Compiler Construction

Parsing II

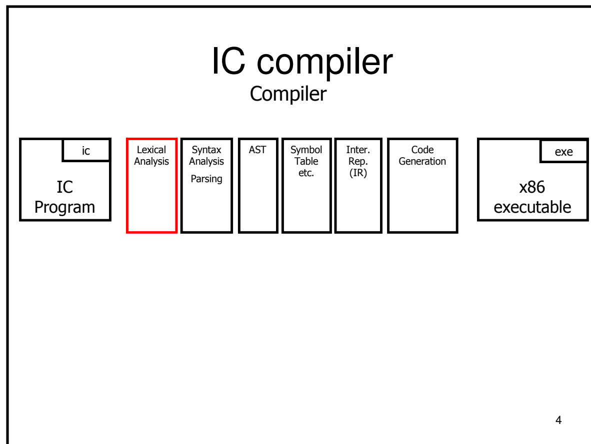
Ran Shaham and Ohad Shacham
School of Computer Science
Tel-Aviv University

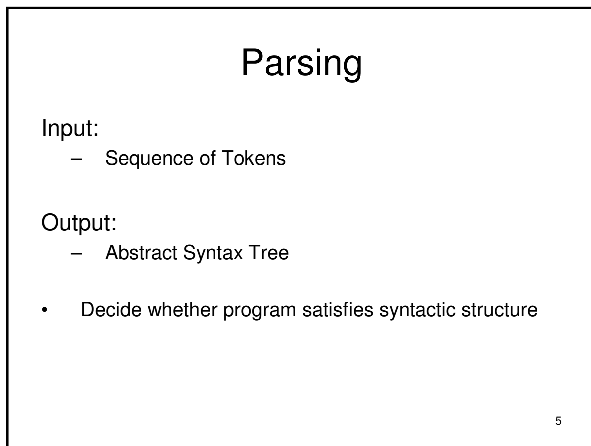
Administration

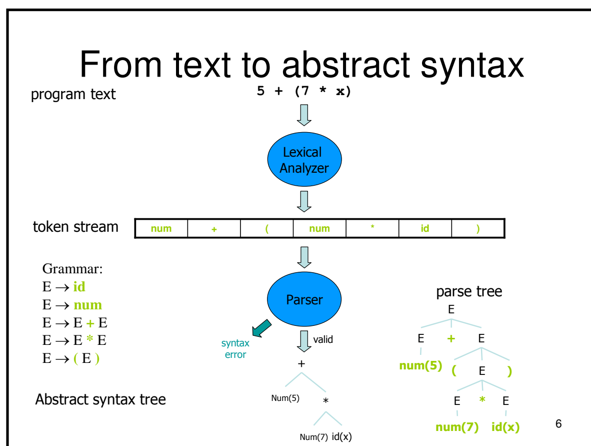
- Forum
  - https://forums.cs.tau.ac.il/viewforum.php?f=64
- Submit only source files
- Add +1 to yyline
- Please read IC Spec carefully
  - No --
  - No ++
  - Class identifier starts with Upper case letter
  - Other identifiers starts with lower case letter

PA1 submission

- Sources only
- According to the given hierarchy
- A brief, clear, and concise description of your code structure and testing strategy
- Put it in: ~/IC_COMPILER/PA1/
- Don't develop in this directory and try to compile using ant


IC compiler
Compiler
ic
IC Program
Lexical Analysis
Syntax Analysis
Parsing
AST
Symbol Table etc.
Inter. Rep. (IR)
Code Generation
exe
x86 executable
4


Parsing
Input:
– Sequence of Tokens
Output:
– Abstract Syntax Tree
• Decide whether program satisfies syntactic structure
5


From text to abstract syntax
program text
5 + (7 * x)
Lexical Analyzer
token stream
num + (num * id)
Grammar:
E → id
E → num
E → E + E
E → E * E
E → (E)
Parser
syntax error
valid
parse tree
E
E + E
num(5) (E)
E * E
num(7) id(x)
Abstract syntax tree
+
Num(5) *
Num(7) id(x)
6

Usage

- Syntax analysis
  - Checks the input syntax validity
- Semantic analysis
  - Checks the input meaning

7

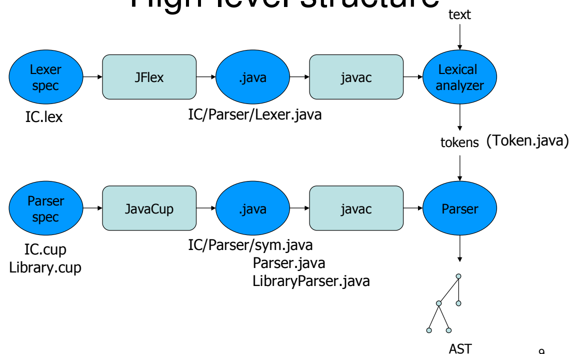
Expression calculator

```
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?

8

High-level structure

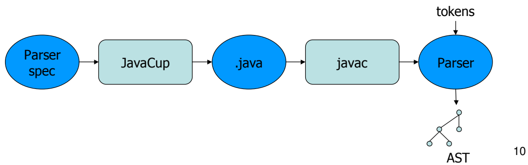


9

Cup

- Constructor of Useful Parsers

- Automatic LALR(1) parser generator
  - Input: cup spec file
  - Output: Syntax analyzer in Java



10

Expression calculator

```
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

11

Ambiguities

a * b + c

a + b + c

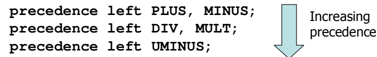
12

Expression calculator

```
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
   ;
```



Contextual precedence

13

Disambiguation

Each terminal assigned with precedence

- By default all terminals have lowest precedence
- User can assign his own precedence
  - `MINUS expr %prec UMINUS`
- CUP assigns each production a precedence
  - Precedence of last terminal in production
    - `expr MINUS expr`
  - User specified contextual precedence
    - `MINUS expr %prec UMINUS`

14

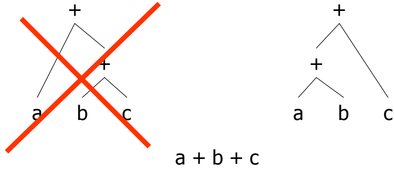
Disambiguation

- 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

15

Resolving ambiguity

```
precedence left PLUS
```



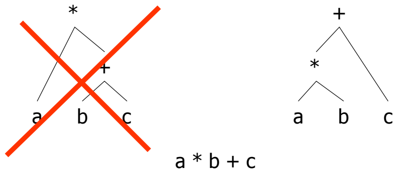
+
+
a b c

a + b + c

16

Resolving ambiguity

```
precedence left PLUS
precedence left MULT
```



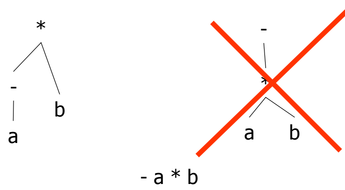
+
*
a b c

a * b + c

17

Resolving ambiguity

```
MINUS expr %prec UMINUS
```



-
*
a b

- a * b

18

Resolving ambiguity

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

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
   ;
```

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

Rule has precedence of UMINUS

19

More CUP directives

- **`precedence nonassoc NEQ`**
  - Non-associative operators: `< > == !=` etc.
  - `1<2<3` identified as an error (semantic error?)
  - 6 == 7 == 8 == 9
- **`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

20

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)`

21

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

22

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

23

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

24

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

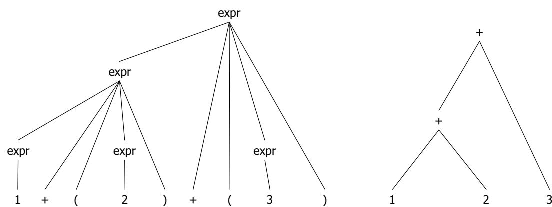
25

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

26

Parse tree vs. AST



27

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

28

AST construction

1 + (2) + (3)

expr + (2) + (3)

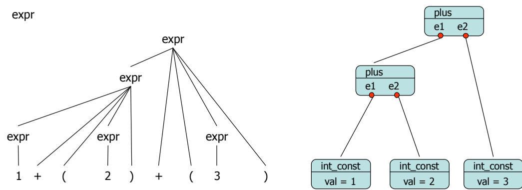
expr + (expr) + (3)

expr + (3)

expr + (expr)

expr

```
expr ::= expr:e1 PLUS expr:e2
     {: RESULT = new plus(e1,e2); :}
   | LPAREN expr:e RPAREN
     {: RESULT = e; :}
   | INT_CONST:i
     {: RESULT = new int_const(…, i); :}
```



29

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
   ;
```

30

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

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:

op type field of Expr

Alternative 2 class for each op:

PlusExpr

MinusExpr

MultExpr

DivExpr

UnaryMinusExpr

ValueExpr

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
        {: RESULT = new Expr(e1,e2,"PLUS"); :}
        | expr:e1 MINUS expr:e2
        {: RESULT = new Expr(e1,e2,"MINUS"); :}
        | expr:e1 MULT expr:e2
        {: RESULT = new Expr(e1,e2,"MULT"); :}
        | expr:e1 DIV expr:e2
        {: RESULT = new Expr(e1,e2,"DIV"); :}
        | MINUS expr:e1
        {: RESULT = new Expr(e1,"UMINUS"); :} %prec
   UNMINUS
        | LPAREN expr:e1 RPAREN
        {: RESULT = e1; :}
        | NUMBER:n
        {: RESULT = new Expr(n); :}
   ;
```

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;
   }
}
```

34

Computing meaning

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

35

PA2

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

36

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

37

PA2 – step 2

- 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 Almozlino
- Change CUP actions to construct AST nodes

38

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 libicFile = 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);
        }
      }
  ...
```

39