Implementierung einer *Constraint Grammar* für das Deutsche

Alex Linke, Rona Linke, Björn Wilmsmann

Sprachwissenschaftliches Institut

Ruhr-Universität Bochum

Ablauf des Programms

- Korpus einlesen
- Tokenisieren des Korpus

  ```
  $tokenizer -> tokenize()
  ```

- Extrahieren der Sätze aus den einzelnen Token

  ```
  $tokenizer -> extract_sentences()
  ```

- Morphologische Analyse

  ```
  $lookup -> lookup()
  ```

Ablauf des Programms

- Erster Durchlauf der Disambiguierung
- evtl. weitere Durchläufe
- Ausgabe der Kohorten
- Berechnung und Ausgabe der Laufzeit

Nun ein ausführliches Beispiel...

Alle zum Tagging auf Constraint-Basis notwendigen Schritte werden an folgendem Beispielsatz verdeutlicht:

„Sie werden den Todesstern zerstören!“

Tokenisierung I

Lingua::Tokenize – Perl-Modul zur Tokenisierung:

```
my @tokens = $tokenizer->tokenize( {
        normalize => 1,
        tokendef => 'words',
        input => \$wholeCorpus
} );
```

Rückgabe:

- Tokenliste in unveränderter Reihenfolge
- normalisiert

Tokenisierung II

Der Beispielsatz liegt nun in der folgenden segmentierten und normalisierten Form vor:

„sie“	„werden“	„den“	„Todesstern“	„zerstören“	„!“

Tokenisierung II

Der Beispielsatz liegt nun in der folgenden segmentierten und normalisierten Form vor:

„sie“	„werden“	„den“	„Todesstern“	„zerstören“	„!“

Statistisches Kriterium: (mit Korpus-Filter)

„Sie“ $\rightarrow$ „sie“,

wenn für die Textfrequenz gilt:

$$F(\text{„}sie\text{“}) > F(\text{„}Sie\text{“})$$

Extraktion der Types

Durch die bisherigen Aufbereitungsschritte sind alle *Tokens* des Inputs ermittelt worden.

Zur anschließenden morphologischen Analyse müssen jedoch nur *Types* betrachtet werden, denn die morphologische Analyse ist unabhängig vom Kontext!

```
my @types = $tokenizer->unify( \@tokens );
```

Rückgabewert: Liste aller Types

Aufbereitung der Lexika

Welche Informationen werden im Lexikon hinterlegt?

- Stammform
- Flexionsendungen
- Tags
  - Wortart
  - morphologische Kategorien

Tags

Wortarten:
Tags aus dem STTS mit Ergänzungen aus dem UIS für Interpunktion

morphologische Kategorien:
es werden nicht alle möglichen Kategorien angegeben - verwendet werden:
Person, Numerus, Kasus,
Komplement_Numerus und Komplement_Kasus

Die einzelnen Lexika

Lexika der **geschlossenen Klassen** können komplett erfasst werden.

Lexika der **offenen Klassen** benötigen neben den vorhanden Einträgen eine Möglichkeit, weitere Formen zu erkennen.

Es werden einzelne Lexika pro STTS-Tag erstellt, die zusammen einen *Transducer* bilden.

Dafür werden die Programmiersprachen **xfst** und **lexc** verwendet.

Das Lexikon

Der Transducer nimmt konkrete Wortformen zur Eingabe und gibt die *morphologische Analyse* zurück.

Wortform → Stamm+Tags/Merkmale

Das Lexikon

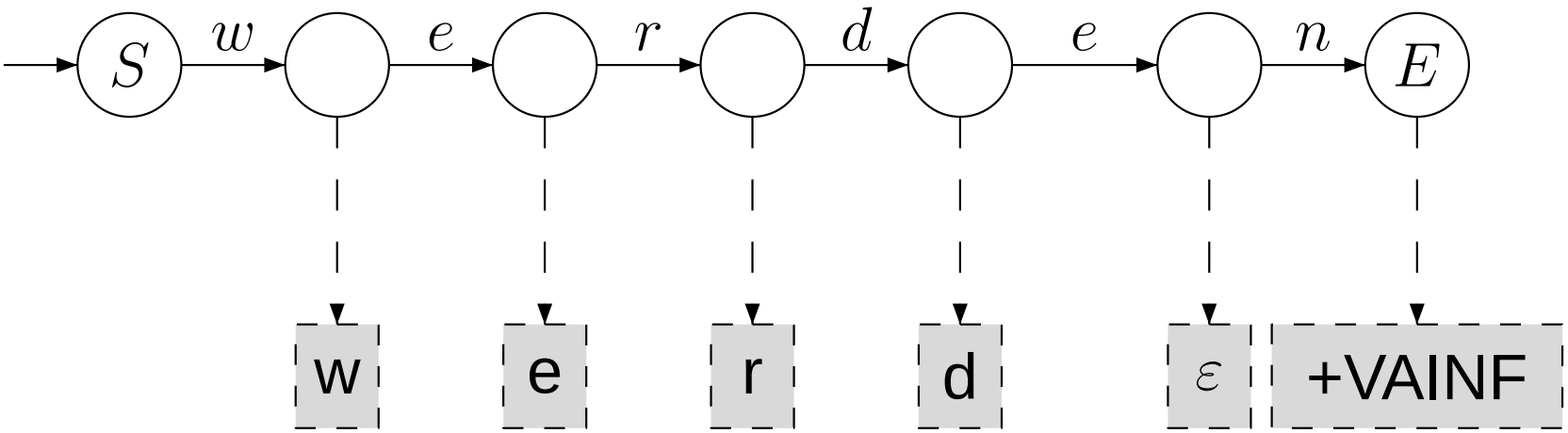
Der Transducer nimmt konkrete Wortformen zur Eingabe und gibt die *morphologische Analyse* zurück.

Wortform → Stamm+Tags/Merkmale

Flexionssuffixe und orthographische Besonderheiten werden zur Zuweisen von Tags/Merkmalen benutzt.

Morphologische Analyse

`Input:` „werden“ `Output:` werd+VAINF



Während der *Analyse* der Wortform „werden“ wird durch schrittweises Konsumieren von Inputzeichen ein Output durch den Transducer erzeugt. Flexionssuffixe können dabei in Tags/Merkmale übersetzt werden.

Morphologische Analyse II

XFST::Lookup – Perl-Modul zum Ansprechen des **XEROX** Finite State Tools *lookup*

```
my %analysed = $lookup->lookup( {
    strategy => 'strategy.txt',
    words => \@types
} );
```

Rückgabe: Hash of Arrays

- *Key*: Wortform
- *Value*: Array der zugehörigen Kohorte

Analyse einer Wortform

. . . erfolgt nach folgender Strategie:

- „Suchen“ der Wortform im Lexikon
  wenn es keinen Eintrag gibt . . .
- „Suchen“ der Wortform in einem Wortarten-Guesser
  wenn die Wortform immer noch nicht erkannt ist . . .
- Zuordnung aller Wortarten

Jeder Input kann analysiert werden!

Analyse einer Wortform

für Wortformen aus dem Lexikon:

Anhand von Stamm und Flexionsendung erfolgt die morphologische Analyse

Am Beispiel „*werden*“

Stamm:*werd-* Flexionsendung: *-en*

$\Rightarrow$ alle möglichen Lesarten werden ausgegeben (Kohorte)

Ambiguitäten auf Wortebene

Wortform	Stamm	Analyse/Tags
werden	werd	+VAFIN+3P+PL
"	"	+VAFIN+1P+PL
"	"	+VVFIN+PRAES+1P+PL
"	"	+VVFIN+PRAES+3P+PL
"	"	+VAINF
"	"	+VVINF

Ambiguitäten auf Wortebene

Wortform	Stamm	Analyse/Tags
werden	werd	+VAFIN+3P+PL
"	"	~~+VAFIN+1P+PL~~
"	"	~~+VVFIN+PRAES+1P+PL~~
"	"	~~+VVFIN+PRAES+3P+PL~~
"	"	~~+VAINF~~
"	"	~~+VVINF~~

Diese Lesarten sind (an dieser Stelle) **falsch**!

Die Disambiguierung

Die Funktion `disambiguate()` nimmt drei Argumente:

1. die zu disambiguierenden Sätze (Array)
2. die Kohorten der Wortformen (Hash)
3. ein Flag, das angibt, ob die Kohorten nach Worten (=0) oder nach ihrer Position im Korpus indiziert werden

Rückgabewert:
eine Hashreferenz mit den disambiguierten Kohorten

```
Position im Korpus => Kohorte
```

Ablauf der Funktion

Einlesen aller Constraints

```
foreach(jeder Satz)
    foreach (jeden Token)
```

. . . finde die passende Kohorte

```
foreach(Lesart aus dieser Kohorte)
```

. . . Lese Constraints für diese Lesart ein

```
foreach (Constraint)
```

. . . extrahiere Modus

Die Modi

- DISCARD_THIS (=0):
  hole Kohorte an Umgebungsposition, die im Constraint angegeben wurde und vergleiche jede Lesart mit den Bedingungen, die im Constraint für diese Position angeführt sind. Falls die Bedingung in einer Lesart der Umgebungskohorte erfüllt ist, verwerfe die gerade behandelte Lesart.

analog dazu:

Die Modi

- DISCARD_THIS_UNLESS (Komplement von =0)
- DISCARD_OTHERS (=!)
- DISCARD_OTHERS_UNLESS (Komplement von =!)
- DISCARD_STRICTLY_UNLESS (=!!)
- DISCARD_STRICTLY (Komplement von =!!)
- CHANGE_THIS =* (keine Entsprechung)
- CHANGE_THIS_UNLESS =* (Komplement von CHANGE_THIS)

Ablauf der Funktion

Speichern der disambiguierten Kohorten in einem Hash

```
Position im Korpus => Kohorte
```

Speichern der disambiguierten Kohorten in einer Datei

Rückgabe der disambiguierten Kohorten

Einige Probleme. . .

- Perl-Modul *Lingua::Tokenize*
  - Tokenisierung ist grundlegend, aber nicht trivial!
- Morphologische Analyse
  - Basiert auf *sprachspezifischen* Lexika
  - Offene Klassen können nicht vollständig im Lexikon erfasst werden
  - eine Modellierung dieser Klassen durch einen *Guesser* ist nur bedingt möglich
  - Akzeptieren alternativer Schreibweisen (orthographische Variation)