

Compiler Construction

Parsing

Rina Zviel-Girshin and Ohad Shacham
School of Computer Science
Tel-Aviv University

Administration

- Please use the forum for questions
<https://forums.cs.tau.ac.il/viewforum.php?f=70>
- Don't compile in the submission directory
- Check whether your group appears in the list
 - Send me an email if you can't find a team
 - Send me your team if you found one and didn't send an email
 - Please send Name, Id, nova Id, and leader

2

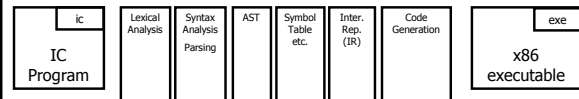
Complementary Class

- November 26 – Schreiber 07
 - 9:00 – 10:00
 - 13:00 – 14:00
- November 27 – Schreiber 07
 - 10:00 – 11:00

Does anyone plan to come on Friday?

3

IC compiler Compiler



4

Parsing

Input:

- Sequence of Tokens
- A context free grammar

Output:

- Abstract Syntax Tree
- Decide whether program satisfies syntactic structure

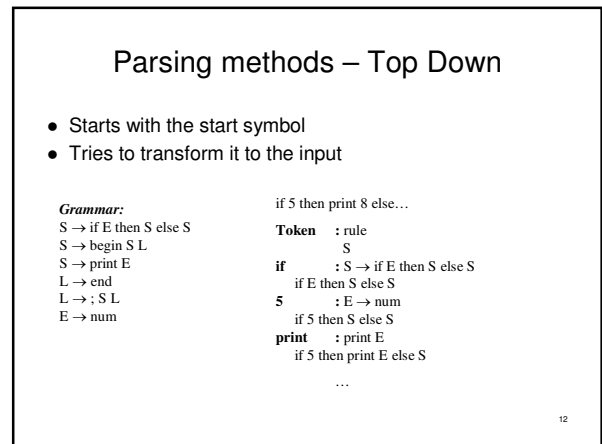
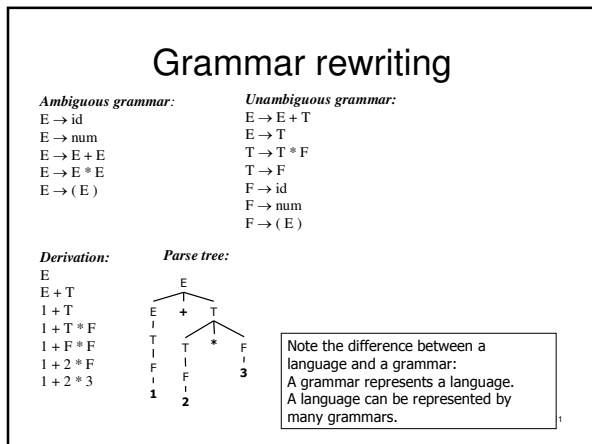
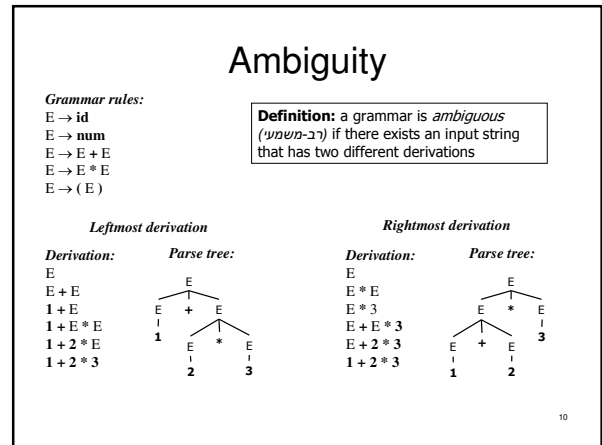
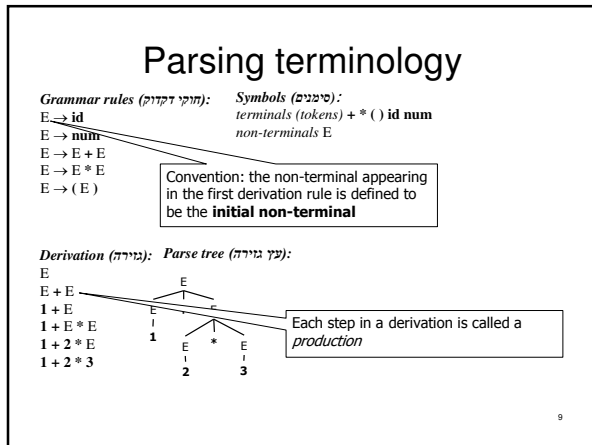
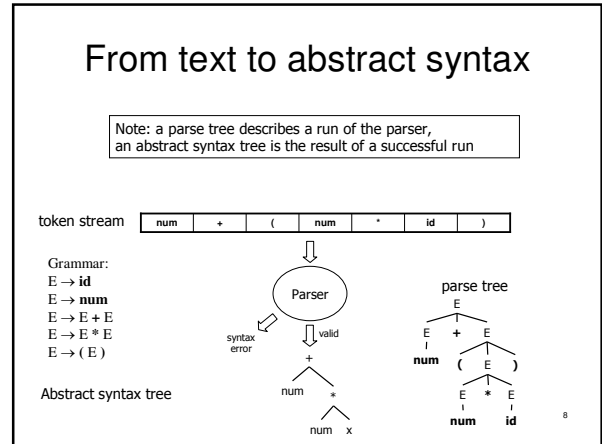
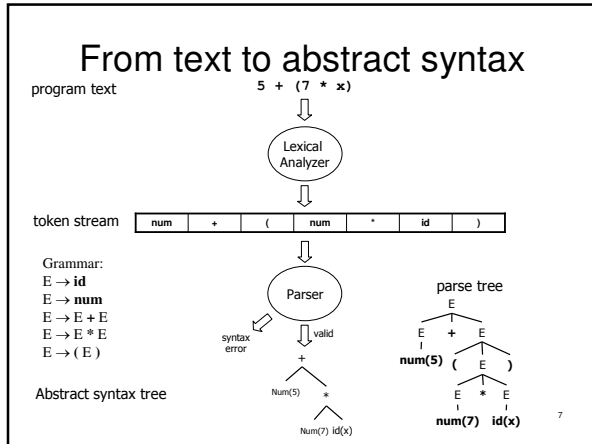
5

Parsing

- Context Free Grammars (CFG)
- Captures program structure (hierarchy)
- Employ formal theory results
- Automatically create "efficient" parsers

Grammar:
S → if E then S
 else S
S → print E
E → num

6



Parsing methods – Bottom Up

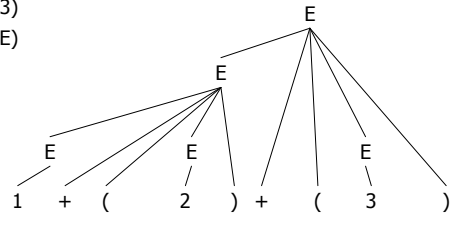
- Starts with the input
- Attempt to rewrite it to the start symbol
- Widely used in practice
- LR(0), SLR(1), LR(1), LALR(1)
 - We will focus only on the theory of LR(0)
- JavaCup implements LALR(1)

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

Bottom Up - problems

- Ambiguity

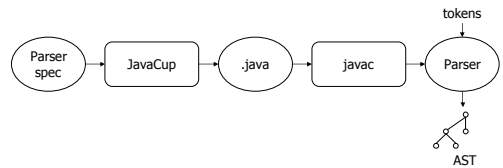
$E = E + E$
 $E = i$

1 + 2 + 3 -> (1 + 2) + 3 ?
 1 + 2 + 3 -> 1 + (2 + 3) ?

15

Cup

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



16

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
;

```

Is 2+3+4+5 a valid expression?

17

Ambiguities



$a * b + c$



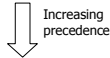
$a + b + c$

18

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

;

19

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
 - expr MINUS expr
 - User specified contextual precedence
 - MINUS expr %prec UMINUS

20

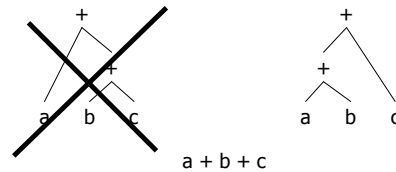
Disambiguation

- 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

21

Resolving ambiguity

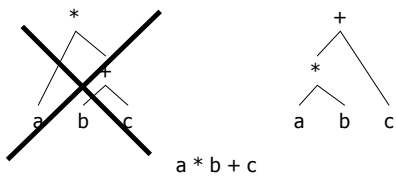
precedence left PLUS



22

Resolving ambiguity

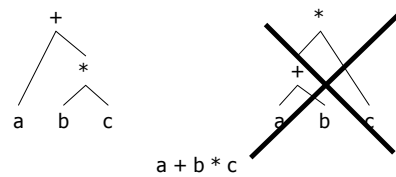
precedence left PLUS
precedence left MULT



23

Resolving ambiguity

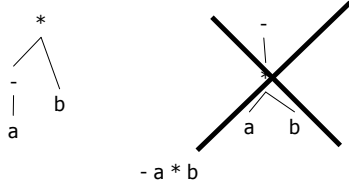
precedence left PLUS
precedence left MULT



24

Resolving ambiguity

precedence left PLUS
 precedence left MULT
 MINUS expr %prec UMINUS



25

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

26

More CUP directives

- **precedence nonassoc NEQ**
 - Non-associative operators: < > == != etc.
 - 1<2<3 identified as an 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

27

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

28

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

29

Assigning meaning

```
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:e1 RPAREN
      | NUMBER:n
      ;
{ : RESULT = new Integer(e1.intValue() + e2.intValue()); ; }
{ : RESULT = new Integer(e1.intValue() - e2.intValue()); ; }
{ : RESULT = new Integer(e1.intValue() * e2.intValue()); ; }
{ : RESULT = new Integer(e1.intValue() / e2.intValue()); ; }
{ : RESULT = new Integer(0 - e1.intValue()); ; } %prec UMINUS
{ : RESULT = e1; ; }
{ : RESULT = n; ; }
```

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

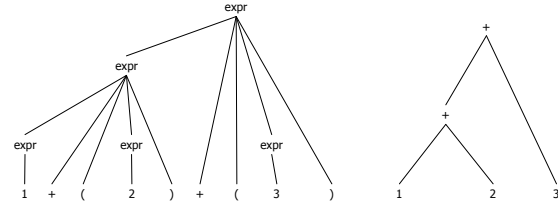
30

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

31

Parse tree vs. AST



32

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

33

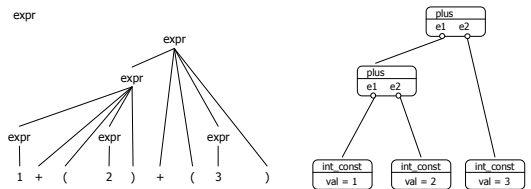
AST construction

```

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); :}
    
```



34

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
           ;
    
```

35

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

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 – next week

- 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;
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