# WCCL

## A [morpho-syntactic feature] toolkit

*Adam Radziszewski*
*Adam Wardyński*
*Tomasz Śniatowski*

Institute of Informatics
Wrocław University of Technology

Work financed by Innovative Economy Programme POIG.01.01.02-14-013/09

# Outline

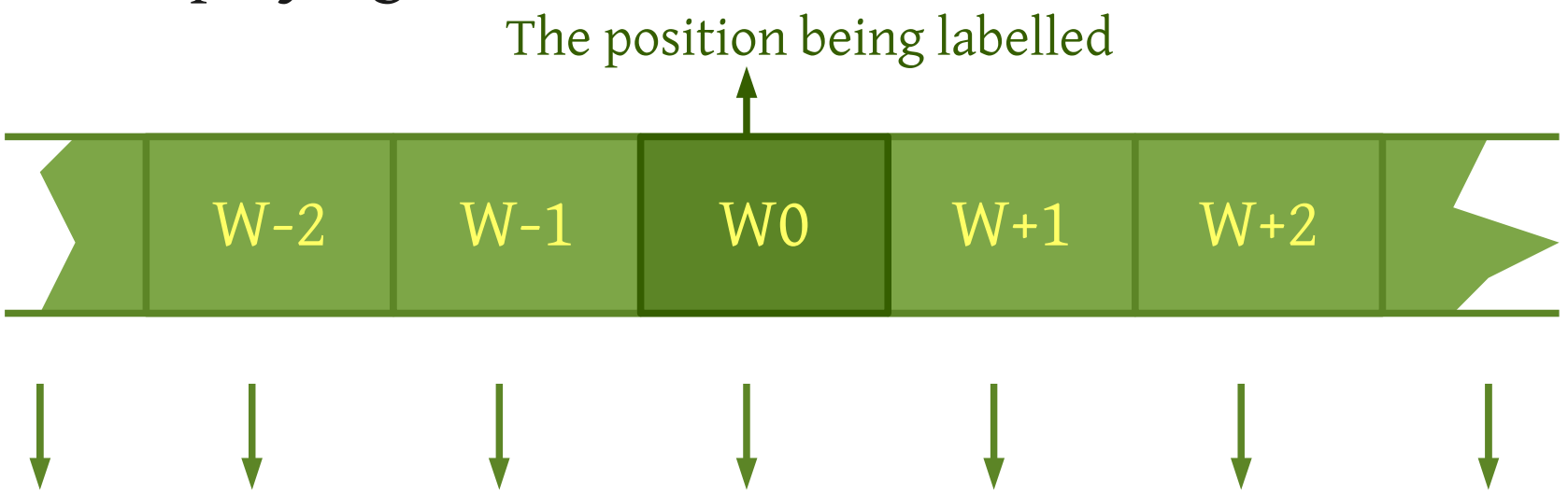
- Background
- The toolkit
- Formalism overview
- Examples
- Application: yet another tagger for Polish
- Conclusions

# Background

- Suppose we want to make a new tagger (chunker, NER module) for Polish based on an ML classifier
- We've got a rich library of classifier implementations
  - In fact, there are many such libraries, e.g. Orange, NLTK, WEKA, scikits.learn, lexcsd.classify, TiMBL
- We've got a working morphological analyser and corpus I/O routines
  - E.g. Morfeusz SGJP and MACA package to provide simple API and I/O routines
- We'd like to write as little code as possible, preferably in a high-level language to test our concept

# Background (2)

A typical approach to sequence labelling (tagging, chunking) when employing ML classifiers




Contextual information taken from local neighbourhood of the position being labelled. Represented as fixed-width vector of *feature* values, e.g.:

| | | | | |
|---|---|---|---|---|
| FORM[-2] | FORM[-1] | FORM[0] | FORM[1] | FORM[2] |
| POS[-2] | POS[-1] | POS[0] | POS[1] | POS[2] |

# Importance of features

- Slavic languages
  - Free word order and rich inflection
  - Syntactic dependencies manifested in morpho-syntactic properties of word forms
- Successful taggers for Polish decompose tags into parts
- Explicit tests for agreement help in chunking and extraction of distributional similarity measure
- Some knowledge engineering required to construct features
- Reusability of features and feature extraction module

# Our solution

- WCCL — *Wrocław Corpus Constraint Language*
- Formalism for feature extraction/construction
- Conceptually based on JOSKIPI formalism (TaKIPI tagger)
- New implementation
  - A universal toolkit: command-line utils applicable for pipeline processing
  - Library with simple C++ and Python APIs for rapid NLP application development
  - Based on Corpus2, compatible with MACA (configurable tagset, morph analysis, tokenisation, Unicode, XML)

# Motivating example

```
$ echo "Boję się myszy." \
 | maca-analyse morfeusz-nkjp
Boję            newline
    bać         fin:sg:pri:imperf
    boja        subst:sg:acc:f
się             space
    się         qub
myszy           space
    mysz        subst:sg:gen:f
    mysz        subst:sg:dat:f
    mysz        subst:sg:loc:f
    mysz        subst:sg:voc:f
    mysz        subst:pl:nom:f
    mysz        subst:pl:gen:f
    mysz        subst:pl:acc:f
    mysz        subst:pl:voc:f
.               none
    .           interp
```

```
$ echo "Boję się myszy." \
 | maca-analyse morfeusz-nkjp \
 | wccl-run - -i plain \
   class[-1] class[0] cas[0]

# orth   class[-1]    class[0]     cas[0]
1 Boję   {}           {fin,subst}  {acc}
2 się    {fin,subst}  {qub}        {}
3 myszy  {qub}        {subst}      {nom,gen,dat,acc,loc,voc}
4 .      {subst}      {interp}     {}
```

# Basics of the formalism

- Functional expressions
- Operating on morphologically analysed sentences
- Strongly typed
  - Set of symbols (domain dependent on tagset, e.g. {nom,acc})
  - Set of strings ["tak","nie"]
  - Boolean (True, False)
  - Position relative to the “centre”: -1, 0 or absolute: begin, end
- Variables over the 4 data types
- Functions with side-effects: iteration and variable assignment

# Tagset-related functions

- POS mnemonics and attribute values become valid constants
  {adv, subst}
  {nom, acc, inst}
- Attribute mnemonics become functions

  cas[0]

  in(cas[-1],{nom, acc})

## NKJP tagset def file

```
[ATTR]
nmb    sg pl
cas    nom gen dat acc inst loc
       voc
gnd    m1 m2 m3 f n
per    pri sec ter
deg    pos com sup
…
[POS]
adv      [deg]
subst    nmb cas gnd
```

# Predicates / constraints

- Logical connectives and set relation tests
- Tests for morphological agreement on given attributes
  - Simple point-to-point agreement agrpp(-1,1, {nmb,gnd,cas})
  - Agreement throughout a range agr(0,end, {nmb,gnd,cas})
- Search operators mimicking predicates
  - Side-effects: changing value of the iteration variable
  - llook and rlook — look for a position satisfying a condition
  - only, atleast — quantifiers

# Example: iteration

```
if(
   rlook(0, end, $P,
      in({subst}, class[$P])
   ),
   base[$P]
)
```

| # | orth | class[0] | base[0] | **if(rlook(…), base[$P])** |
|---|---|---|---|---|
| 1 | Dwie | {num} | ["dwa"] | ["Czeszka"] |
| 2 | ładne | {adj} | ["ładny"] | ["Czeszka"] |
| 3 | Czeszki | {subst} | ["Czeszka"] | ["Czeszka"] |
| 4 | siedziały | {praet} | ["siedzieć"] | ["skoda"] |
| 5 | w | {prep} | ["w"] | ["skoda"] |
| 6 | skodzie | {subst} | ["skoda"] | ["skoda"] |
| 7 | i | {conj,interj,qub} | ["i"] | ["pilzner"] |
| 8 | sączyły | {praet} | ["sączyć"] | ["pilzner"] |
| 9 | pilznera | {subst} | ["pilzner"] | ["pilzner"] |
| 10 | . | {interp} | ["."] | [] |

# Example: WCCL file for a tagger

```
import("nkjp1.lex", "fq")
@ "default" (
   class[-3]; class[-2]; class[-1]; class[0]; class[1]; class[2];
   cas[-3]; cas[-2]; cas[-1]; cas[0]; cas[1]; cas[2];
   gnd[-3]; gnd[-2]; gnd[-1]; gnd[0]; gnd[1]; gnd[2];
   nmb[-3]; nmb[-2]; nmb[-1]; nmb[0]; nmb[1]; nmb[2];
   lex(lower(orth[-3]), "fq");
   lex(lower(orth[-2]), "fq");
   lex(lower(orth[-1]), "fq");
   lex(lower(orth[0]), "fq");
   lex(lower(orth[1]), "fq");
   lex(lower(orth[2]), "fq");
   agrpp(-1,0,{nmb,gnd,cas});
   agrpp(0,1,{nmb,gnd,cas});
   and(inside(-2), wagr(-2,0,{nmb,gnd,cas}));
   and(inside(-1), inside(1), wagr(-1,1,{nmb,gnd,cas}));
   and(inside(2), wagr(0,2,{nmb,gnd,cas}))
)
```

# Instead of a case study

- Memory-based tagger for Polish
- Features from the previous slide
- State-of-the-art performance
  - 93.04% on NKJP
  - Pantera: 92.92%
  - Voting (memory-based, Pantera, RFTagger): 94.26%
- Implemented in Python
  - 421 lines of code (TaKIPI: 6435, Pantera: 8475)
  - WCCL Python wrappers
  - Classifier implementation: TiMBL API

# Conclusions

- GPL'ed: nlp.pwr.wroc.pl/redmine/projects/joskipi/wiki
- Already useful toolkit
  - The new tagger (via Python API)
  - Used for extraction distributional similarity measure (via command-line utils)
- Further work planned
  - Direct support for MULTEXT tagsets
  - Test as a corpus querying system

Thank you for your attention!