

Looking for Errors

A Declarative Formalism for Resource-Adaptive
Language Checking



Bredenkamp, Crysmann & Petrea
DFKI, Saarbrücken, Germany

contact: crysmann@dfki.de



Overview

- ❑ Main aim:
 - provide tools for rapid development of grammar and controlled language checkers

- ❑ Outline:
 - Background:
Phenomenon-based language checking
 - Integration of backend NLP components
 - Declarative error specification
 - Development environment



Background

- ❑ Phenomenon-based language checking
 - Based on the notion of “triggers”
 - Identification of candidates (as cheaply as possible)
 - Confirmation by focussed deeper processing
 - Specific rules for specific error types
- ❑ Motivation:
 - Scarce distribution of grammar errors
 - Use appropriate technology (resource-adaptive)
 - Controlled languages always client-specific & positively defined formal grammars unavailable



Multiple NLP Backends I

- ❑ Combine virtues of different backend technologies, e.g.
 - Speed
 - Robustness
 - Precision
- ❑ Flexible combination of different types of linguistic resources
- ❑ Overall robustness independent of individual backend components (resource-independent)



Multiple NLP Backends II

- ❑ Tokenisation
- ❑ Morphological Analysis (MULTEXT mmorph)
- ❑ Probabilistic POS Tagging (TnT; Brants 1999)
- ❑ Probabilistic NP Chunking (Skut/Brants 1998)
- ❑ Topological Parsing (Braun et al. 2000)
- ❑ Deep NLP (work in progress)



Declarative Specification

- ❑ Permits tight integration of multiple backends (descriptions of linguistic objects)
- ❑ Facilitates division of labour between linguistic tasks and engineering aspects
 - Quickly respond to client-specific demands
 - Individual checking components benefit from global optimisations
- ❑ Formalism reflects basic checking architecture



Specification Formalism I

- ❑ Token enriched with multiple annotations (token, lemma, part-of-speech, morphology)
- ❑ Word-level annotations represented as feature structures
- ❑ FS-pattern matching
- ❑ Bottom-up integration of (partial) parsing



Specification Formalism II

- ❑ #ERROR mWn
- ❑ #OBJS
 - @poss ::= [TOK “^[MmDdSs]eines”];
 - @wissens ::= [TOK “^Wissens\$”];
 - @nach ::= [TOK “nach”];

 - @prep ::= [POS “^APPR(ART)?\$”];
 - @dat_obj ::= [POS “^(ART|ADJA)\$”
MORPH.READING.INFLECTION.case “dat”];
- ❑ #RULES
 - TRIGGER(70) ==
@poss^1 @wissens^2 @nach^3 ->
\$meines^1 \$Wissens^2 \$nach^3;

 - NEG_EV(10) ==
\$nach [*] @dat_obj^1 -> \$dat^1;

 - NEG_EV(30) ==
&&cin(\$nach, “^PP\$^1) && cin(\$dat, ^1);

 - POS_EV(30) ==
\$nach @prep;

❑ Object definitions

❑ Rules



Specification Formalism II

- ❑ #ERROR mWn
- ❑ #OBJS
 - @poss ::= [TOK “^[MmDdSs]eines”];
 - @wissens ::= [TOK “^Wissens\$”];
 - @nach ::= [TOK “nach”];

 - @prep ::= [POS “^APPR(ART)?\$”];
 - @dat_obj ::= [POS “^(ART|ADJA)\$”
MORPH.READING.INFLECTION.case “dat”];
- ❑ #RULES
 - TRIGGER(70) ==
@poss^1 @wissens^2 @nach^3 ->
\$meines^1 \$Wissens^2 \$nach^3;

 - NEG_EV(10) ==
\$nach []* @dat_obj^1 -> \$dat^1;

 - NEG_EV(30) ==
&&cin(\$nach, “^PP\$^1) && cin(\$dat, ^1);

 - POS_EV(30) ==
\$nach @prep;

❑ Object definitions

❑ Rules



Specification Formalism II

❑ #ERROR mWn

❑ #OBJS

@poss ::= [TOK "^[MmDdSs]eines"];
@wissens ::= [TOK "^[Wissens\$"];
@nach ::= [TOK "nach"];

@prep ::= [POS "^[APPR(ART)?\$];
@dat_obj ::= [POS "^(ART|ADJA)\$"
MORPH.READING.INFLECTION.case "dat"];
❑ Object definitions

❑ #RULES

TRIGGER(70) ==

@poss^1 @wissens^2 @nach^3 ->
\$meines^1 \$Wissens^2 \$nach^3;

NEG_EV(10) ==

\$nach []* @dat_obj^1 -> \$dat^1;

NEG_EV(30) ==

&&cin(\$nach,"^PP\$^1) && cin(\$dat,^1);

POS_EV(30) ==

\$nach @prep;

❑ Rules

- Trigger (determine error candidates)



Specification Formalism II

❑ #ERROR mWn

❑ #OBJS

@poss ::= [TOK "^[MmDdSs]eines"];

@wissens ::= [TOK "^Wissens\$"];

@nach ::= [TOK "nach "];

@prep ::= [POS "^APPR(ART)?\$];

@dat_obj ::= [POS "^(ART|ADJA)\$"

MORPH.READING.INFLECTION.case "dat"];

❑ #RULES

TRIGGER(70) ==

@poss^1 @wissens^2 @nach^3 ->

\$meines^1 \$Wissens^2 \$nach^3;

NEG_EV(10) ==

\$nach []* @dat_obj^1 -> \$dat^1;

NEG_EV(30) ==

&&cin(\$nach,"^PP\$^1) && cin(\$dat,^1);

POS_EV(30) ==

\$nach @prep;

❑ Object definitions

❑ Rules

• Trigger

• Negative evidence
(discard false alarms)



Specification Formalism II

- ❑ #ERROR mWn
- ❑ #OBJS
@poss ::= [TOK "^[MmDdSs]eines"];
@wissens ::= [TOK "^[Wissens\$"];
@nach ::= [TOK "nach"];

@prep ::= [POS "^[APPR(ART)?\$];
@dat_obj ::= [POS "^(ART|ADJA)\$"
MORPH.READING.INFLECTION.case "dat"];

- ❑ Object definitions

- ❑ #RULES
TRIGGER(70) ==
@poss^1@wissens^2 @nach^3 ->
\$meines^1 \$Wissens^2 \$nach^3;

NEG_EV(10) ==
\$nach []* @dat_obj^1-> \$dat^1;

NEG_EV(30) ==
&&cin(\$nach,"^APP\$^1) && cin(\$dat,^1);

POS_EV(30) ==
\$nach @prep;

- ❑ Rules

- Trigger
- Negative evidence
- Positive evidence
(confirm error candidates)



Specification Formalism III

- ❑ #ERROR mWn
- ❑ #OBJS
 - @poss ::= [TOK “^[MmDdSs]eines”];
 - @wissens ::= [TOK “^Wissens\$”];
 - @nach ::= [TOK “nach”];

 - @prep ::= [POS “^APPR(ART)?\$”];
 - @dat_obj ::= [POS “^(ART|ADJA)\$”
MORPH.READING.INFLECTION.case “dat”];
- ❑ #RULES
 - TRIGGER(70) ==
 - @poss^1 @wissens^2 @nach^3 ->
\$meines^1 \$Wissens^2 \$nach^3;

 - NEG_EV(10) ==
 - \$nach []* @dat_obj^1 -> \$dat^1;

 - NEG_EV(30) ==
 - &&cin(\$nach, “^PP\$^1) && cin(\$dat, ^1);

 - POS_EV(30) ==
 - \$nach @prep;

❑ Resources

❑ Expressiveness



Specification Formalism III

❑ #ERROR mWn

❑ #OBJS

@poss ::= [TOK “^[MmDdSs]eines”];

@wissens ::= [TOK “^Wissens\$”];

@nach ::= [TOK “nach”];

@prep ::= [POS “^APPR(ART)?\$”];

@dat_obj ::= [POS “^(ART|ADJA)\$”

MORPH.READING.INFLECTION.case “dat”];

❑ #RULES

TRIGGER(70) ==

@poss^1 @wissens^2 @nach^3 ->

\$meines^1 \$Wissens^2 \$nach^3;

NEG_EV(10) ==

\$nach []* @dat_obj^1 -> \$dat^1;

NEG_EV(30) ==

&&cin(\$nach, “^PP\$^1) && cin(\$dat, ^1);

POS_EV(30) ==

\$nach @prep;

❑ Resources

- Token (robust)

❑ Expressiveness



Specification Formalism III

- ❑ #ERROR mWn
- ❑ #OBJS
 - @poss ::= [TOK “^[MmDdSs]eines”];
 - @wissens ::= [TOK “^Wissens\$”];
 - @nach ::= [TOK “nach”];
 - @prep ::= [POS “^APPR(ART)?\$”];
 - @dat_obj ::= [POS “^(ART|ADJA)\$”
MORPH.READING.INFLECTION.case “dat”];

- ❑ Resources

- Token (robust)
- POS tags (robust)

- ❑ #RULES
 - TRIGGER(70) ==
 - @poss^1 @wissens^2 @nach^3 ->
 - \$meines^1 \$Wissens^2 \$nach^3;

- ❑ Expressiveness

NEG_EV(10) ==
\$nach []* @dat_obj^1 -> \$dat^1;

NEG_EV(30) ==
&&cin(\$nach, “^PP\$^1) && cin(\$dat, ^1);

POS_EV(30) ==
\$nach @prep;



Specification Formalism III

- ❑ #ERROR mWn
- ❑ #OBJS

```
@poss ::= [TOK "^[MmDdSs]eines"];
@wissens ::= [TOK "^[Wissens$"];
@nach ::= [TOK "nach" ];

@prep ::= [POS "^[APPR(ART)?$"];
@dat_obj ::= [POS "^(ART|ADJA)$"
MORPH.READING.INFLECTION.case "dat"];
```
- ❑ #RULES

```
TRIGGER(70) ==
@poss^1@wissens^2 @nach^3 ->
  $meines^1 $Wissens^2 $nach^3;

NEG_EV(10) ==
$nach []* @dat_obj^1-> $dat^1;

NEG_EV(30) ==
&&cin($nach,"^PP$^1) && cin($dat,^1);

POS_EV(30) ==
$nach @prep;
```

❑ Resources

- Token (robust)
- POS tags (robust)
- Morphological analysis

❑ Expressiveness



Specification Formalism III

- ❑ #ERROR mWn
- ❑ #OBJS

```
@poss ::= [TOK "^[MmDdSs]eines"];
@wissens ::= [TOK "^[Wissens$"];
@nach ::= [TOK "nach" ];

@prep ::= [POS "^[APPR(ART)?$"];
@dat_obj ::= [POS "^(ART|ADJA)$"
MORPH.READING.INFLECTION.case "dat"];

#RULES
TRIGGER(70) ==
@poss^1 @wissens^2 @nach^3 ->
  $meines^1 $Wissens^2 $nach^3;

NEG_EV(10) ==
$nach []* @dat_obj^1 -> $dat^1;

NEG_EV(30) ==
&&cin($nach,"^PP$^1) && cin($dat,^1);

POS_EV(30) ==
$nach @prep;
```

❑ Resources

- Token (robust)
- POS tags (robust)
- Morphological analysis

❑ Expressiveness

- regular expressions over feature values



Specification Formalism III

- ❑ #ERROR mWn
- ❑ #OBJS

```
@poss ::= [TOK "^[MmDdSs]eines"];
@wissens ::= [TOK "^[Wissens$"];
@nach ::= [TOK "nach" ];

@prep ::= [POS "^[APPR(ART)?$"];
@dat_obj ::= [POS "^(ART|ADJA)$"
MORPH.READING.INFLECTION.case "dat"];

#RULES
TRIGGER(70) ==
@poss^1 @wissens^2 @nach^3 ->
  $meines^1 $Wissens^2 $nach^3;

NEG_EV(10) ==
$nach []* @dat_obj^1 -> $dat^1;

NEG_EV(30) ==
&&cin($nach,"^PP$^1) && cin($dat,^1);

POS_EV(30) ==
$nach @prep;
```

❑ Resources

- Token (robust)
- POS tags (robust)
- Morphological analysis

❑ Expressiveness

- regular expressions over feature values
- negation/disjunction over feature structures



Specification Formalism IV

- ❑ #ERROR mWn
- ❑ #OBJS
 - @poss ::= [TOK “^[MmDdSs]eines”];
 - @wissens ::= [TOK “^Wissens\$”];
 - @nach ::= [TOK “nach”];

 - @prep ::= [POS “^APPR(ART)?\$”];
 - @dat_obj ::= [POS “^(ART|ADJA)\$”
MORPH.READING.INFLECTION.case “dat”];
- ❑ #RULES
 - TRIGGER(70) ==
 - @poss^1 @wissens^2 @nach^3 ->
\$meines^1 \$Wissens^2 \$nach^3;

 - NEG_EV(10) ==
 - \$nach []* @dat_obj^1 -> \$dat^1;

 - NEG_EV(30) ==
 - &&cin(\$nach, “^PP\$^1) && cin(\$dat, ^1);

 - POS_EV(30) ==
 - \$nach @prep;

❑ Rules



Specification Formalism IV

- ❑ #ERROR mWn
- ❑ #OBJS
 - @poss ::= [TOK “^[MmDdSs]eines”];
 - @wissens ::= [TOK “^Wissens\$”];
 - @nach ::= [TOK “nach”];

 - @prep ::= [POS “^APPR(ART)?\$”];
 - @dat_obj ::= [POS “^(ART|ADJA)\$”
MORPH.READING.INFLECTION.case “dat”];
- ❑ #RULES
 - TRIGGER(70) ==
 - @poss^1 @wissens^2 @nach^3 ->
 - \$meines^1 \$Wissens^2 \$nach^3;

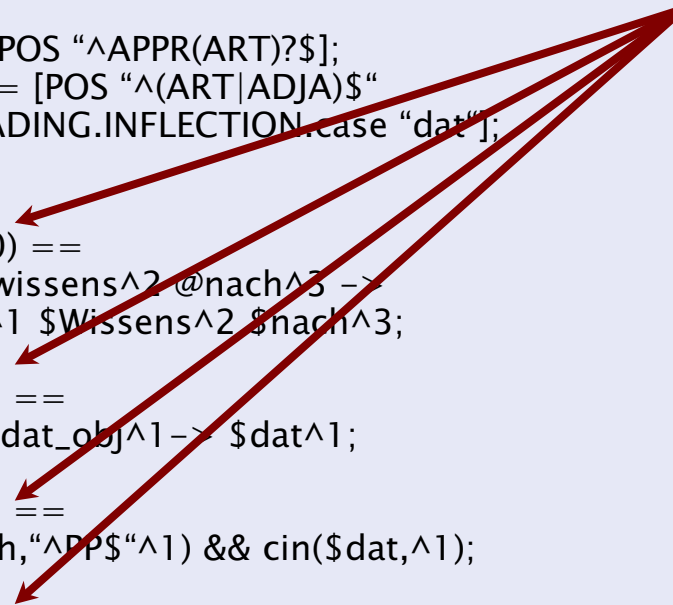
 - NEG_EV(10) ==
 - \$nach []* @dat_obj^1 -> \$dat^1;

 - NEG_EV(30) ==
 - &&cin(\$nach, “^PP\$^1) && cin(\$dat, ^1);

 - POS_EV(30) ==
 - \$nach @prep;

❑ Rules

- Weighted with confidence measure





Specification Formalism IV

- ❑ #ERROR mWn
- ❑ #OBJS
 - @poss ::= [TOK “^[MmDdSs]eines”];
 - @wissens ::= [TOK “^Wissens\$”];
 - @nach ::= [TOK “nach”];

 - @prep ::= [POS “^APPR(ART)?\$”];
 - @dat_obj ::= [POS “^(ART|ADJA)\$”
MORPH.READING.INFLECTION.case “dat”];
- ❑ #RULES
 - TRIGGER(70) ==
 - @poss^1 @wissens^2 @nach^3 ->
 - \$meines^1 \$Wissens^2 \$nach^3;

 - NEG_EV(10) ==
 - \$nach []* @dat_obj^1 -> \$dat^1;

 - NEG_EV(30) ==
 - &&cin(\$nach, “^PP\$^1) && cin(\$dat, ^1);

 - POS_EV(30) ==
 - \$nach @prep;

❑ Rules

- Weighted with confidence measure
- Regexp over feature structures (LHS)



Specification Formalism IV

- ❑ #ERROR mWn
- ❑ #OBJS
 - @poss ::= [TOK “^[MmDdSs]eines”];
 - @wissens ::= [TOK “^Wissens\$”];
 - @nach ::= [TOK “nach”];

 - @prep ::= [POS “^APPR(ART)?\$”];
 - @dat_obj ::= [POS “^(ART|ADJA)\$”
MORPH.READING.INFLECTION.case “dat”];
- ❑ #RULES
 - TRIGGER(70) ==
 - @poss^1 @wissens^2 @nach^3 ->
 - \$meines^1 \$Wissens^2 \$nach^3;

 - NEG_EV(10) ==
 - \$nach []* @dat_obj^1 -> \$dat^1;

 - NEG_EV(30) ==
 - &&cin(\$nach, “^PP\$^1) && cin(\$dat, ^1);

 - POS_EV(30) ==
 - \$nach @prep;

❑ Rules

- Weighted with confidence measure
- Regexp over feature structures (LHS)
- Match assigned to named variables (RHS)



Specification Formalism IV

- ❑ #ERROR mWn
- ❑ #OBJS
 - @poss ::= [TOK “^[MmDdSs]eines”];
 - @wissens ::= [TOK “^Wissens\$”];
 - @nach ::= [TOK “nach”];

 - @prep ::= [POS “^APPR(ART)?\$”];
 - @dat_obj ::= [POS “^(ART|ADJA)\$”
MORPH.READING.INFLECTION.case “dat”];
- ❑ #RULES
 - TRIGGER(70) ==
 - @poss^1 @wissens^2 @nach^3 ->
 - \$meines^1 \$Wissens^2 \$nach^3;

 - NEG_EV(10) ==
 - \$nach []* @dat_obj^1 -> \$dat^1;

 - NEG_EV(30) ==
 - &&cin(\$nach, “^PP\$^1) && cin(\$dat, ^1);

 - POS_EV(30) ==
 - \$nach @prep;

❑ Rules

- Weighted with confidence measure
- Regexp over feature structures (LHS)
- Match assigned to named variables (RHS)
 - assignment by local coindexation (^)



Specification Formalism IV

- ❑ #ERROR mWn
- ❑ #OBJS

```
@poss ::= [TOK "^[MmDdSs]eines"];
@wissens ::= [TOK "^[Wissens$"];
@nach ::= [TOK "nach" ];

@prep ::= [POS "^[APPR(ART)?$"];
@dat_obj ::= [POS "^(ART|ADJA)$"
MORPH.READING.INFLECTION.case "dat"];
```
- ❑ #RULES

```
TRIGGER(70) ==
@poss^1@wissens^2 @nach^3 ->
  $meines^1 $Wissens^2 $nach^3;

NEG_EV(10) ==
$nach []* @dat_obj^1-> $dat^1;

NEG_EV(30) ==
&&cin($nach,"^APP$^1) && cin($dat,^1);

POS_EV(30) ==
$nach @prep;
```

❑ Rules

- Weighted with confidence measure
- Regexp over feature structures
- Match assigned to named variables (RHS)
- Evidence rules may target named variables



Specification Formalism IV

- ❑ #ERROR mWn
- ❑ #OBJS

```
@poss ::= [TOK "^[MmDdSs]eines"];
@wissens ::= [TOK "^[Wissens$"];
@nach ::= [TOK "nach" ];

@prep ::= [POS "^[APPR(ART)?$"];
@dat_obj ::= [POS "^(ART|ADJA)$"
MORPH.READING.INFLECTION.case "dat"];
```
- ❑ #RULES

```
TRIGGER(70) ==
@poss^1@wissens^2 @nach^3 ->
  $meines^1 $Wissens^2 $nach^3;

NEG_EV(10) ==
  $nach []* @dat_obj^1-> $dat^1;

NEG_EV(30) ==
  &&cin($nach,"^APP$^1) && cin($dat,^1);

POS_EV(30) ==
  $nach @prep;
```

❑ Rules

- Weighted with confidence measure
- Regexp over feature structures
- Match assigned to named variables (RHS)
- Evidence rules may target named variables
 - interface between trigger and validation rules



Specification Formalism IV

- ❑ #ERROR mWn
- ❑ #OBJS
 - @poss ::= [TOK “^[MmDdSs]eines”];
 - @wissens ::= [TOK “^Wissens\$”];
 - @nach ::= [TOK “nach”];

 - @prep ::= [POS “^APPR(ART)?\$”];
 - @dat_obj ::= [POS “^(ART|ADJA)\$”
MORPH.READING.INFLECTION.case “dat”];
- ❑ #RULES
 - TRIGGER(70) ==
@poss^1 @wissens^2 @nach^3 ->
\$meines^1 \$Wissens^2 \$nach^3;

 - NEG_EV(10) ==
\$nach []* @dat_obj^1 -> \$dat^1;

 - NEG_EV(30) ==
&&cin(\$nach, “^PP\$^1) && cin(\$dat, ^1);

 - POS_EV(30) ==
\$nach @prep;

❑ Rules

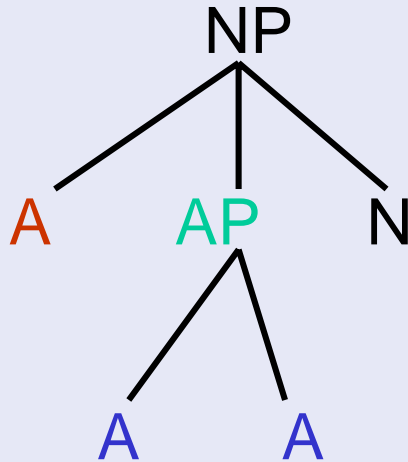
- Weighted with confidence measure
- Regexp over feature structures
- Match assigned to named variables
- Evidence rules may target named variables
- Relational constraints



Tree constraints I

- ❑ Currently:
 - NP/PP chunks: `cin`, `cancestor`, `cpath`
 - Sentence topology: `sin`, `sancestor`, `spath`
- ❑ Set of 3 constraints per tree backend:
 - $x_{in}(Node, NT\text{-}pattern)$
determine minimal (lowest) NT node dominating *Node* that matches the regexp in *NT-pattern*
 - $x_{ancestor}(Terminal1, Terminal2, NT\text{-}pattern)$
determine minimal node dominating both *Terminal1* and *Terminal2* that matches the regexp in *NT-pattern*
 - $x_{path}(Terminal, NT\text{-}Node, NT\text{-}pattern)$
restrict the path from *Terminal* to *NT-Node* to contain a node matching the regexp in *NT-pattern*

Tree constraints II



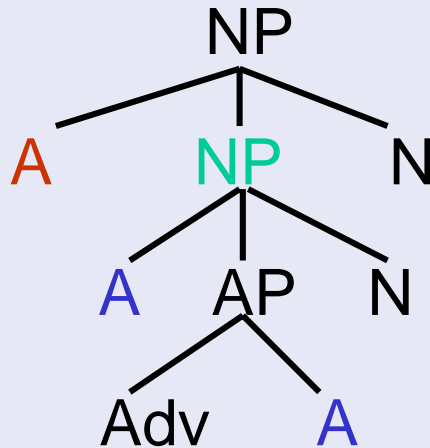
```
@adj ::= [POS "^A$"];
```

```
@adj^1 []* @adj^2
  && cin(^1, ""^3)
  && cin(^2, ^3);
```

□ Standard tree configurations

- Immediate sisterhood
- Domain-locality (extended sisterhood)
- Government
- C-command
- C-command & bounding nodes

Tree constraints II

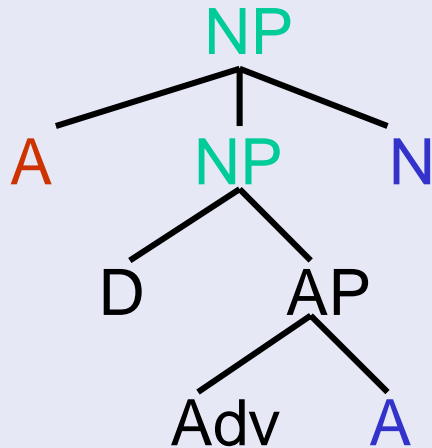


```
@adj ::= [POS "^A$"];
```

```
@adj^1 []* @adj^2
  && cin(^1, "NP"^3)
  && cin(^2, ^3);
```

- Standard tree configurations
 - Immediate sisterhood
 - Domain-locality (extended sisterhood)
 - Government
 - C-command
 - C-command & bounding nodes

Tree constraints II

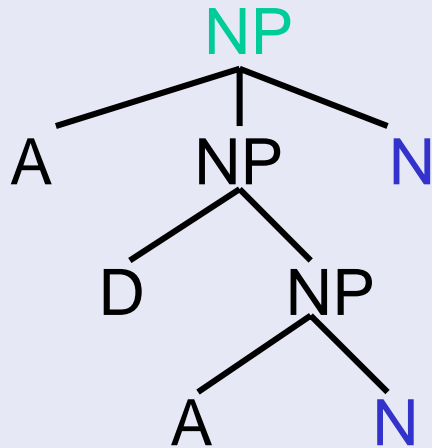


```
@adj ::= [POS "^A$"];
@noun ::= [POS "^N$"];
```

```
@adj^1 []* @noun^2
  && cin(^1, "NP"^3)
  && cin(^2, "NP"^4)
  && cin(^3, ^4);
```

- Standard tree configurations
 - Immediate sisterhood
 - Domain-locality (extended sisterhood)
 - **Government**
 - C-command
 - C-command & bounding nodes

Tree constraints II

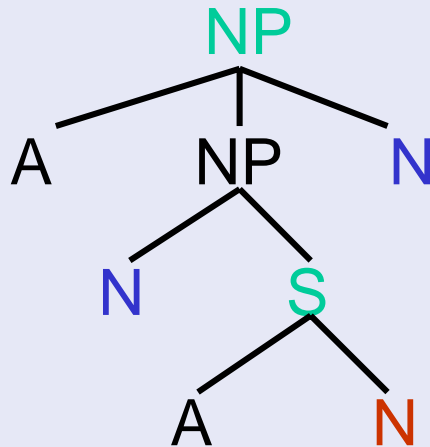


```
@noun ::= [POS "^N$" ] ;
```

```
@noun^1 [ ]* @noun^2
  && cin(^2, "NP"^3)
  && cancestor(^1, ^2, ^3) ;
```

- Standard tree configurations
 - Immediate sisterhood
 - Domain-locality (extended sisterhood)
 - Government
 - C-command
 - C-command & bounding nodes

Tree constraints II



```
@noun ::= [POS "^N$"];
```

```
@noun^1 []* @noun^2
  && cin(^2, "NP"^3)
  && cancestor(^1, ^2, ^3)
  && -cpath(^1, ^3, "^S$");
```

- Standard tree configurations
 - Immediate sisterhood
 - Domain-locality (extended sisterhood)
 - Government
 - C-command
 - C-command & bounding nodes



Example error

- ❑ #ERROR mWn
- ❑ #OBJS

```
@poss ::= [TOK "^[MmDdSs]eines"];
@wissens ::= [TOK "^Wissens$"];
@nach ::= [TOK "nach" ];

@prep ::= [POS "^APPR(ART)?$"];
@dat_obj ::= [POS "^(ART|ADJA)$"
MORPH.READING.INFLECTION.case "dat"];
```
- ❑ #RULES

```
TRIGGER(70) ==
@poss^1@wissens^2 @nach^3 ->
  $meines^1 $Wissens^2 $nach^3;

NEG_EV(10) ==
$nach []* @dat_obj^1-> $dat^1;

NEG_EV(30) ==
&&cin($nach,"^PP$^1) && cin($dat,^1);

POS_EV(30) ==
$nach @prep;
```

- ❑ „meines Wissens nach“

- Erroneous blend:
„meines Wissens (gen)“
„meinem Wissen nach (dat)“
- „nach“ can be pre-/postposition
(selecting dative case)



Example error

- ❑ #ERROR mWn
- ❑ #OBJS
 - @poss ::= [TOK “^[MmDdSs]eines”];
 - @wissens ::= [TOK “^Wissens\$”];
 - @nach ::= [TOK “nach”];

 - @prep ::= [POS “^APPR(ART)?\$”];
 - @dat_obj ::= [POS “^(ART|ADJA)\$”
MORPH.READING.INFLECTION.case “dat”];
- ❑ #RULES
 - TRIGGER(70) ==
 - @poss^1 @wissens^2 @nach^3 ->
 - \$meines^1 \$Wissens^2 \$nach^3;

 - NEG_EV(10) ==
 - \$nach [*] @dat_obj^1 -> \$dat^1;

 - NEG_EV(30) ==
 - &&cin(\$nach, “^PP\$^1) && cin(\$dat, ^1);

 - POS_EV(30) ==
 - \$nach @prep;

❑ „meines Wissens nach“

- Erroneous blend:
 - „meines Wissens (gen)“
 - „meinem Wissen nach (dat)“
- „nach“ can be pre-/postposition

- Error:
 - findet [meines Wissens nach]_{PP}
 - [vor dem Essen]_{PP} statt
- False Alarm:
 - findet [meines Wissens]_{NP}
 - [nach dem Essen]_{PP} statt



Example error

- ❑ #ERROR mWn
- ❑ #OBJS

```
@poss ::= [TOK "^[MmDdSs]eines"];
@wissens ::= [TOK "^Wissens$"];
@nach ::= [TOK "nach" ];
```



```
@prep ::= [POS "^APPR(ART)?$"];
@dat_obj ::= [POS "^(ART|ADJA)$"
MORPH.READING.INFLECTION.case "dat"];
```
- ❑ #RULES

```
TRIGGER(70) ==
@poss^1@wissens^2 @nach^3 ->
  $meines^1 $Wissens^2 $nach^3;
```



```
NEG_EV(10) ==
$nach []* @dat_obj^1-> $dat^1;
```



```
NEG_EV(30) ==
&&cin($nach,"^PP$^1) && cin($dat,^1);
```



```
POS_EV(30) ==
$nach @prep;
```

❑ „meines Wissens nach“

- Erroneous blend:
„meines Wissens (gen)“
„meinem Wissen nach (dat)“
- „nach“ can be pre-/postposition
- Error:
findet [meines Wissens nach]_{PP}
[vor dem Essen]_{PP} statt
- False Alarm:
findet [meines Wissens]_{NP}
[nach dem Essen]_{PP} statt





Example error

- ❑ #ERROR mWn
- ❑ #OBJS
 - @poss ::= [TOK “^[MmDdSs]eines”];
 - @wissens ::= [TOK “^Wissens\$”];
 - @nach ::= [TOK “nach”];

 - @prep ::= [POS “^APPR(ART)?\$”];
 - @dat_obj ::= [POS “^(ART|ADJA)\$”
MORPH.READING.INFLECTION.case “dat”];
- ❑ #RULES
 - TRIGGER(70) ==
 - @poss^1 @wissens^2 @nach^3 ->
 - \$meines^1 \$Wissens^2 \$nach^3;

 - NEG_EV(10) ==
 - \$nach [*] @dat_obj^1 -> \$dat^1;

 - NEG_EV(30) ==
 - &&cin(\$nach, “^PP\$^1) && cin(\$dat, ^1);

 - POS_EV(30) ==
 - \$nach @prep;

❑ „meines Wissens nach“

- Erroneous blend:
 - „meines Wissens (gen)“
 - „meinem Wissen nach (dat)“
- „nach“ can be pre-/postposition

- Error:
 - findet [meines Wissens nach]_{PP}
 - [vor dem Essen]_{PP} statt
- False Alarm:
 - findet [meines Wissens]_{NP}
 - [nach dem Essen]_{PP} statt

Example error

```

❑ #ERROR mWn
❑ #OBJS
@poss ::= [TOK "^[MmDdSs]eines"];
@wissens ::= [TOK "^Wissens$"];
@nach ::= [TOK "nach" ];

@prep ::= [POS "^APPR(ART)?$"];
@dat_obj ::= [POS "^(ART|ADJA)$"
MORPH.READING.INFLECTION.case "dat"];

❑ #RULES
TRIGGER(70) ==
@poss^1@wissens^2 @nach^3 ->
  $meines^1 $Wissens^2 $nach^3;

NEG_EV(10) ==
$nach [* @dat_obj^1-> $dat^1;

NEG_EV(30) ==
&&cin($nach,"^PP$^1) && cin($dat,^1);

POS_EV(30) ==
$nach @prep;

```

❑ „meines Wissens nach“

- Erroneous blend:
„meines Wissens (gen)“
„meinem Wissen nach (dat)“
- „nach“ can be pre-/postposition

- Error:
findet [meines Wissens nach]_{PP}
[vor dem Essen]_{PP} statt
- False Alarm:
findet [meines Wissens]_{NP}
[nach dem Essen]_{PP} statt

Example error

```

❑ #ERROR mWn
❑ #OBJS
@poss ::= [TOK "^[MmDdSs]eines"];
@wissens ::= [TOK "^Wissens$"];
@nach ::= [TOK "nach" ];

@prep ::= [POS "^APPR(ART)?$"];
@dat_obj ::= [POS "(ART|ADJA)$"
MORPH.READING.INFLECTION.case "dat"];

❑ #RULES
TRIGGER(70) ==
@poss^1@wissens^2 @nach^3 ->
  $meines^1 $Wissens^2 $nach^3;

NEG_EV(10) ==
$nach [* @dat_obj^1-> $dat^1;

NEG_EV(30) ==
&&cin($nach,"^PP$^1) && cin($dat,^1);

POS_EV(30) ==
$nach @prep;
  
```

❑ „meines Wissens nach“

- Erroneous blend:
„meines Wissens (gen)“
„meinem Wissen nach (dat)“
- „nach“ can be pre-/postposition
- Error:
findet [meines Wissens nach]_{PP}
[vor dem Essen]_{PP} statt
- False Alarm:
findet [meines Wissens]_{NP}
[nach dem Essen]_{PP} statt



Development environment I

- Integrated environment on Unix and Windows

The screenshot shows the Flag application window with a menu bar (File, View, Check, Help) and a toolbar. The main window is divided into two panes. The left pane displays a list of 13 German sentences, some with words highlighted in red. The right pane, titled 'Results:', shows error messages for specific sentences, such as 'NP-internal agreement: Adjectives must have identical inflection' for Sentence 3.

Location: file:/C:/development/Flag/Demo/html/sab-new.html

1. Der Dokumentellauftrag wird gelöscht.
2. Er wird böse, weil ich die Vertellaufträge gelöscht habe.
3. Wenn Sie einen Wert aus neu erzeugten oder nicht gespeicherte Tabellen löschen, können Sie ihn nicht mehr ändern.
4. Wenn Sie einen Wert aus dem Bestand löschen, können Sie ihn nicht mehr ändern.f
5. Wenn Sie einen Wert aus dem aktuellem Bestand löschen, können Sie seinen Wert nicht mehr ändern.
6. Wenn sie eine Position aus der Tabelle löschen, werden ihre Werte gelöscht.
7. Bitte zuerst Werk markieren.
8. Ende des Block markieren.

9. Sie haben keine Berechtigung für das Analyseprogramm.
10. Sie haben keine Berechtigung, Positionen ohne Beziehung zu ...

11. Sollen die Dokumentenverwaltungssätze gesichert werden?
12. Möchten Sie die geänderte Werte des Dokumentinfosatzes sichern?
13. Geben Sie Ihren Namen nicht ein, wenn Sie die Meldung nicht erhalten möchten.

Results:

Sentence 3
ERROR: from 1 to 10
NP-internal agreement:
Adjectives must have
identical inflection

Sentence 5
ERROR: from 1 to 8
NP-internal agreement:
Determiner and adjective
do not agree

Sentence 8
ERROR: from 1 to 3
NP-internal agreement:
Determiner does not agree
with the noun

Sentence 12
ERROR: from 1 to 5
NP-internal agreement:
Determiner and adjective
do not agree

Done.



Development environment III

- Rule trace & chunk browser
 - NP chunks
 - Sentence topology

Wenn Sie einen Wert aus neu erzeugten oder nicht gespeicherte Tabellen löschen, können Sie ihn nicht mehr ändern.

Rules Full trace

```
50; Body: {{{-@something}}, @Adj_e##adj_l, []* , @adj_n_e##adj_r}
TRIGGER
50; Body: {{{-@something}}, @Adj_en##adj_l, []* , @adj_n_en##adj_r}
TRIGGER
50; Body: {{{-@something}}, @Adj_em##adj_l, []* , @adj_n_em##adj_r}
TRIGGER
40; Body: {{{-@something}}, @Adj_em##adj_lm, []* , @adj_n_em##adj_rm}
TRIGGER
50; Body: {{{-@something}}, @Adj_es##adj_l, []* , @adj_n_es##adj_r}
TRIGGER
70; Body: {{{-@something}}, @Adj_er##adj_l, [{{@mods}}]* , @adj_n_er##adj_r}
TRIGGER
70; Body: {{{-@something}}, @Adj_e##adj_l, [{{@mods}}]* , @adj_n_e##adj_r}
TRIGGER
70; Body: {{{-@something}}, @Adj_en##adj_l, [{{@mods}}]* , @adj_n_en##adj_r}
TRIGGER
70; Body: {{{-@something}}, @Adj_em##adj_l, [{{@mods}}]* , @adj_n_em##adj_r}
TRIGGER
60; Body: {{{-@something}}, @Adj_em##adj_lm, [{{@mods}}]* , @adj_n_em##adj_rm}
TRIGGER
70; Body: {{{-@something}}, @Adj_es##adj_l, [{{@mods}}]* , @adj_n_es##adj_r}
POS_EV
40; Body: {@det##$$0, [{{@mods}}]* , $adj_lm}; Constraints: [cin({$$0, C?[NPJP*2]},
cin({$adj_lm, C?[NPJP*2]}, cin({$adj_rm, C?[NPJP*2]}))
POS_EV
40; Constraints: [cin({$adj_l, C?[NPJP*2]}, cin({$adj_r, C?[NPJP*2]}))
NEG_EV
50; Body: {{{$adj_l}| {$adj_lm}}, [-(@vfin)]* , @vfin#verb, [-(@vfin)]* , {{{$adj_r}}
}
NEG_EV
30; Body: {{{$adj_l}| {$adj_lm}}, @noun##noun}
NEG_EV
30; Body: {@det##det, {{{$adj_r}| {$adj_rm}}}
```

Feature structures CChunks SChunks Configuration

- CCHUNKS
 - KOUS
 - PPER
 - NP
 - PP
 - APPR
 - aus
 - AP
 - ADJD
 - neu
 - ADJA
 - erzeugten
 - KON
 - oder
 - AP
 - PTKNEG
 - nicht
 - ADJA
 - gespeicherte
 - NN
 - Tabellen
- W
- \$
- W
- PPER
- PPER
- AVP
- W
- \$

- ❑ Full trace & chunk browser
 - Variable binding
 - Constraint resolution



Future work

- ❑ Integrate additional lexical resources into rule formalism, e.g.
 - Semantic ontologies
 - Terminological databases
- ❑ Provide path equations as constraints
 - permitting the representation of DAGs
 - specialisation of previous matches
- ❑ Integrate deep NLP (project Whiteboard)
- ❑ Optimise pattern matching engine



Conclusion

- ❑ Phenomenon-based approach to checking
 - Cheap identification of error candidates
 - Focussed processing for error confirmation
 - Resource-adaptivity

- ❑ Separation of error specification and underlying NLP components
 - customisable
 - extendible