CS 536 Announcements for Thursday, February 24, 2022

Programming Assignment 2
- due Friday, February 25

Last Time
- syntax-directed translation
- abstract syntax trees
- implementing ASTs

Today
- Java CUP
- midterm 1 info

Next Time
- wrap up Java CUP
- review

Parser generators
Tools that take an SDT spec and build an AST
- YACC
- Java CUP

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

```
parser Specification -> Java CUP -> parser.java
                       -> sym.java
```
Java CUP

**parser.java**
- constructor takes argument of type Yylex
- 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

**Parts of Java CUP specification**

Grammar rules with actions:
```java
expr ::= INTLITERAL
  | ID
  | expr PLUS expr
  | expr TIMES expr
  | LPAREN expr RPAREN
```

Terminal and nonterminal declarations:
```java
terminal    INTLITERAL;
terminal    ID;
terminal    PLUS;
terminal    TIMES;
terminal    LPAREN;
terminal    RPAREN;
non terminal expr;
```

Precedence and associativity declarations:
```java
precedence left PLUS;
precedence left TIMES;
```
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 INTLITERAL;
terminal ID;
terminal PLUS;
terminal TIMES;
terminal LPAREN;
terminal RPAREN;

non terminal expr;
```
Java CUP Example (cont.)

Step 2: add actions to CFG rules

exr  ::=  INTLITERAL
          {:

          :
          |  ID
          {:

          :
          |  expr    PLUS    expr
          {:

          :
          |  expr    TIMES   expr
          {:

          :
          |  LPAREN   expr    RPAREN
          {:

          :
          ;
Input: 2 + 3
Midterm 1
Wednesday, March 2, 7:30 – 9 pm
B102 Van Vleck

Scanning
• general : what does a scanner do; how does it fit into the design of a compiler
• underlying model : FSMs, DFAs vs NFAs, translating NFA → DFA
• specification of a scanner : regular expressions, JLex specifications
  • you do not need to know all of JLex's special characters

Context-Free Grammars
• specification of a language's syntax via a CFG
• derivations (left-most, right-most)
• parse trees
• expression grammars (precedence, associativity)
• list grammars
• ambiguous grammars
• recursive grammar (left recursive, right recursive)

Syntax-Directed Translation
• "plain" translations
  • writing rules of the form "s1.trans ="
  • being able to define translations of any types (integer, AST nodes, etc.)
• Java CUP translations
  • using :xx to name the translation associated with a symbol
  • defining translations by assigning to RESULT

Watch Piazza for info about
• additional homeworks posted on CFGs & SDTs
• sample midterm
• more details about topics