

Edgar Huckert  
Ringofenstr. 23  
63773 Goldbach  
W.Germany  
Mail:edgar.huckert@huckert.com

08-09-2002

## ***HUpars - a parser for natural languages***

## 1. Summary

HUpars is a purely syntactic parser for natural languages. In its actual version it contains no features for semantic analysis. HUpars reads a lexicon, a set of syntactic and morphosyntactic rules, some parsing informations (word boundary and sentence boundary characters), an input sentence and produces from all that an analysis tree. HUpars is thus in a large degree language independent.

The rules form a context free grammar with some notational extensions. The extended notation allows more "natural" rules than in common context free grammars which are normally only suited for formal languages.

HUpars is not deterministic: it produces not a single solution but the set of all possible syntactic derivations. It is the task of a (not included) semantic component to filter out the semantic probable solutions.

The interface to a human user or to a software environment is not very developed: I generate one or more tree structures that explain the syntactic structure of the sentence to be analyzed. A linear bracket notation is also generated: this notation is more adapted to be passed as a parameter to a software environment.

I found that the parsing speed of HUpars is quite good compared to the usual LISP or PROLOG parsers - I know that parsers cannot be compared in speed alone. On my laptop (Pentium 848 Mhz) it analyzes 800 German medium-size sentences in 200 - 300 milliseconds.

HUpars is a hobby tool created around 1987 by E.Huckert. The basic ideas found in HUpars go back to a parser developed by Prof. Bockhaus (University of Konstanz, then University of Heidelberg, now TU Berlin). The original program ran on an IBM 360 system and was written in PL/I. The actual version has been much improved in speed and flexibility. I wrote it without knowledge of the original code (although Prof. Brockhaus was my boss when I was member of the "Sonderforschungsbereich 99" at the University of Konstanz/Germany). I even don't know whether the original parser used a top-down or a bottom-up model. HUpars in any case uses the bottom-up model.

HUpars is a package containing the following components:

- the main program HUpars offers
  - a simple tokenizer
  - dictionary lookup
  - simple morphology (segmentation)
  - a bottom up parser
  - visualization routines

- some auxiliary programs:

- syncheck: a syntax checker form rules and lexicon
- scanner: an advanced tokenizer

HUwbsort: a sort program that eliminates duplicate entries  
dictmant: a GUI (Windows) based dictionary maintenance program

Please do not distribute the code without my permission.

## 2. Getting started

Start HUpars by saying in the simplest case:

```
HUpars
```

HUpars will ask you for a configuration file. The present version contains two configuration files: KONF.DEU for German and KONF.FRZ for French.

HUpars will ask you to input a sentence for analysis. Take care that all words of the input sentence have an entry in the lexicon - HUpars does not start the analysis if the lexicon is incomplete. If the analysis completes you get one or more analysis trees and the corresponding result in bracket notation.

If it fails you get an analysis list. You need some experience to read this list. It is impossible to output an analysis tree for an incomplete analysis because no complete tree (a path through the analysis list) has been created.

Stop HUpars by entering a dummy (empty) line.

You may enter some command line parameters (for VMS: make HUpars a "foreign command"). The following command line parameters are accepted:

- c configfile  
(ex: "-c konf.eng" selects English grammar)
- a axiom  
(default axiom is SATZ, -a SENT selects SENT as the axiom symbol in your grammar)
- i file: accept a series of sentences (one sentence per line) from a file
- tree: add a two dimensional analysis tree to the output (the bracket notation is always output). This is pseudo graphics based on fixed character widths.
- list: add a list representation of the parsing result to the output. A list representation is always shown if the parser cannot reach the axiom.
- subcat: show the values for the subcategories
- v: trace, verbose (switches on a trace flag - for implementers)

It is not necessary to enter command line parameters. The program uses defaults and asks for the configuration file. For the analysis of very large sentences it is better to select output in brackets notation as the trees fit not on a single screen page (the tree output should be more comfortable).

A typical start command for HUpars could thus be:

```
HUpars -c konf.deu -tree -i sentences.txt
```



### 3. The Configuration File

HUpars reads the file names for the lexicon and the grammar from a configuration file. A typical configuration file looks like this:

```
MESSAGES.DEU
32,33,38,39,40,41,44,45,46,47,59,61,63,95
33,46,59,63
REGELN.DEU
LEXI.DEU
ENDU.DEU
```

The first line contains the name of a message file. Each message of the parser program (except the startup messages) is referenced in the program via a message number: this file contains the message text related to the message numbers. The program is thus translatable to different user environments languages.

You see in the two following lines informations for the low level parsing ("parsing" is an ambiguous word). The second line contains codes (decimal notation) for the identification of words (word boundary codes), the third line contains sentence boundary codes. On the fourth line you have to enter the file name of a grammar (here REGELN:DEU). The next line contains the name of a lexicon file (her LEXI.DEU). The last line contains a file name for the endings - for the small morphological component of HUpars.

The order of the lines in the configuration file is important: don't change it. If you plan to write grammars or lexicons for new languages, just write a new configuration file!

## 4. The Lexicon

The lexicon attributes a lexical category to a word (a lexeme) of a sentence. In our case we also use categories for parts of words (at the moment for stems and endings). Note that we use "complex categories", i.e. categories consisting of a main category (ex: NOUN) and a (possibly empty) sequence of one or more subcategories with values.

Let me explain the structure of the lexical entries with the following extract:

```
ARBEIT=RVER REK(1,7)
BIN=KOPU NUM(1)PER(1)TEM(1)
BIST=KOPU NUM(1)PER(2)TEM(1)
DEN=ARTI NUM(1)GEN(1)CAS(4)
```

You see that a lexeme is separated from its category by an equal sign. You see then a main category (4 characters) followed by one or more subcategories (3 characters) with their values. The values for subcategories stand in brackets. If a subcategory has more than one value (ARE in English can be 1.,2. or 3. person) then separate the values by commas.

In the program code appears an additional separator: a colon. It is used to add a semantical item (called a "semanteme") to the end of the lexical entry. I have included no example for the use of the semantemes. I used semantemes some time ago for the addition of semantic informations guiding the generation process of database queries in a 4GL (fourth gen. language).

The subcategory values are actually represented internally by bit strings limited to a byte, i.e. values from 1 to 8 are allowed.

The lexicon must be sorted. The utility program SYNCHECK can be used to check the sort order (and the correctness of the entries). HUpars makes no syntax checks when reading the lexicon.

## 5. The Grammar

A context free grammar is used to define the analysis rules. We use here essentially a Chomsky normal form with some extensions. A "normal" Chomsky normal form only has productions of the type

$$A = B + C$$
$$A = a$$

where the latter form denotes lexical rules. We use three types of rules:

$$A = B + C$$
$$A = B$$
$$A = a$$

Note the second type: we use here technically unnecessary rules (like synonyms in semantics) which help the linguist to write short and "natural" grammars (rules of the type "each personal pronoun is a noun group").

Besides the fact that our rules are written in analysis direction (i.e.  $A + B = C$  instead of  $C = A + B$ ) we use complex categories (main categories + subcategories + values for subcategories). Complex categories are the essential mechanism to reduce the number of rules: a good grammar for an indo-european type of language like German, French or English will show less than 100 rules. Using the classic rules several thousand rules would be needed thus making the grammar unmanageable.

I explained the lexical rules (the lexicon) in chapter 4. Let me explain some rules with the following example:

```
$ E.Huckert 4/84
$ Kontextfreie Syntax Deutsch
$
$ Jedes Adjektiv in Grundform ist Adverb
ADJE NUM(1)GEN(1)CAS(1) =
ADVE; .
$
$ Adjektiv + Substantiv : "lieber Junge"
ADJE NUM()GEN()CAS() +
SUBS NUM()GEN()CAS() =
SUBS NUM()GEN()CAS();
(1,1,1)(2,2,2)(3,3,3).
```

You see here two rules: the first is of the type  $A = B$  and the second of the type  $A + B = C$ . Lines starting by '\$' are comments (each rule should have a comment explaining its function). Subcategory values can be explicitly specified (ex:  $GEN(1)$ ) or can be omitted which means that all values are allowed ( $GEN()$ ).

At the end of the second rule you see a series of restrictions on subcategory values which is to be read as: the value of the first subcategory

of the first rule part must be in the set of the values of the first subcategory of the second rule part and the result (the intersection) must be transported in the value of the first subcategory of the third rule part. Assume that (at run time) NUM(2,3) is in a category ADJE and NUM(3,4) is in the category SUBS: then the intersection NUM(3) will be transported into the resulting category SUBS. Restrictions over subcategories are the convenient way to formulate linguistic congruence rules.

The characters '+', '=', ';' and '.' are used as separators between rule parts. Use the SYNCHECK program to check the correctness of the rules. HUpars makes no syntax check when reading the rules.

## 6. Recommended Tests

When you test a grammar try to proceed this way:

1. Write the grammar (the rules)
2. Write a lexicon
3. Test grammar and lexicon with program syncheck
5. Write or change the configuration file
4. Test with program HUpars

I recommend to start testing with very simple sentences and to vary them step by step. The following example uses some French sentences as test inputs:

```
pierre travaille.
pierre travaillait.
pierre est gentil.
pierre est tres gentil.
la femme est tres gentille.
il travaille au jardin.
ils travaillent au jardin.
la femme travaille au jardin.
la jolie femme travaille au jardin.
une femme tres jolie travaille au jardin.
pierre donne le ballon a la femme.
pierre donne le joli ballon a la femme.
nous donnons le joli ballon a la femme.
pierre et jean travaillent au jardin
pierre et la fille travaillent au jardin.
pierre et la fille travaillaient au jardin.
```

The next examples show German test sentences:

```
peter arbeitet.
peter arbeitet gerne.
peter arbeitet gerne im garten.
peter arbeitet gerne heimlich im garten.
der mann und die frau arbeiten.
der junge sieht die frau.
der junge sieht heimlich die frau.
der junge sieht gerne heimlich die frau.
der junge sieht gerne heimlich die frau im garten.
```

der junge, der die frau sieht, hebt gerne den klotz.

The last example shows some English test sentences:

```
john works.  
john is working.  
john works in the garden.  
the woman works in the garden.  
a nice woman is working in the garden.  
a very nice woman is working in the garden.  
john and mary work in the garden.  
john and mary work simultaneously in the garden.
```

The example input sentences show increasing syntactic complexity.  
(note that I didn't include accents in the sample French lexicon).

All grammars are actually restricted to declarative sentences.  
This is not a built-in restriction of the parsers. In any case  
the grammars can be expanded to cover more interesting extracts  
of the respective languages. Note that you also need an elaborated  
lexicon if you plan more serious tests.

The German data files have the extension .DEU, the French data  
files the extension .FRZ and the English data files the extension .ENG

## 7. Weak points

When looking at my code after a pause of several years I  
found the following weak points - I am sure that there are  
more:

- The grammars are not as good as I thought
- The lexicons are just samples - not more
- I use no informations like sentence boundary chars  
(interrogation, exclamation etc.) during the parsing process
- I use no diacritic characters like French accents
- I make no use of lower/upper case informations
- The program does not recognize abbreviations or word groups