

Winter 2007-2008 Compiler Construction T4 – Syntax Analysis (Parsing, part 2 of 2)

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

javadoc

```

/** The entry point for the IC compiler.
 */
class Compiler {
  /** Determines whether the tokens discovered
   *   by the scanner should be written to System.out.
   */
  public static boolean emitTokens = false;

  /** Print usage information for the compiler.
   *   * @param includeSpecialOptions Should special options be printed.
   */
  public void printUsage(boolean includeSpecialOptions) {
    ...
  }
}

```

- Enter at src directory: javadoc IC/*.java
- Or use Ant, enter: ant javadocs

Today

IC
Language

Lexical
Analysis

Syntax
Analysis
Parsing

AST

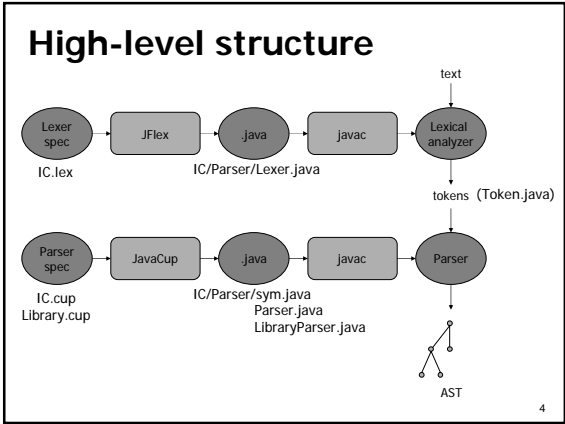
Symbol
Table
etc.

Inter.
Rep.
(IR)

Code
Generation

exe
Executable
code

- Today:
 - LR(0) parsing algorithms
 - JavaCup
 - AST intro
 - PA2
 - Missing: error recovery



Expression calculator

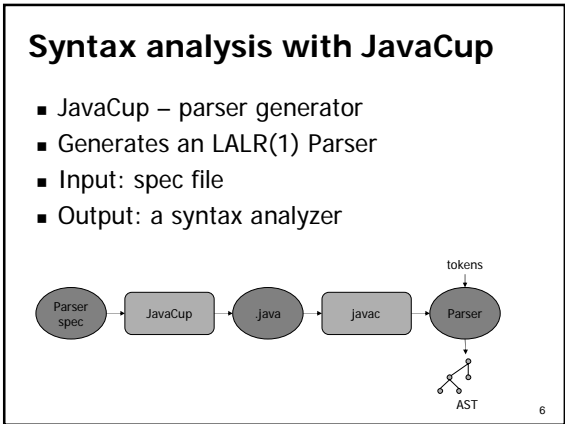
```

expr -> expr + expr
      | expr - expr
      | expr * expr
      | expr / expr
      | - expr
      | ( expr )
      | number

```

Goals of expression calculator parser:

- Is 2+3+4+5 a valid expression?
- What is the meaning (value) of this expression?



JavaCup spec file

- Package and import specifications
- User code components
- Symbol (terminal and non-terminal) lists
 - Terminals go to `sym.java`
 - Types of AST nodes
- Precedence declarations
- The grammar
 - Semantic actions to construct AST

7

Expression Calculator – 1st Attempt

```
terminal Integer NUMBER;
terminal PLUS, MINUS, MULT, DIV;
terminal LPAREN, RPAREN;
non terminal Integer expr;

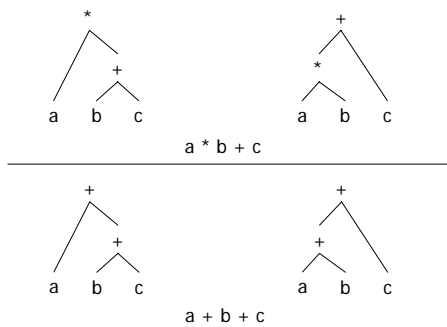
expr ::= expr PLUS expr
      | expr MINUS expr
      | expr MULT expr
      | expr DIV expr
      | MINUS expr
      | LPAREN expr RPAREN
      | NUMBER
;

```

Symbol type explained later

8

Ambiguities



9

Expression Calculator – 2nd Attempt

```
terminal Integer NUMBER;
terminal PLUS, MINUS, MULT, DIV;
terminal LPAREN, RPAREN;
terminal UMINUS;
non terminal Integer expr;

precedence left PLUS, MINUS;
precedence left DIV, MULT;
precedence left UMINUS;

expr ::= expr PLUS expr
      | expr MINUS expr
      | expr MULT expr
      | expr DIV expr
      | MINUS expr %prec UMINUS
      | LPAREN expr RPAREN
      | NUMBER
;

```

Increasing precedence

Contextual precedence

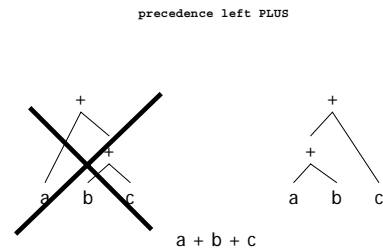
10

Parsing ambiguous grammars using precedence declarations

- Each terminal assigned with precedence
 - By default all terminals have lowest precedence
 - User can assign his own precedence
 - CUP assigns each production a precedence
 - Precedence of last terminal in production
 - or user-specified contextual precedence
- On shift/reduce conflict resolve ambiguity by comparing precedence of terminal and production and decides whether to shift or reduce
- In case of equal precedences **left/right** help resolve conflicts
 - left** means reduce
 - right** means shift
- More information on [precedence declarations](#) in CUP's manual

11

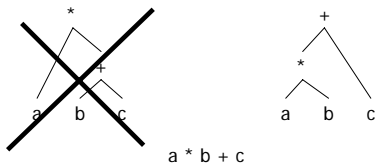
Resolving ambiguity



12

Resolving ambiguity

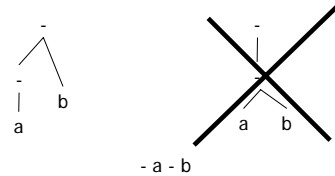
precedence left PLUS
precedence left MULT



13

Resolving ambiguity

MINUS expr %prec UMINUS



14

Resolving ambiguity

```
terminal Integer NUMBER;
terminal PLUS,MINUS,MULT,DIV;
terminal LPAREN, RPAREN;
terminal UMINUS;
```

UMINUS never returned
by scanner
(used only to define precedence)

```
precedence left PLUS, MINUS;
precedence left DIV, MULT;
precedence left UMINUS;
```

```
expr ::= expr PLUS expr
      | expr MINUS expr
      | expr MULT expr
      | expr DIV expr
      | MINUS expr %prec UMINUS
      | LPAREN expr RPAREN
      | NUMBER
      ;
```

Rule has
precedence of
UMINUS

15

More CUP directives

- **precedence nonassoc NEQ**
 - Non-associative operators: < > == != etc.
 - 1<2<3 identified as an error (semantic error?)
- **start non-terminal**
 - Specifies start non-terminal other than first non-terminal
 - Can change to test parts of grammar
- **Getting internal representation**
 - Command line options:
 - -dump_grammar
 - -dump_states
 - -dump_tables
 - -dump

16

CUP API

- Link on the course web page to API
 - Parser extends `java_cup.runtime.lr_parser`
- Various methods to report syntax errors, e.g., override `syntax_error(Symbol cur_token)`

17

Scanner integration

```
import java_cup.runtime.*;
%%
%cup
%eofval{
    return new Symbol(sym.EOF);
%eofval}
NUMBER={0-9}+
%%
<YYINITIAL>"+" { return new Symbol(sym.PLUS); }
<YYINITIAL>"-" { return new Symbol(sym.MINUS); }
<YYINITIAL>"*" { return new Symbol(sym.MULT); }
<YYINITIAL>"/" { return new Symbol(sym.DIV); }
<YYINITIAL>"(" { return new Symbol(sym.LPAREN); }
<YYINITIAL>")" { return new Symbol(sym.RPAREN); }
<YYINITIAL>{NUMBER} {
    return new Symbol(sym.NUMBER, new Integer(yytext()));
}
<YYINITIAL>"\n" { }
<YYINITIAL> "." { }
```

Generated from token
declarations in .cup file

Parser gets terminals from the scanner

18

Recap

- Package and import specifications and user code components
- Symbol (terminal and non-terminal) lists
 - Define building-blocks of the grammar
- Precedence declarations
 - May help resolve conflicts
- The grammar
 - May introduce conflicts that have to be resolved

19

Assigning meaning

```

expr ::= expr PLUS expr
      | expr MINUS expr
      | expr MULT expr
      | expr DIV expr
      | MINUS expr %prec UMINUS
      | LPAREN expr RPAREN
      | NUMBER
    
```

- So far, only validation
- Add Java code implementing semantic actions

20

Assigning meaning

```

expr ::= expr:e1 PLUS expr:e2
      { : RESULT = new Integer(e1.intValue() + e2.intValue()); :}
      | expr:e1 MINUS expr:e2
      { : RESULT = new Integer(e1.intValue() - e2.intValue()); :}
      | expr:e1 MULT expr:e2
      { : RESULT = new Integer(e1.intValue() * e2.intValue()); :}
      | expr:e1 DIV expr:e2
      { : RESULT = new Integer(e1.intValue() / e2.intValue()); :}
      | MINUS expr:e1
      { : RESULT = new Integer(0 - e1.intValue()); :} %prec UMINUS
      | LPAREN expr:e1 RPAREN
      { : RESULT = e1; :}
      | NUMBER:n
      { : RESULT = n; :}
    ,
    
```

- Symbol labels used to name variables
- RESULT names the left-hand side symbol

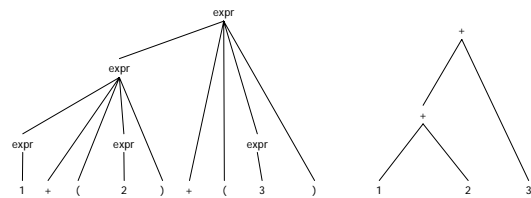
21

Building an AST

- More useful representation of syntax tree
 - Less clutter
 - Actual level of detail depends on your design
- Basis for semantic analysis
- Later annotated with various information
 - Type information
 - Computed values

22

Parse tree vs. AST



23

AST construction

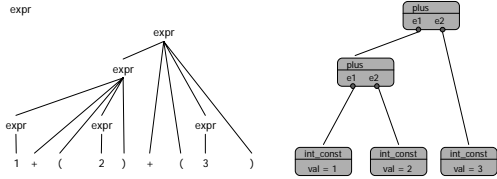
- AST Nodes constructed during parsing
 - Stored in push-down stack
- Bottom-up parser
 - Grammar rules annotated with actions for AST construction
 - When node is constructed all children available (already constructed)
 - Node (RESULT) pushed on stack
- Top-down parser
 - More complicated

24

AST construction

```

1 + (2) + (3)      expr ::= expr:e1 PLUS expr:e2
                    { : RESULT = new plus(e1,e2); : }
expr + (2) + (3)   | LPAREN expr:e RPAREN
                    { : RESULT = e; : }
expr + (expr) + (3) | INT_CONST:i
                    { : RESULT = new int_const(..., i); : }
expr + (3)
expr + (expr)
expr
    
```



25

Designing an AST

```

terminal Integer NUMBER;
terminal PLUS,MINUS,MULT,DIV,LPAREN,RPAREN,SEMI;
terminal UMINUS;
non terminal Integer expr;
non terminal expr_list, expr_part;
precedence left PLUS, MINUS;
precedence left DIV, MULT;
precedence left UMINUS;

expr_list ::= expr_list expr_part
            | expr_part

expr_part ::= expr:e { : System.out.println(" = " + e); : } SEMI

expr ::= expr PLUS expr
       | expr MINUS expr
       | expr MULT expr
       | expr DIV expr
       | MINUS expr %prec UMINUS
       | LPAREN expr RPAREN
       | NUMBER
    
```

26

Designing an AST

- Rules of thumb
 - Interfaces or abstract classes for non-terminals with alternatives
 - Class for each non-terminal or group of related non-terminals with similar functionality
- Remember - bottom-up
 - When constructing a node children nodes already constructed
 - but parent not constructed yet

27

Designing an AST

```

expr_list ::= expr_list expr_part
            | expr_part

expr_part ::= expr SEMI

expr ::= expr PLUS expr
       | expr MINUS expr
       | expr MULT expr
       | expr DIV expr
       | MINUS expr %prec UMINUS
       | LPAREN expr RPAREN
       | NUMBER
    
```

ExprProgram
Expr
Alternative 1: class for each op:
op type PlusExpr
field of Expr MinusExpr
MultExpr
DivExpr
UnaryMinusExpr
ValueExpr

28

Designing an AST

```

terminal Integer NUMBER;
non terminal Expr expr; expr_part;
non terminal ExprProgram expr_list;
expr_list ::= expr_list:el expr_part:ep
            { : RESULT = el.addExpressionPart(ep); : }
            | expr_part:ep
            { : RESULT = new ExprProgram(ep); : }
;

expr_part ::= expr:e SEMI
            { : RESULT = e; : }
;

expr ::= expr:e1 PLUS expr:e2
       | expr:e1 MINUS expr:e2
       | expr:e1 MULT expr:e2
       | expr:e1 DIV expr:e2
       | MINUS expr:e1
       | LPAREN expr:el RPAREN
       | NUMBER:n
       { : RESULT = new Expr(n); : }
    
```

29

Designing an AST

```

public abstract class ASTNode {
    // common AST nodes functionality
}

public class Expr extends ASTNode {
    private int value;
    private Expr left;
    private Expr right;
    private String operator;

    public Expr(Integer val) {
        value = val.intValue();
    }
    public Expr(Expr operand, String op) {
        this.left = operand;
        this.operator = op;
    }
    public Expr(Expr left, Expr right, String op) {
        this.left = left;
        this.right = right;
        this.operator = op;
    }
}
    
```

30

Computing meaning

- Evaluate expression by AST traversal
- Traversal for debug printing
- Later – annotate AST
- More on AST next recitation

31

PA2

- Write parser for IC
- Write parser for `libc.sig`
- Check syntax
 - Emit either “Parsed [file] successfully!” or “Syntax error in [file]: [details]”
- -print-ast option
 - Prints one AST node per line

32

PA2 – step 1

- Understand IC grammar in the manual
 - Don't touch the keyboard before understanding spec
- Write a debug JavaCup spec for IC grammar
 - A spec with “debug actions” : print-out debug messages to understand what's going on
- Try “debug grammar” on a number of test cases
- Keep a copy of “debug grammar” spec around
- Optional: perform error recovery
 - Use JavaCup error token

33

PA2 – step 2

- Design AST class hierarchy
 - Flesh out AST class hierarchy
 - Don't touch the keyboard before you understand the hierarchy
 - Keep in mind that this is the basis for later stages
- Web-site contains an AST adapted with permission from Tovi Almozilino
- Change CUP actions to construct AST nodes

34

Partial example of main

```
import java.io.*;
import IC.Lexer.Lexer;
import IC.Parser.*;
import IC.AST.*;

public class Compiler {
    public static void main(String[] args) {
        try {
            FileReader txtFile = new FileReader(args[0]);
            Lexer scanner = new Lexer(txtFile);
            Parser parser = new Parser(scanner);
            // parser.parse() returns Symbol, we use its value
            ProgAST root = ((ProgAST) parser.parse().value);
            System.out.println("Parsed " + args[0] + " successfully!");
        } catch (SyntaxError e) {
            System.out.print("Syntax error in " + args[0] + ": " + e);
        }

        if (libraryFileSpecified) {...
            try {
                FileReader libcFile = new FileReader(libPath);
                Lexer scanner = new Lexer(libicFile);
                LibraryParser parser = new LibraryParser(scanner);
                ClassAST root = (ClassAST) parser.parse().value;
                System.out.println("parsed " + libPath + " successfully!");
            } catch (SyntaxError e) {
                System.out.print("Syntax error in " + libPath + " " + e);
            }
        }
        ...
    }
}
```

35

See you next week

36