

**Winter 2006-2007
Compiler Construction
T3 – Syntax Analysis
(Parsing, part 1 of 2)**

Mooly Sagiv and Roman Manevich
School of Computer Science
Tel-Aviv University

Material taught in lecture

- The task of syntax analysis
- Error handling
- Context-Free Grammars
 - Derivations and Parsing Trees
 - Leftmost derivation, rightmost derivation
 - Parse trees
- Ambiguous grammars
 - Rewriting grammar
- Top-down (predictive) parsing
 - LL(k) / Recursive-descent
- Top-Down vs. Bottom-Up parsing

2

Today

<small>ic</small>	<small>Lexical Analysis</small>	<small>Syntax Analysis Parsing</small>	<small>AST</small>	<small>Symbol Table etc.</small>	<small>Inter. Rep. (IR)</small>	<small>Code Generation</small>	<small>exe</small>
IC Language							Executable code

- Today:
 - Review
 - Grammars, parse trees, ambiguity
 - Top-down parsing
 - **Bottom-up parsing**
- Next week:
 - Conflict resolution
 - Shift/Reduce parsing via JavaCup
 - (Error handling)
 - AST intro.
 - PA2

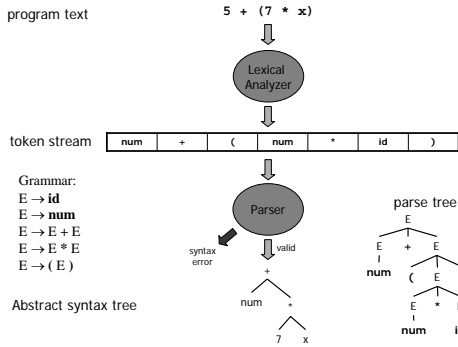
3

Goals of parsing

- Programming language has syntactic rules
 - Context-Free Grammars
- Decide whether program satisfies syntactic structure
 - Error detection
 - Error recovery
 - Simplification: rules on tokens
- Build Abstract Syntax Tree

4

From text to abstract syntax



Terminology

Grammar rules:

$E \rightarrow \text{id}$
 $E \rightarrow \text{num}$
 $E \rightarrow E + E$
 $E \rightarrow E * E$
 $E \rightarrow (E)$

Symbols:

terminals (tokens) + * () id num
non-terminals E

Derivation:

E
 $E + E$
 $1 + E$
 $1 + E + E$
 $1 + 2 + E$
 $1 + 2 * 3$

Parse tree:

```

graph TD
    E1[E] --- P1[+]
    E1 --- E2[E]
    E2 --- P2[(]
    E2 --- E3[E]
    E3 --- P3[id]
    E3 --- P4[id]
    
```

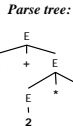
6

Ambiguity

Grammar rules:
 $E \rightarrow id$
 $E \rightarrow num$
 $E \rightarrow E + E$
 $E \rightarrow E * E$
 $E \rightarrow (E)$

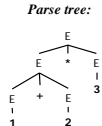
Leftmost derivation

Derivation:
 E
 $E + E$
 $1 + E$
 $1 + E + E$
 $1 + 2 + E$
 $1 + 2 * 3$



Rightmost derivation

Derivation:
 E
 $E * E$
 $E * 3$
 $E + E * 3$
 $E + 2 * 3$
 $1 + 2 * 3$

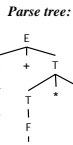


Grammar rewriting

Ambiguous grammar:
 $E \rightarrow id$
 $E \rightarrow num$
 $E \rightarrow E + E$
 $E \rightarrow E * E$
 $E \rightarrow (E)$

Non-ambiguous grammar:
 $E \rightarrow E + T$
 $E \rightarrow T$
 $T \rightarrow T * F$
 $T \rightarrow F$
 $F \rightarrow id$
 $F \rightarrow (E)$

Derivation:
 E
 $E + T$
 $1 + T$
 $1 + T * F$
 $1 + F * F$
 $1 + 2 * F$
 $1 + 2 * 3$



Parsing methods

- Top-down / predictive / recursive descent without backtracking : LL(1)
 - "L" – left-to-right scan of input
 - "L" – leftmost derivation
 - "1" – predict based on one token look-ahead
 - For every non-terminal and token **predict** the next production
- Bottom-up : LR(0), SLR(1), LR(1), LALR(1)
 - "L" – left-to-right scan of input
 - "R" – rightmost derivation (in the reversed order)
 - For every potential right hand side and token decide when a production is found

Top-down parsing

- Builds parse tree in preorder
- LL(1) example

<p>Grammar:</p> <p>$S \rightarrow \text{if } E \text{ then } S \text{ else } S$ $S \rightarrow \text{begin } S \text{ L}$ $S \rightarrow \text{print } E$ $L \rightarrow \text{end}$ $L \rightarrow ; S L$ $E \rightarrow \text{num}$</p>	<p>if 5 then print 8 else...</p> <p>Token : rule</p> <p>if : $S \rightarrow \text{if } E \text{ then } S \text{ else } S$ if E then S else S</p> <p>5 : $E \rightarrow \text{num}$ if 5 then S else S</p> <p>print : print E if 5 then print E else S</p> <p>...</p>
---	---

10

Problem: left recursion

- Left recursion: $E \rightarrow E + T$
 - Symbol on left also first symbol on right
- Predictive parsing fails when two rules can start with same token
- Rewrite grammar using left-factoring
- Nullable, FIRST, FOLLOW sets

<p>Arithmetic expressions:</p> <p>$E \rightarrow E + T$ $E \rightarrow T$ $T \rightarrow T * F$ $T \rightarrow F$ $F \rightarrow \text{id}$ $F \rightarrow (E)$</p>	<p>Left factored grammar:</p> <p>$E \rightarrow T E^p$ $F \rightarrow \text{id}$ $E^p \rightarrow + T E^p$ $F \rightarrow (E)$ $E^p \rightarrow$ $T \rightarrow F T^p$ $T^p \rightarrow * F T^p$ $T^p \rightarrow$</p>
--	---

11

More left recursion

- Non-terminal with two rules starting with same prefix

<p>Grammar:</p> <p>$S \rightarrow \text{if } E \text{ then } S \text{ else } S$ $S \rightarrow \text{if } E \text{ then } S$</p>	<p>Left factored grammar:</p> <p>$S \rightarrow \text{if } E \text{ then } S X$ $X \rightarrow$ $X \rightarrow \text{else } S$</p>
--	--

12

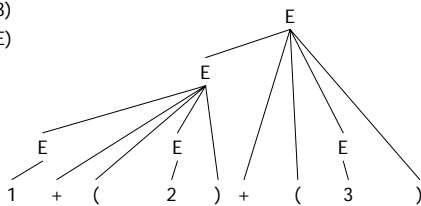
Bottom-up parsing

- No problem with left recursion
- Widely used in practice
- LR(0), SLR(1), LR(1), LALR(1)
- JavaCup implements LALR(1)

13

Bottom-up parsing

- 1 + (2) + (3)
 - E + (2) + (3)
 - E + (E) + (3)
 - E + (3)
 - E + (E)
 - E
- $E \rightarrow E + (E)$
 $E \rightarrow i$



14

Shift-reduce parsing

- Parser stack: symbols (terminal and non-terminals) + automaton states
- Parsing actions: sequence of shift and reduce operations
- Action determined by top of stack and k input tokens
- Shift: move next token to top of stack
- Reduce: for rule $X \rightarrow A B C$
pop C, B, A then push X
- Convention: \$ stands for end of file

15

Items

Items indicate the position inside a rule:
 LR(0) items are of the form $A \rightarrow \alpha \bullet t$
 (LR(1) items are of the form $A \rightarrow \alpha \bullet t, \beta$)

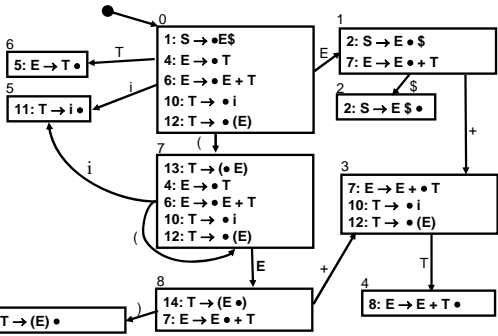
Grammar:

$S \rightarrow ES$
 $E \rightarrow T$
 $E \rightarrow E + T$
 $T \rightarrow i$
 $T \rightarrow (E)$

- 1: $S \rightarrow \bullet ES$
- 2: $S \rightarrow E \bullet S$
- 3: $S \rightarrow ES \bullet$
- 4: $E \rightarrow \bullet T$
- 5: $E \rightarrow T \bullet$
- 6: $E \rightarrow \bullet E + T$
- 7: $E \rightarrow E \bullet + T$
- 8: $E \rightarrow E + \bullet T$
- 9: $E \rightarrow E + T \bullet$
- 10: $T \rightarrow \bullet i$
- 11: $T \rightarrow i \bullet$
- 12: $T \rightarrow \bullet (E)$
- 13: $T \rightarrow (\bullet E)$
- 14: $T \rightarrow (E \bullet)$
- 15: $T \rightarrow (E) \bullet$

19

Automaton states



20

Identifying handles

- Create a finite state automaton over grammar symbols
 - Sets of LR(0) items
- Use automaton to build parser tables
 - shift For items $A \rightarrow \alpha \bullet t \beta$ on token t
 - reduce For items $A \rightarrow \alpha \bullet$ on every token
- Any grammar has
 - Transition diagram
 - GOTO table
- Not every grammar has deterministic action table
- When no conflicts occur use a DPDA which pushes states on the stack

21

Non-LR(0) grammars

- When conflicts occur the grammar is not LR(0)
 - Parsing table contains non-determinism
 - shift-reduce conflicts
 - reduce-reduce conflicts
 - shift-shift conflicts?
- Known cases
 - Operator precedence
 - Operator associativity
 - Dangling if-then-else
 - Unary minus
- Solutions
 - Develop equivalent non-ambiguous grammar
 - Patch parsing table to shift/reduce
 - Precedence and associativity of tokens
 - Stronger parser algorithm: SLR/LR(1)/LALR(1)

22

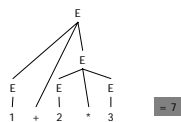
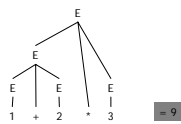
Precedence and associativity

- Precedence
 - $E \rightarrow E + E * E$
 - $E \rightarrow E + E \bullet$
- Reduce
 - + precedes *
- Shift
 - * precedes +

23

Precedence and associativity

- Precedence
 - $E \rightarrow E + E * E$
 - $E \rightarrow E + E \bullet$
- Reduce
 - + precedes *
- Shift
 - * precedes +



24

Precedence and associativity

- Precedence
 - $E \rightarrow E+E \bullet *E$
 - $E \rightarrow E+E \bullet$
- Reduce
 - + precedes *
- Shift
 - * precedes +
- Associativity
 - $E \rightarrow E+E \bullet +E$
 - $E \rightarrow E+E \bullet$
- Shift
 - + right-associative
- Reduce
 - + left-associative

25

Dangling else/if-else ambiguity

Grammar:
 $S \rightarrow \text{if } E \text{ then } S \text{ else } S$
 $S \rightarrow \text{if } E \text{ then } S$
 $S \rightarrow \text{other}$

if a then if b then e1 else e2
which interpretation should we use?

- (1) if a then { if b then e1 else e2 } -- standard interpretation
- (2) if a then { if b then e1 } else e2

shift/reduce conflict

LR(1) items:	token:
$S \rightarrow \text{if } E \text{ then } S \bullet$	else
$S \rightarrow \text{if } E \text{ then } S \bullet \text{ else } S$	(any)

26

See you next week

27

Grammar hierarchy

