CS 536 Announcements for Tuesday, March 1, 2022

Midterm 1
- Wednesday, March 2, 7:30 – 9 pm
- B102 Van Vleck
- bring your student ID

Last Time
- Java CUP
- midterm 1 info

Today
- wrap up Java CUP
  - translating lists
  - handling unary minus
- review / midterm 1 questions

Next Time
- no lecture Thursday, March 3

---

Java CUP

\[
\text{parser.cup} \rightarrow \text{Java CUP} \rightarrow \text{parser source}
\]

\text{parser.java}

- constructor takes argument of type Yylex
- parse method
  - returns Symbol whose value field contains translation of root nonterm
- uses output of JLex
  - depends on scanner and TokenVal classes
  - \text{sym.java} defines the communication language
- uses definitions of AST classes (in \text{ast.java})
Parts of Java CUP specification

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

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

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

/*
 * Grammar rules with actions
 */
expr ::=   INTLITERAL:i
{:  RESULT = new IntLitNode(i.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;
:}
Translating lists

Example

idList \rightarrow idList \text{ COMMA ID} \mid \text{ 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
  bottom-up parser

Example

CFG: \hspace{1cm} idList \rightarrow idList \text{ COMMA ID} \mid \text{ ID}

Goal: the translation of an idList is a \text{ LinkedList} of Strings

Example

Input: \hspace{1cm} x, y, z

Output:

\hspace{1cm} \text{ add on to list}
Java CUP specification for this syntax-directed translation

Terminal and nonterminal declarations:

Terminal
- `IdTokenVal ID_j`
- `Comma`

Non terminal
- `LinkedList <String> idList_{j}`

Grammar rules and actions:

```
 idList : @ = idList : L , COMMA ID : i
   e = L.addLast ( i.idVal ) ;
   RESULT = L_{j} ;
   : 3

  | ID : i
   e = L = new LinkedList <String> () ;
    L.add ( i.idVal ) ;
   RESULT = L_{j} ;
   : 3
```
Handling unary minus

/* precedences and associativities of operators */
precedence left PLUS, MINUS;  \underbrace{\text{binary minus has lowest precedence}}
precedence left TIMES, DIVIDE;
precedence nonassoc UMINUS; \underbrace{\text{"phony" token (never returned by scanner)}}
precedence nonassoc UMINUS; \underbrace{\text{Unary minus has highest precedence}}

/* 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);
        :}
    |   . . .
    ;

Precedence of a rule is that of the last token of the rule, unless assigned a specific precedence via Toproc
Parsing: two approaches

Top-down "goal driven"
start at start non-terminal
grow parse tree downward until have entire token sequence

Example
CFG:
expr → expr + term | term
term → term * ID | ID

Derive 1D + 1D

Bottom-up "data driven"
start with terminal sequence
generate ever larger subtrees until get to start non-terminal