Implementation of PCFG parsers
Praktikum NLP

Thomas Ruprecht    Richard Mörbitz

Chair for foundations of programming
Institute for theoretical computer science
TU Dresden

May 10th
Outline

1. Overview

2. CKY parsing

3. Deductive parsing

4. Conclusion
Overview

Parser invocation:
```
./pcfg_tool parse grammar.rules grammar.lexicon < sentences
```

Parser mainloop:
1. read PCFG $G$ from files
2. read sentence $w$ (1 line) from stdin
3. $G, w$ weighted CKY parsing or deductive parsing
4. print best derivation tree $d$
Outline

1. Overview
2. CKY parsing
3. Deductive parsing
4. Conclusion
The CKY parsing algorithm (formal notation)

**Require:** CFG $G = (N, \Sigma, R, S)$ in CNF, sentence $w = w_1...w_n$ where $w_1, ..., w_n \in \Sigma$

**Ensure:** family of sets $(c_{i,j} \subseteq N \mid 0 \leq i < j \leq n)$ such that $A \in c_{i,j} \iff D^A_G(w_{i+1}...w_j) \neq \emptyset$

1: function \texttt{cky}(R, w_1...w_n)
2: \hspace{1em} for $i := 1$ to $n$ do
3: \hspace{2em} $c_{i-1,i} := \{A \mid A \rightarrow w_i \in R\}$
4: \hspace{1em} for $r := 2$ to $n$ do
5: \hspace{2em} \hspace{1em} for $i := 0$ to $n - r$ do
6: \hspace{3em} $j := i + r$
7: \hspace{2em} \hspace{2em} $c_{i,j} := \{A \mid i < m < j, A \rightarrow BC \in R : B \in c_{i,m}, C \in c_{m,j}\}$
8: \hspace{1em} return $(c_{i,j} \mid 0 \leq i < j \leq n)$

where

- CNF = Chomsky normal form
- $D^A_G(w_i ... w_j)$ = set of (left) derivations of $A$ in $G$ that result in $w_i ... w_j$
- $c_{i,j}$ = set of all nonterminals that derive $w_{i+1} ... w_j$
The CKY parsing algorithm (imperative flavor)

Require: CFG $G = (N, \Sigma, R, S)$ in CNF, sentence $w = w_1...w_n$ where $w_1, ..., w_n \in \Sigma$
Ensure: family of sets $(c_{i,j} \subseteq N \mid 0 \leq i < j \leq n)$ such that $A \in c_{i,j} \iff D^A_G(w_{i+1}...w_j) \neq \emptyset$

1: function cky($R$, $w_1...w_n$) 
2: $(c_{i,j} := \emptyset \mid 0 \leq i < j \leq n)$ 
3: for $i := 1$ to $n$ do 
4: \hspace{1em} for $A \rightarrow w_i \in R$ do 
5: \hspace{2em} $c_{i-1,i} := c_{i-1,i} \cup \{A\}$ \hspace{1em} \hspace{1em} $c_{i-1,i} := \{A \mid A \rightarrow w_i \in R\}$ 
6: for $r := 2$ to $n$ do 
7: \hspace{1em} for $i := 0$ to $n - r$ do 
8: \hspace{2em} $j := i + r$ 
9: \hspace{1em} for $A \in N$ do 
10: \hspace{2em} for $m := i + 1$ to $j - 1$ do 
11: \hspace{3em} for $A \rightarrow BC \in R$ do 
12: \hspace{4em} if $B \in c_{i,m}$ and $C \in c_{m,j}$ then 
13: \hspace{5em} $c_{i,j} := c_{i,j} \cup \{A\}$ \hspace{1em} \hspace{1em} $c_{i,j} := \{A \mid i < m < j, A \rightarrow BC \in R: B \in c_{i,m}, C \in c_{m,j}\}$ 
14: return $(c_{i,j} \mid 0 \leq i < j \leq n)$
From unweighted to weighted CKY parsing

<table>
<thead>
<tr>
<th></th>
<th>Unweighted</th>
<th>Weighted</th>
</tr>
</thead>
<tbody>
<tr>
<td>data structure for spans</td>
<td>set of nonterminals (c_{i,j} \subseteq N \mid 0 \leq i &lt; j \leq n)</td>
<td>map nonterminals to real numbers (c_{i,j} : N \rightarrow \mathbb{R} \mid 0 \leq i &lt; j \leq n)</td>
</tr>
<tr>
<td>update operation</td>
<td>(c_{i,j} := c_{i,j} \cup {A})</td>
<td>(c_{i,j}(A) := \max{c_{i,j}(A), p(r)})</td>
</tr>
</tbody>
</table>

Convention: we will write \(c_{i,j,A}\) rather than \(c_{i,j}(A)\)
The CKY parsing algorithm + weights

**Require:** PCFG \((N, \Sigma, R, S, p)\) in CNF, sentence \(w = w_1 \ldots w_n\) where \(w_1, \ldots, w_n \in \Sigma\)

**Ensure:** family \((c_{i,j,A} \in \mathbb{R} \mid 0 \leq i < j \leq n, A \in N)\) such that

\[
c_{i,j,A} = \max \left( \{p(d) \mid d \in D^A_G(w_{i+1} \ldots w_j)\} \cup \{0\} \right)
\]

1: function \(cky(R, p, w_1 \ldots w_n)\)
2: \((c_{i,j,A} := 0 \mid 0 \leq i < j \leq n, A \in N)\)
3: for \(i := 1\) to \(n\) do
4: \(\quad\) for \(A \rightarrow w_i \in R\) do
5: \(\quad\quad\) \(c_{i-1,i,A} := \max\{c_{i-1,i,A}, p(A \rightarrow w_i)\}\)
6: \(\) for \(r := 2\) to \(n\) do
7: \(\quad\) for \(i := 0\) to \(n - r\) do
8: \(\quad\) \(j := i + r\)
9: \(\quad\) for \(A \in N\) do
10: \(\quad\quad\) for \(m := i + 1\) to \(j - 1\) do
11: \(\quad\quad\) for \(A \rightarrow BC \in R\) do
12: \(\quad\quad\quad\) \(c_{i,j,A} := \max\{c_{i,j,A}, p(A \rightarrow BC) \cdot c_{i,m,B} \cdot c_{m,j,C}\}\)
13: return \((c_{i,j,A} \mid 0 \leq i < j \leq n, A \in N)\)
Adding chain rules

Types of rules accepted by classic CKY:

\[ A \rightarrow w \quad A \rightarrow BC \]

(with \( A, B, C \in N \) and \( w \in \Sigma \))

What we also want:

\[ A \rightarrow B \]

(with \( A, B \in N \))

Solution: two-phase procedure per cell \( c_{i,j} \)

1. perform classic CKY step
2. compute closure under unary rules
The CKY parsing algorithm + weights + chain rules

**Require:** binary PCFG \( (N, \Sigma, R, S, p) \), sentence \( w = w_1 \ldots w_n \) where \( w_1, \ldots, w_n \in \Sigma \)

**Ensure:** family \( (c_{i,j,A} \in \mathbb{R} \mid 0 \leq i < j \leq n, A \in N) \) such that
\[
c_{i,j,A} = \max \{p(d) \mid d \in D^A_G(w_{i+1} \ldots w_j)\} \cup \{0\}
\]

1: \textbf{function} \( \text{cky}(R, \mu, w_1 \ldots w_n) \)
2: \( (c_{i,j,A} := 0 \mid 0 \leq i < j \leq n, A \in N) \)
3: \textbf{for} \( i := 1 \) \textbf{to} \( n \) \textbf{do}
4: \textbf{for} \( A \rightarrow w_i \in R \) \textbf{do}
5: \( c_{i-1,i,A} := p(A \rightarrow w_i) \)
6: \( (c_{i-1,i,A} \mid A \in N) = \text{unary Closure}(R, p, (c_{i-1,i,A} \mid A \in N)) \)
7: \textbf{for} \( r := 2 \) \textbf{to} \( n \) \textbf{do}
8: \textbf{for} \( i := 0 \) \textbf{to} \( n - r \) \textbf{do}
9: \( j := i + r \)
10: \textbf{for} \( A \in N \) \textbf{do}
11: \textbf{for} \( m := i + 1 \) \textbf{to} \( j - 1 \) \textbf{do}
12: \textbf{for} \( A \rightarrow BC \in R \) \textbf{do}
13: \( c_{i,j,A} := \max \{c_{i,j,A}, p(A \rightarrow BC) \cdot c_{i,m,B} \cdot c_{m,j,C}\} \)
14: \( (c_{i,j,A} \mid A \in N) = \text{unary Closure}(R, p, (c_{i,j,A} \mid A \in N)) \)
15: \textbf{return} \( (c_{i,j,A} \mid 0 \leq i < j \leq n, A \in N) \)
The CKY parsing algorithm + weights + chain rules

16: function unary_closure($R$, $p$, $(c_A \mid A \in N)$)
17:    queue := \{(A, c_A) \mid A \in N, c_A \neq 0\}
18:    (c_A := 0 \mid A \in N)
19:    while queue \neq \emptyset do
20:        (B, q) := \arg\max_{(\hat{B}, \hat{q}) \in queue} \hat{q}
21:        queue := queue \setminus \{(B, q)\}
22:        if $c_B < q$ then
23:            $c_B := q$
24:            for $A \rightarrow B \in R$ do
25:                queue := queue \cup \{(A, p(A \rightarrow B) \cdot q)\}
26:    return $(c_A \mid A \in N)$
About backtraces

What we have: best weight $c_{i,j,A}$ for derivations in $D^A_G(w_{i+1}...w_j)$ for each $A \in N$, $0 \leq i < j \leq n$

What we want: the best derivation of $S$ that results in $w = w_1 ... w_n$

During the CKY algorithm:

- store backtraces (indicators how a weight was computed)
- for best derivation: at most one backtrace per span and nonterminal
- update when weight is updated

After the CKY algorithm:

- recursively read trees from backtraces:

  Require: family of backtraces $b$, each otf. ⊥, or $A \rightarrow t$, or $(A \rightarrow B, i, j)$ or $(A \rightarrow BC, i, m, m, j)$

  1: function BEST_TREE($b_{i,j,A} \mid 0 \leq i < j \leq n, A \in N, i, j, A$)
  2: if $b_{i,j,A}$ otf. $A \rightarrow t$ then return $A \rightarrow t$
  3: else if $b_{i,j,A}$ otf. $(A \rightarrow B, i, j)$ then return $(A \rightarrow B)(\text{BEST_TREE}(b, i, j, B))$
  4: else if $b_{i,j,A}$ otf. $(A \rightarrow BC, i, m, m, j)$ then
  5: return $(A \rightarrow BC)(\text{BEST_TREE}(b, i, m, B), \text{BEST_TREE}(b, m, j, C))$
Let’s talk about data structures

- access to grammar rules depends on loops:
  - access by first nonterminal on rhs
  - for some, that be no concern

Map<Nt, (Rule, Wt)>
Set<(Rule, Wt)>
Let’s talk about data structures

- access to grammar rules depends on loops:
  - access by first nonterminal on rhs
  - for some, that be no concern

- weights for each span and nonterminal:
  - usually in a \( (\frac{|w| \cdot (|w| + 1)}{2} \cdot |N|) \)-dimensional vector (dense)
  - or hashmap (sparse)

\[
\text{Vec}<\text{Wt}>
\]

\[
\text{Map}<\text{Nt}, \text{(Rule, Wt)}> \\
\text{Set}<\text{(Rule, Wt)}>
\]

\[
\text{Vec}<\text{Wt}>
\]

\[
\text{Map}<\text{(Int, Int, Nt), Wt}>
\]
Let’s talk about data structures

- access to grammar rules depends on loops:
  - access by first nonterminal on rhs
  - for some, that be no concern
- weights for each span and nonterminal:
  - usually in a \((\frac{|w| \cdot (|w|+1)}{2} \cdot |N|)\)-dimensional vector (dense)
  - or hashmap (sparse)
- storing backtraces:
  - each backtrace: rule and spans and nonterminals (on rule’s right-hand side)
    \(Bt = Bin(Rule, [Int; 4]) + Chain(Rule, [Int; 2]) + Term(Rule)\)
  - one backtrace for each span and nonterminal (dense)
  - or in a hashmap (sparse)
  - or do not store them at all
Outline

1. Overview
2. CKY parsing
3. Deductive parsing
4. Conclusion
Deduction systems [Ned03]

- rule-based system (derivation of items)
- derive consequence ($c$) from antecedents ($a_1, \ldots, a_k$) for some $k \in \mathbb{N}$

$$a_1, \ldots, a_k \quad c$$
Deduction systems [Ned03]

- rule-based system (derivation of *items*)
- derive consequence \((c)\) from antecedents \((a_1, \ldots, a_k)\)
  for some \(k \in \mathbb{N}\)
- compute weight of consequence using weight of antecedents \((w_1, \ldots, w_k)\)

\[
\frac{a_1 : w_1, \ldots, a_k : w_k}{c : f(w_1, \ldots, w_k)}
\]
Deduction systems [Ned03]

- rule-based system (derivation of *items*)
- derive consequence \(c\) from antecedents \((a_1, \ldots, a_k)\) for some \(k \in \mathbb{N}\)
- compute weight of consequence using weight of antecedents \((w_1, \ldots, w_k)\)
- side condition \(b\)

\[
\frac{a_1: w_1, \ldots, a_k: w_k}{c: f(w_1, \ldots, w_k)} b
\]
Deduction system for parsing weighted cfg [Ned03]

- item \((i, A, j)\) for each nonterminal \(A\) spanning \(w_{i+1} \ldots w_j\)
Deduction system for parsing weighted cfg [Ned03]

- item \((i, A, j)\) for each nonterminal \(A\) spanning \(w_{i+1}...w_j\)
- predict initial items \( (i-1, A, i) : p(A \rightarrow w) A \rightarrow w_i \in R \land w = w_1...w_i...w_n \)
item \((i, A, j)\) for each nonterminal \(A\) spanning \(w_{i+1}\ldots w_j\)

predict initial items \(\frac{\begin{array}{c} (i-1,A,i) : p(A \rightarrow w_i) \\ A \rightarrow w_i \in R \wedge w = w_1\ldots w_i\ldots w_n \end{array}}{(i-1,A,i)}\)

combine items \(\frac{\begin{array}{c} (i,0,B_1,i_1) : w_1, (i_1,B_2,i_2) : w_2, \ldots, (i_{k-1},B_k,i_k) : w_k \\ (i_0,A,i_k) : p(A \rightarrow B_1\ldots B_k) \cdot w_1\ldots w_k \\ A \rightarrow B_1\ldots B_k \in R \end{array}}{(i_0,A,i_k)}\)
Deduction system for parsing weighted cfg [Ned03]

- item \((i, A, j)\) for each nonterminal \(A\) spanning \(w_{i+1} \ldots w_j\)
- predict initial items \(\underbrace{(i-1, A, i)}_{(i-1, A, i)} : p(\overbrace{A \rightarrow w_i}^{w_i} \in R \land w = w_1 \ldots w_i \ldots w_n}\)
- combine items \(\underbrace{(i_0, B_1, i_1)}_{(i_0, B_1, i_1)} : w_1, (i_1, B_2, i_2) : w_2, \ldots, (i_{k-1}, B_k, i_k) : w_k}_{(i_0, B_1, i_1) \ldots (i_{k-1}, B_k, i_k)} \rightarrow A \rightarrow B_1 \ldots B_k \in R\)
- goal item: \((0, S, |w|)\)
Deduction system for parsing weighted cfg [Ned03]

- item \((i, A, j)\) for each nonterminal \(A\) spanning \(w_{i+1} \ldots w_j\)
- predict initial items \(\frac{(i-1, A, i)}{p(A \rightarrow w_i)} A \rightarrow w_i \in R \land w = w_1 \ldots w_i \ldots w_n\)
- combine items \(\frac{(i_0, B_1, i_1)}{p(A \rightarrow B_1 \ldots B_k)} A \rightarrow B_1 \ldots B_k \in R\)
- goal item: \((0, S, |w|)\)

- deduction system \(\Rightarrow\) weighted hypergraph
  - edge from antecedents to consequence
  - can be explored with respect weight
  - hyperpaths to goal item correspond to parse trees

\[\begin{array}{c}
\text{NP} \rightarrow \text{She} \\
\downarrow \\
(0, \text{NP}, 1)
\end{array}\quad
\begin{array}{c}
\text{VP} \rightarrow \text{VP PP} \\
\downarrow \\
(1, \text{VP}, 7)
\end{array}\]

\[\begin{array}{c}
\text{S} \rightarrow \text{NP VP} \\
\downarrow \\
(0, \text{S}, 7)
\end{array}\]
Weighted deductive parsing algorithm

Require: weighted binary cfg \((N, \Sigma, R, S, p)\), word \(w_1...w_n\) where \(w_1, ..., w_n \in \Sigma\)

Ensure: family \((c_{i,j,A} \in \mathbb{R} \mid 0 \leq i < j \leq n, A \in N)\) such that
\[
c_{i,j,A} = \max \left( \{p(d) \mid d \in D_G^A(w_i...w_j)\} \cup \{0\} \right)
\]

1: function deduce\((R, p, w_1...w_n)\)
2: \(\text{queue} := \{(i - 1, A, i, p(A \rightarrow w_i)) \mid 1 \leq i \leq n, A \rightarrow w_i \in R\}\)
3: \(c_{i,j,A} := 0 \mid 0 \leq i < j \leq n, A \in N\)
4: while \(\text{queue} \neq \emptyset\) do
5: \((i, A, j, q) := \text{argmax}_{(i, \hat{A}, \hat{j}, \hat{q}) \in \text{queue}} \hat{q}\)
6: \(\text{queue} := \text{queue} \setminus \{(i, A, j, q)\}\)
7: if \(c_{i,j,A} = 0\) then
8: \(c_{i,j,A} := q\)
9: \(\text{queue} := \text{queue} \cup \{(i, A', j', p(A' \rightarrow AC) \cdot q \cdot c_{j,j',C}) \mid j < j' \leq n, A' \rightarrow AC \in R\}\)
10: \(\text{queue} := \text{queue} \cup \{(i', A', j, p(A' \rightarrow BA) \cdot c_{i',i,B} \cdot q) \mid 0 \leq i' < i, A' \rightarrow BA \in R\}\)
11: \(\text{queue} := \text{queue} \cup \{(i, A', j, p(A' \rightarrow A) \cdot q) \mid A' \rightarrow A \in R\}\)
12: return \((c_{i,j,A} \mid 0 \leq i < j \leq n, A \in N)\)
Let’s talk about data structures ... again

- access of grammar from each rhs nonterminal

\[ \text{Map}<Nt, (\text{Rule}, Wt)> \]
Let’s talk about data structures … again

- access of grammar from each rhs nonterminal
- each item may need to store a backtrace

\[
\text{Map}\langle \text{Nt}, (\text{Rule, Wt})\rangle \\
\text{(Int, Nt, Int, Wt, Bt)}
\]
Let’s talk about data structures ... again

- access of grammar from each rhs nonterminal: $\text{Map}<\text{Nt}, \langle\text{Rule}, \text{Wt}\rangle>$
- each item may need to store a backtrace: $\langle\text{Int}, \text{Nt}, \text{Int}, \text{Wt}, \text{Bt}\rangle$
- storing the found items and their weights:
  - access from left: $\text{Map}<\langle\text{Int}, \text{Nt}\rangle, \text{Set}<\langle\text{Int}, \text{Nt}, \text{Int}, \text{Wt}\rangle>$
  - access from right: $\text{Map}<\langle\text{Nt}, \text{Int}\rangle, \text{Set}<\langle\text{Int}, \text{Nt}, \text{Int}, \text{Wt}\rangle>$
Let’s talk about data structures … again

- access of grammar from each rhs nonterminal
- each item may need to store a backtrace
- storing the found items and their weights:
  - access from left
  - access from right
- storing backtraces:
  - store applied rule and antecedent items
    \[ Bt = \text{Bin}(\text{Rule}, [\text{Int}; 4]) + \text{Chain}(\text{Rule}, [\text{Int}; 2]) + \text{Term}(\text{Rule}) \]
  - one backtrace for each item
  - or do not store them at all
Outline

1. Overview
2. CKY parsing
3. Deductive parsing
4. Conclusion
General Comments and Tips

- order of loops in CKY algorithm doesn’t matter that much, *but*¹:
  - may be used to cache-optimize,
  - may lead to other optimizations

¹Bodenstab [Bod09] discusses this in detail.
General Comments and Tips

- order of loops in CKY algorithm doesn’t matter that much, *but*¹:
  - may be used to cache-optimize,
  - may lead to other optimizations
- deductive parsers may not need to expand the whole search space

¹Bodenstab [Bod09] discusses this in detail.
General Comments and Tips

- order of loops in CKY algorithm doesn’t matter that much, *but*¹:
  - may be used to cache-optimize,
  - may lead to other optimizations
- deductive parsers may not need to expand the whole search space
- try to think about efficient access in your data structures
  - don’t search in lists
  - indexed access: maps
  - check if you *really* need sets/maps
  - flat data structures are faster than stacked heap allocations

¹Bodenstab [Bod09] discusses this in detail.
General Comments and Tips

- order of loops in CKY algorithm doesn’t matter that much, but\(^1\):
  - may be used to cache-optimize,
  - may lead to other optimizations
- deductive parsers may not need to expand the whole search space
- try to think about efficient access in your data structures
  - don’t search in lists
  - indexed access: maps
  - check if you really need sets/maps
  - flat data structures are faster than stacked heap allocations
- try not to over-engineer it

\(^1\)Bodenstab [Bod09] discusses this in detail.


