# Lecture 13

## Top-Down Parsing

*CS 241: Foundations of Sequential Programs*
Winter 2018

Troy Vasiga et al
University of Waterloo

# Parsing

Given a grammar $G$ and a word $w$, find a derivation for $w$.
Two strategies:

1. Top-down: find a non-terminal and replace it with a right-hand side of a rule.

2. Bottom-up: replace a right-hand side with a non-terminal.

In both of the above strategies, we have to make the correct decision at each step.

# Parsing Algorithm

- There is a backtracking algorithm for parsing in any CFG
  - try each rule in turn
  - if we can move "forward", do so
  - if we cannot move "forward", go back a step and try the "next" rule
  - stop when we find the derivation

- Backtracking is not practical.

- We will look at two (linear-time) algorithms.

# Stack-based Parsing

For top-down parsing, we use a stack to remember information about our derivations and/or processed input.

# Augmenting Grammars

Empty words and empty stacks can cause hassles.

We augment our grammars by adding "beginning" and "ending" characters.

Example:

$$
\begin{aligned}
2.\ S &\rightarrow AyB \\
3.\ A &\rightarrow ab \\
4.\ A &\rightarrow cd \\
5.\ B &\rightarrow z \\
6.\ B &\rightarrow wz
\end{aligned}
$$

# A simple parse

# Top-down parsing with a stack

Invariant:

$$\text{derivation} = \text{input already read} + \text{stack}$$

# Stack Example

| Derivation | Input read | Input to be read | Stack | Actions |
|---|---|---|---|---|
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

## Observations:

How do we apply these rules? What does "expand" mean?

How do we know we are done?

How to know which rule to use?

# LL(1) Parsing

We need: Predict(A, x) $=$ A $\rightarrow \alpha$ so long as

- A is on top of the stack, and
- x is the first symbol of input to be read

Definition of an LL(1) grammar:

Meaning of:

- L
- L
- 1

# Constructing a Predictor Table

CFG:

1. S' $\rightarrow \vdash$ S $\dashv$
2. S $\rightarrow$ AyB
3. A $\rightarrow$ ab
4. A $\rightarrow$ cd
5. B $\rightarrow$ z
6. B $\rightarrow$ wz

| | a | b | c | d | y | w | z | $\vdash$ | $\dashv$ |
|---|---|---|---|---|---|---|---|---|---|
| S' | | | | | | | | | |
| S | | | | | | | | | |
| A | | | | | | | | | |
| B | | | | | | | | | |

# Constructing a Predictor Table (with $\epsilon$)

CFG:

1. S' $\rightarrow$ $\vdash$ S $\dashv$
2. S $\rightarrow$ AyB
3. A $\rightarrow$ ab
4. A $\rightarrow$ cd
5. B $\rightarrow$ z
6. B $\rightarrow$ wz
7. B $\rightarrow$ $\epsilon$

| | a | b | c | d | y | w | z | $\vdash$ | $\dashv$ |
|---|---|---|---|---|---|---|---|---|---|
| S' | | | | | | | | | |
| S | | | | | | | | | |
| A | | | | | | | | | |
| B | | | | | | | | | |

## Algorithm to construct predicator table

Below, $\alpha, \beta \in (N \cup T)^*$, $x, y \in T$, $A \in N$

Empty($\alpha$) $=$ true if $\alpha \Rightarrow^* \epsilon$

First($\alpha$) $= \{x \mid \alpha \Rightarrow^* x\beta\}$

Follow(A) $= \{y \mid S' \Rightarrow^* \alpha A y \beta\}$

Predict($A$, $x$) $=$

# LL(1) Parsing algorithm

```
Input: w
push S'
for each x ∈ w
        while (top of stack is some A ∈ N) {
                pop A
                if Predict(A, x) = {A → α}
                        push α
                else
                        reject
        }
        pop c
        if c ≠ x reject
end for
accept w
```

# Non LL(1) Grammars

# Converting non-LL(1) grammars to LL(1) grammars

Factoring

# A non LL(1) language

$$L = \{a^n b^m | n \geq m \geq 0\}$$

Grammar (ambiguous)

Grammar (unambiguous)