cat:npcomp:X).
minus_npcomp_y     macro @npcomp_y(minus).
plus_npcomp_y      macro @npcomp_y(plus).
```

```
 % - - - - - - - - - - - - - - - - - - - - - -
 % relevant for < noun >, < adj >, < det >
 % - - - - - - - - - - - - - - - - - - - - - -

decl_y(X)          macro @head_y(decl:X).
st_y               macro @decl_y(st).
wk_y               macro @decl_y(wk).

 % - - - - - - - - - - - - - - - -
 % relevant for < func >
 % - - - - - - - - - - - - - - - -

spec_y(X)          macro @head_y(spec:X).

 % - - - - - - - - - - - - - - - -
 % relevant for < det >
 % - - - - - - - - - - - - - - - -

restind_y(X)       macro @cont_y(restind:X).

 % - - - - - - - - - - - - - - - -
 % relevant for < subst >
 % - - - - - - - - - - - - - - - -

mod_y(X)           macro @head_y(mod:X).
no_mod_y           macro @mod_y(nothing).

prd_y(X)           macro @head_y(prd:X).
minus_prd_y        macro @prd_y(minus).
plus_prd_y         macro @prd_y(plus).

 % - - - - - - - - - - - - - - - -
 % relevant for < noun >
 % - - - - - - - - - - - - - - - -

index_y(X)         macro (loc:cont:index:X).
restr_y(X)         macro (loc:cont:restr:X).

per_y(X)           macro @index_y(per:X).
first_y            macro @per_y(first).
second_y           macro @per_y(second).
third_y            macro @per_y(third).

num_y(X)           macro @index_y(num:X).
sing_y             macro @num_y(sing).
plur_y             macro @num_y(plur).

third_sing_y       macro (@third_y, @sing_y).
third_plur_y       macro (@third_y, @plur_y).

gend_y(X)          macro @index_y(gend:X).
masc_y             macro @gend_y(masc).
fem_y              macro @gend_y(fem).
neutr_y            macro @gend_y(neutr).

case_y(X)          macro @head_y(case:X).
nom_y              macro @case_y(nom).
acc_y              macro @case_y(acc).
dat_y              macro @case_y(dat).
gen_y              macro @case_y(gen).
```



```
 % - - - - - - - - - - - - - - - -
 % relevant for < verb >
 % - - - - - - - - - - - - - - - -

vform_y(X)         macro @head_y(vform:X).
fin_y              macro @vform_y(fin).
bse_y              macro @vform_y(bse).
psp_y              macro @vform_y(psp).
n_fin_y            macro @vform_y(n_fin).

aux_y(X)           macro @head_y(aux:X).
minus_aux_y        macro @aux_y(minus).
plus_aux_y         macro @aux_y(plus).

inv_y(X)           macro @head_y(inv:X).
minus_inv_y        macro @inv_y(minus).
plus_inv_y         macro @inv_y(plus).

flip_y(X)          macro @head_y(flip:X).
minus_flip_y       macro @flip_y(minus).
plus_flip_y        macro @flip_y(plus).


 % ##########################################
 % II. linguistically motivated abbreviations
 % ##########################################

 % ;;;;;;;
 % general
 % ;;;;;;;

 % < sign >
x_null_s           macro word.
x_bar_s            macro phrase.
xp                 macro (phrase, (synsem: @xp_y)).

 % < synsem >
xp_y               macro @comps_sat_y.

 % ;;;;;;;;;;;;;;
 % * adjectives *
 % ;;;;;;;;;;;;;;

 % < synsem >
ap_y               macro (@adj_y, @sat_y).

 % ;;;;;;;;;;;;;;;
 % * determiners *
 % ;;;;;;;;;;;;;;;

 % < sign >
dp                 macro (phrase, (synsem: @dp_y)).

 % < synsem >
dp_y               macro (@det_y, @sat_y).

 % ;;;;;;;;
 % * noun *
 % ;;;;;;;;

 % < sign >
n                  macro (sign,  (synsem: @n_y)).
```

```
n_bar            macro (phrase, (synsem: @n_bar_y)).
np               macro (phrase, (synsem: @np_y)).
np(Index,Case)   macro (@np, @index(Index), @case(Case)).

n_cont(Prop)     macro (synsem: @n_cont_y(Prop)).

subj_np          macro (@np, @nom).
obj_np           macro (@np, @case( n_nom )).


 % < synsem >
n_y              macro (@noun_y, @subj_sat_y).
n_bar_y          macro (@n_y, @comps_sat_y).
np_y             macro (@n_y, @comps_sat_y, @spr_sat_y).

n_cont_y         macro @cont_y(nom_obj).
n_cont_y(Prop)   macro @cont_y( ((index: Index),
                                 (restr: @sset(((quants: []),
                                                (nucl: (prop, Prop,
                                                        (inst: Index))))))))).

 % ;;;;;;;;;;;;;;;;
 % * prepositions *
 % ;;;;;;;;;;;;;;;;

 % < synsem >
p_y              macro (@prep_y, @no_mod_y).

 % ;;;;;;;;;
 % * verbs *
 % ;;;;;;;;;

 % < sign >
v                macro (synsem: @v_y).

v(Vform,Sem,Subj,Comps,Flip)
                 macro (synsem: (@v_y, @v_cont_y(Sem),
                                 @vform_y(Vform), @flip_y(Flip),
                                 @val_subj_y(Subj), @val_comps_y(Comps))).

v_null           macro (word, @v).
v_null(Vform)    macro (@v_null, @vform(Vform)).
v_bar            macro (phrase, @v).
vp               macro (phrase, (synsem: @vp_y)).
s                macro (phrase, (synsem: @s_y)).
s_bar            macro (phrase, (synsem: @s_bar_y)).

main_v           macro (synsem: @main_v_y).
aux_v            macro (synsem: @aux_v_y).

v_cont           macro (synsem: @v_cont_y).
v_cont(X)        macro (synsem: @v_cont_y(X)).
v_nucl(X)        macro (synsem: @v_nucl_y(X)).

fin_v(Per,Num,S_Subj,Comps)
                 macro (synsem: @fin_v_y(Per,Num,S_Subj,Comps)).

 % < synsem >
v_y              macro (@verb_y, @v_cont_y, @spr_sat_y).
vp_y             macro (@v_y, @comps_sat_y).
s_y              macro (@v_y, @sat_y).
s_bar_y          macro (@v_y, @sat_y, @marking_y(marked)).
```



```
main_v_y         macro (@v_y, @minus_aux_y).
aux_v_y          macro (@v_y, @plus_aux_y).

aux_v_y(Vform)   macro (@aux_v_y, @vform_y(Vform)).
v_cont_y         macro (@cont_y( (psoa,(quants:[])))).
                     %  @conx_y((backgr: eset))).
v_cont_y(Cont)   macro (@v_cont_y, @cont_y(Cont)).
v_nucl_y(Nucl)   macro (@v_cont_y, @cont_y((nucl: Nucl))).

 % Note on fin_v:
 % fin_v enforces a singleton list subj value!
 % This is incorrect for real raising verbs, which don't enforce
 % the existence of a subject: e.g. "mich scheint zu frieren"

fin_v_y(Per,Num,S_Subj,Comps)
                 macro (@head_y(fin_v),
                        @fin_y,
                        @val_subj_y( [ (S_Subj,@noun,@nom,@per(Per),
                                                          @num(Num)) ] ),
                        @val_comps_y(Comps)).

n_fin_v_y(Vform,Subj,Comps)
                 macro (@head_y(n_fin_v), @vform_y( (Vform, n_fin) ),
                        @val_subj_y(Subj), @val_comps_y(Comps)).

 % Note on arg_raising_vp_compl:
 % The "n_sign_list" restriction on the raised arguments deserves some
 % notice.  As can be seen in the hierarchy, the introduction of that
 % type causes a flood of new types, some of which have to be introduced
 % to keep the hierarchy a lattice (no multiple mgus) and satisfy the
 % fic.  The alternative idea to put the restriction into the phrase
 % structure rules, in which auxiliaries construct, works (cleanly) only
 % via a specification on the mother of the vc rules, which keeps verbal
 % complements from getting passed on.  This is linguistically correct,
 % since no verbs subcategorize for two verbal complements and one always
 % gets realized in the vc schema. Note that head-subj-comps and
 % head-comps rules always realize all complements.  However this
 % technique cannot keep a finite auxiliary from directly combining with
 % two verbal complements, one of which was raised.  Therefore the
 % "n_sign_list" seems to be the best solution (even though it screws up
 % the hierarchy as noted).

arg_raising_vp_compl_y(Vp_vform, Vp_flip)
                 macro (@v_nucl_y( (arg:Vp_sem) ),
                        @val_subj_y(Subj),
                        @val_comps_y( [(@v(Vp_vform, Vp_sem, Subj,
                                           Vp_comps, Vp_flip), @minus_npcomp)
                                       | (Vp_comps, n_sign_list)] )).


 % ######################################
 % III. macros to specify lexical entries
 % ######################################

 % ;;;;;;;
 % general
 % ;;;;;;;

 % Note on all_lex:
 %   in_slash is empty, to_slash is not specified
 %   i.e finite intransitive verbs can immediately combine with filler
```

```
all_lex(S)          macro (word, (synsem: (S, @no_in_slash_y, @minus_npcomp_y)),
                           @no_lr).
det_lex1(S)         macro (@all_lex(S), @unmarked).
mark_lex1(S)        macro (@all_lex(S),              (qstore: eset)).

 % Note on subst_lex:
 %    Since so far no predicative constructions are treated in this fragment,
 %    minus_prd is included for all substantive entries.

subst_lex(S)        macro (@all_lex(S), @unmarked, (qstore: eset), @minus_prd).

 % Note on no_mod_lex:
 %    In this fragment the only elements which can modify, are adjectives.
 %    Therefore all other substantial elements should use the no_mod_lex macro.

no_mod_lex(S)       macro (@subst_lex(S), @no_mod).

 % ;;;;;;;;;;;;;;
 % * adjectives *
 % ;;;;;;;;;;;;;;

wk_adj_lex(Prop,Case,Gend,Num,Prd) macro @adj_lex(Prop,Case,Gend,Num,Prd,wk,st).
st_adj_lex(Prop,Case,Gend,Num,Prd) macro @adj_lex(Prop,Case,Gend,Num,Prd,st,wk).

adj_lex(Prop,Case_arg,Gend,Num,Prd,Adj_decl,Det_decl)
   macro (@subst_lex( (@adj_y,
                       @sat_y,
                       @case_y( (Case, Case_arg) ),
                       @gend_y(Gend),
                       @third_y,
                       @num_y(Num),
                       @prd_y(Prd),
                       @decl_y(Adj_decl),
                       @mod_y( (@noun_y, @case_y(Case),
                                @subj_sat_y, @comps_sat_y,
                                @val_spr_y( [(synsem: (@det_y, @sat_y,
                                                       @decl_y(Det_decl)))] ),
                                @index_y(I), @restr_y(N_restr)) ),
                       @index_y(I),
                       @restr_y(@set( ((quants: []),(nucl: (Prop, (inst:I)))),
                                      N_restr))))).

 % ;;;;;;;;;;;;;;
 % * determiner *
 % ;;;;;;;;;;;;;;


det_lex2(Case_arg,Gend,Num,Decl)
   macro @det_lex1( (@det_y,
                     @case_y( (Case, Case_arg) ),
                     @decl_y(Decl),
                     @spec_y( (@noun_y, @case_y(Case), @cont_y(N_cont),
                              @subj_sat_y, @comps_sat_y,
                              @val_spr_y([(@det, @sat)])) ),
                     @sat_y,
                     @cont_y( (quant,
                              (restind: (npro, N_cont,
                                         (index: ((per:third),
                                                  (num:Num),
                                                  (gend:Gend))))))))).
```



```
det_lex(Sem_det,Case,Gend,Num,Decl)
   macro (@det_lex2(Case,Gend,Num,Decl),
                   @cont( (Cont, (det:Sem_det)) ),
                   (qstore: @sset(Cont))).


 % ;;;;;;;;;;;
 % * markers *
 % ;;;;;;;;;;;

mark_lex(M)
    macro @mark_lex1( (@marker_y, @marking_y(M), @sat_y,
                       @spec_y( (@verb_y, @fin_y, @sat_y,
                                 @unmarked_y) )) ).


 % ;;;;;;;;;
 % * nouns *
 % ;;;;;;;;;

 % Notes on lexical hierarchy macros for nouns
 % - n_lex1 specifies information common to all nouns
 % - To accommodate the exceptional character of np_lex
 %   it does not inherit from all_lex (the general lex hierarchy). However
 %   n_lex inheriting from n_lex1 also inherits from the general hierarchy
 %   via no_mod_lex.

 % information common to all nominal lexical entries
n_lex1         macro (@noun, @subj_sat, @comps_sat, @per(third)).


 % noun with declension class specification
n_lex(Case,Num,Gend,Prop,Det_decl)
               macro (@n_lex1,
                      @case(Case),
                      @val_spr([ (@det, @sat, @case(Case)) ]),
                      @no_mod_lex( (@case_y(Case),
                                    @val_spr_y([ @decl(Det_decl) ]),
                                    @n_cont_y(Prop),
                                    @num_y(Num),
                                    @gend_y(Gend)))).


 % noun without declension class specification
n_indif_lex(Case,Num,Gend,Prop)
               macro @n_lex(Case,Num,Gend,Prop,_).


 % Notes on np_lex:
 % The following two macros are not inheriting from a sub-macro of all_lex !
 % - np_lex is exceptional in two ways: its type is lex_phrase, and
 %      it has a nonempty qstore.
 %
 % -> All non-colliding specifications of the general lexical hierarchy
 %    (relevant for nouns is no_mod_lex) are specified by hand.
 % -> The noun specific properties are specified via using n_lex1.

np_lex(Prop,Case,Gend) macro
   % standard no_mod_lex without word and qstore specification:
        (@no_in_slash, @minus_npcomp, @unmarked, @minus_prd, @no_mod,
   % standard noun:
         @n_lex1,
```

```
  % extra:
          lex_phrase,
          @n_cont(Prop),
          @sat,
          @num(sing), @gend(Gend), @case(Case),
          @cont(Cont),
          (qstore: @sset( ((det:def), (restind:Cont)) )),
          (retr: [])).


 % ;;;;;;;;;
 % * verbs *
 % ;;;;;;;;;

 % Note on the specification of verbal entries:
 % Only bse verbs need to be specified. Finite and psp forms are
 % obtained via lexical rules.

 % ---------------
 % a) auxiliaries
 % ---------------
 % Properties of auxiliaries:
 % 1. subcategorization frame of a vp embedding verb
 % 2. obligatory argument raising
 % 3. subject-raising
 %
 % Note concerning raising:
 % aux_lex enforces a singleton list subj value!
 % This is incorrect for real raising verbs, which don't enforce
 % the existence of a subject.
 %
 % Note concerning aux-flip:
 % No flip values are specified here, since bse auxs flip optionally.
 % Haben und werden are specified as inheriting the flip value of
 % its complement (to allow double flip, e.g.  wird haben kommen koennen)
 % via specifying flip sharing in addition to  aux_lex in aux_flip_id_lex.
 % The lexical rules forming finite and psp verbs then demand minus_flip,
 % and remove possibly existing flip reentrancies.

aux_lex(Qfpsoa, Vp_vform)
               macro @no_mod_lex( (@head_y(n_fin_v), @aux_v_y(bse),
                                  @v_nucl_y(Qfpsoa),
                                  @arg_raising_vp_compl_y(Vp_vform,_))).
aux_flip_id_lex(Qfpsoa, Vp_vform)
               macro @no_mod_lex( (@head_y(n_fin_v), @aux_v_y(bse),
                                  @v_nucl_y(Qfpsoa),
                                  @flip_y(Flip),
                                  @arg_raising_vp_compl_y(Vp_vform, Flip))).

 % -------------
 % b) main verbs
 % -------------

intrans_lex(Nucl,I0)
               macro @main_v_lex1(Nucl, @n_fin_v_y(bse, [@np(I0,_)], [])).
trans_lex(Nucl,I0,I1,C1)
               macro @main_v_lex1(Nucl, @n_fin_v_y(bse, [@np(I0,_)],
                                                     [@np(I1,C1)])).

ditrans_lex(Nucl,I0,I1,C1,I2,C2)
               macro @main_v_lex1(Nucl, @n_fin_v_y(bse, [@np(I0,_)],
                                                     [@np(I2,C2), @np(I1,C1)])).
```



```
 % - - - - - - - - - - -
 % auxiliary macros
 % - - - - - - - - - - -

 % Note on main_v_lex1:
 % Main verbs are all specified to be minus_flip.

 % < sign >
main_v_lex1(N,S) macro @no_mod_lex( (S, @main_v_y, @v_nucl_y(N), @minus_flip_y) ).



 % ****************************************************************
 %    Lexicon
 % ****************************************************************

 % ---------------------------------------------------
 % adjectives: [wk/st]_adj_lex(Prop,Case,Gend,Num,Prd)
 % ---------------------------------------------------

 % weak:
kleinen  ---> @wk_adj_lex(small, dat, _,            sing, minus).
kleinen  ---> @wk_adj_lex(small, _,   _,            plur, minus).
kleinen  ---> @wk_adj_lex(small, acc, masc,         sing, minus).
kleine   ---> @wk_adj_lex(small, nom, fem,          sing, minus).
kleine   ---> @wk_adj_lex(small, acc, fem,          sing, minus).
kleine   ---> @wk_adj_lex(small, nom, masc,         sing, minus).
kleine   ---> @wk_adj_lex(small, nom, neutr,        sing, minus).
kleine   ---> @wk_adj_lex(small, acc, neutr,        sing, minus).

 % strong:
kleinen  ---> @st_adj_lex(small, acc, masc,         sing, minus).
kleinen  ---> @st_adj_lex(small, dat, _,            plur, minus).
kleine   ---> @st_adj_lex(small, nom, fem,          sing, minus).
kleine   ---> @st_adj_lex(small, acc, fem,          sing, minus).
kleine   ---> @st_adj_lex(small, nom, _,            plur, minus).
kleine   ---> @st_adj_lex(small, acc, _,            plur, minus).
kleines  ---> @st_adj_lex(small, nom, neutr,        sing, minus).
kleines  ---> @st_adj_lex(small, acc, neutr,        sing, minus).
kleiner  ---> @st_adj_lex(small, dat, fem,          sing, minus).
kleiner  ---> @st_adj_lex(small, nom, masc,         sing, minus).
kleinem  ---> @st_adj_lex(small, dat, (masc;neutr), sing, minus).

 % weak:
gute    ---> @wk_adj_lex(good, nom,      _,    sing, minus).
guten   ---> @wk_adj_lex(good, acc,      masc, sing, minus).
gute    ---> @wk_adj_lex(good, acc,(fem;neutr),sing, minus).
guten   ---> @wk_adj_lex(good, dat,      _,    sing, minus).
guten   ---> @wk_adj_lex(good, _,        _,    plur, minus).

 % strong:
guter   ---> @st_adj_lex(good, nom,      masc, sing, minus).
gute    ---> @st_adj_lex(good, nom,      fem,  sing, minus).
gutes   ---> @st_adj_lex(good, nom,      neutr,sing, minus).
guten   ---> @st_adj_lex(good, acc,      masc, sing, minus).
gute    ---> @st_adj_lex(good, acc,      fem,  sing, minus).
gutes   ---> @st_adj_lex(good, acc,      neutr,sing, minus).
guten   ---> @st_adj_lex(good, dat,      _,    sing, minus).

gute    ---> @st_adj_lex(good, nom,     _,     plur, minus).
gute    ---> @st_adj_lex(good, acc,     _,     plur, minus).
```

```
guten   ---> @st_adj_lex(good, dat,     _,     plur, minus).

 % ---------------------------------------------------
 % determiners
 % ---------------------------------------------------

 % 1. definite (all strong)
der     ---> @det_lex(def, nom,       masc,  sing, st).
die     ---> @det_lex(def, nom,       fem,   sing, st).
die     ---> @det_lex(def, acc,       fem,   sing, st).
das     ---> @det_lex(def, nom,       neutr, sing, st).
das     ---> @det_lex(def, acc,       neutr, sing, st).

den     ---> @det_lex(def, acc, masc,         sing, st).

dem     ---> @det_lex(def, dat, (masc;neutr), sing, st).
der     ---> @det_lex(def, dat,       fem,    sing, st).

die     ---> @det_lex(def, nom,       _,      plur, st).
den     ---> @det_lex(def, dat,       _,      plur, st).
die     ---> @det_lex(def, acc,       _,      plur, st).


 % 2. indefinite


 % a) strong (not needed in PVP fragment)

 % b) weak

 % i. singular: can undergo indef_wk_det_lr and empty_ein_lr
ein     ---> @det_lex(indef, nom, (masc;neutr),  sing, wk).
eine    ---> @det_lex(indef, nom, fem,           sing, wk).
eine    ---> @det_lex(indef, acc, fem,           sing, wk).

einen   ---> @det_lex(indef, acc, masc,  sing, wk).
ein     ---> @det_lex(indef, acc, neutr, sing, wk).

einem   ---> @det_lex(indef, dat, (masc;neutr), sing, wk).
einer   ---> @det_lex(indef, dat, fem,          sing, wk).

 % ii. plural: can undergo indef_wk_det_lr
einige  ---> @det_lex(indef, nom, _, plur, wk).
einiger ---> @det_lex(indef, gen, _, plur, wk).
einige  ---> @det_lex(indef, acc, _, plur, wk).
einigen ---> @det_lex(indef, dat, _, plur, wk).

 % ---------------------------------------------------
 % marker: mark_lex(Marking)
 % ---------------------------------------------------

dass    ---> @mark_lex(dass).

 % ---------------------------------------------------
 % nouns: n_lex(Case,Num,Gend,Prop,Det_decl) (Det_decl optional)
 % ---------------------------------------------------

mann        ---> @n_indif_lex( nom, sing, masc,  man ).
mann        ---> @n_indif_lex( acc, sing, masc,  man ).
mann        ---> @n_indif_lex( dat, sing, masc,  man ).
maenner     ---> @n_indif_lex( nom, plur, masc,  man ).
```



```
maennern    ---> @n_indif_lex( acc, plur, masc,  man ).
maennern    ---> @n_indif_lex( dat, plur, masc,  man ).

tisch       ---> @n_indif_lex( nom, sing, masc,  table ).
tisch       ---> @n_indif_lex( acc, sing, masc,  table ).
tisch       ---> @n_indif_lex( dat, sing, masc,  table ).
tische      ---> @n_indif_lex( nom, plur, masc,  table ).
tische      ---> @n_indif_lex( acc, plur, masc,  table ).
tischen     ---> @n_indif_lex( dat, plur, masc,  table ).

buch        ---> @n_indif_lex( nom, sing, neutr, book ).
buch        ---> @n_indif_lex( acc, sing, neutr, book ).
buch        ---> @n_indif_lex( dat, sing, neutr, book ).
buecher     ---> @n_indif_lex( nom, plur, neutr, book ).
buecher     ---> @n_indif_lex( acc, plur, neutr, book ).
buechern    ---> @n_indif_lex( dat, plur, neutr, book ).

geschichte  ---> @n_indif_lex( _,   sing, fem,   story ).
geschichten ---> @n_indif_lex( _,   plur, fem,   story ).

 % nouns showing strong and weak forms:
 %
 % masc
verwandte   ---> @n_lex( nom, sing, masc,  relative, st ).
verwandten  ---> @n_lex( acc, sing, masc,  relative, st ).
verwandten  ---> @n_lex( dat, sing, masc,  relative, st ).
verwandten  ---> @n_lex( _,   plur, (masc;fem),  relative, st ).

verwandter  ---> @n_lex( nom, sing, masc,  relative, wk ).
verwandten  ---> @n_lex( acc, sing, masc,  relative, wk ).
verwandten  ---> @n_lex( dat, sing, masc,  relative, wk ).

verwandte   ---> @n_lex( nom, plur, (masc;fem),  relative, wk ).
verwandte   ---> @n_lex( acc, plur, (masc;fem),  relative, wk ).
verwandten  ---> @n_lex( dat, plur, (masc;fem),  relative, wk ).

 % fem
verwandte   ---> @n_lex( nom, sing, fem,   relative, st ).
verwandte   ---> @n_lex( acc, sing, fem,   relative, st ).
verwandten  ---> @n_lex( dat, sing, fem,   relative, st ).
 % fem plur = masc plur

verwandte   ---> @n_lex( nom, sing, fem,   relative, wk ).
verwandte   ---> @n_lex( acc, sing, fem,   relative, wk ).
verwandten  ---> @n_lex( dat, sing, fem,   relative, wk ).
 % fem plur = masc plur

 % ---------------------------------------------------
 % proper name nps
 % ---------------------------------------------------

karl   ---> @np_lex(karl,  (nom;dat;acc),   masc).
peter  ---> @np_lex(peter, (nom;dat;acc),   masc).
anna   ---> @np_lex(anna,  (nom;dat;acc),   fem).
sarah  ---> @np_lex(sarah, (nom;dat;acc),   fem).

 % To be able to test nps with case without having to
 % build them first:

karlN  ---> @np_lex(karl,  nom, masc).
karlD  ---> @np_lex(karl,  dat, masc).
karlA  ---> @np_lex(karl,  acc, masc).
```

```
peterN ---> @np_lex(peter, nom, masc).
peterD ---> @np_lex(peter, dat, masc).
peterA ---> @np_lex(peter, acc, masc).

annaN  ---> @np_lex(anna,  nom, fem).
annaD  ---> @np_lex(anna,  dat, fem).
annaA  ---> @np_lex(anna,  acc, fem).

sarahN ---> @np_lex(sarah, nom, fem).
sarahD ---> @np_lex(sarah, dat, fem).
sarahA ---> @np_lex(sarah, acc, fem).


 % ---------------------------------------------------
 % main_verbs
 % ---------------------------------------------------

 % intransitive
lachen    ---> @intrans_lex( (laugh, (laugher:X)), X ).

 % transitive
lieben    ---> @trans_lex( (love,(lover:X), (loved:Y)), X, Y, acc ).
finden    ---> @trans_lex( (find,(finder:X),(found:Y)), X, Y, acc ).
lesen     ---> @trans_lex( (read,(reader:X),(read:Y)),  X, Y, acc ).

 % ditransitive
geben     ---> @ditrans_lex( (give,(giver:X),(receiver:Y),(given:Z)), X,
                             Y, dat, Z, acc ).
erzaehlen ---> @ditrans_lex( (tell,(teller:X),(tellee:Y),(told:Z)), X,
                             Y, dat, Z, acc ).

 % database: v_entry(bse,3.sing,3.plur,(1=sein/2=haben), psp)
v_entry(lachen,    lacht,    lachen,    2, gelacht).
v_entry(lieben,    liebt,    lieben,    2, geliebt).
v_entry(finden,    findet,   finden,    2, gefunden).
v_entry(lesen,     liest,    lesen,     2, gelesen).
v_entry(geben,     gibt,     geben,     2, gegeben).
v_entry(erzaehlen, erzaehlt, erzaehlen, 2, erzaehlt).

 % ---------------------------------------------------
 % auxiliaries
 % ---------------------------------------------------

 % aux_lex(Qfpsoa, Vp_vform)
haben   ---> @aux_flip_id_lex(have,psp).
werden  ---> @aux_flip_id_lex(will,bse).
koennen ---> @aux_lex(can,bse).
muessen ---> @aux_lex(must,bse).
wollen  ---> @aux_lex(want,bse).

 % database: v_entry(bse,3.sing,3.plur,(1=sein/2=haben), psp)
v_entry(haben,     hat,      haben,     2, gehabt).
v_entry(werden,    wird,     werden,    1, geworden).
v_entry(koennen,   kann,     koennen,   2, gekonnt).
v_entry(muessen,   muss,     muessen,   2, gemusst).
v_entry(wollen,    will,     wollen,    2, gewollt).
```



```
 % ******************************************************************
 %    Lexical Rules
 % ******************************************************************

 % Each rule only applies once. If several applications would have been
 % necessary, the recursion was put into the def clauses attached: e.g.
 % several complements are extracted via celr in one application, using
 % a call to power_list.

:- lex_rule_depth(5).          % any specification > 0 will do

 % ######################################################
 %  I. Morphological rules
 % ######################################################

 %=======================================
 % 1. Simple morphological rules
 %=======================================
 % applies to: all indefinite determiners with semantic contribution
 % action: introduces negative indefinite
 % note:
 % All ``ein'' lexical entries, except those with no semantic
 % contribution (empty qstore) have a corresponding negative
 % indefinite entry.

ein2kein lex_rule
   (In,  @cont( (det:indef) ), (qstore: neset) )
**>
   (Out, @cont( (det:neg_indef) ))
if
  pass0( (In,  (lr: L)),
         (Out, (lr: [ein2kein | L ])))
morphs
   (einig,X) becomes (kein,X),
   ein       becomes kein,
   (eine,X)  becomes (keine,X).


 %=======================================
 % 2. Morphological with syntactic effect
 %=======================================

 % ----------------------------------
 % bse verbs -> fin verbs
 % ----------------------------------
 % applies to: bse verbs
 % action: introduces past participle forms for all verbs
 % note:
 % Since finite verbs with more than one slash element can never
 % be part of a sentence (only ditransitive *bse* verbs having slashed
 % both objects to occur as topicalized main verb only pvp make use
 % of more than one slash element) they only bse verbs with no or one
 % slash element can undergo finitivization. This however is only
 % to avoid useless lexical entries from occurring in the lexicon, not
 % doing so would not change the correctness of the grammar.

 % - - - - - - -
 % a) 3rd sing: version that inserts subj before comps
 % - - - - - - -

bse_to_3_sing lex_rule
   (In,  @bse, @val_subj([Subj]), @val_comps(Comps), @in_slash(Slash))
```

```
**>
   (Out, @inv(Inv), @fin_v(third,sing,Subj,Comps))
if
  (max_1_list(Slash,Inv),
   pass1a((In,  (lr:LR)),
          (Out, (lr:[bse_to_3_sing|LR]))))
morphs
   Bse becomes Sing when
      lookup_v(Bse,Sing,_,_,_).

 % - - - - - - -
 % b) 3rd plur: version that inserts subj before comps
 % - - - - - - -

bse_to_3_plur lex_rule
   (In,  @bse, @val_subj([Subj]), @val_comps(Comps), @in_slash(Slash))
**>
   (Out, @inv(Inv), @fin_v(third,plur,Subj,Comps))
if
   (max_1_list(Slash,Inv),
    pass1a((In,  (lr:LR)),
           (Out, (lr:[bse_to_3_plur|LR]))))
morphs
   Bse becomes Plur when
      lookup_v(Bse,_,Plur,_,_).

 % ----------------------------------
 % bse verbs -> psp verbs
 % ----------------------------------
 % applies to: bse verbs
 % action: introduces past participle forms for all verbs
 % note: only past participles formed with haben
 %       are part of the fragment.

bse_to_psp lex_rule
   (In,  @bse)
**>
   (Out, @psp, @minus_flip)
if
   pass1b((In,  (lr:LR)),
          (Out, (lr:[bse_to_psp|LR])))
morphs
   Bse becomes Psp when
      lookup_v(Bse,_,_,2,Psp).

 % ------------------------------------------
 % bse auxiliaries -> Ersatzinfinitv (= bse)
 % ------------------------------------------
 % applies to: bse auxiliaries
 % action: introduces substitute infinitives of modals
 %         and markes them as obligatory flip triggering
 %         (= plus_flip)

bse_to_subst_bse lex_rule
   (In, @aux_v, @bse)
**>
   (Out, @psp, @plus_flip)
if
   pass1b((In,  (lr:LR)),
          (Out, (lr:[bse_to_subst_bse|LR])))
morphs
   X becomes X.
```



```
 % #####################################################
 %  II. Introducing unbounded dependencies
 % #####################################################

 % ---------------------------------
 % Complement Extraction Lexical Rule
 % ---------------------------------
 % applies to: bse full verbs (aux:-)
 % action: extract any number of complements of bse verbs
 % note:
 %  - modified to account for slash taking
 %    sign- instead of local-values
 %  - psp is obtained by bse -> psp lr
 %  - fin is obtained by bse -> fin lr
 %    Note: maximally allow slash lists of length one
 %          (cf. comment at fin lr)

celr lex_rule
   (In,  @bse,
         @minus_aux,
         @val_comps(C1),
         @no_in_slash)
**>
   (Out, @val_comps(C2),
         @in_slash( (E,ne_list) ))
if
  (pass2((In,  (lr:LR)),
         (Out, (lr:[celr|LR]))),
   power_list(C1, E, C2))
morphs
   X becomes X.


 % ---------------------------------
 % Subject Extraction Lexical Rule
 % ---------------------------------
 % applies to: finite verbs
 % action: extracts the subject of finite verbs
 % note: All finite verbs with extracted subject
 %       appear in inverted constructions.

selr lex_rule
   (In,  @fin,
         @val_subj(Subj),
         @no_in_slash)
**>
   (Out, @subj_sat,
         @in_slash(Subj),
         @plus_inv)
if
  pass2b((In,  (lr:LR)),
         (Out, (lr:[selr|LR])))
morphs
   X becomes X.
```

```
 % ----------------------------------
 % PVP Extraction Lexical Rule
 % ----------------------------------
 % applies to: bse aux with empty in-slashes
 % action: shifts valence from comps to nonlocal (see paper for details)
 % note:
 %  - finites are then obtained via finitivization lex rules
 %  - The +INV in output is only relevant for finite auxiliaries

pvpelr lex_rule
   (In,  @verb, @plus_aux,
         @bse, @no_in_slash,
         @val_comps([ (@verb,
                      @vform(Vform),
                      @val_comps(Comps),
                      @val_subj(Subj),
                      @cont(Cont))
                    | Comps ]))
**>
   (Out, @val_comps((L, n_sign_list)),
         @in_slash([ (@verb,
                      @vform(Vform),
                      @comps_sat,
                      @val_subj(Subj),
                      @cont(Cont),
                      @in_slash(L)) ]))
if
   (pass2((In,  (lr:LR)),
          (Out, (lr:[pvpelr|LR]))))
morphs
   X becomes X.


 % ######################################################
 %  III. Specials
 % ######################################################

 %----------------------------------
 % Object Order Lexical Rule
 %----------------------------------
 % Not used in the test examples, since of no theoretical interest
 %----------------------------------
 % applies to: ditransitive verbs with comps dat < acc (standard in lexicon)
 % action: orders comps acc < dat
 %         thereby freeing word order of complements
 %         of ditransitive verbs realized together in
 %         a construction

%obj_order_lr lex_rule
%   (In,  @minus_aux,
%         @bse,
%         @val_comps([(A,@np,@dat), (B,@np,@acc)]))
%**>
%   (Out, @val_comps([B,A]))
%if
%   pass3((In,  (lr:LR)),
%         (Out, (lr:[obj_order_lr|LR]))))
%morphs
%   X becomes X.
```



```
 % ####################################################################
 % Passing of unchanged information for all lexical rules:
 % ####################################################################

 % --------------------------------------------------------------------
 % I. 1) Predicates and macros for morphological lexical rules
 % --------------------------------------------------------------------

 % carrying over everything but det (of quant subtype of cont)
 % note: only applies to determiners

pass0(@core0(Qstore,Retr,Nonloc,Cat,Restind,Conx),
      @core0(Qstore,Retr,Nonloc,Cat,Restind,Conx)) if true.

core0(Qstore,Retr,Nonloc,Cat,Restind,Conx) macro
   (word, (qstore:Qstore),
          (retr: Retr),
          (synsem: ((nonloc: Nonloc),
                    (loc: ((cat: Cat),
                           (cont: restind: Restind),
                           (conx: Conx)))))).

 % --------------------------------------------------------------------
 % I. 2) Predicates and macros for morphological lexical rules
 %       with syntactic effect
 % --------------------------------------------------------------------

 % carrying over everything but inv, vform, flip, val_comps and val_subj

pass1a(@core1a(Qstore,Retr,Aux,Mod,Prd,Marking,Npcomp,Val_spr,Cont,Conx,Nonloc),
       @core1a(Qstore,Retr,Aux,Mod,Prd,Marking,Npcomp,Val_spr,Cont,Conx,Nonloc))
   if true.

core1a(Qstore,Retr,Aux,Mod,Prd,Marking,Npcomp,Val_spr,Cont,Conx,Nonloc) macro
   (word, (qstore:Qstore),
          (retr: Retr),
          (synsem: (@aux_y(Aux),
                    @mod_y(Mod),
                    @prd_y(Prd),
                    @marking_y(Marking),
                    @npcomp_y(Npcomp),
                    @val_spr_y(Val_spr),
                    @cont_y(Cont),
                    @conx_y(Conx),
                    (nonloc: Nonloc)))).

 % --------------------------------------------------------------------
 % carrying over everything but vform and flip

pass1b(@core1b(Qstore,Retr,Aux,Inv,Mod,Prd,Marking,Npcomp,Val,Cont,Conx,Nonloc),
       @core1b(Qstore,Retr,Aux,Inv,Mod,Prd,Marking,Npcomp,Val,Cont,Conx,Nonloc))
   if true.

core1b(Qstore,Retr,Aux,Inv,Mod,Prd,Marking,Npcomp,Val,Cont,Conx,Nonloc) macro
   (word, (qstore:Qstore),
          (retr: Retr),
          (synsem: (@aux_y(Aux),
                    @inv_y(Inv),
                    @mod_y(Mod),
                    @prd_y(Prd),
                    @marking_y(Marking),
                    @npcomp_y(Npcomp),
```

```
                    @val_y(Val),
                    @cont_y(Cont),
                    @conx_y(Conx),
                    (nonloc: Nonloc)))).


 % ---------------------------------------------------------------------
 % database_access: lookup_v(bse,3.sing,3.plur,psp_aux_nr, psp)
 % to convert atoms (easier to write lexicon entries)
 % to lists (wanted by ALE)

lookup_v(Bse,Sing,Plur,Aux,Psp) :-
   v_entry(Bse1,Sing1,Plur1,Aux,Psp1),
   explode(Bse1,Bse),
   explode(Sing1,Sing),
   explode(Plur1,Plur),
   explode(Psp1,Psp).


 % ---------------------------------------------------------------------
 % III. Predicates and macros for udc introducing lexical rules
 % ---------------------------------------------------------------------
 % pass everything, but comps and in_slash

pass2(@core2(Qstore,Retr,Head,Marking,Npcomp,Val_subj,Val_spr,Cont,Conx,To_bind),
      @core2(Qstore,Retr,Head,Marking,Npcomp,Val_subj,Val_spr,Cont,Conx,To_bind))
   if true.

core2(Qstore,Retr,Head,Marking,Npcomp,Val_subj,Val_spr,Cont,Conx,To_bind) macro
   (word, (qstore: Qstore),
          (retr: Retr),
          (synsem: (@head_y(Head),
                    @marking_y(Marking),
                    @npcomp_y(Npcomp),
                    @val_subj_y(Val_subj),
                    @val_spr_y(Val_spr),
                    @cont_y(Cont),
                    @conx_y(Conx),
                    @to_bind_y(To_bind)))).

 % pass everything, but subj and in_slash

pass2b(@core2b(Qstore,Retr,Head,Marking,Npcomp,Val_comps,Val_spr,Cont,Conx,To_bind),
       @core2b(Qstore,Retr,Head,Marking,Npcomp,Val_comps,Val_spr,Cont,Conx,To_bind))
   if true.

core2b(Qstore,Retr,Head,Marking,Npcomp,Val_comps,Val_spr,Cont,Conx,To_bind) macro
   (word, (qstore: Qstore),
          (retr: Retr),
          (synsem: (@head_y(Head),
                    @marking_y(Marking),
                    @npcomp_y(Npcomp),
                    @val_comps_y(Val_comps),
                    @val_spr_y(Val_spr),
                    @cont_y(Cont),
                    @conx_y(Conx),
                    @to_bind_y(To_bind)))).
```



```
 % ---------------------------------------------------------------------
 % IV. Predicates and macros for word order freeing lexical rules
 % ---------------------------------------------------------------------
 % everything but the comps value stays the same

pass3(@core3(Qstore,Retr,Aux,Flip,Inv,Mod,Prd,Vform,Marking,Npcomp,
             Val_subj,Val_spr,Cont,Conx,Nonloc),
      @core3(Qstore,Retr,Aux,Flip,Inv,Mod,Prd,Vform,Marking,Npcomp,
             Val_subj,Val_spr,Cont,Conx,Nonloc)) if true.


core3(Qstore,Retr,Aux,Flip,Inv,Mod,Prd,Vform,Marking,Npcomp,
      Val_subj,Val_spr,Cont,Conx,Nonloc) macro
   (word, (qstore: Qstore),
          (retr: Retr),
          (synsem: (@aux_y(Aux),
                    @flip_y(Flip),
                    @inv_y(Inv),
                    @mod_y(Mod),
                    @prd_y(Prd),
                    @vform_y(Vform),
                    @marking_y(Marking),
                    @npcomp_y(Npcomp),
                    @val_subj_y(Val_subj),
                    @val_spr_y(Val_spr),
                    @cont_y(Cont),
                    @conx_y(Conx),
                    (nonloc:Nonloc)))).


 % *****************************************************************
 %    Phrase Structure Rules
 % *****************************************************************

 % General notes on ps rules:
 % - Principles applying to all rules are fed via the clause:
 %   principles(Mother, Syntactic_Head, Semantic_Head, List_of_Dtrs, Marking_Dtr)
 %   Marking_Dtr is the marker, if there is one, else the head
 % - Order on Comps List: <most oblique element ... least oblique element>
 % - Subject (least oblique np) is encoded in
 %     - subj attribute, or
 %     - extracted (only possible for finite verbs)

 %==================================================
 % (1) head subj complement
 %==================================================

 % ------------------------
 % finite sentences (inverted, or non-inverted)
 % ------------------------

 % - - - - - - -
 % a) inverted
 % - - - - - - -

 % intransitive verb + subj
 % aux + subj (vpelr aux)
hsc_plus_inv1 rule
   (Mother, @hsc_mother) ===>
      cat> (Head, @hsc_head([S],[]), @plus_inv),
      cat> (S, @subj_np),
      goal> hsc_goal(hsci1, Mother, Head, S, []).
```

```
 % transitive verb + obj + subj
 % aux + vc + subj
 % aux + obj1 + subj (vpelr aux)
hsc_plus_inv2 rule
   (Mother, @hsc_mother) ===>
      cat> (Head,  @hsc_head([S],[C]), @plus_inv),
      cat> (S, @subj_np),
      cat> C,
      goal> hsc_goal(hsci2, Mother, Head, S, [C]).

 % ditransitive verb + obj1 + obj2 + subj
 % aux + vc + obj + subj
 % aux + obj1 + obj2 + subj (vpelr aux)
hsc_plus_inv3 rule
   (Mother, @hsc_mother) ===>
      cat> (Head, @hsc_head([S],[C1,C2]), @plus_inv),
      cat> (S, @subj_np),
      cat> (C2, @obj_np),
      cat> C1,
      goal> hsc_goal(hsci3, Mother, Head, S, [C1,C2]).

 % aux + vc + obj1 + obj2 + subj
hsc_plus_inv4 rule
   (Mother, @hsc_mother) ===>
      cat> (Head, @hsc_head([S],[C1,C2,C3]), @plus_inv, @plus_aux),
      cat> (S, @subj_np),
      cat> (C3, @obj_np),
      cat> (C2, @obj_np),
      cat> C1,
      goal> hsc_goal(hsci4, Mother, Head, S, [C1,C2,C3]).

 % - - - - - - -
 % b) non-inverted
 % - - - - - - -

 % intransitive verb + subj
hsc_minus_inv1 rule
   (Mother, @hsc_mother) ===>
      cat> (S, @subj_np),
      cat> (Head,  @hsc_head([S],[]), @minus_inv, @minus_aux),
      goal> hsc_goal(hscni1, Mother, Head, S, []).

 % transitive verb + obj + subj
hsc_minus_inv2a rule
   (Mother, @hsc_mother) ===>
      cat> (S, @subj_np),
      cat> (C, @obj_np),
      cat> (Head,  @hsc_head([S],[C]), @minus_inv, @minus_aux),
      goal> hsc_goal(hscni2a, Mother, Head, S, [C]).

 % aux + vc + subj
hsc_minus_inv2a rule
   (Mother, @hsc_mother) ===>
      cat> (S, @subj_np),
      cat> (C, @minus_flip),
      cat> (Head,  @hsc_head([S],[C]), @minus_inv, @plus_aux),
      goal> hsc_goal(hscni2a, Mother, Head, S, [C]).

 % aux + vc[flip] + subj
hsc_minus_inv2b rule
   (Mother, @hsc_mother) ===>
```



```
      cat> (S, @subj_np),
      cat> (Head,  @hsc_head([S],[C]), @minus_inv, @plus_aux),
      cat> (C, @plus_flip),
      goal> hsc_goal(hscni2b, Mother, Head, S, [C]).

 % ditransitive verb + obj1 + obj2 + subj
hsc_minus_inv3a rule
   (Mother, @hsc_mother) ===>
      cat> (S, @subj_np),
      cat> (C2, @obj_np),
      cat> (C1, @noun),
      cat> (Head,  @hsc_head([S],[C1,C2]), @minus_inv, @minus_aux),
      goal> hsc_goal(hscni3a, Mother, Head, S, [C1,C2]).

 % aux + vc + obj + subj
hsc_minus_inv3a rule
   (Mother, @hsc_mother) ===>
      cat> (S, @subj_np),
      cat> (C2, @obj_np),
      cat> (C1, @minus_flip),
      cat> (Head,  @hsc_head([S],[C1,C2]), @minus_inv, @plus_aux),
      goal> hsc_goal(hscni3a, Mother, Head, S, [C1,C2]).

 % aux + vc[flip] + obj + subj
hsc_minus_inv3b rule
   (Mother, @hsc_mother) ===>
      cat> (S, @subj_np),
      cat> (C2, @obj_np),
      cat> (Head,  @hsc_head([S],[C1,C2]), @minus_inv, @plus_aux),
      cat> (C1, @plus_flip),
      goal> hsc_goal(hscni3b, Mother, Head, S, [C1,C2]).

 % aux + vc + obj1 + obj2 + subj
hsc_minus_inv4a rule
   (Mother, @hsc_mother) ===>
      cat> (S, @subj_np),
      cat> (C3, @obj_np),
      cat> (C2, @obj_np),
      cat> (C1, @minus_flip),
      cat> (Head,  @hsc_head([S],[C1,C2,C3]), @minus_inv, @plus_aux),
      goal> hsc_goal(hscni4a, Mother, Head, S, [C1,C2,C3]).

 % aux + vc[flip] + obj1 + obj2 + subj
hsc_minus_inv4b rule
   (Mother, @hsc_mother) ===>
      cat> (S, @subj_np),
      cat> (C3, @obj_np),
      cat> (C2, @obj_np),
      cat> (Head,  @hsc_head([S],[C1,C2,C3]), @minus_inv, @plus_aux),
      cat> (C1, @plus_flip),
      goal> hsc_goal(hscni4b, Mother, Head, S, [C1,C2,C3]).


 % --------
 % hsc core
 % --------

hsc_mother      macro (phrase, @plus_npcomp, @sat).
hsc_head(S,C)   macro (word, @verb, @fin,
                       @val_subj(S), @spr_sat, @val_comps(C),
                       @no_to_slash).
```

```
hsc_goal(Subtype, Mother, Head, Subj, Comps) if
   dtrs(Mother, @head_subj_comp_cs(Subtype, Head, Subj, Comps)),
   principles(Mother, Head, Head, [Head, Subj | Comps], Head).

 %=================================================
 % (2) head complement
 %=================================================

  % ------------------------
  % (A) Verbal complex construction
  %     = aux(n_fin) + vc
  % ------------------------

   % (vc + aux_n_fin)
   % Note: Subj and inherited arguments are passed up.
vc_minus_flip rule
   (Mother, @vc_mother(S,Rest)) ===>
      cat> (C, @vc_comp, @minus_flip),
      cat> (Head, @vc_head(S,[C|Rest])),
      goal> hc_goal(vcnf, Mother, Head, [C]).

   % (aux_n_fin + vc[flip])
vc_plus_flip rule
   (Mother, @vc_mother(S,Rest)) ===>
      cat> (Head, @vc_head(S,[C|Rest])),
      cat> (C, @vc_comp, @plus_flip),
      goal> hc_goal(vcf, Mother, Head, [C]).

  % ------------------------
  % (B) inverted finite sentences with extracted subject
  % ------------------------
  % Note on verbs with extracted subj:
  % Such verbs are marked +INV by the selr.

  % Note on vc_plus_inv:
  % Note that in the case of aux_fin+ vc no nominal complements are realized.
  % Nonetheless it is a head-complement rule and the mother bears the
  % specification [NPCOMPS +]. That specification is intended to mean "level
  % where NP elements are discharged, in case there are any to discharge"

  % (trans + obj)       subj extracted
  % (aux_fin + vc) subj extracted
hc_plus_inv1 rule
   (Mother, @hc_mother_fin) ===>
      cat> (Head, @hc_head_fin([C]), @plus_inv),
      cat> C,
      goal> hc_goal(hci1, Mother, Head, [C]).

  % (aux_fin + vc + obj) subj extracted
  % (ditrans + obj1 + obj2) subj extracted
hc_plus_inv2 rule
   (Mother, @hc_mother_fin) ===>
      cat> (Head, @hc_head_fin([C1,C2]), @plus_inv),
      cat> (C2, @obj_np),
      cat> C1,
      goal> hc_goal(hci2, Mother, Head, [C1,C2]).

  % (aux_fin + vc + obj1 + obj2) subj extracted
hc_plus_inv3 rule
   (Mother, @hc_mother_fin) ===>
      cat> (Head, @hc_head_fin([C1,C2,C3]), @plus_inv, @plus_aux),
      cat> (C3, @obj_np),
```



```
      cat> (C2, @obj_np),
      cat> (C1, @vc_comp),
      goal> hc_goal(hci3, Mother, Head, [C1,C2,C3]).

  % ------------------------
  % (C) non-inverted non-finite possibly partial vps (always topicalized)
  % ------------------------
  % Note: The subject is never empty, since only finite verbs
  %       can undergo selr and only non-finite verbs construct here.

    % 0-place verbs are topicalized as word

    % (verb + obj)
    % (vc + raised obj)
hc_minus_inv1 rule
   (Mother, @hc_mother(S)) ===>
      cat> (C, @obj_np),
      cat> (Head,  @hc_head_n_fin(S,[C]), @minus_inv),
      goal> hc_goal(hcni1, Mother, Head, [C]).

    % (verb + obj1 + obj2)
    % (vc + raised obj1 + raised obj2)
hc_minus_inv2 rule
   (Mother, @hc_mother(S)) ===>
      cat> (C2, @obj_np),
      cat> (C1, @obj_np),
      cat> (Head,  @hc_head_n_fin(S,[C1,C2]), @minus_inv),
      goal> hc_goal(hcni2, Mother, Head, [C1,C2]).

 % -------
 % vc core
 % -------

vc_mother(S,C)      macro (@verb,
                           @minus_npcomp,
                           @val_subj(S), @spr_sat, @val_comps(C)).
vc_head(S,C)        macro (word,
                           @n_fin,
                           @hc_head(S,C)).
vc_comp             macro @verb.

 % -------
 % hc core
 % -------

hc_mother(S)        macro (@verb,
                           @comps_sat,
                           @plus_npcomp,
                           @val_subj(S), @spr_sat).
hc_mother_fin       macro @hc_mother([]).

hc_head(S,C)        macro (@val_subj(S), @spr_sat, @val_comps(C),
                           @no_to_slash).
hc_head_n_fin(S,C) macro (@hc_head(S ,C), @n_fin).
hc_head_fin(C)      macro (word, @hc_head([],C), @fin).


hc_goal(Subtype, Mother, Head, Comps) if
   dtrs(Mother, @head_comp_cs(Subtype, Head, Comps)),
   principles(Mother, Head, Head, [Head|Comps], Head).
```

```
%==================================================
% (3) marker head
%==================================================

marker_head rule
   (Mother, phrase, @val(Val)) ===>
cat>
   Marker,
cat>
   (Head, @minus_inv, @unmarked, @val(Val),
          @no_to_slash),
goal>
   (spec_p(Head,Marker),
    dtrs(Mother, @marker_head_cs(Marker,Head)),
    principles(Mother,Head,Head,[Head,Marker],Marker)).

%==================================================
% (4) spr head
%==================================================

% Note on spr_head:
% Specifier and head select each other. This causes cyclic structures
% to arise in the head-dtr as well as the mother of spr_head constructions!
% Since ALE represents cyclic feature structures by cyclic PROLOG structures,
% versions of Sicstus before 2.1 #7 and Quintus will bomb upon trying to
% parse such a construction. The rest of the grammar works with
% older versions of Sicstus (0.7) as well.

spr_head rule
   (Mother, @sat) ===>
cat>
   (Spr, @sat),
cat>
   (Head, @val_spr([Spr]), @subj_sat, @comps_sat,
          @no_to_slash),
goal>
   (spec_p(Head,Spr),
    dtrs(Mother, @spr_head_cs(Spr,Head)),
    principles(Mother,Head,Head,[Head,Spr],Head)).

%==================================================
% (5) adjunct head
%==================================================

adjunct_head rule
   (Mother, phrase, @val(Val)) ===>
cat>
   (Adjunct, @mod(Synsem)),
cat>
   (Head, (synsem: Synsem), @val(Val),
           @no_to_slash),
goal>
   (dtrs(Mother, @adjunct_head_cs(Adjunct,Head)),
    principles(Mother,Head,Adjunct,[Head,Adjunct],Head)).

%==================================================
% (6) filler head
%==================================================

filler_head rule
   (Mother, phrase, @sat) ===>
cat>
```



```
   (Filler, @in_slash(Filler_in_slash_list)),
cat>
   (Head, @verb, @fin, @sat, @plus_inv,
          @in_slash([Filler]),
          @to_slash([Filler | Filler_in_slash_list])),
goal>
   (dtrs(Mother, @filler_head_cs(Filler,Head)),
    principles(Mother,Head,Head,[Head,Filler],Head)).

% *****************************************************************
%   Definite Clauses
% *****************************************************************

% #################
% I. Non-Linguistic
% #################

% ---------------
% lists of quant
% ---------------

append_quant([],L,L) if true.
append_quant([H|T1],L,[H|T2]) if
  append_quant(T1,L,T2).

% insert_quant(+element, +list, -(list&element)
%
insert_quant(X,L,R) if list_del_quant(X,R,L).

% list_del_quant(quant,quant_list,quant_list - quant)
%
list_del_quant(X, [X|R], R) if true.
list_del_quant(X, [Y|R1], [Y|R2]) if
   list_del_quant(X,R1,R2).

% --------------
% lists of quant and set of quant
% --------------

% set_sublist_rest_quant(Set1,List,Set2)
% Extracts elements out of Set1 and add them in any order to List, the
% rest is stored in Set2.

set_sublist_rest_quant(eset,[],eset) if  true.
set_sublist_rest_quant(@set(H,T),L,@set(H,R)) if
  set_sublist_rest_quant(T,L,R).
set_sublist_rest_quant(@set(H,T),L,R) if
  set_sublist_rest_quant(T,L1,R),
  insert_quant(H,L1,L).

% ---------------
% lists of signs
% ---------------

max_1_list([], _) if true.
max_1_list([_], plus) if true.

append([],L,L) if true.
append([H|T1],L,[H|T2]) if
  append(T1,L,T2).
```

```
 % list_del(element,list,list - element)
 %
list_del(X, [X|R], R) if true.
list_del(X, [Y|R1], [Y|R2]) if
   list_del(X,R1,R2).

 % power_list(list1, powerlist_element of list 1, elements not in powerlist_element)
 %
power_list(L,L,[]) if true.
power_list(L,M,[F|R]) if
   list_del(F,L,O),
   power_list(O,M,R).

 % list_diff(+list1,+list2,-(list1 - elements of list2))  (-> token-identity-check)
 %
list_diff(X, [], X) if true.
list_diff(L, [E|F], R) if
   list_del(E, L, L1),
   list_diff(L1, F, R).

 % switch_list_element(element_old,element_new,list_old,list_new)
 %
switch_list_element(E,E1,[E|R],[E1|R]) if true.
switch_list_element(E,E1,[F|R],[F|R1]) if
   switch_list_element(E,E1,R,R1).

 % ----
 % sets
 % ----

 % union_quant(set1_quant,set2_quant,result_quant) = set version of append_quant
 %
union_quant(eset,X,X) if true.
union_quant(@set(X,R),Y,@set(X,Z)) if
   union_quant(R,Y,Z).

 % - - - - - - -
 % specific sets
 % - - - - - - -

 % union_qstore(+list(dtrs), -union(qstore_of_dtrs))
 %
union_qstore([], eset) if true.
union_qstore([(qstore: Q) | R], S) if
   union_qstore(R,S1),
   union_quant(Q,S1,S).

 % collect_in_slash(+list(dtrs), -list(inherited slashes))
 %
collect_in_slash([], []) if true.
collect_in_slash([@in_slash(In) | R], In2) if
   collect_in_slash(R,In1),
   append(In,In1,In2).

 % #################
 % II. Non Linguistic on theory-specific data types
 % #################

 % building parse tree (separated here from real linguistics):
 %

dtrs((dtrs:Const_struc), Const_struc) if true.
```



```
 % ##############
 % III. Linguistic
 % ##############

 % ==========================================
 % A. Principles applied to all constructions
 % ==========================================

principles(Mother, Head, Sem_head, Dtr_list, Marking_dtr) if
   (hfp(Mother, Head),
    nfp(Mother, Head, Dtr_list),
    sem_p(Mother, Sem_head, Dtr_list),
    marking_p(Mother,Marking_dtr)).

 % ----------------------
 % Head-Feature Principle
 % ----------------------

hfp(@head(X),@head(X)) if true.

 % ------------------------
 % Nonlocal Principle
 % ------------------------

nfp(@in_slash(Diff), @to_slash(Head_To_slash), Dtr_list) if
   (collect_in_slash(Dtr_list,In_slash),
    list_diff(In_slash,Head_To_slash,Diff)).

 % -------------------
 % Semantics Principle
 % -------------------

sem_p( (@cont(Cont), qstore:Qstore, retr:Retr),
       @cont(Cont_Hd),
       Dtrs) if
  union_qstore(Dtrs, Qstore_Dtrs),
  sem_p_choice(Cont, Qstore, Retr, Cont_Hd, Qstore_Dtrs).

sem_p_choice( (psoa,(quants:Q,    nucl:N)), Qstore, Retr,
              (psoa,(quants:Q_Hd, nucl:N)), Qstore_Dtrs) if
   set_sublist_rest_quant(Qstore_Dtrs,Retr,Qstore),
   append_quant((Retr,e_list),Q_Hd,Q). % no quantifiers retrieved (cf. uDRS)

sem_p_choice((Cont,(nom_obj;quant)),Qstore,[],Cont,Qstore) if true.

 % ------------------------
 % Marking Principle
 % ------------------------

marking_p(@marking(X), @marking(X)) if true.

 % ==========================================
 % B. Principles applied to some constructions
 % ==========================================

 % -------------------
 % Specifier Principle
 % -------------------
 % applied to:
 % head-marker and head-specifier constructions
 % ( = constructions with a functional daughter,
 %     i.e. whose head value is "func")
```

```
spec_p( (synsem:                   Head_synsem),
        (synsem:loc:cat:head:spec: Head_synsem) ) if true.


 % *******************************************************************
 %  System stuff for use in the grammar
 % *******************************************************************

 % Predicates used for lexicon access:

transform([],[]).
transform([A|R1],[B|R2]) :-
   name(B,[A]),
   transform(R1,R2).

 % explode(+,-): converts atoms to lists of characters; very
 % inefficient and works only in one direction
 %
explode(In,Out) :-
   name(In,L1),
   transform(L1,Out).


 % *******************************************************************
 %  Test sentences
 % *******************************************************************
 %
 % A note on the number of solutions
 % ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 % Most test sentences only have one admissible structure. If more than
 % one solution is found, the implementation is incorrect. This is not
 % true for certain cases:
 %
 % 1. NPs can have several parses because of one morphology fitting
 %    several case and number specifications.
 %
 % 2. Finite sentences with intransitive verbs can be inverted or
 %    non-inverted. The following pairs are therefore indistinguishable:
 % a) finite verb in last position  = finite verb in second position
 % b) finite verb flipped before vc = finite verb in second position
 %
 % 3. In verbal complexes in which the highest auxiliary has bse
 % morphology, that auxiliary is ambiguous between bse and subst-bse.
 %
 % Note: The examples marked with a * have no correct parses.
 %
 %
 % ================================================
 %  I. General Constructions
 % ================================================

 % NPs

t(1,[der,kleine,mann]).
t(2,[ein,guter,tisch]).
t(3,[einige,kleine,verwandte]).
t(4,[kein,guter,kleiner,tisch]).
```



```
 % Verbal projections

 %%%%%%%%%%
 % 1 verb
 %%%%%%%%%%

 % A. VPs

 % bse
t(5,[das,gute,buch,finden]).
t(6,[dem,verwandten,das,gute,buch,geben]).

 % psp
t(7,[das,gute,buch,gefunden]).
t(8,[dem,verwandten,das,gute,buch,gegeben]).


 % B. Sentences

 % non-inverted
t(9,[der,kleine,mann,lacht]).
t(10,[der,kleine,mann,das,gute,buch,findet]).
t(11,[der,kleine,mann,dem,verwandten,das,gute,buch,gibt]).

 % non-inverted & complementizer
t(12,[dass,der,kleine,mann,lacht]).
t(13,[dass,der,kleine,mann,das,gute,buch,findet]).
t(14,[dass,der,kleine,mann,dem,verwandten,das,gute,buch,gibt]).

 % inverted
t(15,[lacht,der,kleine,mann]).
t(16,[findet,der,kleine,mann,das,gute,buch]).
t(17,[gibt,der,kleine,mann,dem,verwandten,das,gute,buch]).

 % inverted + subj topicalized:
t(18,[der,kleine,mann,lacht]).
t(19,[der,kleine,mann,findet,das,gute,buch]).
t(20,[der,kleine,mann,gibt,dem,verwandten,das,gute,buch]).

 % inverted + acc-obj topicalized:
t(21,[das,gute,buch,findet,der,kleine,mann]).
t(22,[das,gute,buch,gibt,der,kleine,mann,dem,verwandten]).

 % inverted + dat-obj topicalized:
t(23,[dem,verwandten,gibt,der,kleine,mann,das,gute,buch]).


 %%%%%%%%%%
 % 2 verbs
 %%%%%%%%%%

 % A. VPs

 % bse
t(24,[das,gute,buch,finden,wollen]).
t(25,[dem,verwandten,das,gute,buch,geben,wollen]).

 % psp
t(26,[das,gute,buch,gefunden,haben]).
t(27,[dem,verwandten,das,gute,buch,gegeben,haben]).
```

```
 % B. Sentences

 % non-inverted
t(28,[der,kleine,mann,lachen,wird]).
t(29,[der,kleine,mann,das,gute,buch,finden,wird]).
t(30,[der,kleine,mann,dem,verwandten,das,gute,buch,geben,wird]).

 % non-inverted & complementizer
t(31,[dass,der,kleine,mann,lachen,wird]).
t(32,[dass,der,kleine,mann,das,gute,buch,finden,wird]).
t(33,[dass,der,kleine,mann,dem,verwandten,das,gute,buch,geben,wird]).

 % inverted
t(34,[wird,der,kleine,mann,lachen]).
t(35,[wird,der,kleine,mann,das,gute,buch,finden]).
t(36,[wird,der,kleine,mann,dem,verwandten,das,gute,buch,geben]).

 % inverted + subj topicalized:
t(37,[der,kleine,mann,wird,lachen]).
t(38,[der,kleine,mann,wird,das,gute,buch,finden]).
t(39,[der,kleine,mann,wird,dem,verwandten,das,gute,buch,geben]).

 % inverted + acc-obj topicalized:
t(40,[das,gute,buch,wird,der,kleine,mann,finden]).
t(41,[das,gute,buch,wird,der,kleine,mann,dem,verwandten,geben]).

 % inverted + dat-obj topicalized:
t(42,[dem,verwandten,wird,der,kleine,mann,das,gute,buch,geben]).


 %%%%%%%%%%
 % 3 verbs
 %%%%%%%%%%

 % A. VPs

 % bse
t(43,[das,gute,buch,finden,wollen,werden]).
t(44,[dem,verwandten,das,gute,buch,geben,wollen,werden]).

 % psp
t(45,[das,gute,buch,gefunden,haben,werden]).
t(46,[dem,verwandten,das,gute,buch,gegeben,haben,werden]).


 % B. Sentences

 % non-inverted
t(47,[der,kleine,mann,lachen,wollen,wird]).
t(48,[der,kleine,mann,das,gute,buch,finden,wollen,wird]).
t(49,[der,kleine,mann,dem,verwandten,das,gute,buch,geben,wollen,wird]).

 % non-inverted & complementizer
t(50,[dass,der,kleine,mann,lachen,wollen,wird]).
t(51,[dass,der,kleine,mann,das,gute,buch,finden,wollen,wird]).
t(52,[dass,der,kleine,mann,dem,verwandten,das,gute,buch,geben,wollen,wird]).

 % inverted
t(53,[wird,der,kleine,mann,lachen,wollen]).
t(54,[wird,der,kleine,mann,das,gute,buch,finden,wollen]).
t(55,[wird,der,kleine,mann,dem,verwandten,das,gute,buch,geben,wollen]).
```



```
 % inverted + subj topicalized:
t(56,[der,kleine,mann,wird,lachen,wollen]).
t(57,[der,kleine,mann,wird,das,gute,buch,finden,wollen]).
t(58,[der,kleine,mann,wird,dem,verwandten,das,gute,buch,geben,wollen]).

 % inverted + acc-obj topicalized:
t(59,[das,gute,buch,wird,der,kleine,mann,finden,wollen]).
t(60,[das,gute,buch,wird,der,kleine,mann,dem,verwandten,geben,wollen]).

 % inverted + dat-obj topicalized:
t(61,[dem,verwandten,wird,der,kleine,mann,das,gute,buch,geben,wollen]).


 %%%%%%%%%%
 % 4 verbs
 %%%%%%%%%%

 % A. Vps

 % bse
t(62,[das,gute,buch,finden,koennen,wollen,werden]).
t(63,[dem,verwandten,das,gute,buch,geben,koennen,wollen,werden]).

 % psp -> flipped
t(64,[das,gute,buch,werden,haben,finden,koennen]).
t(65,[dem,verwandten,das,gute,buch,werden,haben,geben,koennen]).


 % B. Sentences

 % non-inverted
t(66,[der,kleine,mann,lachen,koennen,wollen,wird]).
t(67,[der,kleine,mann,das,gute,buch,finden,koennen,wollen,wird]).
t(68,[der,kleine,mann,dem,verwandten,das,gute,buch,geben,koennen,wollen,wird]).

 % non-inverted & complementizer
t(69,[dass,der,kleine,mann,lachen,koennen,wollen,wird]).
t(70,[dass,der,kleine,mann,das,gute,buch,finden,koennen,wollen,wird]).
t(71,[dass,der,kleine,mann,dem,verwandten,das,gute,buch,geben,koennen,wollen,wird]).

 % inverted
t(72,[wird,der,kleine,mann,lachen,koennen,wollen]).
t(73,[wird,der,kleine,mann,das,gute,buch,finden,koennen,wollen]).
t(74,[wird,der,kleine,mann,dem,verwandten,das,gute,buch,geben,koennen,wollen]).

 % inverted + subj topicalized:
t(75,[der,kleine,mann,wird,lachen,koennen,wollen]).
t(76,[der,kleine,mann,wird,das,gute,buch,finden,koennen,wollen]).
t(77,[der,kleine,mann,wird,dem,verwandten,das,gute,buch,geben,koennen,wollen]).

 % inverted + acc-obj topicalized:
t(78,[das,gute,buch,wird,der,kleine,mann,finden,koennen,wollen]).
t(79,[das,gute,buch,wird,der,kleine,mann,dem,verwandten,geben,koennen,wollen]).

 % inverted + dat-obj topicalized:
t(80,[dem,verwandten,wird,der,kleine,mann,das,gute,buch,geben,koennen,wollen]).
```

```
% ==============================================
%  II. Special Constructions
% ==============================================

% ----------------------------------------------
% P(VP) Topicalization
% ----------------------------------------------

%%%%%%%%%%
% 1 verb
%%%%%%%%%%

% main v
t(81,[lachen,wird,der,kleine,mann]).
t(82,[finden,wird,der,kleine,mann,das,gute,buch]).
t(83,[geben,wird,der,kleine,mann,dem,verwandten,das,gute,buch]).

% main v + obj
t(84,[das,gute,buch, finden,wird,der,kleine,mann]).
t(85,[das,gute,buch,geben,wird,der,kleine,mann,dem,verwandten]).
t(86,[dem,verwandten,geben,wird,der,kleine,mann,das,gute,buch]).

% main v + obj1 + obj2
t(87,[dem,verwandten,das,gute,buch,geben,wird,der,kleine,mann]).

%%%%%%%%%%
% 2 verbs
%%%%%%%%%%

% a) aux stays in back

% main v
t(88,[lachen,wird,der,kleine,mann,koennen]).
t(89,[finden,wird,der,kleine,mann,das,gute,buch,koennen]).
t(90,[geben,wird,der,kleine,mann,dem,verwandten,das,gute,buch,koennen]).

% main v + obj
t(91,[das,gute,buch, finden,wird,der,kleine,mann,koennen]).
t(92,[das,gute,buch,geben,wird,der,kleine,mann,dem,verwandten,koennen]).
t(93,[dem,verwandten,geben,wird,der,kleine,mann,das,gute,buch,koennen]).

% main v + obj1 + obj2
t(94,[dem,verwandten,das,gute,buch,geben,wird,der,kleine,mann,koennen]).

% b) aux topicalized as well

% main v + aux
t(95,[lachen,koennen,wird,der,kleine,mann]).
t(96,[finden,koennen,wird,der,kleine,mann,das,gute,buch]).
t(97,[geben,koennen,wird,der,kleine,mann,dem,verwandten,das,gute,buch]).

% main v + aux + obj
t(98,[das,gute,buch,finden,koennen,wird,der,kleine,mann]).
t(99,[das,gute,buch,geben,koennen,wird,der,kleine,mann,dem,verwandten]).
t(100,[dem,verwandten,geben,koennen,wird,der,kleine,mann,das,gute,buch]).

% main v + aux + obj1 + obj2
t(101,[dem,verwandten,das,gute,buch,geben,koennen,wird,der,kleine,mann]).
```



```
%%%%%%%%%%
% 3 verbs
%%%%%%%%%%

% a) both auxs stay in back

% main v
t(102,[lachen,wird,der,kleine,mann,koennen,wollen]).
t(103,[finden,wird,der,kleine,mann,das,gute,buch,koennen,wollen]).
t(104,[geben,wird,der,kleine,mann,dem,verwandten,das,gute,buch,koennen,wollen]).

% main v + obj
t(105,[das,gute,buch,finden,wird,der,kleine,mann,koennen,wollen]).
t(106,[das,gute,buch,geben,wird,der,kleine,mann,dem,verwandten,koennen,wollen]).
t(107,[dem,verwandten,geben,wird,der,kleine,mann,das,gute,buch,koennen,wollen]).

% main v + obj1 + obj2
t(108,[dem,verwandten,das,gute,buch,geben,wird,der,kleine,mann,koennen,wollen]).
% b) one aux topicalized as well

% main v + aux
t(109,[lachen,koennen,wird,der,kleine,mann,wollen]).
t(110,[finden,koennen,wird,der,kleine,mann,das,gute,buch,wollen]).
t(111,[geben,koennen,wird,der,kleine,mann,dem,verwandten,das,gute,buch,wollen]).

% main v + aux + obj
t(112,[das,gute,buch,finden,koennen,wird,der,kleine,mann,wollen]).
t(113,[das,gute,buch,geben,koennen,wird,der,kleine,mann,dem,verwandten,wollen]).
t(114,[dem,verwandten,geben,koennen,wird,der,kleine,mann,das,gute,buch,wollen]).

% main v + aux + obj1 + obj2
t(115,[dem,verwandten,das,gute,buch,geben,koennen,wird,der,kleine,mann,wollen]).
% c) both auxs topicalized

% main v + aux + aux
t(116,[lachen,koennen,wollen,wird,der,kleine,mann]).
t(117,[finden,koennen,wollen,wird,der,kleine,mann,das,gute,buch]).
t(118,[geben,koennen,wollen,wird,der,kleine,mann,dem,verwandten,das,gute,buch]).

% main v + aux + aux + obj
t(119,[das,gute,buch,finden,koennen,wollen,wird,der,kleine,mann]).
t(120,[das,gute,buch,geben,koennen,wollen,wird,der,kleine,mann,dem,verwandten]).
t(121,[dem,verwandten,geben,koennen,wollen,wird,der,kleine,mann,das,gute,buch]).

% main v + aux + aux + obj1 + obj2
t(122,[dem,verwandten,das,gute,buch,geben,koennen,wollen,wird,der,kleine,mann]).


% ----------------------------------------------
% AUX Flip
% ----------------------------------------------

%%%%%%%%%%
% 3 verbs
%%%%%%%%%%

% Optional
t(123,[dass,der,kleine,mann,wird,lachen,wollen]).
t(124,[dass,der,kleine,mann,das,gute,buch,wird,finden,wollen]).
t(125,[dass,der,kleine,mann,dem,verwandten,das,gute,buch,wird,geben,wollen]).
```

```
% Obligatory (substitute infinitive)
t(126,[dass,der,kleine,mann,hat,lachen,wollen]).
t(127,[dass,der,kleine,mann,das,gute,buch,hat,finden,wollen]).
t(128,[dass,der,kleine,mann,dem,verwandten,das,gute,buch,hat,geben,wollen]).


%%%%%%%%%%
% 4 verbs
%%%%%%%%%%

% Optional
t(129,[dass,der,kleine,mann,wird,lachen,koennen,wollen]).
t(130,[dass,der,kleine,mann,das,gute,buch,wird,finden,koennen,wollen]).
t(131,[dass,der,kleine,mann,dem,verwandten,das,gute,buch,wird,geben,koennen,wollen]).
t(132,[dass,der,kleine,mann,wird,wollen,lachen,koennen]).
t(133,[dass,der,kleine,mann,das,gute,buch,wird,wollen,finden,koennen]).
t(134,[dass,der,kleine,mann,dem,verwandten,das,gute,buch,wird,wollen,geben,koennen]).

% Obligatory (substitute infinitive)
t(135,[dass,der,kleine,mann,wird,haben,lachen,koennen]).
t(136,[dass,der,kleine,mann,das,gute,buch,wird,haben,finden,koennen]).
t(137,[dass,der,kleine,mann,dem,verwandten,das,gute,buch,wird,haben,geben,koennen]).


% ===============================================
%  III. Some interesting linguistic examples
% ===============================================

% Note: ungrammatical examples are marked with a *

% -----------------------------------------------
%  AUX Flip
% -----------------------------------------------

% --------------
% 1. obligatory
% --------------

% a) VL

% double aux flip (wollen = substitute infinitive)
t(138,[dass,karl,das,buch,wird,haben,lesen,wollen]).

% wird not flipped: ungrammatical
t(139,[dass,karl,das,buch,wird,lesen,wollen,haben]).   % *
t(140,[dass,karl,das,buch,wird,lesen,gewollt,haben]).  % *

% both auxs not flipped:
t(141,[dass,karl,das,buch,lesen,wollen,haben,wird]).   % *
t(142,[dass,karl,das,buch,lesen,gewollt,haben,wird]).

% b) V2

% double aux flip (wollen = substitute infinitive)
t(143,[karl,wird, das,buch,haben,lesen,wollen]).

t(144,[karl,wird,das,buch,lesen,wollen,haben]).   % *
t(145,[karl,wird,das,buch,lesen,gewollt,haben]).
```



```
% --------------
% 2. optional
% --------------

% a) VL

% flipped:
t(146,[dass,der,mann,anna,wird,finden,muessen]).

% not flipped:
t(147,[dass,der,mann,anna,finden,muessen,wird]).

% no V2, since only werden and haben are assumed to be able to flip


% -----------------------------------------------
%  Topicalization
% -----------------------------------------------

% --------------
% 1. NPs
% --------------

% Nom NP
t(148,[der,kleine,verwandte,hat,gelacht]).

% Akk NP
t(149,[das,buch,hat,karl,lesen,wollen]).

% Dat NP
t(150,[dem,mann,hat,sarah,die,geschichte,erzaehlen,muessen]).


% --------------
% 2. possibly partial VPs
% --------------

%main v
t(151,[lachen,wird,anna]).

% partial vp
t(152,[ein,kleines,buch,geben,wird,sarah,karl]).
t(153,[den,verwandten,erzaehlen,muss,anna,die,geschichte]).

% full vp
t(154,[dem,kleinen,mann,das,gute,buch,geben,wird,karl]).


% -----------------------------------------------
%  Some fancy stuff
% -----------------------------------------------

t(155,[karl,geben,muessen,wird,anna,das,buch,haben,wollen]).
t(156,[einigen,kleinen,maennern,erzaehlt,haben,muessen,wird,der,verwandte,die,gute,geschichte,wollen]).
```

Appendix B: A test run

For this test run, ALE 1.0 (release 1 May 1993) under SICStus 2.1 #9 was used on a Sun SPARC 10 with two 40MHz Super Sparc CPUs and 48Mb Ram.

```
| ?- test_all.
 * 1 >> 0 der 1 kleine 2 mann 3
O.k.    ** CPU time used (sec): 0.2,  Solutions:  1
******************************************************************
 * 2 >> 0 ein 1 guter 2 tisch 3
O.k.    ** CPU time used (sec): 0.1,  Solutions:  1
******************************************************************
 * 3 >> 0 einige 1 kleine 2 verwandte 3
O.k.    ** CPU time used (sec): 0.7,  Solutions:  4
******************************************************************
 * 4 >> 0 kein 1 guter 2 kleiner 3 tisch 4
O.k.    ** CPU time used (sec): 0.2,  Solutions:  1
******************************************************************
 * 5 >> 0 das 1 gute 2 buch 3 finden 4
O.k.    ** CPU time used (sec): 0.5,  Solutions:  1
******************************************************************
 * 6 >> 0 dem 1 verwandten 2 das 3 gute 4 buch 5 geben 6
O.k.    ** CPU time used (sec): 1.0,  Solutions:  1
******************************************************************
 * 7 >> 0 das 1 gute 2 buch 3 gefunden 4
O.k.    ** CPU time used (sec): 0.4,  Solutions:  1
******************************************************************
 * 8 >> 0 dem 1 verwandten 2 das 3 gute 4 buch 5 gegeben 6
O.k.    ** CPU time used (sec): 0.9,  Solutions:  1
******************************************************************
 * 9 >> 0 der 1 kleine 2 mann 3 lacht 4
O.k.    ** CPU time used (sec): 0.3,  Solutions:  2
******************************************************************
 * 10 >> 0 der 1 kleine 2 mann 3 das 4 gute 5 buch 6 findet 7
O.k.    ** CPU time used (sec): 0.9,  Solutions:  1
******************************************************************
 * 11 >> 0 der 1 kleine 2 mann 3 dem 4 verwandten 5 das 6 gute 7 buch 8 gibt 9
O.k.    ** CPU time used (sec): 1.4,  Solutions:  1
******************************************************************
 * 12 >> 0 dass 1 der 2 kleine 3 mann 4 lacht 5
O.k.    ** CPU time used (sec): 0.4,  Solutions:  1
******************************************************************
 * 13 >> 0 dass 1 der 2 kleine 3 mann 4 das 5 gute 6 buch 7 findet 8
O.k.    ** CPU time used (sec): 1.0,  Solutions:  1
******************************************************************
 * 14 >> 0 dass 1 der 2 kleine 3 mann 4 dem 5 verwandten 6 das 7 gute 8 buch 9 gibt 10
O.k.    ** CPU time used (sec): 1.5,  Solutions:  1
******************************************************************
 * 15 >> 0 lacht 1 der 2 kleine 3 mann 4
O.k.    ** CPU time used (sec): 0.3,  Solutions:  1
******************************************************************
 * 16 >> 0 findet 1 der 2 kleine 3 mann 4 das 5 gute 6 buch 7
O.k.    ** CPU time used (sec): 0.8,  Solutions:  1
******************************************************************
 * 17 >> 0 gibt 1 der 2 kleine 3 mann 4 dem 5 verwandten 6 das 7 gute 8 buch 9
O.k.    ** CPU time used (sec): 1.3,  Solutions:  1
******************************************************************
 * 18 >> 0 der 1 kleine 2 mann 3 lacht 4
O.k.    ** CPU time used (sec): 0.3,  Solutions:  2
******************************************************************
 * 19 >> 0 der 1 kleine 2 mann 3 findet 4 das 5 gute 6 buch 7
O.k.    ** CPU time used (sec): 1.0,  Solutions:  1
```



```
******************************************************************
 * 20 >> 0 der 1 kleine 2 mann 3 gibt 4 dem 5 verwandten 6 das 7 gute 8 buch 9
O.k.    ** CPU time used (sec): 1.4,  Solutions:  1
******************************************************************
 * 21 >> 0 das 1 gute 2 buch 3 findet 4 der 5 kleine 6 mann 7
O.k.    ** CPU time used (sec): 0.9,  Solutions:  1
******************************************************************
 * 22 >> 0 das 1 gute 2 buch 3 gibt 4 der 5 kleine 6 mann 7 dem 8 verwandten 9
O.k.    ** CPU time used (sec): 1.5,  Solutions:  1
******************************************************************
 * 23 >> 0 dem 1 verwandten 2 gibt 3 der 4 kleine 5 mann 6 das 7 gute 8 buch 9
O.k.    ** CPU time used (sec): 1.4,  Solutions:  1
******************************************************************
 * 24 >> 0 das 1 gute 2 buch 3 finden 4 wollen 5
O.k.    ** CPU time used (sec): 1.1,  Solutions:  2
******************************************************************
 * 25 >> 0 dem 1 verwandten 2 das 3 gute 4 buch 5 geben 6 wollen 7
O.k.    ** CPU time used (sec): 2.3,  Solutions:  2
******************************************************************
 * 26 >> 0 das 1 gute 2 buch 3 gefunden 4 haben 5
O.k.    ** CPU time used (sec): 0.9,  Solutions:  2
******************************************************************
 * 27 >> 0 dem 1 verwandten 2 das 3 gute 4 buch 5 gegeben 6 haben 7
O.k.    ** CPU time used (sec): 2.0,  Solutions:  2
******************************************************************
 * 28 >> 0 der 1 kleine 2 mann 3 lachen 4 wird 5
O.k.    ** CPU time used (sec): 0.5,  Solutions:  1
******************************************************************
 * 29 >> 0 der 1 kleine 2 mann 3 das 4 gute 5 buch 6 finden 7 wird 8
O.k.    ** CPU time used (sec): 1.3,  Solutions:  1
******************************************************************
 * 30 >> 0 der 1 kleine 2 mann 3 dem 4 verwandten 5 das 6 gute 7 buch 8 geben 9 wird 10
O.k.    ** CPU time used (sec): 2.2,  Solutions:  1
******************************************************************
 * 31 >> 0 dass 1 der 2 kleine 3 mann 4 lachen 5 wird 6
O.k.    ** CPU time used (sec): 0.5,  Solutions:  1
******************************************************************
 * 32 >> 0 dass 1 der 2 kleine 3 mann 4 das 5 gute 6 buch 7 finden 8 wird 9
O.k.    ** CPU time used (sec): 1.4,  Solutions:  1
******************************************************************
 * 33 >> 0 dass 1 der 2 kleine 3 mann 4 dem 5 verwandten 6 das 7 gute 8 buch 9 geben 10 wird 11
O.k.    ** CPU time used (sec): 2.4,  Solutions:  1
******************************************************************
 * 34 >> 0 wird 1 der 2 kleine 3 mann 4 lachen 5
O.k.    ** CPU time used (sec): 0.5,  Solutions:  1
******************************************************************
 * 35 >> 0 wird 1 der 2 kleine 3 mann 4 das 5 gute 6 buch 7 finden 8
O.k.    ** CPU time used (sec): 1.4,  Solutions:  1
******************************************************************
 * 36 >> 0 wird 1 der 2 kleine 3 mann 4 dem 5 verwandten 6 das 7 gute 8 buch 9 geben 10
O.k.    ** CPU time used (sec): 2.4,  Solutions:  1
******************************************************************
 * 37 >> 0 der 1 kleine 2 mann 3 wird 4 lachen 5
O.k.    ** CPU time used (sec): 0.5,  Solutions:  1
******************************************************************
 * 38 >> 0 der 1 kleine 2 mann 3 wird 4 das 5 gute 6 buch 7 finden 8
O.k.    ** CPU time used (sec): 1.4,  Solutions:  1
******************************************************************
 * 39 >> 0 der 1 kleine 2 mann 3 wird 4 dem 5 verwandten 6 das 7 gute 8 buch 9 geben 10
O.k.    ** CPU time used (sec): 2.1,  Solutions:  1
```

```
*******************************************************************
 * 40 >> 0 das 1 gute 2 buch 3 wird 4 der 5 kleine 6 mann 7 finden 8
O.k.    ** CPU time used (sec): 1.3,  Solutions:  1
*******************************************************************
 * 41 >> 0 das 1 gute 2 buch 3 wird 4 der 5 kleine 6 mann 7 dem 8 verwandten 9 geben 10
O.k.    ** CPU time used (sec): 2.3,  Solutions:  1
*******************************************************************
 * 42 >> 0 dem 1 verwandten 2 wird 3 der 4 kleine 5 mann 6 das 7 gute 8 buch 9 geben 10
O.k.    ** CPU time used (sec): 2.3,  Solutions:  1
*******************************************************************
 * 43 >> 0 das 1 gute 2 buch 3 finden 4 wollen 5 werden 6
O.k.    ** CPU time used (sec): 2.2,  Solutions:  2
*******************************************************************
 * 44 >> 0 dem 1 verwandten 2 das 3 gute 4 buch 5 geben 6 wollen 7 werden 8
O.k.    ** CPU time used (sec): 4.4,  Solutions:  2
*******************************************************************
 * 45 >> 0 das 1 gute 2 buch 3 gefunden 4 haben 5 werden 6
O.k.    ** CPU time used (sec): 2.0,  Solutions:  2
*******************************************************************
 * 46 >> 0 dem 1 verwandten 2 das 3 gute 4 buch 5 gegeben 6 haben 7 werden 8
O.k.    ** CPU time used (sec): 4.1,  Solutions:  2
*******************************************************************
 * 47 >> 0 der 1 kleine 2 mann 3 lachen 4 wollen 5 wird 6
O.k.    ** CPU time used (sec): 0.8,  Solutions:  1
*******************************************************************
 * 48 >> 0 der 1 kleine 2 mann 3 das 4 gute 5 buch 6 finden 7 wollen 8 wird 9
O.k.    ** CPU time used (sec): 2.2,  Solutions:  1
*******************************************************************
 * 49 >> 0 der 1 kleine 2 mann 3 dem 4 verwandten 5 das 6 gute 7 buch 8 geben 9 wollen 10 wird 11
O.k.    ** CPU time used (sec): 3.9,  Solutions:  1
*******************************************************************
 * 50 >> 0 dass 1 der 2 kleine 3 mann 4 lachen 5 wollen 6 wird 7
O.k.    ** CPU time used (sec): 0.9,  Solutions:  1
*******************************************************************
 * 51 >> 0 dass 1 der 2 kleine 3 mann 4 das 5 gute 6 buch 7 finden 8 wollen 9 wird 10
O.k.    ** CPU time used (sec): 2.3,  Solutions:  1
*******************************************************************
 * 52 >> 0 dass 1 der 2 kleine 3 mann 4 dem 5 verwandten 6 das 7 gute 8 buch 9 geben 10 wollen 11
        wird 12
O.k.    ** CPU time used (sec): 4.1,  Solutions:  1
*******************************************************************
 * 53 >> 0 wird 1 der 2 kleine 3 mann 4 lachen 5 wollen 6
O.k.    ** CPU time used (sec): 0.9,  Solutions:  1
*******************************************************************
 * 54 >> 0 wird 1 der 2 kleine 3 mann 4 das 5 gute 6 buch 7 finden 8 wollen 9
O.k.    ** CPU time used (sec): 2.4,  Solutions:  1
*******************************************************************
 * 55 >> 0 wird 1 der 2 kleine 3 mann 4 dem 5 verwandten 6 das 7 gute 8 buch 9 geben 10 wollen 11
O.k.    ** CPU time used (sec): 4.4,  Solutions:  1
*******************************************************************
 * 56 >> 0 der 1 kleine 2 mann 3 wird 4 lachen 5 wollen 6
O.k.    ** CPU time used (sec): 1.0,  Solutions:  2
*******************************************************************
 * 57 >> 0 der 1 kleine 2 mann 3 wird 4 das 5 gute 6 buch 7 finden 8 wollen 9
O.k.    ** CPU time used (sec): 2.6,  Solutions:  1
*******************************************************************
 * 58 >> 0 der 1 kleine 2 mann 3 wird 4 dem 5 verwandten 6 das 7 gute 8 buch 9 geben 10 wollen 11
O.k.    ** CPU time used (sec): 4.0,  Solutions:  1
*******************************************************************
 * 59 >> 0 das 1 gute 2 buch 3 wird 4 der 5 kleine 6 mann 7 finden 8 wollen 9
O.k.    ** CPU time used (sec): 2.1,  Solutions:  1
```



```
*******************************************************************
 * 60 >> 0 das 1 gute 2 buch 3 wird 4 der 5 kleine 6 mann 7 dem 8 verwandten 9 geben 10 wollen 11
O.k.    ** CPU time used (sec): 4.1,  Solutions:  1
*******************************************************************
 * 61 >> 0 dem 1 verwandten 2 wird 3 der 4 kleine 5 mann 6 das 7 gute 8 buch 9 geben 10 wollen 11
O.k.    ** CPU time used (sec): 4.1,  Solutions:  1
*******************************************************************
 * 62 >> 0 das 1 gute 2 buch 3 finden 4 koennen 5 wollen 6 werden 7
O.k.    ** CPU time used (sec): 4.4,  Solutions:  2
*******************************************************************
 * 63 >> 0 dem 1 verwandten 2 das 3 gute 4 buch 5 geben 6 koennen 7 wollen 8 werden 9
O.k.    ** CPU time used (sec): 8.0,  Solutions:  2
*******************************************************************
 * 64 >> 0 das 1 gute 2 buch 3 werden 4 haben 5 finden 6 koennen 7
O.k.    ** CPU time used (sec): 3.2,  Solutions:  2
*******************************************************************
 * 65 >> 0 dem 1 verwandten 2 das 3 gute 4 buch 5 werden 6 haben 7 geben 8 koennen 9
O.k.    ** CPU time used (sec): 6.3,  Solutions:  2
*******************************************************************
 * 66 >> 0 der 1 kleine 2 mann 3 lachen 4 koennen 5 wollen 6 wird 7
O.k.    ** CPU time used (sec): 1.6,  Solutions:  1
*******************************************************************
 * 67 >> 0 der 1 kleine 2 mann 3 das 4 gute 5 buch 6 finden 7 koennen 8 wollen 9 wird 10
O.k.    ** CPU time used (sec): 3.6,  Solutions:  1
*******************************************************************
 * 68 >> 0 der 1 kleine 2 mann 3 dem 4 verwandten 5 das 6 gute 7 buch 8 geben 9 koennen 10 wollen 11
        wird 12
O.k.    ** CPU time used (sec): 6.6,  Solutions:  1
*******************************************************************
 * 69 >> 0 dass 1 der 2 kleine 3 mann 4 lachen 5 koennen 6 wollen 7 wird 8
O.k.    ** CPU time used (sec): 1.8,  Solutions:  1
*******************************************************************
 * 70 >> 0 dass 1 der 2 kleine 3 mann 4 das 5 gute 6 buch 7 finden 8 koennen 9 wollen 10 wird 11
O.k.    ** CPU time used (sec): 3.8,  Solutions:  1
*******************************************************************
 * 71 >> 0 dass 1 der 2 kleine 3 mann 4 dem 5 verwandten 6 das 7 gute 8 buch 9 geben 10 koennen 11
        wollen 12 wird 13
O.k.    ** CPU time used (sec): 6.9,  Solutions:  1
*******************************************************************
 * 72 >> 0 wird 1 der 2 kleine 3 mann 4 lachen 5 koennen 6 wollen 7
O.k.    ** CPU time used (sec): 1.8,  Solutions:  1
*******************************************************************
 * 73 >> 0 wird 1 der 2 kleine 3 mann 4 das 5 gute 6 buch 7 finden 8 koennen 9 wollen 10
O.k.    ** CPU time used (sec): 4.1,  Solutions:  1
*******************************************************************
 * 74 >> 0 wird 1 der 2 kleine 3 mann 4 dem 5 verwandten 6 das 7 gute 8 buch 9 geben 10 koennen 11
        wollen 12
O.k.    ** CPU time used (sec): 7.4,  Solutions:  1
*******************************************************************
 * 75 >> 0 der 1 kleine 2 mann 3 wird 4 lachen 5 koennen 6 wollen 7
O.k.    ** CPU time used (sec): 2.1,  Solutions:  2
*******************************************************************
 * 76 >> 0 der 1 kleine 2 mann 3 wird 4 das 5 gute 6 buch 7 finden 8 koennen 9 wollen 10
O.k.    ** CPU time used (sec): 4.6,  Solutions:  1
*******************************************************************
 * 77 >> 0 der 1 kleine 2 mann 3 wird 4 dem 5 verwandten 6 das 7 gute 8 buch 9 geben 10 koennen 11
        wollen 12
O.k.    ** CPU time used (sec): 6.9,  Solutions:  1
*******************************************************************
 * 78 >> 0 das 1 gute 2 buch 3 wird 4 der 5 kleine 6 mann 7 finden 8 koennen 9 wollen 10
O.k.    ** CPU time used (sec): 3.6,  Solutions:  1
```

```
********************************************************************
 * 79 >> 0 das 1 gute 2 buch 3 wird 4 der 5 kleine 6 mann 7 dem 8 verwandten 9 geben 10 koennen 11
        wollen 12
O.k.    ** CPU time used (sec): 6.9,  Solutions:  1
********************************************************************
 * 80 >> 0 dem 1 verwandten 2 wird 3 der 4 kleine 5 mann 6 das 7 gute 8 buch 9 geben 10 koennen 11
        wollen 12
O.k.    ** CPU time used (sec): 6.9,  Solutions:  1
********************************************************************
 * 81 >> 0 lachen 1 wird 2 der 3 kleine 4 mann 5
O.k.    ** CPU time used (sec): 0.5,  Solutions:  1
********************************************************************
 * 82 >> 0 finden 1 wird 2 der 3 kleine 4 mann 5 das 6 gute 7 buch 8
O.k.    ** CPU time used (sec): 1.3,  Solutions:  1
********************************************************************
 * 83 >> 0 geben 1 wird 2 der 3 kleine 4 mann 5 dem 6 verwandten 7 das 8 gute 9 buch 10
O.k.    ** CPU time used (sec): 2.0,  Solutions:  1
********************************************************************
 * 84 >> 0 das 1 gute 2 buch 3 finden 4 wird 5 der 6 kleine 7 mann 8
O.k.    ** CPU time used (sec): 1.4,  Solutions:  1
********************************************************************
 * 85 >> 0 das 1 gute 2 buch 3 geben 4 wird 5 der 6 kleine 7 mann 8 dem 9 verwandten 10
O.k.    ** CPU time used (sec): 2.2,  Solutions:  1
********************************************************************
 * 86 >> 0 dem 1 verwandten 2 geben 3 wird 4 der 5 kleine 6 mann 7 das 8 gute 9 buch 10
O.k.    ** CPU time used (sec): 2.2,  Solutions:  1
********************************************************************
 * 87 >> 0 dem 1 verwandten 2 das 3 gute 4 buch 5 geben 6 wird 7 der 8 kleine 9 mann 10
O.k.    ** CPU time used (sec): 2.2,  Solutions:  1
********************************************************************
 * 88 >> 0 lachen 1 wird 2 der 3 kleine 4 mann 5 koennen 6
O.k.    ** CPU time used (sec): 0.9,  Solutions:  1
********************************************************************
 * 89 >> 0 finden 1 wird 2 der 3 kleine 4 mann 5 das 6 gute 7 buch 8 koennen 9
O.k.    ** CPU time used (sec): 2.1,  Solutions:  1
********************************************************************
 * 90 >> 0 geben 1 wird 2 der 3 kleine 4 mann 5 dem 6 verwandten 7 das 8 gute 9 buch 10 koennen 11
O.k.    ** CPU time used (sec): 3.2,  Solutions:  1
********************************************************************
 * 91 >> 0 das 1 gute 2 buch 3 finden 4 wird 5 der 6 kleine 7 mann 8 koennen 9
O.k.    ** CPU time used (sec): 1.9,  Solutions:  1
********************************************************************
 * 92 >> 0 das 1 gute 2 buch 3 geben 4 wird 5 der 6 kleine 7 mann 8 dem 9 verwandten 10 koennen 11
O.k.    ** CPU time used (sec): 3.1,  Solutions:  1
********************************************************************
 * 93 >> 0 dem 1 verwandten 2 geben 3 wird 4 der 5 kleine 6 mann 7 das 8 gute 9 buch 10 koennen 11
O.k.    ** CPU time used (sec): 3.1,  Solutions:  1
********************************************************************
 * 94 >> 0 dem 1 verwandten 2 das 3 gute 4 buch 5 geben 6 wird 7 der 8 kleine 9 mann 10 koennen 11
O.k.    ** CPU time used (sec): 2.8,  Solutions:  1
********************************************************************
 * 95 >> 0 lachen 1 koennen 2 wird 3 der 4 kleine 5 mann 6
O.k.    ** CPU time used (sec): 0.9,  Solutions:  1
********************************************************************
 * 96 >> 0 finden 1 koennen 2 wird 3 der 4 kleine 5 mann 6 das 7 gute 8 buch 9
O.k.    ** CPU time used (sec): 1.9,  Solutions:  1
********************************************************************
 * 97 >> 0 geben 1 koennen 2 wird 3 der 4 kleine 5 mann 6 dem 7 verwandten 8 das 9 gute 10 buch 11
O.k.    ** CPU time used (sec): 3.1,  Solutions:  1
********************************************************************
 * 98 >> 0 das 1 gute 2 buch 3 finden 4 koennen 5 wird 6 der 7 kleine 8 mann 9
O.k.    ** CPU time used (sec): 2.2,  Solutions:  1
```



```
********************************************************************
 * 99 >> 0 das 1 gute 2 buch 3 geben 4 koennen 5 wird 6 der 7 kleine 8 mann 9 dem 10 verwandten 11
O.k.    ** CPU time used (sec): 3.5,  Solutions:  1
********************************************************************
 * 100 >> 0 dem 1 verwandten 2 geben 3 koennen 4 wird 5 der 6 kleine 7 mann 8 das 9 gute 10 buch 11
O.k.    ** CPU time used (sec): 3.6,  Solutions:  1
********************************************************************
 * 101 >> 0 dem 1 verwandten 2 das 3 gute 4 buch 5 geben 6 koennen 7 wird 8 der 9 kleine 10 mann 11
O.k.    ** CPU time used (sec): 3.8,  Solutions:  1
********************************************************************
 * 102 >> 0 lachen 1 wird 2 der 3 kleine 4 mann 5 koennen 6 wollen 7
O.k.    ** CPU time used (sec): 1.9,  Solutions:  2
********************************************************************
 * 103 >> 0 finden 1 wird 2 der 3 kleine 4 mann 5 das 6 gute 7 buch 8 koennen 9 wollen 10
O.k.    ** CPU time used (sec): 3.9,  Solutions:  2
********************************************************************
 * 104 >> 0 geben 1 wird 2 der 3 kleine 4 mann 5 dem 6 verwandten 7 das 8 gute 9 buch 10 koennen 11
         wollen 12
O.k.    ** CPU time used (sec): 6.0,  Solutions:  2
********************************************************************
 * 105 >> 0 das 1 gute 2 buch 3 finden 4 wird 5 der 6 kleine 7 mann 8 koennen 9 wollen 10
O.k.    ** CPU time used (sec): 3.1,  Solutions:  2
********************************************************************
 * 106 >> 0 das 1 gute 2 buch 3 geben 4 wird 5 der 6 kleine 7 mann 8 dem 9 verwandten 10 koennen 11
         wollen 12
O.k.    ** CPU time used (sec): 5.1,  Solutions:  2
********************************************************************
 * 107 >> 0 dem 1 verwandten 2 geben 3 wird 4 der 5 kleine 6 mann 7 das 8 gute 9 buch 10 koennen 11
         wollen 12
O.k.    ** CPU time used (sec): 5.1,  Solutions:  2
********************************************************************
 * 108 >> 0 dem 1 verwandten 2 das 3 gute 4 buch 5 geben 6 wird 7 der 8 kleine 9 mann 10 koennen 11
         wollen 12
O.k.    ** CPU time used (sec): 4.1,  Solutions:  2
********************************************************************
 * 109 >> 0 lachen 1 koennen 2 wird 3 der 4 kleine 5 mann 6 wollen 7
O.k.    ** CPU time used (sec): 1.4,  Solutions:  1
********************************************************************
 * 110 >> 0 finden 1 koennen 2 wird 3 der 4 kleine 5 mann 6 das 7 gute 8 buch 9 wollen 10
O.k.    ** CPU time used (sec): 2.8,  Solutions:  1
********************************************************************
 * 111 >> 0 geben 1 koennen 2 wird 3 der 4 kleine 5 mann 6 dem 7 verwandten 8 das 9 gute 10 buch 11
         wollen 12
O.k.    ** CPU time used (sec): 4.4,  Solutions:  1
********************************************************************
 * 112 >> 0 das 1 gute 2 buch 3 finden 4 koennen 5 wird 6 der 7 kleine 8 mann 9 wollen 10
O.k.    ** CPU time used (sec): 2.8,  Solutions:  1
********************************************************************
 * 113 >> 0 das 1 gute 2 buch 3 geben 4 koennen 5 wird 6 der 7 kleine 8 mann 9 dem 10 verwandten 11
         wollen 12
O.k.    ** CPU time used (sec): 4.5,  Solutions:  1
********************************************************************
 * 114 >> 0 dem 1 verwandten 2 geben 3 koennen 4 wird 5 der 6 kleine 7 mann 8 das 9 gute 10 buch 11
         wollen 12
O.k.    ** CPU time used (sec): 4.6,  Solutions:  1
********************************************************************
 * 115 >> 0 dem 1 verwandten 2 das 3 gute 4 buch 5 geben 6 koennen 7 wird 8 der 9 kleine 10 mann 11
         wollen 12
O.k.    ** CPU time used (sec): 4.4,  Solutions:  1
********************************************************************
 * 116 >> 0 lachen 1 koennen 2 wollen 3 wird 4 der 5 kleine 6 mann 7
O.k.    ** CPU time used (sec): 1.7,  Solutions:  1
```

```
*******************************************************************
 * 117 >> 0 finden 1 koennen 2 wollen 3 wird 4 der 5 kleine 6 mann 7 das 8 gute 9 buch 10
O.k.    ** CPU time used (sec): 3.1,  Solutions:  1
*******************************************************************
 * 118 >> 0 geben 1 koennen 2 wollen 3 wird 4 der 5 kleine 6 mann 7 dem 8 verwandten 9 das 10 gute 11
          buch 12
O.k.    ** CPU time used (sec): 4.7,  Solutions:  1
*******************************************************************
 * 119 >> 0 das 1 gute 2 buch 3 finden 4 koennen 5 wollen 6 wird 7 der 8 kleine 9 mann 10
O.k.    ** CPU time used (sec): 3.6,  Solutions:  1
*******************************************************************
 * 120 >> 0 das 1 gute 2 buch 3 geben 4 koennen 5 wollen 6 wird 7 der 8 kleine 9 mann 10 dem 11
          verwandten 12
O.k.    ** CPU time used (sec): 5.6,  Solutions:  1
*******************************************************************
 * 121 >> 0 dem 1 verwandten 2 geben 3 koennen 4 wollen 5 wird 6 der 7 kleine 8 mann 9 das 10 gute 11
          buch 12
O.k.    ** CPU time used (sec): 5.7,  Solutions:  1
*******************************************************************
 * 122 >> 0 dem 1 verwandten 2 das 3 gute 4 buch 5 geben 6 koennen 7 wollen 8 wird 9 der 10 kleine 11
          mann 12
O.k.    ** CPU time used (sec): 6.2,  Solutions:  1
*******************************************************************
 * 123 >> 0 dass 1 der 2 kleine 3 mann 4 wird 5 lachen 6 wollen 7
O.k.    ** CPU time used (sec): 1.1,  Solutions:  1
*******************************************************************
 * 124 >> 0 dass 1 der 2 kleine 3 mann 4 das 5 gute 6 buch 7 wird 8 finden 9 wollen 10
O.k.    ** CPU time used (sec): 2.1,  Solutions:  1
*******************************************************************
 * 125 >> 0 dass 1 der 2 kleine 3 mann 4 dem 5 verwandten 6 das 7 gute 8 buch 9 wird 10 geben 11
          wollen 12
O.k.    ** CPU time used (sec): 3.3,  Solutions:  1
*******************************************************************
 * 126 >> 0 dass 1 der 2 kleine 3 mann 4 hat 5 lachen 6 wollen 7
O.k.    ** CPU time used (sec): 1.0,  Solutions:  1
*******************************************************************
 * 127 >> 0 dass 1 der 2 kleine 3 mann 4 das 5 gute 6 buch 7 hat 8 finden 9 wollen 10
O.k.    ** CPU time used (sec): 2.0,  Solutions:  1
*******************************************************************
 * 128 >> 0 dass 1 der 2 kleine 3 mann 4 dem 5 verwandten 6 das 7 gute 8 buch 9 hat 10 geben 11
          wollen 12
O.k.    ** CPU time used (sec): 3.2,  Solutions:  1
*******************************************************************
 * 129 >> 0 dass 1 der 2 kleine 3 mann 4 wird 5 lachen 6 koennen 7 wollen 8
O.k.    ** CPU time used (sec): 2.3,  Solutions:  1
*******************************************************************
 * 130 >> 0 dass 1 der 2 kleine 3 mann 4 das 5 gute 6 buch 7 wird 8 finden 9 koennen 10 wollen 11
O.k.    ** CPU time used (sec): 3.8,  Solutions:  1
*******************************************************************
 * 131 >> 0 dass 1 der 2 kleine 3 mann 4 dem 5 verwandten 6 das 7 gute 8 buch 9 wird 10 geben 11
          koennen 12 wollen 13
O.k.    ** CPU time used (sec): 5.8,  Solutions:  1
*******************************************************************
 * 132 >> 0 dass 1 der 2 kleine 3 mann 4 wird 5 wollen 6 lachen 7 koennen 8
O.k.    ** CPU time used (sec): 1.8,  Solutions:  1
*******************************************************************
 * 133 >> 0 dass 1 der 2 kleine 3 mann 4 das 5 gute 6 buch 7 wird 8 wollen 9 finden 10 koennen 11
O.k.    ** CPU time used (sec): 3.4,  Solutions:  1
*******************************************************************
 * 134 >> 0 dass 1 der 2 kleine 3 mann 4 dem 5 verwandten 6 das 7 gute 8 buch 9 wird 10 wollen 11
          geben 12 koennen 13
O.k.    ** CPU time used (sec): 5.5,  Solutions:  1
```



```
*******************************************************************
 * 135 >> 0 dass 1 der 2 kleine 3 mann 4 wird 5 haben 6 lachen 7 koennen 8
O.k.    ** CPU time used (sec): 1.8,  Solutions:  1
*******************************************************************
 * 136 >> 0 dass 1 der 2 kleine 3 mann 4 das 5 gute 6 buch 7 wird 8 haben 9 finden 10 koennen 11
O.k.    ** CPU time used (sec): 3.3,  Solutions:  1
*******************************************************************
 * 137 >> 0 dass 1 der 2 kleine 3 mann 4 dem 5 verwandten 6 das 7 gute 8 buch 9 wird 10 haben 11
          geben 12 koennen 13
O.k.    ** CPU time used (sec): 5.3,  Solutions:  1
*******************************************************************
 * 138 >> 0 dass 1 karl 2 das 3 buch 4 wird 5 haben 6 lesen 7 wollen 8
O.k.    ** CPU time used (sec): 2.4,  Solutions:  1
*******************************************************************
 * 139 >> 0 dass 1 karl 2 das 3 buch 4 wird 5 lesen 6 wollen 7 haben 8
 ### No parse!!!!  ** CPU time used (sec): 1.8,  Solutions:  0
*******************************************************************
 * 140 >> 0 dass 1 karl 2 das 3 buch 4 wird 5 lesen 6 gewollt 7 haben 8
 ### No parse!!!!  ** CPU time used (sec): 1.3,  Solutions:  0
*******************************************************************
 * 141 >> 0 dass 1 karl 2 das 3 buch 4 lesen 5 wollen 6 haben 7 wird 8
 ### No parse!!!!  ** CPU time used (sec): 1.7,  Solutions:  0
*******************************************************************
 * 142 >> 0 dass 1 karl 2 das 3 buch 4 lesen 5 gewollt 6 haben 7 wird 8
O.k.    ** CPU time used (sec): 2.0,  Solutions:  1
*******************************************************************
 * 143 >> 0 karl 1 wird 2 das 3 buch 4 haben 5 lesen 6 wollen 7
O.k.    ** CPU time used (sec): 2.6,  Solutions:  2
*******************************************************************
 * 144 >> 0 karl 1 wird 2 das 3 buch 4 lesen 5 wollen 6 haben 7
 ### No parse!!!!  ** CPU time used (sec): 2.5,  Solutions:  0
*******************************************************************
 * 145 >> 0 karl 1 wird 2 das 3 buch 4 lesen 5 gewollt 6 haben 7
O.k.    ** CPU time used (sec): 2.4,  Solutions:  2
*******************************************************************
 * 146 >> 0 dass 1 der 2 mann 3 anna 4 wird 5 finden 6 muessen 7
O.k.    ** CPU time used (sec): 1.3,  Solutions:  1
*******************************************************************
 * 147 >> 0 dass 1 der 2 mann 3 anna 4 finden 5 muessen 6 wird 7
O.k.    ** CPU time used (sec): 1.4,  Solutions:  1
*******************************************************************
 * 148 >> 0 der 1 kleine 2 verwandte 3 hat 4 gelacht 5
O.k.    ** CPU time used (sec): 1.0,  Solutions:  1
*******************************************************************
 * 149 >> 0 das 1 buch 2 hat 3 karl 4 lesen 5 wollen 6
O.k.    ** CPU time used (sec): 1.5,  Solutions:  2
*******************************************************************
 * 150 >> 0 dem 1 mann 2 hat 3 sarah 4 die 5 geschichte 6 erzaehlen 7 muessen 8
O.k.    ** CPU time used (sec): 2.9,  Solutions:  1
*******************************************************************
 * 151 >> 0 lachen 1 wird 2 anna 3
O.k.    ** CPU time used (sec): 0.2,  Solutions:  1
*******************************************************************
 * 152 >> 0 ein 1 kleines 2 buch 3 geben 4 wird 5 sarah 6 karl 7
O.k.    ** CPU time used (sec): 1.4,  Solutions:  1
*******************************************************************
 * 153 >> 0 den 1 verwandten 2 erzaehlen 3 muss 4 anna 5 die 6 geschichte 7
O.k.    ** CPU time used (sec): 1.6,  Solutions:  2
*******************************************************************
 * 154 >> 0 dem 1 kleinen 2 mann 3 das 4 gute 5 buch 6 geben 7 wird 8 karl 9
O.k.    ** CPU time used (sec): 1.8,  Solutions:  1
```

```
*********************************************************************
 * 155 >> 0 karl 1 geben 2 muessen 3 wird 4 anna 5 das 6 buch 7 haben 8 wollen 9
O.k.    ** CPU time used (sec): 7.4,  Solutions:  2
*********************************************************************
 * 156 >> 0 einigen 1 kleinen 2 maennern 3 erzaehlt 4 haben 5 muessen 6 wird 7 der 8 verwandte 9
          die 10 gute 11 geschichte 12 wollen 13
O.k.    ** CPU time used (sec): 6.4,  Solutions:  1
*********************************************************************
Total CPU time: 6.86833 min  --  Average per sentence: 2.64167 sec
```