# CUP

## Lecture 12
## Section 4.9, CUP Manual

Robb T. Koether

Hampden-Sydney College

Fri, Feb 20, 2009

# Outline

# The CUP Parser Generator



- The CUP parser generator creates Java source code for an LR parser that uses the LALR algorithm.
  - CUP = Common Useful Parsers.
  - LALR = Look-Ahead LR.
- LALR is similar to LR(1), whose tables are much larger than the LR(0) tables.
- LALR tables are compact, the same size as the LR(0) tables.
- A LALR grammar is an LR(1) grammar whose LR(1) tables may compressed without creating conflicts.

# CUP Input and Output



- CUP reads a description of the grammar from a .cup file.
- From that description, it produces two files: parser.java and sym.java.
- The file parser.java contains Java source code for the parser.
- Using the -parser command-line argument, we can rename the parser class.

# CUP Input and Output



- The `sym` class is a class that defines the terminals of the grammar as integer constants.
- It is similar to the `Token` class that we have been using.
- For example, the '`+`' token is now `sym.PLUS` and it is given a numerical value.
- Using the `-symbols` command-line argument, we can rename the `sym` class.

# CUP Usage



- Typically invocations of CUP:

```
$ java java_cup.Main < grammar.cup
$ java java_cup.Main -parser MyParser < grammar.cup
$ java java_cup.Main -symbols Token < grammar.cup
```

# CUP and JLex



- CUP may be used with or without JLex.
- In Lab 6, we will see exactly how to make the two programs work together; it is easy.
- It is a little more involved to interface CUP with a hand-written lexer such as MyLexer.
- We will not take the time to do that.

# CUP Input



## Parser Code

```
parser code
{:
    public static void main(String args[]) throws Exception
    {
        parser myParser = new parser(new Yylex(System.in));
    }
:}
```

- The CUP file begins with any parser code that the programmer wants to add to the parser class.
- Type the directive `parser code`, and then include the code within the delimiters `{:` and `:}`.

# CUP Terminals



## Terminals

```
terminal NUM, PLUS, TIMES;
terminal String ID;
```

- List the terminals of the grammar.
- Type `terminal`, followed by a list of symbolic names for the terminals (e.g., `ID`, `NUM`, `PLUS`).
- These are used to create the `sym` class.
- `JLex` will use these for its tokens.
- Terminals may have an associated data type. For example, `ID` may have the type `String`.

# CUP Nonterminals



## Nonterminals

```
nonterminal func, glbl, prog;
nonterminal Integer type;
nonterminal LinkedList stmt;
```

- List the nonterminals of the grammar.
- Type `nonterminal`, followed by a list of symbolic names for the nonterminals (e.g., func, type, stmt).
- Nonterminals may have an associated data type. For example, `type` may have the type `Integer`.

# CUP Precedence Rules



## Precedence Rules

```
precedence right ASSIGN;
precedence left PLUS, MINUS;
precedence left TIMES, DIVIDE;
```

- For each precedence rule,
  - Type `precedence`,
  - State the associativity (left, right, or nonassoc),
  - Write the names of the terminals that represent the operators at that precedence level.
- The precedence rules are listed from lowest to highest precedence.

# CUP Productions



## Productions

```
stmt ::= RETURN expr:e SEMI
         {: RESULT = SemanticAction.ret(e); :}
expr ::= lval:v ASSIGN expr:e
         {: RESULT = SemanticAction.assign(v, e); :}
```

- The format of a production is

  *nonterm* `::=` *rightside* `{:` *action* `:};`
  where
  - *rightside* is a string of grammar symbols.
  - *action* is zero or more Java statements.
- If there is more than one production for a nonterminal, they are separated by `|`.

# The Start Symbol



## The Start Symbol

```
start with prog;
```

- The start symbol is assumed to be the nonterminal on the left side of the first production.
- Otherwise, specify it using the `start with` directive.

# The Action and Goto Tables



- CUP produces the action, reduce, and goto tables.
- We can use the command-line arguments

  `-dump_grammar`

  `-dump_tables`

  to see the numbered grammar symbols and productions and the action and goto tables.

# Example



**Example (CUP and JLex)**

Download and run the `CUP-JLex String Demo`.

# Example



## Example (The CUP Productions)

0. $stmts \longrightarrow stmts\ stmt$
1. $S' \longrightarrow stmts\ \$$
2. $stmts \longrightarrow \varepsilon$
3. $A \longrightarrow \varepsilon$
4. $stmt \longrightarrow expr\ A\ \mathbf{;}$
5. $expr \longrightarrow >\ expr$
6. $expr \longrightarrow <\ expr$
7. $expr \longrightarrow expr\ +\ expr$
8. $expr \longrightarrow \mathbf{num}\ *\ expr$
9. $expr \longrightarrow (\ expr\ )$
10. $expr \longrightarrow \mathbf{string}$

# Example

## Example

### The CUP Action Table

| | $ | error | NUM | STR | > | < | + | * | ( | ) | ; |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | r2 | | r2 | r2 | r2 | r2 | | | r2 | | |
| 1 | s9 | | s2 | s3 | s5 | s4 | | | s7 | | |
| 2 | | | | | | | | s18 | | | |
| 3 | | | | | | | r10 | | | r10 | r10 |
| 4 | | | s2 | s3 | s5 | s4 | | | s7 | | |
| 5 | | | s2 | s3 | s5 | s4 | | | s7 | | |
| 6 | r0 | | r0 | r0 | r0 | r0 | | | r0 | | |
| 7 | | | s2 | s3 | s5 | s4 | | | s7 | | |
| 8 | | | | | | | s11 | | | | r3 |
| 9 | r1 | | | | | | | | | | |
| 10 | | | | | | | | | | | s13 |
| 11 | | | s2 | s3 | s5 | s4 | | | s7 | | |
| 12 | | | | | | | r7 | | | r7 | r7 |
| 13 | r4 | | r4 | r4 | r4 | r4 | | | r4 | | |
| 14 | | | | | | | s11 | | | s15 | |
| 15 | | | | | | | r9 | | | r9 | r9 |
| 16 | | | | | | | r5 | | | r5 | r5 |
| 17 | | | | | | | r6 | | | r6 | r6 |
| 18 | | | s2 | s3 | s5 | s4 | | | s7 | | |
| 19 | | | | | | | r8 | | | r8 | r8 |

# Example

## Example

### The CUP Goto Table

| | $expr$ | $stmt$ | $stmts$ | NT$0 |
|---|---|---|---|---|
| 0 | | | 1 | |
| 1 | 8 | 6 | | |
| 2 | | | | |
| 3 | | | | |
| 4 | 17 | | | |
| 5 | 16 | | | |
| 6 | | | | |
| 7 | 14 | | | |
| 8 | | | | 10 |
| 9 | | | | |
| 10 | | | | |
| 11 | 12 | | | |
| 12 | | | | |
| 13 | | | | |
| 14 | | | | |
| 15 | | | | |
| 16 | | | | |
| 17 | | | | |
| 18 | 19 | | | |
| 19 | | | | |

# Assignment



## Homework

- The grammar

$$R \quad \rightarrow \quad R \cup R \mid RR \mid R^* \mid (R) \mid \mathbf{a} \mid \mathbf{b}$$

  generates all regular expressions on the alphabet $\Sigma = \{\mathbf{a}, \mathbf{b}\}$.
- Create a `JLex` file and a `CUP` file that will parse the regular expressions.
- Let the actions be simply to display the productions that were matched.