CS536 Lecture 12

Thursday 26 February 2015

Last class:

• Top-Down Parsing

Today:

• LL(1) grammars

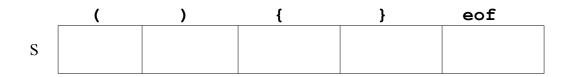
Algorithm

```
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
```

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 <i>immediately</i> 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 β doesn't begin with A .
Transform it to:
More generally:
Pictorially:
This introduces problems with associativity though

Left Factored Grammar

If a gram	mar 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:

Building a Parse (Selector) Table

How can we be sure that a particular production $A \rightarrow \alpha$ is the right one to apply?

Terminals that could possibly start α : FIRST set

Terminal that could come after α : FOLLOW set

FIRST(α): The set of terminals that can begin the strings derivable from α . Includes ϵ if α can derive ϵ .

Formally:

 $FIRST(\alpha) = \{ t \mid t \text{ is terminal and } \alpha \Rightarrow + t\beta \} \cup \{ \in \} \text{ (if } \alpha \text{ can derive } \in \text{)}.$