Morphological Analysis
Context-Free Grammars

Daniel Zeman

November 27, 2020

EUROPEAN UNION
European Structural and Investment Fund
Operational Programme Research,
Development and Education

Charles University
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

Warning

- We are going to observe a number of reasons why pure CFGs are not very suitable for MA
- Nevertheless we are going to study them because:
  - Extensions such as unification grammars (see later) are much more suitable
  - CFGs are also used to describe sentence structure (syntax)
  - The chart parsing algorithm is interesting enough and we would be looking at it anyway, sooner or later

Context-Free Grammars

- Quadruple (T, N, S, P)
  - T ... alphabet of terminal symbols, usually lowercase letters
  - N ... alphabet of non-terminal symbols, usually uppercase letters
  - $S \in N$ ... starting non-terminal symbol
  - P ... set of rewrite rules of the form $X \rightarrow \xi$ where $X \in N$ and $\xi \in (T \cup N)^*$
- A string can be derived in a CFG if it can be created by repeated application of the rules on the start symbol

Morphological Example

- The first step towards a context-free description of the structure of Czech words could roughly look like this
- Non-terminals start with uppercase, terminals with lowercase
  - Word → Comparison Negation Stem Suffix | Stem Suffix
  - Comparison → *nej*
  - Negation → *ne*
  - Stem → *abatyš* | *abbé* | *abdikac* | *abdikov* | ...
  - Suffix → λ | *a* | *ovi* | *e* | *em* | *y* | *u* | *o* | *ou* | ...
- Distinguish stems that permit concrete groups of affixes
- Solve irregularities, alternations of stem-final consonants, ...
- Problem: The grammar would be too large!

Example Czech Paradigms: *žena* "woman", *matka* "mother"

- *žena* – *ženy* (sg. – pl.)
- *ženy* – *žen*
- *ženě* – *ženám*
- *ženu* – *ženy*
- *ženo* – *ženy*
- *ženě* – *ženách*
- *ženou* – *ženami*

- *matka* – *matky* (nom)
- *matky* – *matek* (gen)
- *matce* – *matkám* (dat)
- *matku* – *matky* (acc)
- *matko* – *matky* (voc)
- *matce* – *matkách* (loc)
- *matkou* – *matkami* (ins)

Morphology and CFG: Too Many Paradigms

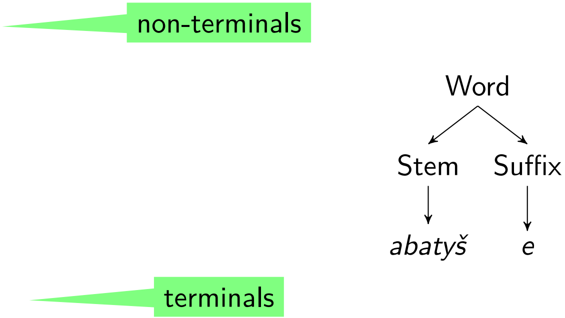
- *žena* "woman"
  - *žen* | *vlád* | *mát* | *láv* | ...
  - + *a* | *y* | *ě* | *u* | *o* | *ě* | *ou* |
    y | *λ* | *ám* | *y* | *y* | *ách* | *ami*
- *matka* "mother"
  - *mat* | *bab* | *vlaj* | ...
  - + *ka* | *ky* | *ce* | *ku* | *ko* | *ce* | *kou* |
    ky | *ek* | *kám* | *ky* | *ky* | *kách* | *kami*
- *banka* "bank"
  - *ban* | ...
  - + *ka* | *ky* | *ce* | *ku* | *ko* | *ce* | *kou* |
    ky | *k* | *kám* | *ky* | *ky* | *kách* | *kami*

Traditional school grammars of Czech assign all these words to the paradigm *žena* "woman"

Morphology and CFG: Too Many Paradigms

- *barva* "color"
  - *bar* | *lar* | *kur* | … | *bit* | *pit* | …
  - + *va* | *vy* | *vě* | *vu* | *vo* | *vě* | *vou* |
    vy | *ev* | *vám* | *vy* | *vy* | *vách* | *vami*
- *tráva* "grass"
  - *tr* | *kr* | *šť* | … (but not e.g. *k*!)
  - + *áva* | *ávy* | *ávě* | *ávu* | *ávo* | *ávě* | *ávou* |
    ávy | *av* | *ávám* | *ávy* | *ávy* | *ávách* | *ávami*
- *louka* "meadow"
  - *l* | *m* | … (but not e.g. *prv*, *mrav*!)
  - + *ouka* | *ouky* | *ouce* | *ouku* | *ouko* | *ouce* | *oukou* |
    ouky | *uk* | *oukám* | *ouky* | *ouky* | *oukách* | *oukami*

Example of a Derivation Tree



Context-Free Generation

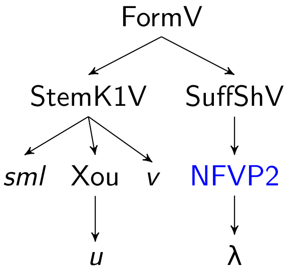
- Input:
  - `<l>matka<t>NNFS6-----A----`
- Expected output:
  - `<f>matce`
- Grammar:
  - FormMatka → StemMatka SuffMatka
  - StemMatka → mat | bab | vlaj | …
  - SuffMatka → MatS1 | MatS2 | MatS3 | MatS4 | …
  - MatS1 → ka ; MatS2 → ky ; MatS3 → ce ; MatS4 → ku ; …
  - MatP1 → ky ; MatP2 → ek ; MatP3 → kám ; MatP4 → ky ; …

Context-Free Generation

- Supplementary rule:
  - Names of some non-terminals contain information from morphological tags
  - In this particular case: the last two characters of non-terminals immediately under the non-terminal whose name begins with "Suff"
- In theory we could proceed like this:
  - First analyze the lemma *matka*. It will turn out that it consists of *mat* + `MatS1`
  - Replace by required `MatS6`

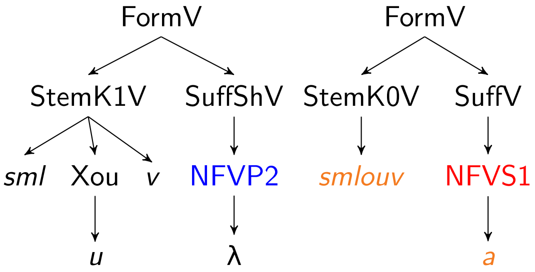
The Result of the Analysis

- After analysis, generate the base form (lemma; per definition it is `S1`). **`FormV`** refers to the correct paradigm



The Result of the Analysis

- After analysis, generate the base form (lemma; per definition it is **`S1`**). **`FormV`** refers to the correct paradigm



Context-Free Analysis and Generation

- Generation
  - Start with the start symbol
  - Choose a non-terminal in the current string. Choose a rule and rewrite the symbol.
    - When applicable, choose non-terminal by the supplementary rule
    - Sometimes (often!) we have to select one rule of many applicable
  - The string is complete if it contains only terminals

Context-Free Analysis and Generation

- Generation
  - Start with the start symbol
  - Choose a non-terminal in the current string. Choose a rule and rewrite the symbol.
    - When applicable, choose non-terminal by the supplementary rule
    - Sometimes (often!) we have to select one rule of many applicable
  - The string is complete if it contains only terminals

- Analysis
  - Start with a string of terminal symbols (characters in word form)
  - Look for parts that can be replaced by non-terminals. Non-deterministic!
  - Goal: the start symbol

Non-determinism

nesu "I carry", *beru* "I take", *mažu* "I grease", *jdu* "I go"

- `FormV13 → StemV13 SuffV13`
- `StemV13 → nes | ber | maž | jd | …`
- `SuffV13 → V13PS1 | V13PS2 | V13PS3 | V13PP1 | V13PP2 | V13PP3`
- `V13PS1 → u ; V13PS2 → eš ; V13PS3 → e`
- `V13PP1 → eme | em ; V13PP2 → ete ; V13PP3 → ou`

nesou (V13PS1)

Non-determinism

nesu “I carry”, *beru* “I take”, *mažu* “I grease”, *jdu* “I go”

- `FormV13 → StemV13 SuffV13`
- `StemV13 → nes | ber | maž | jd | …`
- `SuffV13 → V13PS1 | V13PS2 | V13PS3 | V13PP1 | V13PP2 | V13PP3`
- `V13PS1 → u ; V13PS2 → eš ; V13PS3 → e`
- `V13PP1 → eme | em ; V13PP2 → ete ; V13PP3 → ou`

nesou (V13PP3)

Non-determinism

nesu "I carry", *beru* "I take", *mažu* "I grease", *jdu* "I go"

- `FormV13` → `StemV13 SuffV13`
- `StemV13` → `nes | ber | maž | jd | …`
- `SuffV13` → `V13PS1 | V13PS2 | V13PS3 | V13PP1 | V13PP2 | V13PP3`
- `V13PS1` → `u ; V13PS2` → `eš ; V13PS3` → `e`
- `V13PP1` → `eme | em ; V13PP2` → `ete ; V13PP3` → `ou`

nesouc (converb)

Non-determinism

nesu "I carry", *beru* "I take", *mažu* "I grease", *jdu* "I go"

- `FormV13` → `StemV13 SuffV13`
- `StemV13` → `nes | ber | maž | jd | …`
- `SuffV13` → `V13PS1 | V13PS2 | V13PS3 | V13PP1 | V13PP2 | V13PP3`
- `V13PS1` → `u ; V13PS2` → `eš ; V13PS3` → `e`
- `V13PP1` → `eme | em ; V13PP2` → `ete ; V13PP3` → `ou`

nesoucí (present active participle-adjective)

Non-determinism

nesu “I carry”, *beru* “I take”, *mažu* “I grease”, *jdu* “I go”

- `FormV13 → StemV13 SuffV13`
- `StemV13 → nes | ber | maž | jd | …`
- `SuffV13 → V13PS1 | V13PS2 | V13PS3 | V13PP1 | V13PP2 | V13PP3`
- `V13PS1 → u ; V13PS2 → eš ; V13PS3 → e`
- `V13PP1 → eme | em ; V13PP2 → ete ; V13PP3 → ou`

nesoucím (dative plural of the participle-adjective)

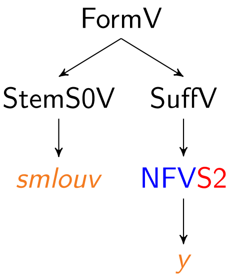
Non-determinism

nesu "I carry", *beru* "I take", *mažu* "I grease", *jdu* "I go"

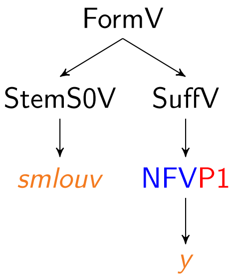
- `FormV13` → `StemV13 SuffV13`
- `StemV13` → `nes | ber | maž | jd | …`
- `SuffV13` → `V13PS1 | V13PS2 | V13PS3 | V13PP1 | V13PP2 | V13PP3`
- `V13PS1` → `u ; V13PS2` → `eš ; V13PS3` → `e`
- `V13PP1` → `eme | em ; V13PP2` → `ete ; V13PP3` → `ou`

nesoucími (instrumental plural of the participle-adjective)

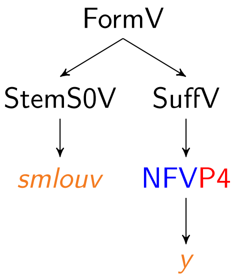
Homonymy



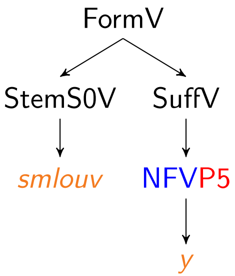
Homonymy



Homonymy



Homonymy



Algorithm for Context-Free Analysis

- Top-down
  - Start with one non-terminal – the start symbol

Algorithm for Context-Free Analysis

- Top-down
  - Start with one non-terminal – the start symbol
  - Expand one non-terminal: find a rule where it forms the left-hand side and replace it by the rule's right-hand side

Algorithm for Context-Free Analysis

- Top-down
  - Start with one non-terminal – the start symbol
  - Expand one non-terminal: find a rule where it forms the left-hand side and replace it by the rule's right-hand side
  - Repeat until there are no non-terminals

Algorithm for Context-Free Analysis

- Top-down
  - Start with one non-terminal – the start symbol
  - Expand one non-terminal: find a rule where it forms the left-hand side and replace it by the rule's right-hand side
  - Repeat until there are no non-terminals
  - If the resulting string is not the analyzed word, backtrack and choose a different rule for some non-terminal

Algorithm for Context-Free Analysis

- Top-down
  - Start with one non-terminal – the start symbol
  - Expand one non-terminal: find a rule where it forms the left-hand side and replace it by the rule's right-hand side
  - Repeat until there are no non-terminals
  - If the resulting string is not the analyzed word, backtrack and choose a different rule for some non-terminal
  - If the resulting string is the analyzed word, backtrack anyway as there may exist other analyses

Algorithm for Context-Free Analysis

- Top-down
  - Start with one non-terminal – the start symbol
  - Expand one non-terminal: find a rule where it forms the left-hand side and replace it by the rule's right-hand side
  - Repeat until there are no non-terminals
  - If the resulting string is not the analyzed word, backtrack and choose a different rule for some non-terminal
  - If the resulting string is the analyzed word, backtrack anyway as there may exist other analyses
  - If all combinations have been considered, the set of analyses is complete. Is it empty? Then the word is not in the language

Algorithm for Context-Free Analysis

- Bottom-up
  - Start with a sequence of terminals – the analyzed word

Algorithm for Context-Free Analysis

- Bottom-up
  - Start with a sequence of terminals – the analyzed word
  - Collapse one non-terminal: search the analyzed string for the right-hand side of a rule, replace by the left-hand-side non-terminal

Algorithm for Context-Free Analysis

- Bottom-up
  - Start with a sequence of terminals – the analyzed word
  - Collapse one non-terminal: search the analyzed string for the right-hand side of a rule, replace by the left-hand-side non-terminal
  - Repeat until no rule found

Algorithm for Context-Free Analysis

- Bottom-up
  - Start with a sequence of terminals – the analyzed word
  - Collapse one non-terminal: search the analyzed string for the right-hand side of a rule, replace by the left-hand-side non-terminal
  - Repeat until no rule found
  - If the result is not the start symbol, backtrack and choose a different rule at some point in the history

Algorithm for Context-Free Analysis

- Bottom-up
  - Start with a sequence of terminals – the analyzed word
  - Collapse one non-terminal: search the analyzed string for the right-hand side of a rule, replace by the left-hand-side non-terminal
  - Repeat until no rule found
  - If the result is not the start symbol, backtrack and choose a different rule at some point in the history
  - If the result is the start symbol, backtrack anyway as there may exist other analyses

Algorithm for Context-Free Analysis

- Bottom-up
  - Start with a sequence of terminals – the analyzed word
  - Collapse one non-terminal: search the analyzed string for the right-hand side of a rule, replace by the left-hand-side non-terminal
  - Repeat until no rule found
  - If the result is not the start symbol, backtrack and choose a different rule at some point in the history
  - If the result is the start symbol, backtrack anyway as there may exist other analyses
  - If all combinations have been considered, the set of analyses is complete. Is it empty? Then the word is not in the language

Top-Down Analysis

- Somehow we must enforce heading to the terminal string being analyzed
- Solution: continuously check that the terminals in the current *state* correspond to a prefix of the string
- State of the analysis: string of terminals and non-terminals, a period delimits the checked (read) prefix

Example of Top-Down Analysis

Analyzing the word *matce*

. `Form`

Example of Top-Down Analysis

Analyzing the word *matce*

- `Form`
- `FormNFeka`

Example of Top-Down Analysis

Analyzing the word *matce*

- `Form`
- `FormNFeka`
- `StemNFeka SuffNFeka`

Example of Top-Down Analysis

Analyzing the word *matce*

- `Form`
- `FormNFeka`
- `StemNFeka SuffNFeka`
- `bár SuffNFeka`

Example of Top-Down Analysis

Analyzing the word *matce*

- `Form`
- `FormNFeka`
- `StemNFeka SuffNFeka`
- `bud SuffNFeka`

Example of Top-Down Analysis

Analyzing the word *matce*

- `Form`
- `FormNFeka`
- `StemNFeka SuffNFeka`
- … `SuffNFeka`

Example of Top-Down Analysis

Analyzing the word *matce*

- `Form`
- `FormNFeka`
- `StemNFeka SuffNFeka`
- `mat SuffNFeka`

Example of Top-Down Analysis

Analyzing the word *matce*

```
. Form
. FormNFeka
. StemNFeka SuffNFeka
. mat SuffNFeka
mat . SuffNFeka
```

Example of Top-Down Analysis

Analyzing the word *matce*

```
. Form
. FormNFeka
. StemNFeka SuffNFeka
. mat SuffNFeka
mat . SuffNFeka
mat . NFekaS1
```

Example of Top-Down Analysis

Analyzing the word *matce*

```
. Form
. FormNFeka
. StemNFeka SuffNFeka
. mat SuffNFeka
mat . SuffNFeka
mat . NFekaS1
mat . ka
```

Example of Top-Down Analysis

Analyzing the word *matce*

```
. Form
. FormNFeka
. StemNFeka SuffNFeka
. mat SuffNFeka
mat . SuffNFeka
mat . NFekaS1
mat . ka
mat . NFekaS2
```

Example of Top-Down Analysis

Analyzing the word *matce*

```
. Form
. FormNFeka
. StemNFeka SuffNFeka
. mat SuffNFeka
mat . SuffNFeka
mat . NFekaS1
mat . ka
mat . NFekaS2
mat . ky
```

Example of Top-Down Analysis

Analyzing the word *matce*

```
. Form
. FormNFeka
. StemNFeka SuffNFeka
. mat SuffNFeka
mat . SuffNFeka
mat . NFekaS1
mat . ka
mat . NFekaS2
mat . ky
mat . NFekaS3
```

Example of Top-Down Analysis

Analyzing the word *matce*

```
. Form
. FormNFeka
. StemNFeka SuffNFeka
. mat SuffNFeka
mat . SuffNFeka
mat . NFekaS1
mat . ka
mat . NFekaS2
mat . ky
mat . NFekaS3
mat . ce
```

Example of Top-Down Analysis

Analyzing the word *matce*

```
. Form
. FormNFeka
. StemNFeka SuffNFeka
. mat SuffNFeka
mat . SuffNFeka
mat . NFekaS1
mat . ka
mat . NFekaS2
mat . ky
mat . NFekaS3
mat . ce
matce . ☺
```

An Observation about the Lexicon

- In practice the lexicon should be separated and implemented more effectively

- The last non-terminal above the lexicon is so-called pre-terminal
- It knows the list of strings belonging to it and it can search the strings quickly
- Implementation: hash table, search tree, trie...

An Observation about (Left) Recursion

- cs: iteratives: *dělat – dělávat – dělávávat – dělávávávat – dělávávávávat…*
- Form → FormV5
- FormV5 → StemV5 Iter SuffV5 | StemV5 SuffV5
- StemV5 → děl | lét | …
- SuffV5 → V5INF | V5PS1 | V5PS2 | …
- Iter → Iter áv | áv
- V5INF → at | ati ; V5PS1 → ám ; V5PS2 → áš

. Form

An Observation about (Left) Recursion

- cs: iteratives: *dělat – dělávat – dělávávat – dělávávávat – dělávávávávat...*
- `Form → FormV5`
- `FormV5 → StemV5 Iter SuffV5 | StemV5 SuffV5`
- `StemV5 → děl | lét | …`
- `SuffV5 → V5INF | V5PS1 | V5PS2 | …`
- `Iter → Iter áv | áv`
- `V5INF → at | ati ; V5PS1 → ám ; V5PS2 → áš`

. FormV5

An Observation about (Left) Recursion

- cs: iteratives: *dělat – dělávat – dělávávat – dělávávávat – dělávávávávat…*

- `Form → FormV5`
- `FormV5 → StemV5 Iter SuffV5 | StemV5 SuffV5`
- `StemV5 → děl | lét | …`
- `SuffV5 → V5INF | V5PS1 | V5PS2 | …`
- `Iter → Iter áv | áv`
- `V5INF → at | ati ; V5PS1 → ám ; V5PS2 → áš`

. StemV5 Iter SuffV5

An Observation about (Left) Recursion

- cs: iteratives: *dělat – dělávat – dělávávat – dělávávávat – dělávávávávat…*

- `Form → FormV5`
- `FormV5 → StemV5 Iter SuffV5 | StemV5 SuffV5`
- `StemV5 → děl | lét | …`
- `SuffV5 → V5INF | V5PS1 | V5PS2 | …`
- `Iter → Iter áv | áv`
- `V5INF → at | ati ; V5PS1 → ám ; V5PS2 → áš`

. `děl Iter SuffV5`

An Observation about (Left) Recursion

- cs: iteratives: *dělat – dělávat – dělávávat – dělávávávat – dělávávávávat…*

- `Form → FormV5`
- `FormV5 → StemV5 Iter SuffV5 | StemV5 SuffV5`
- `StemV5 → děl | lét | …`
- `SuffV5 → V5INF | V5PS1 | V5PS2 | …`
- `Iter → Iter áv | áv`
- `V5INF → at | ati ; V5PS1 → ám ; V5PS2 → áš`

`děl . Iter SuffV5`

An Observation about (Left) Recursion

- cs: iteratives: *dělat – dělávat – dělávávat – dělávávávat – dělávávávávat...*

- `Form → FormV5`
- `FormV5 → StemV5 Iter SuffV5 | StemV5 SuffV5`
- `StemV5 → děl | lét | …`
- `SuffV5 → V5INF | V5PS1 | V5PS2 | …`
- `Iter → Iter áv | áv`
- `V5INF → at | ati ; V5PS1 → ám ; V5PS2 → áš`

`děl . Iter áv SuffV5`

An Observation about (Left) Recursion

- cs: iteratives: *dělat – dělávat – dělávávat – dělávávávat – dělávávávávat…*

- `Form → FormV5`
- `FormV5 → StemV5 Iter SuffV5 | StemV5 SuffV5`
- `StemV5 → děl | lét | …`
- `SuffV5 → V5INF | V5PS1 | V5PS2 | …`
- `Iter → Iter áv | áv`
- `V5INF → at | ati ; V5PS1 → ám ; V5PS2 → áš`

`děl . Iter áv áv SuffV5`

An Observation about (Left) Recursion

- cs: iteratives: *dělat – dělávat – dělávávat – dělávávávat – dělávávávávat…*
- `Form` → `FormV5`
- `FormV5` → `StemV5 Iter SuffV5 | StemV5 SuffV5`
- `StemV5` → `děl | lét | …`
- `SuffV5` → `V5INF | V5PS1 | V5PS2 | …`
- `Iter` → `Iter áv | áv`
- `V5INF` → `at | ati ; V5PS1` → `ám ; V5PS2` → `áš`

`děl . Iter áv áv áv SuffV5`

An Observation about (Left) Recursion

- cs: iteratives: *dělat – dělávat – dělávávat – dělávávávat – dělávávávávat...*

- Form → FormV5
- FormV5 → StemV5 Iter SuffV5 | StemV5 SuffV5
- StemV5 → děl | lét | ...
- SuffV5 → V5INF | V5PS1 | V5PS2 | ...
- Iter → Iter áv | áv
- V5INF → at | ati ; V5PS1 → ám ; V5PS2 → áš

děl . Iter áv áv áv áv SuffV5 ☹

Recursion and Infinite-Loop Prevention

- Convert the grammar to a non-left-recursive one
- Make sure that the recurrent rules are not used until necessary
- If there are more than one recurrent rule, try all combinations
- There is a finite number of combinations—the recursion of any rule can be stopped once the number of symbols is greater than the number of input terminals

- Ban left recursion, permit right recursion
  (`Iter` $\rightarrow$ `áv | áv Iter`)
- Perform bottom-up analysis

Example of Bottom-Up Analysis

Analyzing the word *matce*

```
. matce
```

Example of Bottom-Up Analysis

Analyzing the word *matce*

. **`matce`**

. **`NNíS7 atce`**

Example of Bottom-Up Analysis

Analyzing the word *matce*

- `matce`
- `NNíS7 atce`
- `SuffNNí atce`

Example of Bottom-Up Analysis

Analyzing the word *matce*

- `matce`
- `NNíS7 atce`
- `SuffNNí atce`
- `NFaD7 tce`

Example of Bottom-Up Analysis

Analyzing the word *matce*

- `matce`
- `NNíS7 atce`
- `SuffNNí atce`
- `NFaD7 tce`
- `SuffNFa tce`

Example of Bottom-Up Analysis

Analyzing the word *matce*

- `matce`
- `NNíS7 atce`
- `SuffNNí atce`
- `NFaD7 tce`
- `SuffNFa tce`
- `StemNFeka ce`

Example of Bottom-Up Analysis

Analyzing the word *matce*

```
. matce
. NNíS7 atce
. SuffNNí atce
. NFaD7 tce
. SuffNFa tce
. StemNFeka ce
StemNFeka . ce
```

Example of Bottom-Up Analysis

Analyzing the word *matce*

```
. matce
. NNíS7 atce
. SuffNNí atce
. NFaD7 tce
. SuffNFa tce
. StemNFeka ce
StemNFeka . ce
StemNFeka . NFekaS3
```

Example of Bottom-Up Analysis

Analyzing the word *matce*

```
. matce
. NNíS7 atce
. SuffNNí atce
. NFaD7 tce
. SuffNFa tce
. StemNFeka ce
StemNFeka . ce
StemNFeka . NFekaS3
StemNFeka . SuffNFeka
```

Example of Bottom-Up Analysis

Analyzing the word *matce*

```
. matce
. NNíS7 atce
. SuffNNí atce
. NFaD7 tce
. SuffNFa tce
. StemNFeka ce
StemNFeka . ce
StemNFeka . NFekaS3
StemNFeka . SuffNFeka
StemNFeka SuffNFeka .
```

Example of Bottom-Up Analysis

Analyzing the word *matce*

```
. matce
. NNíS7 atce
. SuffNNí atce
. NFaD7 tce
. SuffNFa tce
. StemNFeka ce
StemNFeka . ce
StemNFeka . NFekaS3
StemNFeka . SuffNFeka
StemNFeka SuffNFeka .
FormNFeka .
```

Example of Bottom-Up Analysis

Analyzing the word *matce*

```
. matce
. NNíS7 atce
. SuffNNí atce
. NFaD7 tce
. SuffNFa tce
. StemNFeka ce
StemNFeka . ce
StemNFeka . NFekaS3
StemNFeka . SuffNFeka
StemNFeka SuffNFeka .
FormNFeka .
Form .
```

☺

How to Remember Alternate Paths

- In case of crash we must return to the last fork where more rules were available
- So we must remember the forks

- Possibility: stack of alternate states
- At fork, don't just generate one new state. Generate all possible continuations. Store them on a stack
- Pick the top state from the stack, make it the current state, go on with it
- In case of crash discard the state and pick the next one from the stack

Context-Free Analysis as a Path Searching Problem

- The analysis can be viewed as finding a path in a tree of possibilities from the root to the leaves
- Depth-first search: the list of possibilities is a stack (LIFO)
- Breadth-first search: the list of possibilities is a queue (FIFO)

- Breadth-first search requires more memory for alternate states but it faces fewer recursion-related problems

The Bottom-Up Algorithm

- Pick the next state from the stack (queue) and make it the current state

The Bottom-Up Algorithm

- Pick the next state from the stack (queue) and make it the current state
- Consider all substrings of the current state that end at the period
  - Compare them to the right-hand sides of all rules
  - For every match generate a new state:
    - Detected right-hand side is replaced with left-hand side of the given rule
    - The rest of the state is identical to the current state
  - Put the newly generated state on the stack

The Bottom-Up Algorithm

- Pick the next state from the stack (queue) and make it the current state
- Consider all substrings of the current state that end at the period
  - Compare them to the right-hand sides of all rules
  - For every match generate a new state:
    - Detected right-hand side is replaced with left-hand side of the given rule
    - The rest of the state is identical to the current state
  - Put the newly generated state on the stack
- Finally generate a state with shifted period
  - Only difference from current state: period shifted 1 symbol to the right
  - Put the new state on the stack

Example of Bottom-Up Analysis Including the Stack

Grammar

S $\rightarrow$ C D

C $\rightarrow$ c | B C

D $\rightarrow$ d | d C

B $\rightarrow$ b | ab

Current State

. a b c d b c

Stack

Example of Bottom-Up Analysis Including the Stack

Grammar

S → C D

C → c | B C

D → d | d C

B → b | ab

Current State

. a b c d b c

Stack

a . b c d b c

Example of Bottom-Up Analysis Including the Stack

Grammar

S → C D

C → c | B C

D → d | d C

B → b | ab

Current State

a . b c d b c

Stack

Example of Bottom-Up Analysis Including the Stack

Grammar

S → C D

C → c | B C

D → d | d C

B → b | ab

Current State

a . b c d b c

Stack

a b . c d b c

Example of Bottom-Up Analysis Including the Stack

Grammar

S → C D

C → c | B C

D → d | d C

B → b | ab

Current State

a b . c d b c

Stack

Example of Bottom-Up Analysis Including the Stack

Grammar

S → C D

C → c | B C

D → d | d C

B → b | ab

Current State

a b . c d b c

Stack

a b c . d b c

B . c d b c

a B . c d b c

Example of Bottom-Up Analysis Including the Stack

Grammar

S → C D

C → c | B C

D → d | d C

B → b | ab

Current State

a b c . d b c

Stack

B . c d b c

a B . c d b c

Example of Bottom-Up Analysis Including the Stack

Grammar

S → C D

C → c | B C

D → d | d C

B → b | ab

Current State

a b c . d b c

Stack

a b c d . b c

a b C . d b c

B . c d b c

a B . c d b c

Example of Bottom-Up Analysis Including the Stack

Grammar

S → C D

C → c | B C

D → d | d C

B → b | ab

Current State

a b c d . b c

Stack

a b C . d b c

B . c d b c

a B . c d b c

Example of Bottom-Up Analysis Including the Stack

Grammar

S → C D

C → c | B C

D → d | d C

B → b | ab

Current State

a b c d . b c

Stack

a b c d b . c

a b c D . b c

a b C . d b c

B . c d b c

a B . c d b c

Example of Bottom-Up Analysis Including the Stack

Grammar

S → C D

C → c | B C

D → d | d C

B → b | ab

Current State

a b c d b . c

Stack

a b c D . b c

a b C . d b c

B . c d b c

a B . c d b c

Example of Bottom-Up Analysis Including the Stack

Grammar

S → C D

C → c | B C

D → d | d C

B → b | ab

Current State

a b c d b . c

Stack

a b c d b c .

a b c d B . c

a b c D . b c

a b C . d b c

B . c d b c

a B . c d b c

Example of Bottom-Up Analysis Including the Stack

Grammar

S → C D

C → c | B C

D → d | d C

B → b | ab

Current State

a b c d b c .

Stack

a b c d B . c

a b c D . b c

a b C . d b c

B . c d b c

a B . c d b c

Example of Bottom-Up Analysis Including the Stack

Grammar

S → C D

C → c | B C

D → d | d C

B → b | ab

Current State

a b c d b c .

Stack

a b c d b C .

a b c d B . c

a b c D . b c

a b C . d b c

B . c d b c

a B . c d b c

Example of Bottom-Up Analysis Including the Stack

Grammar

S → C D

C → c | B C

D → d | d C

B → b | ab

Current State

a b c d b C .

Stack

a b c d B . c

a b c D . b c

a b C . d b c

B . c d b c

a B . c d b c

Example of Bottom-Up Analysis Including the Stack

Grammar

S → C D

C → c | B C

D → d | d C

B → b | ab

Current State

a b c d b C .

Stack

a b c d B . c

a b c D . b c

a b C . d b c

B . c d b c

a B . c d b c

Example of Bottom-Up Analysis Including the Stack

Grammar

S → C D
C → c | B C
D → d | d C
B → b | ab

Current State

a b c d B . C

Stack

a b c D . b c
a b C . d b c
B . c d b c
a B . c d b c

Example of Bottom-Up Analysis Including the Stack

Grammar

S → C D

C → c | B C

D → d | d C

B → b | ab

Current State

a b c d B . C

Stack

a b c d B c .

a b c D . b c

a b C . d b c

B . c d b c

a B . c d b c

Example of Bottom-Up Analysis Including the Stack

Grammar

S $\rightarrow$ C D

C $\rightarrow$ c | B C

D $\rightarrow$ d | d C

B $\rightarrow$ b | ab

Current State

a b c d B c .

Stack

a b c D . b c

a b C . d b c

B . c d b c

a B . c d b c

Example of Bottom-Up Analysis Including the Stack

Grammar

S → C D

C → c | B C

D → d | d C

B → b | ab

Current State

a b c d B c .

Stack

a b c d B C .

a b c D . b c

a b C . d b c

B . c d b c

a B . c d b c

Example of Bottom-Up Analysis Including the Stack

Grammar

S → C D

C → c | B C

D → d | d C

B → b | ab

Current State

a b c d B C .

Stack

a b c D . b c

a b C . d b c

B . c d b c

a B . c d b c

Example of Bottom-Up Analysis Including the Stack

Grammar

S → C D

C → c | B C

D → d | d C

B → b | ab

Current State

a b c d B C .

Stack

a b c d C .

a b c D . b c

a b C . d b c

B . c d b c

a B . c d b c

Example of Bottom-Up Analysis Including the Stack

Grammar

S → C D
C → c | B C
D → d | d C
B → b | ab

Current State

a b c d C .

Stack

a b c D . b c
a b C . d b c
B . c d b c
a B . c d b c

Example of Bottom-Up Analysis Including the Stack

Grammar

S → C D
C → c | B C
D → d | d C
B → b | ab

Current State

a b c d C .

Stack

a b c D .
a b c D . b c
a b C . d b c
B . c d b c
a B . c d b c

Example of Bottom-Up Analysis Including the Stack

Grammar

S → C D
C → c | B C
D → d | d C
B → b | ab

Current State

a b c D .

Stack

a b c D . b c
a b C . d b c
B . c d b c
a B . c d b c

Example of Bottom-Up Analysis Including the Stack

Grammar

S → C D

C → c | B C

D → d | d C

B → b | ab

Current State

a b c D .

Stack

a b c D . b c

a b C . d b c

B . c d b c

a B . c d b c

Example of Bottom-Up Analysis Including the Stack

Grammar

S → C D

C → c | B C

D → d | d C

B → b | ab

Current State

a b c D . b c

Stack

a b C . d b c

B . c d b c

a B . c d b c

Example of Bottom-Up Analysis Including the Stack

Grammar

S → C D

C → c | B C

D → d | d C

B → b | ab

Current State

a b c D . b c

Stack

a b c D b . c

a b C . d b c

B . c d b c

a B . c d b c

Example of Bottom-Up Analysis Including the Stack

Grammar

S $\rightarrow$ C D

C $\rightarrow$ c | B C

D $\rightarrow$ d | d C

B $\rightarrow$ b | ab

Current State

a b c D b . c

Stack

a b C . d b c

B . c d b c

a B . c d b c

Example of Bottom-Up Analysis Including the Stack

Grammar

S → C D
C → c | B C
D → d | d C
B → b | ab

Current State

a b c D b . c

Stack

a b c D b c .
a b c D **B** . c
a b C . d b c
B . c d b c
a B . c d b c

Example of Bottom-Up Analysis Including the Stack

Grammar

S → C D

C → c | B C

D → d | d C

B → b | ab

Current State

a b c d b . c

a b c D b . c

...

a b C d b . c

B c d b . c

a B c d b . c

...

Stack

a b c D b c .

a b c D **B** . c

a b C . d b c

B . c d b c

a B . c d b c

The same right-hand side is recognized at the same position over and over!

Computational Complexity

- Described algorithm is exponential (all paths in a tree must be considered)
  - Problem: The same right-hand side is repeatedly compared and recognized at the same position
- There is a polynomial algorithm: CYK, *chart parser*

Chart Parser

- *Chart* [ča:t] = „přehled, diagram"
  - The principal data structure in the parser
  - It remembers what right-hand sides have been recognized and where
- A note on Czech terminology: *chart parser* could in theory be translated as *analýza s přehledem* but in practice the original English term is used
- Chart parsing is a special case of dynamic programming

Do Not Look for All Combinations. Store Each Constituent Separately!

Grammar

$S \rightarrow C\ D$

$C \rightarrow c \mid B\ C$

$D \rightarrow d \mid d\ C$

$B \rightarrow b \mid ab$

String

${}_0a_1b_2c_3d_4b_5c_6$

Chart

B 0 2

B 1 2

C 2 3

C 1 3

...

S 0 6

State of Analysis

- The input is read one-by-one terminal symbol
- A right-hand side of a rule can be recognized after any terminal
- In addition, the chart contains a list of rules whose right-hand sides are partially read:
  - The period delimits the part of the right-hand side that has been recognized
  - We know the positions in the input string where the right-hand side began and where it currently ends (where the period is)

  - Example: `(B → a . b) (0;1)`

The Chart

- Agenda. List of constituents that have been recognized in the input and are waiting to be processed. The span of each constituent is saved (start and end positions in the input)

- List of "active arcs". Right-hand sides that have been partially recognized in the input. The span of each is saved (start position of the right-hand side and the position of the period—position up to which the right-hand side has been recognized)

- List of processed constituents. The span of each constituent is saved. Constituents are moved here from the Agenda after they have been processed

Chart Parsing Algorithm

1. Start with empty agenda, list of active arcs and list of processed constituents

Chart Parsing Algorithm

1. Start with empty agenda, list of active arcs and list of processed constituents
2. If agenda is empty:
   - If input is empty: exit and return all processed $(S, 0, n)$
   - If input is not empty: read next terminal, add it to agenda

Chart Parsing Algorithm

1. Start with empty agenda, list of active arcs and list of processed constituents
2. If agenda is empty:
   - If input is empty: exit and return all processed $(S, 0, n)$
   - If input is not empty: read next terminal, add it to agenda
3. Pick new current constituent (C, i, j) from agenda. It spans the input substring between positions i and j

Chart Parsing Algorithm

1. Start with empty agenda, list of active arcs and list of processed constituents
2. If agenda is empty:
   - If input is empty: exit and return all processed $(S, 0, n)$
   - If input is not empty: read next terminal, add it to agenda
3. Pick new current constituent (C, i, j) from agenda. It spans the input substring between positions i and j
4. Consider all grammar rules. For every rule $X \rightarrow C X_1 \dots X_n$ add new active arc from i to i of the form $X \rightarrow \bullet C X_1 \dots X_n$ (new rules that start here)

Chart Parsing Algorithm

1. Start with empty agenda, list of active arcs and list of processed constituents
2. If agenda is empty:
   - If input is empty: exit and return all processed $(S, 0, n)$
   - If input is not empty: read next terminal, add it to agenda
3. Pick new current constituent (C, i, j) from agenda. It spans the input substring between positions i and j
4. Consider all grammar rules. For every rule $X \to C X_1 \dots X_n$ add new active arc from i to i of the form $X \to \bullet C X_1 \dots X_n$ (new rules that start here)
5. For every active arc from k to i of the form $X \to X_1 \dots \bullet C \dots X_n$ add new active arc from k to j of the form $X \to X_1 \dots C \bullet \dots X_n$ (rules that continue here)

Chart Parsing Algorithm

1. Start with empty agenda, list of active arcs and list of processed constituents
2. If agenda is empty:
   - If input is empty: exit and return all processed $(S, 0, n)$
   - If input is not empty: read next terminal, add it to agenda
3. Pick new current constituent (C, i, j) from agenda. It spans the input substring between positions i and j
4. Consider all grammar rules. For every rule $X \to C X_1 \dots X_n$ add new active arc from i to i of the form $X \to \bullet C X_1 \dots X_n$ (new rules that start here)
5. For every active arc from k to i of the form $X \to X_1 \dots \bullet\, C \dots X_n$ add new active arc from k to j of the form $X \to X_1 \dots C \bullet \dots X_n$ (rules that continue here)
6. For every active arc from k to j of the form $X \to X_1 \dots X_n C \bullet$ add to agenda a new constituent X spanning k to j unless it is already in agenda or in the list of processed constituents (rules that end here)

Chart Parsing Algorithm

1. Start with empty agenda, list of active arcs and list of processed constituents
2. If agenda is empty:
   - If input is empty: exit and return all processed $(S, 0, n)$
   - If input is not empty: read next terminal, add it to agenda
3. Pick new current constituent (C, i, j) from agenda. It spans the input substring between positions i and j
4. Consider all grammar rules. For every rule $X \rightarrow C X_1 \dots X_n$ add new active arc from i to i of the form $X \rightarrow \bullet C X_1 \dots X_n$ (new rules that start here)
5. For every active arc from k to i of the form $X \rightarrow X_1 \dots \bullet C \dots X_n$ add new active arc from k to j of the form $X \rightarrow X_1 \dots C \bullet \dots X_n$ (rules that continue here)
6. For every active arc from k to j of the form $X \rightarrow X_1 \dots X_n C \bullet$ add to agenda a new constituent X spanning k to j unless it is already in agenda or in the list of processed constituents (rules that end here)
7. Move (C, i, j) from agenda to list of processed constituents. If $C = S$ and i, j spans the entire input, then we found one of the analyses

Chart Parsing Algorithm

1. Start with empty agenda, list of active arcs and list of processed constituents
2. If agenda is empty:
   - If input is empty: exit and return all processed $(S, 0, n)$
   - If input is not empty: read next terminal, add it to agenda
3. Pick new current constituent (C, i, j) from agenda. It spans the input substring between positions i and j
4. Consider all grammar rules. For every rule $X \rightarrow C X_1 \dots X_n$ add new active arc from i to i of the form $X \rightarrow \bullet C X_1 \dots X_n$ (new rules that start here)
5. For every active arc from k to i of the form $X \rightarrow X_1 \dots \bullet C \dots X_n$ add new active arc from k to j of the form $X \rightarrow X_1 \dots C \bullet \dots X_n$ (rules that continue here)
6. For every active arc from k to j of the form $X \rightarrow X_1 \dots X_n C \bullet$ add to agenda a new constituent X spanning k to j unless it is already in agenda or in the list of processed constituents (rules that end here)
7. Move (C, i, j) from agenda to list of processed constituents. If $C = S$ and i, j spans the entire input, then we found one of the analyses
8. Go back to 2

Complexity of Chart Parsing

- Polynomial: $O(gn^3)$ where
  - n ... number of input terminals
  - g ... number of grammar rules

- There are $\frac{(n+1)^2}{2}$ spans (from i to j)
- Maximum number of constituents within one span is less than or equal to the number of grammar rules
- Maximum number of states in which one partially recognized rule can find itself is $n + 1$ (number of possible period positions)

Keep All States of Each Rule Forever!

- *For every active arc from k to i of the form $X \rightarrow X_1 \ldots \bullet C \ldots X_n$ add new active arc from k to j of the form $X \rightarrow X_1 \ldots C \bullet \ldots X_n$ (rules that continue here)*

- After shifting the period keep both the new and the old state of the rule!
- What if the same constituent is recognized later, it starts at the same position but is longer?

Example

Constituents

b (0,1)

Active arcs

B $\rightarrow$ b . A (0,1)

Grammar

A $\rightarrow$ a | a A

B $\rightarrow$ b A

...

Input

b a a c

Example

Grammar

A → a | a A
B → b A
...

Input

b a a c

Active arcs

B → b . A (0,1)
A → a . (1,2)
B → b A . (0,2)

Constituents

b (0,1)
a (1,2)
A (1,2)
B (0,2)

- Recognize $(A, 1, 2)$ but don't discard this active arc!

Example

Grammar

A → a | a A
B → b A
...

Input

b a a c

Active arcs

B → b . A (0,1)
A → a . (1,2)
B → b A . (0,2)
A → a . (2,3)
A → a A . (1,3)
B → b A . (0,3)
...

Constituents

b (0,1)
a (1,2)
A (1,2)
B (0,2)
a (2,3)
A (2,3)
A (1,3)
B (0,3)
...

- Recognize $(A, 1, 2)$ but don't discard this active arc!
- If we later recognize $(A, 1, 3)$ we will need the active arc again!

How to Remember the Analysis?

- So far we can only figure out whether there is an analysis, i.e., whether the string is accepted by the grammar

- We need to know the derivation tree, too. We will use it to read the actual output:
  - `Form ( FormNFeka ( StemNFeka ( m a t ) SuffNFeka ( NFekaS1 ( k a ) ) ) )`

How to Remember the Analysis?

- *For every active arc from k to j of the form $X \rightarrow X_1 \ldots X_n C \bullet$ add to agenda a new constituent X spanning k to j unless it is already in agenda or in the list of processed constituents (rules that end here)*

- Keep with every constituent the information what it is composed of. Same for every partially recognized rule.
- Caution: The same constituent spanning the same input substring may have arisen in several alternate ways!

How to Remember the Analysis?

- S → A | Ab
- A → a | ab
- string *ab*
  - S (A (a b))
  - S (A (a) b)

```
$agenda[0][2]{'S'}{description} = 'S:0:2';
push(@composition, [$agenda[0][1]{'A'}, $agenda[1][2]{'b'}]);
push(@composition, [$agenda[0][2]{'A'}]);
push(@{$agenda[0][2]{'S'}{composition}}, \@composition);
# Print j-th constituent of i-th composition of constituent S:0:2.
print $agenda[0][2]{'S'}{composition}[$i][$j]{description};
```

```
$agenda[$i][$j]{$N}{composition}[$k][$l]
```

- constituent starts at position i
- constituent ends at position j
- constituent is labeled by symbol N

Chart Parsing Example

Grammar

S → C D
C → c | B C
D → d | d C
B → b | ab

String

a b c d b c

Chart

					c	6
				b		5
			d			4
		c				3
	b					2
a						1
0	1	2	3	4	5	

Chart Parsing Example

Grammar

S → C D
C → c | B C
D → d | d C
B → b | ab

String

a b c d b c

Chart

					c, C	6
				b, B		5
			d, D			4
		c, C				3
	b, B					2
a						1
0	1	2	3	4	5	

Chart Parsing Example

Grammar

S → C D
C → c | B C
D → d | d C
B → b | ab

String

a b c d b c

Chart

				C	c, C	6
				b, B		5
		S	d, D			4
	C	c, C				3
B	b, B					2
a						1
0	1	2	3	4	5	

Chart Parsing Example

Grammar

S → C D

C → c | B C

D → d | d C

B → b | ab

String

a b c d b c

Chart

			D	C	c, C	6
				b, B		5
	S	S	d, D			4
C	C	c, C				3
B	b, B					2
a						1
0	1	2	3	4	5	

Chart Parsing Example

Grammar

S → C D
C → c | B C
D → d | d C
B → b | ab

String

a b c d b c

Chart

		S	D	C	c, C	6
				b, B		5
S	S	S	d, D			4
C	C	c, C				3
B	b, B					2
a						1
0	1	2	3	4	5	

Chart Parsing Example

Grammar

S → C D

C → c | B C

D → d | d C

B → b | ab

String

a b c d b c

Chart

	S	S	D	C	c, C	6
				b, B		5
S	S	S	d, D			4
C	C	c, C				3
B	b, B					2
a						1
0	1	2	3	4	5	

Chart Parsing Example

Grammar

S → C D
C → c | B C
D → d | d C
B → b | ab

String

a b c d b c

Chart

S	S	S	D	C	c, C	6
				b, B		5
S	S	S	d, D			4
C	C	c, C				3
B	b, B					2
a						1
0	1	2	3	4	5	

Context-Free Grammars and Morphological Analysis: Summary

- ☺ They nicely describe regular phenomena
- ☺ They can describe long-distance dependencies!
- ☹ "Regular irregularities" may require operations that are not directly supported by CFGs, simulation required
- ☹ The grammar grows unbearably
- ☹ High number of inflection classes ⇒ we need good maintenance tools. When the user is to add a new word we cannot reasonably require the word to be assigned one of 30 almost identical paradigms