First Steps with PLAIN

Version 2.0 (September 2014)

www.plain-nlp.de

These instructions should help you to use PLAIN (Programs for Language Analysis and Inference) right away. The principle is learning by doing. At first you can work with the demonstration material included in the delivery. In doing so you organize your own demo. Then you might augment existing files with your own entries, recreate the database and look at the results. Later you may draw up and execute your own files complying with the examples in the demo files.

You may also consult "The User's Guide to PLAIN" and "The Linguist's Guide to PLAIN" while working with this paper.

Contents

1 Installation

The PLAIN system can be downloaded from *http://www.plain-nlp.de*. Save and execute the file *PlainSetup.exe.* This will invoke the Plain Setup Wizard. Follow the instructions.

After setup the following directories should be created on your computer.

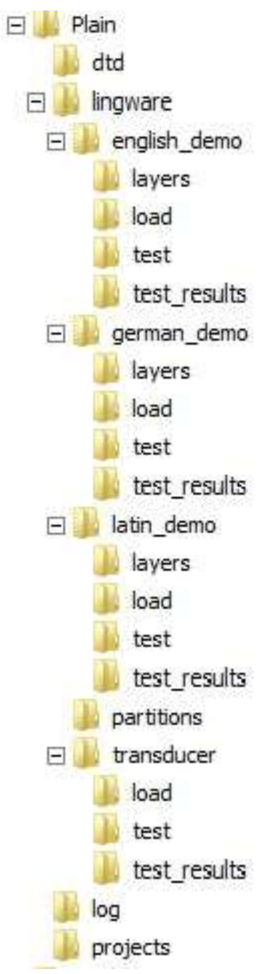


Fig. 1 The *plain* directory and its subdirectories

The directory *Plain* includes the executable *PlainIDE.exe*. Execute this file in order to run PLAIN. The directory *dtd* includes the data definitions for the PLAIN xml resource files. The *lingware* directories contains linguistic material for the purpose of demonstration and testing. The directory *projects* contains definition files and databases of so-called projects. The kind of data in these directories and their use is explained in the following sections. The directory *Plain* also includes *unins000.exe* for removing PLAIN from your computer.

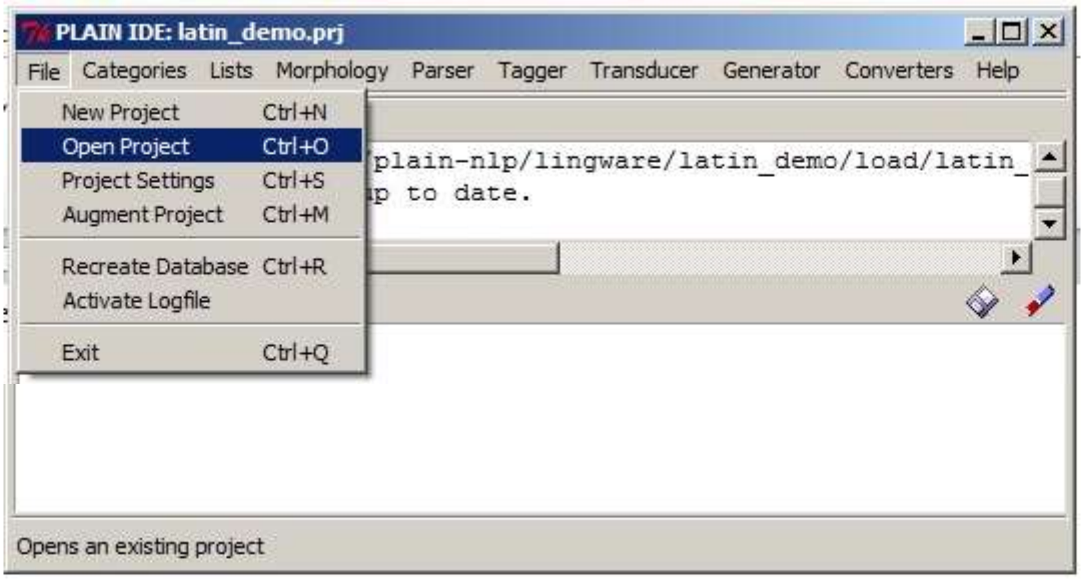
If there is any problem installing and using PLAIN or if you like to give us feed-back, please send an e-mail to *hellwig@cl.uni-heidelberg.de*.

2 Open a project

PLAIN always works on an actual *project*. A project is a particular constellation of linguistic resource files, for example the description of a particular language. These files are turned into an internal database for the program's execution. If a project is opened then a message appears in the *Log Messages* box similar to the following:

20:19:17 Opening project 'latin_demo.prj'

If no project is opened yet or if you want to open another project then you must issue the *OpenProject* command in the *File* menu:.



```
plain-nlp/lingware/latin_demo/load/latin_
p to date.
```

Opens an existing project

Fig. 2 *Open Project* in the *File* menu

Select a project file from the appearing window:

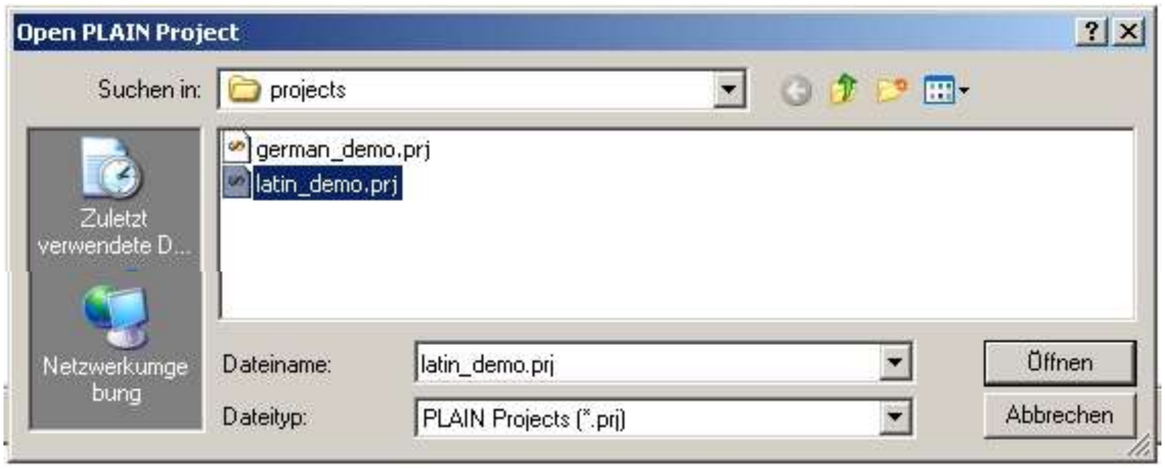


Fig. 3 Selecting the project file *latin_demo.prj*

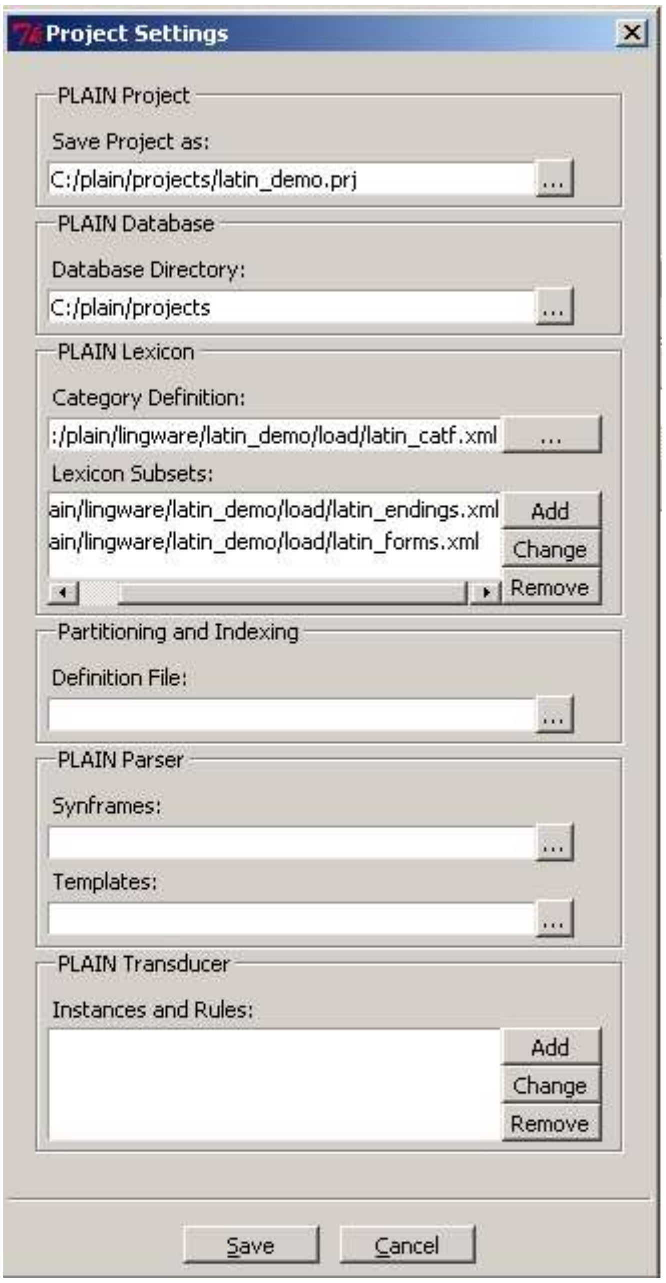
Note: When PLAIN is started then the last project worked on is usually opened automatically. However, when you execute PLAIN for the first time, the program may start with no project open. What you must do in this case is issue the *OpenProject* command. Then browse the directory *C:Plain/projects* which should look like the one in Fig. 3. Select one of the .prj files

3 Recreate Database

The linguistic resource files are stored in a database. This database may be damaged due to a former abnormal program end. So, here is a good advise for the future: Whenever the program does not work as it should then try first the command *Recreate Database* from the *File* menu.

4 Inspecting the actual project

Select *Project Settings* in the *File* menu. The following window appears:


Project Settings

PLAIN Project

Save Project as: C:/plain/projects/latin_demo.prj

PLAIN Database

Database Directory: C:/plain/projects

PLAIN Lexicon

Category Definition: :/plain/lingware/latin_demo/load/latin_catf.xml

Lexicon Subsets:
ain/lingware/latin_demo/load/latin_endings.xml
ain/lingware/latin_demo/load/latin_forms.xml

Partitioning and Indexing

Definition File:

PLAIN Parser

Synframes:

Templates:

PLAIN Transducer

Instances and Rules:

Fig. 4 The *Project Settings* of the project *latin_demo*

As you can see, the project comprises the following resource files:

- the project file .../*projects/latin.demo.prj*
- the category definition file *.../lingware/latin_demo /load/latin_catf.xml*
- the lexicon subsets *.../lingware/latin_demo /load/latin_endings.xml*

  and *.../lingware/latin_demo /load/latin_forms.xml*

Why don't you have a look at the contents of these files in your text editor now. The notation should be widely self explaining. For the exact format refer to "The Linguist's Guide to PLAIN".

Note:

The file .../*load/latin_forms* is derived automatically from the file *.../layers/latin_cardlforms.xml.* The latter file contains so-called cardinal forms. They are the preferred format of PLAIN for drawing up the morpho-syntactic lexicon.

```
<cardlform>silva silvae silvae</cardlform>
<cardlform>servus servi servi</cardlform>
<cardlform>templum templi templa</cardlform>
<cardlform>amare amo amavi amatum</cardlform>
<cardlform>delere deleo delevi deletum</cardlform>
<cardlform>facere facio feci factum</cardlform>
<cardlform>mordere mordeo momordi morsum</cardlform>
<cardlform>spondere spondeo spopondi sponsum</cardlform>
```

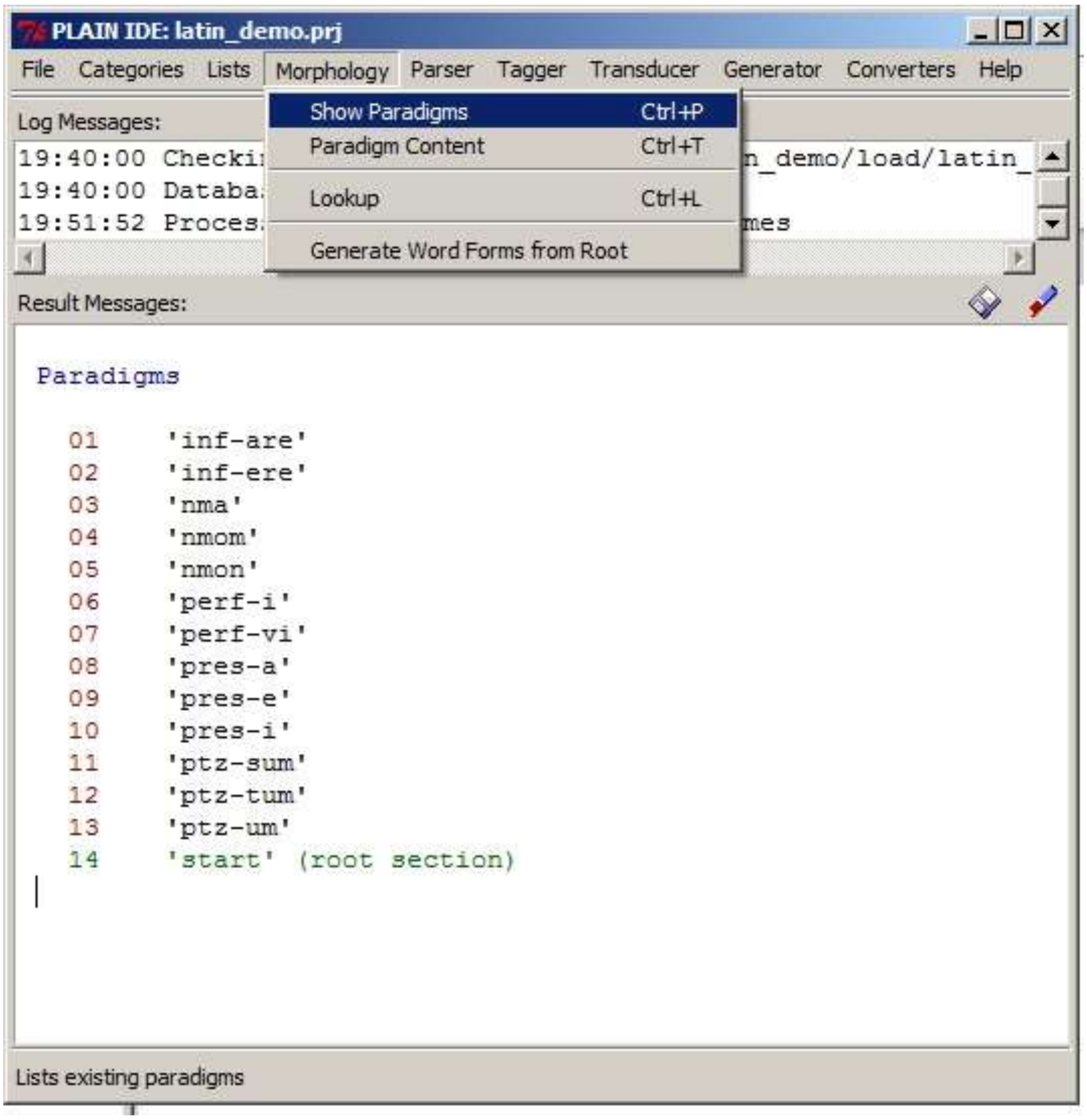
Fig. 5 Cardinal Forms (file *latin_cardlforms.xml*)

The conversion from *.../lingware/latin_demo /layers/latin_cardlforms.xml* into *.../lingware/latin_demo /load/latin_forms* is demonstrated in chapter 12 below.

5 Morphology

Let us start with a demonstration of the morphological component. Press the *Morphology* button of the main menu bar! Then try each command from the drop out menu.

5.1 Show Paradigms


PLAIN IDE: latin_demo.prj
File Categories Lists Morphology Parser Tagger Transducer Generator Converters Help
Show Paradigms Ctrl+P
Paradigm Content Ctrl+T
Lookup Ctrl+L
Generate Word Forms from Root
Log Messages:
Result Messages:
Lists existing paradigms

```
19:40:00 Checki                                    n_demo/load/latin_
19:40:00 Databa
19:51:52 Proces                                    mes
```

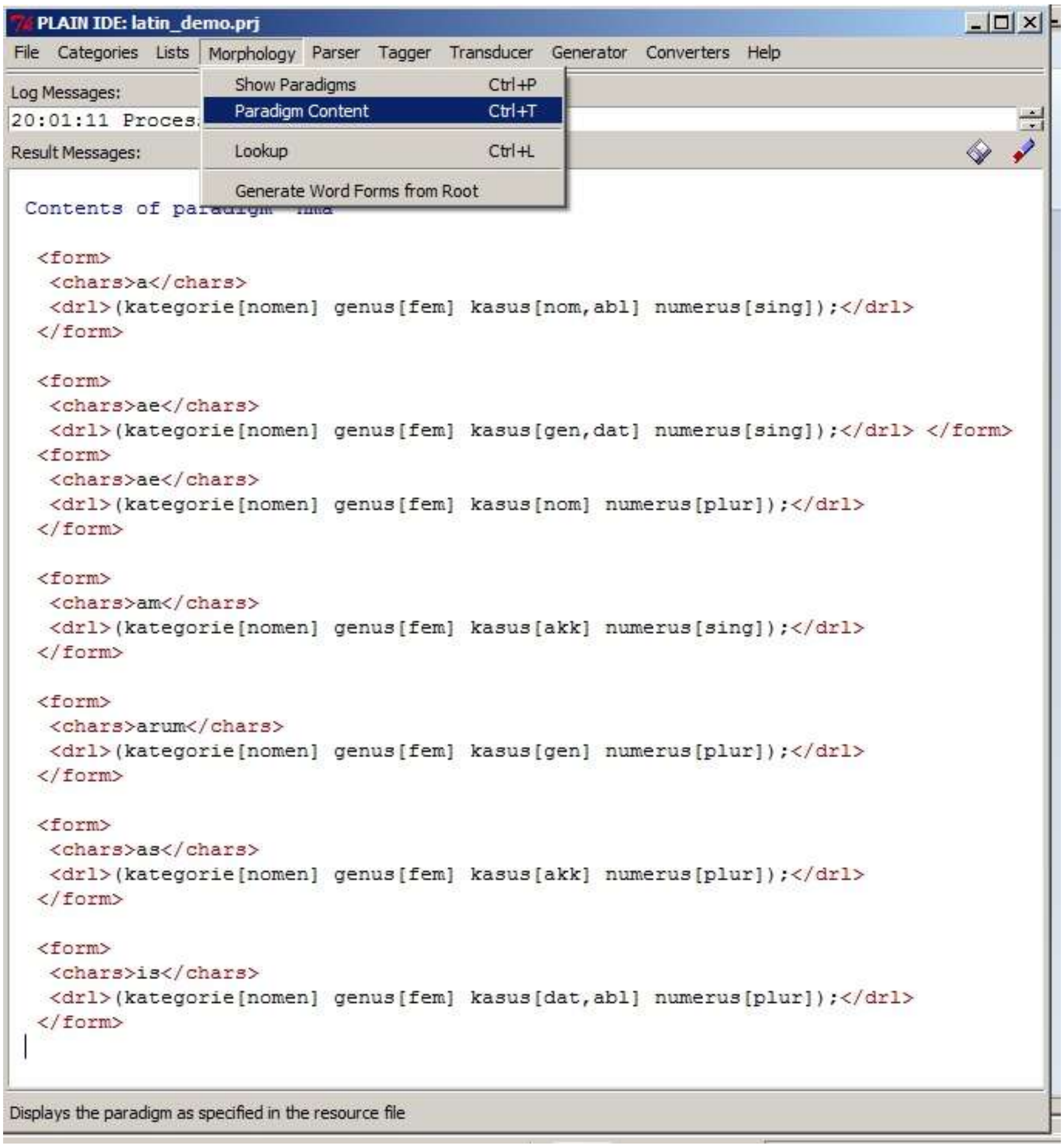
```
Paradigms

   01    'inf-are'
   02    'inf-ere'
   03    'nma'
   04    'nmom'
   05    'nmon'
   06    'perf-i'
   07    'perf-vi'
   08    'pres-a'
   09    'pres-e'
   10    'pres-i'
   11    'ptz-sum'
   12    'ptz-tum'
   13    'ptz-um'
   14    'start' (root section)
```

Fig. 6 Command *Show Paradigms* in the *Morphology* menu

The command *Show Paradigms* displays the IDs of all paradigms stored in the database. A *Paradigm* is a set of linguistic strings in a paradigmatic relationship, e.g. the endings of a particular inflection class. Between several paradigms there may be a syntagmatic relationship. For example, the stems of the Latin words constitute the paradigm *start.* Have a look at the file *.../lingware/latin_demo/load/latin_forms.xml* Here, each stem is linked up with an ending paradigm via a syntagmatic relationship (formalized by the xml-element "contin"). You will find the ending paradigms in the file *.../lingware/latin_demo /load/latin_endings.xml.*

5.2 Paradigm Content

This command just shows the description of a paradigm as it is stored in the resource file. You are prompted for the name of the paradigm to display.


PLAIN IDE: latin_demo.prj
File Categories Lists Morphology Parser Tagger Transducer Generator Converters Help
Show Paradigms Ctrl+P
Paradigm Content Ctrl+T
Lookup Ctrl+L
Generate Word Forms from Root
Log Messages:
Result Messages:

```
20:01:11 Proces
```

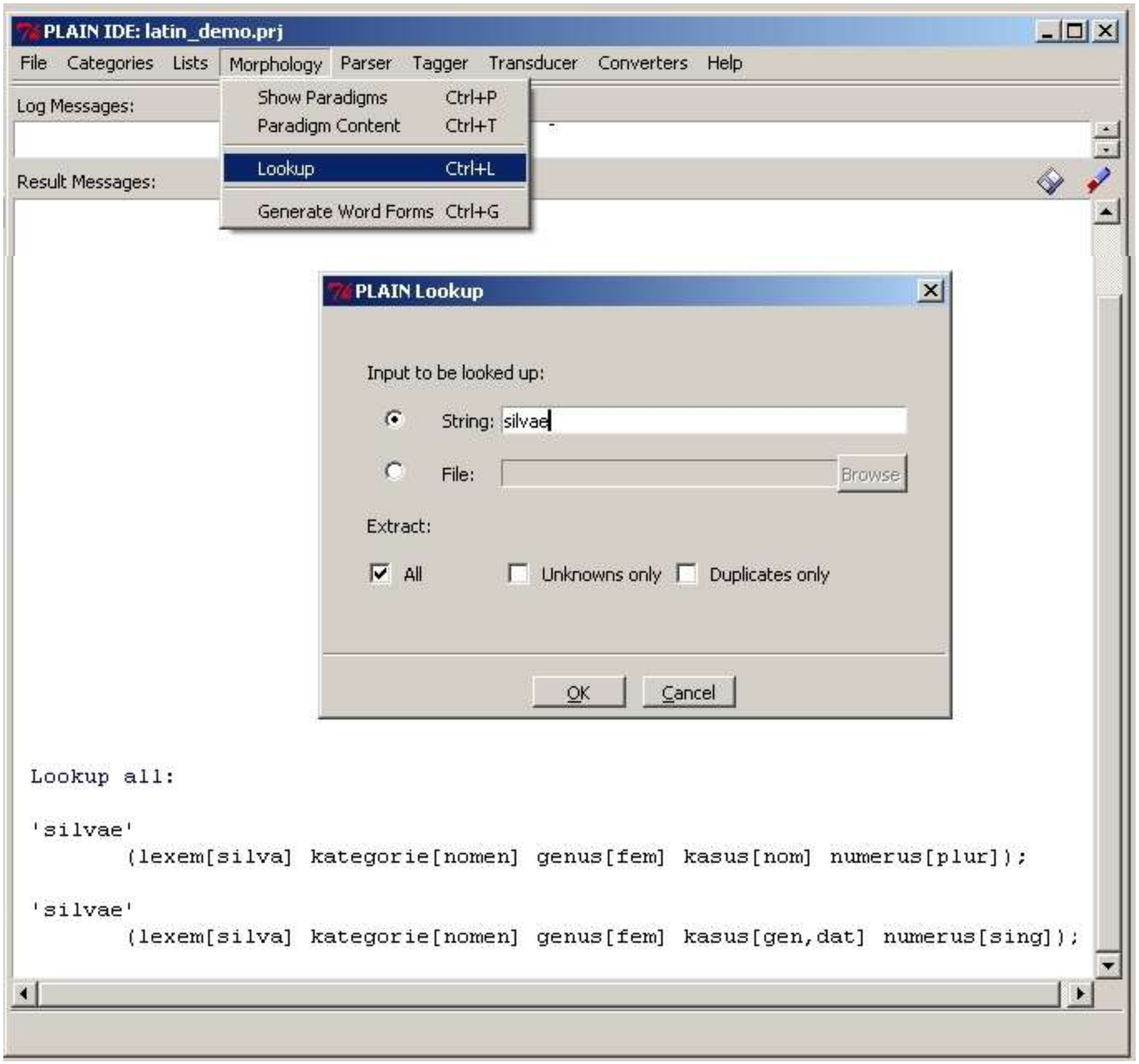
```
Contents of pa

 <form>
  <chars>a</chars>
  <drl>(kategorie[nomen] genus[fem] kasus[nom,abl] numerus[sing]);</drl>
 </form>

 <form>
  <chars>ae</chars>
  <drl>(kategorie[nomen] genus[fem] kasus[gen,dat] numerus[sing]);</drl> </form>
 <form>
  <chars>ae</chars>
  <drl>(kategorie[nomen] genus[fem] kasus[nom] numerus[plur]);</drl>
 </form>

 <form>
  <chars>am</chars>
  <drl>(kategorie[nomen] genus[fem] kasus[akk] numerus[sing]);</drl>
 </form>

 <form>
  <chars>arum</chars>
  <drl>(kategorie[nomen] genus[fem] kasus[gen] numerus[plur]);</drl>
 </form>

 <form>
  <chars>as</chars>
  <drl>(kategorie[nomen] genus[fem] kasus[akk] numerus[plur]);</drl>
 </form>

 <form>
  <chars>is</chars>
  <drl>(kategorie[nomen] genus[fem] kasus[dat,abl] numerus[plur]);</drl>
 </form>
```

Displays the paradigm as specified in the resource file

Fig. 7   Content of the paradigm *nma* (Latin nouns, masculine, a-declension)

5.3 Lookup

Now we are prepared to retrieve words from the lexicon. Select the command *Lookup* from the *Morphology* menu and enter a word in the pop-up window, e.g. *silvae*:



```
Lookup all:

'silvae'
        (lexem[silva] kategorie[nomen] genus[fem] kasus[nom] numerus[plur]);

'silvae'
        (lexem[silva] kategorie[nomen] genus[fem] kasus[gen,dat] numerus[sing]);
```

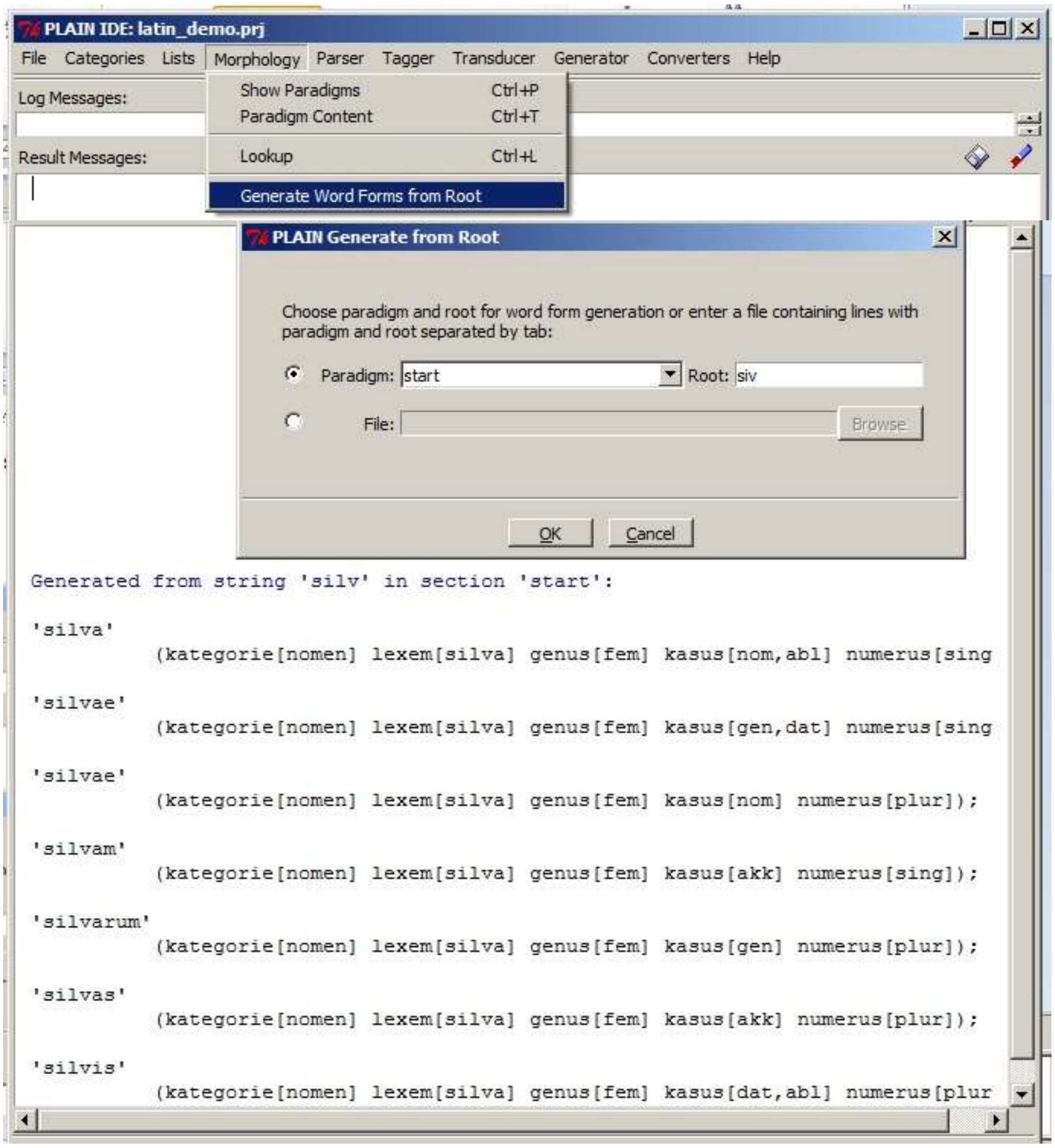
Fig. 8 Looking up the word *silvae* in the Latin lexicon

The word *silvae* is homonymous according to the paradigm *nma.* That is why there are two results. The output illustrates the DRL ("Dependency Representation Language") notation. A DRL categorization consists of a set of attributes, each attribute is followed by a set of values surrounded by square brackets. Please, excuse the fact that we used the terminology of German school teachers in the Latin fragment..

Why don 't you look up other Latin words? Or run the test file! Select *Input to be looked up by File* and choose the file *.../lingware/latin_demo/test/latin_lookup.txt*

5.4 Generate Word Forms from Root

Paradigmatic relationships (within a paradigm) and syntagmatic relationships (between paradigms) in the PLAIN lexicon form a finite state transition network (FTN) which can be used for generation as well as for lookup.



Choose paradigm and root for word form generation or enter a file containing lines with paradigm and root separated by tab:

(●) Paradigm: start Root: siv

( ) File:

```
Generated from string 'silv' in section 'start':

'silva'
          (kategorie[nomen] lexem[silva] genus[fem] kasus[nom,abl] numerus[sing

'silvae'
          (kategorie[nomen] lexem[silva] genus[fem] kasus[gen,dat] numerus[sing

'silvae'
          (kategorie[nomen] lexem[silva] genus[fem] kasus[nom] numerus[plur]);

'silvam'
          (kategorie[nomen] lexem[silva] genus[fem] kasus[akk] numerus[sing]);

'silvarum'
          (kategorie[nomen] lexem[silva] genus[fem] kasus[gen] numerus[plur]);

'silvas'
          (kategorie[nomen] lexem[silva] genus[fem] kasus[akk] numerus[plur]);

'silvis'
          (kategorie[nomen] lexem[silva] genus[fem] kasus[dat,abl] numerus[plur
```

Fig. 9 Generating the Latin word forms with the stem *silv*

You may try to generate word forms from other Latin stems in the paradigm *start*. Use the command *Paradigm Content* in the *Morphology* menu for paradigm *start* to find out which stems exist in the demo version.

You may also run a test file. Select *File* in the *PLAIN Generate* menu and choose the file *.../lingware/latin_demo/test/latin_gener.txt"*

More sophisticated methods of word form generation are introduced in section 11.1.

5.5 Further Lookup Tests

Open the project *german_demo.prj* via the *File* menu. Have a look at the files listed in *Projects Settings* under *Lexicon Subsets:*

- *.../lingware/german_demo/load/de_demo_adjektive.xml*
- *.../lingware/german_demo/load/de_demo_substantive.xml*
- *.../lingware/german_demo/load/de_demo_verben.xml*
- *.../lingware/german_demo/load/de_lexbase.xml*
- *.../lingware/german_demo/load/de_neue_rechtschreibung.xml*

Select the command *Lookup* from the *Morphology* menu with *Input to be looked up by File* and run the files

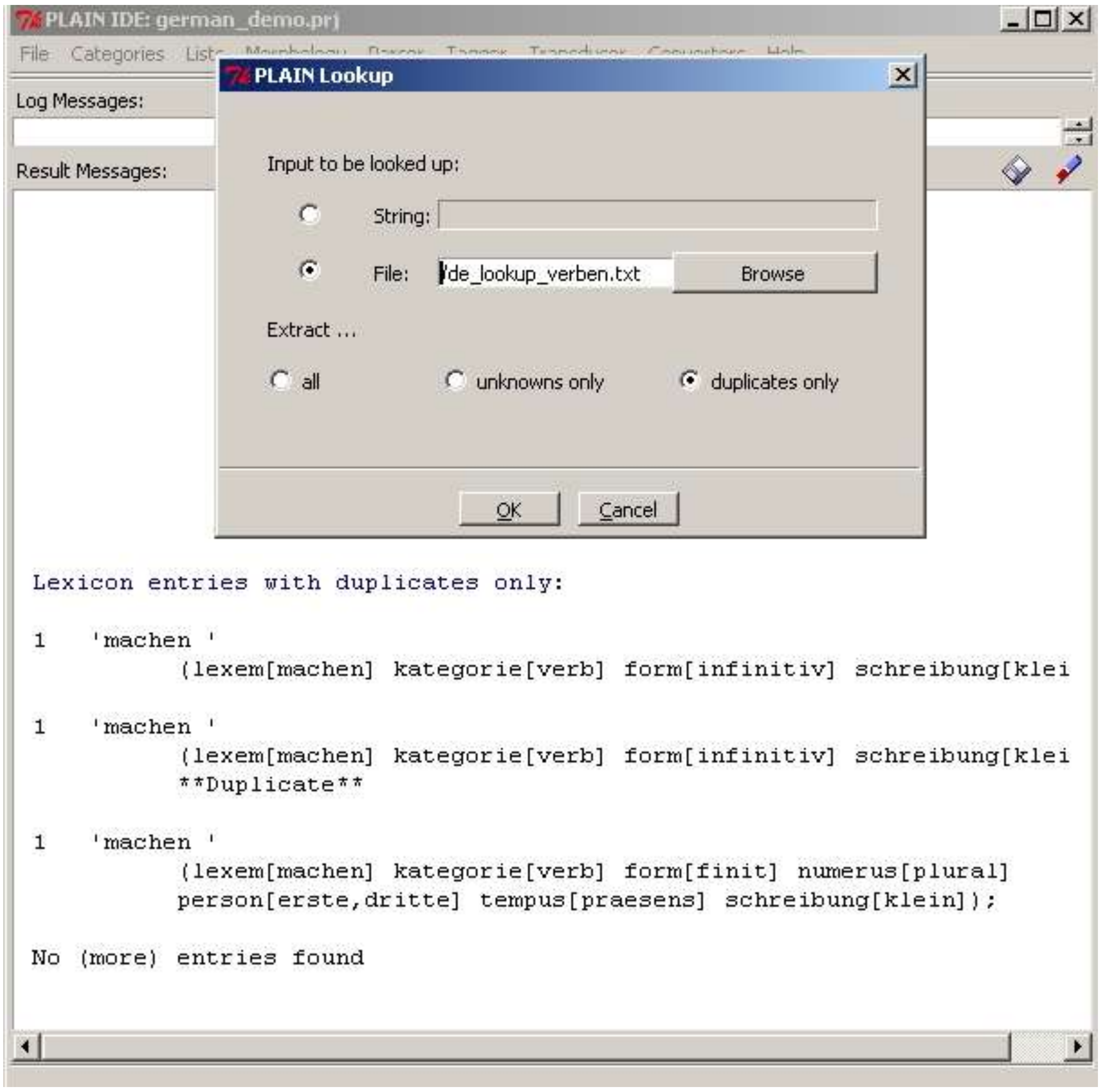
- .../lingware/ *german_demo /test/de*_lookup_adjektive.txt
- .../lingware/ *german_demo /test/*de_lookup_substantive.txt
- .../lingware/ *german_demo /test/*de_lookup_verben.txt

Each word in these test files is an example for another inflectional behavior. And virtually any morphological behavior of German adjectives, substantives and verbs is represented by one example in this collection.

See chapter 12 for the derivation of the *Lexicon Subset* files from cardinal forms.

5.6 Lookup Options

Let us have a quick look at the options of the command *Lookup.*


PLAIN IDE: german_demo.prj
File Categories Lists Morphology Parser Tagger Transducer Converters Help
Log Messages:
Result Messages:
PLAIN Lookup
Input to be looked up:
String:
File: /de_lookup_verben.txt
Browse
Extract ...
all
unknowns only
duplicates only
OK
Cancel

String: [ ] File: [✓]

all [ ] unknowns only [ ] [✓ duplicates only]

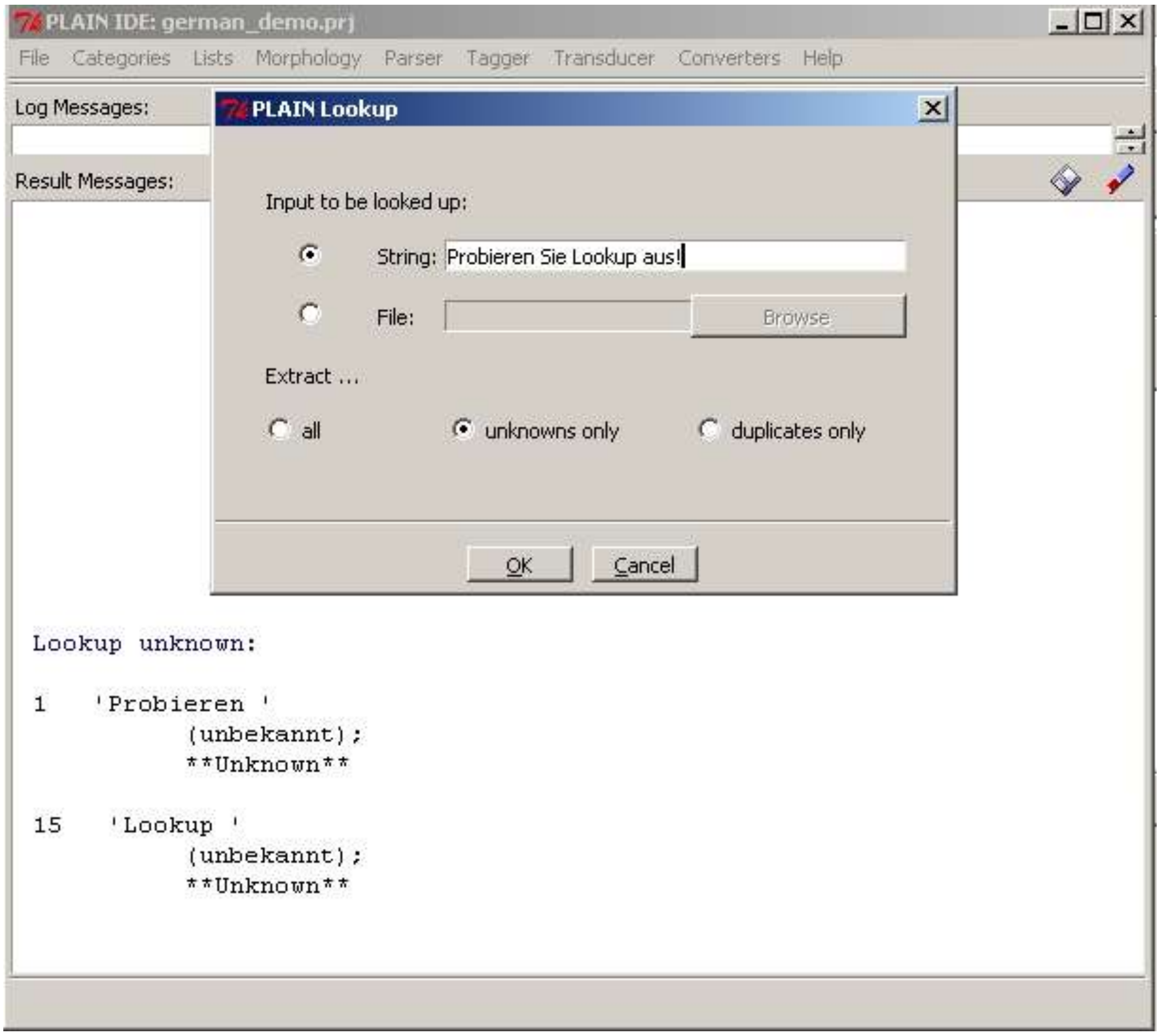
```
Lexicon entries with duplicates only:

1    'machen '
          (lexem[machen] kategorie[verb] form[infinitiv] schreibung[klei

1    'machen '
          (lexem[machen] kategorie[verb] form[infinitiv] schreibung[klei
          **Duplicate**

1    'machen '
          (lexem[machen] kategorie[verb] form[finit] numerus[plural]
          person[erste,dritte] tempus[praesens] schreibung[klein]);

No (more) entries found
```

Fig. 10 Option *Extract duplicates only*

The option *Extract duplicates only* is a tool for avoiding duplicates in the lexicon. Running *Lookup* on a corpus with this option helps to detect mistakes made while drawing up the lexicon. The example in Fig. 10 is due to a mistaken double entry in file *.../lingware/german_demo/load /de_demo_verben.xml:*

```
<form
paradigm="allgemein"><char>mach</char><drl>(lexem[machen])</drl>
<contin paradigm="inf-en"/></form>
<form
paradigm="allgemein"><char>mach</char><drl>(lexem[machen])</drl>
<contin paradigm="inf-en"/></form>
```

Another tool is the option *Extract unknowns only.*


PLAIN IDE: german_demo.prj
File Categories Lists Morphology Parser Tagger Transducer Converters Help
Log Messages:
Result Messages:
PLAIN Lookup
Input to be looked up:
String: Probieren Sie Lookup aus!
File:
Browse
Extract ...
all
unknowns only
duplicates only
OK
Cancel

Input to be looked up: [✓ String]: Probieren Sie Lookup aus!

Extract ... all [✓ unknowns only] duplicates only

```
Lookup unknown:

1    'Probieren '
            (unbekannt);
            **Unknown**

15    'Lookup '
            (unbekannt);
            **Unknown**
```

Fig. 11   Option *Extract unknowns only*

In this case the strings in the input that are NOT retrieved are displayed. Running *Lookup* on a corpus of texts with this option yields vocabulary still missing in the database. This is a prerequisite for bringing the lexicon up to date.

6   The Parser

The *Parser* menu pops up if you press the *Parser* button on the main bar.

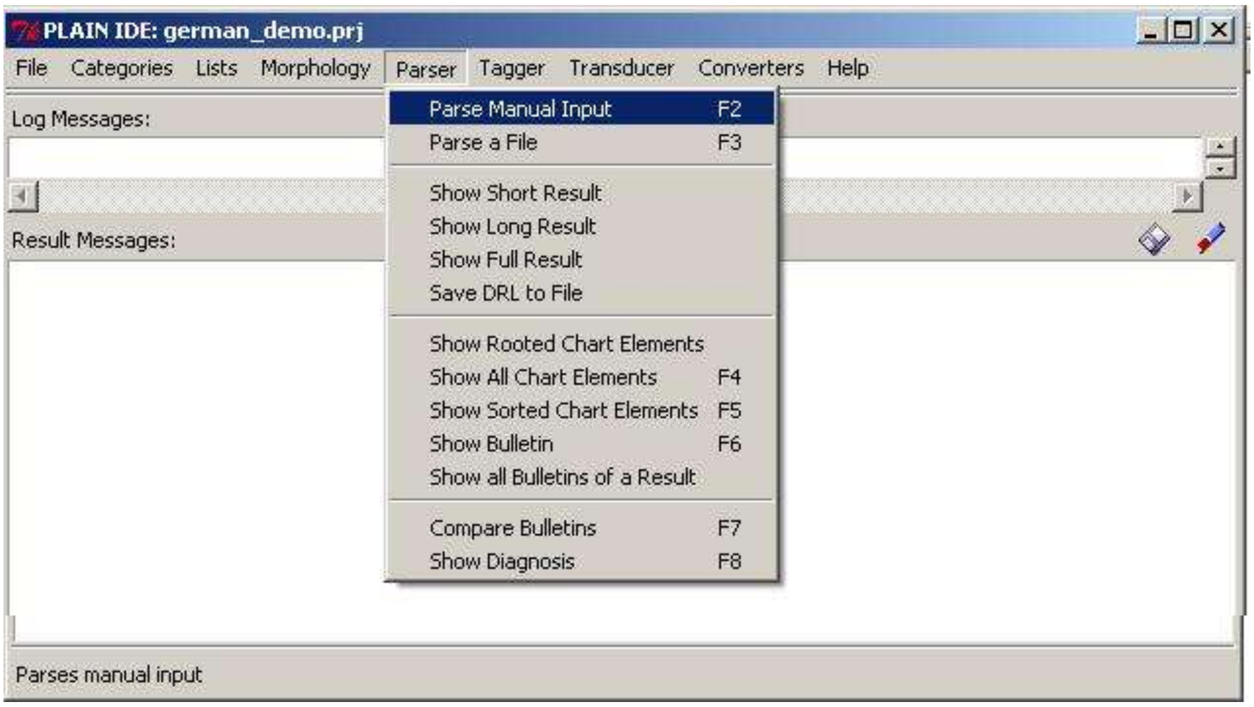


Fig. 12 The *Parser* menu

You can either have PLAIN parse manual input or parse a file.

6.1 Parse Manual Input

If you choose this command then an entry box shows up.

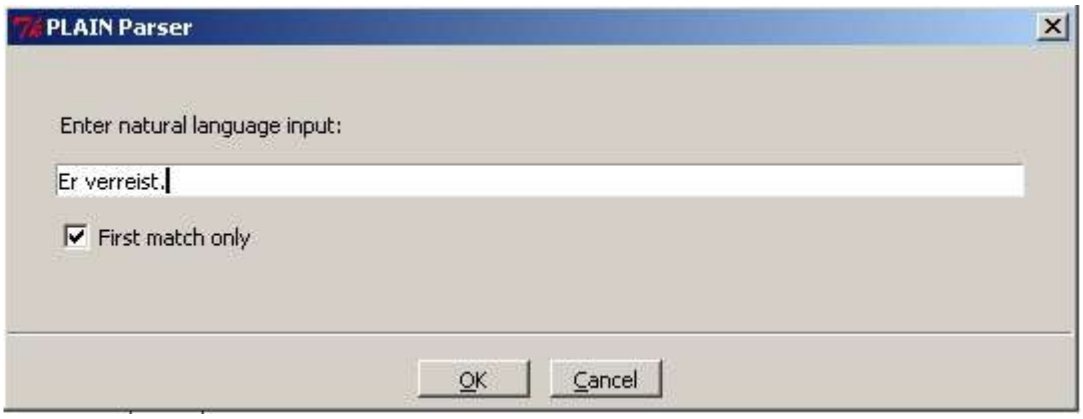


Fig. 13 The input window

If *german_demo* is your opened project then you may enter any input from the following material. Sorry, if you do not speak German. An English demo is to come soon.

Note: Each line must end with a space or a punctuation mark. This is a convention of the actual German data rather than a principle of the PLAIN software. In the actual

German grammar we consider a word delimiter (either a space or a punctuation mark) as a mandatory part of a word. The lexicon is drawn up correspondingly.

Examples illustrating the morpho-syntax, including agreement, compounds, portmanteau morphs:

> verreisen wir
> verreist du
> verreisen sie
> der freundliche Mann
> ein freundlicher Mann
> die Staubecken
> die Staubecke
> das Staubecken
> am Sonntag
> im Buch
> mit Hilfe des Buches

Examples illustrating complements, including sentences, elliptic complement, syntactic resolution of lexical ambiguity, optional or mandatory complements, selectional restrictions, separable prefix, idiomatic expressions, processing of punctuation marks, word order, word order variation:

> Er verreist.
> Verreist ihr?
> Verreise!
> Der Mann gibt dem Kind das Buch.
> Dem Kind gibt der Mann das Buch.
> Das Buch gibt der Mann dem Kind.
> Es gibt keinen Stau. (turn off "first match only" for this example)
> Sie gibt an.
> Der Mann gibt der Frau das Buch ab.
> Der Mann gibt sich freundlich.
> Das Buch gibt dem Mann zu denken.
> Der Frau gibt zu denken, dass er verreist.
> Dass er verreist, steht außer Frage.

Examples illustrating nucleus and raising:

> Der Mann hat dem Kind das Buch abgegeben.
> Der Mann ist verreist.
> Dass die Frau verreist ist, hat dem Mann zu denken gegeben.
> Der Mann ist freundlich.
> Die Frau ist dem Mann überlegen.
> Dass der Mann verreist, ist der Frau unheimlich.

Examples illustrating adjuncts, including selectional features for identifying roles, expected adjuncts, relative clauses:

> Der Mann gibt dem Kind das Buch in der Kirche.
> Er geht in die Kirche.
> Es steht im Buch.

Er steht an der Kirche.
Er verreist am Sonntag.
Die Frau befindet sich in der Kirche.
Die Frau befindet sich. (parsing this example must fail because the expected adjunct is missing)
Er hat dem Mann, welcher verreist, das Buch gegeben.
Das Buch, welches der Mann der Frau gibt, ist spannend.

Examples illustrating discontinuous constituents, including extraposition of complements and adjuncts, coordination of discontinuous constituents:

Das Buch hat er dem Mann gegeben.
Dem Mann hat er Geld gegeben.
Am Sonntag hat er dem Mann das Geld gegeben.
Dem Mann das Geld hat er genommen.
Er hat dem Mann das Buch gegeben, welcher verreist.
Entweder der Frau das Buch oder dem Kind das Geld hat er gegeben.
Das Geld hat er genommen und das Buch hat er gegeben.

Examples illustrating coordination including identical roles in the conjuncts, coordination of complements and adjuncts, coordination and nucleus, sentence coordination:

Der Mann gibt dem Kind das Buch und der Frau das Geld.
Er hat dem Kind das Buch und das Geld gegeben.
Der Mann und das Kind verreisen.
Er hat entweder der Frau oder dem Kind das Geld gegeben.
Er hat der Frau entweder das Buch oder das Geld gegeben.
Er hat entweder der Frau das Buch oder dem Kind das Geld gegeben.
Der Mann gibt dem Kind das Buch am Montag und das Geld am Dienstag.
Er gibt entweder der Frau das Buch am Sonntag oder dem Kind das Geld am Montag.
Hat er der Frau ein Buch gegeben und dem Mann das Geld genommen?

Examples illustrating ellipsis, including various elliptic complements in coordinated constructions:

Er gibt das Geld und nimmt das Buch.
Er gibt und nimmt Geld.
Gibt und nimmt er Geld?
Er gibt und sie nimmt das Geld.
Er gibt und nimmt dem Mann das Geld.
Er gibt dem Mann das Buch und verreist.
Entweder der Mann gibt oder das Kind nimmt das Geld.

Examples illustrating so-called "ophans", i.e. dependents of elliptic heads:

Der glückliche Mann nimmt das Geld und der freundliche gibt dem Kind das Buch.
Der glückliche gibt dem Kind das Geld und der freundliche das Buch.

The resources of the parser are so-called *templates* and so-called *synframes*. Templates describe the construction of a syntactic unit by joining a head constituent and a dependent constituent. Templates incorporate so to say the rules of generative grammar. Synframes associate templates with lexical items. Synframes incorporate so to say the strict subcategorizations of generative grammar. In terms of dependency grammar they define the valency of words.

The above examples are parsed due to the contents of the files *.../lingware/german_demo/load/de_templates.xml* and *.../lingware/german_demo/load /de_synframes.xml.* Perhaps you want to have a look into these files with your editor. Instructions to draw up such resources can be found in "The Linguist's Guide to PLAIN". The files with *Templates* and *Synframes* must be specified in the appropriate entry boxes of *New Project* or *Project Settings.*

6.2 Parse a File

The above examples are also included in the file *.../lingware/german_demo/test/de_parser.txt* Choose the command *Parse a File* and specify this file as input. In this way you save the labor of typing in examples.

6.3 Show Result

There are three formats of parser output: *Show Short Result, Show Long Result, Show Full Result.* If you enter "Er verreist." (*He goes on a journey*) as in Fig. 13 then a "short" result is displayed:

```
(illokution:  aussage'
   (proposition:  verreisen
      (subjekt:  anaphor_sgm')));
```

This format presents an overview of the analysis showing just the dependency structure and the syntagmatic roles and lexical meanings. (This is not proper DRL!)

If you enter the command *Show Long Result* you should get:

```
(rolle[illokution] lexem[aussage'] kategorie[satz] aeusserung[+] wortlaut[.]
schreibung[klein]
    (L rolle[proposition] lexem[verreisen] kategorie[verb] form[finit]
    modus[indikativ] numerus[singular] person[dritte]
    stellungstyp[verb_zweit] tempus[praesens] wortlaut[verreist]
    schreibung[klein] randstellung[+]
        (L rolle[subjekt] lexem[anaphor_sgm'] kategorie[nomen]
        determination[+] genus[maskulin] kasus[nominativ]
        numerus[singular,C] person[dritte,C]
        pronomen[personal] wortlaut[Er ] schreibung[gross]
        s_position[1])));
```

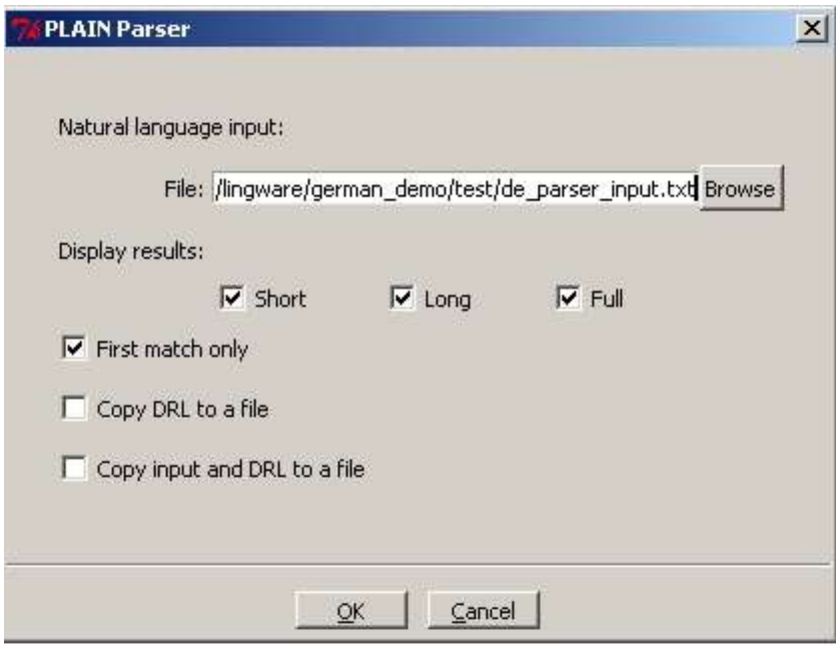
This is DRL format, the formalism of Dependency Unification Grammar (DUG). Each term ("node") in the dependency tree consists of a set of attributes and value. The parser works by "unifying" all these features of all items. Some attributes represent content, some grammatical features, some word order information. If you are interested in a subset of attributes only, you may apply *Attribute Filtering* (see section 8.2).

If you enter the command *Show Full Result* the result is displayed in the following format:

```
   12   '.'
 (rolle[illokution] lexem[aussage'] kategorie[satz] aeusserung[+] wortlaut[.]
schreibung[klein]
    4   'verreist'
 (L rolle[proposition] lexem[verreisen] kategorie[verb] form[finit]
 modus[indikativ] numerus[singular] person[dritte] stellungstyp[verb_zweit]
tempus[praesens] wortlaut[verreist] schreibung[klein] randstellung[+]
    1   'Er '
 (L rolle[subjekt] lexem[anaphor_sgm'] kategorie[nomen] determination[+]
genus[maskulin] kasus[nominativ] numerus[singular,C] person[dritte,C]
pronomen[personal] wortlaut[Er ] schreibung[gross] s_position[1])));
```

Try out the different output formats after parsing other example sentences.

If you *Parse a File* you can choose the type of display by means of the check boxes of the following pop-up window. You can activate more than one check box in order to get more than one output.


PLAIN Parser

Natural language input:

File: /lingware/german_demo/test/de_parser_input.txt

Display results: [✓ Short] [✓ Long] [✓ Full]

[✓ First match only]

☐ Copy DRL to a file

☐ Copy input and DRL to a file

Fig. 14 *Display results* in the *Short, Long*, and *Full* format

6.4 Rest of the parser menu

After *Parsing Manual Input* any of the other commands on the menu (except *Parse a File*) can be executed. You might just try out the list and see what happens, although some functions require more knowledge about PLAIN than can be provided here. Please consult "The Linguist's Guide to PLAIN" and the "Technical Guide to PLAIN".

7 Tagger

PLAIN is devoted to full-fledged syntactic analysis and semantic processing. Tagging input strings is just a by-product. Nevertheless some users might want to use PLAIN as a tagger, especially if a particular PLAIN project has sizeable resources at its disposal.

There are two options.

7.1 Lexical Lookup Only

In the case of this option tagging is done just on the basis of the morphological lexicon. The device is the same as the *Lookup* command in the *Morphology* menu. The difference is that the *Tagger* function creates an output file in xml-format while the *Lookup* function creates a perspicuous display on the screen.

Issue the command *Lexical Lookup Only.* You will be prompted for an input and an output file. For the time being, you may specify the file *.../lingware/german_demo/test/de_tagger.txt* as input. It contains more or less the example sentences of the parser. You have to specify an output file too. Eventually this file will contain an xml-representation of the input strings (within <char> and </char> tags and their classification (within <drl> and </drl> tags.

Activate the check box *First match only* and repeat the test. Now the output file contains only the first <char> <drl> of alternative classifications.

For comparison issue the *Lookup* command in the *Morphology* menu and specify the same file as input.

Try out subsets of tags by means of the command *Attribute filtering* in the menu *Categories* (cf. section 8.2). You can reduce ambiguity and hence alternative classifications by moving attributes from the visible to the hidden part.

7.2 Parse the Input

If you choose this option the input is parsed before tags are assigned to words. Ambiguity is removed and higher level attributes like syntagmatic roles can be retrieved. If no complete analysis of a sentence is achieved than partial parsing is applied.

Issue the command *Parse the Input.* You will be prompted for an input and an output file. For the time being you may specify the file *.../lingware/german_demo/test/de_tagger.txt* as input. Have a look at the output and compare it with the output of the *Lexical Lookup Only* function.

8 Categories

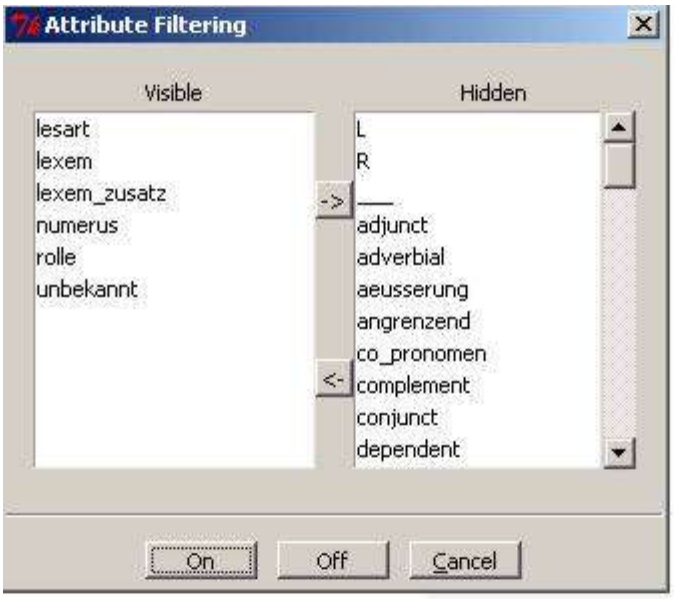
The previous examples should have given an impression of what categories are like in the Dependency Representation Language (DRL). A category is any desired set of attributes and values. Since attributes may be processed differently, they must be defined for each project in the *Category Definition* file. Each attribute must be associated with a type. The details are explained in "The Linguist's Guide to PLAIN". Here, we just have a look at two tools related to categories. Press *Categories* on the main menu bar!

8.1 Show Attributes

Choose the *Show Attributes* command. The set of attributes defined for the current project is shown, preceded by a code for the attribute type. This command makes sense if you want to check the database for the correct representation of categories.

8.2 Attribute Filtering

PLAIN uses complex categories. But the user can vary the DRL representation to a great extent by using the *Attribute Filtering* command. One can choose attributes and move them in the appearing window from *Visible* to *Hidden* and vice versa. If *german_demo* is the current project then move attributes until you have about the following situation. Then click *on.*



Visible	Hidden
lesart	L
lexem	R
lexem_zusatz	___
numerus	adjunct
rolle	adverbial
unbekannt	aeusserung
	angrenzend
	co_pronomen
	complement
	conjunct
	dependent

Fig. 15 *Attribute Filtering*

You can test *Attribute Filtering* now with the command *Show Long Result* in the *Parser* menu. Parse a sentence, e.g. "Er verreist."

If you activate *Attribute Filtering* with the above setting then the parser yields the following result:

```
(rolle[illokution] lexem[aussage']
   (rolle[proposition] lexem[verreisen] numerus[singular]
      (rolle[subjekt] lexem[anaphor_sgm'] numerus[singular,C])));
```

Compare this output with the display of all attributes in section 6.3 Move other attributes from *Visible* to *Hidden* and vice versa and try the *Show Long Result* command of the *Parser* again.

Try out *Attribute Filtering* with other output, for example with the *Lookup* function of the *Morphology* menu or with the *Tagger* output.

If all attributes should be visible again then press the *Off* button in Fig. 15. .

9 Lists

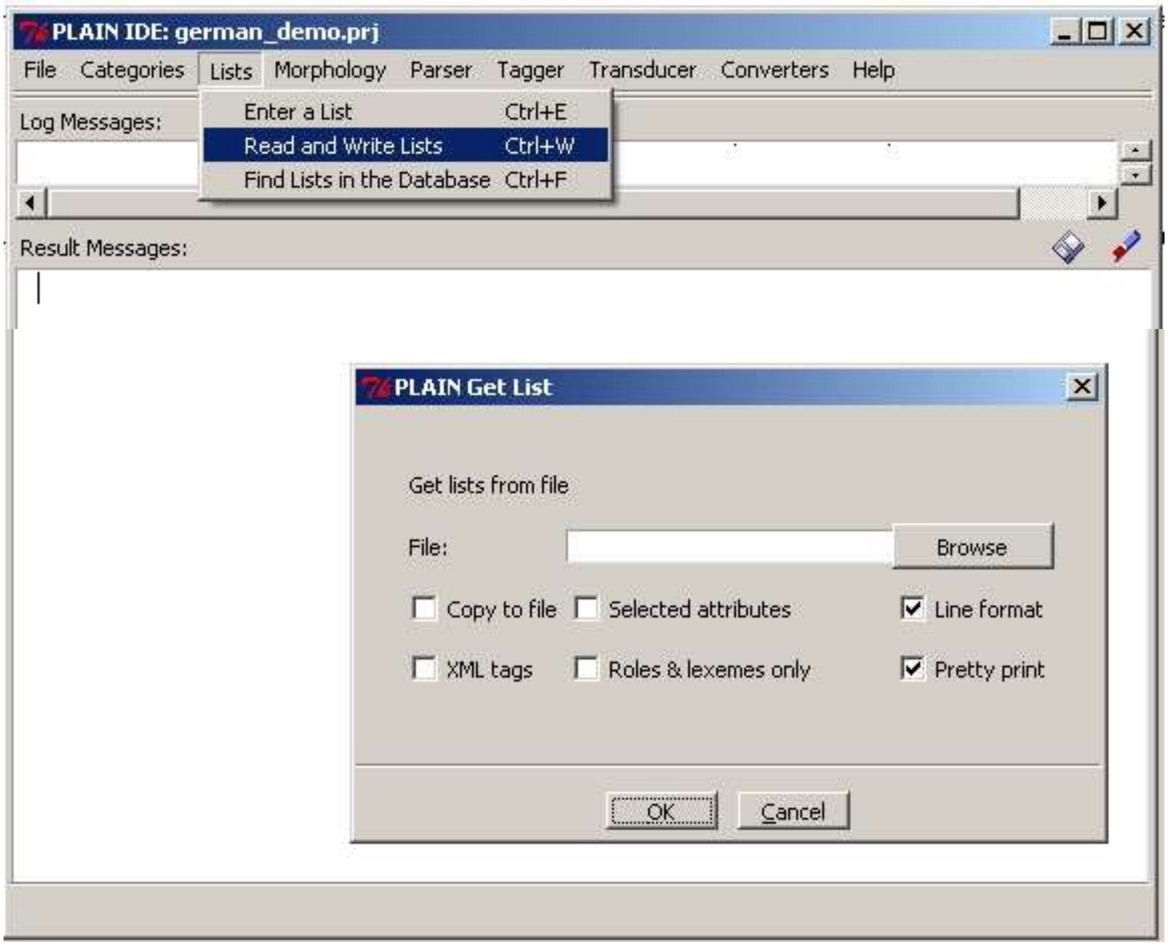
The *List* menu offers a few tools to test lists, to change the list format while reading and writing lists, and to find particular lists in the database.

9.1 Enter a List

Type or copy and paste any DRL expression in the input box.

9.2 Read and Write Lists

Once more *Parse a File* in the *Parser* menu, but activate the check box *Copy DRL to a file.* You will be prompted for the creation of an output file. Specify this file in the *File* box of the *Get List* pop-up window of the *Read and Write Lists* command.. Activate the check boxes at will and then press OK.


PLAIN IDE: german_demo.prj
File Categories Lists Morphology Parser Tagger Transducer Converters Help
Enter a List Ctrl+E
Read and Write Lists Ctrl+W
Find Lists in the Database Ctrl+F
Log Messages:
Result Messages:
PLAIN Get List
Get lists from file
File:
Browse
OK
Cancel

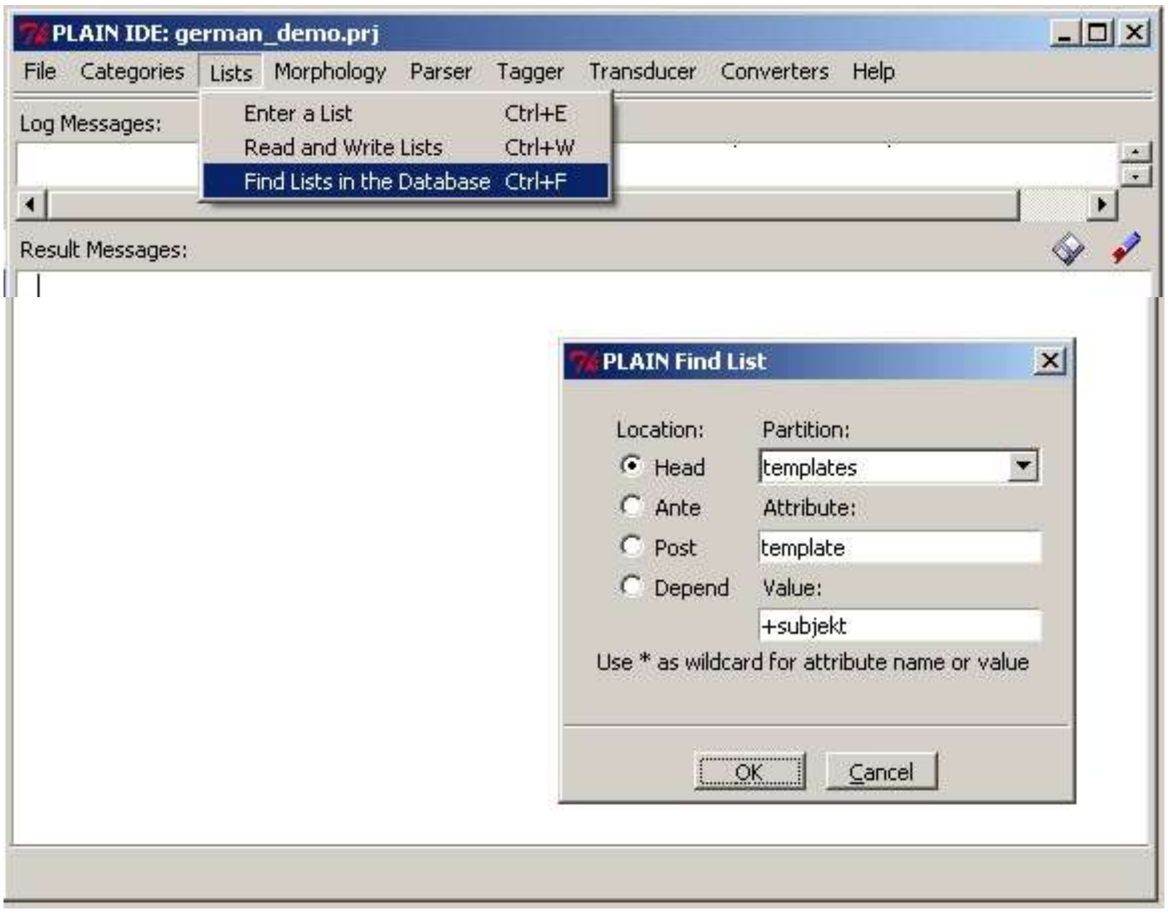
Copy to file: ☐ Selected attributes: ☐ Line format: ☑
XML tags: ☐ Roles & lexemes only: ☐ Pretty print: ☑

Fig. 16 Reviewing and displaying DRL data

9.3 Find Lists in the Database

DRL lists are stored in the database in a special way to facilitate retrieval. On the one hand one can divide the data into so-called partitions. For example the templates used by the parser or the theorems used by the transducer are arranged in different partitions. On the other hand each list may be indexed by particular attributes it contains in a certain position.

Choose the *Find Lists in the Database* command and complete the pop-up window as shown. You will get all subject templates of the *german_demo* project.


PLAIN IDE: german_demo.prj
File Categories Lists Morphology Parser Tagger Transducer Converters Help
Enter a List Ctrl+E
Read and Write Lists Ctrl+W
Find Lists in the Database Ctrl+F
Log Messages:
Result Messages:
PLAIN Find List
OK
Cancel

Location: [✓ Head] Ante Post Depend

Partition: templates

Attribute: template

Value: +subjekt

Use * as wildcard for attribute name or value

Fig. 17   Find all "+subjekt" templates in the database

Have a look at the file *.../lingware/partitions/partions.xml* in your editor. In this file the methods of indexing are defined for each partition.

10 Transducer

Dependency Unification Grammar (DUG) claims that linguistic actions like paraphrasing, translating, inferring, summarizing should be modeled by operations on DRL expressions according to rules. The original DRL expressions are derived from natural language expressions by the *Parser*. New DRL expressions result from operations of the *Transducer.* These DRL expressions must eventually be turned into natural language strings again by the *Generator*.

The rules to be used by the *Transducer* must be drawn up and stored in files which must be specified in the entry box *Instances and Rules* of the *New Project* or *Project Settings* command. For convenience we added the following transducer files to the project *german_demo*:

- *.../lingware/transducer/load/de_passive_rule.xml*
- *.../lingware/transducer/load/de_en_ translation_rules.xml*
- *.../lingware/transducer/load/theorems.xml*

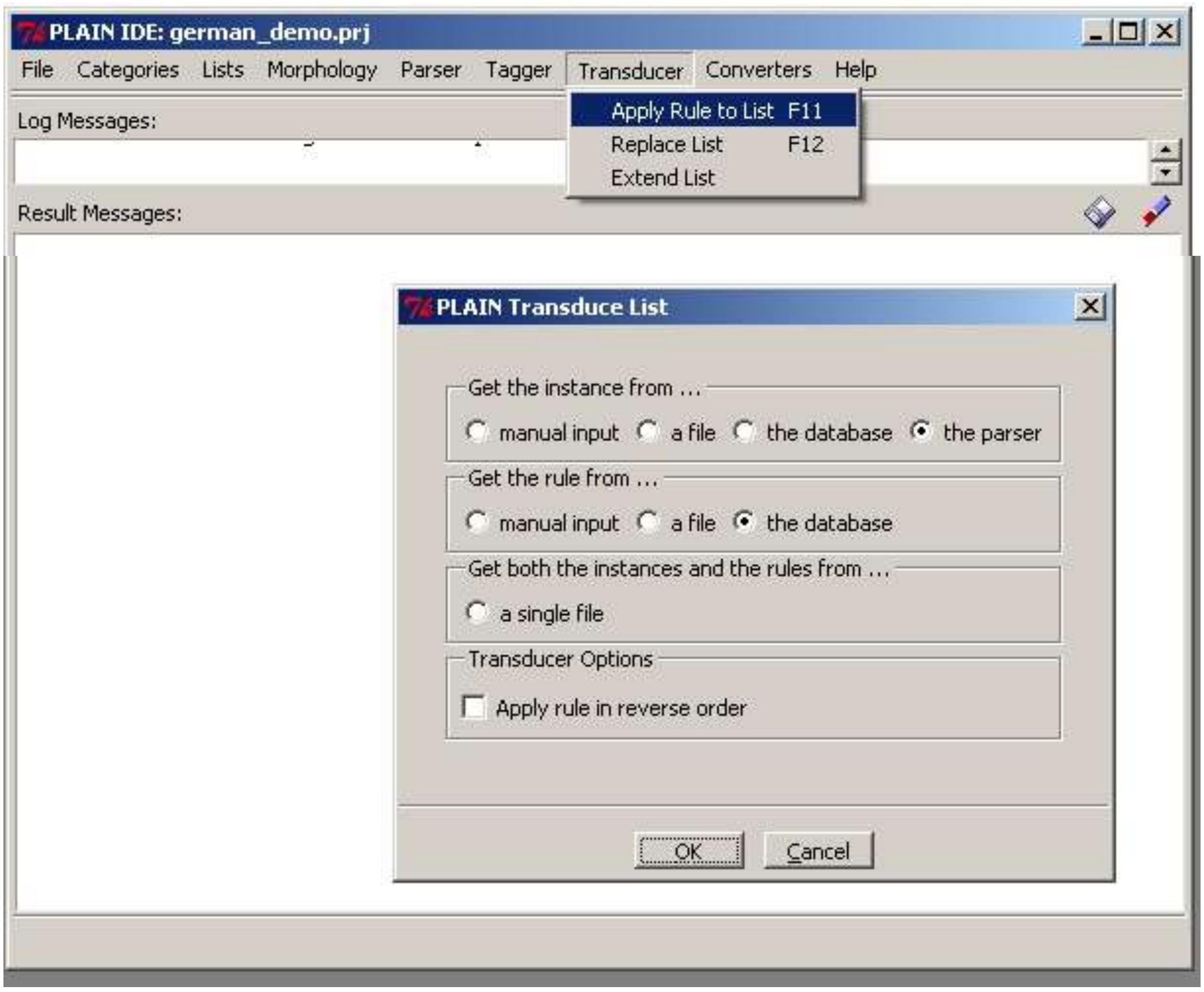
10.1 Apply Rule to List

The function *Apply Rule to List* is designed to draw up and test single rules step by step. Let us use this function for paraphrasing a German sentence in the active voice by a sentence in the passive voice.

First we must parse a sentence with a direct object in order to get its DRL expression. Execute *Parse Manual Input* on the *Parser* menu. Enter for example "Der Mann gibt dem Kind ein Buch." (The man gives a book to the child.) Execute *Attribute Filtering* of the *Category* menu. The following attributes should be visible: "lesart", "lexem", "lexem_zusatz", "numerus", "rolle". Enter *Show Long result.* The following parser result should show up:

```
(rolle[illokution] lexem[aussage']
    (rolle[proposition] lexem[geben] lesart[transferieren]
     numerus[singular]
        (rolle[subjekt] lexem[mann] numerus[singular,C]
            (rolle[determination] lexem[definit'] numerus[singular,C]))
        (rolle[dativ_objekt] lexem[kind] numerus[singular]
            (rolle[determination] lexem[definit'] numerus[singular,C]))
        (rolle[trans_objekt] lexem[buch] numerus[singular]
            (rolle[determination] lexem[ein] numerus[singular,C]))));
```

Now choose *Apply Rule to List* in the *Transducer* menu.


PLAIN IDE: german_demo.prj

File Categories Lists Morphology Parser Tagger Transducer Converters Help

Transducer: Apply Rule to List F11 · Replace List F12 · Extend List

Log Messages:

Result Messages:

PLAIN Transduce List

Get the instance from ...
( ) manual input ( ) a file ( ) the database [✓ the parser]

Get the rule from ...
( ) manual input ( ) a file [✓ the database]

Get both the instances and the rules from ...
( ) a single file

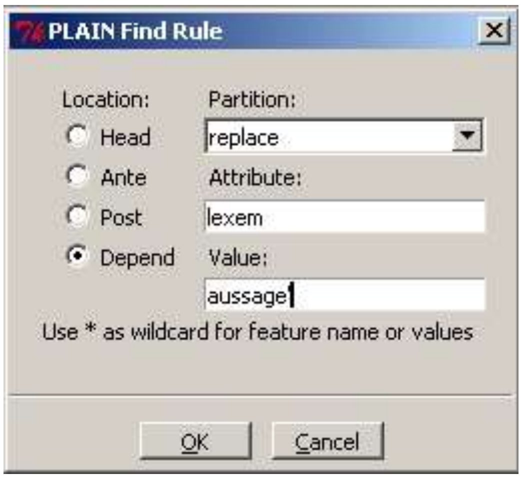
Transducer Options
[ ] Apply rule in reverse order

OK Cancel

Fig. 18 The *Apply Rule to List* pop-up window.

Activate the radio buttons *Get the instance from the parser* and *Get the rule from the database.* Press OK.

The option *Get the rule from the database* invokes the *Find Rule* function. Please enter "replace" in the *partition* box, "lexem" in the *Attribute* box, "aussage'" in the *Value box* (please note the hyphen!) and activate the radio button *Depend.* Press OK.


PLAIN Find Rule

Location: ( ) Head ( ) Ante ( ) Post [✓ Depend]

Partition: replace

Attribute: lexem

Value: aussage

Use * as wildcard for feature name or values

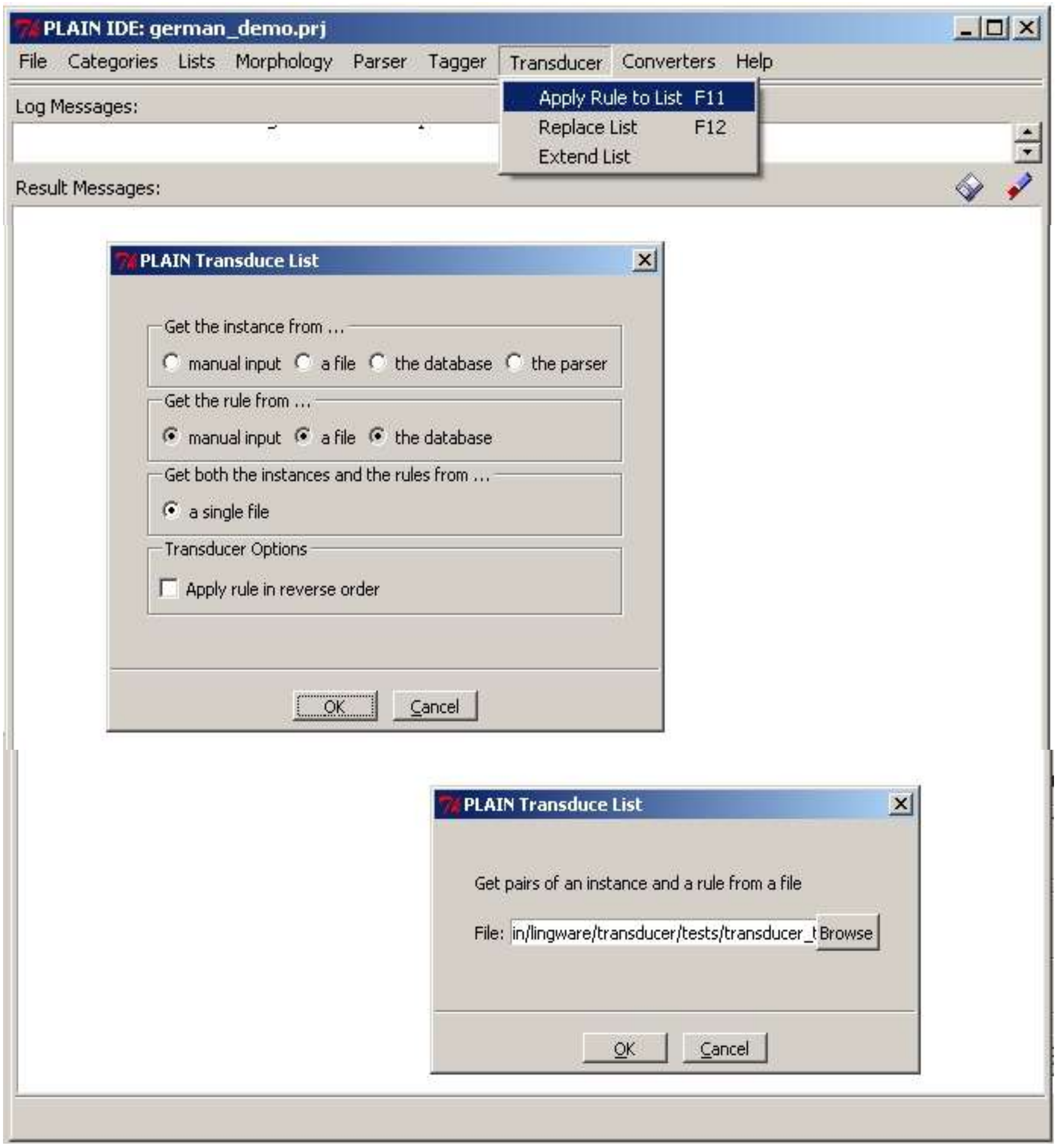
OK Cancel

Fig. 19 Find the rule for passive voice in the database

The replacement rule for active versus passive voice paraphrasing is displayed on the screen. Press OK in the small pop-up window. The result is the DRL expression corresponding to the sentence "Ein Buch wird von dem Mann dem Kind gegeben." (A book is given by the man to the child.)

```
List after replacement:

 (rolle[illokution] lexem[aussage']
    (rolle[proposition] lexem[werden] lesart[passiv] numerus[singular]
       (rolle[subjekt] lexem[buch] numerus[singular]
          (rolle[determination] lexem[ein] numerus[singular,C]))
       (rolle[praedikativ] kategorie[verb] lexem[geben]
       lesart[transferieren]
          (rolle[agens] lexem[von] kategorie[praeposition]
             (rolle[komplement] lexem[mann] numerus[singular,C]
                (rolle[determination] lexem[definit']
                numerus[singular,C])))
          (rolle[dativ_objekt] lexem[kind] numerus[singular]
             (rolle[determination] lexem[definit']
             numerus[singular,C]))))));
```

- You have to consult "The Linguist's Guide to PLAIN" for information about the mechanism of replacement rules in PLAIN.
-
- If you want to study the various possibilities of replacement then you may chose the option *Get both the instances and the rules from a single file* and specify the file *.../lingware/transducer/ test/trasducer_test.txt* as input.



Get the instance from ...
( ) manual input ( ) a file ( ) the database ( ) the parser

Get the rule from ...
(•) manual input (•) a file (•) the database

Get both the instances and the rules from ...
(•) a single file

Transducer Options
☐ Apply rule in reverse order

Get pairs of an instance and a rule from a file
File: in/lingware/transducer/tests/transducer_t

Fig. 20 The *Apply Rule* function applied to a test battery of instances and rules

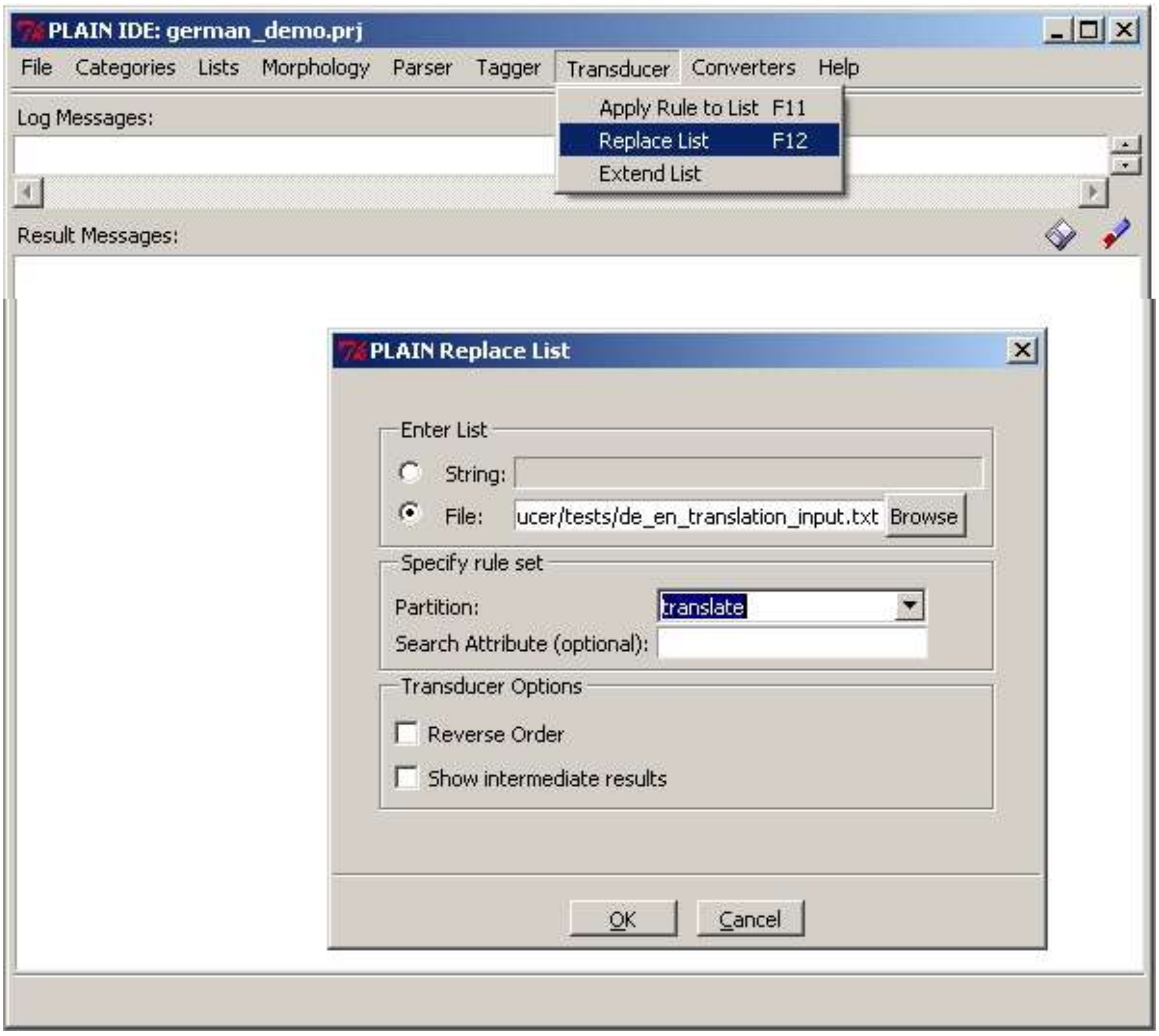
10.2 Replace List

This function applies all available rules repeatedly to an instance and its replacements. Our first example is a translation from German into English and vice versa:

German:	English:
Ich erinnere mich an meine Freundin.	I remember my girl friend.
Du erinnerst mich an meine Freundin.	You remind me of my girl friend.

As can be seen, small differences in the source language can lead to lexically and structurally quite divergent translations.

The DRL expressions of the above German sentences are stored in the file *.../lingware/transducer /test/de_en_translation_input.txt*: Specify this file in the pop-up window, then enter "translate" in the *Partition* box and press OK.



Enter List
- String:
- File: ucer/tests/de_en_translation_input.txt

Specify rule set
- Partition: translate
- Search Attribute (optional):

Transducer Options
- Reverse Order
- Show intermediate results

Fig. 21 Translate German into English

The following output should appear:

```
List to be replaced:

 (rolle[praedikat] lexem[erinnern]
   (rolle[subjekt] lexem[ich])
   (rolle[reflexiv])
   (rolle[praep_obj] lexem[an]
     (lexem[freundin]
       (rolle[dete] lexem[mein]))));

List after replacement:

 (lexem[remember] rolle[praedikat]
   (rolle[trans] lexem[girl_friend]
     (lexem[my] rolle[dete]))
   (lexem[I] rolle[subjekt]));

List to be replaced:

 (rolle[praedikat] lexem[erinnern]
   (rolle[subjekt] lexem[du])
   (rolle[trans] lexem[mich])
   (rolle[praep_obj] lexem[an]
     (lexem[freundin]
       (rolle[dete] lexem[mein]))));

List after replacement:

 (lexem[remind] rolle[praedikat]
   (rolle[trans] lexem[me])
   (rolle[praep_obj] lexem[of]
     (lexem[girl_friend]
       (lexem[my] rolle[dete])))
   (lexem[you] rolle[subjekt]));
```

Have a look at the rules in file *.../lingware/transducer/load/de_en_translation_rules.xml.* The same set of rules allows the translation from English to German as well. Specify the file *.../lingware /transducer/test/de_en_reverse_translation_input.txt* in the *File* box, enter partition *translate,* and activate the *Reverse Order* option. You may also try out *Show intermediate results.*

The second example illustrates the use of PLAIN as a theorem prover by means of replacement rules. Have a look at the contents of file *.../lingware/ transducer/load/theorems.xm* in your editor. Here theorems of propositional logic are formulated as replacement rules. Applying these rules to a proposition results in the terms "l-wahr" (logically true) if the proposition is a tautology, or "l-falsch" (logically false) if the proposition is a contradiction. Otherwise the application results in a disjunctive normal form of the original proposition.

Invoke the *Replace List* command.

Enter the file *.../lingware/transducer/load/theorem_input.xml* in the *File* box and choose "theorems" in the *Partition* box. Then press OK. The following output should appear:

```
List to be replaced:

 (lexem[wenn_dann]
   (rolle[j1] lexem[und]
     (rolle[j] lexem[wenn_dann]
       (rolle[j1] lexem[a])
       (rolle[j2] lexem[b]))
     (rolle[j] lexem[a]))
   (rolle[j2] lexem[b]));

List after replacement:

 (l_wahr);

List to be replaced:

 (lexem[wenn_dann]
   (rolle[j1] lexem[und]
     (rolle[j] lexem[wenn_dann]
       (rolle[j1] lexem[a])
       (rolle[j2] lexem[b]))
     (rolle[j] nicht lexem[a]))
   (rolle[j2] nicht lexem[b]));

List after replacement:

 (lexem[oder_d]
   (rolle[j] lexem[a])
   (rolle[j] nicht lexem[b]));
```

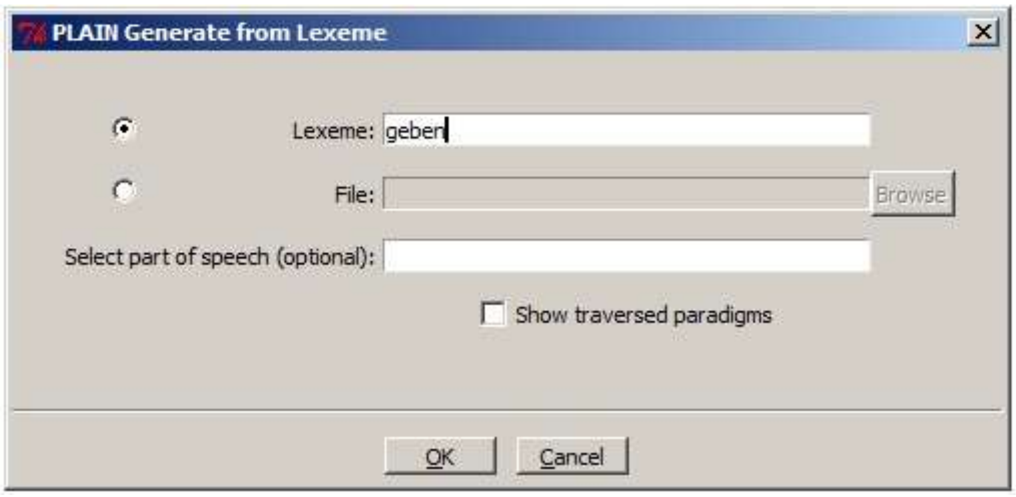
The first list is a DRL formulation of the modus ponens. Since this is a theorem the replacements end with "l_wahr". The second list is a contingent statement. The replacements result in the disjunctive normal form. If you want to study the procedure then activate the check box *Show intermediate results* before you press OK.

11 Generator

While the morphology lookup function receives a word form and yields the attributes appertaining to the substrings of the word, and while the parser reads a sentence and yields the dependency structure of the sentence associated with the unified attributes of each substring, the generator receives a DRL categorization and yields natural language strings.

11.1 Generate Word Forms from Lexeme

Select the *Generator* menu and activate the *Generate Word Forms from Lexeme* Command. The following window appears. Enter 'geben' in the *Lexeme* box.



(● Lexeme) Lexeme: geben

(○ File) File:

Select part of speech (optional):

☐ Show traversed paradigms

Fig. 22 The *Generate Word Forms from Lexeme* window

64 word forms of the verb *geben* and its adjectival and nominal derivations should be displayed together with the respective attributes. They illustrate Ablaut ('geben', 'gab') as well as Umlaut ('gebe', 'gibst', 'gäbe'). Enter *Lexeme* 'geben' again but *Select part of speech* 'adjektiv'. Enter *Lexeme* 'geben' *Select part of speech* 'nomen'. The output is restricted to particular derivations.

Try out other words, e.g. 'sein', 'gehen', 'Buch', 'groß', 'ein'. Activate the *File* radio button and enter

.../lingware/ *german_demo /test/de_genelex.txt.*

This file contains all the lexemes of the German demo version.

If one has a complete list of lexemes of an implementation then one can derive a word forms lexicon with this function, i.e. the set of all possible word forms together with their categorization.

Finally activate the check box *Show traversed paradigms* and enter 'geben' in the *Lexeme* box. Now the word forms are intertwined with the names of the traversed paradigms in angled brackets. For example the following output is created among other:

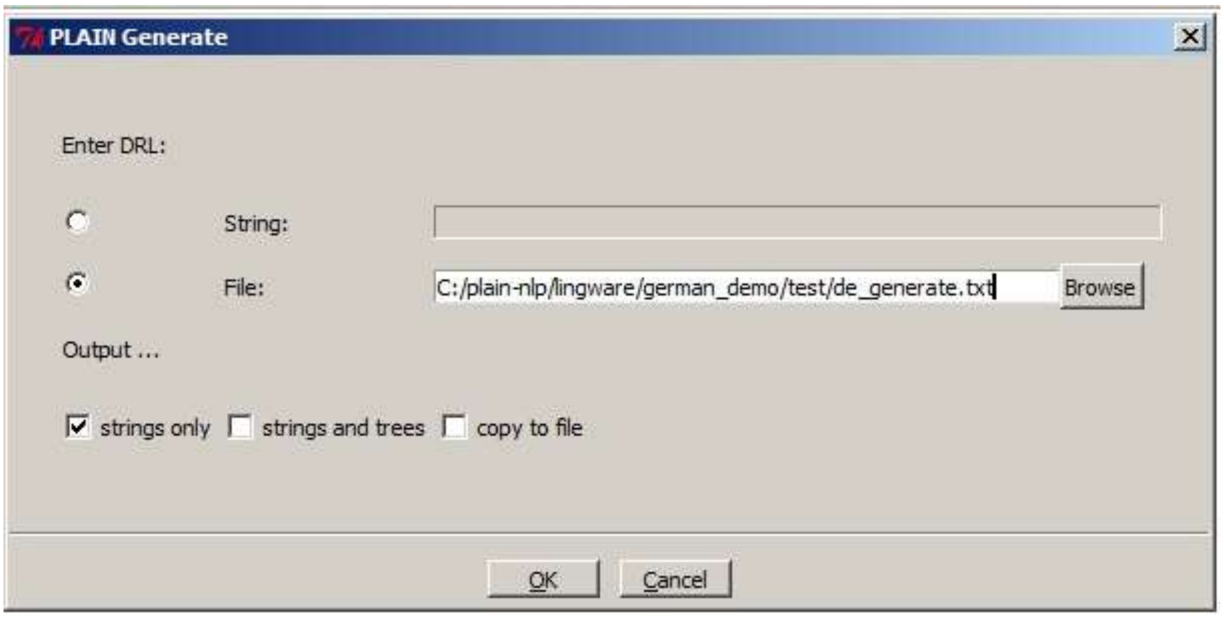
```
'geb<inf-en>end<adj-aktiv><adj><adj-endung>e<leerzeichen>'
        (flexion[stark-schwach,stark] kasus[nominativ,akkusativ]
        numerus[singular] genus[feminin] verwendung[attributiv]
        kategorie[adjektiv] derivation[aktiv_eigenschaft] lexem[geben]
        steigerung[keine]);
```

This is to be read as follows: "There is a word form 'geb-end-e' starting with the stem 'geb' in the paradigm <inf-en> of verbs that form the infinite with 'en'. The following substring 'end' leads to a paradigm <adj-aktiv> of the derivation of the property of the active doer (as opposed to the passive 'gegebene'). Next the generator enters the paradigm <adj-endung> of adjective endings, finding the ending 'e'. The latter gives rise to the attributes nominative or accusative case, singular, feminin. The string ends in the paradigm <leerzeichen> of the empty space as word delimiter. You have to study the resource files in the .../lingware/ *german_demo* /*load* directory in order to understand the whole network of paradigms.

11.2 Generate Natural Language from DRL

If a DRL expression is entered the corresponding natural language string is created. The DRL expression may be underspecified, i.e. only a subset of all applicable attributes occur in the expression. The minimum of attributes in each term (corresponding with a word) is a syntagmatic role and a lexeme. The generator is then trying to retrieve the missing attributes. Since the underspecified expression may be ambiguous, more than one result may be the outcome.

Select the *Generator* menu and activate the *Generate Natural Language from DRL* Command. The following window appears. Activate the *File* radio button and enter *.../lingware/ german_demo /test/de_generate.txt* in the *File* box.



File: C:/plain-nlp/lingware/german_demo/test/de_generate.txt

[✓ strings only] strings and trees copy to file

Fig. 23 The *Generate Natural Language from DRL* window

Please compare the DRL source and the generated strings which are displayed. What is remarkable is the effect of comprehensive attributes versus underspecification. For example:

```
DRL source:
(rolle[determination] lexem[definit'] numerus[plural] kasus[dativ]);

Results:
den

DRL source:
(rolle[determination] lexem[definit'] numerus[plural]);

Results:
der
den
die
```

If the definite article is specified with plural dative, we get the single result 'den'. If case is omitted then we get all the plural forms of the definite article 'der', 'den', 'die'. Please study the other examples.

Generation is a good tool for testing whether the grammar is correct and does not overgenerate. This is a prerequisite for an efficient parser. Check our little fragment by entering the output of the parser into the generator. To do this specify

.../lingware/ german_demo /test_results/de_parser_output_drl.txt

in the *File* box and execute the generator.

12 Converters - Cardlforms to Paradigms

For the time being we want to demonstrate just one converter: *Cardlforms to Paradigms.* Cardinal forms (*cardlforms*) are the common way to enter vocabulary into the system by showing its morphological behavior. *Paradigms* are the data structures that are eventually stored in the database.

Two supporting data sets are necessary: cardinal patterns (*cardlpatterns*) and morphological classes (*morphclasses*). The converter detects the morphological class of each cardinal form by comparison with the cardinal patterns. It then creates the paradigm entries according to the description of the morphological class in question.

Adjectives, nouns and verbs are kept apart in the *geman_demo* project. That is why we have three sets of cardlforms, cardlpatterns and morphclasses in the directory ...*geman_demo/layers*.

Adjectives:

.../lingware/german_demo/layers/de_demo_cardlforms_a.xml
.../lingware/german_demo/layers /de_cardlpatterns_a.xml
.../lingware/german_demo/layers /de_morphclasses_a.xml

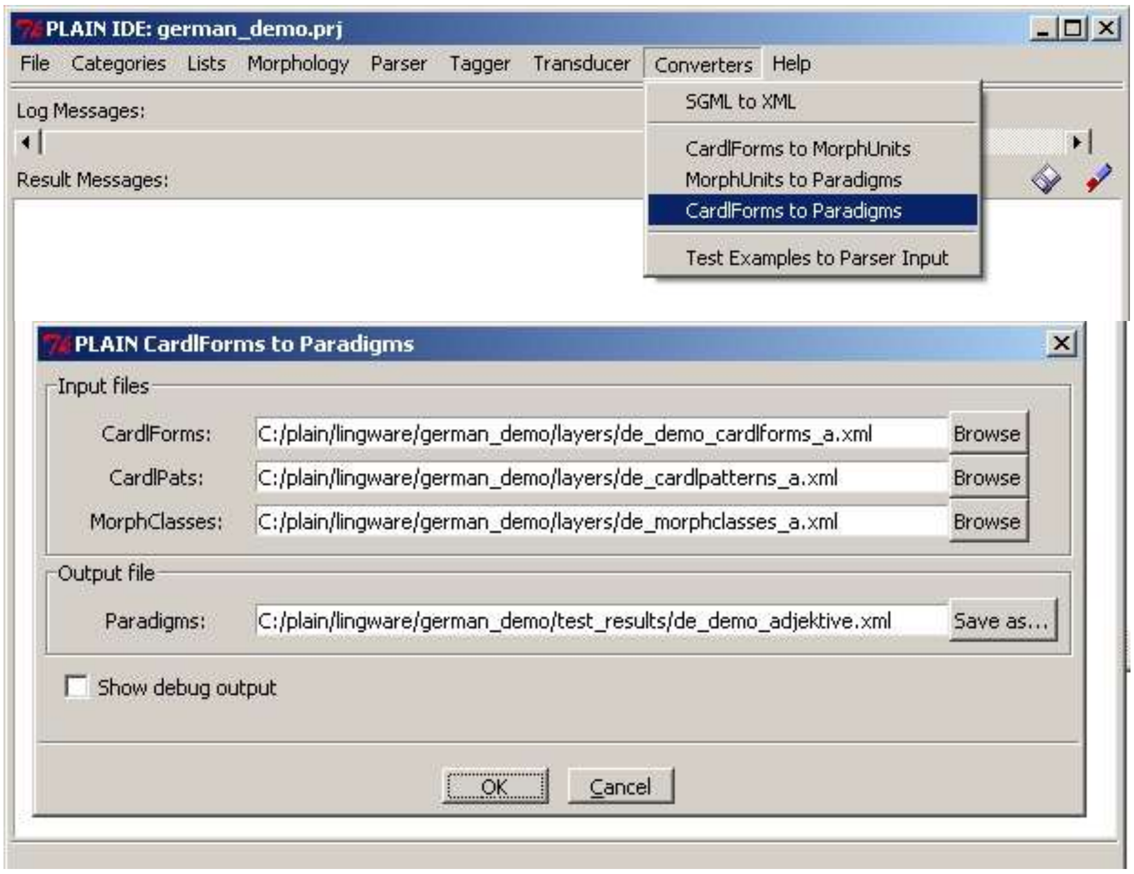
Nouns:

.../lingware/german_demo/layers/de_demo_cardlforms_n.xml
.../lingware/german_demo/layers /de_cardlpatterns_n.xml
.../lingware/german_demo/layers /de_morphclasses_n.xml

Verbs:

.../lingware/german_demo/layers/de_demo_cardlforms_v.xml
.../lingware/german_demo/layers /de_cardlpatterns_v.xml
.../lingware/german_demo/layers /de_morphclasses_v.xml

If you want to convert cardinal forms for adjectives into loadable adjective paradigms then you have to execute command *Cardlforms to Paradigms* in the *Converters* menu and complete the pop-up window in the following manner. Direct the output into a file in the directory *test_results.*



CardlForms: C:/plain/lingware/german_demo/layers/de_demo_cardlforms_a.xml
CardlPats: C:/plain/lingware/german_demo/layers/de_cardlpatterns_a.xml
MorphClasses: C:/plain/lingware/german_demo/layers/de_morphclasses_a.xml
Paradigms: C:/plain/lingware/german_demo/test_results/de_demo_adjektive.xml
☐ Show debug output

Fig. 22 Converter from cardinal forms into loadable *Paradigms*

Do the same with nouns and verbs and then compare the new files of the directory *test_results* with the original files in the directory *load.*