CS 536 Announcements for Wednesday, February 21, 2024

Programming Assignment 2
- due Tuesday, February 20 – accepted until 11:59 pm Thursday
- see late policy on course website

Midterm 1
- Thursday, February 29, 7:30 – 9 pm
- S429 Chemistry
- bring your student ID

Last Time
- implementing ASTs

Today
- Java CUP

Next Week
- review for Midterm 1
- parsing

Parser generators
Tools that take an SDT spec and build an AST
- YACC: Yet Another C Compiler - creates a parser in C
- Java CUP: Constructor of Useful Parsers - creates a parser in Java

Conceptually similar to JLex:
- Input: language rules + actions
- Output: Java code

Week 5 (M)  Page 1
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
  - depends on scanner and `TokenVal` classes
  - `sym.java` defines the communication language
- uses definitions of AST classes (in `ast.java`)

**Parts of Java CUP specification**

**Grammar rules with actions:**

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

Terminal and nonterminal declarations:

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

**Precedence and associativity declarations:**

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

Order (in JavaCUP) indicates precedence:

- `<` indicates low precedence
- `>` indicates high precedence
- `=` indicates same precedence

Precedence can do:

- **nonassoc** LESS, GREATER.
Java CUP Example

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

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

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

Step 2: add precedences and associativities

```java
/*
 * 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
{:
  RESULT = new IntLitNode(c, intVal);

| ID : i
{:
  RESULT = new IdNode(i, idVal);

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

| expr : e1 TIMES expr : e2
{:
  RESULT = new TimesNode(e1, e2);

| LPAREN expr : e RPAREN
{:
  RESULT = e;

;}

General format

nonterm ::= rule1
  S: // action for rule1
  RESULT = ... S

  S: // action for rule2
  RESULT = ... S

  ...
Java CUP Example (cont.)

Input: $2 + 3$

Parse tree w/ translation

Built by parser

expr

expr + expr

PlusNode left right

IntLitNode val 2

IntLitNode val 3

INTLITERAL

INTLITERAL

Built by scanner

IntLit Token Val

IntVal 2

IntVal 3
Translating lists

Example

idList → idList COMMA ID | ID

Left-recursion or right-recursion?

- For top-down parsers must use right recursion.
  - Left-recursion leads to infinite loop.
- For Java CUP use left recursion.

Example

CFG:    idList → idList COMMA ID | ID

Goal: the translation of an idList is a LinkedList of Strings

Example

Input:  x , y , z
Output:   

```
add on to existing list
create list
```

```
Example (cont.)

Java CUP specification for this syntax-directed translation

Terminal and nonterminal declarations:

\[
\text{terminal } \text{IdTokenVal} \quad 10_j \quad \text{terminal } \text{ComMA}_j \\
\text{non terminal } \text{LinkedList(String)} \quad \text{idList}_j
\]

Grammar rules and actions:

\[
idList ::= \text{idList}_j \quad \text{COMMA} \quad \text{ID} \quad \text{?} \\
\{
\text{L.addLast( \text{i.idVal}_j)} \\
\text{RESULT= L}_j \\
\} \\
| \text{ID} \quad \text{?} \\
\{
\text{LinkedList(String)} \quad \text{L} = \text{new LinkedList(String)}()_j \\
\text{L.add( \text{i.idVal}_j)} \\
\text{RESULT= L}_j \\
\} \\
; \\
\]

Handling unary minus

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

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