

Corpus transformations

Practical Course in Natural Language Processing

Boris Petrov

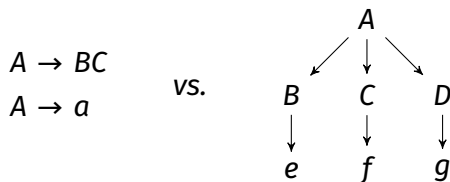
Chair for Algorithmics
Institute for Theoretical Computer Science
TU Dresden

16.06.2026

Practical problems in parsing

Our basic CFG parser has some issues:

- Only binary and unary rules can be processed, but the training corpus is n -ary



- Training corpus does not contain all English words
He is a brilliant deipnosophist . \rightsquigarrow *No parse!*
- Sparse data problem: rare words in the training corpus may get atypical probabilities

Outline

- 1 Binarization and Markovization
- 2 Unknown/rare word handling

Straight-forward binarization

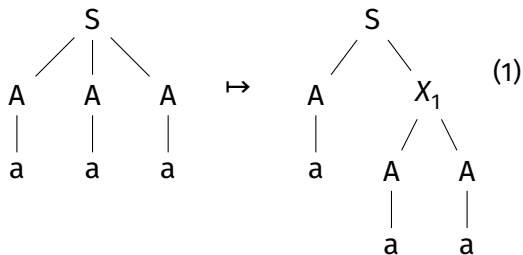
Binarizing the training corpus vs. binarizing the grammar

Straight-forward binarization

Binarizing the training corpus vs. binarizing the grammar

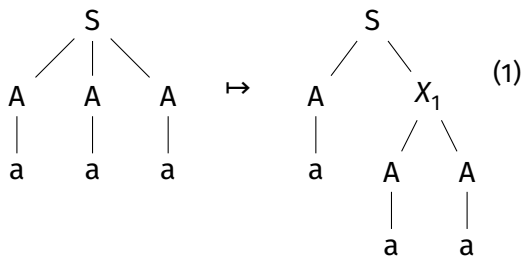
Straight-forward binarization

Binarizing the training corpus vs. binarizing the grammar



Straight-forward binarization

Binarizing the training corpus vs. binarizing the grammar

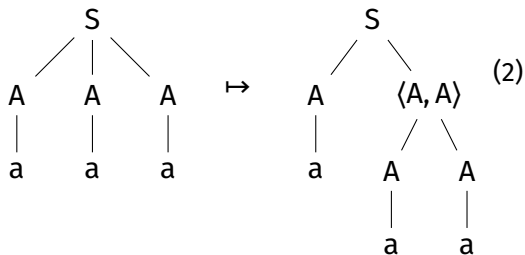


Problems:

- 1 Each occurrence of two consecutive nonterminals yields a new nonterminal X_i

Straight-forward binarization

Binarizing the training corpus vs. binarizing the grammar

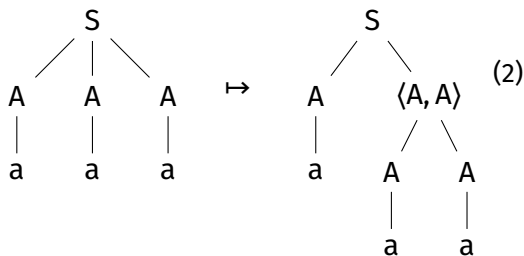


Problems:

- 1 Each occurrence of two consecutive nonterminals yields a new nonterminal X_i

Straight-forward binarization

Binarizing the training corpus vs. binarizing the grammar

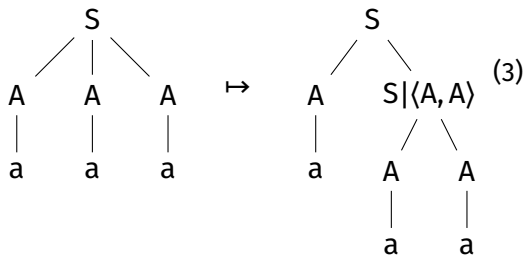


Problems:

- 1 Each occurrence of two consecutive nonterminals yields a new nonterminal X_i
- 2 No information about context is preserved

Straight-forward binarization

Binarizing the training corpus vs. binarizing the grammar

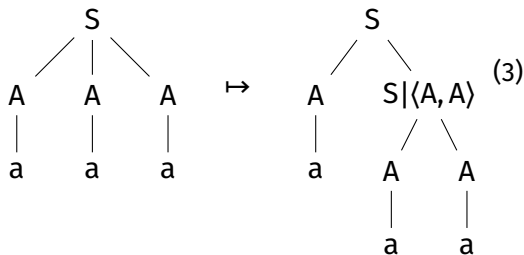


Problems:

- 1 Each occurrence of two consecutive nonterminals yields a new nonterminal X_i
- 2 No information about context is preserved

Straight-forward binarization

Binarizing the training corpus vs. binarizing the grammar



Problems:

- 1 Each occurrence of two consecutive nonterminals yields a new nonterminal X_i
- 2 No information about context is preserved
- 3 Number of right-hand side nonterminals is fixed

Previous slide Markovization with $v = 1$ and „ $h = \infty$ “

In general store $v \in \mathbb{N}_+$ nonterminals towards the root and
 $h \in \mathbb{N}_+$ nonterminals to the left

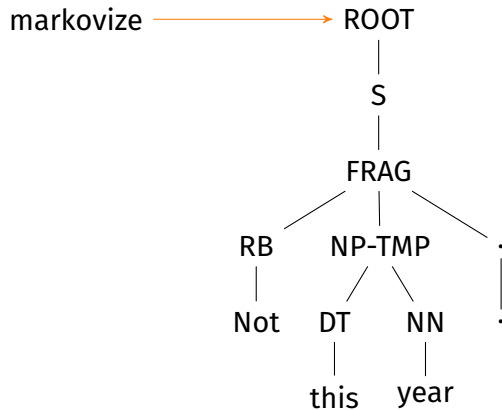
```
1: function MARKOVIZE( $t = \sigma(t_1, \dots, t_k)$ )
2:   if  $t$  is preterminal then
3:     return  $t$ 
4:   else if  $k \leq 2$  then
5:     return ADD_PARENTS( $\sigma$ )(MARKOVIZE( $t_1$ ), ..., MARKOVIZE( $t_k$ ))
6:   else
7:      $\sigma' \leftarrow$  ORIGINALLABEL( $\sigma$ ) | {label of  $t_2$ , ..., label of  $t_{h+1}$ }
8:     return ADD_PARENTS( $\sigma$ )(MARKOVIZE( $t_1$ ), MARKOVIZE( $\sigma'(t_2, \dots, t_k)$ ))
```

Note:

- $\text{ADD_PARENTS}(\sigma) = \sigma^\wedge \langle l_1, \dots, l_{v-1} \rangle$, where the l_i are the labels of the ancestors of σ which occur in the original tree
- If $v = 1$ or there are no parents, leave out $^\wedge \langle \rangle$

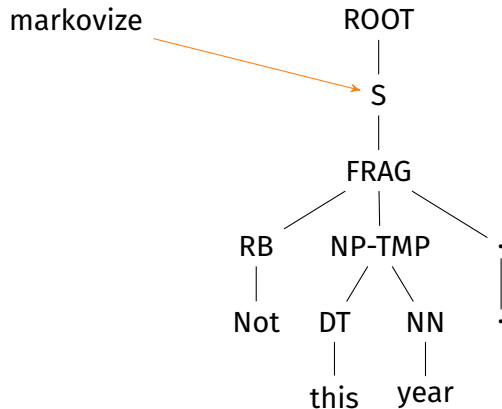
Markovization – example

here: $v = 3, h = \infty$



Markovization – example

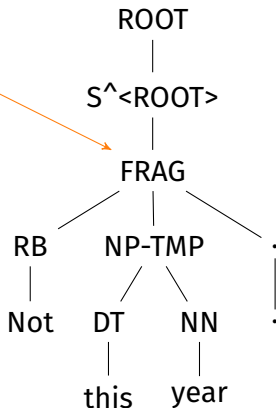
here: $v = 3, h = \infty$



Markovization – example

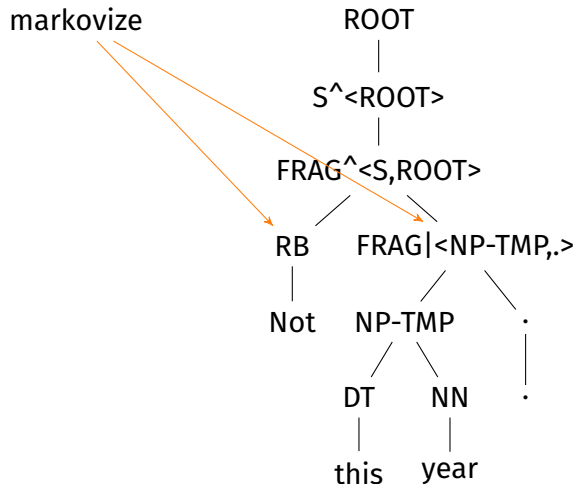
here: $v = 3, h = \infty$

markovize



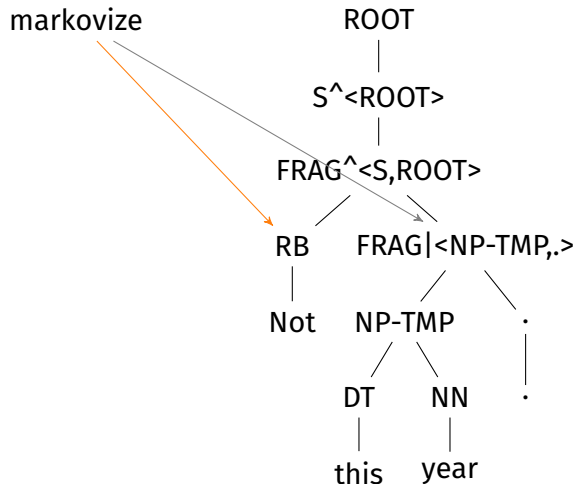
Markovization – example

here: $v = 3$, $h = \infty$



Markovization – example

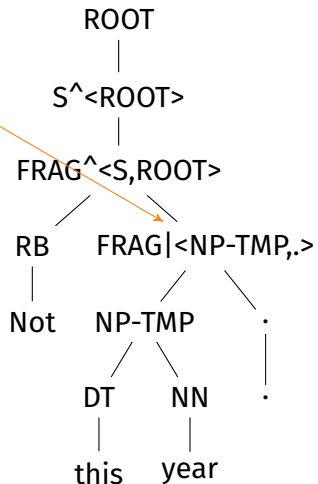
here: $v = 3$, $h = \infty$



Markovization – example

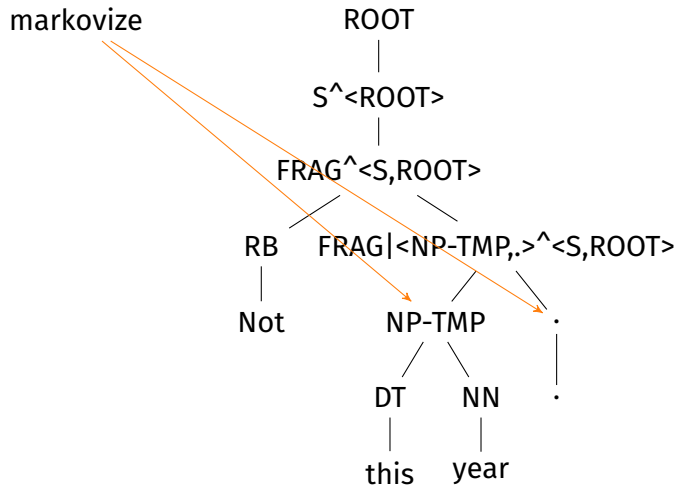
here: $v = 3$, $h = \infty$

markovize



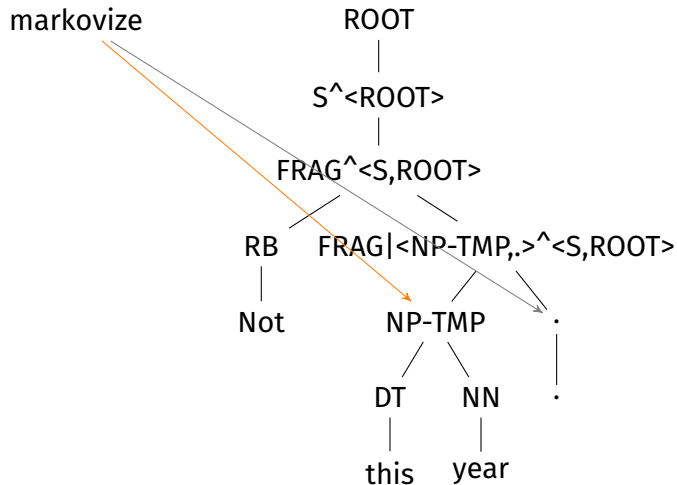
Markovization – example

here: $v = 3, h = \infty$



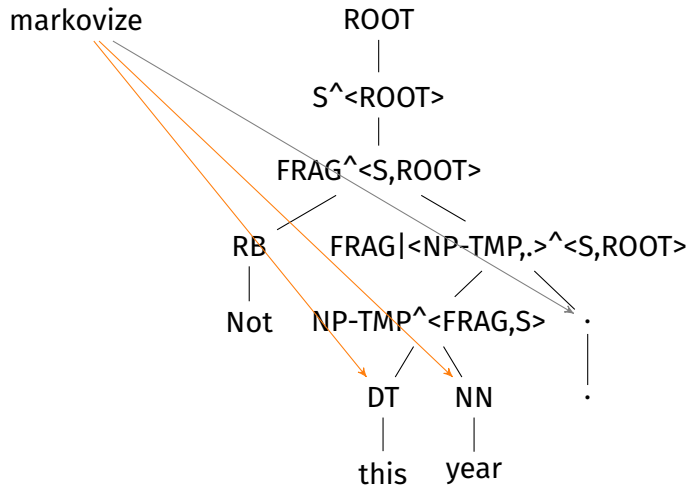
Markovization – example

here: $v = 3, h = \infty$



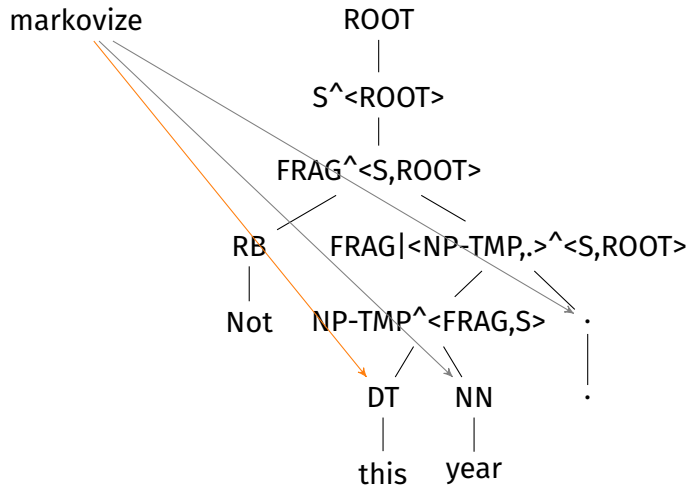
Markovization – example

here: $v = 3, h = \infty$



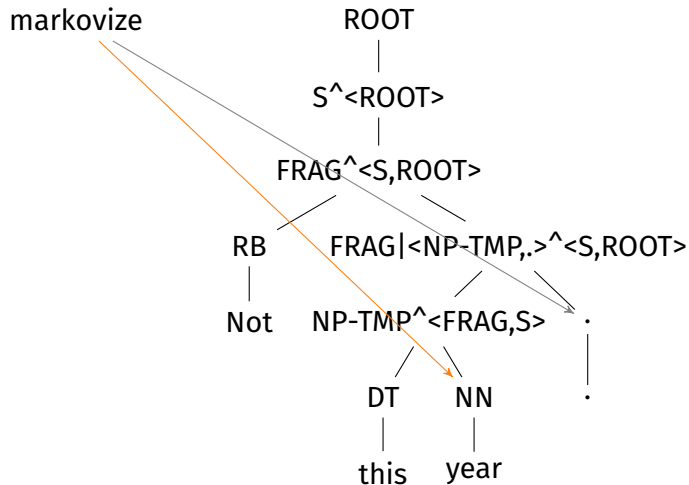
Markovization – example

here: $v = 3, h = \infty$



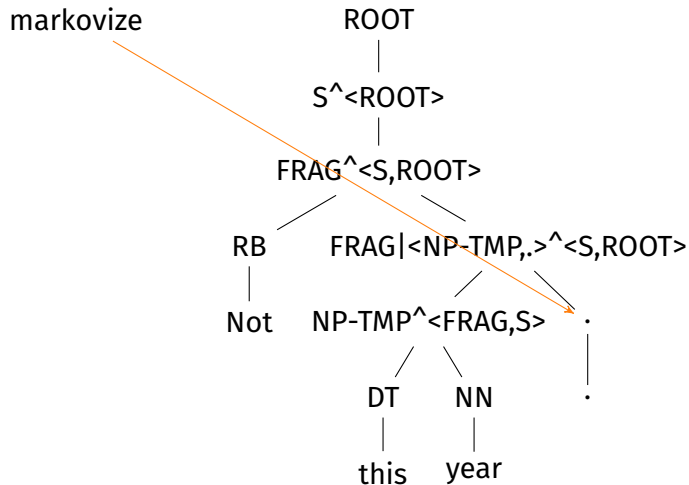
Markovization – example

here: $v = 3, h = \infty$



Markovization – example

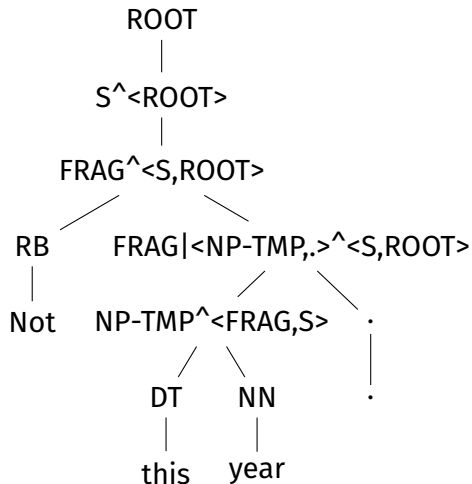
here: $v = 3, h = \infty$



Markovization – example

here: $v = 3, h = \infty$

markovize



After parsing: debinarization

Why?

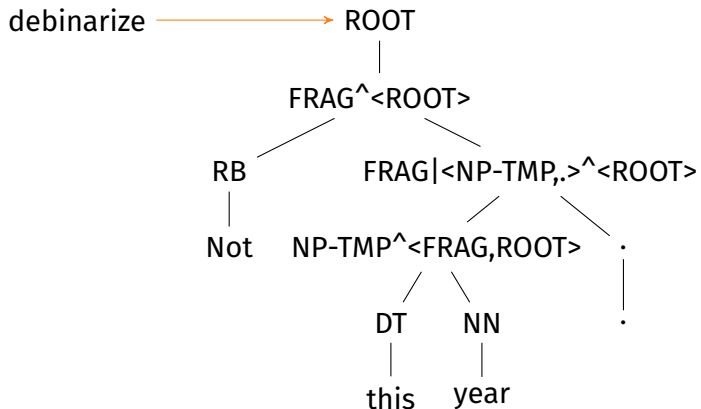
- Debinarized parse trees are linguistically relevant
- Comparison with (debinarized) gold corpus

```
1: function DEBINARIZE( $t = \sigma(t_1, \dots, t_k)$ )
2:   if ROOT( $t_k$ ) is Markovization node with children  $t'_1, t'_2$  then
3:     return DEBINARIZE( $\sigma(t_1, \dots, t_{k-1}, t'_1, t'_2)$ )
4:   else
5:     return  $\sigma(\text{DEBINARIZE}(t_1), \dots, \text{DEBINARIZE}(t_k))$ 
```

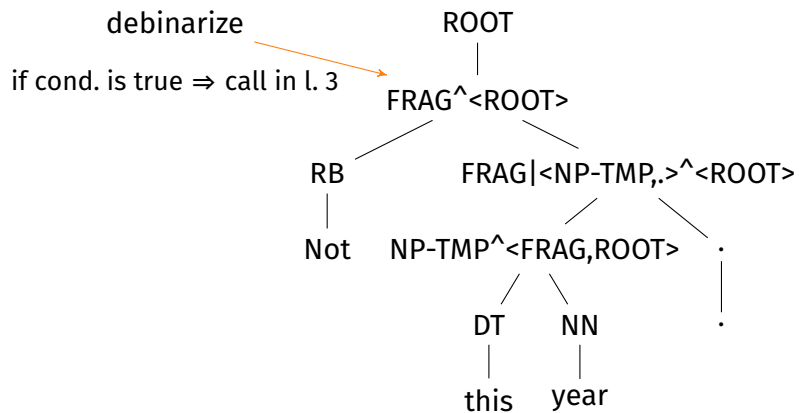
Also: remove ancestor annotation from each node (not shown)!

In material: both binarized and (debinarized) gold corpus

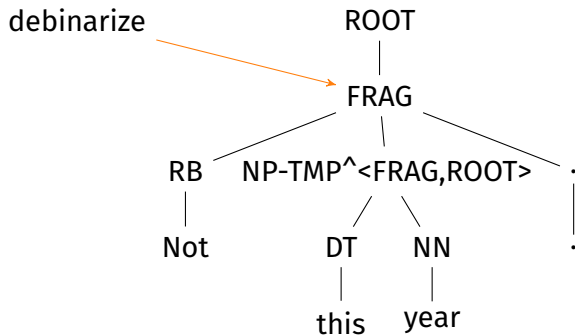
Debinarization – example



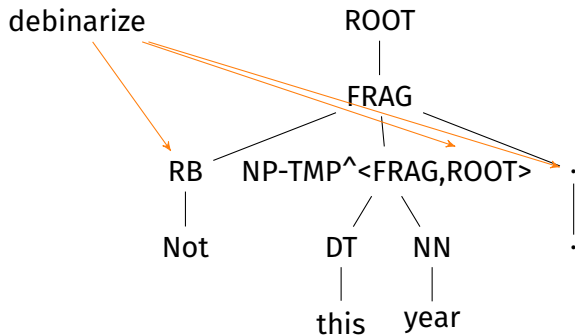
Debinarization – example



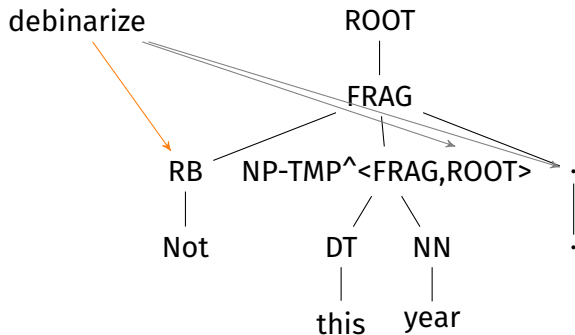
Debinarization – example



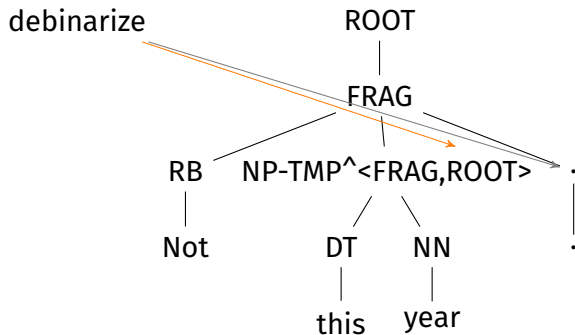
Debinarization – example



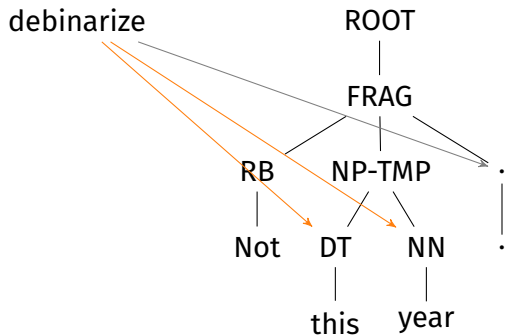
Debinarization – example



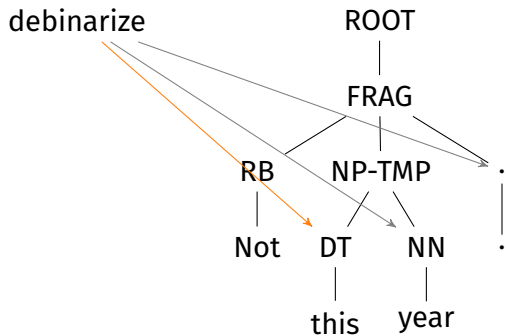
Debinarization – example



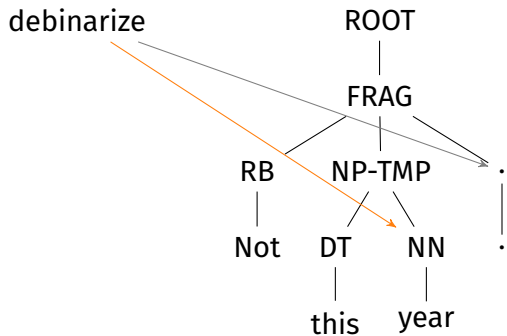
Debinarization – example



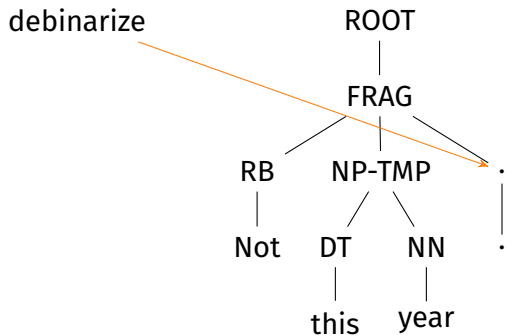
Debinarization – example



Debinarization – example

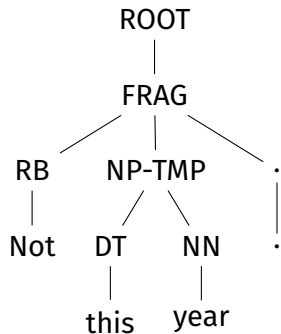


Debinarization – example



Debinarization – example

debinarize



Outline

- 1 Binarization and Markovization
- 2 **Unknown/rare word handling**

Basic unking

Idea:

- 1 Replace words that are *unknown* to the parser by an „unknown token“ (here: UNK)
- 2 Assign some probability mass to rules that generate UNK

Basic unking

Idea:

- 1 Replace words that are *unknown* to the parser by an „unknown token“ (here: UNK)
- 2 Assign some probability mass to rules that generate UNK

Implementation:

- 1 Reflection in the parser: before parsing w (where Σ is the set of terminals)

```
1:  $wmap \leftarrow (w_i \mid i \in \{1, \dots, |w|\})$   
2: for  $i = 1, \dots, |w|$  do  
3:   if  $w_i \notin \Sigma$  then  
4:      $w_i \leftarrow \text{UNK}$ 
```

and after parsing w as t , restore original words

```
1: for  $i = 1, \dots, |w|$  do  
2:    $\text{LEAVES}(t)[i] \leftarrow wmap[i]$ 
```

Basic unking (2)

- 2 Reflection in the grammar: replace rare words in the training corpus by UNK

Require: tree corpus *corpus* with terminal alphabet Σ , *threshold* $\in \mathbb{N}_+$

Ensure: every word occurring \leq *threshold* times in *corpus* is replaced by UNK

1: *wordcount* $\leftarrow (0 \mid i \in \Sigma)$

2: **for** $t \in \text{corpus}$ **do**

3: **for** $i = 1, \dots, |\text{LEAVES}(t)|$ **do**

4: *wordcount*[$\text{LEAVES}(t)[i]$] \leftarrow *wordcount*[$\text{LEAVES}(t)[i]$] + 1

5: **for** $t \in \text{corpus}$ **do**

6: **for** $i = 1, \dots, |\text{LEAVES}(t)|$ **do**

7: **if** *wordcount*[$\text{LEAVES}(t)[i]$] \leq *threshold* **then**

8: $\text{LEAVES}(t)[i] \leftarrow \text{UNK}$

The grammar is induced from the modified corpus!

Refinement

Observation: each unknown word is assigned the same probability

- bad language model: certain words are more likely to occur
- may worsen the parsing of sentences with rare words

Solution: categorize unknown words based on their *signature* [1]

$$\begin{array}{lcl} w_i \leftarrow \text{UNK} & \rightsquigarrow & w_i \leftarrow \text{GETSIGNATURE}(w_i, i) \\ \text{LEAVES}(t)[i] \leftarrow \text{UNK} & \rightsquigarrow & \text{LEAVES}(t)[i] \leftarrow \text{GETSIGNATURE}(w_i, i) \end{array}$$

- Some signatures can be found in the source code of the Berkely parser
- Next slide: „unknownLevel = 4“
- Heuristics, prone to overfitting
- Be creative, but don't overdo it!

```

1: function GETSIGNATURE(word, i)
2:   if |word| = 0 then return UNK
3:   letterSuffix ← ISUPPER(word1) ∧ NONE(ISLOWER, word) ⇒ -AC
                        ISUPPER(word1) ∧ i = 1 ⇒ -SC
                        ISUPPER(word1) ⇒ -C
                        ANY(ISLOWER, word) ⇒ -L
                        ANY(ISLETTER, word) ⇒ -U
                        otherwise ⇒ -S
4:   numberSuffix ← ALL(ISDIGIT, word) ⇒ -N
                        ANY(ISDIGIT, word) ⇒ -n
                        otherwise ⇒ ε
5:   dashSuffix ← ANY((= '-'), word) ⇒ -H
                        otherwise ⇒ ε
6:   periodSuffix ← ANY((= '.'), word) ⇒ -P
                        otherwise ⇒ ε
7:   commaSuffix ← ANY((= ','), word) ⇒ -C
                        otherwise ⇒ ε
8:   wordSuffix ← |word| > 3 ∧ ISLETTER(word|word|) ⇒ -TOLOWER(word|word|)
                        otherwise ⇒ ε
9:   return UNK · letterSuffix · numberSuffix · dashSuffix · periodSuffix · commaSuffix · wordSuffix

```

What to do?

Until 17.07.2025, 23:59,

- implement debinarization (3a)
 - implement trivial unking for corpus and adapt parser (3b)
 - implement 3 of
 - Markovization
 - smoothing
 - pruning
 - heuristic search
- } optimizations, next tutorial

All tasks' solutions are one submission!
You may send in your solutions earlier for feedback.

References I

- [1] Dan Klein und Christopher D Manning. „Accurate unlexicalized parsing“. Association for Computational Linguistics. 2003.