DELPH-IN Summit 2009
Barcelona
22nd July 2009

Tutorial

Chart Mapping in PET

Peter Adolphs
DFKI GmbH
Language Technology Lab
Project Office Berlin

Part 1
Token Mapping

Motivation

- hybrid processing, integrating annotations of preprocessing tools into HPSG parsing
- we need to adapt annotations of different tools to the requirements of our grammar
- example: adapting output of PTB-style tokenizers to the ERG
  - input string: Don't you!
  - tokenizer output: <do, n't, you, !>
  - tokens as expected by the ERG: <don't, you!>

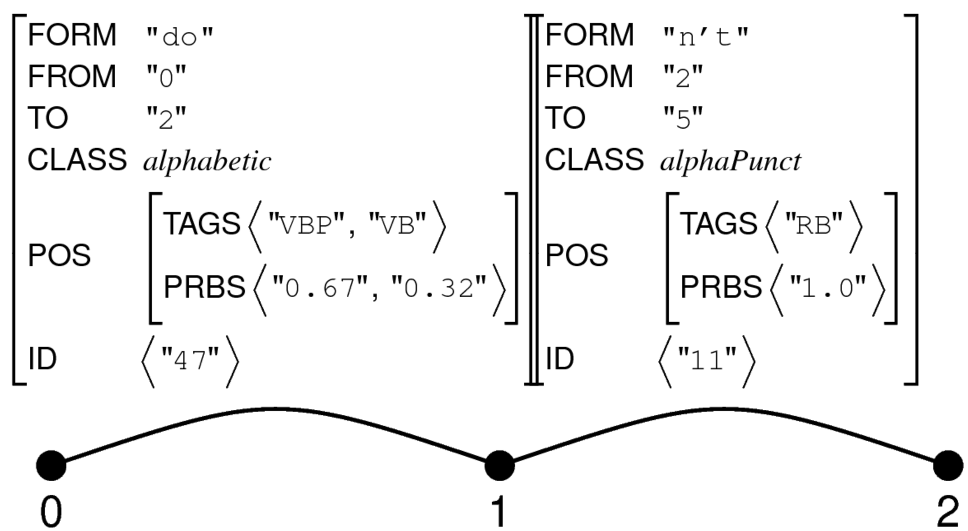
First Example Rule

$$\begin{bmatrix} \text{FORM} & \texttt{"do"} \\ \text{TO} & \boxed{1} \end{bmatrix}, \begin{bmatrix} \text{FORM} & \texttt{"n't"} \\ \text{FROM} & \boxed{1} \end{bmatrix} \rightarrow \begin{bmatrix} \text{FORM} & \texttt{"don't"} \end{bmatrix}$$

- example: recombining split contracted forms
- key concepts:
  - token feature structures
  - generalized chart
  - rewrite rules on chart items

Token Feature Structures

- feature structures for describing tokens
- annotations provided by different tools synthesized in token feature structures
- lattice of structured categories (token feature structures) as input to the parser



Generalized Chart

- tools may assume different tokenization (paradigm case: input from speech recognizers)
- chart: dag whose vertices are abstract objects rather than indexed token boundary positions

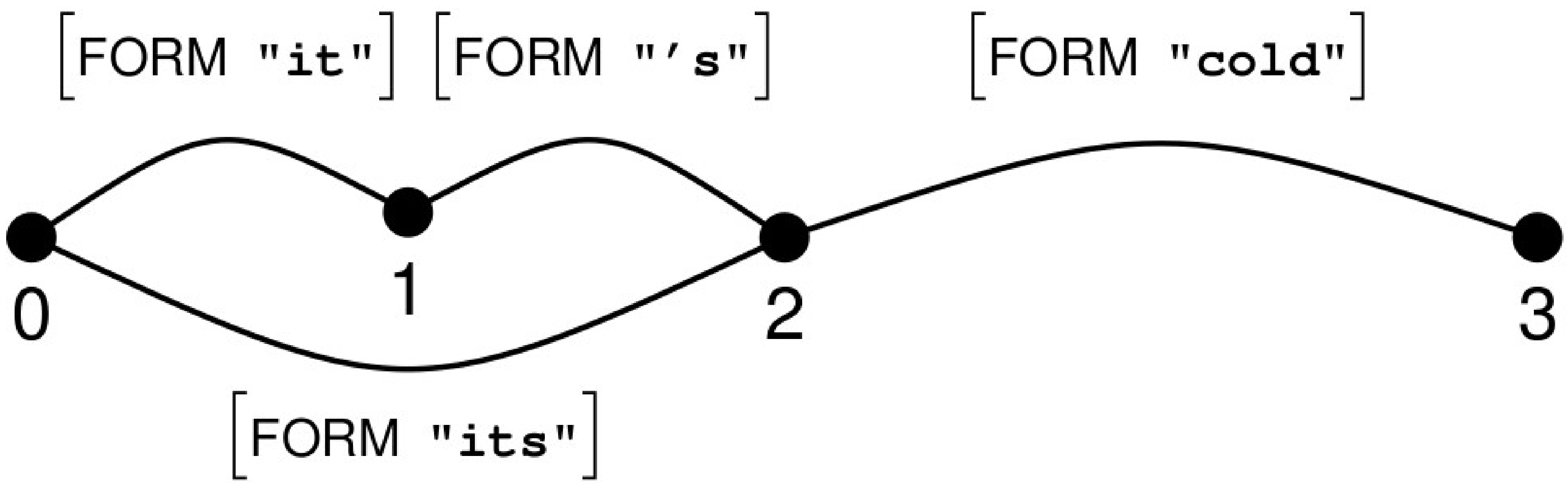


Chart Mapping Rules

- chart mapping: non-monotonic rewrite mechanism on feature structure chart edges
- general format:

$$[\ \text{CONTEXT} :\ ]\ \text{INPUT} \rightarrow \text{OUTPUT}$$

- CONTEXT, INPUT, OUTPUT are sequences of feature structures (each possibly empty)
- resource-sensitive: chart edges that let a rule fire may be removed (namely, all INPUT edges)

Chart Mapping Rules

- rules represented by feature structures
- reentrancies enforce value identity
- example*:

$$\text{chart\_mapping\_rule} := \text{*top*} \ \& \begin{bmatrix} \text{CONTEXT} & *list*, \\ \text{INPUT} & *list*, \\ \text{OUTPUT} & *list* \end{bmatrix}$$

$$\text{ptb\_dont\_tmr} := \text{chart\_mapping\_rule} \ \& \begin{bmatrix} \text{CONTEXT} & \langle\rangle, \\ \text{INPUT} & \left\langle \begin{bmatrix} \text{FORM} & \texttt{"do"} \\ \text{TO} & \boxed{1} \end{bmatrix}, \begin{bmatrix} \text{FORM} & \texttt{"n't"} \\ \text{FROM} & \boxed{1} \end{bmatrix} \right\rangle, \\ \text{OUTPUT} & \left\langle \begin{bmatrix} \text{FORM} & \texttt{"don't"} \end{bmatrix} \right\rangle \end{bmatrix}$$

* this example is incomplete and will be refined later

Copying Information

- OUTPUT items are instantiated by copying the argument in the particular rule match
- specify the values of all relevant features of the OUTPUT, otherwise information will leak
- reentrancies can be used to copy information from INPUT to OUTPUT

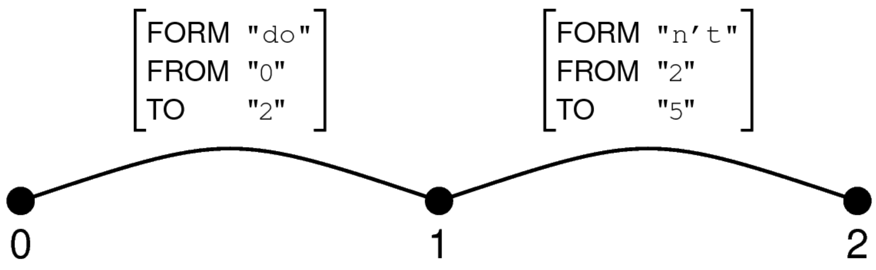
Copying Information

$$
\text{ptb\_dont\_tmr} := \text{chart\_mapping\_rule} \ \& \ \begin{bmatrix}
\text{CONTEXT} & \langle\rangle, \\
\text{INPUT} & \left\langle \begin{bmatrix} \text{FORM} & \texttt{"do"} \\ \text{FROM} & \boxed{1} \\ \text{TO} & \boxed{2} \end{bmatrix}, \begin{bmatrix} \text{FORM} & \texttt{"n't"} \\ \text{FROM} & \boxed{2} \\ \text{TO} & \boxed{3} \end{bmatrix} \right\rangle, \\
\text{OUTPUT} & \left\langle \begin{bmatrix} \text{FORM} & \texttt{"don't"} \\ \text{FROM} & \boxed{1} \\ \text{TO} & \boxed{3} \end{bmatrix} \right\rangle
\end{bmatrix}
$$

Chart Mapping Procedure

- rule matches associate rule arguments with chart items
- the initial rule match is a copy of the rule fs

chart:



initial rule match (incomplete):

$$\begin{bmatrix} \text{chart\_mapping\_rule} & \\ \text{CONTEXT} & \langle\rangle, \\ \text{INPUT} & \left\langle \begin{bmatrix}\text{FORM} & \texttt{"do"}\\ \text{FROM} & \boxed{1}\\ \text{TO} & \boxed{2}\end{bmatrix}, \begin{bmatrix}\text{FORM} & \texttt{"n't"}\\ \text{FROM} & \boxed{2}\\ \text{TO} & \boxed{3}\end{bmatrix} \right\rangle, \\ \text{OUTPUT} & \left\langle \begin{bmatrix}\text{FORM} & \texttt{"don't"}\\ \text{FROM} & \boxed{1}\\ \text{TO} & \boxed{3}\end{bmatrix} \right\rangle \end{bmatrix}$$

Chart Mapping Procedure

- each chart item is unified into the next unbound CONTEXT and INPUT argument of a rule match to yield the next rule match

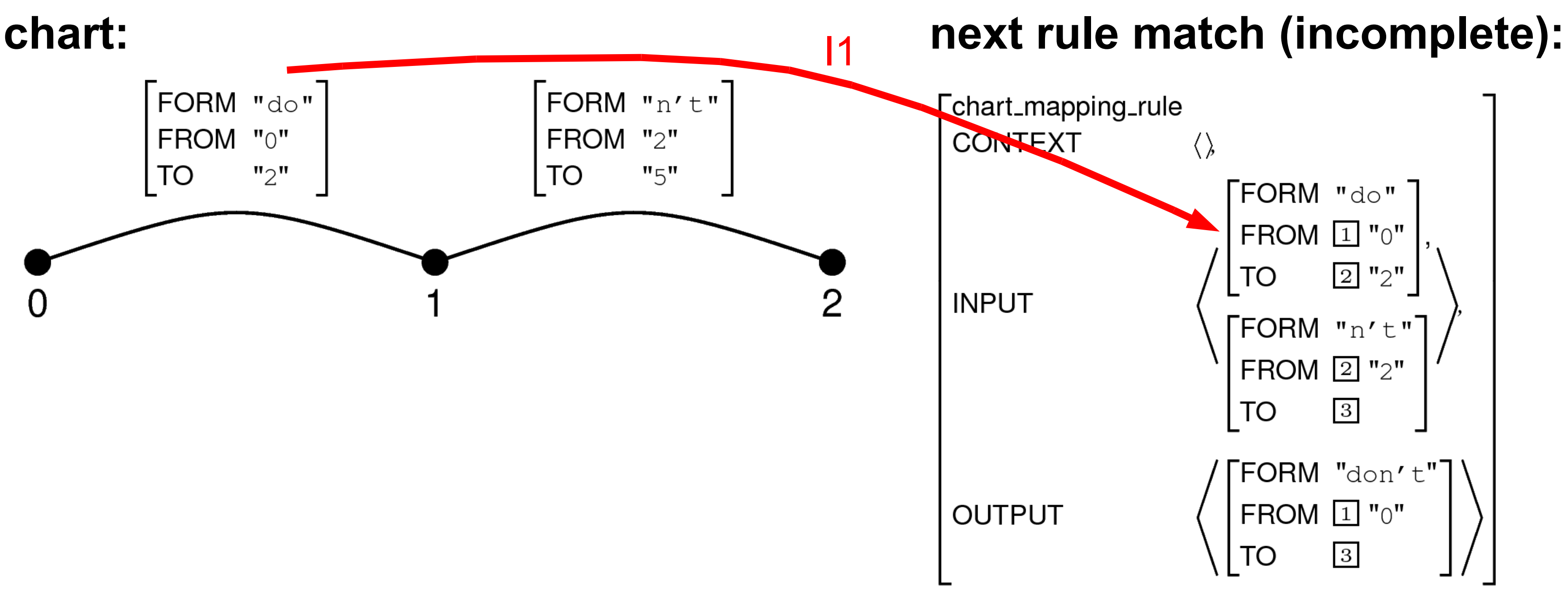


Chart Mapping Procedure

- a rule match is completed if all CONTEXT and INPUT arguments are bound

chart: **next rule match (complete):**

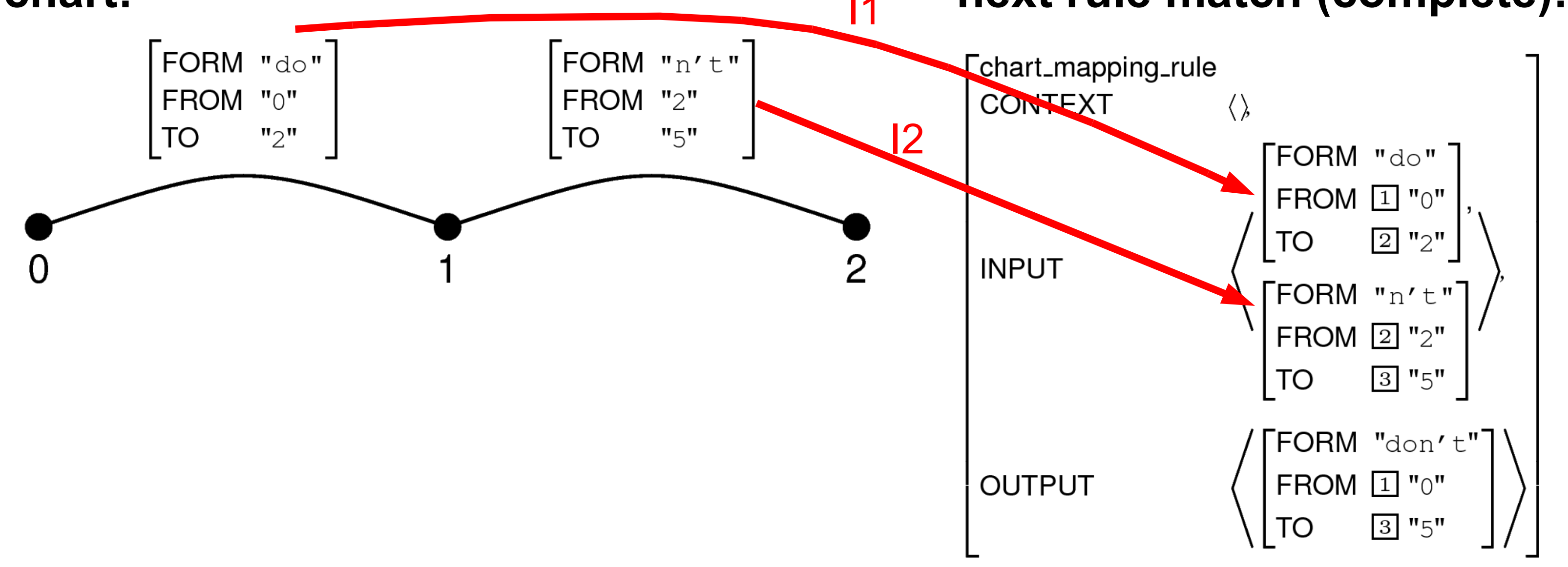


Chart Mapping Procedure

- a rule fires if the rule match is complete
- all INPUT items are removed
- all OUTPUT items are instantiated

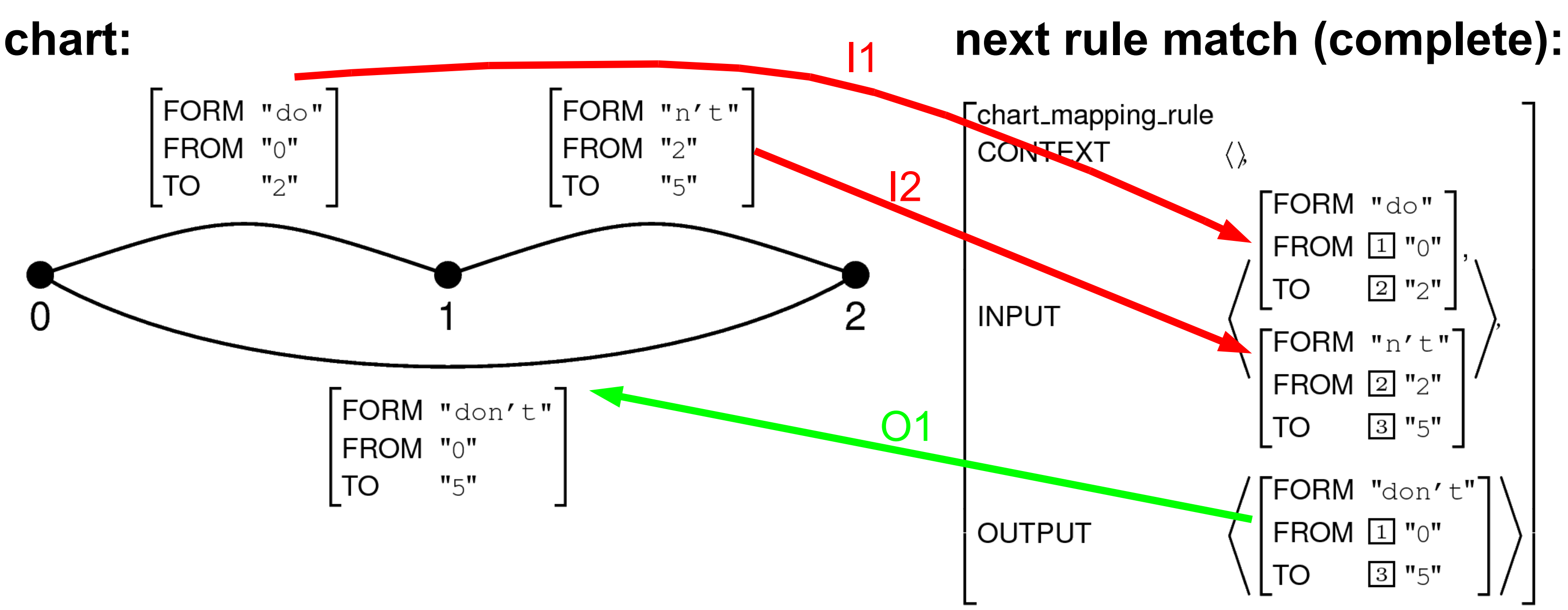
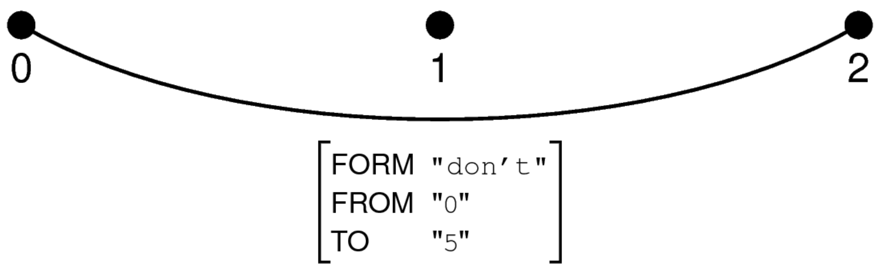


Chart Mapping Procedure

- each rule is applied until its fixpoint is reached
- cascaded architecture: all rules are applied in the order of their definition

chart after ptb_dont_tmr fired:



Regular Expressions

- unification + Perl-style regular expressions
- regex capture groups can be referred to in the output

regular expressions indicated by ^$

$$\text{ptb\_contracted\_tmr} := \begin{bmatrix} \text{chart\_mapping\_rule} & \\ \text{CONTEXT} & \langle\rangle \\ \text{INPUT} & \left\langle \begin{bmatrix}\text{FORM} & \texttt{\textasciicircum(.)\$} \\ \text{FROM} & \boxed{1} \\ \text{TO} & \boxed{2}\end{bmatrix}, \begin{bmatrix}\text{FORM} & \texttt{"n't"} \\ \text{FROM} & \boxed{2} \\ \text{TO} & \boxed{3}\end{bmatrix}\right\rangle \\ \text{OUTPUT} & \left\langle\begin{bmatrix}\text{FORM} & \text{"\$}\left\{\text{I1:FORM:1}\right\}\texttt{n't"} \\ \text{FROM} & \boxed{1} \\ \text{TO} & \boxed{3}\end{bmatrix}\right\rangle\end{bmatrix}$$

insert from input item 1 from path FORM the first capture group

Positional Constraints

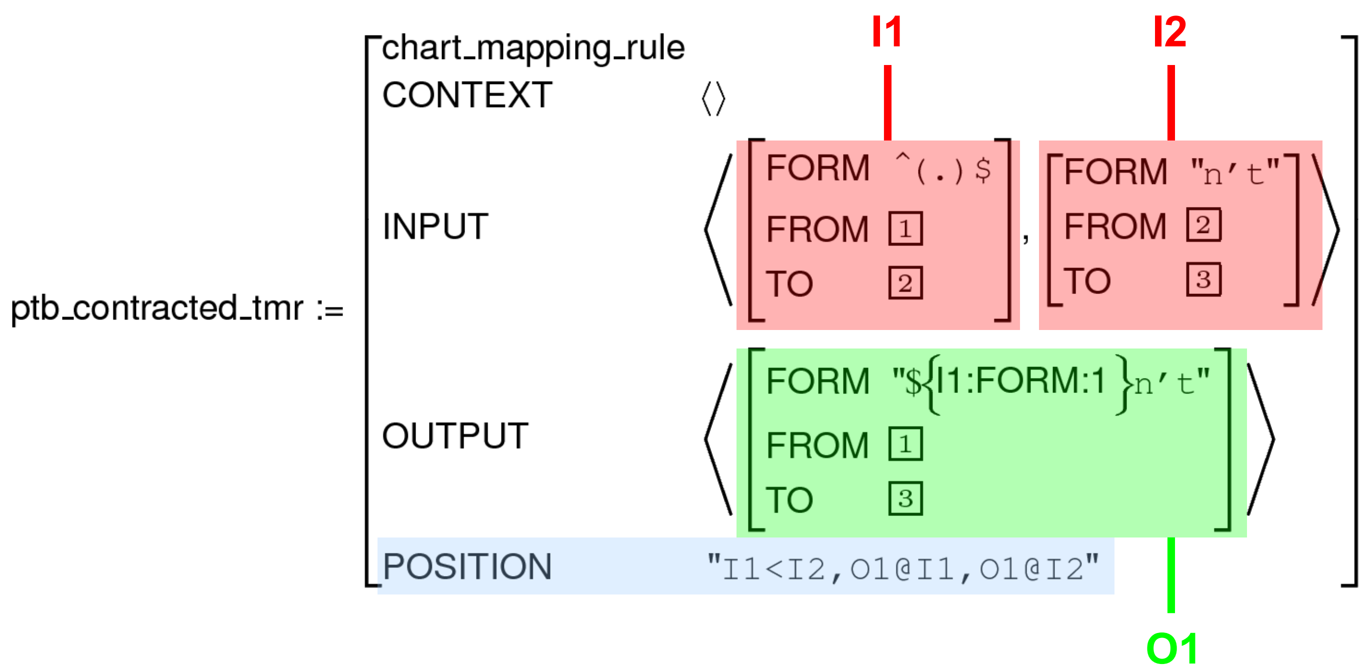
- so far, the positional relations between rule arguments have not really been addressed
- we need to state how CONTEXT and INPUT items positionally related to each other and where to anchor OUTPUT items
- FROM and TO values cannot be used for that purpose (FROM and TO of two adjacent items are usually not the equal)
- positional constraints between items are specified with a simple language

Positional Constraints

- items I1 and I2 are adjacent:
  `I1 < I2` or `I1 > I2`
- item I1 precedes I2 (possibly adjacent):
  `I1 << I2`
- item I1 succeeds I2 (possibly adjacent):
  `I2 >> I1`
- item I1 and I2 are in parallel:
  `I1@I2`
- chart start and chart end can be used too:
  `^` and `$` (e.g. `^ < I1`)

Positional Constraints

- several such constraints can be conjoined
- positional constraints currently as a comma-separated string (subject to change)



Application Examples

- light-weight named entity recognition:

$$\left[\text{FORM} \quad \texttt{\^{}[1-9][0-9]*\$}\right] \longrightarrow \begin{bmatrix} \text{FORM} & \text{"\$}\{\text{l1:FORM:1}\}\text{"} \\ \text{CLASS} & \textit{card\_ne} \end{bmatrix}$$

$$\left[\text{FORM} \quad \texttt{\^{}([0-2]?[0-9]:[0-5][0-9])\$}\right] \longrightarrow \begin{bmatrix} \text{FORM} & \text{"\$}\{\text{l1:FORM:1}\}\text{"} \\ \text{CLASS} & \textit{clockTime\_ne} \end{bmatrix}$$

$$\left[\text{FORM} \quad \texttt{\^{}<?[[:alnum:].\_]+@[[:alnum:]\_]+(\textbackslash.[[:alnum:]\_]+)+>?\$}\right]$$

$$\longrightarrow \begin{bmatrix} \text{FORM} & \text{"\$}\{\text{l1:FORM:1}\}\text{"} \\ \text{CLASS} & \textit{email\_ne} \end{bmatrix}$$

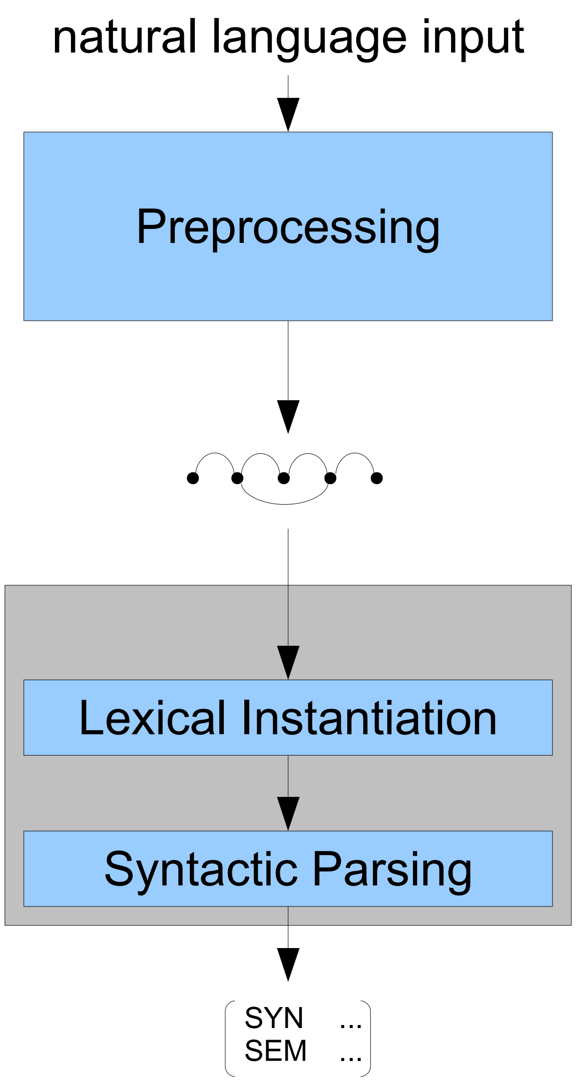
Application Examples

- fixing broken tokenization:

$$\left[\text{FORM} \quad \texttt{\^{}(.+:)([:alnum:].*)\$}\right]$$

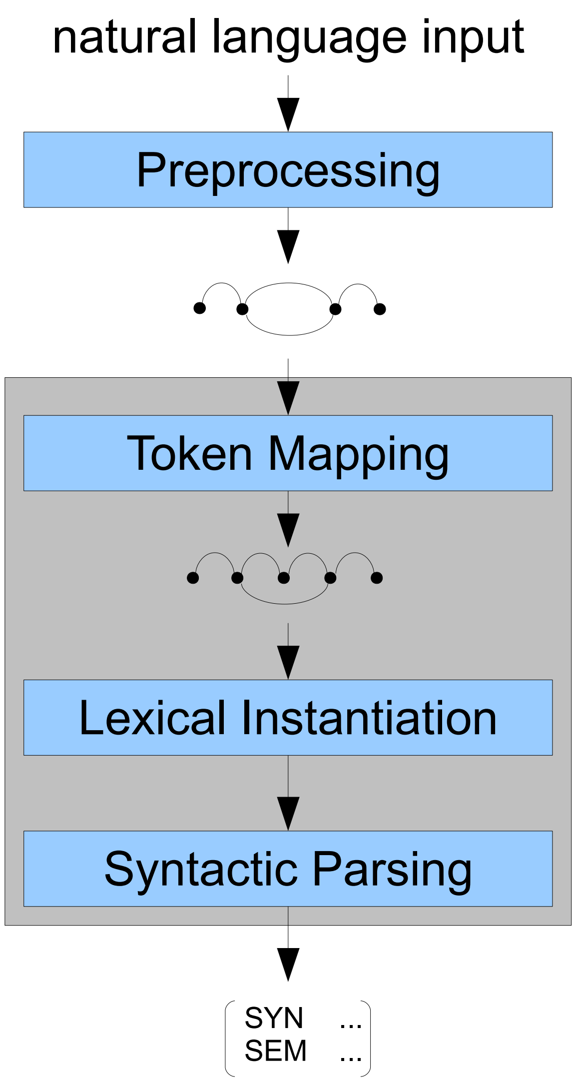
$$\rightarrow \left[\text{FORM} \quad \text{"\$}\left\{\text{l1:FORM:1}\right\}\text{"}\right], \left[\text{FORM} \quad \text{"\$}\left\{\text{l2:FORM:1}\right\}\text{"}\right]$$

Old Architecture



- preprocessing has to deliver an input chart as expected by the grammar
- this has to be ensured by specialized conversion routines without recourse to the grammar
- changes to the grammar have to be reflected in these data adaptation routines

New Architecture



- token mapping performs certain preprocessing steps within the grammar
- advantages:
  - full control for the grammar writer, using the same formalism as for the grammar
  - makes assumptions by the grammar explicit
  - removes complexity from preprocessing

Part 2
Lexical Instantiation
& Lexical Filtering

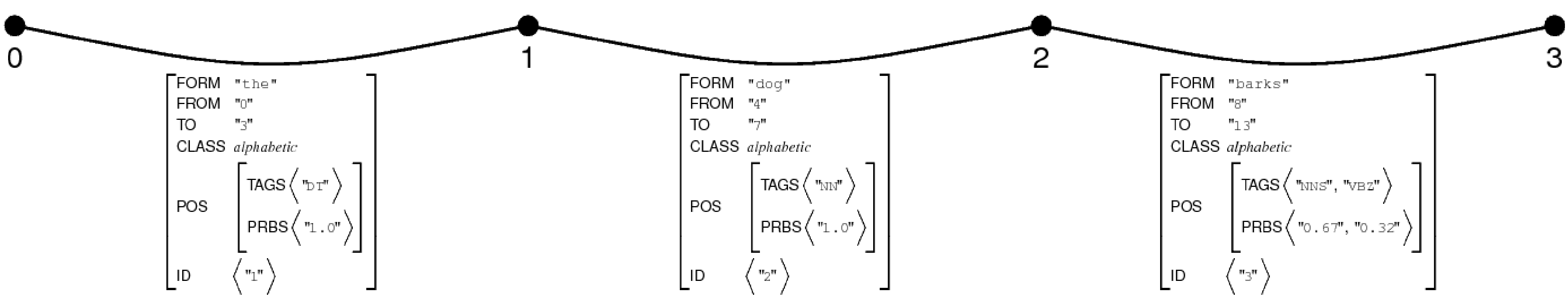
Hybrid Processing

- shaping the search space of the parser:
  - widen search space (e.g. unknown word handling)
  - narrow search space (e.g. prevent edges not conforming to the output of a chunker)
- widening the search space often requires constraining it later; constraints can be:
  - hard: categorial conditions for the removal of chart edges
  - soft: leave it ultimately up to probabilistic disambiguation

Passing Information Into LEs

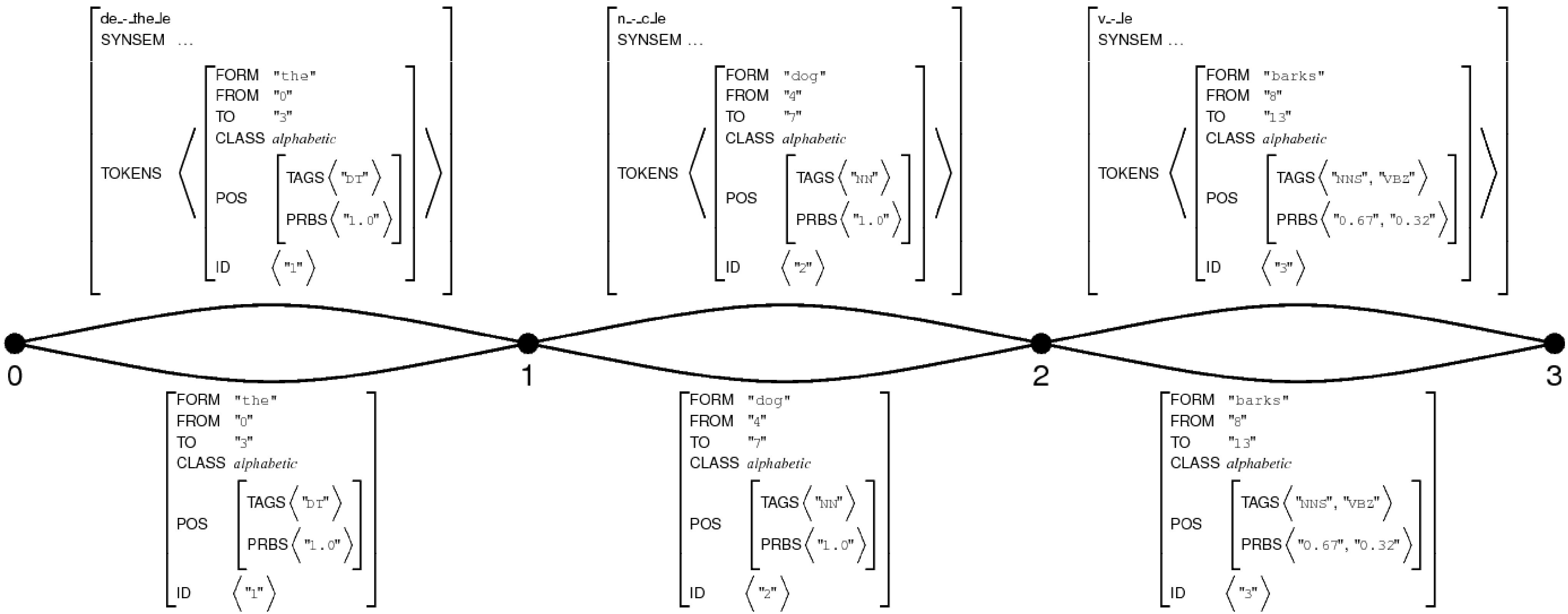
- token fs are unified into lexical items:

$$\text{word\_or\_lexeme} := \begin{bmatrix} \text{sign} & \\ \text{SYNSEM} & \textit{synsem}, \\ \text{TOKENS} & \textit{*list*} \end{bmatrix}$$



Passing Information Into LEs

- token fs are unified into lexical items:



- TOKENS can be used for filtering

Lexical Instantiation of Generics

- selection of appropriate generic les originally controlled by the parser (hard-coded):
  - map from part-of-speech tags to generic les
  - instantiate generic le for highest ranked pos tag where no native le is available
- disadvantage:
  - not flexible enough (e.g. use several taggers)
  - cannot deal with partial lexical coverage,
    e.g. *We'll* ***bus*** *to Paris.*

Lexical Instantiation of Generics

- try to instantiate **all** generic les for **all** tokens
- filtering incompatible tokens by constraints on TOKENS
- example:

$$\text{genericname} := \begin{bmatrix} \text{n\_-\_pn-unk\_le} & \\ \text{ORTH} & \langle\ \text{"\_}\texttt{generic\_nnp\_}\text{"}\ \rangle \\ \text{TOKENS} & \langle \begin{bmatrix} \text{POS.TAGS} & \langle\ \mathit{NNP}, \ldots \rangle \end{bmatrix} \rangle \end{bmatrix}$$

$$\text{generic\_card\_ne} := \begin{bmatrix} \text{aj\_-\_i-crd-gen\_le} & \\ \text{ORTH} & \langle\ \text{"\_}\texttt{generic\_card\_ne\_}\text{"}\ \rangle \\ \text{TOKENS} & \langle \begin{bmatrix} \text{CLASS} & \mathit{card\_ne} \end{bmatrix} \rangle \end{bmatrix}$$

Lexical Instantiation for Generics

- complementary solution to generic instantiation: create le types for unknown words on the fly by a lexical type predictor
  - let the lexical type predictor create generic les according to the statistical model
  - add further generic les based on categorial conditions where you’re absolutely sure (e.g. trusting the output of a specialized gazetteer)

Lexical Filtering

- after lexical instantiation, native and generic les may be available in the same chart cell
- we can restrict lexical instantiation by positing constraints on the token feature structures
- but we might also want to prevent some lexical chart edges in certain contexts (set operations)

Lexical Filtering

- lexical filtering phase, between lexical parsing and syntactic parsing
- same formalism as for token mapping: chart mapping rules but with empty OUTPUT list
- hard constraints on the parser's search space

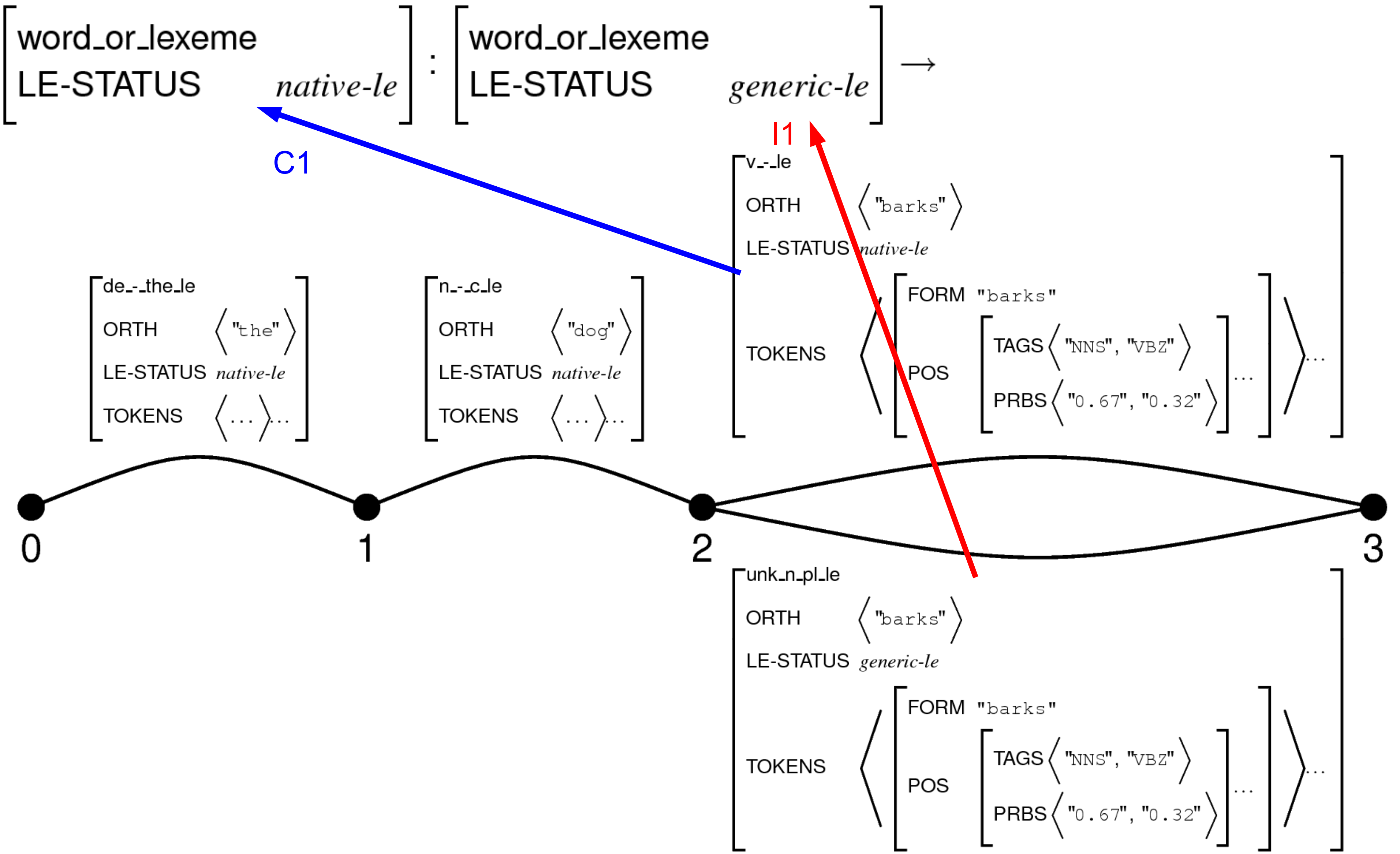
Lexical Filtering

- e.g.: filtering generic lexical entries where native are available (lexical items are extended with an LE-STATUS feature in this example):

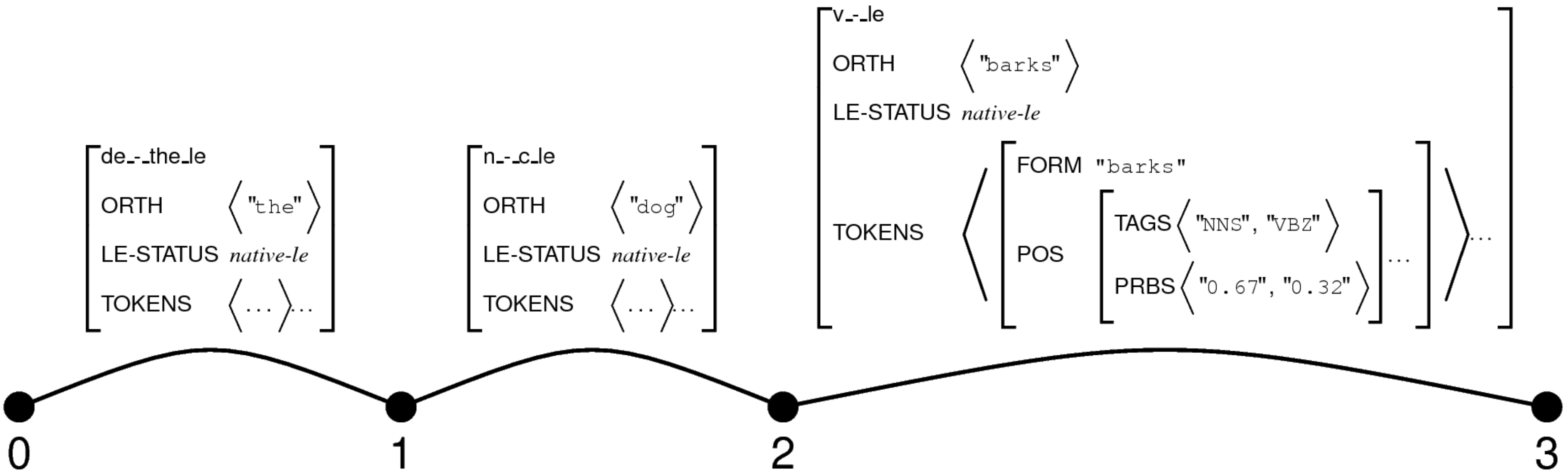
$$\begin{bmatrix}\text{word\_or\_lexeme} & \\ \text{LE-STATUS} & \textit{native-le}\end{bmatrix} : \begin{bmatrix}\text{word\_or\_lexeme} & \\ \text{LE-STATUS} & \textit{generic-le}\end{bmatrix} \rightarrow$$

- actual rules should be more finegrained (e.g. delete generic entries if native entries with same pos are available)

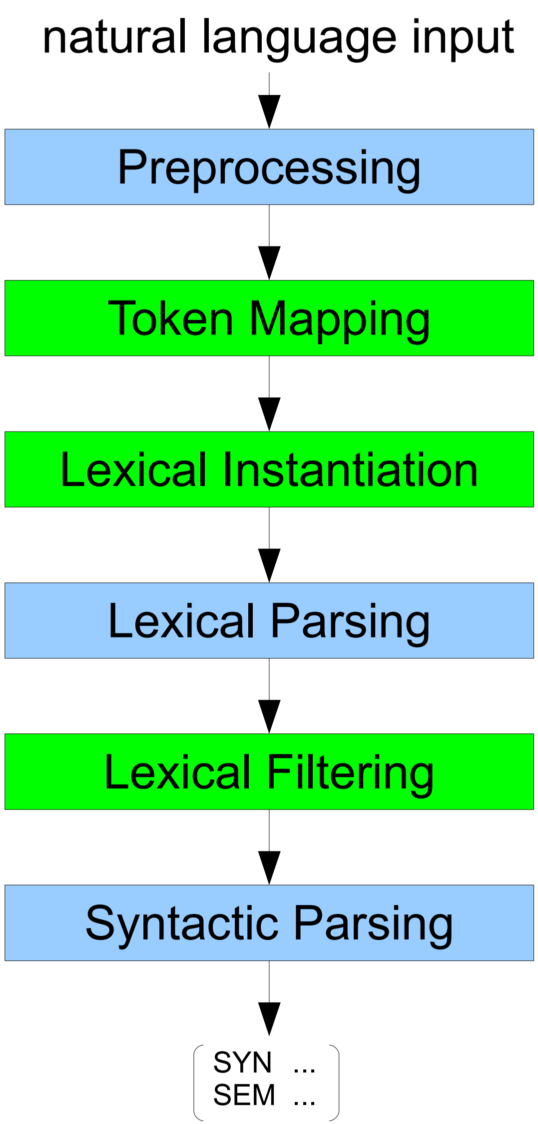
Lexical Filtering



Lexical Filtering



New Architecture



- use feature structures to describe tokens
- chart mapping: resource-sensitive rewriting of feature structure items
- chart mapping on token fs
- generic instantiation driven by compatibility with token fs
- lexical filtering with chart mapping

Part 3
Using Chart Mapping in PET

Changes to the Grammar

- ingredients for using chart mapping in your grammar:
  - types for token fs
  - add token fs to lexical items
  - types for chart mapping rules
  - actual chart mapping rules
  - settings telling PET what to find where
- convention: we used + as a prefix for chart-mapping feature names to prevent clashes with existing feature names

Changes to the Grammar: Types

- token type:

```
token := *top* & [ +FORM string,
                   +FROM string,
                   +TO   string,
                   +POS  pos,      % +TNT tnt in ERG
                   +ID   *diff-list* ].
```

- type for part-of-speech tagger results (aligned lists of tags and probabilities):

```
pos      := *top* & [ +TAGS *list*, +PRBS *list* ].
null_pos := pos   & [ +TAGS < >,    +PRBS < > ].
```

Changes to the Grammar: Types

- token lists:

```
tokens := *top* &
          [ +LIST *list*,
            +LAST token ].
```

- add token feature structures to lexical items:

```
word_or_lexrule := sign &
  [ SYNSEM synsem,
    ORTH [ FROM #from, TO #to ],
    TOKENS tokens &
           [ +LIST & < [ +FROM #from ], ... >,
             +LAST.+TO #to ] ].
```

Changes to the Grammar: Types

- chart mapping rule types:

```
chart_mapping_rule := *top* &
[ +CONTEXT  *list*,
  +INPUT    *list*,
  +OUTPUT   *list*,
  +POSITION string ].
token_mapping_rule := chart_mapping_rule.
lexical_filtering_rule := chart_mapping_rule.
```

- useful: using types for typical chart mapping rule configurations:

```
one_one_tmt := token_mapping_rule &
[ +INPUT  < [ +ID #id, +FROM #from, +TO #to ] >,
  +OUTPUT < [ +ID #id, +FROM #from, +TO #to ] >,
  +POSITION "O1@I1" ]
```

Changes to the Grammar: Rules

- token mapping rules:

```
ptb_slash_tmr := one_one_form_tmt &
[ +INPUT < [ +FORM ^(.*)\\/(.*)$ ] >,
  +OUTPUT < [ +FORM "${I1:+FORM:1}/${I1:+FORM:2}" ] > ].

...
```

- lexical filtering rules:

```
generic+native_lfr :=
  lexical_filtering_rule &
  [ +CONTEXT < [ SYNSEM.PHON.ONSET con_or_voc ] >,
    +INPUT   < [ SYNSEM.PHON.ONSET unk_onset ] >,
    +OUTPUT  < >,
    +POSITION "I1@C1" ].

...
```

Changes to the Grammar

- load token mapping and lexical filtering rules:

```
:begin :type.
:include "cmt.tdl".
:end :type.
:begin :instance :status token-mapping-rule.
:include "tmr.tdl".
:end :instance.
:begin :instance :status lexical-filtering-rule.
:include "lfr.tdl".
:end :instance.
```

- generics (as before):

```
:begin :instance :status generic-lex-entry.
:include "gle.tdl".
:end :instance.
```

Changes to the Grammar: Settings

- paths in cm rules:

```
chart-mapping-context-path  := "+CONTEXT".
chart-mapping-input-path    := "+INPUT".
chart-mapping-output-path   := "+OUTPUT".
chart-mapping-position-path := "+POSITION".
```

- path to token feature structures in lexical items:

```
lexicon-tokens-path := "TOKENS.+LIST".
lexicon-last-token-path := "TOKENS.+LAST"
```

- paths in token fs:

```
token-form-path     := "+FORM".
token-id-path       := "+ID".
token-from-path     := "+FROM".
token-to-path       := "+TO".
token-postags-path  := "+POS.+TAGS".
token-posprobs-path := "+POS.+PRBS".
```

Changes to the Grammar: Settings

- names for the cm sections:

```
token-mapping-rule-status-values :=
      token-mapping-rule.
lexical-filtering-rule-status-values :=
      lexical-filtering-rule.
```

- name for the generic le section (as before):

```
generic-lexentry-status-values :=
      generic-lex-entry.generic-lexentry-status-
values := generic-lex-entry.
```

Input Formats

- existing input formats (String, YY, PIC) can be used with chart mapping
- available information from old input formats is automatically mapped to token fs
- new input format: FSC (Feature Structure Chart)
  - XML-based input format
  - allows you to specify *arbitrary* token feature structures (integrate annotations from any tool)
  - currently only supported by acrocheck

Distribution

- distribution via LOGON Repository (prebuilt)

```
svn co http://svn.emmtee.net/tags/barcelona
$LOGONROOT/bin/flop -t
$LOGONROOT/bin/cheap -t
```

- distribution via PET Repository (sources)

```
svn co http://pet.opendfki.de/repos/pet/branches/cm
autoreconf -i
./configure -with-xml                      # cf README
make
sudo make install
```

- cm branch will be merged to main soon

```
svn co http://pet.opendfki.de/repos/pet/main
```

Invocation

- invocation of chart mapping and new generic instantiation:
  ```
  cheap -cm -default-les=all
  ```
- add -t in logon handon release:
  ```
  $LOGONROOT/bin/cheap -t ...
  ```
- batch parsing in LOGON (cf. CPU definition for ERG with TNT tagger):
  ```
  ./parse --binary --erg+tnt <skeleton>
  ```

Debugging

- not very comfortable at the moment (PET lacks an interactive debugger)
- debugging via bit-flag parameters to -cm option
- subject to be changed to a logging framework

Debugging

- bit-flag parameters to -cm option:
  - bit 0: which rules fired & which max ids of items before and after each chart mapping phase
  - bit 1: which regexs matched
  - bit 2: initial and final token mapping chart
  - bit 3: initial and final lexical filtering chart
  - bit 4: which rules fired + print OUTPUT items
  - bit 8: which items were checked / which matched
- thus -cm=0: chartmapping without logging

Part 4
Wrap Up

Conclusions

- versatile device for many preprocessing tasks
- pre-processing can be better controlled with grammar-specific means
- external information is made accessible to the grammar
- reduces the need for special code inside and outside the parser

Chart Mapping Paper

Adolphs, Peter; Oepen, Stephan; Callmeier, Ulrich; Crysmann, Berthold; Flickinger, Dan & Kiefer, Bernd. 2008. “Some Fine Points of Hybrid Natural Language Parsing”. In *Proceedings of the 6th International Conference of Language Resources and Evaluation (LREC 2008)*. Marrakech, Morocco.

http://www.lrec-conf.org/proceedings/lrec2008/slides/349.pdf

Acknowledgements

- DELPH-IN community and beyond, especially Nuria Bertomeu, Ann Copestake, Remy Sanouillet, Bernd Kiefer, Ulrich Schäfer and Benjamin Waldron for numerous in-depth discussions
- funding:
  - ProFIT program of the German federal state of Berlin and the EFRE program of the EU (to the DFKI projects Checkpoint and KomParse)
  - the University of Oslo (through its scientific partnership with CSLI)

The Chart Mapping Tool in PET

It's New!

It's Flexible!

It's Powerful!

It's Fast!

It's Useful!

The Chart Mapping Tool in PET

It’s Ready For Use!