#### CS412/CS413

### Introduction to Compilers Tim Teitelbaum

Lecture 7: LL parsing and AST construction February 5, 2007

CS 412/413 Spring 2007

Introduction to Compilers

### LL(1) Parsing

- · Last time:
  - how to build a parsing table for an LL(1) grammar (use FIRST/FOLLOW sets)
  - how to construct a recursive-descent parser from the parsing table
- Grammars may not be LL(1)
  - Use left factoring when grammar has multiple productions starting with the same symbol.
  - Other problematic cases?

CS 412/413 Spring 2007

Introduction to Compilers

### if-then-else

• How to write a grammar for if stmts?

 $S \rightarrow if (E) S$ 

 $S \rightarrow if (E) S else S$ 

 $S \rightarrow other$ 

Is this grammar ok?

CS 412/413 Spring 2007

Introduction to Compilers

## No—Ambiguous!

· How to parse?

if  $(E_1)$  if  $(E_2)$   $S_1$  else  $S_2$ 

 $S \rightarrow if (E) S else S$  $S \rightarrow other$ 

 $S \rightarrow if (E) S$ 

 $S \rightarrow if (E) S$  $\rightarrow$  if (E) if (E) S else S

 $S \rightarrow if (E) S else S$  $\rightarrow$  if (E) if (E) S else S

Which "if" is the "else" attached to?

CS 412/413 Spring 2007

Introduction to Compilers

#### Grammar for Closest-if Rule

- Want to rule out if (E) if (E) S else S
- Impose that unmatched "if" statements occur only on the "else" clauses

→ matched | unmatched statement matched → if (E) \_\_\_\_\_ else \_\_\_ other unmatched  $\rightarrow$  if (E) \_\_ if (E) \_\_\_\_\_ else \_

CS 412/413 Spring 2007

Introduction to Compilers

#### Grammar for Closest-if Rule

- Want to rule out if (E) if (E) S else S
- Impose that unmatched "if" statements occur only on the "else" clauses

→ matched | unmatched statement → if (E) matched else matched matched other

unmatched  $\rightarrow$  if (E) \_ | if (E) \_\_\_\_

CS 412/413 Spring 2007 Introduction to Compilers

\_\_\_ else \_

#### Grammar for Closest-if Rule

- Want to rule out if (E) if (E) S else S
- Impose that unmatched "if" statements occur only on the "else" clauses

CS 412/413 Spring 2007

Introduction to Compilers

#### Grammar for Closest-if Rule

- Want to rule out if (E) if (E) S else S
- Impose that unmatched "if" statements occur only on the "else" clauses

```
\begin{array}{lll} \text{statement} & \to & \text{matched} & | & \text{unmatched} \\ & \text{matched} & \to & \text{if (E) matched else matched} \end{array}
```

other

unmatched  $\rightarrow$  if (E) statement

| if (E) matched else unmatched

CS 412/413 Spring 2007

Introduction to Compilers

## LL(1) if-then-else?

```
\begin{array}{lll} \text{statement} & \rightarrow \text{if (E) matched optional-tail } \mid \text{other} \\ \text{matched} & \rightarrow \text{if (E) matched else matched } \mid \text{other} \\ \text{optional-tail} & \rightarrow \text{else tail } \mid \epsilon \\ \text{tail} & \rightarrow \text{if (E) tail } \mid \text{other} \\ \end{array}
```

CS 412/413 Spring 2007

Introduction to Compilers

### LL(1) if-then-else?

 $\begin{array}{ll} \text{statement} & \rightarrow \text{if (E) matched optional-tail } | \text{ other} \\ & \rightarrow \text{if (E) matched else matched } | \text{ other} \\ & \text{optional-tail} & \rightarrow \text{else tail } | \epsilon \\ & \text{tail} & \rightarrow \text{if (E) tail } | \text{ other} \\ \end{array}$ 

CS 412/413 Spring 2007

Introduction to Compilers

#### **Left-Recursive Grammars**

• Left-recursive grammars are not LL(1)!

$$S \to S \alpha \\ S \to \beta$$

- $FIRST(\beta) \subseteq FIRST(S\alpha)$
- Both productions will appear in the predictive table, at row S in all the columns corresponding to symbols in  $\mathsf{FIRST}(\beta)$

CS 412/413 Spring 2007

Introduction to Compilers

11

#### **Eliminate Left Recursion**

 Method for left-recursion elimination: Replace

$$\begin{split} \mathsf{A} &\to \mathsf{A} \; \alpha_1 \; | \; \ldots \; | \; \mathsf{A} \; \alpha_m \\ \mathsf{A} &\to \beta_1 \; | \; \ldots \; | \; \beta_n \end{split}$$
 with 
$$\begin{split} \mathsf{A} &\to \beta_1 \; \mathsf{A'} \; | \; \ldots \; | \; \beta_n \; \mathsf{A'} \\ \mathsf{A'} &\to \alpha_1 \; \mathsf{A'} \; | \; \ldots \; | \; \alpha_m \; \mathsf{A'} \; | \; \epsilon \end{split}$$

• (See the complete algorithm in the Dragon Book)

CS 412/413 Spring 2007

Introduction to Compilers

12

2

#### Creating an LL(1) Grammar

```
    Start with a left-recursive grammar:
        S → S+E
        S → E
        and apply left-recursion elimination algorithm:
        S → ES'
        S' → +E S' | ε
    Start with a right-recursive grammar:
        S → E+S
        S → E
        and apply left-factoring to eliminate common prefixes:
        S → E S'
        S' → + S | ε
```

Introduction to Compilers

13

### **Top-Down Parsing**

- · Now we know:
  - how to build a parsing table for an LL(1) grammar (use FIRST/FOLLOW sets)
  - how to construct a recursive-descent parser from the parsing table
- Can we use recursive descent to build an abstract syntax tree too?

CS 412/413 Spring 2007

Introduction to Compilers

14

16

## Creating the AST

CS 412/413 Spring 2007

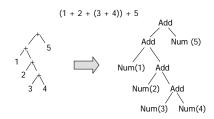
```
abstract class Expr { }

class Add extends Expr {
   Expr left, right;
   Add(Expr L, Expr R) {
      left = L; right = R;
   }
}

class Num extends Expr {
   int value;
   Num (int v) { value = v; }
}

cs 412/413 Spring 2007 Introduction to Compilers 15
```

### **AST Representation**



How can we generate this structure during recursive-descent parsing?

CS 412/413 Spring 2007

Introduction to Compilers

## Creating the AST

- Just add code to each parsing routine to create the appropriate nodes!
- Works because parse tree and call tree have same shape
- parse\_S, parse\_S', parse\_E all return an Expr:

```
void parse_E()
void parse_S()
void parse_S'()

Expr parse_E()
Expr parse_S'()

Expr parse_S'()
```

CS 412/413 Spring 2007

Introduction to Compilers

17

# AST Creation: parse\_E

```
Expr parse_E() {
    switch(token) {
        case num: // E → num
            Expr result = Num (token.value);
            token = input.read(); return result;
        case '(': // E → (S)
            token = input.read();
            Expr result = parse_S();
            if (token != ')') throw new ParseError();
            token = input.read(); return result;
            default: throw new ParseError();
    }
}

CS 412/413 Spring 2007 Introduction to Compilers 18
```

### 

```
Top-down Parsing EBNF

• Recursive-descent code can directly implement the EBNF grammar:

S → E(+E)*

void parse_S () { // parses sequence of E+E+E ... parse_E (); while (true) {
 switch (token) {
 case '+': token = input.read(); parse_E(); break; case ')': case EOF: return; default: throw new ParseError(); }
}

cs 412/413 Spring 2007 Introduction to Compilers 21
```

```
Top-Down Parsing Summary

Language grammar

Left-recursion elimination
Left-factoring

LL(1) grammar

predictive parsing table

recursive-descent parser

parser with AST generation

CS 412/413 Spring 2007 Introduction to Compilers 23
```

```
Exercises

• Which of the following are LL(1)?

(1)

A \rightarrow aACd \mid b
C \rightarrow c \mid \epsilon
(2)

A \rightarrow aACd \mid b
C \rightarrow A \mid \epsilon
(3)

A \rightarrow aAC \mid b
C \rightarrow c \mid \epsilon
cs 412/413 Spring 2007 Introduction to Compilers 24
```