CS 536 Announcements for Wednesday, February 22, 2023

Programming Assignment 2

- due Wednesday, February 22

Midterm 1

- Wednesday, March 1, 7:30 – 9 pm
- B10 Ingraham Hall
- bring your student ID

Last Time

- implementing ASTs

Today

- Java CUP

Next Time

- review for Midterm 1

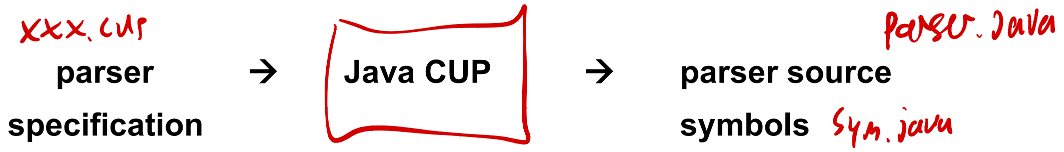
Parser generators

Tools that take an SDT spec and build an AST

- **YACC** Yet another C compiler - creates C parser
- **Java CUP** Constructor of useful parser - creates a parser in java, works with JLex

Conceptually similar to JLex:

- Input: language rules + actions
- Output: Java code



Java CUP

`parser.java`

- constructor takes argument of type `Yylex` ← from scanner
- `parse` method
  - if input correct, returns `Symbol` whose `value` field contains translation of root nonterm
  - if input incorrect, quits on first syntax error
- uses output of JLex ← in brevis.jlex
  - depends on scanner and `TokenVal` classes
  - `sym.java` defines the communication language = defines token names used by jlex & java cup
- uses definitions of AST classes (in `ast.java`)

Parts of Java CUP specification

Grammar rules with actions:

```
expr ::= INTLITERAL
       |   ID
       |   expr PLUS expr
       |   expr TIMES expr
       |   LPAREN expr RPAREN
       ;
```

← critical

Terminal and nonterminal declarations:

```
terminal     INTLITERAL;
terminal     ID;
terminal     PLUS;
terminal     TIMES;
terminal     LPAREN;
terminal     RPAREN;

non terminal expr;
```

Precedence and associativity declarations:

Order listed indicates prescedence from lowest - highest

```
precedence left PLUS;
precedence left TIMES;
```

↑ associativity

can indicate nonassoc less;

Java CUP Example

defined in ast.java

Assume:

- Java class ExpNode with subclasses IntLitNode, IdNode, PlusNode, TimesNode
- PlusNode and TimesNode each have two children
- IdNode has a String field (for the identifier)
- IntLitNode has an int field (for the integer value)
- INTLITERAL token is represented by IntLitTokenVal class and has field intVal
- ID token is represented by IdTokenVal class and has field idVal

Step 1: add types to terminals and nonterminals

```
/*
 * Terminal declarations
 */
terminal INTLITERAL;
terminal ID;
terminal PLUS;
terminal TIMES;
terminal LPAREN;
terminal RPAREN;
```

Need type if want to use value associated with token

IdTokenVal ID;

terminal IntLitTokenVal

from scanner (... .jlex)

```
/*
 * Nonterminal declarations
 */
non terminal expr;
```

Type required for all nonterms

nonterminal ExpNode expr;

from ast.java

Step 2: add precedences and associativities

```
/*
 * Precedence and associativity declarations
 */
precedence left PLUS;
precedence left TIMES;
```

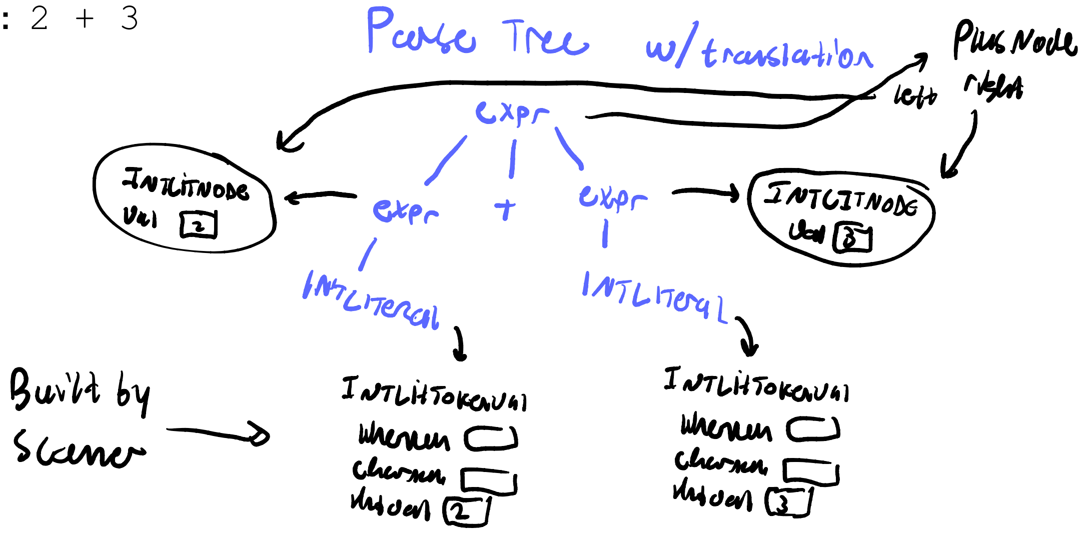
Java CUP Example (cont.)

Step 3: add actions to CFG rules

```
/*
 * Grammar rules with actions
 */
expr ::= INTLITERAL:i
           {:
           Result= new IntLitNode(i);

           :}  ←  these are subclasses
        |  ID
           {:


           :}
        |  expr:e1   PLUS   expr:e2
           {:
             Result = new PlusNode(e1,e2);

           :}
        |  expr      TIMES   expr
           {:
           Result= new TimesNode(e1,e2);
           :}
        |  LPAREN   expr:e    RPAREN
           {:
             result=e;

           :}
        ;
        Format!
        nonterm ::= rule1
                      {: //action for rule1)
                      Resut =... ;
                     :}
                    | rule2
                    {:
                      Resut = ...;
                    :}
                    :
                    ;
```

Java CUP Example (cont.)

Input: 2 + 3



Translating lists

Example

left recursive → idList → idList COMMA ID | ID

Left-recursion or right-recursion?

- for top-down parsers must use right recursion

left recursion: infinite loop!

- for Java CUP

↳ bottom up: left recursion good!

Example

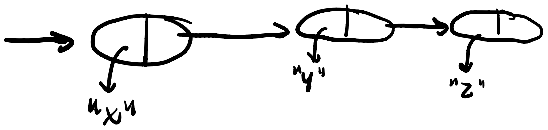
CFG: idList → idList COMMA ID | ID

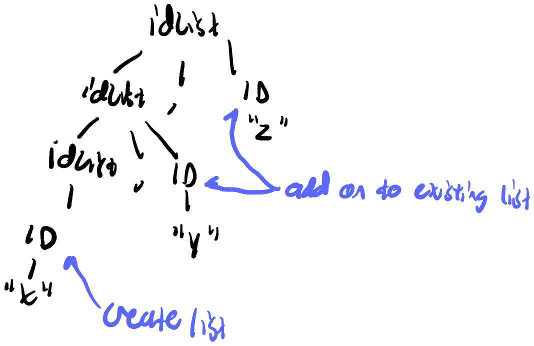
Goal: the translation of an `idList` is a `LinkedList` of `String`s

Example

Input: `x , y , z`

Output:





Example (cont.)

Java CUP specification for this syntax-directed translation

Terminal and nonterminal declarations: idList → idList comma ID

| ID

terminal IDtokenVal ID;

ID;

terminal comma

comma;

nonterminal LinkedList<String> idList;

Grammar rules and actions:

```
idList ::= idList:l      COMMA       ID:i
           {: l.addLast(i.idVal);
            Result=L;


           :}
        |  ID:i
           {: LinkedList<String> L= new LinkedList <String>();
            L.add(i.idVal);
         Result=L;



           :}
        ;
```

Handling unary minus

```
/*
 * precedences and associativities of operators
 */
precedence left PLUS, MINUS;
precedence left TIMES, DIVIDE;




/*
 * grammar rules
 */
exp  ::= . . .
     |    MINUS exp:e
       {: RESULT = new UnaryMinusNode(e);
       :}
     |    exp:e1 PLUS exp:e2
       {: RESULT = new PlusNode(e1, e2);
       :}
     |    exp:e1 MINUS exp:e2
       {: RESULT = new MinusNode(e1, e2);
       :}
        . . .
     ;
```