Program System

PLAIN

"Programs for Language Analysis and INference"

EXAMPLES OF APPLICATION

by

Peter Hellwig

Computing Unit and Linguistics and International Studies Department

University of Surrey

Guildford

Surrey

(VERSION 1.3., OCTOBER 1985)

Permanent Address:

Germanistisches Seminar
Universitaet Heidelberg

Postfach 1 5 76 , D-69 Heidelberg

Table of Contents

P r e f a c e

This paper is a demonstration of the language processing system PLAIN. The listings are the same as the output which appears on the screen. The majority of the examples are drawn from the German application of the system. However, the PLAIN approach to language analysis is by no means restricted to a specific language.

The explanations given with each example had to remain brief. For a full account of the system see the bibliography at the end of the paper. A comprehensive description in English is given in (L-7). The theoretical foundation is discussed in (L-12).

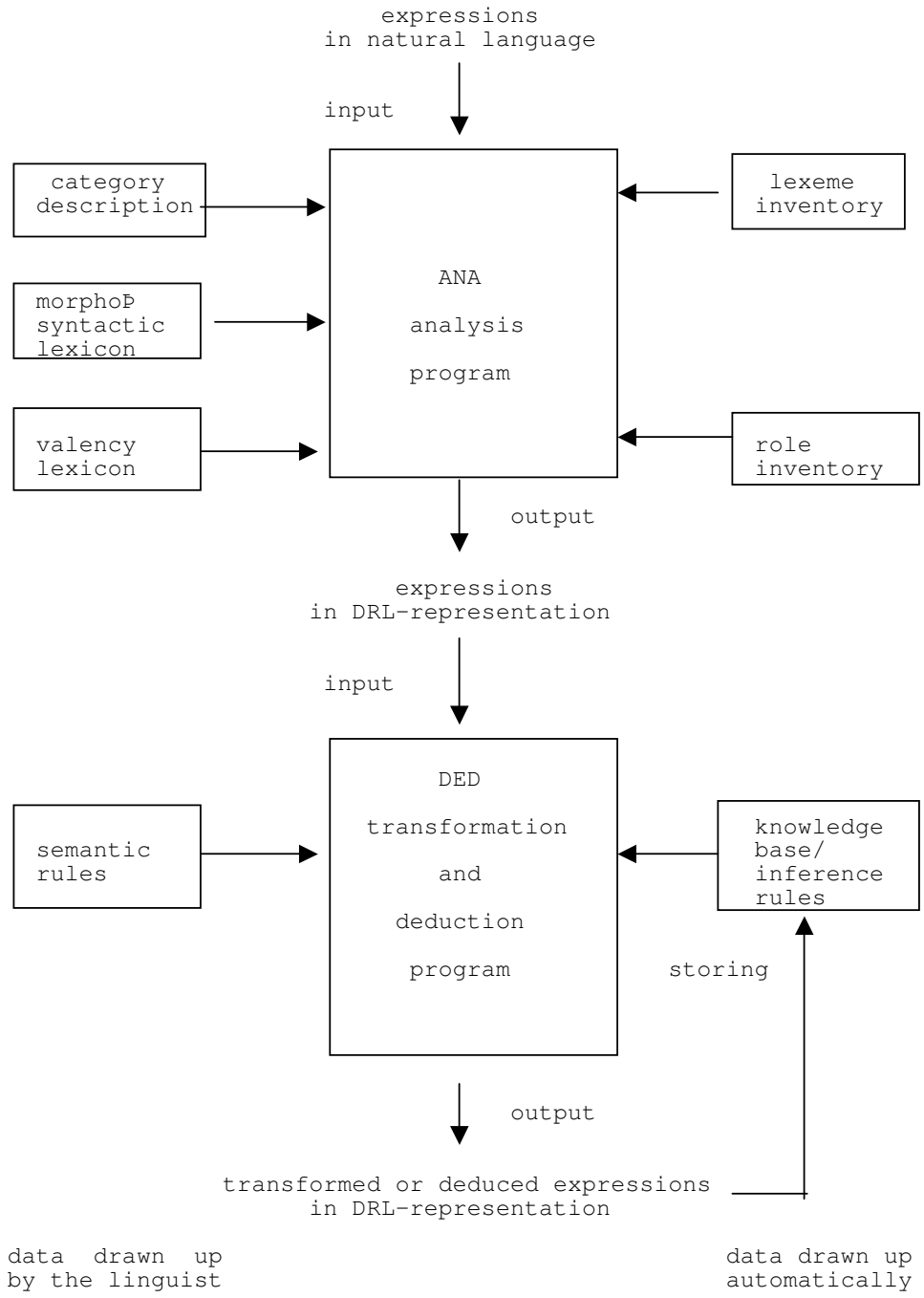
I hope, that the reader will recognize features which coincide with recent developments in formal linguistics and language processing, for instance, the notion of head of a construction, the use of complex categories together with some sort of unification, and the consideration of grammatical functions or roles. The formalism used in PLAIN covers most if not all of the phenomena which are dealt with by GPSG and LFG. The parsing algorithm of PLAIN is similar in effect to the word expert approach. The linguistic descriptions reflect the idea of frames as a means of representation. PLAIN is characterized by its use of dependency relations rather than constituent analysis and by the smooth linking of individual lexical meaning, gramatical function, and form in one unified notation.

This paper was prepared while I was staying at the University of Surrey in Guildford during September and October 1985. My visit to Surrey was supported by the University of Surrey Research Committee and the German Academic Exchange Service (DAAD). During my stay I implemented PLAIN on the IBM 3081 D computer of the University of Cambridge under Phoenix/MVS.

I am very grateful to all of my collegues at the University of Surrey who made my stay extraordinarily interesting and pleasant. My special thanks are due to Dr. Kurshid Ahmad of the Section of Knowledge Based Systems and to Ms. Margaret Rogers of the Linguistic and International Studies Department for arranging my visit and taking good care of me.

Guildford, October 1985 Peter Hellwig

(1) Survey of the main components of PLAIN



The two main components of PLAIN are the programs ANA and DED. ANA analyses natural language input and converts it into expressions of DRL (Dependency Representation Language). For this purpose the following data on the input language is needed: a definition of the morpho-syntactic categories used, a morpho-syntactic lexicon, and a valency lexicon. DED transfers DRL expressions according to given rules into new DRL expressions. It is suited to cover essential functions within automatic translation, theorem proving and deductive question answering. So-called meta-semantic rules determine the logical form of the input; these rules result in DRL expressions that are themselves rules according to which DED subsequently carries out deductions. The meta-semantic rules also include translation rules which convert DRL representations of one language into DRL representations of another language. The data sets on the left of the diagram have to be compiled by linguists for the particular input languages. Taken together they form a complete single-language or bilingual lexicon with a specification of morphology, syntax and semantics. The data sets on the right of the diagram result from the actual use of PLAIN. They are built up automatically. The lexeme and role inventory is created by any lexeme and role names being used in DRL expressions. The knowledge base is set up as follows: any statements on an object domain are entered, analysed by ANA, transformed by DED and finally stored in a database. In this process, general statements on the object domain are converted into inference rules. In addition to the components shown, PLAIN has further programs to define categories and other special symbols (DFN), to update the lexicon (LEX), to organise network-like databases (UBS) and to make various print-outs (DFN, LEX, PBS).

(2) Analysis Example

The example shows the DRL expression which is the result of the analysis of the German sentence, "Wer will PLAIN kennenlernen?" ("Who wants to get to know PLAIN?"). It is a list (in the sense of list-processing) or a "tree" representing the linguistic structure of the sentence. The parentheses as well as the indentation show the dependency relationships of the elements. Each element with all of the elements it dominates, i.e. everything that is between two complementary parentheses, forms a partial tree. The composition of a list by partial trees implicitly reflects the constituent structure of the input phrase. Every list element contains a role marker, a lexeme and a complex category, separated by a colon. In the example the role markers have capital letters. The element "(:kennen)" does not have a role marker because it belongs lexically to the dominating element "(*:lernen)". The sentence semantics is also represented by lexemes, e.g. the property of being a wh-question is indicated by the lexeme "w-frage". The third position is for morpho-syntactic categories which consist of a main category (generally a word class) and a sequence of subcategories representing grammatical features. The notation of subcategories comprises a name (such as, for example, "num" for number) and so-called values in parentheses (in the case of "num", for instance, "1" for singular and "2" for plural).

```
*** ANALY - INPUT FOR ANALYSIS:

'WER WILL PLAIN KENNENLERNEN? '
```

```
*** ANALY - RESULT OF ANALYSIS:

LIST  29:

  1  (ILLOC: w_frage': satz typ<1>
  2       (PRAED: wollen: verb con<1> fin<1> stl<2> ufl<1> wpr<1>
  3            (TEMPS: praesens')
  4            (MODUS: indikativ')
  5            (SUBJE: w_person': nomn cas<1> flg<1> num<C>
               per<3,C> rel<C> stl<2,3,C> sub<1,C> wpr<1,C>)
  6            (ADPRD: lernen: verb fin<2> flg<2> num<C> per<C>
               stl<2,C> sub<1,C> wpr<C>
  7                 (: kennen: prfx flg<1> kmp<1> stl<2,3,5,C>)
  8                 (CASAK: PLAIN: nomn cas<4> flg<1> gvb<1,C>
                    prn<0> rel<C> stl<2,3,5,C> wpr<C>))));

*** ANA - STATISTICS:

      WORDS=   5, LISTS BUILT BY LEXICON LOOKUP=     13
      AVERAGE   READINGS  PER  WORD  (HOMONYMY)=   2.00
      AVERAGE MORPHEMES  PER  WORD (COMPLEXITY)=   1.43
      SLOTS  EXAMINED=     112,  DERIVED  LISTS=     16
      DERIVED  LISTS / AV. MINIMUM (EFFICIENCY)=   2.60
      RESTARTS (DISCONTINUITY)=  0, CPU-SECONDS=   1.26
```

(3) Representation for further processing

This is the analysis result of 2 without the morpho-syntactic categories. After all, the latter describe surface features of the input languages. After the analysis has been completed, they can be omitted. The representation of the contents requires only roles and lexemes. The result contains the following information on the input sentence. The sentence has the illocution of a complementary question (wh-question). The predicate consists of the modal element "wollen" (want) with the tense "present" and mode "indicative" and the main part "kennen-lernen" (get to know). The subject of "kennen-lernen" is being asked for, the object is "PLAIN".

```
1  (ILLOC: w_frage'
2       (PRAED: wollen
3            (TEMPS: praesens')
4            (MODUS: indikativ')
5            (SUBJE: w_person')
6            (ADPRD: lernen
7                 (: kennen)
8                 (CASAK: PLAIN))));
```

(4) Definition of the morpho-syntactic categories

The morpho-syntactic categories that are to be used to describe the surface characteristics of an input language can be defined at will in PLAIN. Here, such a definition is printed for German. It is a prerequisite for the analysis in (2). The names consisting of four letters mean main categories (in general, word classes). The names made up of three letters are the appertaining subcategories representing grammatical features. Since there are all kinds of different grammatical features, different algorithms must be available to process them. (For example, there is quite a difference between checking the congruence between the subject and the predicate with respect to number and person and checking the positions of the respective elements in a sentence.) PLAIN provides various operations which can be associated with a subcategory via a number. The operations correspond to the various types of features linguists have noticed when studying languages. In the category definition, the number of the suitable operation is indicated in the first position after each feature name. (Expressed in the terminology of artificial intelligence, this means that each grammatical feature is processed by a special "expert".) The second number after each feature name indicates the number of possible values, e.g. in the case of the feature "num" (number) this is "2" (singular and plural), in the case of "per" (person) it is "3" (first, second and third person). Wherever "x" is entered, the number of the possible values is already determined by the operation. An explanation of the contents of the categories can be found in (L-10).

```
satz con 13 x flg 11 x  gvb 3 2
     num  3 2 per  1 3  rel  4 14
     stl  1 5 sub  1 8 tmp  4 6 typ  1 6 ufl 12 x wpr  2 2

verb con 13 x fin 1 4 flg 11 x
     gvb 3 2 kmp  2 1
     neg 5 1 num  3 2 per  1 3 pfk  4 2 rel  4 14
     stl  1 5 sub  1 8 tmp  4 6 typ 1 6 ufl 12 x wpr 2 2 prn 2 5

nomn adv  2 1 cas  1 4
     dte 1 3  erg  6 x fle  1 2 flg 11 x
     gen  1 3 gvb 3 2 kmp  2 1 neg  5 1 num  3 2 per  1 3
     prn  2 5 qua  2 1 rel 4 14 stl  1 5 sub  1 8
     ufl 12 x wpr  2 2

adje avw 4 2 cas 1 4 con 13 x
     fle 1 2  flg 11 x gen  1 3 kmp  2 1 lim 14 x num  3 2 per  1 3
     qua  2 1 rel 4 14 stl  1 5 sub  1 8
     ufl 12 x wpr  2 2 zhl 2 3

dete cas  1 4 con 13 x erg  6 x fle  1 2 flg 11 x
     gen  1 3 kmp  2 1 neg  5 1 num  3 2 qua  2 1 rel  4 14
     ufl 12 x wpr  2 2

prpo adv  2 1 cas  1 4 con 13 x
     fle  1 2 flg 11 x gen  1 3 neg 5 1 num  3 2 per  1 3 prn  2 5
     rel  4 14 stl  1 5 ufl 12 x  wpr  2 2 gvb 3 2

konj adv 2 1 con 13 x
     tmp  4 6 typ 1 6 ufl 12 x

prfx flg 11 x
     kmp  2 1 stl  1 5 ufl 12 x

gemo

ptkl adv 2 1 con 13 x
     flg 11 x kmp  2 1
     prn 2 5 rel  4 14 stl  1 5 sub  1 8 ufl 12 x wpr  2 2

*    adv 2 1 con 13 x flg 11 x gvb 3 2 kmp 2 1
     prn  2 5 rel  4 14 stl  1 5 ufl 12 x wpr  2 2
```

(5) Morpho-syntactic lexicon

At the beginning of the analysis, the input character string is broken down into basic segments each of which is assigned one or more DRL elements with a lexeme and a morpho-syntactic category. This happens through looking up in the morpho-syntactic lexicon. The character sequence of the input is compared with the entries in this lexicon. The lexicon is arranged in so-called sections, especially into sections for stems and for endings. References lead from one section to another which corresponds to the possible continuation of the character strings in the words. A backtracking mechanism makes sure that all of the possible decompositions, including those of compounds, are found. For example, the word 'KENNENLERNEN' is processed according to the part of the lexicon printed here as follows. First the segment 'KENNEN' is found in the section of the stems; a DRL term is formed for this segment with the lexeme "kennen". Section 128 is referred to for continuation. In Section 128 under '', i.e. without another character being added, the category is found: prefix in compound ("prfx kmp <1>"). Since there is no reference to another section, the analysis of the first word componenet 'KENNEN' ends here. Now the section of the stems is tried again and 'LERN' is found. After a new DRL element "(*:lernen)" has been created, the processing continues in section 20. This is a kind of "shunt", from which all of the endings of the weak verbs can be reached. The correct continuation is in section 101. The end of the word is reached with the ending 'EN '. The category associated with this ending, verb in the infinitive ("verb fin <2>") is added to the DRL representation for the segment 'LERNEN '. (Of course, the other ending 'EN ', for verbs in the present indicative with the features finite, plural, 1st or 3rd person ("verb fin<1> num <2>, per <1,3>") is also recognised by the analysis program. This reading, however, was not successful in the analysis (2).

```
*** PUTLX - LEXICON SECTION NO.  3:
Testvokabular aus "Bausteine des Deutschen"

'LERN'
     (*: lernen);
     SECTION  20

'KENNEN'
     (*: kennen);
     SECTION 128
```

```
*** PUTLX - LEXICON SECTION NO. 128:
Abtrennbare Praefixe des Verb
```

```
''
prfx kmp<1>

' '
prfx
```

```
*** PUTLX - LEXICON SECTION NO.  20:
Schwache Verben, Perfekt mit 'ge-', wie 'machen' -
alle Formen
sowie Infinitiv- und Partizip-Praesens-Ableitungen
```

```
''
CONTINUATION: SECTION 101
CONTINUATION: SECTION 112
CONTINUATION: SECTION 113
CONTINUATION: SECTION 116
CONDITION:    gemo
CONTINUATION: SECTION 124
CONTINUATION: SECTION  75
CONTINUATION: SECTION  83
```

```
*** PUTLX - LEXICON SECTION NO. 101:
Verben - Infinitiv,
Praesens Indikativ und Konjunktiv,
Imperativ Plural
```

```
'E '
((TEMPS: praesens')(MODUS: indikativ'));
verb fin<1> num<1> per<1> tmp<1>

'E '
((TEMPS: praesens')(MODUS: konjunktiv'));
verb fin<1> num<1> per<3> tmp<1>

'EN '
((TEMPS: praesens')(MODUS: indikativ'));
verb fin<1> num<2> per<1,3> tmp<1>

'EN '
verb fin<2>

'EST '
((TEMPS: praesens')(MODUS: konjunktiv'));
verb fin<1> num<1> per<2> tmp<1>

'ET '
((TEMPS: praesens')(MODUS: konjunktiv'));
verb fin<1> num<2> per<2> tmp<1>

'ST '
((TEMPS: praesens')(MODUS: indikativ'));
verb fin<1> num<1> per<2> tmp<1>

'T '
((TEMPS: praesens')(MODUS: indikativ'));
verb fin<1> num<1> per<3> tmp<1>
verb fin<1> num<2> per<2> tmp<1>

'T '
((MODUS: imperativ_pl'));
verb fin<4> num<2> per<2>
```

(6) Valency lexicon

In a dependency representation the syntactic relationships between the basic elements of a language are represented directly. This corresponds to the valency principle which says that the structure of complex expressions are determined by the combination capability of the basic elements. According to this approach, the syntax of a language is a matter of a lexicon in which the combination capabilities of the language elements are described. A possible way to formulate the combination capabilities is to assign slots to each lexeme for all of the elements which it is to be superordinated to in the structural tree, thereby also taking into account the morpho-syntactic categorization. In the DRL formalism, slots are represented by list elements with a variable in the place of the lexeme. The classes of elements which are allowed to fill a slot can be delimited as precisely as necessary by morpho-syntactic categorization. The slots themselves are made functionally identifiable by a role marker. If the same complement can go with different lexemes, a slot pattern is set up and references are made to this from the individual lexicon entries. The part of the German valency lexicon shown contains the references and the corresponding patterns for the vocabulary of (2). Apparently this way of describing the syntactic relationships is in agreement with the "frame" approach advocated by Artificial Intelligence. In the terminology adopted for PLAIN the set of all slots associated to a lexeme (by means of a reference to slot patterns) is also called a "valency frame".

Valency references:

```
1  (: /> <CF.SEPARATE>
2       (: wollen)
3       (& <CON>
4            (: +hilfsverb: verb pfk<1>)
5            (: +infinitiv)));

1  (: /> <CF.SEPARATE>
2       (*: lernen)
3       (& <CON>
4            (: +vollverb)
5            (: +perfekt_haben)
6            (: +praefix
7                 (: ? kennen))
8            (: +subjekt)
9            (: +akkusativ)));
```

Appertaining valency patterns :

```
1  (: +infinitiv: verb fin<1>
2       (, <AUT>
3            (ADPRD: = NICHT zu <SLOTOBL>: verb fin<2>
             flg<2> num<C> per<C> stl<2,5,C> sub<C> wpr<C>)
4            (ADPRD: = NICHT zu <SLOTOBL>: verb fin<2> num<C>
             per<C> rel<C> stl<3,C> sub<C> ufl<1> wpr<C>)));
```

```
1  (: +praefix: verb
2       (, <AUT>
3            (: _ ? <SLOT>: prfx flg<2> stl<1,4,C>)
4            (: _ ? <SLOT>: prfx flg<1> kmp<1> stl<2,3,5,C>)));

1  (: +subjekt: verb fin<1> per<3>
2       (, <AUT>
3            (SUBJE: = <SLOTOBL>: satz rel<7> sub<1,C> typ<2>)
4            (SUBJE: = <SLOTOBL>: nomn cas<1> flg<1> num<C>
             per<C> rel<C> stl<1,2,3,C> sub<1,C> wpr<C>)
5            (SUBJE: = <SLOTOBL>: nomn cas<1> flg<2> num<C>
             per<C> prn<0,1,2> stl<1,2,4,5,C> sub<1,C>)));


1  (: +akkusativ: verb
2       (, <AUT>
3            (CASAK: _ <SLOT>: satz gvb<1,C> rel<10> typ<2>)
4            (CASAK: _ <SLOT>: nomn cas<4> flg<1> gvb<1,C>
             prn<0,1,2,3,4> rel<C> stl<1,2,3,5,C> wpr<C>)
5            (CASAK: _ <SLOT>: nomn cas<4> flg<2> gvb<1,C>
             prn<0,1,2> stl<1,4,C>)
6            (CASAK: _ <SLOT>: nomn cas<4> flg<1> gvb<1,C>
             num<C> per<C> prn<5> stl<1,2,3,5,C>)
7            (CASAK: _ <SLOT>: nomn cas<4> flg<2> gvb<1,C>
             num<C> per<C> prn<5> stl<1,4,C>))));
```

(7) Documentation of the analysis in (2)

This is the record of the analysis of the German sentence in (2). First, by means of looking up in the morpho-syntactic lexicon, the segments of the input sentence have been assigned simple DRL lists with a lexeme and a category. These lists were enlarged by the slot elements from the patterns of the valency lexicon. The results are LIST 1 through LIST 13. Subsequently, increasingly complex lists were constructed by one list filling a slot in the other one. In this way, a dependency representation was created for increasingly large segments of the input, until finally a syntactic structure description for the entire input existed.

```
*** ANALY - HISTORY OF LIST  29:


LIST   1 (BUILT BY LEXICON LOOKUP):
'WER '
MORPHEM(S): 1, WORD(S): 1

  1  (*: w_person': nomn cas<1> dte<1> num<1> per<3> prn<3,4>
     rel<7> wpr<1>);
```

```
LIST   2 (BUILT BY LEXICON LOOKUP):
'WILL '
MORPHEM(S): 1, WORD(S): 01

  1  (*: wollen: verb fin<1> num<1> per<1,3> pfk<1> stl<2,3,5>
     tmp<1>
  2        (TEMPS: praesens')
  3        (MODUS: indikativ')
  4        (, <AUT>
  5             (, <AUT>
  6                  (SUBJE: = <SLOTOBL>: satz per<3,C> rel<7>
                     sub<1,C> typ<2>)
  7                  (SUBJE: = <SLOTOBL>: nomn cas<1> flg<1>
                     num<C> per<C> rel<C> stl<1,2,3,C> sub<1,C>
                     wpr<C>)
  8                  (SUBJE: = <SLOTOBL>: nomn cas<1> flg<2>
                     num<C> per<C> prn<0,1,2> stl<1,2,4,5,C>
                     sub<1,C>))
  9             (, <AUT>
 10                  (SUBJU: = es <SLOTOBL>: ptkl flg<1>
                     stl<1,2,3,C> sub<2,C>)
 11                  (SUBJU: = es <SLOTOBL>: ptkl stl<1,2,4,5,C>
                     sub<2,C> ufl<2>))
 12             (SUBJR: = <SLOTOBL>: nomn cas<1> flg<1> num<1>
                rel<C> stl<1,2,C> sub<3,C> wpr<1,C>)
 13             (SUBJE: = zu <SLOTOBL>: satz sub<4,C> typ<6>)
 14             (SUBJE: = dass <SLOTOBL>: satz sub<5,C> typ<5>)
 15             (SUBJE: = ob <SLOTOBL>: satz sub<6,C> typ<5>)
 16             (SUBJE: = <SLOTOBL>: satz sub<7,C> typ<2> wpr<2>)
 17             (SUBJE: = <SLOTOBL>: satz sub<8,C> typ<2> wpr<1>))
 18        (, <AUT>
 19             (ADPRD: = NICHT zu <SLOTOBL>: verb fin<2> flg<2>
                num<C> per<C> stl<2,5,C> sub<C> wpr<C>)
 20             (ADPRD: = NICHT zu <SLOTOBL>: verb fin<2> num<C>
                per<C> rel<C> stl<3,C> sub<C> ufl<1> wpr<C>)));

SATISFYING: mit_infinitiv';

- LOOKING FOR SLOTS WAS RESTRICTED FOR THIS LIST -
```

```
LIST   5 (BUILT BY LEXICON LOOKUP):
'PLAIN '
MORPHEM(S): 1, WORD(S): 001

  1  (*: PLAIN: nomn cas<1,3,4> gen<3> num<1> per<3>);
```

```
LIST   8 (BUILT BY LEXICON LOOKUP):
'KENNEN'
MORPHEM(S): 1, WORD(S): 0001

  1  (*: kennen: prfx kmp<1>);
```

```
LIST   9 (BUILT BY LEXICON LOOKUP):
'LERNEN'
MORPHEM(S): 01, WORD(S): 0001

  1  (*: lernen: verb fin<2> pfk<1> stl<2,3,5> sub<1>
  2        (, <AUT>
  3             (: _ ? <SLOT>: prfx flg<2> stl<1,4,C>
  4                  (: ? kennen))
  5             (: _ ? <SLOT>: prfx flg<1> kmp<1> stl<2,3,5,C>
  6                  (: ? kennen)))
  7        (AGENS: _ praep_agens' <SLOT>: prpo flg<1> rel<C>
           stl<2,3,5,C> wpr<C> gvb<2,C>)
  8        (, <AUT>
  9             (CASAK: _ <SLOT>: satz gvb<1,C> rel<10> typ<2>)
 10             (CASAK: _ <SLOT>: nomn cas<4> flg<1> gvb<1,C>
                prn<0,1,2,3,4> rel<C> stl<1,2,3,5,C> wpr<C>)
 11             (CASAK: _ <SLOT>: nomn cas<4> flg<2> gvb<1,C>
                prn<0,1,2> stl<1,4,C>)));

- NO SLOT FOUND FOR THIS LIST -
```

```
LIST  13 (BUILT BY LEXICON LOOKUP):
'? '
MORPHEM(S): 1, WORD(S): 00001

  1  (ILLOC: w_frage': satz typ<1>
  2        (PRAED: = <SLOTOBL>: verb con<1> fin<1> stl<1,2>
           ufl<1> wpr<1,2>));

- LOOKING FOR SLOTS WAS RESTRICTED FOR THIS LIST -
```

```
LIST  14 (DERIVED FROM LIST   1 AND LIST   2):
'WER WILL '
MORPHEM(S): 1, WORD(S): 11

  1  (*: wollen: verb fin<1> num<1> per<3> pfk<1> rel<7>
     stl<2,3> sub<1> tmp<1> wpr<1>
  2        (TEMPS: praesens')
  3        (MODUS: indikativ')
  4        (SUBJE: w_person': nomn cas<1> flg<1> num<C> per<3,C>
           rel<C> stl<2,3,C> sub<1,C> wpr<1,C>)
  5        (, <AUT>
  6             (ADPRD: = NICHT zu <SLOTOBL>: verb fin<2> flg<2>
                num<C> per<C> stl<2,5,C> sub<C> wpr<C>)
  7             (ADPRD: = NICHT zu <SLOTOBL>: verb fin<2> num<C>
                per<C> rel<C> stl<3,C> sub<C> ufl<1> wpr<C>)));

SATISFYING: mit_infinitiv';

- LOOKING FOR SLOTS WAS RESTRICTED FOR THIS LIST -
```

```
LIST  18 (DERIVED FROM LIST   8 AND LIST   9):
'KENNENLERNEN'
MORPHEM(S): 1, WORD(S): 0001

  1  (*: lernen: verb fin<2> pfk<1> stl<2,3,5> sub<1>
  2       (: kennen: prfx flg<1> kmp<1> stl<2,3,5,C>)
  3       (AGENS: _ praep_agens' <SLOT>: prpo flg<1> rel<C>
          stl<2,3,5,C> wpr<C> gvb<2,C>)
  4       (, <AUT>
  5            (CASAK: _ <SLOT>: satz gvb<1,C> rel<10> typ<2>)
  6            (CASAK: _ <SLOT>: nomn cas<4> flg<1> gvb<1,C>
               prn<0,1,2,3,4> rel<C> stl<1,2,3,5,C> wpr<C>)
  7            (CASAK: _ <SLOT>: nomn cas<4> flg<2> gvb<1,C>
               prn<0,1,2> stl<1,4,C>)));
```

```
LIST  23 (DERIVED FROM LIST   5 AND LIST  18):
'PLAIN KENNENLERNEN'
MORPHEM(S): 1, WORD(S): 0011

  1  (*: lernen: verb fin<2> gvb<1> pfk<1> stl<2,3,5> sub<1>
  2       (: kennen: prfx flg<1> kmp<1> stl<2,3,5,C>)
  3       (AGENS: _ praep_agens' <SLOT>: prpo flg<1> rel<C>
          stl<2,3,5,C> wpr<C> gvb<2,C>)
  4       (CASAK: PLAIN: nomn cas<4> flg<1> gvb<1,C> prn<0>
          rel<C> stl<2,3,5,C> wpr<C>));
```

```
LIST  27 (DERIVED FROM LIST  23 AND LIST  14):
'WER WILL PLAIN KENNENLERNEN'
MORPHEM(S): 1, WORD(S): 1111
  1  (*: wollen: verb fin<1> num<1> per<3> pfk<1> rel<7> stl<2>
     sub<1> tmp<1> wpr<1>
  2       (TEMPS: praesens')
  3       (MODUS: indikativ')
  4       (SUBJE: w_person': nomn cas<1> flg<1> num<C> per<3,C>
          rel<C> stl<2,3,C> sub<1,C> wpr<1,C>)
  5       (ADPRD: lernen: verb fin<2> flg<2> num<C> per<C>
          stl<2,C> sub<1,C> wpr<C>
  6            (: kennen: prfx flg<1> kmp<1> stl<2,3,5,C>)
  7            (CASAK: PLAIN: nomn cas<4> flg<1> gvb<1,C> prn<0>
               rel<C> stl<2,3,5,C> wpr<C>)));

SATISFYING: mit_infinitiv';
```

```
LIST  29 (DERIVED FROM LIST  27 AND LIST  13):
'WER WILL PLAIN KENNENLERNEN? '
MORPHEM(S): 1, WORD(S): 11111

  1  (ILLOC: w_frage': satz typ<1>
  2       (PRAED: wollen: verb con<1> fin<1> stl<2> ufl<1> wpr<1>
  3            (TEMPS: praesens')
  4            (MODUS: indikativ')
  5            (SUBJE: w_person': nomn cas<1> flg<1> num<C>
               per<3,C> rel<C> stl<2,3,C> sub<1,C> wpr<1,C>)
  6            (ADPRD: lernen: verb fin<2> flg<2> num<C> per<C>
               stl<2,C> sub<1,C> wpr<C>
  7                 (: kennen: prfx flg<1> kmp<1> stl<2,3,5,C>)
  8                 (CASAK: PLAIN: nomn cas<4> flg<1> gvb<1,C>))));
```

(8) Updating the morpho-syntactic lexicon by paradigm

In order to enlarge the morpho-syntactic lexicon, one can enter a paradigm of inflectional and derivative forms by which PLAIN recognizes the morphology of the word and then carries out an appropriate entry in the lexicon. The prerequisite for this is the presence of patterns which indicate both the expected form of the input as well as the ending sections with which the stems of the words are to be linked. In the example, the input format for the verb "LERNEN" is documented and so is the pattern that fits this paradigm. The pattern indicates that the word stem "LERN" is to be linked to section 20. Thus the result of the input by paradigm is the same as in (5).The other patterns shown are part of the total number of 230 paradigms which completely account for the German morphology. The underscore in the patterns always stands for the stem of the basic form. Included in the parentheses, the stem changes from Ablaut and Umlaut are specified. The device does not only serve to simplify drawing up the lexicon. The patterns also represent a theoretically important systematization of the morphology, since they show the relationships of the various forms.

Input:

```
LERNEN LERNST LERNTEST GELERNT DER LERNER DAS GELERNTE
```

Pattern matching with this input:

```
 _EN _ST _TEST <_TEST> GE_T |20| <DER _ER |77|> <DAS GE_TE |85|>;
```

Sample of other patterns:

```
_EN |28| _ST _(E/A)TEST |37| <_(E/A)TEST>
GE_(E/A)T |39| <DER _ER |77|> <DAS GE_(E/A)TE |85|>;
   (KENNEN KENNST KANNTEST GEKANNT DER KENNER DAS GEKKANNTE)

_EN |32| _(AU/AEU)ST |107| _(AU/IE)ST |35| <_(AU/IE)EST>
GE_EN |40| <DER _(AU/AEU)ER |77|> <DAS GE_ENE |87|>;
   (LAUFEN LAEUFST LIEFST LIEFEST GELAUFEN DER LAEUFER)

_EN |32| _(E/I)ST |34| _(E/A)ST |118| <_(E/AE)EST |123|>
_EN |126| <DER _ER |77|> <DAS _ENE |87|>;
(VERGEBEN VERGIBST VERGABST VERGAEBEST VERGEBEN DAS VERGEBENE)

_EN |32| _(E/I)ST |34| _(E/A)(FF/F)ST |118| <_(E/AE)(FF/F)EST |123|>
GE_(E/O)EN |40| <DER _ER |77|> <DAS GE_(E/O)ENE |87|>;
   (TREFFEN TRIFFST TRAFST TRAEFEST GETROFFEN DAS GETROFFENE)

_EN |32| _(EH/IEH)ST |34| _(E/A)ST |118| <_(E/AE)EST |123|>
GE_(E/O)EN |40| <DER _ER |77|> <DAS GE_(E/O)ENE |87|>;
   (STEHLEN STIEHLST STAHLST STAEHLEST GESTOHLEN DAS GESTOHLENE)
```

```
DER _(EN+) |41| _(EN+)S _(EN+) ;
   (DER LAPPEN LAPPENS LAPPEN)

DER _(EN+) |131| _(EN+)S _(+E)(EN+) |160|;
   (DER GARTEN GARTENS GAERTEN)

DER _ |44| _S _N;
   (DER SEE SEES SEEN)

DER _ |131| _S _(+E)E |166|;
   (DER ANFANG ANFANGS ANFAENGE)

DER _ |46| _S _S;
   (DER UHU UHUS UHUS)

DER _ |132| _ES _(+E)ER |168|;
   (DER WALD WALDES WAELDER)

DIE _ |140| _ _(+E) |161|;
  (DIE MUTTER MUTTER MUETTER)

DIE _ |55| _ _EN;
  (DIE FRAU FRAU FRAUEN)

DIE _ |57| _ _NEN;
(DIE BAEUERIN BAEUERIN BAEUERINNEN)

DAS _ |62| _S _;
(DAS MESSER MESSERS MESSER)

DAS _ |68| _ES _EN;
(DAS OHR OHRES OHREN)

DER _ |133| _(US+)SES _(US/EN) |160|;
(DER ZYKLUS ZYKLUSSES ZYKLEN)

DER _ |130| _(US+) _(US/I) |160|;
(DER TERMINUS TERMINUS TERMINI)

_ _CHEN |78| ;
(TISCH TISCHCHEN)

_ _(+E)LEIN |78| ;
(AST AESTLEIN)


_ |195| <_ER AM _STEN |193|> <IST _ |190|> <DAS _E |191|> ;
(ERSTAUNLICHERWEISE)
(VERMUTLICH DAS VERMUTLICHE)
(EISERN IST EISERN DAS EISERNE)
(LIEB LIEBER AM LIEBSTEN IST LIEB DAS LIEBE)

_ |195| <_(+E)ER AM _(+E)STEN |193|> <IST _ |190|> <DAS _E |191|> ;
(JUNG JUENGER AM JUENGSTEN IST JUNG DAS JUNGE)

_ |195| <_(+E)ER AM _(+E)ESTEN |194|> <IST _ |190|> <DAS _E |191|> ;
(KURZ KUERZER AM KUERZESTEN IST KURZ DAS KURZE)
```

(9) Inflection test

After drawing up a morpho-syntactic lexicon, it is possible to have an output made of all of the word forms that can be derived from a certain stem. This is a way to check the correctness of the entries. Here, the printout is to demonstrate how much information has been made available simply by the fact that the few forms of the paradigm for "LERNEN" were entered in (8).Note that some of the categories contain several alternative feature values. The numerous word forms themselves are not stored, but rather put together out of stems, derivative elements and endings. Analogously, the analysis program breaks down word forms into their components by going through the morpho-syntactic lexicon.

```
(*: lernen);
CONDITION:    -EMPTY CATEGORY-
verb kmp<1>
'LERN-'                                                  SE 101

((TEMPS: praesens')(MODUS: indikativ'));
verb fin<1> num<1> per<1,3> tmp<1>
'LERN-E '                                                SE 101

((TEMPS: praesens')(MODUS: indikativ'));
verb fin<1> num<2> per<1,3> tmp<1>
'LERN-EN '                                               SE 101

verb fin<2>
'LERN-EN '                                               SE 101

((TEMPS: praesens')(MODUS: konjunktiv'));
verb fin<1> num<1> per<2> tmp<1>
'LERN-EST '                                              SE 101

((TEMPS: praesens')(MODUS: konjunktiv'));
verb fin<1> num<2> per<2> tmp<1>
'LERN-ET '                                               SE 101

((TEMPS: praesens')(MODUS: indikativ'));
verb fin<1> num<1> per<2> tmp<1>
'LERN-ST '                                               SE 101

((TEMPS: praesens')(MODUS: indikativ'));
verb fin<1> num<1> per<3> tmp<1>
verb fin<1> num<2> per<2> tmp<1>
'LERN-T '                                                SE 101

((MODUS: imperativ_pl'));
verb fin<4> num<2> per<2>
'LERN-T '                                                SE 101

CONDITION:    -EMPTY CATEGORY-
((MODUS: imperativ_sg'));
verb fin<4> num<1> per<2>
'LERN-E '                                                SE 112
```

```
   CONDITION:    -EMPTY CATEGORY-
   ((MODUS: imperativ_sg'));
   verb fin<4> num<1> per<2>
   'LERN- '                                                SE 113

   CONDITION:    -EMPTY CATEGORY-
   ((TEMPS: praeteritum')(MODUS: indikativ'));
   verb fin<1> num<1> per<1,3> tmp<2>
   'LERN-TE '                                              SE 116

   ((TEMPS: praeteritum')(MODUS: indikativ'));
   verb fin<1> num<2> per<1,3> tmp<2>
   'LERN-TEN '                                             SE 116

   ((TEMPS: praeteritum')(MODUS: indikativ'));
   verb fin<1> num<1> per<2> tmp<2>
   'LERN-TEST '                                            SE 116

   ((TEMPS: praeteritum')(MODUS: indikativ'));
   verb fin<1> num<2> per<2> tmp<2>
   'LERN-TET '                                             SE 116

   CONDITION:    gemo
   verb fin<3>
   (GE) 'LERN-T '                                          SE 124

   CONDITION:    -EMPTY CATEGORY-
   ((DERIV: verb>subst')(FOCUS: abstrakt'));
   nomn cas<1,3,4> gen<3> num<1> per<3>
   'LERN-EN- '                                             SE 151

   nomn cas<2> gen<3> num<1> per<3>
   'LERN-EN-S '                                            SE 151

   CONDITION:    -EMPTY CATEGORY-
   ((FOCUS: aktiv'));
   adje avw<2> cas<1> fle<1> gen<1> num<1>
   adje avw<2> cas<1,4> fle<1,2> gen<2> num<1>
   adje avw<2> cas<1,4> fle<1> gen<3> num<1>
   adje avw<2> cas<1,4> fle<2> num<2>
   'LERN-END-E '                                           SE 191

   adje avw<2> cas<3> fle<2> gen<1,3> num<1>
   'LERN-END-EM '                                          SE 191

   adje avw<2> cas<2> fle<1,2> gen<1,3> num<1>
   adje avw<2> cas<3> fle<1> gen<1,3> num<1>
   adje avw<2> cas<4> fle<1,2> gen<1> num<1>
   adje avw<2> cas<2,3> fle<1> gen<2> num<1>
   adje avw<2> cas<1,2,4> fle<1> num<2>
   adje avw<2> cas<3> fle<1,2> num<2>
   'LERN-END-EN '                                          SE 191

   adje avw<2> cas<1> fle<2> gen<1> num<1>
   adje avw<2> cas<2,3> fle<2> gen<2> num<1>
   adje avw<2> cas<2> fle<2> num<2>
   'LERN-END-ER '                                          SE 191

   adje avw<2> cas<1,4> fle<2> gen<3> num<1>
   'LERN-END-ES '                                          SE 191
```

```
((FOCUS: aktiv'));
ptkl adv<1>
'LERN-END- '                                                    SE 195

(*: lernen);
((DERIV: verb>subst')(FOCUS: agent'));
nomn cas<1,3,4> gen<1> num<1> per<3>
'LERN-ER- '                                                     SE 131

nomn cas<2> gen<1> num<1> per<3>
'LERN-ER-S '                                                    SE 131

nomn cas<1,2,4> num<2> per<3>
'LERN-ER- '                                                     SE 161

nomn cas<3> num<2> per<3>
'LERN-ER-N '                                                    SE 161

((DERIV: verb>subst')(FOCUS: agent')(GENUS:
femininum'));
nomn gen<2> num<1> per<3>
'LERN-ERIN- '                                                   SE 140

nomn num<2> per<3>
'LERN-ERIN-NEN '                                                SE 164

(*: lernen);
((FOCUS: passiv'));
adje avw<1>
(GE) 'LERN-T- '                                                 SE 190

((FOCUS: passiv'));
adje avw<2> cas<1> fle<1> gen<1> num<1>
adje avw<2> cas<1,4> fle<1,2> gen<2> num<1>
adje avw<2> cas<1,4> fle<1> gen<3> num<1>
adje avw<2> cas<1,4> fle<2> num<2>
(GE) 'LERN-T-E '                                                SE 191

adje avw<2> cas<3> fle<2> gen<1,3> num<1>
(GE) 'LERN-T-EM '                                               SE 191

adje avw<2> cas<2> fle<1,2> gen<1,3> num<1>
adje avw<2> cas<3> fle<1> gen<1,3> num<1>
adje avw<2> cas<4> fle<1,2> gen<1> num<1>
adje avw<2> cas<2,3> fle<1> gen<2> num<1>
adje avw<2> cas<1,2,4> fle<1> num<2>
adje avw<2> cas<3> fle<1,2> num<2>
(GE) 'LERN-T-EN '                                               SE 191

adje avw<2> cas<1> fle<2> gen<1> num<1>
adje avw<2> cas<2,3> fle<2> gen<2> num<1>
adje avw<2> cas<2> fle<2> num<2>
(GE) 'LERN-T-ER '                                               SE 191

adje avw<2> cas<1,4> fle<2> gen<3> num<1>
(GE) 'LERN-T-ES '                                               SE 191
```

(10) Paradigms of German morphology

This is a survey of the word forms which - depending on the word class - have to be entered in order to enlarge the German morpho-syntactic lexicon by paradigm.

	Example:
Verbs:	
1) Infinitive	SPRECHEN
2) 2nd person singular indikative present	SPRICHST
3) 2nd person singular indikative past t.	SPRACHST
4) < 2nd person singular subjunctive past t. >	SPRAECHEST
5) Past participle	GESPROCHEN
6) < DER	DER
7) Nomen agentis to be derived from verb	SPRECHER
8) < DAS	DAS
9) Adjective to be derived from past participle neuter nominative singular >	GESPROCHENE
Regular nouns:	
1) Definite article, nominative singular	DER
2) Noun in the nominative singular	VOGEL
3) Noun in the genetive singular	VOGELS
4) Noun in the nominative plural	VOEGEL
Nouns with adjectival inflexion:	
1) Indefinite article, nominative singular	EIN
2) The noun as with the indefinite article	BEAMTER
3) Definite article, nominative singular	DER
4) The noun as with the definite article	BEAMTE
Singularia tantum:	
1) Definite article, nominative singulat	DAS
2) Noun in the nominative singular	VIEH
3) Noun in the genitive singular	VIEHS
Pluralia tantum:	
1) DIE	DIE
2) Noun in the nominative plural	LEUTE
3) DEN	DEN
4) Noun in the dative plural	LEUTEN
Diminuativa:	
1) The basic form of the noun	BAUM
2) The noun with the ending 'CHEN' or 'LICH'	BAEUMCHEN
Adjectives:	
1) Basic form	LANG
if forming the comparative:	
2) < Comparative	LAENGER
3) AM	AM
4) Superlative >	LAENGSTEN
if used predicatively:	
5) < IST	IST
6) Basic form >	LANG
if used attributively:	
7) < DAS	DAS
8) Inflected neuter singular >	LANGE

(11) German valency patterns

This is a survey of the valency patterns which – at the stage of (L-10) – are available to describe the syntagmatic characteristics of the German lexemes. By means of paradigm input according to (10) and a selection of valency patterns from (11), one can easily expand the vocabulary of the German version of PLAIN. The patterns can be lexically constrained by adding the appropriate lexemes marked by "?".

```
Verbs:
    +vollverb
    +perfekt_haben / +perfekt_sein
    +reflexiv_dat / +reflexiv_akk / +reflexiv_praep(: ... ?)
    +praefix, +verbzusatz
    +subjektlos, +subjekt, +subjekt_es, +subjekt_relat,
    +subjekt_infzu, +subjekt_dass, +subjekt_ob, +subjekt_wie,
    +subjekt_wfrage
    +trans, +trans_infzu, +trans_dass, +trans_ob, +trans_wie,
    +trans_wfrage
    +genitiv, +genitiv_infzu, +genitiv_dass, +genitiv_ob,
    +genitiv_wie, +genitiv_wfrage
    +dativ
    +akkusativ, +akkusativ_infzu, +akkusativ_dass, +akkusativ_ob,
    +akkusativ_wie, +akkusativ_wfrage
    +praep_casus (: ... ?)
    +indirekte_rede, +direkte_rede
    sprechaktverb' ?
    +ohne_passiv
    +adverb (: ...?)
             modal'
             situativ' = lokal', temporal'
             relational' = konditional', konzessiv', konsekutiv',
             kausal', restriktiv', final', richtung'

Nouns:
    +individuativ / +kontinuativ / +abstrakt / +name / +name_d
    +attr_adje, +attr_relsatz
    +attr_zuordnung, +attr_in_relation, +attr_eigen
    +attr_subjekt, +attr_agens
    +attr_trans, +attr_trans_infzu, +attr_trans_dass,
    +attr_trans_ob, +attr_trans_wie, +attr_trans_wfrage
    +attr_praep (: ... ?)

Adjektives:
    +reflexiv_dat / +reflexiv_akk / +reflexiv_praep(: ... ?)
    +subjektlos, +subjekt, +subjekt_es, +subjekt_relat,
    +subjekt_infzu, +subjekt_dass, +subjekt_ob, +subjekt_wie,
    +subjekt_wfrage
    +genitiv, +genitiv_infzu, +genitiv_dass, +genitiv_ob,
    +genitiv_wie, +genitiv_wfrage
    +dativ
    +akkusativ, +akkusativ_infzu, +akkusativ_dass, +akkusativ_ob,
    +akkusativ_wie, +akkusativ_wfrage
    +praep_casus (: ... ?)
```

(12) French lexicon segment in input format

This is a small French lexicon that was created ad hoc during a demonstration in order to have PLAIN analyse two French sentences. Data for PLAIN is drawn up in this format with a text editor and then loaded by means of a batch program. The numbers at the beginning of the lines as well as certain symbles, such as "te", "se", "st", control the lexicon program; "3" means entry into the morpho-syntactic lexicon, "4" means entry into the valency lexicon.

```
2 Franzoesisches Lexikon - Staemme;

3 '| ' te (: apostrophe); st
4 (: -> (: apostrophe)
        (*: = : * ufl<1> kmp<1>));

3 '? ' te (MODE: interrogatif); ca satz; st
4 (MODE: interrogatif
        (PREDICAT: = : verb ufl<1>));

3 'DONN' te (*: donner); se 51 st
4 (*: donner
        (SUJET: = : nomn flg<1> num<C> per<C>)
        (OBJET1: _ : nomn cas<3>)
        (OBJET2: _ : nomn cas<2>));

3 'EN ' te (*: indefini); ca nomn cas<2>; st

3 'M' te (*: locuteur);
  ca nomn cas<3> kmp<1> num<1> per<1>; st

3 'TU ' te (*: allocute);
  ca nomn cas<1> num<1> per<2>; st

3 '. ' te (MODE: affirmatif); ca satz; st
4 (MODE: affirmatif
        (PREDICAT: = : verb ufl<1>));

3 'PIERRE' te (*:Pierre); se 52 st

3 'CUISINE' te (*:cuisine); se 53
4 (*:cuisine (REFERENCE: = : dete gen<C> num<C>));

3 'DANS ' te (*:dans:prpo ); st
4 (*:dans (: = : nomn ufl<2>));

3 'LA ' te (*: definit: dete gen<2> num<1>); st

3 'ENTR' te (*:entrer); se 51 st
4 (*:entrer
     (SUJET: = : nomn flg<1> num<C> per<C>)
     (DIR: _  dans: prpo flg<2>));
```

(13) Analysis of French sentences

These are the two sentences that PLAIN can analyse on the basis of the lexicon in (12).

```
*** ANALY - INPUT FOR ANALYSIS:

'TU M| EN DONNES ? '
```

```
*** ANALY - RESULT OF ANALYSIS:

LIST  15:

  1  (MODE: interrogatif: satz
  2       (PREDICAT: donner: verb ufl<1>
  3            (TEMPS: praesens)
  4            (SUJET: allocute: nomn flg<1> num<C> per<2,C>)
  5            (OBJET1: locuteur: nomn cas<3>)
  6            (OBJET2: indefini: nomn cas<2>)));

*** ANA - STATISTICS:

      WORDS=   5, LISTS BUILT BY LEXICON LOOKUP=      6
      AVERAGE   READINGS  PER  WORD  (HOMONYMY)=   1.00
      AVERAGE MORPHEMES  PER  WORD (COMPLEXITY)=   1.20
      SLOTS  EXAMINED=      13,  DERIVED  LISTS=      9
      DERIVED  LISTS / AV. MINIMUM (EFFICIENCY)=   1.80
      RESTARTS (DISCONTINUITY)=  0, CPU-SECONDS=   0.23
```

```
*** ANALY - INPUT FOR ANALYSIS:

'PIERRE ENTRE DANS LA CUISINE . '
```

```
*** ANALY - RESULT OF ANALYSIS:

LIST  12:

  1  (MODE: affirmatif: satz
  2       (PREDICAT: entrer: verb ufl<1>
  3            (TEMPS: praesens)
  4            (SUJET: Pierre: nomn flg<1> num<C> per<C>)
  5            (DIR: dans: prpo flg<2>
  6                 (: cuisine: nomn ufl<2>
  7                      (REFERENCE: defini: dete gen<2,C>
                         num<C>)))));

*** ANA - STATISTICS:

      WORDS=   6, LISTS BUILT BY LEXICON LOOKUP=      6
      AVERAGE   READINGS  PER  WORD  (HOMONYMY)=   1.00
      AVERAGE MORPHEMES  PER  WORD (COMPLEXITY)=   1.00
      SLOTS  EXAMINED=      12,  DERIVED  LISTS=      6
      DERIVED  LISTS / AV. MINIMUM (EFFICIENCY)=   1.20
      RESTARTS (DISCONTINUITY)=  0, CPU-SECONDS=   0.22
```

(14) Lexical variants with different valency frames

Lexical variants arise from the fact that there are different completion possibilities for the same word. As a rule the meanings are then different as well. In the valency lexicon this is taken into account by various valency references, as shown here for the German verb "erinnern". The first variant is "jemanden an etwas erinnern". In English this means "to remind someone of something". The second variant is the reflexive "sich an etwas erinnern" and is translated as "to remember something".

```
 1  (: /> <CF.SEPARATE>
 2        (*: erinnern)
 3        (& <CON>
 4             (: +vollverb)
 5             (: +perfekt_haben)
 6             (: +subjekt)
 7             (: +trans +)
 8             (: +praep_casus
 9                  (: ? an)
10                  (: ? daran_ob)
11                  (: ? daran_dass))));
```

```
 1  (: /> <CF.SEPARATE>
 2        (*: erinnern)
 3        (& <CON>
 4             (: +vollverb)
 5             (: +perfekt_haben)
 6             (: +reflexiv_akk +)
 7             (: +subjekt)
 8             (: +praep_casus
 9                  (: ? an)
10                  (: ? daran_ob)
11                  (: ? daran_dass))));
```

(15) Disambiguation by different valency frames

Different valence frames often make it possible to clarify lexical ambiguities. The two variants of "erinnern" in the example above can be recognised by the different role structures of the analysis results. The two sentences are correctly translated in 23 into an English DRL-representation with "remind" and "remember".

*** ANALY - INPUT FOR ANALYSIS:

'DU ERINNERST MICH AN MEINE FREUNDIN. '

*** ANALY - RESULT OF ANALYSIS:

LIST 27:

```
  1  (ILLOC: aussage': satz typ<1>
  2       (PRAED: erinnern: verb con<1> fin<1> stl<1> ufl<1>
          wpr<0>
  3            (TEMPS: praesens')
  4            (MODUS: indikativ')
  5            (SUBJE: adressat_sg': nomn cas<1> flg<1> num<C>
               per<2,C> rel<C> stl<1,3,C> sub<1,C> wpr<C>)
  6            (TRANS: sprecher_sg': nomn cas<4> flg<2> gvb<1,C>
               prn<1> stl<1,C>)
  7            (CASPP: an: prpo prn<0> rel<C> wpr<C>
  8                 (: freundin: nomn cas<4,C> num<C> per<3,C>
                    prn<C> rel<C> ufl<2> wpr<C>
  9                      (REFER: definit': dete cas<C> fle<1,C>
                         flg<1> gen<C> num<1> wpr<C>)
 10                      (NUMRS: singular')
 11                      (ZUORD: sprecher_sg': dete cas<1,4,C>
                         fle<1,C> flg<1> gen<2,C> num<C> rel<C>
                         wpr<C>)))));
```

*** ANALY - INPUT FOR ANALYSIS:

'ICH ERINNERE MICH AN MEINE FREUNDIN. '

*** ANALY - RESULT OF ANALYSIS:

```
  1  (ILLOC: aussage': satz typ<1>
  2       (PRAED: erinnern: verb con<1> fin<1> stl<1> ufl<1>
          wpr<0>
  3            (TEMPS: praesens')
  4            (MODUS: indikativ')
  5            (: reflexiv': nomn cas<4> flg<2> num<C> per<1,C>
               prn<5> stl<1,C>)
  6            (SUBJE: sprecher_sg': nomn cas<1> flg<1> num<C>
               per<1,C> rel<C> stl<1,3,C> sub<1,C> wpr<C>)
  7            (CASPP: an: prpo prn<0> rel<C> wpr<C>
  8                 (: freundin: nomn cas<4,C> num<C> per<3,C>
                    prn<C> rel<C> ufl<2> wpr<C>
  9                      (REFER: definit': dete cas<C> fle<1,C>
                         flg<1> gen<C> num<1> wpr<C>)
 10                      (NUMRS: singular')
 11                      (ZUORD: sprecher_sg': dete cas<1,4,C>
                         fle<1,C> flg<1> gen<2,C> num<C> rel<C>
                         wpr<C>)))));
```

(16) Numbers as examples of compounds

German numbers are an example of extensive and complexly structured compounds. However, for PLAIN it hardly makes any difference whether the analysis involves assembling the parts of a compound in order to construct a word or assembling words in order to construct a sentence. In each case the basic segments are taken from the morpho-syntactic lexicon and the combinational capacity is described by slots. In the German valency lexicon, the lexeme "zig" (ten), "hundert" (hundred), and "tausend" (thousand), in so far as they are part of a compound (feature "kmp<1>"), each have two slots with the roles MULTP and ADDIT (i.e. the semantic relationship between the corresponding partial trees and the dominating term is that of multiplication or addition). (17) shows how the analysis works on this basis. Thus word formation is simply a part of the syntax. The transitions between compounds and other syntactic constructions are completely smooth. Only the subcategory "kmp" marks the difference. Compare the two English numbers. They consist of separate words as well as compounds. However, the dependency structure is the same as in German. Incidentally, all four numbers are equivalent.In (28), this equivalence is deduced by the program DED.

```
*** ANALY - INPUT FOR ANALYSIS:

'NEUNZEHNHUNDERTVIERUNDACHTZIG '
```

```
*** ANALY - RESULT OF ANALYSIS:

LIST  37:

  1  (*: hundert: adje kmp<1> num<2> qua<1> zhl<2>
  2       (MULTP: zehn: adje ufl<1> kmp<1> zhl<2>
  3            (ADDIT: neun: adje ufl<1> kmp<1> zhl<1>))
  4       (ADDIT: zig: adje ufl<2> fle<C> gen<C> num<C>
          zhl<2>
  5            (MULTP: acht: adje ufl<1> kmp<1> zhl<1>)
  6            (ADDIT: und: adje con<1> flg<1>
  7                 (: vier: adje ufl<1> kmp<1> zhl<2>))));
```

```
*** ANALY - INPUT FOR ANALYSIS:

'EINTAUSENDNEUNHUNDERTVIERUNDACHTZIG '
```

```
*** ANALY - RESULT OF ANALYSIS:

LIST  37:

  1  (*: tausend: adje kmp<1> num<2> qua<1> zhl<2>
  2       (MULTP: eins: adje ufl<1> kmp<1> zhl<2>)
  3       (ADDIT: hundert: adje ufl<2> fle<C> gen<C> num<C>
          zhl<2>
  4            (MULTP: neun: adje ufl<1> kmp<1> zhl<2>)
  5            (ADDIT: zig: adje ufl<2> fle<C> gen<C>
               num<C> zhl<2>
  6                 (MULTP: acht: adje ufl<1> kmp<1> zhl<1>)
  7                 (ADDIT: und: adje con<1> flg<1>
  8                      (: vier: adje ufl<1> kmp<1>
                         zhl<2>)))));
```

```
*** ANALY - INPUT FOR ANALYSIS:

'NINETEEN HUNDRED EIGHTY-FOUR '
```

```
*** ANALY - RESULT OF ANALYSIS:

LIST  12:

  1  (*: hundred: adje num<2> qua<1> nmb<4>
  2       (MULTP: ten: adje ufl<1> nmb<2>
  3            (ADDIT: nine: adje ufl<1> kmp<1> nmb<1>))
  4       (ADDIT: ten: adje ufl<2> nmb<3>
  5            (MULTP: eight)
  6            (ADDIT: four: adje ufl<2> nmb<1>)));
```

```
*** ANALY - INPUT FOR ANALYSIS:

'ONE THOUSAND NINE HUNDRED AND EIGHTY-FOUR '
```

```
*** ANALY - RESULT OF ANALYSIS:

LIST  21:

  1  (*: thousand: adje num<2> qua<1> nmb<5>
  2       (MULTP: one: adje ufl<1> nmb<1>)
  3       (ADDIT: hundred: adje ufl<2> nmb<4>
  4            (MULTP: nine: adje ufl<1> nmb<1>)
  5            (ADDIT: ten: adje ufl<2> nmb<3>
  6                 (MULTP: eight)
  7                 (ADDIT: four: adje ufl<2> nmb<1>))));
```

(17) Record of (16) and lexical constraints

This is the record of the analysis of the first number in (16).One can see that the slots have been constrained by lexemes which follow up the slot variable. For "hundert" (hundred) in LIST 3, for example, the fillers of both slots have to be a number that is "kleiner_100", i.e. it must be smaller than hundred. In the valency lexicon one can freely declare lexical constraints for slots and one can also classify lexemes according to such declarations. The classification of lexemes is transitive, e.g. by indicating ("less_than_10" is "less_than_100"), the selection feature "less_than_100" is transferred to all lexemes for which "less_than_10" was specified.

```
*** ANALY - HISTORY OF LIST  63:


LIST   1 (BUILT BY LEXICON LOOKUP):
'NEUN'
MORPHEM(S): 1, WORD(S): 1

  1  (*: neun: adje kmp<1> qua<1> zhl<1,2>);

SATISFYING: kleiner_10';  kleiner_100';
```

```
LIST   2 (BUILT BY LEXICON LOOKUP):
'ZEHN'
MORPHEM(S): 01, WORD(S): 1

  1  (*: zehn: adje kmp<1> qua<1> zhl<2>
  2        (ADDIT: _ kleiner_10' <SLOT>: adje kmp<1> ufl<1>
           zhl<1>));

SATISFYING: kleiner_100';
```

```
LIST   3 (BUILT BY LEXICON LOOKUP):
'HUNDERT'
MORPHEM(S): 001, WORD(S): 1

  1  (*: hundert: adje kmp<1> qua<1> zhl<2>
  2        (MULTP: _ kleiner_100' <SLOT>: adje kmp<1> ufl<1>
           zhl<2>)
  3        (ADDIT: _ kleiner_100' <SLOT>: adje fle<C> gen<C>
           num<C> ufl<2> zhl<2>));

SATISFYING: kleiner_1000';

- NO SLOT FOUND FOR THIS LIST -
```

```
LIST   4 (BUILT BY LEXICON LOOKUP):
'VIER'
MORPHEM(S): 0001, WORD(S): 1

  1  (*: vier: adje kmp<1> qua<1> zhl<1,2>);

SATISFYING: kleiner_10';  kleiner_100';
```

```
LIST   5 (BUILT BY LEXICON LOOKUP):
'UND'
MORPHEM(S): 00001, WORD(S): 1

  1  (ADDIT: und': adje zhl<2>
  2        (: = kleiner_10' + <SLOTOBL>: adje kmp<1> ufl<1>
           zhl<2>));

SATISFYING: und_1';

- LOOKING FOR SLOTS WAS RESTRICTED FOR THIS LIST -
```

```
LIST   7 (BUILT BY LEXICON LOOKUP):
'ACHT'
MORPHEM(S): 000001, WORD(S): 1

  1  (*: acht: adje kmp<1> qua<1> zhl<1,2>);

SATISFYING: kleiner_10';  kleiner_100';
```

```
LIST   8 (BUILT BY LEXICON LOOKUP):
'ZIG '
MORPHEM(S): 0000001, WORD(S): 1

  1  (*: zig: adje num<2> qua<1> zhl<2>
  2        (MULTP: = kleiner_10' <SLOTOBL>: adje kmp<1> ufl<1>
           zhl<1>)
  3        (ADDIT: _ und_1' <SLOT>: adje con<1> flg<1>));

SATISFYING: kleiner_100';

- LOOKING FOR SLOTS WAS RESTRICTED FOR THIS LIST -
```

```
LIST  26 (DERIVED FROM LIST   1 AND LIST   2):
'NEUNZEHN'
MORPHEM(S): 11, WORD(S): 1

  1  (*: zehn: adje kmp<1> qua<1> zhl<2>
  2        (ADDIT: neun: adje kmp<1> ufl<1> zhl<1>));

SATISFYING: kleiner_100';
```

```
LIST  29 (DERIVED FROM LIST   4 AND LIST   5):
'VIERUND'
MORPHEM(S): 00011, WORD(S): 1

  1  (ADDIT: und': adje zhl<2>
  2       (: vier: adje kmp<1> ufl<1> zhl<2>));

SATISFYING: und_1';  kleiner_10';  kleiner_100';
```

```
LIST  32 (DERIVED FROM LIST   7 AND LIST   8):
'ACHTZIG '
MORPHEM(S): 0000011, WORD(S): 1

  1  (*: zig: adje num<2> qua<1> zhl<2>
  2       (MULTP: acht: adje kmp<1> ufl<1> zhl<1>)
  3       (ADDIT: _ und_1' <SLOT>: adje con<1> flg<1>));

SATISFYING: kleiner_100';
```

```
LIST  44 (DERIVED FROM LIST  26 AND LIST   3):
'NEUNZEHNHUNDERT'
MORPHEM(S): 111, WORD(S): 1

  1  (*: hundert: adje kmp<1> qua<1> zhl<2>
  2       (MULTP: zehn: adje kmp<1> ufl<1> zhl<2>
  3            (ADDIT: neun: adje kmp<1> ufl<1> zhl<1>))
  4       (ADDIT: _ kleiner_100' <SLOT>: adje fle<C> gen<C>
          num<C> ufl<2> zhl<2>));

SATISFYING: kleiner_1000';

- NO SLOT FOUND FOR THIS LIST -
```

```
LIST  48 (DERIVED FROM LIST  29 AND LIST  32):
'VIERUNDACHTZIG '
MORPHEM(S): 0001111, WORD(S): 1

  1  (*: zig: adje num<2> qua<1> zhl<2>
  2       (MULTP: acht: adje kmp<1> ufl<1> zhl<1>)
  3       (ADDIT: und': adje con<1> flg<1>
  4            (: vier: adje kmp<1> ufl<1> zhl<2>)));

SATISFYING: kleiner_100';
```

```
LIST  63 (DERIVED FROM LIST  48 AND LIST  44):
'NEUNZEHNHUNDERTVIERUNDACHTZIG '
MORPHEM(S): 1, WORD(S): 1

  1  (*: hundert: adje kmp<1> num<2> qua<1> zhl<2>
  2       (MULTP: zehn: adje kmp<1> ufl<1> zhl<2>
  3            (ADDIT: neun: adje kmp<1> ufl<1> zhl<1>))
  4       (ADDIT: zig: adje fle<C> gen<C> num<C> ufl<2> zhl<2>
  5            (MULTP: acht: adje kmp<1> ufl<1> zhl<1>)
  6            (ADDIT: und': adje con<1> flg<1>
  7                 (: vier: adje kmp<1> ufl<1> zhl<2>))));
SATISFYING: kleiner_1000';
```

(18) Agreement

Agreement between grammatical features plays a major role in German. Combinations of several features correspond to a word form which is compatible or not to another word form in a syntactic construction. Hence, for example, "EIN GUTER MANN" and "DER GUTE MANN" are well-formed, whereas "DER GUTER MANN" and "EIN GUTE MANN" are not. The fact that PLAIN accepts the first two constructions and rejects the latter two is achieved by entering the features number ("num"), case ("cas"), and strong versus weak inflection ("fle") in the slots for both the article and the adjective and marking them with the symbol 'C'. This causes the intersection to be formed from the feature values of the dependent elements and of the dominant term. Only when this intersection is not empty is the filling of the slot permissible.

```
*** ANALY - INPUT FOR ANALYSIS:

'EIN GUTER MANN '

*** ANALY - RESULT OF ANALYSIS:

LIST  13:

  1  (*: mann: nomn cas<1> fle<2> gen<1> num<1> per<3>
  2        (REFER: ein: dete cas<1,C> fle<2,C> flg<1> gen<1,C>
           num<1,C> wpr<C>)
  3        (NUMRS: singular')
  4        (ATTRB: gut: adje avw<2> cas<1,C> con<1> fle<2,C>
           flg<1> gen<1,C> num<1,C> qua<0>));
```

```
*** ANALY - INPUT FOR ANALYSIS:

'DER GUTE MANN '

*** ANALY - RESULT OF ANALYSIS:

LIST  13:

  1  (*: mann: nomn cas<1> fle<1> gen<1> num<1> per<3>
  2        (REFER: d': dete cas<1,C> fle<1,C> flg<1> gen<1,C>
           num<1,C> wpr<C>)
  3        (NUMRS: singular')
  4        (ATTRB: gut: adje avw<2> cas<1,C> con<1> fle<1,C>
           flg<1> gen<1,C> num<1,C> qua<0>));
```

```
*** ANALY - INPUT FOR ANALYSIS:

'DER GUTER MANN '

*** ANALY - NO COHERENT ANALYSIS ACHIEVED
```

```
*** ANALY - INPUT FOR ANALYSIS:

'EIN GUTE MANN '

*** ANALY - NO COHERENT ANALYSIS ACHIEVED
```

(19) Homography

Many words in German are ambigous both lexically and with respect to their grammaticall category. In the example, the noun "LIEBE" (love) and the adjective "LIEB" (dear) as well as the verb "GENIESSEN" (enjoy) and the noun "GENOSSE" (friend) give rise to the homographs LIEBE and GENOSSEN. These result in two analysis results for the input German sentence "ICH HABE LIEBE GENOSSEN". LIST 53 is "I have enjoyed love". LIST 54 is "I have dear friends".

Input into the morpho-syntactic lexicon (by paradigm):

```
DIE LIEBE LIEBE
LIEB LIEBER AM LIEBSTEN IST LIEB DAS LIEBE
GENIESSEN GENIESST GENOSSEST GENOSSEN
DER GENOSSE GENOSSEN GENOSSEN
```

*** ANALY - INPUT FOR ANALYSIS:

```
'ICH HABE LIEBE GENOSSEN. '
```

*** ANALY - RESULT OF ANALYSIS:

LIST 53:

```
1  (ILLOC: aussage'
2       (PRAED: perfekt'
3            (MODUS: indikativ')
4            (SUBJE: sprecher_sg')
5            (ADPRD: geniessen
6                 (TRANS: liebe
7                      (REFER: kontinuativ')))));
```

LIST 54:

```
1  (ILLOC: aussage'
2       (PRAED: haben
3            (TEMPS: praesens')
4            (MODUS: indikativ')
5            (SUBJE: sprecher_sg')
6            (TRANS: genosse
7                 (REFER: indefinit')
8                 (NUMRS: plural')
9                 (ATTRB: lieb))));
```

(20) Contextual resolution of homographies

PLAIN does not give special treatment to homographs. Insofar as they can be resolved at all they will be disambiguated during the course of the analysis when some of the readings do not fit in the broader context. Whereas the ambiguity of "STAUBECKEN" ("Stau-Becken" versus "Staub-Ecken", i.e. "dam up-basin" (reservoir) versus "dusty corners") remains unresolved even when the plural article "DIE" is added, the singular article "DAS" eliminates the one reading because of the congruence requirements of gender and number which are met only by "BECKEN".

```
*** ANALY - INPUT FOR ANALYSIS:

'DIE STAUBECKEN '
```

```
*** ANALY - RESULT OF ANALYSIS:

LIST  24:

  1  (*: ecke: nomn cas<1,4> dte<1> fle<1> kmp<0> num<2> per<3>
  2        (REFER: d': dete flg<1> pos<1> cas<1,4,C> fle<1,C>
           gen<C> neg<C> num<2> qua<C> wpr<C>)
  3        (NUMRS: plural')
  4        (ZUORD: staub: nomn ufl<1> kmp<1>));

LIST  25:

  1  (*: becken: nomn cas<1,4> dte<1> fle<1> kmp<0> num<2> per<3>
  2        (REFER: d': dete flg<1> pos<1> cas<1,4,C> fle<1,C>
           gen<C> neg<C> num<2> qua<C> wpr<C>)
  3        (NUMRS: plural')
  4        (ZUORD: stau: nomn ufl<1> kmp<1>));
```

```
*** ANALY - INPUT FOR ANALYSIS:

'DAS STAUBECKEN '
```

```
*** ANALY - RESULT OF ANALYSIS:

LIST  22:

  1  (*: becken: nomn cas<1,4> dte<1> fle<1> gen<3> kmp<0> num<1>
     per<3>
  2        (REFER: d': dete flg<1> pos<1> cas<1,4,C> fle<1,C>
           gen<3,C> neg<C> num<1> wpr<C>)
  3        (NUMRS: singular')
  4        (ZUORD: stau: nomn ufl<1> kmp<1>));
```

(21) Discontinuous constituents

The two sentences analysed here contain discontinuous constituents. In the first sentence "ICH HABE DAS MAEDCHEN GESEHEN DAS DU SUCHST" (I have seen the girl you are looking for) the relative clause has to go past the participle "GESEHEN" and has to be subordinated to the TRANS-object "DAS MAEDCHEN" which in turn depends on the participle "GESEHEN". In the second sentence "HEUTE" SAGTE SIE "ARBEITE ICH" ("Today", she said, "I am going to work") the adverb "HEUTE" must be subordinated to the verb "ARBEITE", and then the clause "HEUTE ARBEITE ICH" has to be subordinated to the main clause "SAGTE SIE" as direct speech (role "ZITAT"). An insertion like "SAGTE SIE" can occur at almost any place in a reported speech. This means: a grammatical formalism for German must permit almost unlimited discontinuity. In PLAIN this is achieved by the fact that the order relationships of the segments have not been made the implicit principle of the structure trees, as is the case in Phrase Structure Grammar, but that they are treated as morpho-syntactic features just like number, case, gender, etc. Like other features, they are represented by subcategories. No difficulties arise from discontinuity and long distance dependencies any more. (In the example the following subcategories denote positional phenomena: con = continuity, ufl = adjacent sequence, flg = sequence in general, pos = absolute position, stl = type of sentence according to the positioning of the verb.)

```
*** ANALY - INPUT FOR ANALYSIS:

'ICH HABE DAS MAEDCHEN GESEHEN, DAS DU SUCHST. '
```

```
*** ANALY - RESULT OF ANALYSIS:

LIST  71:

  1  (ILLOC: aussage': satz typ<1>
  2       (PRAED: perfekt': verb con<1> fin<1> stl<2> ufl<1>
          wpr<0>
  3            (MODUS: indikativ')
  4            (SUBJE: sprecher_sg': nomn cas<1> flg<1> num<C>
               per<1,C> rel<C> stl<2,3,C> sub<1,C> wpr<C>)
  5            (ADPRD: sehen: verb fin<3> flg<2> gvb<1> num<C>
               per<C> pfk<1> rel<C> stl<2,C> sub<1,C> wpr<C>
  6                 (TRANS: maedchen: nomn cas<4> flg<1>
                    gvb<1,C> prn<0> rel<C> stl<2,3,5,C> wpr<C>
  7                      (REFER: d': dete cas<1,4,C> fle<1,C>
                         flg<1> gen<3,C> num<1> wpr<C>)
  8                      (NUMRS: singular')
  9                      (ATTRB: hypo': satz flg<2> rel<3> typ<2>
 10                           (PRAED: suchen: verb con<1> fin<1>
                              rel<C> stl<3> typ<2,C> wpr<C>
 11                                (TEMPS: praesens')
 12                                (MODUS: indikativ')
 13                                (SUBJE: adressat_sg': nomn
                                   cas<1> flg<1> num<C> per<2,C>
                                   rel<C> stl<1,3,C> sub<1,C>
                                   wpr<C>)
 14                                (TRANS: relativ_sgn': nomn
                                   cas<4> flg<1> gvb<1,C> prn<4>
                                   rel<C> stl<1,3,C>
                                   wpr<C>)))))));
```

*** ANALY - INPUT FOR ANALYSIS:

'"HEUTE", SAGTE SIE, "ARBEITE ICH". '

*** ANALY - RESULT OF ANALYSIS:

LIST 137:

```
  1  (ILLOC: aussage': satz typ<1>
  2       (PRAED: sagen: verb con<1> fin<1> stl<1> ufl<1> wpr<0>
  3            (TEMPS: praeteritum')
  4            (MODUS: indikativ')
  5            (SUBJE: anaphor_sgf': nomn cas<1> flg<2> num<C>
               per<3,C> prn<1> stl<1,4,C> sub<1,C>)
  6            (ZITAT: arbeiten: verb fin<1> num<1> per<1>
               stl<1,4> sub<1> tmp<1>
  7                 (TEMPS: praesens')
  8                 (MODUS: indikativ')
  9                 (SUBJE: sprecher_sg': nomn cas<1> flg<2>
                    num<C> per<1,C> prn<1> stl<1,4,C> sub<1,C>)
 10                 (ADVRB: heute: ptkl adv<1> stl<1>))));
```

(22) Punctuation and initial slots

A few intermediate steps of the analysis of the second sentence in (21) are recorded here in order to show how the punctuation can be processed. The first quotation mark of "HEUTE" has been assigned the artificial lexeme "zitat" (quotation) in the morpho-syntactic lexicon, the second quotation mark has received the lexeme "zitat-ende" (end of quotation). These lexemes are defined in the valency lexicon as equivalent to slots (formally by a direct reference to a slot). They disappear from the representation when the slots are filled by adjacent fragments of the sentence. But they are kept as selectional features associated with the segments which have filled the slots. The selectional constraint "zitat" in LIST 3 has the effect that this slot (which stands for the second quotation mark) will be suitable only for a segment that includes the first quotation mark and thus has inherited the corresponding feature. Together with the subcategories continuity ("con<1>") and adjacency ("ufl<1>"), specified in the slot, this has the effect that the quotation will be properly closed by the second quotation mark. This occurred in LIST 32. The following comma is also defined as being equivalent to a slot (in LIST 7). Since the selectional feature "zitat" is required again, which can originate only from the first quotation mark, the comma will "eat-up" the entire first part of the quotation. The second part of the quotation "ARBEITE ICH" and the second comma are processed in a similar way, so that the analysis of the entire quotation is available as a discontinous constituent in LIST 86. Finally, this list is united with the list corresponding to the segment "SAGTE SIE" in list 119.

```
*** ANALY - HISTORY OF LIST 137:


LIST   1 (BUILT BY LEXICON LOOKUP):
'"'
MORPHEM(S): 1, WORD(S): 1

  1  (*: = <SLOTOBL>: * con<1> ufl<2>);

SATISFYING: zitat';

- LOOKING FOR SLOTS WAS SKIPPED FOR THIS LIST -
```

```
LIST   2 (BUILT BY LEXICON LOOKUP):
'HEUTE'
MORPHEM(S): 1, WORD(S): 01

  1  (*: heute: ptkl adv<1>);

SATISFYING: temporal';  situativ';
```

```
LIST   3 (BUILT BY LEXICON LOOKUP):
'"'
MORPHEM(S): 1, WORD(S): 001

  1  (*: = zitat' <SLOTOBL>: * con<1> ufl<1>);

SATISFYING: zitat_ende';

- LOOKING FOR SLOTS WAS SKIPPED FOR THIS LIST -
```

```
LIST   7 (BUILT BY LEXICON LOOKUP):
', '
MORPHEM(S): 1, WORD(S): 0001

  1  (*: = zitat' <SLOTOBL>: * con<1> ufl<1>);

SATISFYING: komma';

- LOOKING FOR SLOTS WAS SKIPPED FOR THIS LIST -
```

```
LIST  25 (DERIVED FROM LIST   2 AND LIST   1):
'"HEUTE'
MORPHEM(S): 1, WORD(S): 11

  1  (*: heute: ptkl adv<1>);

SATISFYING: zitat';  temporal';  situativ';
```

```
LIST  32 (DERIVED FROM LIST  25 AND LIST   3):
'"HEUTE"'
MORPHEM(S): 1, WORD(S): 111

  1  (*: heute: ptkl adv<1>);

SATISFYING: zitat_ende';  zitat';  temporal';  situativ';
```

```
LIST  37 (DERIVED FROM LIST  32 AND LIST   7):
'"HEUTE", '
MORPHEM(S): 1, WORD(S): 1111

  1  (*: heute: ptkl adv<1>);

SATISFYING: komma';  zitat_ende';  zitat';  temporal';
     situativ';
```

```
LIST  86 (DERIVED FROM LIST  37 AND LIST  44):
'"HEUTE", , "ARBEITE ICH"'
MORPHEM(S): 1, WORD(S): 11110011111

  1  (*: arbeiten: verb fin<1> num<1> per<1> stl<1,4> sub<1>
     tmp<1>
  2       (TEMPS: praesens')
  3       (MODUS: indikativ')
  4       (SUBJE: sprecher_sg': nomn cas<1> flg<2> num<C>
          per<1,C> prn<1> stl<1,4,C> sub<1,C>)
  5       (ADVRB: _ ? <SLOT>: satz typ<5>
  6            (: ? modal')
  7            (: ? situativ')
  8            (: ? relational'))
  9       (ADVRB: heute: ptkl adv<1> stl<1>)
 10       (ADVRB: __ ? <SLOTSEQ>: * adv<1> flg<1> stl<1,2,3,C>
          wpr<C>
 11            (: ? modal')
 12            (: ? situativ')
 13            (: ? relational'))
 14       (ADVRB: __ ? <SLOTSEQ>: * adv<1> flg<2> stl<1,2,4,5,C>
 15            (: ? modal')
 16            (: ? situativ')
 17            (: ? relational')));

SATISFYING: komma';  zitat_ende';  zitat';
```

```
LIST 119 (DERIVED FROM LIST  86 AND LIST  26):
'"HEUTE", SAGTE SIE, "ARBEITE ICH"'
MORPHEM(S): 1, WORD(S): 11111111111

  1  (*: sagen: verb fin<1> num<1> per<3> stl<1,4> sub<1> tmp<2>
  2       (TEMPS: praeteritum')
  3       (MODUS: indikativ')
  4       (SUBJE: anaphor_sgf': nomn cas<1> flg<2> num<C>
          per<3,C> prn<1> stl<1,4,C> sub<1,C>)
  5       (ZITAT: arbeiten: verb fin<1> num<1> per<1> stl<1,4>
          sub<1> tmp<1>
  6            (TEMPS: praesens')
  7            (MODUS: indikativ')
  8            (SUBJE: sprecher_sg': nomn cas<1> flg<2> num<C>
               per<1,C> prn<1> stl<1,4,C> sub<1,C>)
  9            (ADVRB: heute: ptkl adv<1> stl<1>)));

SATISFYING: sprechaktverb';
```

(23) Translation German - English

Starting with this example, the system component DED is to be illustrated. The DRL representation of the two sentences in 15, with the ambiguous verb "erinnern" is transformed here into a DRL representation for English sentences. (Within the framework of an automatic translation system, a generation component would have to make English sentences out of these lists. Such a component has not yet been implemented).

```
*** DED - LIST_1:
(Du erinnerst mich an meine Freundin.)

  1  (ILLOC: aussage'
  2       (PRAED: erinnern
  3            (TEMPS: praesens')
  4            (MODUS: indikativ')
  5            (SUBJE: adressat_sg')
  6            (TRANS: sprecher_sg')
  7            (CASPP: an
  8                 (: freundin
  9                      (REFER: definit')
 10                      (NUMRS: singular')
 11                      (ZUORD: sprecher_sg')))));

*** REPLC - LIST AFTER REPLACEMENT
(You remind me of my girl friend.)

  1  (ILLOC: assertion'
  2       (PRAED: remind
  3            (TEMPS: praesens')
  4            (MODUS: indikativ')
  5            (SUBJE: you)
  6            (TRANS: me)
  7            (CASPP: of
  8                 (: girl friend
  9                      (REFER: definit')
 10                      (NUMRS: singular')
 11                      (ASSOC: my)))));

*** DED - LIST_1:
(Ich erinnere mich an meine Freundin.)

  1  (ILLOC: aussage'
  2       (PRAED: erinnern
  3            (TEMPS: praesens')
  4            (MODUS: indikativ')
  5            (: reflexiv')
  6            (SUBJE: sprecher_sg')
  7            (CASPP: an
  8                 (: freundin
  9                      (REFER: definit')
 10                      (NUMRS: singular')
 11                      (ZUORD: sprecher_sg')))));
```

```
*** REPLC - LIST AFTER REPLACEMENT
(I remember my girl friend.)

 1  (ILLOC: assertion'
 2       (PRAED: remember
 3            (TEMPS: praesens')
 4            (MODUS: indikativ')
 5            (SUBJE: I)
 6            (TRANS: girl friend
 7                 (REFER: definit')
 8                 (NUMRS: singular')
 9                 (ASSOC: my))));
```

(24) The translation rules used in (23)

These are the rules used for the transformation in (23). Each rule is made up of a rule symbol (in the first list element). Rule symbols have to be defined (by means of the program DFN) as marking a rule of a certain type. In the case here, we are dealing with replacement rules. Every rule is made up of two partial trees, subordinated to the rule symbol. The first tree is a pattern for an initial structure, the second one is a pattern for a target structure. Roles, lexemes, and entire partial trees can be substituted in rules by variables. When the rules are applied, variables in the pattern for the target structure are replaced by precisely the elements or trees that corresponded to the same variables in the initial structure. Variables allow for sufficient generalization, roles and lexemes for sufficient restriction of the rule applications.

```
1  (: D-E==> <REPLACE>
2       (SUBJE: sprecher_sg')
3       (SUBJE: I));
```

```
1  (: D-E==> <REPLACE>
2       (ZUORD: sprecher_sg')
3       (ASSOC: my));
```

```
1  (: D-E==> <REPLACE>
2       (*: freundin
3            (-))
4       (*: girl friend
5            (-)));
```

```
1  (: D-E==> <REPLACE>
2       (TRANS: sprecher_sg')
3       (TRANS: me));
```

```
 1  (: D-E==> <REPLACE>
 2       (SUBJE: adressat_sg')
 3       (SUBJE: you));
```

```
 1  (: D-E==> <REPLACE>
 2       (*: erinnern
 3            (: reflexiv')
 4            (CASPP: an
 5                 (: $1))
 6            (-1))
 7       (*: remember
 8            (-1)
 9            (TRANS: $1)));
```

```
 1  (: D-E==> <REPLACE>
 2       (*: erinnern
 3            (TRANS: $1)
 4            (CASPP: an
 5                 (-1))
 6            (-2))
 7       (*: remind
 8            (-2)
 9            (TRANS: $1)
10            (CASPP: of
11                 (-1))));
```

```
 1  (: D-E==> <REPLACE>
 2       (*: aussage'
 3            (-))
 4       (*: assertion'
 5            (-)));
```

(25) Documentation of part of the transformation of (23)

In DRL one can easily write rules according to which every DRL expression can be freely transformed into almost any other one. The structure of the target expression can be completely different from that of the initial expression. The following printout shows the phase of the translation in (23)in which the German reflexive verb "erinnern" with a prepositional object is transformed into the English verb "remember" with a direct object. The correspondence of variables and elements is kept in a register (see "LIST OF VARIABLES").

```
*** REPLC - LIST AFTER REPLACEMENT:

   1  (ILLOC: aussage'
   2       (PRAED: erinnern
   3            (TEMPS: praesens')
   4            (MODUS: indikativ')
   5            (: reflexiv')
   6            (SUBJE: I)
   7            (CASPP: an
   8                 (: girl friend
   9                      (REFER: definit')
  10                      (NUMRS: singular')
  11                      (ASSOC: my)))));

*** REPLC - RULE USED FOR REPLACEMENT:

   1  (: D-E==> <REPLACE>
   2       (*: erinnern
   3            (: reflexiv')
   4            (CASPP: an
   5                 (: $1))
   6            (-1))
   7       (*: remember
   8            (-1)
   9            (TRANS: $1)));

*** PVLST - LIST OF VARIABLES:

       TYPE: 11  NAME: *    REFERS TO:  2
       TYPE: 12  NAME: -1   REFERS TO:  3
       TYPE: 12  NAME: -1   REFERS TO:  4
       TYPE: 12  NAME: -1   REFERS TO:  6
       TYPE: 22  NAME: $1   REFERS TO:  8

*** REPLC - LIST (  2) REPLACED ACCORDING TO RULE (  7)
```

```
*** REPLC - LIST AFTER REPLACEMENT:

   1  (ILLOC: aussage'
   2       (PRAED: remember
   3            (TEMPS: praesens')
   4            (MODUS: indikativ')
   5            (SUBJE: I)
   6            (TRANS: girl friend
   7                 (REFER: definit')
   8                 (NUMRS: singular')
   9                 (ASSOC: my))));
```

(26) Analysis and translation of a Chinese sentence

During the visit of a Chinese guest, a Chinese sentence ("WO ZE U KE HUNG PINGO.") was analysed ad hoc and translated into the DRL representation for a German sentence ("Ich esse einen roten Apfel." = "I am eating a red apple."). The rules were formulated for the direction German - Chinese and then applied the other way around. (This is possible with replacement rules as long as they do not comprise any deletion). Note the rule according to which the dominating substantive of the nominal group in German is translated into a subordinated element in Chinese ("an apple" in Chinese corresponds roughly to "one thing apple").

```
*** ANALY - INPUT FOR ANALYSIS:

'WO ZE U KE HUNG PINGO.'
```

```
*** ANALY - RESULT OF ANALYSIS:

LIST  18:

  1  (ILLOC: ZHENG_XUQEU
  2       (PRAED: ZE : verb ufl<1>
  3            (SUBJE: WO : nomn ufl<1>)
  4            (TRANS: KE : nomn ufl<2>
  5                 (QUANT: U : dete ufl<1>)
  6                 (APPOS: PINGO : nomn ufl<2>
  7                      (ATTRB: HUNG : adje ufl<1>)))));
```

The appertaining grammatical description:

```
  1  (*: WO : nomn);

  1  (*: ZE : verb
  2       (SUBJE: _ <SLOT>: nomn ufl<1>)
  3       (TRANS: _ <SLOT>: nomn ufl<2>));

  1  (*: U : dete);

  1  (*: KE : nomn
  2       (QUANT: _ <SLOT>: dete ufl<1>)
  3       (APPOS: _ <SLOT>: nomn ufl<2>));

  1  (*: HUNG : adje);

  1  (*: PINGO : nomn
  2       (ATTRB: _ <SLOT>: adje ufl<1>));

  1  (ILLOC: ZHENG_XUQEU
  2       (PRAED: _ <SLOT>: verb ufl<1>));
```

Translation:

```
 1  (ILLOC: ZHENG_XUQEU
 2       (PRAED: ZE
 3            (SUBJE: WO )
 4            (TRANS: KE
 5                 (QUANT: U )
 6                 (APPOS: PINGO
 7                      (ATTRB: HUNG )))));
```

*** REPLC - LIST AFTER REPLACEMENT

```
 1  (ILLOC: aussage'
 2       (PRAED: essen
 3            (TEMPS: praesens')
 4            (MODUS: indikativ')
 5            (SUBJE: sprecher_sg')
 6            (TRANS: apfel
 7                 (REFER: ein)
 8                 (NUMRS: singular')
 9                 (ATTRB: rot))));
```

The rules made up for this translation:

```
 1  (: D-CH==> <REPLACE>
 2       (*: aussage'
 3            (-))
 4       (*: ZHENG_XUQEU
 5            (-)));
```

```
 1  (: D-CH==> <REPLACE>
 2       (*: essen
 3            (TEMPS: praesens')
 4            (MODUS: indikativ')
 5            (SUBJE: $1)
 6            (TRANS: $2)
 7            (-))
 8       (*: ZE
 9            (SUBJE: $1)
10            (TRANS: $2)
11            (-)));
```

```
 1  (: D-CH==> <REPLACE>
 2       (*: sprecher_sg')
 3       (*: WO ));
```

```
 1  (: D-CH==> <REPLACE>
 2       (*: apfel
 3            (-))
 4       (*: PINGO
 5            (-)));
```

```
 1  (: D-CH==> <REPLACE>
 2       (*: #
 3            (REFER: ein)
 4            (NUMRS: singular')
 5            (-))
 6       (*: KE
 7            (QUANT: U )
 8            (APPOS: #
 9                 (-))));
```

(27) Translation and equivalence of numbers

Here the English numbers from 16 are first translated into a German representation and this is then converted into a decimal representation. Comparison of the results shows that the numbers are equivalent.

```
*** DED - LIST_1:

  1  (*: hundred
  2        (MULTP: ten
  3             (ADDIT: nine))
  4        (ADDIT: ten
  5             (MULTP: eight)
  6             (ADDIT: four)));

*** REPLC - LIST AFTER REPLACEMENT

  1  (*: hundert
  2        (MULTP: zehn
  3             (ADDIT: neun))
  4        (ADDIT: zig
  5             (MULTP: acht)
  6             (ADDIT: vier)));

*** DED - LIST_2:

  1  (*: thousand
  2        (MULTP: one
  3        (ADDIT: hundred
  4             (MULTP: nine)
  5             (ADDIT: ten
  6                  (MULTP: eight)
  7                  (ADDIT: four))));

*** REPLC - LIST AFTER REPLACEMENT

  1  (*: tausend
  2        (MULTP: eins)
  3        (ADDIT: hundert
  4             (MULTP: neun)
  5             (ADDIT: zig
  6                  (MULTP: acht)
  7                  (ADDIT: vier))));
```

```
*** DED - LIST_1:

  1  (*: hundert
  2       (MULTP: zehn
  3            (ADDIT: neun))
  4       (ADDIT: zig
  5            (MULTP: acht)
  6            (ADDIT: und
  7                 (: vier))));

*** REPLC - LIST AFTER REPLACEMENT

  1  (*: 1
  2       (: 9
  3            (: 8
  4                 (: 4))));

*** DED - LIST_2:

  1  (*: tausend
  2       (MULTP: eins)
  3       (ADDIT: hundert
  4            (MULTP: neun)
  5            (ADDIT: zig
  6                 (MULTP: acht)
  7                 (ADDIT: und
  8                      (: vier)))));

*** REPLC - LIST AFTER REPLACEMENT

  1  (*: 1
  2       (: 9
  3            (: 8
  4                 (: 4))));
```

*** DED - LIST_1 (1) IS EQUAL TO LIST_2 (1)

(28) Conversion rules for numbers

The rules for converting German numerals into decimal representation which were used in (27) are a further illustration of the capabilities of the DRL formalism. The task was to first generate the positions for the ten, hundred and thousand powers and then - depending on the roles MULTP and ADDIT - to put the right values in the right places.

```
(:=>(*:eins)
    (*:1));
(:=>(*:zwei)
    (*:2));
(:=>(*:drei)
    (*:3));
(:=>(*:vier)
    (*:4));
(:=>(*:fuenf)
    (*:5));
(:=>(*:sechs)
    (*:6));
(:=>(*:sieben)
    (*:7));
(:=>(*:acht)
    (*:8));
(:=>(*:neun)
    (*:9));
(:=>(ADDIT:und(:$1))
    (ADDIT:$1));

(:=>(*:zehn(-1))
    (*:1(:0)(-1)));
(:=>(*:zig(-1))
    (*:1(:0)(-1)));
(:=>(*:hundert(-1))
    (*:1(:0(:0))(-1)));
(:=>(*:tausend(-1))
    (*:1(:0(:0(:0)))(-1)));

(:=>(*:1(:$1)(MULTP:#1))
    (*:#1(:$1)));
(:=>(*:1(:$1)(MULTP:#1(:#2)))
    (*:#1(:#2(:$1)));
(:=>(*:1(:$1)(MULTP:#1(:#2(:#3))))
    (*:#1(:#2(:#3(:$1))));
(:=>(*:1(:0)(ADDIT:#1)(-1))
    (*:1(:#1)(-1)));
(:=>(*:1(:0(:0))(ADDIT:#1)(-1))
    (*:1(:0(:#1))(-1)));
(:=>(*:1(:0(:0))(ADDIT:#1(:#2))(-1))
    (*:1(:#1(:#2))(-1)));
(:=>(*:1(:0(:0(:0)))(ADDIT:#1)(-1))
    (*:1(:0(:0(:#1)))(-1)));
(:=>(*:1(:0(:0(:0)))(ADDIT:#1(:#2))(-1))
    (*:1(:0(:#1(:#2)))(-1)));
(:=>(*:1(:0(:0(:0)))(ADDIT:#1(:#2(:#3)))(-1))
    (*:1(:#1(:#2(:#3)))(-1)));
```

(29) Transformation into another grammatical formalism

A Dutch example sentence from the "Linguistic Specifications" for the European translation project EUROTRA (Version May 1984, page 119) was analysed with PLAIN and brought approximately into the form that is intended for EUROTRA. This is a phrase structure grammar in which one constituent in every construction is labelled "governor" (GOV). In contrast to the PLAIN analysis, the structure trees have nodes for non-terminal units. As it is shown, these can be generated by means of the component DED. It is to be supposed that any phrase structure representation can be generated from the dependency trees of ANA.

```
*** ANALY - INPUT FOR ANALYSIS:

'MICHAEL VERLOOR ZIJN HAAR TIJDENS EEN
 LEKKEREFRISDRANKTELEVISIEREKLAMESPOTBRAND '
```

```
*** ANALY - RESULT OF ANALYSIS:

LIST  37:

  1  (PRAED: verliezen: verb
  2       (SUBJ: Michael: nomn ufl<1>)
  3       (OBJ: haar: nomn ufl<2>
  4            (ATTRN: zijn: dete flg<1>))
  5       (ADV: tijdens: prpo flg<2>
  6            (PCOMPL: brand: nomn ufl<2>
  7                 (DETE: een: dete flg<1>)
  8                 (ATTRN: spot: nomn kmp<1> ufl<1>
  9                      (ATTRN: frisdrank: nomn flg<1> kmp<1>
 10                           (ATTRA: lekker: adje flg<1>))
 11                      (ATTRN: reklame: nomn kmp<1> ufl<1>
 12                           (ATTRN: televisie: nomn kmp<1>
                              ufl<1>))))));
```

```
*** DED - LIST_1:

  1  (PRAED: verliezen
  2       (SUBJ: Michael)
  3       (OBJ: haar
  4            (ATTRN: zijn))
  5       (ADV: tijdens
  6            (PCOMPL: brand
  7                 (DETE: een)
  8                 (ATTRN: spot
  9                      (ATTRN: frisdrank
 10                           (ATTRA: lekker))
 11                      (ATTRN: reklame
 12                           (ATTRN: televisie))))));
```

```
*** REPLC - LIST AFTER REPLACEMENT

 1  (PRAED'
 2       (PHVB'
 3            (V,GOV: verliezen)
 4            (SUBJ'
 5                 (GN,COMPL'
 6                      (N,GOV: Michael)))
 7            (OBJ'
 8                 (GN,COMPL'
 9                      (N,GOV: haar)
10                      (GN.MOD
11                           (N,GOV: zijn))))
12            (ADV'
13                 (PP,MOD'
14                      (Prep,GOV: tijdens)
15                      (GN,COMPL'
16                           (N,GOV: brand)
17                           (DET.MOD: een)
18                           (GN.MOD
19                                (N,GOV: spot)
20                                (GN.MOD
21                                     (N,GOV: frisdrank)
22                                     (GADJ.MOD
23                                          (Adj,GOV: lekker)))
24                                (GN.MOD
25                                     (N,GOV: reklame)
26                                     (GN.MOD
27                                         (N,GOV: televisie)));
```

The replacement rules for this transformation:

```
(:P-ELS==>(PRAED:$)          (PRAED'(PHVB:$));
(:P-ELS==>(SUBJ:$)           (SUBJ'(GN,COMPL:$));
(:P-ELS==>(OBJ:$)            (OBJ'(GN,COMPL:$));
(:P-ELS==>(ADV:$)            (ADV'(PP,MOD:$));
(:P-ELS==>(PCOMPL:$)         (GN,COMPL:$);
(:P-ELS==>(PHVB:#(-))        (PHVB'(V,GOV:#)(-));
(:P-ELS==>(GN,COMPL:#(-))    (GN,COMPL'(N,GOV:#)(-));
(:P-ELS==>(PP,MOD:#(-))      (PP,MOD'(Prep,GOV:#)(-));
(:P-ELS==>(ATTRN:#(-))       (GN.MOD(N,GOV:#)(-));
(:P-ELS==>(ATTRA:#(-))       (GADJ.MOD(Adj,GOV:#)(-));
(:P-ELS==>(DETE:#)           (DET.MOD:#);
```

(30) Using PLAIN as theorem prover

This is the first example in the illustration of PLAIN as a logic program. In principle it is possible to prove or disprove a formula by transforming it according to logical laws. This is shown here for a DRL formula which corresponds to modus ponens ((a -> b) & a) -> b. The final reduction to the expression "1-true" (i.e. logically true) is the proof that this formula is a theorem.

```
*** REPLC - REPLACEMENT OF LIST

  1  (*: if then
  2       (J1: and <SYSTERM>
  3            (J: if then
  4                 (J1: a)
  5                 (J2: b))
  6            (J: a))
  7       (J2: b));

*** REPLC - RULE USED FOR REPLACEMENT:

  1  (: => <REPLACE>
  2       (*: if then
  3            (J1: $1)
  4            (J2: $2))
  5       (*: or_d <SYSTERM>
  6            (J: NOT $1)
  7            (J: $2)));
```

```
*** REPLC - LIST AFTER REPLACEMENT:

  1  (*: if then
  2       (J1: and <SYSTERM>
  3            (J: or_d <SYSTERM>
  4                 (J: NOT a)
  5                 (J: b))
  6            (J: a))
  7       (J2: b));

*** REPLC - RULE USED FOR REPLACEMENT:

  1  (: => <REPLACE>
  2       (*: and <SYSTERM>
  3            (J: or_d <SYSTERM>
  4                 (J: $1)
  5                 (J: -1))
  6            (J: NOT $1)
  7            (J: -2))
  8       (*: and <SYSTERM>
  9            (J: or_d <SYSTERM>
 10                 (J: -1))
 11            (J: NOT $1)
 12            (J: -2)));
```

```
*** REPLC - LIST AFTER REPLACEMENT:

  1  (*: if then
  2       (J1: and <SYSTERM>
  3            (J: or_d <SYSTERM>
  4                 (J: b))
  5            (J: a))
  6       (J2: b));

*** REPLC - RULE USED FOR REPLACEMENT:

  1  (: => <REPLACE>
  2       (*: or_d <SYSTERM>
  3            (J: $))
  4       (*: $));
```

```
*** REPLC - LIST AFTER REPLACEMENT:

  1  (*: if then
  2       (J1: and <SYSTERM>
  3            (J: b)
  4            (J: a))
  5       (J2: b));

*** REPLC - RULE USED FOR REPLACEMENT:

  1  (: => <REPLACE>
  2       (*: if then
  3            (J1: $1)
  4            (J2: $2))
  5       (*: or_d <SYSTERM>
  6            (J: NOT $1)
  7            (J: $2)));
```

```
*** REPLC - LIST AFTER REPLACEMENT:

  1  (*: or_d <SYSTERM>
  2       (J: NOT and <SYSTERM>
  3            (J: b)
  4            (J: a))
  5       (J: b));

*** REPLC - RULE USED FOR REPLACEMENT:

  1  (: => <REPLACE>
  2       (*: NOT and <SYSTERM>
  3            (J: -))
  4       (*: or_d <SYSTERM>
  5            (J: NOT -)));
```

```
*** REPLC - LIST AFTER REPLACEMENT:

  1  (*: or_d <SYSTERM>
  2       (J: or_d <SYSTERM>
  3            (J: NOT b)
  4            (J: NOT a))
  5       (J: b));

*** REPLC - RULE USED FOR REPLACEMENT:

  1  (: => <REPLACE>
  2       (*: or_d <SYSTERM>
  3            (J: or_d <SYSTERM>
  4                 (-1))
  5            (-2))
  6       (*: or_d <SYSTERM>
  7            (-1)
  8            (-2)));
```

```
*** REPLC - LIST AFTER REPLACEMENT:

  1  (*: or_d <SYSTERM>
  2       (J: NOT b)
  3       (J: NOT a)
  4       (J: b));

*** REPLC - RULE USED FOR REPLACEMENT:

  1  (: => <REPLACE>
  2       (*: or_d <SYSTERM>
  3            (J: $)
  4            (J: NOT $)
  5            (-))
  6       (*: l_true <SYSTERM>));
```

```
*** REPLC - LIST AFTER REPLACEMENT:

  1  (*: l_true <SYSTERM>);

*** REPLC - REPLACEMENT TERMINATED
```

(31) A more complex theorem

This is a somewhat more complicated theorem. The listing of the deduction is fairly long, so that it cannot be printed out. The example originates from

> Werner Dilger, Martin Schneider: Automatisierbare and automatisierte Beweisverfahren fuer die Logik erster Stufe. SFB 99 Universitaet Konstanz, 1979, S. 15.)

```
((p -> q) & (r -> s)) -> ((p v r) -> (q v s))


   *** DED - LIST_1:

      1  (*: if then
      2       (J1: and
      3            (J: if then
      4                 (J1: p)
      5                 (J2: q))
      6            (J: if then
      7                 (J1: r)
      8                 (J2: s)))
      9       (J2: if then
     10            (J1: or_d
     11                 (J: p)
     12                 (J: r))
     13            (J2: or_d
     14                 (J: q)
     15                 (J: s))));


   *** REPLC - LIST AFTER REPLACEMENT

      1  (*: l_true);

   *** REPLC - REPLACEMENT TERMINATED

   *** DED - TASK 18 CPU-SECONDS   3.81
```

(32) The propositional calculus laws as replacement rules

These are the rules used in (30) and (31). They result from the laws of propositional calculus by the fact that in any formula, the parts that correspond to the antecedent of a tautological conditional can be replaced by the postcedent of that conditional. In the form presented here, the replacement rules are a bit complicated since disjunction and conjunction are treated as many-place predicates. Incidentlly, the envisaged inference component of PLAIN as a question answering system will not be based on the process of proving or disproving theorems. However, the rules will be needed to generate normal forms.

```
 1  (: => <REPLACE>              (Reduction of implication)
 2       (*: if then
 3            (J1: $1)
 4            (J2: $2))
 5       (*: or_d
 6            (J: NOT $1)
 7            (J: $2)));
```

```
 1  (: => <REPLACE>              (Reduction of replication)
 2       (*: only if then
 3            (J1: $1)
 4            (J2: $2))
 5       (*: or_d
 6            (J: NOT $2)
 7            (J: $1)));
```

```
 1  (: => <REPLACE>              (Reduction of equivalence)
 2       (*: if and only if then
 3            (J1: $1)
 4            (J2: $2))
 5       (*: and
 6            (J: or_d
 7                 (J: NOT $1)
 8                 (J: $2))
 9            (J: or_d
10                 (J: $1)
11                 (J: NOT $2))));
```

```
 1  (: => <REPLACE>              (Reduction of exclusive or)
 2       (*: either or
 3            (J1: $1)
 4            (J2: $2))
 5       (*: and
 6            (J: or_d
 7                 (J: $1)
 8                 (J: $2))
 9            (J: or_d
10                 (J: NOT $1)
11                 (J: NOT $2))));
```

```
(: => <REPLACE>                (Negation of implication)
     (*: NOT if then
          (J1: $1)
          (J2: $2))
     (*: and
          (J: $1)
          (J: NOT $2)));

(: => <REPLACE>                (1st of De Morgan's laws)
     (*: NOT or_d
          (J: -))
     (*: and
          (J: NOT -)));

(: => <REPLACE>                (2nd of De Morgan's laws)
     (*: NOT and
          (J: -))
     (*: or_d
          (J: NOT -)));

(: => <REPLACE>                (Distributivity of
     (*: or_d                   disjunction)
          (J: and
               (J: $1)
               (J: $2)
               (J: -1))
          (J: -2))
     (*: and
          (J: or_d
               (J: $1)
               (J: -2))
          (J: or_d
               (J: and
                    (J: $2)
                    (J: -1))
               (J: -2))));

(: => <REPLACE>                (Idempotence of disjunction)
     (*: or_d
          (J: $)
          (J: $)
          (-))
     (*: or_d
          (J: $)
          (-)));

(: => <REPLACE>                (Simplification of many-place
     (*: or_d                  disjunction)
          (J: or_d
               (-1))
          (-2))
     (*: or_d
          (-1)
          (-2)));
```

```
 1  (: => <REPLACE>              (Elimination of disjunction
 2       (*: or_d                with one element)
 3            (J: $))
 4       (*: $));

 1  (: => <REPLACE>              (Elimination of disjunction
 2       (*: or_d));             without elements)

 1  (: => <REPLACE>              (Elimination of negated
 2       (*: NOT or_d));         disjunction without element)

 1  (: => <REPLACE>              (Derivation of tautology
 2       (*: or_d                from disjunction)
 3            (J: $)
 4            (J: NOT $)
 5            (-))
 6       (*: l_true));

 1  (: => <REPLACE>              (Elimination of disjunction
 2       (*: or_d                with tautological element)
 3            (J: l_true)
 4            (-))
 5       (*: l_true));

 1  (: => <REPLACE>              (Elimination of antinomy
 2       (*: or_d                from disjunction)
 3            (J: l_false)
 4            (J: $)
 5            (-))
 6       (*: or_d
 7            (J: $)
 8            (-)));

 1  (: => <REPLACE>              (Restricted distributivity
 2       (*: and                 of conjunction)
 3            (J: or_d
 4                 (J: $1)
 5                 (J: -1))
 6            (J: NOT $1)
 7            (J: -2))
 8       (*: and
 9            (J: or_d
10                 (J: -1))
11            (J: NOT $1)
12            (J: -2)));

 1  (: => <REPLACE>              (Idempotence of conjunction)
 2       (*: and
 3            (J: $)
 4            (J: $)
 5            (-))
 6       (*: and
 7            (J: $)
 8            (-)));
```

```
(: => <REPLACE>                 (Simplification of many-place
     (*: and                    conjunction)
          (J: and
               (-1))
          (-2))
     (*: and
          (-1)
          (-2)));

(: => <REPLACE>                 (Elimination of conjunction
     (*: and                    with one element)
          (J: $))
     (*: $));

(: => <REPLACE>                 (Elimination of conjunction
     (*: and));                 without elements)

(: => <REPLACE>                 (Elimination of negated
     (*: NOT and));             conjunctions without elements)

(: => <REPLACE>                 (Derivation of antinomy from
     (*: and                    conjunction)
          (J: $)
          (J: NOT $)
          (-))
     (*: l_false));

(: => <REPLACE>                 (Elimination of conjunction
     (*: and                    with antinomy)
          (J: l_false)
          (-))
     (*: l_false));

(: => <REPLACE>                 (Elimination of tautology
     (*: and                    from conjunction)
          (J: l_true)
          (J: $)
          (-))
     (*: and
          (J: $)
          (-)));
```

(33) Generating and using inference rules

Three axioms lead to the inference concept of PLAIN. Firstly, in a formal system, every deduction is a generation of new expressions on the basis of given expressions according to given rules. Secondly, every logical conclusion is based on a regularity in the object domain. Thirdly, any regularity can be described in natural language as the need arises. The idea of PLAIN inference derived from these principles is the following. Descriptions of the regularities in an object domain which are entered in natural language are used to generate the rules for the formal deductions of the system. This is done according to another set of rules (we call them "meta-semantic rules"), which are to be drawn up for the various syntactic structures of sentences by which regularities are expressed in the natural input language. In the example this is shown using the familiar Aristotelian syllogism. If the general statement "All men are mortal." is true, then one can derive from any sentence matching "X is a man." a sentence "X is mortal." as well as - by means of contraposition - from any sentence matching "X is not mortal." a sentence "X is not a man." The variable "X" matches with any expression which meets the syntactic function of subject in a well formed sentence. In the environment of question answering, one can use the relationship which is established by the general statement in order to prove the statement in reverse order. In order to prove "X is not a man." one can attempt to prove "X is not mortal.", and to prove "X is mortal." one can try to prove "X is a man." All of these applications of the general relationship expressed by the natural language sentence are provided for by a rule of the type "EXPAND". The relation between the subtrees of this rule corresponds to a disjunction: "not is X a man - or - X is mortal". (Since we allow many-place disjunction, an EXPAND rule can also consist of any number of subtrees subordinated to the rule head.) The listings show how the EXPAND rule is generated from the original sentence by replacements according to the meta-semantic rules. The latter are linguistic rules. In turning natural language sentences into inference rules, they provide a semantic interpretation for some lexemes (like "all" and "be", for instance) and for the syntactic structures of the input language. Thus, their validity is analytic. The EXPAND rules, on the contrary, belong to the world knowledge. They arise from the sentences that describe an object domain and can be true or false. They have only empirical validity.

The following technical regulations for EXPAND rules exist. In order to carry out an inference, an EXPAND rule will be applied to a DRL expression, if this expression matches with the negation of one of the subtrees of the rule. The result of the application will be (a conjunction of) the expressions which are formed according to the pattern of the other subtrees in the rule. (Compare the inference drawn from "Socrates is a man." with the subtree "NOT is X a man" of the EXPAND rule matching this expression, the result beeing "Socrates is mortal.") In order to carry out a proof, an EXPAND rule will be applied, if a DRL expression matches one of the subtrees of the rule directly. The result of the application will be (a disjunction of) the negated expressions which are formed according to the pattern of the other subtrees of the rule. (See the expansion of the question "Is Socrates mortal?" by the question "Is Socrates a man?", where the subtree "Is X mortal" of the EXPAND rule matches with the first sentence and the subtree "NOT is X a man" is the pattern for the the sentence to be constructed and negated.

Generating a deduction rule (EXPAND-rule) from a statement:

```
*** DED - LIST_1:
(All men are mortal.)

  1  (ILLOC: assertion'
  2       (PRAED: be
  3            (TEMPS: present_tense')
  4            (MODUS: indicative')
  5            (SUBJE: man
  6                 (REFER: all)
  7                 (NUMRS: plural'))
  8            (ADPRD: mortal)));
```

```
*** REPLC - RULE USED FOR REPLACEMENT:

  1  (: => <REPLACE>
  2       (PRAED: #1
  3            (SUBJE: #2
  4                 (REFER: all)
  5                 (NUMRS: plural')
  6                 (-2))
  7            (-1))
  8       (PRAED: if then
  9            (J1: be
 10                 (TEMPS: present_tense')
 11                 (MODUS: indicative')
 12                 (SUBJE: $)
 13                 (PDKTV: #2
 14                      (REFER: a)
 15                      (-2))
 16                 (-3))
 17            (J2: #1
 18                 (SUBJE: $)
 19                 (-1)
 20                 (-3))));
```

```
*** REPLC - LIST AFTER REPLACEMENT:

  1  (ILLOC: assertion'
  2       (PRAED: if then
  3            (J1: be
  4                 (TEMPS: present_tense')
  5                 (MODUS: indicative')
  6                 (SUBJE: $)
  7                 (PDKTV: man
  8                      (REFER: a))
  9                 (-3))
 10            (J2: be
 11                 (SUBJE: $)
 12                 (TEMPS: present_tense')
 13                 (MODUS: indicative')
 14                 (ADPRD: mortal)
 15                 (-3))));
```

```
*** REPLC - RULE USED FOR REPLACEMENT:

  1  (: => <REPLACE>
  2       (*: if then
  3            (J1: $1)
  4            (J2: $2))
  5       (*: or_d <SYSTERM>
  6            (J: NOT $1)
  7            (J: $2)));
```

```
*** REPLC - LIST AFTER REPLACEMENT:

  1  (ILLOC: assertion'
  2       (PRAED: or_d <SYSTERM>
  3            (J: NOT be
  4                 (TEMPS: present_tense')
  5                 (MODUS: indicative')
  6                 (SUBJE: $)
  7                 (PDKTV: man
  8                      (REFER: a))
  9                 (-3))
 10            (J: be
 11                 (SUBJE: $)
 12                 (TEMPS: present_tense')
 13                 (MODUS: indicative')
 14                 (ADPRD: mortal)
 15                 (-3))));

*** REPLC - RULE USED FOR REPLACEMENT:

  1  (: => <REPLACE>
  2       (ILLOC: assertion'
  3            (PRAED: or_d <SYSTERM>
  4                 (J: -)))
  5       (: | <EXPAND>
  6            (*: -)));
```

```
*** REPLC - LIST AFTER REPLACEMENT:

  1  (: | <EXPAND>
  2       (*: NOT be
  3            (TEMPS: present_tense')
  4            (MODUS: indicative')
  5            (SUBJE: $)
  6            (PDKTV: man
  7                 (REFER: a))
  8            (-3))
  9       (*: be
 10            (SUBJE: $)
 11            (TEMPS: present_tense')
 12            (MODUS: indicative')
 13            (ADPRD: mortal)
 14            (-3)));

*** REPLC - REPLACEMENT TERMINATED

*** DED - FILING LIST_1 INTO PARTITION   1 OF FILE
```

Applying the rule in an inference:

```
*** DED - LIST_1:
(Socrates is a man.)

  1  (ILLOC: assertion'
  2       (PRAED: be
  3            (TEMPS: present_tense')
  4            (MODUS: indicative')
  5            (SUBJE: Sokrates)
  6            (PDKTV: man
  7                 (REFER: a))));

*** EXPAN - RULE APPLIED TO SUBLIST (  2):

  1  (: | <EXPAND>
  2       (*: NOT be
  3            (TEMPS: present_tense')
  4            (MODUS: indicative')
  5            (SUBJE: $)
  6            (PDKTV: man
  7                 (REFER: a))
  8            (-3))
  9       (*: be
 10            (SUBJE: $)
 11            (TEMPS: present_tense')
 12            (MODUS: indicative')
 13            (ADPRD: mortal)
 14            (-3)));

*** EXPAN - DERIVED LIST(S):

  1  (ILLOC: assertion'
  2       (PRAED: be
  3            (SUBJE: Sokrates)
  4            (TEMPS: present_tense')
  5            (MODUS: indicative')
  6            (ADPRD: mortal)));

*** EXPAN - EXPANSION TERMINATED
```

Applying the rule in order to expand a question:

```
*** DED - LIST_1:
(Is Socrates mortal?)

  1  (ILLOC: question'
  2       (PRAED: be
  3            (TEMPS: present_tense')
  4            (MODUS: indicative')
  5            (SUBJE: Sokrates)
  6            (ADPRD: mortal)));

*** EXPAN - RULE APPLIED TO SUBLIST (  2):

  1  (: | <EXPAND>
  2       (*: NOT be
  3            (TEMPS: present_tense')
  4            (MODUS: indicative')
  5            (SUBJE: $)
  6            (PDKTV: man
  7                 (REFER: a))
  8            (-3))
  9       (*: be
 10            (SUBJE: $)
 11            (TEMPS: present_tense')
 12            (MODUS: indicative')
 13            (ADPRD: mortal)
 14            (-3)));

*** EXPAN - DERIVED LIST(S):

  1  (ILLOC: question'
  2       (PRAED: be
  3            (TEMPS: present_tense')
  4            (MODUS: indicative')
  5            (SUBJE: Sokrates)
  6            (PDKTV: man
  7                 (REFER: a))));

*** EXPAN - EXPANSION TERMINATED
```

(34) Meta-semantic rules of German

The crucial idea of PLAINs semantics is to formulate deduction rules directly in terms of syntactic representations of particular natural languages. No translation into a "deep" semantic representation as, for instance, logical calculus is envisaged, though, of course, this could be done by means of replacement rules. The reason for the direct approach is the following. It is probable that every syntactic feature of a natural language corresponds to some semantic distinction. Therefore, the artificial language which will be suited best in order to map all of the semantic differences of a natural languge will be the structural description of that natural language itself. The DRL formalism is flexible enough to associate meta-semantic rules to any lexeme or syntactic structure, thus providing the means for simulating the inferences which can be drawn on the basis of this lexeme or structure in natural language. Allthough only simple cases of general

statements in German were studied up to now, a few of the rules are listed below in order to illustrate the freedom of dealing with syntactic variety which DRL provides.

```
 1  (: => <REPLACE>                          ("Jeder x tut z")
 2       (PRAED: #1                          (Every x does z)
 3            (SUBJE: #2
 4                 (REFER: jed)
 5                 (NUMRS: singular')
 6                 (-2))
 7            (-1))
 8       (PRAED: wenn dann
 9            (J1: sein
10                 (TEMPS: praesens')
11                 (MODUS: indikativ')
12                 (SUBJE: $)
13                 (PDKTV: #2
14                      (REFER: ein)
15                      (-2))
16                 (-3))
17            (J2: #1
18                 (SUBJE: $)
19                 (-1)
20                 (-3))));
```

```
 1  (: => <REPLACE>                           ("Kein x tut z")
 2       (PRAED: #1                           (No x does z)
 3            (SUBJE: #2
 4                 (REFER: kein)
 5                 (NUMRS: singular')
 6                 (-2))
 7            (-1))
 8       (PRAED: wenn dann
 9            (J1: sein
10                 (TEMPS: praesens')
11                 (MODUS: indikativ')
12                 (SUBJE: $)
13                 (PDKTV: #2
14                      (REFER: ein)
15                      (-2))
16                 (-3))
17            (J2: NICHT #1
18                 (SUBJE: $)
19                 (-1)
20                 (-3))));
```

```
 1  (: => <REPLACE>                          ("Wer x tut, tut z")
 2       (PRAED: #1                          (He who does x does z)
 3            (SUBJE: hypo'
 4                 (PRAED: #2
 5                      (SUBJE: w_person')
 6                      (-2)))
 7            (-1))
 8       (PRAED: wenn dann
 9            (J1: #2
10                 (SUBJE: $)
11                 (-2))
12            (J2: #1
13                 (SUBJE: $)
14                 (-1))));
```

```
 1  (: => <REPLACE>                          ("x ist kein z")
 2       (*: #1                              (x is not z)
 3            (PDKTV: #2
 4                 (REFER: kein)
 5                 (-2))
 6            (-1))
 7       (*: NICHT #1
 8            (PDKTV: #2
 9                 (REFER: ein)
10                 (-2))
11            (-1)));
```

(35) Meaning postulates

According to the same priciples as in (33), here the input "Ein Junggeselle ist ein unverheirateter Mann" (A bachelor is an unmarried male) and "Ledig ist, wer unverheiratet ist" (Someone who is single is unmarried) is converted into EXPAND rules. Then these rules are applied to the DRL representation of "Fritz ist ein Junggeselle" (Fred is a bachelor) and result in DRL expressions which say "Fritz ist ein Mann" (Fred is a male), "Fritz nicht verheiratet" (Fred is not married), and "Fritz ist ledig" (Fred is single). Statements like the two original sentences above – according to Carnap – are so-called meaning postulates. They are used to describe the meanings of "Junggeselle" and of "ledig". The semantic features of componential analysis can easily be recognised in the derived clauses. Should one view the corresponding relationships between concepts, as it is usually done, as part of the linguistic knowledge and store them in the lexicon? It is striking that meaning postulates can be smoothly converted into inference rules by means of the same meta-rules which are used to convert descriptions of object domains into inference rules. Therefore, one can think of treating meaning postulates as part of the world knowledge. Theoretically this is also justifiable. "Junggeselle" as well as "ledig" are terms for referential use. Their meaning cannot be learned without the learner being familiar with any object domain. World understanding and referential semantics go hand in hand. The meanings of the expressions of language change as the knowledge about the world changes. This is very obvious in scientific terminology, but it is also true elsewhere. A flexible language processing program, e.g. an expert system, must therefore be constructed in such a manner that the meanings of the referential terms are not fixed but rather that they can be influenced by the user, who is free to enter meaning postulates as well as other encyclopedic knowledge.

```
*** DED - LIST_1:
(Jeder Junggeselle ist ein unverheirateter Mann.)

  1  (ILLOC: aussage'
  2       (PRAED: sein
  3            (TEMPS: praesens')
  4            (MODUS: indikativ')
  5            (SUBJE: junggeselle
  6                 (REFER: jed)
  7                 (NUMRS: singular'))
  8            (PDKTV: mann
  9                 (REFER: ein)
 10                 (ATTRB: unverheiratet))));

*** REPLC - LIST AFTER REPLACEMENT

  1  (& <CON>
  2       (: | <EXPAND>
  3            (*: sein
  4                 (ADPRD: NICHT verheiratet)
  5                 (SUBJE: $)
  6                 (TEMPS: praesens')
  7                 (MODUS: indikativ')
  8                 (-3))
  9            (*: NICHT sein
 10                 (TEMPS: praesens')
 11                 (MODUS: indikativ')
 12                 (SUBJE: $)
 13                 (PDKTV: junggeselle
 14                      (REFER: ein))
 15                 (-3)))
 16       (: | <EXPAND>
 17            (*: sein
 18                 (PDKTV: mann
 19                      (REFER: ein))
 20                 (SUBJE: $)
 21                 (TEMPS: praesens')
 22                 (MODUS: indikativ')
 23                 (-3))
 24            (*: NICHT sein
 25                 (TEMPS: praesens')
 26                 (MODUS: indikativ')
 27                 (SUBJE: $)
 28                 (PDKTV: junggeselle
 29                      (REFER: ein))
 30                 (-3))));

*** DED - FILING LIST_1 INTO PARTITION   1 OF FILE
         CONJUNCT SUBLISTS ARE STORED SEPARATELY
```

```
*** DED - LIST_2:
(Ledig ist, wer unverheiratet ist.)

  1  (ILLOC: aussage'
  2       (PRAED: sein
  3            (TEMPS: praesens')
  4            (MODUS: indikativ')
  5            (SUBJE: hypo'
  6                 (PRAED: sein
  7                      (TEMPS: praesens')
  8                      (MODUS: indikativ')
  9                      (SUBJE: w_person')
 10                      (ADPRD: unverheiratet)))
 11            (ADPRD: ledig)));


*** REPLC - LIST AFTER REPLACEMENT

  1  (: | <EXPAND>
  2       (*: sein
  3            (SUBJE: $)
  4            (TEMPS: praesens')
  5            (MODUS: indikativ')
  6            (ADPRD: ledig))
  7       (*: sein
  8            (SUBJE: $)
  9            (TEMPS: praesens')
 10            (MODUS: indikativ')
 11            (ADPRD: verheiratet)));

*** DED - FILING LIST_1 INTO PARTITION   1 OF FILE
```

```
*** DED - LIST_1:
(Fritz ist ein Junggeselle.)

  1  (ILLOC: aussage'
  2       (PRAED: sein
  3            (TEMPS: praesens')
  4            (MODUS: indikativ')
  5            (SUBJE: Fritz)
  6            (PDKTV: junggeselle
  7                 (REFER: ein))));
```

```
*** RESEX - RESULTS OF INFERENCE:

CONNECTIVE   1: und
```

```
LIST   2 (SUB   1):

  1  (ILLOC: aussage'
  2       (PRAED: sein
  3            (TEMPS: praesens')
  4            (MODUS: indikativ')
  5            (SUBJE: Fritz)
  6            (PDKTV: junggeselle
  7                 (REFER: ein))));

LIST   3 (SUB   1):

  1  (ILLOC: aussage'
  2       (PRAED: sein
  3            (TEMPS: praesens')
  4            (MODUS: indikativ')
  5            (SUBJE: Fritz)
  6            (PDKTV: mann
  7                 (REFER: ein))));

LIST   4 (SUB   1):

  1  (ILLOC: aussage'
  2       (PRAED: NICHT sein
  3            (ADPRD: verheiratet)
  4            (TEMPS: praesens')
  5            (MODUS: indikativ')
  6            (SUBJE: Fritz)));

LIST   5 (SUB   1):

  1  (ILLOC: aussage'
  2       (PRAED: sein
  3            (ADPRD: ledig)
  4            (TEMPS: praesens')
  5            (MODUS: indikativ')
  6            (SUBJE: Fritz)));
*
```

(36) Drawing up a knowledge base

This is to show the possibilities the PLAIN approach offers for the construction of an expert system. If rules have been formulated to deduce EXPAND rules from general sentences according to their syntactic forms, then descriptions of any object domain can be entered by the user in natural language. In the example below the domain is a small fraction of German traffic law. (The German sentences fed into the system describe the following regularities: "Anyone who stops for more than three minutes or leaves his car is parking. Anyone who parks less than five meters from an intersection is guilty of an offence. Anyone who is guilty of an offence either pays a fine or charges will be filed against him.") EXPAND rules are deduced from these descriptions. The declaratively stated knowledge is thus converted into algorithmic form. The EXPAND rules are stored in the knowledge base. In order to explore the linguistic and logical consequences of the various descriptions and rules, the program DED is used to apply repeatedly all of the available EXPAND rules to a given DRL expression until all of the possible derivations are produced. Such a generation of all of deducible DRL expressions for a given expression is called an "expansion" (hence the name "EXPAND for the type of rules).

```
*** DED - LIST_1:

("Wer mehr als drei Minuten haelt oder seinen Wagen verlaesst,
parkt" - Anyone who stops for more than three minutes or leaves
his car is parking)

  1  (ILLOC: aussage'
  2       (PRAED: parken
  3            (TEMPS: praesens')
  4            (MODUS: indikativ')
  5            (SUBJE: hypo'
  6                 (PRAED: oder_d <SYSTERM>
  7                      (J: halten
  8                           (TEMPS: praesens')
  9                           (MODUS: indikativ')
 10                           (SUBJE: w_person')
 11                           (ADVRB: mehr
 12                                (COMPR: als
 13                                     (: minute
 14                                          (QUANT: drei)))))
 15                      (J: verlassen
 16                           (TEMPS: praesens')
 17                           (MODUS: indikativ')
 18                           (SUBJE: w_person')
 19                           (TRANS: wagen
 20                                (NUMRS: singular')
 21                                (REFER: d')
 22                                (ZUORD: sein)))))));

*** REPLC - LIST AFTER REPLACEMENT

  1  (& <CON>
  2       (: | <EXPAND>
  3            (*: NICHT verlassen
  4                 (TEMPS: praesens')
  5                 (MODUS: indikativ')
  6                 (SUBJE: $)
  7                 (TRANS: wagen
  8                      (NUMRS: singular')
  9                      (REFER: d')
 10                      (ZUORD: sein)))
 11            (*: parken
 12                 (SUBJE: $)
 13                 (TEMPS: praesens')
 14                 (MODUS: indikativ')))
 15       (: | <EXPAND>
 16            (*: NICHT halten
 17                 (TEMPS: praesens')
 18                 (MODUS: indikativ')
 19                 (SUBJE: $)
 20                 (ADVRB: mehr
 21                      (COMPR: als
 22                           (: minute
 23                                (QUANT: 3)))))
 24            (*: parken
 25                 (SUBJE: $)
 26                 (TEMPS: praesens')
 27                 (MODUS: indikativ'))));

*** DED - FILING LIST_1 INTO PARTITION   1 OF FILE
         CONJUNCT SUBLISTS ARE STORED SEPARATELY
```

*** DED - LIST_1:

("Wer weniger als 5 m von einer Kreuzung entfernt parkt,
begeht eine Ordnungswidrigkeit." - Anyone who parks less than
five meters from an intersection is guilty of an offence.)

```
 1  (ILLOC: aussage'
 2       (PRAED: begehen
 3            (TEMPS: praesens')
 4            (MODUS: indikativ')
 5            (SUBJE: hypo'
 6                 (PRAED: parken
 7                      (TEMPS: praesens')
 8                      (MODUS: indikativ')
 9                      (SUBJE: w_person')
10                      (ADVRB: entfernt
11                           (CASPP: von
12                                (: kreuzung
13                                     (NUMRS: singular')
14                                     (REFER: ein)))
15                           (ADVRB: weniger
16                                (COMPR: als
17                                     (: meter
18                                          (QUANT: fuenf)))))))
19            (TRANS: ordnungswidrigkeit
20                 (NUMRS: singular')
21                 (REFER: ein))));
```

*** REPLC - LIST AFTER REPLACEMENT

```
 1  (: | <EXPAND>
 2       (*: NICHT parken
 3            (SUBJE: $)
 4            (TEMPS: praesens')
 5            (MODUS: indikativ')
 6            (ADVRB: entfernt
 7                 (CASPP: von
 8                      (: kreuzung
 9                           (NUMRS: singular')
10                           (REFER: ein)))
11                 (ADVRB: weniger
12                      (COMPR: als
13                           (: meter
14                                (QUANT: 5))))))
15       (*: begehen
16            (SUBJE: $)
17            (TEMPS: praesens')
18            (MODUS: indikativ')
19            (TRANS: ordnungswidrigkeit
20                 (NUMRS: singular')
21                 (REFER: ein))));
```

*** DED - FILING LIST_1 INTO PARTITION   1 OF FILE

```
*** DED - LIST_1:

("Wenn jemand eine Ordnungswidrigkeit begeht, zahlt er
Bussgeld, oder es wird Anzeige gegen ihn erhoben." - If
someone is guilty of an offence then he either pays a fine
or charges are filed against him.)

  1  (ILLOC: aussage'
  2       (PRAED: wenn dann
  3            (J1: begehen
  4                 (TEMPS: praesens')
  5                 (MODUS: indikativ')
  6                 (SUBJE: jemand)
  7                 (TRANS: ordnungswidrigkeit
  8                      (NUMRS: singular')
  9                      (REFER: ein)))
 10            (J2: oder_d
 11                 (J1: zahlen
 12                      (TEMPS: praesens')
 13                      (MODUS: indikativ')
 14                      (SUBJE: anaphor_sgm')
 15                      (TRANS: bussgeld
 16                           (REFER: ein)))
 17                 (J2: passiv'
 18                      (TEMPS: praesens')
 19                      (MODUS: indikativ')
 20                      (ADPRD: erheben
 21                           (: anzeige))
 22                      (CASPP: gegen
 23                           (: anaphor_sgm'))))));

*** REPLC - LIST AFTER REPLACEMENT

  1  (: | <EXPAND>
  2       (*: zahlen
  3            (TEMPS: praesens')
  4            (MODUS: indikativ')
  5            (SUBJE: $)
  6            (TRANS: bussgeld
  7                 (REFER: ein)))
  8       (*: passiv'
  9            (TEMPS: praesens')
 10            (MODUS: indikativ')
 11            (ADPRD: erheben
 12                 (: anzeige))
 13            (CASPP: gegen
 14                 (: $)))
 15       (*: NICHT begehen
 16            (TEMPS: praesens')
 17            (MODUS: indikativ')
 18            (SUBJE: $)
 19            (TRANS: ordnungswidrigkeit
 20                 (NUMRS: singular')
 21                 (REFER: ein)))));

*** DED - FILING LIST_1 INTO PARTITION   1 OF FILE
```

(37) Inference

The task of inference is formally defined for PLAIN as the expansion of a premise p by the conjunction of all conclusions k1 to kn, which can be generated from p on the basis of a given knowledge base. Here such an expansion is documented on the basis of (36) for the sentence "Emil haelt mehr als 3 Minuten weniger als 5 m entfernt von einer Kreuzung." (Emil stops for more than 3 minutes less than 5 meters away from an intersection). The resulting DRL structure can be read as follows: "Emil stops for more than 3 minutes less than 5 meters away from an intersection, hence he is parking, hence he is guilty of an offence, hence he must either pay a fine or charges will be filed against him.")

*** DED - LIST_1:

("Emil haelt mehr als 3 Minuten lang weniger als 5 m von einer Kreuzung entfernt." - Emil stops for more than 3 minutes less than 5 meters away from an intersection.)

```
 1  (ILLOC: aussage'
 2       (PRAED: halten
 3            (TEMPS: praesens')
 4            (MODUS: indikativ')
 5            (SUBJE: Emil)
 6            (ADVRB: mehr
 7                 (COMPR: als
 8                      (: minute
 9                           (QUANT: 3))))
10            (ADVRB: entfernt
11                 (CASPP: von
12                      (: kreuzung
13                           (NUMRS: singular')
14                           (REFER: ein)))
15                 (ADVRB: weniger
16                      (COMPR: als
17                           (: meter
18                                (QUANT: 5)))))));
```

```
*** RESEX - RESULTS OF INFERENCE:

CONNECTIVE   1: und
```

```
LIST   2 (SUB   1):

  1  (ILLOC: aussage'
  2       (PRAED: halten
  3            (TEMPS: praesens')
  4            (MODUS: indikativ')
  5            (SUBJE: Emil)
  6            (ADVRB: mehr
  7                 (COMPR: als
  8                      (: minute
  9                           (QUANT: 3))))
 10            (ADVRB: entfernt
 11                 (CASPP: von
 12                      (: kreuzung
 13                           (NUMRS: singular')
 14                           (REFER: ein)))
 15                 (ADVRB: weniger
 16                      (COMPR: als
 17                           (: meter
 18                                (QUANT: 5)))))));

LIST   3 (SUB   1):

  1  (ILLOC: aussage'
  2       (PRAED: parken
  3            (SUBJE: Emil)
  4            (TEMPS: praesens')
  5            (MODUS: indikativ')
  6            (ADVRB: entfernt
  7                 (CASPP: von
  8                      (: kreuzung
  9                           (NUMRS: singular')
 10                           (REFER: ein)))
 11                 (ADVRB: weniger
 12                      (COMPR: als
 13                           (: meter
 14                                (QUANT: 5)))))));

LIST   4 (SUB   1):

  1  (ILLOC: aussage'
  2       (PRAED: begehen
  3            (SUBJE: Emil)
  4            (TEMPS: praesens')
  5            (MODUS: indikativ')
  6            (TRANS: ordnungswidrigkeit
  7                 (NUMRS: singular')
  8                 (REFER: ein))));
```

```
CONNECTIVE   5 (SUB   1): oder_d
_____________________________________________________________________

LIST   6 (SUB   5):

  1  (ILLOC: aussage'
  2       (PRAED: zahlen
  3            (TEMPS: praesens')
  4            (MODUS: indikativ')
  5            (SUBJE: Emil)
  6            (TRANS: bussgeld
  7                 (REFER: ein))));

LIST   7 (SUB   5):

  1  (ILLOC: aussage'
  2       (PRAED: passiv'
  3            (TEMPS: praesens')
  4            (MODUS: indikativ')
  5            (ADPRD: erheben
  6                 (: anzeige))
  7            (CASPP: gegen
  8                 (: Emil)))
*
```

(38) Proofs

The task of proving can be defined in a formal system as the expansion of a conclusion k by a disjunction of all the sets of premises p1 to pm from which k results on the basis of the given knowledge base, and as a subsequent evaluation of this expansion on the basis of facts and logical laws. (An expansion proves k on logical grounds, if it can be reduced according to the rules in (32) to "l_wahr" i.e. tautologically true.) The program for the evaluation is not yet implemented. Within the framework of question answering, this approach can be used as follows. A question like "Wird gegen Fritz Anzeige erhoben?" (Are charges filed against Fred?) is expanded according to EXPAND rules by all questions which, when answered in the affirmative enable the system to answer the original question in the affirmative. If a question can be answered, then the expansion is used to explain the way it was derived to the user. For example, "Charges are filed against Fred, because he is guilty of an offence, because he parked less than 5 meters away from an intersection, since he stopped for more than 3 minutes, and he did not pay a fine." In an expert system such transparency of the inferences is highly desirable. If there is not enough information to evaluate the expansion affirmatively, then the expansion can be used to generate counter questions, for example "Did Fred get out of his car?", "Has he already paid a fine?" etc. The capability of posing counter questions is also an important requirement for expert systems.

*** DED - LIST_1:

("Wird gegen Fritz Anzeige erhoben?" - Are charges filed against Fred?)

```
1  (ILLOC: e_frage'
2       (PRAED: passiv'
3            (TEMPS: praesens')
4            (MODUS: indikativ')
5            (ADPRD: erheben
6                 (: anzeige))
7            (CASPP: gegen
8                 (: Fritz)))
```

```
*** RESEX - RESULTS OF PROOF EXPANSION:

CONNECTIVE   1: oder_d
```

```
LIST   2 (SUB   1):

  1  (ILLOC: e_frage'
  2       (PRAED: passiv'
  3            (TEMPS: praesens')
  4            (MODUS: indikativ')
  5            (ADPRD: erheben
  6                 (: anzeige))
  7            (CASPP: gegen
  8                 (: Fritz)))

CONNECTIVE   3 (SUB   1): und
```

```
LIST   4 (SUB   3):

  1  (ILLOC: e_frage'
  2       (PRAED: NICHT zahlen
  3            (TEMPS: praesens')
  4            (MODUS: indikativ')
  5            (SUBJE: Fritz)
  6            (TRANS: bussgeld
  7                 (REFER: ein))));

CONNECTIVE   5 (SUB   3): oder_d
```

```
LIST   6 (SUB   5):

  1  (ILLOC: e_frage'
  2       (PRAED: begehen
  3            (TEMPS: praesens')
  4            (MODUS: indikativ')
  5            (SUBJE: Fritz)
  6            (TRANS: ordnungswidrigkeit
  7                 (NUMRS: singular')
  8                 (REFER: ein))));

LIST   7 (SUB   5):

  1  (ILLOC: e_frage'
  2       (PRAED: parken
  3            (SUBJE: Fritz)
  4            (TEMPS: praesens')
  5            (MODUS: indikativ')
  6            (ADVRB: entfernt
  7                 (CASPP: von
  8                      (: kreuzung
  9                           (NUMRS: singular')
 10                           (REFER: ein)))
 11                 (ADVRB: weniger
 12                      (COMPR: als
 13                           (: meter
 14                                (QUANT: 5)))))));
```

LIST 8 (SUB 5):

```
 1  (ILLOC: e_frage'
 2       (PRAED: verlassen
 3            (TEMPS: praesens')
 4            (MODUS: indikativ')
 5            (SUBJE: Fritz)
 6            (TRANS: wagen
 7                 (NUMRS: singular')
 8                 (REFER: d')
 9                 (ZUORD: sein))
10            (ADVRB: entfernt
11                 (CASPP: von
12                      (: kreuzung
13                           (NUMRS: singular')
14                           (REFER: ein)))
15                 (ADVRB: weniger
16                      (COMPR: als
17                           (: meter
18                                (QUANT: 5)))))));
```

LIST 9 (SUB 5):

```
 1  (ILLOC: e_frage'
 2       (PRAED: halten
 3            (TEMPS: praesens')
 4            (MODUS: indikativ')
 5            (SUBJE: Fritz)
 6            (ADVRB: mehr
 7                 (COMPR: als
 8                      (: minute
 9                           (QUANT: 3))))
10            (ADVRB: entfernt
11                 (CASPP: von
12                      (: kreuzung
13                           (NUMRS: singular')
14                           (REFER: ein)))
15                 (ADVRB: weniger
16                      (COMPR: als
17                           (: meter
18                                (QUANT: 5)))))));
```

B i b l i o g r a p h y o n P L A I N

Peter Hellwig, University of Heidelberg

(L-1) Formal-desambiguierte Repraesentation. Vorueberlegungen zur maschinellen Bedeutungsanalyse auf der Grundlage der Valenzidee. (Dissertation Heidelberg 1974.) Stuttgart 1978.

(L-2) "Ein Computermodell fuer das Folgern in natuerlicher Sprache". In: P. Eisenberg (Hg.): Semantik und kuenstliche Intelligenz. Beitraege zur automatischen Sprachbearbeitung II. Berlin, New York 1977. pp. 59 - 85.

(L-3) "Dependenzanalyse und Bedeutungspostulate - eine Alternative zur generativen Transformationsgrammatik". In: Linguistische Berichte 52 (1977), pp. 32 - 51.

(L-4) "Logisch-funktionale Saetze natuerlicher Sprache im Rahmen eines automatischen Deduktionssystem". In: W.U. Dressler, W. Meid (Hg.): Proceedings of the Twelfth International Congress of Linguists. Vienna, August 28 - September 2, 1977. Innsbruck 1978, pp. 589 - 591.

(L-5) "PLAIN - Ein Programmsystem zur Sprachbeschreibung und maschinellen Sprachbearbeitung". In: Sprache und Datenverarbeitung. Heft 1/1978, pp. 16 - 31.

(L-6) "PLAIN - Eine Valenzsyntax und wozu sie gut ist". In: D. Krallmann (Hg.): Dialogsysteme und Textverarbeitung. LDV-Fittings e.V., Essen 1980, pp. 17 - 43.

(L-7) "PLAIN - A Program System for Dependency Analysis and for Simulating Natural Language Inference." In: Leonard Bolc (Hg.): Representation and Processing of Natural Language. Muenchen, Wien, London 1980, pp. 271 - 376.

(L-8) PLAIN - Ein Programmsystem zur dependentiellen Syntaxanalyse und zur Simulation von Folgerungen in natuerlicher Sprache. Germanistisches Seminar der Universitaet Heidelberg, Postfach 105760, D-69 Heidelberg. (PLAIN - Manual)

(L-9) Programmsystem PLAIN - "Programs for Language Analysis and Inference" - Benutzungsanleitung. Germanistisches Seminar der Universitaet Heidelberg, Postfach 105760, D-69 Heidelberg. (PLAIN User's Guide)

(L-10) "Bausteine des Deutschen". Daten fuer das Programmsystem PLAIN. Germanistisches Seminar der Universitaet Heidelberg, Postfach 105760, D-69 Heidelberg.

(L-11) "Rechnen mit Sprache - Moeglichkeiten und Probleme der maschinellen Sprachverarbeitung." In "Der Deutschunterricht", Heft 4/1983, pp.15-38.

(L-12) "Dependency Unification Grammar (DUG)", 1986. To appear in COLING 1986.