Einführung in die Computerlinguistik

Parsing

Laura Kallmeyer
Heinrich-Heine-Universität Düsseldorf

Sommersemester 2013

Overview

1. Introduction
2. Top-Down Parsing
3. Shift Reduce Parsing
4. Chart Parsing: CYK

Introduction (1)

A parser is a device that accepts a word w and a grammar G as input and that

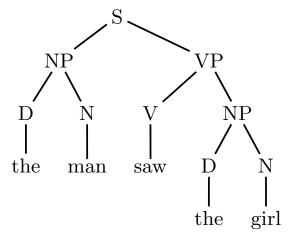
1. decides whether w is in the language generated by the grammar and
2. if so, it provides a syntactic analysis for w or, if w is ambiguous, a set of analyses, oftentimes represented in a compact way as a derivation forest.

A device that does only the first part of the task is called a recognizer.

Introduction (2)

Example for parsing:
Input: “the man saw the girl”.

Output:



Input: “the man saw saw the girl”. Output: no.

Top-Down Parsing (1)

CFG parser that is

- a top-down parser: we start with S and subsequently replace lefthand sides of productions with righthand sides.
- a directional parser: the expanding of non-terminals (with appropriate righthand sides) is ordered; we start with the leftmost non-terminal and go through the righthand sides of productions from left to right.

  In particular: we determine the start position of the span of the ith symbol in a rhs only after having processed the $i-1$ preceding symbols.
- a LL-parser: we process the input from left to right while constructing a leftmost derivation.

First proposed by Sheila Greibach (for CFGs in GNF).

Top-Down Parsing (2)

Assume a CFG without left recursion $A \stackrel{+}{\Rightarrow} A\alpha$.

The parser goes through different pairs of remaining input and sentential form (a stack), starting with w and the start symbol S.

In each step, we

- either scan the next input symbol, provided it corresponds to the top of the sentential form
- or we non-deterministically predict a production that expands the top of the sentential form, provided this is a non-terminal. In this case we replace it with the rhs of a production.

Success, if we end with empty remaining input and empty sentential form.

Top-Down Parsing (3)

Example: $S \rightarrow aSb \,|\, c$, input $aacbb$.

1.	$aacbb$	S	initial
2.	$aacbb$	aSb	predict from 1.
4.	$aacbb$	c	predict from 1.
5.	$acbb$	Sb	scan from 2.
6.	$acbb$	$aSbb$	predict from 5.
7.	$acbb$	cb	predict from 5.
8.	cbb	Sbb	scan from 6.
9.	cbb	$aSbbb$	predict from 8.
10.	cbb	cbb	predict from 8.
11.	bb	bb	scan from 10.
12.	b	b	scan from 11.
13.	ε	ε	scan from 12.

Top-Down Parsing (4)

Function `top-down` with arguments

- w: remaining input;
- α: remaining sentential form (a stack).

`top-down(`w`,`α`)` iff $\alpha \stackrel{*}{\Rightarrow} w$ (for $\alpha \in (N \cup T)^*, w \in T^*$)

Initial call:

`top-down(`w`,`S`)`

Top-Down Parsing (5)

function top-down(w,α):

  out = false;

  if $w = \alpha = \epsilon$, then out = true;

  else if $w = aw'$ and $\alpha = a\alpha'$,

    then out = top-down(w',α')   scan

    else if $\alpha = X\alpha'$ with $X \in N$,

      then for all $X \rightarrow X_1 \dots X_k$:

        if top-down(w, $X_1 \dots X_k\alpha'$)   predict

          then out = true;

  return out

Top-Down Parsing (6)

How to turn the recognizer into a parser:

Add an analysis stack to the parser that allows you to construct the parse tree.

Assume that for each $A \in N$, the righthand sides of A-productions are numbered (have indices).

Whenever

- a production is applied (prediction step), the lefthand side is pushed on the analysis stack together with the index of the righthand side;
- a terminal a is scanned, a is pushed on the analysis stack.

  (This is needed for backtracking in a depth-first strategy.)

Top-Down Parsing (7)

function top-down(w,α,Γ):

  out = false;

  if $w = \alpha = \epsilon$,

    then output Γ; out = true;

  else if $w = aw'$ and $\alpha = a\alpha'$,

    then out = top-down(w',α', $a\Gamma$)

    else if $\alpha = X\alpha'$ with $X \in N$,

      then for all $X \rightarrow X_1 \dots X_k$ with rhs-index i:

        if top-down(w, $X_1 \dots X_k\alpha'$, $\langle X, i\rangle\Gamma$)

          then out = true;

  return out

Shift-Reduce Parsing (1)

CFG parser that is

- a bottom-up parser: we start with the terminals and subsequently replace righthand sides of productions with lefthand sides.
- a directional parser: the replacing of righthand sides with lefthand sides is ordered corresponding to a rightmost derivation.
- a LR-parser: we process the input from left to right while constructing a rightmost derivation.
- a Shift-reduce-parser: the two operations of the parser are shift and reduce.

Shift-Reduce Parsing (2)

The parser consists of

- a stack (initially empty) $\Gamma \in (N \cup T)^*$
- the remaining input (initially w).

Idea:

- w is shifted on the stack while, whenever the top of the stack is the rhs of a production in reverse order, this is replaced with the lhs.
- Success if $\Gamma = S$ and remaining input ϵ.

Shift-Reduce Parsing (3)

For convenience we write the stack with its top on the right.

Example: $S \to ABC, A \to a \mid Aa, B \to b \mid bB, C \to c$

$w = aabbbc$.

	$aabbbc$	
a	$abbbc$	shift
A	$abbbc$	reduce, $A \to a$
Aa	$bbbc$	shift
A	$bbbc$	reduce, $A \to Aa$
Ab	bbc	shift
AB	bbc	reduce, $B \to b$
ABb	bc	shift

Shift-Reduce Parsing (4)

ABb	bc	
AB	bc	reduce, $B \to Bb$
ABb	c	shift
AB	c	reduce, $B \to Bb$
ABc		shift
ABC		reduce, $C \to c$
S		reduce, $S \to ABC$

If we apply the productions in reverse order we obtain a rightmost derivation:

$S \Rightarrow ABC \Rightarrow ABc \Rightarrow ABbc \Rightarrow ABbbc \Rightarrow Abbbc \Rightarrow Aabbbc \Rightarrow aabbbc$

Shift-Reduce Parsing (5)

Assume a grammar without ϵ-productions and without loops.

```
function bottom-up(w,Γ):
    if w = ϵ and Γ = S then true
    else reduce(w,Γ) or shift(w,Γ)

function shift(w,Γ):
    out = false
    if w = aw′ and a ∈ T
        then out = bottom-up(w′,Γa)
    return out
```

Shift-Reduce Parsing (6)

function reduce(w,Γ):

  out = false;

  for every $A \to \alpha \in P$:

    if $\Gamma = \Gamma'\alpha$ and bottom-up(w,$\Gamma' A$)

    then out = true;

  return out

Initial call: bottom-up(w, ϵ)

Kallmeyer CL-Einführung

CYK (1)

The CYK parser is

- a bottom-up parser: we start with the terminals in the input string and subsequently compute recognized parse trees by going from already recognized rhs of productions to the non-terminal on the lefthand side.
- a non-directional parser: the order of the completing of subtrees is not necessarily from left to right.
- a chart parser: we store every intermediate result in a chart and can reuse it in different contexts. This avoids computing the same subtree several times. Particularly useful for ambiguous grammars such as natural language grammars.

Independently proposed by Cocke, Kasami and Younger in the 60s.

Kallmeyer CL-Einführung

CYK (2)

A CFG is in Chomsky Normal Form iff all productions are either of the form $A \to a$ or $A \to B\ C$.

If the grammar has this form,

- we need to check only for two categories B, C, in order to construct an A with $A \to B\ C$.
- we can be sure that the spans always become longer when applying productions $A \to B\ C$. I.e., if l_1 and l_2 are the lengthes of B and C, then the length of the resulting A is $l_1 + l_2 > max(l_1, l_2)$.

Every CFG can be transformed into an equivalent CFG in CNF.

Kallmeyer CL-Einführung

CYK (3)

The chart C is an $n \times n$-array. The first index is the index of the first terminal in the span and the second gives the length of a span.

$A \in C_{i,l}$ indicates that we have found an A with a span starting at index i and having length l.

Algorithm:

$C_{i,1} = \{A \mid A \to w_i \in P\}$ scan

for all $l \in [1..n]$:

 for all $i \in [1..n]$:

  for every $A \to B\ C$:

   if there is a $l_1 \in [1..l-1]$ such that

    $B \in C_{i,l_1}$ and $C \in C_{i+l_1,l-l_1}$,

   then $C_{i,l} = C_{i,l} \cup \{A\}$ complete

CYK (4)

Example: $S \to C_aC_b \mid C_aS_B, S_B \to SC_b, C_a \to a, C_b \to b$. (From $S \to aSb \mid ab$ with transformation into CNF.)

$w = aaabbb$.

l							
6	S						
5		S_B					
4		S					
3			S_B				
2			S				
1	C_a	C_a	C_a	C_b	C_b	C_b	
	1	2	3	4	5	6	i
	a	a	a	b	b	b	