# An Open Source Tool for Partial Parsing and Morphosyntactic Disambiguation

Andreas Völlger
paper from Adam Przepirkowski and Aleksander Buczynski

25. Oktober 2010

# table of contents

# What to expect

- formalism for simultaneous partial parsing
- and morphosyntactic disambiguation

# What to expect

- formalism for simultaneous partial parsing
- and morphosyntactic disambiguation
- unified in ♠

# What to expect

- formalism for simultaneous partial parsing
- and morphosyntactic disambiguation
- unified in ♠

# common pipeline approach

- first morphosyntactic tagging accomplished
- second shallow or partial parsing
- syntactic parsers differ in what they assume (disambiguated/non-disambiguated)
- parsing systems usually allow ambiguous input
- shallow/partial parsers usually expect fully disambiguated input

# observations

- morphosyntactic tagging & shallow parsing inform each other
- should be performed parallel, no sequence

- rules often implicitly encode same linguistic intuitions
- formalism needed - encode disambiguation and structure in one rule

## Code - Basic Format

Rule: “some rule id here“
Left: ;
Match: [pos~~“prep“][base~“co|kto“];
Right: ;
Eval: unify(case,1,2); group(PG,1,2);

- find sequence of two tokens, first unamb. preposition, second form of lexeme ’co’ (what) or ’kto’ (who)
- if interpretation of tokens with same case
- mark as syntactic group of type PG (prep. group), syntactic head = 1, semantic head = 2
- RegExp quantifiers possible

## Code - Basic Format

Rule: “some rule id here“
Left: ;
Match: [pos~~“prep“][base~“co|kto“];
Right: ;
Eval: unify(case,1,2); group(PG,1,2);

- find sequence of two tokens, first unamb. preposition, second form of lexeme ’co’ (what) or ’kto’ (who)
- if interpretation of tokens with same case
- mark as syntactic group of type PG (prep. group), syntactic head = 1, semantic head = 2
- RegExp quantifiers possible

**token specification**

Left, Match, Right same syntax and semantic

Code

```
Match: [pos~~"subst"];
```

all morphosyntactic interpretations of token are nominial

Code

```
Match: [pos~"subst"];
```

there exists a nominial interpretation

## group specification

**Code**

```
[semh=[pos~"subst"]]
```

syntactic group with semantic head is a token with only nominial interpretations

**following specification**

**Code**

```
ns; sb; se;
```

ns = no space;
sb = setence begin;
se = sentence end;

*take a deep breath - example*

## group specification

**Code**

```
[semh=[pos~"subst"]]
```

syntactic group with semantic head is a token with only nominial interpretations

**following specification**

**Code**

```
ns; sb; se;
```

ns = no space;
sb = setence begin;
se = sentence end;

*take a deep breath - example*

## Code

```
[pos~~“adv“] ([pos~~“prep“] [pos~“subst“] ns? [pos~“interp“]? se
| [synh=[pos~~“prep“]])
```

- find adverb, followed by prep. group

## Code

[pos~~“adv“] ([pos~~“prep“] [pos~“subst“] ns? [pos~“interp“]? se | [synh=[pos~~“prep“]])

- find adverb, followed by prep. group
- prep. group either sequence of disambiguated prep and possible noun at sentence end

## Code

[pos~~“adv“] ([pos~~“prep“] [pos~“subst“] ns? [pos~“interp“]? se | [synh=[pos~~“prep“]])

- find adverb, followed by prep. group
- prep. group either sequence of disambiguated prep and possible noun at sentence end
- or already recognised prep. group

## Code

[pos~~“adv“] ([pos~~“prep“] [pos~“subst“] ns? [pos~“interp“]? se | [synh=[pos~~“prep“]])

- find adverb, followed by prep. group
- prep. group either sequence of disambiguated prep and possible noun at sentence end
- or already recognised prep. group

- evaluating to true/false (false aborts rule)
- morphosyntactic actions like add, delete
- syntactic actions like unify, agree
- possible functionalities: agree, unify, delete, leave, add, set, word, group

# File Format

```
< tok >
        <orth>Po</orth>
                <lex><base>po</base><ctag>prep:acc</ctag></lex>
                <lex><base>po</base><ctag>prep:loc</ctag></lex>
< /tok >

< tok >
        <orth>co</orth>
                <lex><base>co</base><ctag>subst:sg:nom:n</ctag></lex>
                <lex><base>co</base><ctag>subst:sg:acc:n</ctag></lex>
< /tok >
```

# Algorithm Overview

- parser loads tagset
- parser loads rules
- parser compile rules
- parser maintain two representations for sentences

  **object oriented syntactic tree**, used for easy manipulation
  **string representation** used for regexp matching (not human readable!)

# Efficiency

- 167 rules of varying complexity
- 34MB XML input file
- more than 174.000 segments, 16.000 sentences
- Intel Core2Duo T7200
- processing within 4 minutes, 700 words per second
- within process 21.000 syntactic words, 22.000 syntactic groups

# Conclusion

- formalism allows to encode morphosyntactic disambiguation and shallow parsing in the same formalism
- possibly in the same rule
- more flexible than usual shallow parsing formalisms (assume disambiguated input)
- also than usual unification-based formalisms (couple disambiguation via unification)
- flexibility makes ♠ language independent!

# Thanks for listening