### CS536 Lecture 11

Tuesday 24 February 2015

## Reminders:

- Reading
- HW5 assigned, Due March 2.

### Last class:

- Parsing
  - CYK

# Today:

• Top-Down Parsing

## Parsing: Approaches (Review)

Top-down ("Goal-Driven"):

Bottom-Up ("Data-Driven"):

## **CYK Algorithm:**

- Decides whether a string can be generated by a grammar
- Bottom-Up
- Takes worst-case O(n³|G|) time
- Only works on a specific grammar form

#### **Chomsky Normal Form:**

All rules must be in one of two forms:

$$X \rightarrow \mathbf{t}$$
 $X \rightarrow AB$ 

The empty string may only appear on the RHS of the start symbol.

Why CNF?

## Transforming a grammar to CNF

Eliminate useless rules (reduce bloat):

$$S \rightarrow X \mid Y$$
  
 $X \rightarrow ()$ 

$$Y \rightarrow (YY)$$

$$Z \rightarrow (X)$$

Eliminate epsilon-rules:

$$A \rightarrow \in | N$$

$$B \to ID(A)$$

$$C \rightarrow A$$
 SEMICOLON A

Eliminate unit productions:

$$A \rightarrow N$$

$$B\to \mathrm{ID}\;(\,A\,)$$

Fix RHS terminals:

$$N \to \mathrm{ID}$$
 ,  $N$ 

Fix RHS nonterminals:

$$X \rightarrow ABC$$

Exercise: Convert the following grammar to CNF

$$F \rightarrow ID (A)$$

$$A \rightarrow \in \mid N$$

$$N \to \mathbf{ID}^{'} | \mathbf{ID}, N$$

### **Parsing Algorithms**

CYK is nice and works on all (CF) grammars ... but expensive!

We can do better for subsets of grammars. Linear time, even.

- LL (1)
- LALR (1)

Benefits of restricting grammars:

#### LL (1):

- Left to right (first L)
- Leftmost Derivation only (second L)
- Token lookahead of 1 (1)
- Top-down, "predictive" parser

## LALR (1):

- Special lookahead procedure (LA)
- Left to right (second L)
- Rightmost Derivation (R)
- Token lookahead of 1 (1)
- Bottom-Up parsing

LALR(1) is strictly more powerful and also more difficult to understand.

Tradeoff: simplicity vs. power.

# **Top-Down Parsers**

Start at the start nonterminal.

"Predict" what productions to use to reach terminal string.

General diagram:

### Algorithm

```
stack.push(eof)
stack.push(Start non-term)
current token = scanner.getToken()
Repeat
    if stack.top is a terminal y
        match y with current token
        pop y from the stack
        current token = scanner.next token()
    if stack.top is a <u>nonterminal</u> X
        get table[X, current token]
        pop X from the stack
        push production's RHS (each symbol from R to L)
Until one of the following:
    stack is empty
    stack.top is a terminal not matching current token
    stack.top is a non-term and parse table entry is empty
```

# Example

Grammar:  $S \rightarrow (S) \mid \{S\} \mid \epsilon$ 

Parse Table:

	(	)	{	}	eof
S					

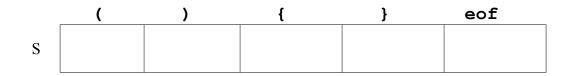
Example input: ( { } ) EOF

Example bad input: ( ( } EOF

## LL(1) grammars

Example grammar:  $S \rightarrow (S) \mid \{S\} \mid (S) \mid E$ 

Parse Table:



How do we know then whether a grammar is LL(1)?

Also, how do we build the selector table?

Key: If each entry in the selector table has 1 production, the grammar is LL(1).

A grammar is *not* LL(1) if:

- It is left-recursive
- If it is not left-factored

## **Left-Recursion**

Recall: A grammar is left-recursive (in $X$ ) if $X \Rightarrow + X\alpha$ . It is immediately left-recursive if $X \to X\alpha$ .
We can remove left-recursion without changing the language recognized.
Consider $A \rightarrow A\alpha \mid \beta$ (why must the second production exist?) where $\beta$ doesn't begin with $A$ .
Transform it to:
More generally:
Pictorially:
This introduces problems with associativity though

#### **Left Factored Grammar**

If a grammar	is not	left-factored,	it cannot be	LL(1).
--------------	--------	----------------	--------------	--------

Definition (of **not** being left-factored): A nonterminal has two or more productions whose RHS has a common prefix.

Example:  $E \rightarrow (E)$ 

This grammar is *not* left-factored.

Fixing this:

$$A \to \alpha \beta_1 \mid \alpha \beta_2$$

to

More generally:

Pictorially:

# **Combined Example**

Grammar:  $E \rightarrow (E) \mid E \mid E \mid (D)$ 

First, remove left-recursion:

Then, left-factoring

# **Another Example**

#### Grammar: