

Bottom-Up Parsing

Chapter 4, Part III
Bottom-Up Parsing

2/16/2008 CSS810 Spring 2008 1

Top-down v.s. Bottom-up

- Top-down parsers
 - Start at the root of the parse tree and grow toward leaves
 - Pick a production & try to match the input
- Bottom-up parsers
 - Start at the leaves and grow toward root
 - As input is consumed, encode possibilities in an internal state

2/16/2008 CSS810 Spring 2008 2

Bottom-Up Parsing

- A more powerful parsing technology
- LR grammars – more expressive than LL
 - Construct right-most derivation of program
 - Left-recursive grammars, virtually all programming languages are left-recursive
 - Easier to express syntax
- Shift-reduce parsers
 - Parsers for LR grammars
 - Automatic parser generators (yacc, bison)

2/16/2008 CSS810 Spring 2008 3

Bottom-Up Parsing

- Right-most derivation – Backward
 - Start with the tokens
 - End with the start symbol
 - Match substring on RHS of production, replace by LHS

$$\begin{aligned}
 (1+2+(3+4))+5 &\leftarrow (E+2+(3+4))+5 \\
 &\leftarrow (S+2+(3+4))+5 \leftarrow (S+E+(3+4))+5 \\
 &\leftarrow (S+(3+4))+5 \leftarrow (S+(E+4))+5 \leftarrow (S+(S+4))+5 \\
 &\leftarrow (S+(S+E))+5 \leftarrow (S+(S))+5 \leftarrow (S+E)+5 \leftarrow (S)+5 \\
 &\leftarrow E+5 \leftarrow S+E \leftarrow S
 \end{aligned}$$

2/16/2008 CSS810 Spring 2008 4

Bottom-Up Parsing

$S \rightarrow S + E \mid E$
 $E \rightarrow \text{num} \mid (S)$

(1+2+(3+4))+5
 $\leftarrow (E+2+(3+4))+5$
 $\leftarrow (S+2+(3+4))+5$
 $\leftarrow (S+E+(3+4))+5$

Advantage of bottom-up parsing:
can postpone the selection of productions until more of the input is scanned

2/16/2008 CSS810 Spring 2008 5

Top-Down Parsing

$S \rightarrow S + E \mid E$
 $E \rightarrow \text{num} \mid (S)$

$S \rightarrow S+E \rightarrow E+E \rightarrow (S)+E \rightarrow (S+E)+E$
 $\rightarrow (S+E+E)+E \rightarrow (E+E+E)+E$
 $\rightarrow (1+E+E)+E \rightarrow (1+2+E)+E \dots$

In left-most derivation, entire tree above token (2) has been expanded when encountered

2/16/2008 CSS810 Spring 2008 6

Top-Down v.s. Bottom-Up

Bottom-up: Don't need to figure out as much of the parse tree for a given amount of input → More time to decide what rules to apply

Top-down Bottom-up

2/16/2008 CSS810 Spring 2008 7

Terminology: LL vs LR

- **LL(k)**
 - Left-to-right scan of input
 - Left-most derivation
 - k symbol lookahead
 - [Top-down or predictive] parsing or LL parser
 - Performs pre-order traversal of parse tree
- **LR(k)**
 - Left-to-right scan of input
 - Right-most derivation
 - k symbol lookahead
 - [Bottom-up or shift-reduce] parsing or LR parser
 - Performs post-order traversal of parse tree

2/16/2008 CSS810 Spring 2008 8

Another example

- Consider the following grammar:
 - $S \rightarrow aABe$
 - $A \rightarrow Abc \mid b$
 - $B \rightarrow d$
- Consider sentence: $abbcd$
 - $abbcd \Rightarrow aAbcde \Rightarrow aAde \Rightarrow aABe \Rightarrow S$
- The parser must find a substring β of the tree's frontier that matches some production $A \rightarrow \beta$ that occurs as one step in the rightmost derivation
- Informally, we call this substring β a handle

2/16/2008 CSS810 Spring 2008 9

Handles

- A handle of a string is a substring which matches the RHS of a production and whose reduction represents one step along the reverse of a rightmost derivation.
- A handle of a sentential form γ is a production $A \rightarrow \beta$ and a position of γ where the string β may be found and replaced by A to produce the previous sentential form in a rightmost derivation of γ

2/16/2008 CSS810 Spring 2008 10

Shift-Reduce Parsing

- Parsing actions: A sequence of **shift** and **reduce** operations
- Parser state: A stack of terminals and non-terminals (grows to the right)
- Current derivation step = stack + input

| Derivation step | stack | Unconsumed input |
|----------------------------|-------|------------------|
| $(1+2+(3+4))+5 \leftarrow$ | | $(1+2+(3+4))+5$ |
| $(E+2+(3+4))+5 \leftarrow$ | (E | $+2+(3+4))+5$ |
| $(S+2+(3+4))+5 \leftarrow$ | (S | $+2+(3+4))+5$ |
| $(S+E+(3+4))+5 \leftarrow$ | (S+E | $+(3+4))+5$ |
| ... | | |

2/16/2008 CSS810 Spring 2008 11

Shift-Reduce Actions

- Parsing is a sequence of shifts and reduces
- Shift: move look-ahead token to stack

| stack | input | action |
|-------|----------------|---------|
| (| $1+2+(3+4))+5$ | shift 1 |
| (1 | $+2+(3+4))+5$ | |
- Reduce: Replace symbols β from top of stack with non-terminal symbol X corresponding to the production: $X \rightarrow \beta$ (e.g., pop β , push X)

| stack | input | action |
|-------|-------------|----------------------------|
| (S+E | $+(3+4))+5$ | reduce $S \rightarrow S+E$ |
| (S | $+(3+4))+5$ | |

2/16/2008 CSS810 Spring 2008 12

Shift-Reduce Parsing

$S \rightarrow S + E \mid E$
 $E \rightarrow \text{num} \mid (S)$

| derivation | stack | input stream | action |
|---------------|-------|---------------|-----------------------------------|
| (1+2+(3+4))+5 | | (1+2+(3+4))+5 | shift |
| (1+2+(3+4))+5 | (| 1+2+(3+4))+5 | shift |
| (1+2+(3+4))+5 | (1 | +2+(3+4))+5 | reduce $E \rightarrow \text{num}$ |
| (E+2+(3+4))+5 | (E | +2+(3+4))+5 | reduce $S \rightarrow E$ |
| (S+2+(3+4))+5 | (S | +2+(3+4))+5 | shift |
| (S+2+(3+4))+5 | (S+ | 2+(3+4))+5 | shift |
| (S+2+(3+4))+5 | (S+2 | +(3+4))+5 | reduce $E \rightarrow \text{num}$ |
| (S+E+(3+4))+5 | (S+E | +(3+4))+5 | reduce $S \rightarrow S+E$ |
| (S+(3+4))+5 | (S | +(3+4))+5 | shift |
| (S+(3+4))+5 | (S+ | (3+4))+5 | shift |
| (S+(3+4))+5 | (S+(| 3+4))+5 | shift |
| (S+(3+4))+5 | (S+(3 | +4))+5 | reduce $E \rightarrow \text{num}$ |
| ... | | | |

2/16/2008 CSS810 Spring 2008 13

Potential Problems

- How do we know which action to take: whether to shift or reduce, and which production to apply
- Issues
 - Sometimes can reduce but should not
 - Sometimes can reduce in different ways

2/16/2008 CSS810 Spring 2008 14

Action Selection Problem

- Given stack β and look-ahead symbol b , should parser:
 - Shift b onto the stack making it βb ?
 - Reduce $X \rightarrow \gamma$ assuming that the stack has the form $\beta = \alpha\gamma$ making it αX ?
- If stack has the form $\alpha\gamma$, should apply reduction $X \rightarrow \gamma$ (or shift) depending on stack prefix α ?
 - α is different for different possible reductions since γ 's have different lengths

2/16/2008 CSS810 Spring 2008 15

LR Parsing Engine

- Basic mechanism
 - Use a set of parser states
 - Use stack with alternating symbols and states
 - E.g., 1 (6 S 10 + 5 (blue = state numbers)
 - Use parsing table to:
 - Determine what action to apply (shift/reduce)
 - Determine next state
- The parser actions can be precisely determined from the table

2/16/2008 CSS810 Spring 2008 16

LR Parsing Table

| State | Terminals | Non-terminals |
|-------|----------------------------|---------------|
| | Next action and next state | Next state |
| | Action table | Goto table |

2/16/2008 CSS810 Spring 2008 17

LR Parsing Table Example

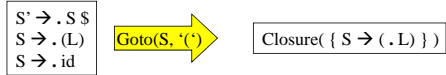
$S \rightarrow (L) \mid \text{id}$
 $L \rightarrow S \mid L,S$

| State | Input terminal | | | | | Non-terminals | |
|-------|----------------|-------|-------|-------|--------|---------------|----|
| | (|) | id | , | \$ | S | L |
| 1 | s3 | | s2 | | | g4 | |
| 2 | S→id | S→id | S→id | S→id | S→id | | |
| 3 | s3 | | s2 | | | g7 | g5 |
| 4 | | | | | accept | | |
| 5 | | | s6 | | s8 | | |
| 6 | S→(L) | S→(L) | S→(L) | S→(L) | S→(L) | | |
| 7 | L→S | L→S | L→S | L→S | L→S | | |
| 8 | s3 | | s2 | | | g9 | |
| 9 | L→L,S | L→L,S | L→L,S | L→L,S | L→L,S | | |

2/16/2008 CSS810 Spring 2008 18

The Goto Operation

- Goto operation = describes transitions between parser states, which are sets of items
- Algorithm: for state S and a symbol Y
 - If the item $[X \rightarrow \alpha \cdot Y \beta]$ is in I, then
 - $\text{Goto}(I, Y) = \text{Closure}(\{X \rightarrow \alpha Y \cdot \beta\})$



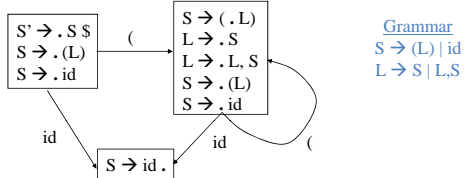
Exercise

```

E' -> E
E -> E + T | T
T -> T * F | F
F -> (E) | id
    
```

1. If $I = \{ [E' \rightarrow \cdot E] \}$, then $\text{Closure}(I) = ??$
2. If $I = \{ [E' \rightarrow E \cdot], [E \rightarrow E \cdot + T] \}$, then $\text{Goto}(I, +) = ??$

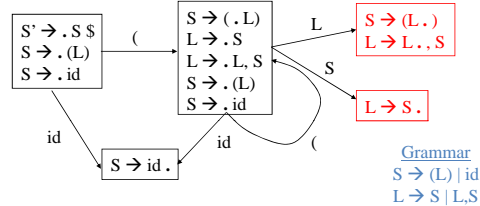
Shift: Terminal Symbols



Grammar
 $S \rightarrow (L) | id$
 $L \rightarrow S | L, S$

In new state, include all items that have appropriate input symbol just after dot, advance do in those items and take closure

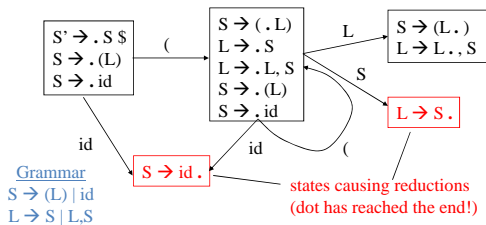
Goto: Non-terminal Symbols



Grammar
 $S \rightarrow (L) | id$
 $L \rightarrow S | L, S$

same algorithm for transitions on non-terminals

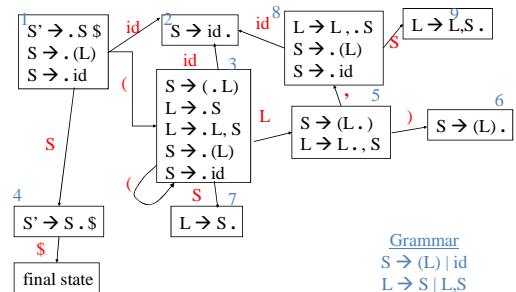
Applying Reduce Actions



Grammar
 $S \rightarrow (L) | id$
 $L \rightarrow S | L, S$

Pop RHS off stack, replace with LHS X ($X \rightarrow \beta$), then rerun DFA (e.g., (x))

Full DFA



Grammar
 $S \rightarrow (L) | id$
 $L \rightarrow S | L, S$

Computed LR Parsing Table

| State | Input terminal | | | | | Non-terminals | |
|-------|----------------|-------|-------|-------|--------|---------------|----|
| | (|) | id | , | \$ | S | L |
| 1 | s3 | | s2 | | | g4 | |
| 2 | S→id | S→id | S→id | S→id | S→id | | |
| 3 | s3 | | s2 | | | g7 | g5 |
| 4 | | | | | accept | | |
| 5 | | s6 | | s8 | | | |
| 6 | S→(L) | S→(L) | S→(L) | S→(L) | S→(L) | | |
| 7 | L→S | L→S | L→S | L→S | L→S | | |
| 8 | s3 | | s2 | | | g9 | |
| 9 | L→L,S | L→L,S | L→L,S | L→L,S | L→L,S | | |

blue = shift red = reduce

2/16/2008

CS5810 Spring 2008

31

Building the Parsing Table

- States in the table = states in the DFA
- For transition $S \rightarrow S'$ on terminal C:
 - Table[S,C] += Shift(S')
- For transition $S \rightarrow S'$ on non-terminal N:
 - Table[S,N] += Goto(S')
- If S is a reduction state $X \rightarrow \beta$ then:
 - Table[S,*] += Reduce($X \rightarrow \beta$)

2/16/2008

CS5810 Spring 2008

32

Parsing Algorithm

- Algorithm: look at entry for current state s and input terminal a
 - If Table[s,a] = shift t then shift:
 - push(t), let a be the next input symbol
 - If Table[s,a] = $X \rightarrow \alpha$ then reduce:
 - pop(| α |), t= top(), push(GOTO[t,X]), output $X \rightarrow \alpha$

2/16/2008

CS5810 Spring 2008

33

Parsing Example ((a),b)

| derivation | stack | input | action | $S \rightarrow (L) id$ $L \rightarrow S L,S$ |
|------------|-----------|-----------|---------------|---|
| ((a),b) ← | 1 | ((a),b)\$ | shift, goto 3 | |
| ((a),b) ← | 1(3 | (a),b)\$ | shift, goto 3 | |
| ((a),b) ← | 1(3(3 | a),b)\$ | shift, goto 2 | |
| ((a),b) ← | 1(3(3a2 |),b)\$ | reduce S→id | |
| ((S),b) ← | 1(3(3S7 |),b)\$ | reduce L→S | |
| ((L),b) ← | 1(3(3L5 |),b)\$ | shift, goto 6 | |
| ((L),b) ← | 1(3(3L5)6 |),b)\$ | reduce S→(L) | |
| (S,b) ← | 1(3S7 |),b)\$ | reduce L→S | |
| (L,b) ← | 1(3L5 |),b)\$ | shift, goto 8 | |
| (L,b) ← | 1(3L5,8 | b)\$ | shift, goto 9 | |
| (L,b) ← | 1(3L5,8b2 |)\$ | reduce S→id | |
| (L,S) ← | 1(3L5,8S9 |)\$ | reduce L→L,S | |
| (L) ← | 1(3L5 |)\$ | shift, goto 6 | |
| (L) ← | 1(3L5)6 | \$ | reduce S→(L) | |
| S ← | 1S4 | \$ | done | |

2/16/2008

CS5810 Spring 2008

34

Reductions

- On reducing $X \rightarrow \beta$ with stack $\alpha\beta$
 - Pop β off stack, revealing prefix α and state
 - Take single step in DFA from top state
 - Push X onto stack with new DFA state
- Example

| derivation | stack | input | action |
|------------|---------|-------|---------------|
| ((a),b) ← | 1(3(3 | a),b) | shift, goto 2 |
| ((a),b) ← | 1(3(3a2 |),b) | reduce S → id |
| ((S),b) ← | 1(3(3S7 |),b) | reduce L → S |

2/16/2008

CS5810 Spring 2008

35

LR(0) Summary

- LR(0) parsing recipe:
 - Start with LR(0) grammar
 - Compute LR(0) states and build DFA:
 - Use the closure operation to compute states
 - Use the goto operation to compute transitions
 - Build the LR(0) parsing table from the DFA
- This can be done automatically

2/16/2008

CS5810 Spring 2008

36

The Parser Generator Yacc

- A Yacc source program has three parts
 - Declarations
 - Translation rules
 - Supporting C routines
- Unix system command
 - yacc foo.y transforms foo.y into y.tab.c
 - gcc y.tab.c –ly obtains binary file a.out

2/16/2008

CSS810 Spring 2008

37

Yacc Example

```

E → E + T | T           %{
T → T * F | F           #include <ctype.h>
F → (E) | digit         %}
                          %token DIGIT

line : expr '\n' {printf("%d\n", $1); }
      ;
expr  : expr '+' term {$$ = $1 + $3; }
      | term
      ;
term  : term '*' factor {$$ = $1 * $3; }
      | factor
factor : '(' expr ')' {$$ = $2; }
      | DIGIT
      ;

```

2/16/2008

CSS810 Spring 2008

38

Yacc Example

```

%%
yylex() {
    int c;
    c = getchar();
    if (isdigit(c)) {
        yylval = c - '0';
        return DIGIT;
    }
    return c;
}

```

2/16/2008

CSS810 Spring 2008

39

Using Yacc with Ambiguous Grammars

- Unless otherwise instructed
 - A reduce/reduce conflict is resolved by choosing the conflicting production listed first
 - A shift/reduce conflict is resolved in favor of shift

2/16/2008

CSS810 Spring 2008

40

Yacc With Ambiguous Grammars

```

E → E + E | E - E           %{...
| E * E | E / E | -E       %}
number                     %token DIGIT
                          %left '+' '-'
                          %left '*' '/'
                          %right UMINUS
                          %%

lines : lines expr '\n' {printf("%d\n", $2); }
      | /*empty*/
      ;
expr  : expr '+' expr {$$ = $1 + $3; }
      | expr '*' expr {$$ = $1 * $3; }
      ;

```

2/16/2008

CSS810 Spring 2008

41

Yacc With Lex

```

number [0-9]+\e.?[0-9]*\e.[0-9]+
%%
[ ] { }
{number} {sscanf(yytext, "%lf", &yylval);
          return NUMBER;
}
\n|. {return yytext[0];}
%%
yylex() {
    int c;
    c = getchar();
    if (isdigit(c)) {
        yylval = c - '0';
        return DIGIT;
    }
    return c;
}

```

2/16/2008

CSS810 Spring 2008

42