Parsing

Top-Down Parsing

Laura Kallmeyer

Heinrich-Heine-Universität Düsseldorf

Winter 2017/18

Table of contents

Introduction

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).
Grune and Jacobs (2008)

The recognizer (1)

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

Function `top-down` with arguments

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

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

Initial call:
`top-down`(w, S)

The recognizer (2)

Top-down recognizer

```
def top-down(w, α):
    out = false
    if w = α = ε:
        out = true
    elif w = aw' and α = aα':
        out = top-down(w', α')                    scan
    elif α = Xα' with X ∈ N:
        for X → X_1 ... X_k in P:
            if top-down(w, X_1 ... X_k α'):       predict
                out = true
    return out
```

The recognizer (3)

This is exactly what the following PDA-construction for a CFG does:

- start with stack Z_0 and q_0.
- $\delta(q_0, \epsilon, Z_0) = \{\langle q_1, SZ_0\rangle\}$
- $\langle q_1, \alpha\rangle \in \delta(q_1, \epsilon, A)$ for all $A \to \alpha$
- $\langle q_1, \epsilon\rangle \in \delta(q_1, a, a)$ for all $a \in T$.
- $\delta(q_1, \epsilon, Z_0) = \{\langle q_f, \epsilon\rangle\}$

(LL-PDA construction in JFLAP)

The recognizer (4)

Example: Top-down recognizer

$G = \langle N, T, P, S \rangle,\ N = \{A, B\},\ T = \{a, b, c\}$

$P = \{S \to ASB \,|\, AASB \,|\, c, A \to a, B \to b\}$

Input $w = aacb$. Calls of `top-down` (order is depth-first)

	stack α	w	
1.	S	aacb	
2.	ASB	aacb	pred(1)
3.	aSB	aacb	pred(2)
4.	SB	acb	scan(3)
5.	ASBB	acb	pred(4)
6.	aSBB	acb	pred(5)
7.	SBB	cb	scan(6)
4 unseccessful predicts			
8.	cBB	cb	pred(7)
scan – predict – scan			
9.	b	–	pred
10.	AASB	aacb	pred(1)
11.	aASB	aacb	pred(10)
scan – predict			
12.	SB	cb	scan
4 unseccessful predicts			
13.	cB	cb	pred(12)
14.	B	b	scan(13)
15.	b	b	pred(14)
16.	–	–	scan(15)

The parser (1)

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.)

The parser (2)

Top-down parser

```
def top-down(w, α, Γ):
    out = false
    if w = α = ϵ:
        output Γ
        out = true
    elif w = aw′ and α = aα′:
        out = top-down(w′, α′, aΓ)
    elif α = Xα′ with X ∈ N:
        for X → X_1 ... X_k in P with rhs-index i:
            if top-down(w, X_1 ... X_k α′, ⟨X, i⟩Γ):
                out = true
    return out
```

The recognizer (3)

Example: Top-down parser

$G = \langle N, T, P, S \rangle,\ N = \{A, B\},\ T = \{a, b, c\}$

$P = \{S \to ASB \,|\, AASB \,|\, c, A \to a, B \to b\}$

Input $w = aacb$. Consider only the successful predicts and scans (X_i is a notation for $\langle X, i \rangle$):

stack α	w	analysis stack
S	aacb	
AASB	aacb	S_2
aASB	aacb	A_1S_2
ASB	acb	aA_1S_2
aSB	acb	$A_1aA_1S_2$
SB	cb	$aA_1aA_1S_2$
cB	cb	$S_3aA_1aA_1S_2$
B	b	$cS_3aA_1aA_1S_2$
b	b	$B_1cS_3aA_1aA_1S_2$
–	–	$bB_1cS_3aA_1aA_1S_2$

the analysis stack gives a leftmost derivation in reverse order.

Leftmost derivation: $S_2A_1A_1S_3B_1$

The parser (4)

Problematic grammars for this parser: CFGs that allow for left-recursions. Solutions:

- Eliminate the left-recursion.
  Drawback: derivation trees change considerably.
- Make sure, grammar does not contain ϵ-productions or loops.
  Then do an additional check (when predicting):

```
. . .
   then for all X → X_1 ... X_k:
      if |w| ≥ |X_1 ... X_k α'|
         and top-down(w, X_1 ... X_k α')
         then out := true;
```

This check is useful even for grammars that are not left-recursive.

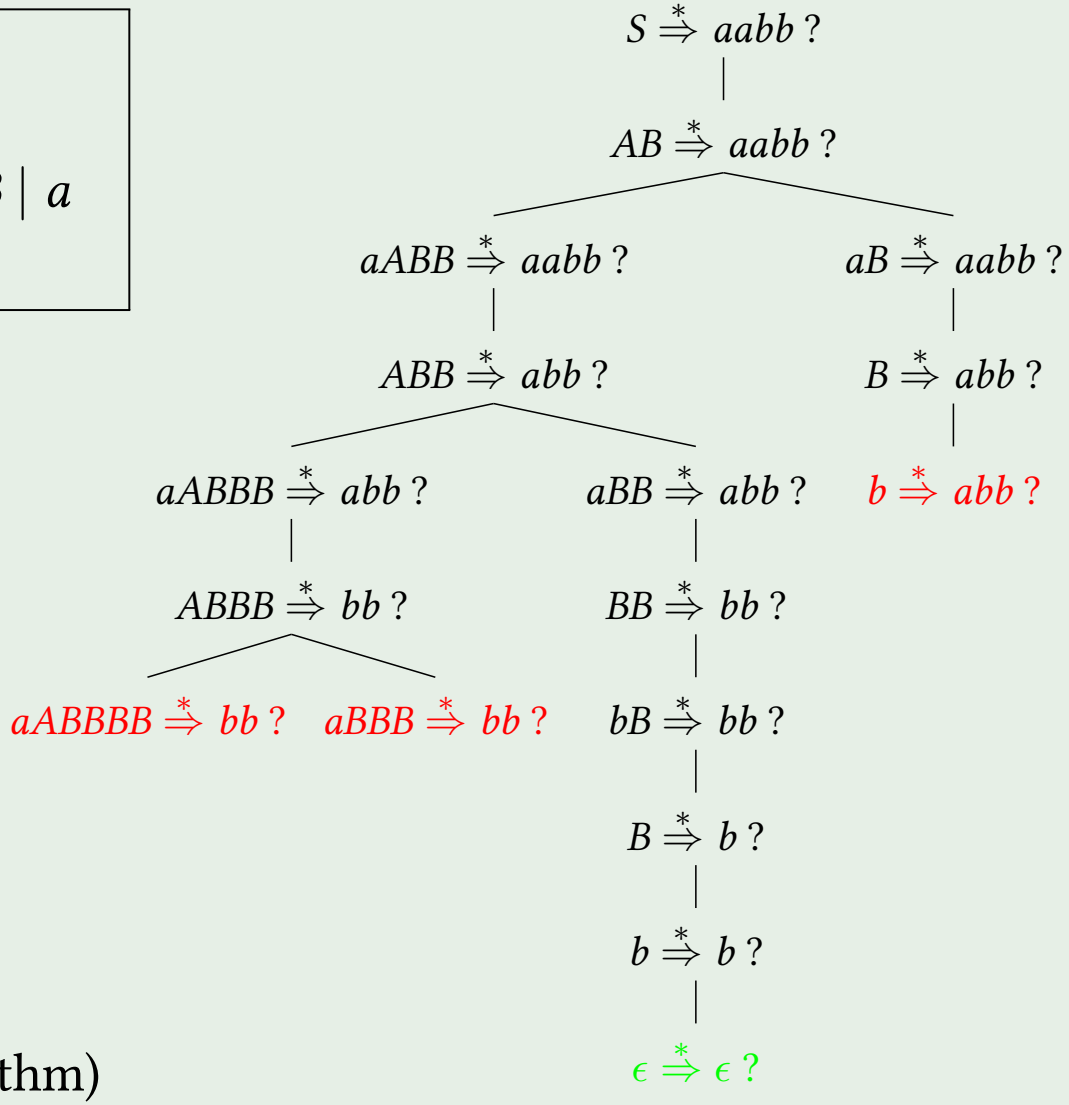
An example (1)

Grammar

$S \to AB$

$A \to aAB \mid a$

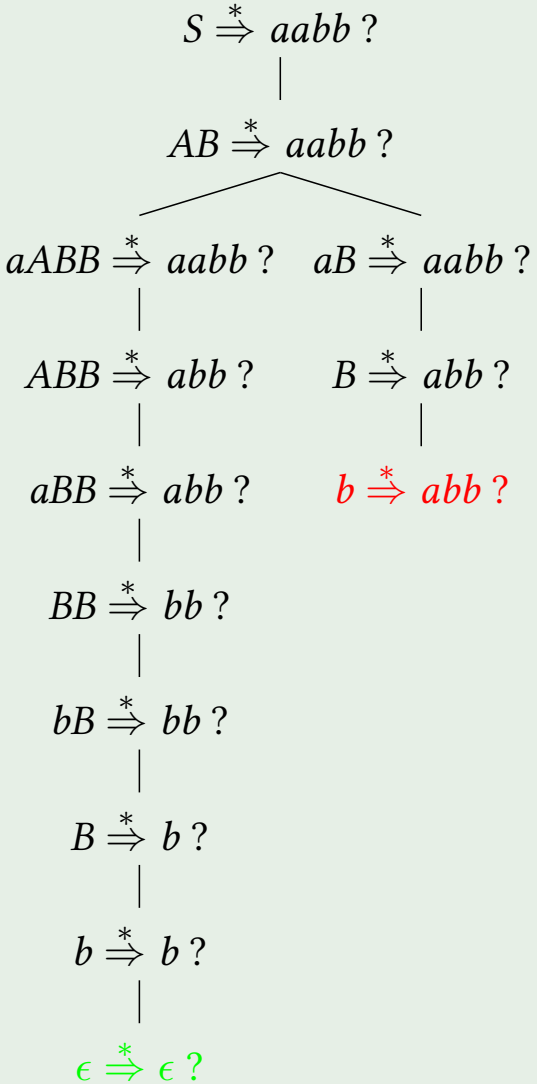
$B \to b$



(basic algorithm)

An example (2)

(check that word length $\geq$ length of sentential form)



Control structures (1)

In general, directional top-down parsing is non-deterministic because of multiple righthand sides for single non-terminals.
Two different control strategies: You can

- either proceed depth-first (proceed the righthand sides one after the other, each time pursuing the possible derivation tree up to the moment where we either find a parse tree or fail)
  If we fail, we have to go back and try the next possibility (back-tracking). For this, we have to reverse the operations made on the stacks.
- or proceed breadth-first (try all righthand sides in parallel)
  Usually, all possible predicts are done before scanning the next input symbol.

These are different control structures, they are not part of the general top-down parsing algorithm.

Control structures (2)

Advantages and disadvantages:
Breadth-first:

- Needs a lot of memory.

Depth-first (backtracking):

- Does not need much memory.
- If all parse trees are searched for and the grammar is known to be ambiguous, more time-consuming than breadth-first.

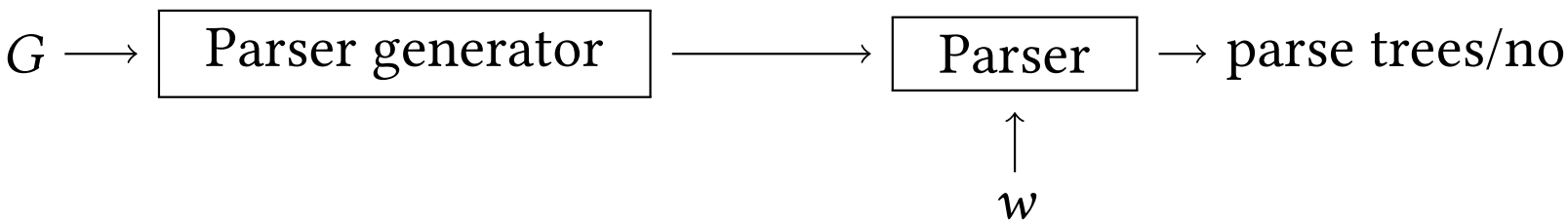
$\Rightarrow$ No perfect solution. The best option depends on the grammar, the input, the task (exhaustive parsing or not), the programming language used…

Parser generators (1)

In general, we can

- either implement a general CFG parser (perhaps for a restricted type of CFG) that takes G and w as input

- or generate a specific parser for a given grammar. The new parser receives only w as input.



Parser generators (2)

Parser generators for top-down (LL) parsers often use a technique called recursive descent:

- for each non-terminal X, a procedure is generated that tries all rhs of X-productions with calls for all non-terminals it encounters (one procedure $\simeq$ one production)
- procedures can call each other, in particular, they can call (directly or via other intermediate calls) itself again (recursive)

Some recursive descent parser generators:

- JavaCC, Java Compiler Compiler:
  `https://javacc.dev.java.net/`
- ANTLR, ANother Tool for Language Recognition (generates C++, Java, Python, C#):
  `http://www.antlr.org/`

Conclusion

Important features of directional top-down parsing:

- LL-parsing: input processed from left to right, constructs a leftmost derivation;
- parsing steps prediction and scan;
- non-deterministic in general;
- different control structures (breadth-first, depth-first);
- does not work for left-recursive CFGs;
- parser generation with recursive descent.

Grune, D. and Jacobs, C. (2008). *Parsing Techniques. A Practical Guide*. Monographs in Computer Science. Springer. Second Edition.