Tagging and parsing German using Spejd

Andreas Völlger
Reykjavik University
Reykjavik, Iceland
andreasv10@ru.is

Abstract

Spejd is a newer tool for morphosyntactic disambiguation and shallow parsing. Contrary to other shallow parsing formalisms, the rules of the Spejd grammar allow explicit morphosyntactic disambiguation statements, independently of structure building statements. With Spejd it is possible, that a single grammar rule may contain structure building operations as well as morphosyntactic disambiguation operations. The formalism is Java based and available under the GNU license. Furthermore Morphy, which offers morphosyntactic analysis and synthesis, is used to derive the input data for Spejd.

1 Introduction

There were two observations, which motivated the work on the formalism. First, that morphosyntactic disambiguation and shallow parsing in general should be performed in parallel, rather than in sequence.

Morphosyntactic disambiguation focuses on word structure, their smallest elements, called morphemes. Prefixes and suffixes are examples of morphemes, they are called affixes. There are a lot of wordstems for possible correct affixes. Often these affixes give information about the grammatical use of the word like gender. An affix mostly has several possible meanings, such that morphosyntactic analysis offers ambiguous information. An additional tool has to process the data and disambiguate the information.

Shallow Parsing or chunking is the analysis of sentences. Each sentence has constituents like noun groups or verb groups. Shallow parsing does not take care of the sentence structure, but rather on structure of the defined groups. to conclude shallow parsing focuses on predefined sentence segments, whereas full parsing focuses on the whole sentence structure.

The second observation, it should be possible to encode both the disambiguation and parsing within a single rule, because they often implicitly encode the same linguistic intuitions. The formalism originally abbreviated to SPADE, which means *Shallow Parsing and Disambiguation Engine*. Due to the existence of an earlier system, the name was abbreviated to ♠ and the acronym Spejd was used.

The formalism itself connects several regular grammars - in other words cascades them. Each of these grammars is expressed by a rather complex single rule which specifies morphosyntactic disambiguation as well as structure building operations. Morphy (Morphy, 2010) is a closed source morphosyntactic analyser for German, which helps to derive the Spejd input and later on checks against the solution of the rules. Morphy offers different formats, for example an XML like output format, which differs a lot from the Spejd expected input format. There is also the possibility to detect sentence breaks.

For analysis a German fourth class dictate is used (Diktat, 2005) for several reasons. First, dictates from lower classes include names much more often and Spejd has problems with detecting names, so it is a source of possible errors. A second reason is that sentences of higher classes are possibly too nested to show that Spejd can do the expected work. If there are more specialised and optimized rules it is possible to test Spejd on more difficult texts, but for that aim it is a decent choice.

In this paper the used sentence is ”Die Katze fängt die Maus.”, translated into english ”The cat catches the mouse.”. The sentence offers several possibilities to show in an easy way how and what Spejd can do.

2 Related work

Syntactic parser differ in their assumption of input. Either they assume morphosyntactically disambiguated input or non disambiguated input. Generally ambigous input is expected in deep parsing systems which are based on unification, whereas fully disambiguated input is usually expected in shallow or partial parsers. There are some shallow parsing systems which allow disambiguation and parsing, for example (Neumann, 2000) or (Marimon, 2000). Some other approaches like (Karlsson, 1995) introduced a formalism for dependency parsing and disambiguation. These assume, that every word has assigned its possible role in a lexicon. While processing some of these roles are rejected. However, this method is a pure disambiguation formalism.

3 Procedure

After getting a brief overview on the available Spejd papers the next step was trying to figure out how Spejd and the tagset in combination with the ruleset work, due to incomplete documentation. A good starting point is the Spejd config file, where one get a brief overview of the expected files and options combined with some lines of documentation. For that reason the first part was to alter existing rules and make them work in the expected way and to get a feeling how Spejd proceeds. After altering *Left*, *Match* and *Right* parts, the next step was to work with some *Match* functions, which wasn't as easy as expected. At that point one has a brief feeling on how Spejd works and can proceed with the own text due to the fact that the sample is given in Polish, but still works.

The next part is to set up another working example, in that case in German, by creating a valid IPIPAN xml file. Based on this working example, new rules can now be created, processed and later on improved. On change, it is recommended to document every single rule, for subsequent changes could destroy the whole rule proceeding.

4 Formalism

Spejds needs a pre-defined tagset as part of the configuration. Therefore a new tagset with attributes e. g. for number, gender and case was built.

```
number = SIN PLU
gender = NEU FEM MAS
case   = NOM GEN DAT AKK
```

The tagset file contains part of speech information, which are necessary for the rules that will be applied on the input. A possible part of speech section looks like the following one. The attributes in box brackets are optional and should be located at the end of the part of speech entry.

```
PRO = number person personal
      [possesiv]
ART = case number gender
      [definit]
SUB = case number gender
```

This tagset definition requires, that one input word whose type is "PRO" (pronoun) needs at least one attribute for number, gender and case. A definition attribute for possesive pronouns is optional. The word classes from Morphy are listed in (Morphy - word classes, 2010). The abbreviations in capital letters are attributes from the Morphy output and Spejd input which cannot be chosen arbitrarily, but are also defined in (Morphy - word classes, 2010).

A rule is defined to have at least *Rule*, *Match* and *Eval* part. The Rule part is a string identifier and can be named freely. It is the only entry in the definition which does not end with a semicolon. The Left, Match and Right part of the rule is handled in the same way, but the Match part is obligatory. Each of these three parts can be modified by regular expression quantifiers. The Eval part describes the methods performed when rules match. There are several possible functions which are detailed later. Comments can be added by using the hash character #.

```
Rule:  "ART_SUB"
Left:  ;
Match: [pos~"ART"][pos~"SUB"];
# article followed by noun
Right: ;
Eval:  unify(case number
       gender, 1, 2);
```

The rule above finds a sequence of two tokens in which the first token is an article and the second a noun. Interpretations of case, number and gender attributes of both tokens are retained when they do agree, else they are rejected. By using double tildes the rule checks against disambigous input. If for example a noun can only be a noun and no other part of speech type, all other morphosyntactic interpretations of the token are nominial. By using one tilde there exists a nominial interpretation of the token with the part of speech tag SUB.

Possible Eval functions are:

- agree
- unify
- delete
- leave
- add
- set
- word
- group

For the current approach not all functions are needed. The most often used functions are described here.

agree(category ..., token ...) - checks the grammatical categories of the specified tokens. It is allowed to use several numbers of arguments for category and token.

unify(category ..., token ...) - retains those interpretations of the tokens which agree in specified category. Only works if *agree* works, otherwise nothing changes.

leave(condition, token ...) - retains only the interpretation which matches specified condition.

delete(condition, token ...) - remove all interpretations of given token matching specified condition.

group(name, syntactic head, semantic head) - group all tokens matching specified conditions. Additionally a syntactic and semantic head is specified, but currently unused.

5 Parsing

While using several functions within a single rule it is necessary to document every single step. Subsequent changes can cause errors in fact of changing token numbers. If the previous rule is altered by adding a possible adjective between article and noun, the tokens in the Eval part also have to change, otherwise it could cause an error.

The proceeding when parsing German is visualised in figure 1.

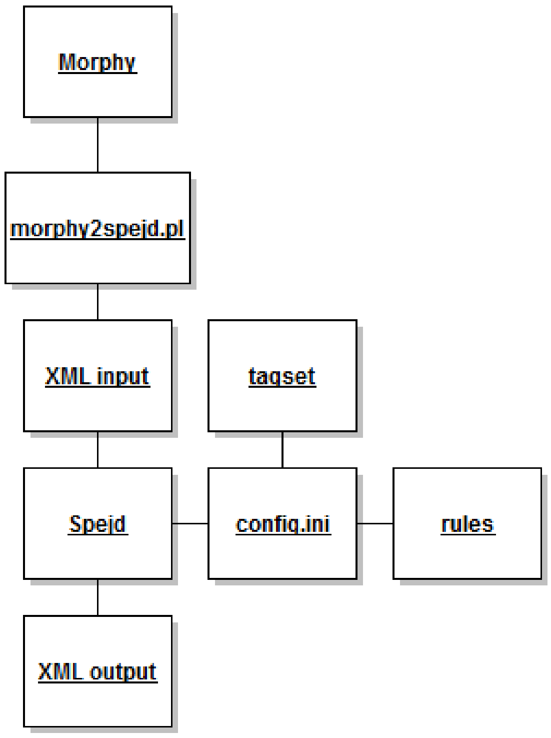

Figure 1: system architecture

```
Rule:  "ART_SUB"
Left:  ;
Match: [pos~"ART"]
       [pos~"ADJ"]*
       [pos~"SUB"];
# article followed by noun
Right: ;
Eval:  unify(case number
       gender, 1, 3);
```

The first step is to derive the information from Morphy's morphological analyser. Morphy delivers different output formats and options. This output offers too many information, thus it needs to be processed further to bring it in the expected Spejd XML scheme. Because there are some problems while processing umlauts, they have to be converted. Furthermore Morphy offers possibilities for analysed words, which are marked with an asterisk. For processing the words, these entries are unnecessary and can sometimes be discarded, usually if they are not at sentence beginning.

```
@
Die
*der PRO PER NOM SIN FEM
*der PRO PER NOM PLU ALG
*der PRO PER AKK SIN FEM
*der PRO PER AKK PLU ALG
```

```
⋆der PRO DEM AKK SIN FEM
⋆der PRO DEM NOM SIN FEM
⋆der PRO DEM NOM PLU ALG
⋆der PRO DEM AKK PLU ALG
⋆der ART DEF NOM PLU MAS
⋆der ART DEF AKK PLU MAS
⋆der ART DEF NOM SIN FEM
⋆der ART DEF AKK SIN FEM
⋆der ART DEF NOM PLU FEM
⋆der ART DEF AKK PLU FEM
⋆der ART DEF NOM PLU NEU
⋆der ART DEF AKK PLU NEU
@
Katze
Katze SUB NOM SIN FEM
Katze SUB GEN SIN FEM
Katze SUB DAT SIN FEM
Katze SUB AKK SIN FEM
@
faengt
fangen VER 3 SIN PRAE NON
...
```

When processing the morphy output a perl script converts the plain text into the expected xml format. This is done by some rather complex regular expressions. It is necessary to notice, that the first abbreviation in capital letters is a part of speech definition of the word, all following abbreviations are attributes. Additionally the xml frame is added so the output is a valid IPIPAN xml file.

```
<?xml version="1.0" encoding=
      "UTF-8"?>
<!DOCTYPE cesAna SYSTEM
      "xcesAnaIPI.dtd">
<cesAna xmlns:xlink="http://
      www.w3.org/1999/xlink"
      type="pre_morph"
      version="IPI-1.2">
<chunkList xml:base="text.xml">
<chunk type="p" xlink:href=
      "#dv1hd1">
<chunk type="s">
<tok>
<orth>Die</orth>
<lex><base>der</base><ctag>
PRO:PER:NOM:SIN:FEM
</ctag></lex>
<lex><base>der</base><ctag>
PRO:PER:NOM:PLU:ALG
</ctag></lex>
<lex><base>der</base><ctag>
PRO:PER:AKK:SIN:FEM
</ctag></lex>
<lex><base>der</base><ctag>
PRO:PER:AKK:PLU:ALG
</ctag></lex>
<lex><base>der</base><ctag>
PRO:DEM:AKK:SIN:FEM
</ctag></lex>
<lex><base>der</base><ctag>
PRO:DEM:NOM:SIN:FEM
</ctag></lex>
<lex><base>der</base><ctag>
PRO:DEM:NOM:PLU:ALG
</ctag></lex>
<lex><base>der</base><ctag>
PRO:DEM:AKK:PLU:ALG
</ctag></lex>
<lex><base>der</base><ctag>
ART:DEF:NOM:PLU:MAS
</ctag></lex>
<lex><base>der</base><ctag>
ART:DEF:AKK:PLU:MAS
</ctag></lex>
<lex><base>der</base><ctag>
ART:DEF:NOM:SIN:FEM
</ctag></lex>
<lex><base>der</base><ctag>
ART:DEF:AKK:SIN:FEM
</ctag></lex>
<lex><base>der</base><ctag>
ART:DEF:NOM:PLU:FEM
</ctag></lex>
<lex><base>der</base><ctag>
ART:DEF:AKK:PLU:FEM
</ctag></lex>
<lex><base>der</base><ctag>
ART:DEF:NOM:PLU:NEU
</ctag></lex>
<lex><base>der</base><ctag>
ART:DEF:AKK:PLU:NEU
</ctag></lex>
</tok>

<tok>
<orth>Katze</orth>
<lex><base>Katze</base><ctag>
SUB:NOM:SIN:FEM</ctag></lex>
<lex><base>Katze</base><ctag>
SUB:GEN:SIN:FEM</ctag></lex>
<lex><base>Katze</base><ctag>
```

```
SUB:DAT:SIN:FEM</ctag></lex>
<lex><base>Katze</base><ctag>
SUB:AKK:SIN:FEM</ctag></lex>
</tok>

<tok>
<orth>faengt</orth>
<lex><base>fangen</base><ctag>
VER:3:SIN:PRAE:NON</ctag></lex>
</tok>

...

</chunk>
</chunk>
</chunkList>
```

The generated IPIPAN like xml file can now be handled by using the following command.

```
java -jar spejd.jar input.xml
```

It is also possible to batch all xml files in a directory by not specifying an xml file at the end of the statement. Spejd throws exceptions if part of speech definitions or attributes are not defined. In case to not lose any information actually unused entries have to be stored as dummy entries in a single attribute like the following entry. Another way of processing is to remove all unused entries via an regular expression in the perl script.

```
temp = PRP KON KJ1
```

The Spejd output could look like the example below.

```
<?xml version="1.0"
  encoding="UTF-8"?>
<!DOCTYPE cesAna SYSTEM
  "xcesAnaIPI.dtd">
<cesAna xmlns:xlink=
  "http://www.w3.org/1999/xlink"
  type="pre_morph"
  version="IPI-1.2">
<chunkList xml:base="text.xml">
<chunk type="p"
 xlink:href="#dv1hd1">
<chunk type="s">

<group id="a8"
  rule="ART_SUB_VER_ART_SUB"
  synh="a2" semh="a2"
  type="ART_SUB_VER">
<group id="a6" rule="ART_SUB"
  synh="a2" semh="a2"
  type="ART_SUB">

<tok id="a1">
<orth>Die</orth>
<lex><base>der</base><ctag>
PRO:PER:NOM:SIN:FEM</ctag></lex>
<lex><base>der</base><ctag>
PRO:PER:AKK:SIN:FEM</ctag></lex>
<lex><base>der</base><ctag>
PRO:DEM:AKK:SIN:FEM</ctag></lex>
<lex><base>der</base><ctag>
PRO:DEM:NOM:SIN:FEM</ctag></lex>
<lex><base>der</base><ctag>
ART:DEF:NOM:SIN:FEM</ctag></lex>
<lex><base>der</base><ctag>
ART:DEF:AKK:SIN:FEM</ctag></lex>
</tok>

<tok id="a2">
<orth>Katze</orth>
<lex><base>Katze</base><ctag>
SUB:AKK:SIN:FEM</ctag></lex>
</tok>

</group>

<tok id="a3">
<orth>faengt</orth>
<lex><base>fangen</base><ctag>
VER:3:SIN:PRAE:NON</ctag></lex>
</tok>

...

</group>
</chunk>
</chunk>
</chunkList>
```

The first recognised new entries compared to the previous xml file are the grouping tags. In every Eval part it is possible to group the rules. Especially for debugging and better readability the xml files it is a good way to show, which rule matches. Otherwise no grouping tag would be written to the xml output. Furthermore the rule attribute of each group tag offers information on which rule matched, so one can check whether a rule matched in the expected way. An additional type attribute offers information on the group name (first attribute of the group function).

The matched rules in the example are ART_SUB and ART_SUB_VER_ART_SUB. ART_SUB is a extended function of the rule above and ART_SUB_VER_ART_SUB is a rule for sentences with structure ART_SUB, followed by a verb, followed by ART_SUB. The first two parts of the rule have to match in number and are grouped with ART_SUB_VER. Additionally the German grammar defines that in this special sentence structure the first ART_SUB combination is in accusative case and second is nominative case, so the rule only leaves these cases.

In detail it looks like:

```
Rule "ART_SUB_VER_ART_SUB"
Left:  ;
Match: [type="ART_SUB"]
   [pos~"VER"]
   [type="ART_SUB"];
Right: ;
Eval:  unify(number, 1, 2);
   group(ART_SUB_VER, 1, 1);
   leave(case~"AKK",1);
   leave(case~"NOM",3);
```

A problem still existing is that the leave method does not perform as expected. Leave only proceeds on the noun in *Match: [type="ART_SUB"]* and does not change the article case.

6 Evaluation

The tested sample text (Diktat, 2005) includes 81 words. Morphy provides 513 morphosyntactic different entries. After Spejd processed the xml file, it contains 244 different entries, which means a reduction of more than 50%, while only applying a few rules. This means, that every single entry has 3 morphological tags. On deeper observation one can see that some combinations are fraught with problems, but there is no solution. The issue is that an arcticle in front of a noun can also be a pronoun or verb. Hence there are more morphosyntactic possibilities for the noun, for example 44 possibilities for a single adjective. By including an expression for excactly this combination, the hit ratio in the text could be increased by more than eight percent, this means about 2.5 tags per word.

7 Conclusion

The approach has shown, that it is possible to disambiguate sentences with Spejd. By using a prototypical implementation of rule- and tagset it was shown, that Spejd works for German texts and concerning the defined rules very well. By improving the existing rules and adding new rules it should be possible to reach nearly full disambiguation. Furthermore, rules have to be well documented an validated because the rules are not easy to read. Also if new attributes are added to an existing rule, problems can occur when having unchanged item numbers in the eval functions because these do not change dynamically. Although using any kind of reference, they are still hard coded, which is cause for possible errors.

References

Neumann, G., Braun, C., Piskorski, J. 2000. *A divide-and-conquer strategy for shallow parsing of German free texts*. In: Proceedings of the 6th Applied Natural Language Processing Conference, Seatle, WA, ACL.

Marimon, M., Porta, J. 2000. *PoS disambiguation and partial parsing bidirectional interaction*. In: ELRA: Proceedings of the Third International Conference on Language Resources and Evaluation, LREC 2000.

Karlsson, F., Voutilainen, A., Heikkilä, J., Anttila, A. 2000. *Constraint Grammar: A Language-Independent System for Parsing Unrestricted Text*. Mouton de Gruyter, Berlin.

Leizius, W. *Morphy*. http://www.wolfganglezius.de/doku.php?id=cl:morphy. last visited: 01.12.2010.

Leizius, W. *Morphy*. http://www.wolfganglezius.de/lib/exe/fetch.php?media=cl:wklassen.pdf. last visited: 01.12.2010.

Diktat, 4.Klasse. http://www.note1plus.de/BilderS12-PU-Aufg/PU-Realsch4-2005/Diktat.pdf. last visited: 01.12.2010.

Buczynski, A., Przepirkowski, A. 2008. *An Open Source Tool for Partial Parsing and Morphosyntactic Disambiguation.*. In: Proceedings of LREC 2008.

Buczynski, A., Wawer, A. 2008. *Shallow parsing in sentiment analysis of product reviews.*. In: Proceedings of the Partial Parsing workshop at LREC 2008, pp. 14-18.