Class 7

Important Terminology

Derivation –

Leftmost versus Rightmost derivation -

Sentential Form –

Sentence –

Parse Tree –

Concrete syntax Tree –

Abstract syntax Tree –

Ambiguous grammar -

The Chomsky Hierarchy
By Norman Chomsky 1959

Four Main Types of Grammars based on their form:

Type 0 – unrestricted grammars

Type 1 – context-sensitive grammars

Type 2 – context-free grammars

Type 3 – regular grammars

The MeggyJava Syntax

1. How is it different from Java?
2. What is surprising to you?
3. What is not accepted that you would like to see in a language?

Parser Generation: JavaCup Spec Formats

- Package & import declarations
- User code components (linking with the lexer)
- Symbols (terminal & non terminal) lists Precedence declaration
- Grammar (context-free)

Package & import declarations

```
package miny_pascal; import

java_cup.runtime.*;
import java.io.FileInputStream;
import java.io.InputStream;
```

User code components

```
/* Preliminaries to set up and use the scanner. */
parser code
{:
        public Node root = null;
        public static parser getParser(String       throws Exception {
               pPath) InputStream is = null;
               is = new FileInputStream(pPath);
               return new parser(new Ylex(is));
        }
        public Node getTree() throws Exception { if (root ==
        null) {
                              this.parse();
        }
        return root;
        }
        public static void main(String args[]) throws Exception
        { new parser(new Ylex(System.in)).parse();
        }
        :}
```

Terminals & non terminals

```
/* Terminals (tokens returned by the scanner). */
terminal PROGRAM, BEGIN, END, DECLARE, PROCEDURE, FUNCTION, …
terminal BOOLEAN, ARRAY, OF, ASSIGN, LC, RC, IF,       ELSE, …
terminal THEN, READ, WRITE, TRUE, FALSE, ADD, MIN,     GOTO;
terminal MUL, DIV, MOD, LES, LEQ, EQU, NEQ, GRE, GEQ,
terminal AND, OR; NOT, CASE, FOR, FIN, IDENTICAL,      NEW;
         FROM, BY, TO,
terminal UMIN, COLON, SEMI, LPAR, RPAR, LPAR_SQ, RPAR_SQ, DOT, COMMA, PTR;

/* Terminals with attached values */
terminal Integer INTCONST;
terminal String IDE;
terminal Double REALCONST;
terminal String STRING;

/* Non terminals */
non      Node var, assign, program, stat_seq, loop_stat, case_stat, …
terminal Node expr, atom, block, stat, nonlable_stat, cond_stat, case,
non      Node … var_decl, type, simple_type, array_type, record_type, …
terminal Node record_list, dim, dim_list, proc_decl, formal_list, …
non      Node inout_stat, new_stat;
terminal
non
terminal
non
terminal
```

Precedence declaration

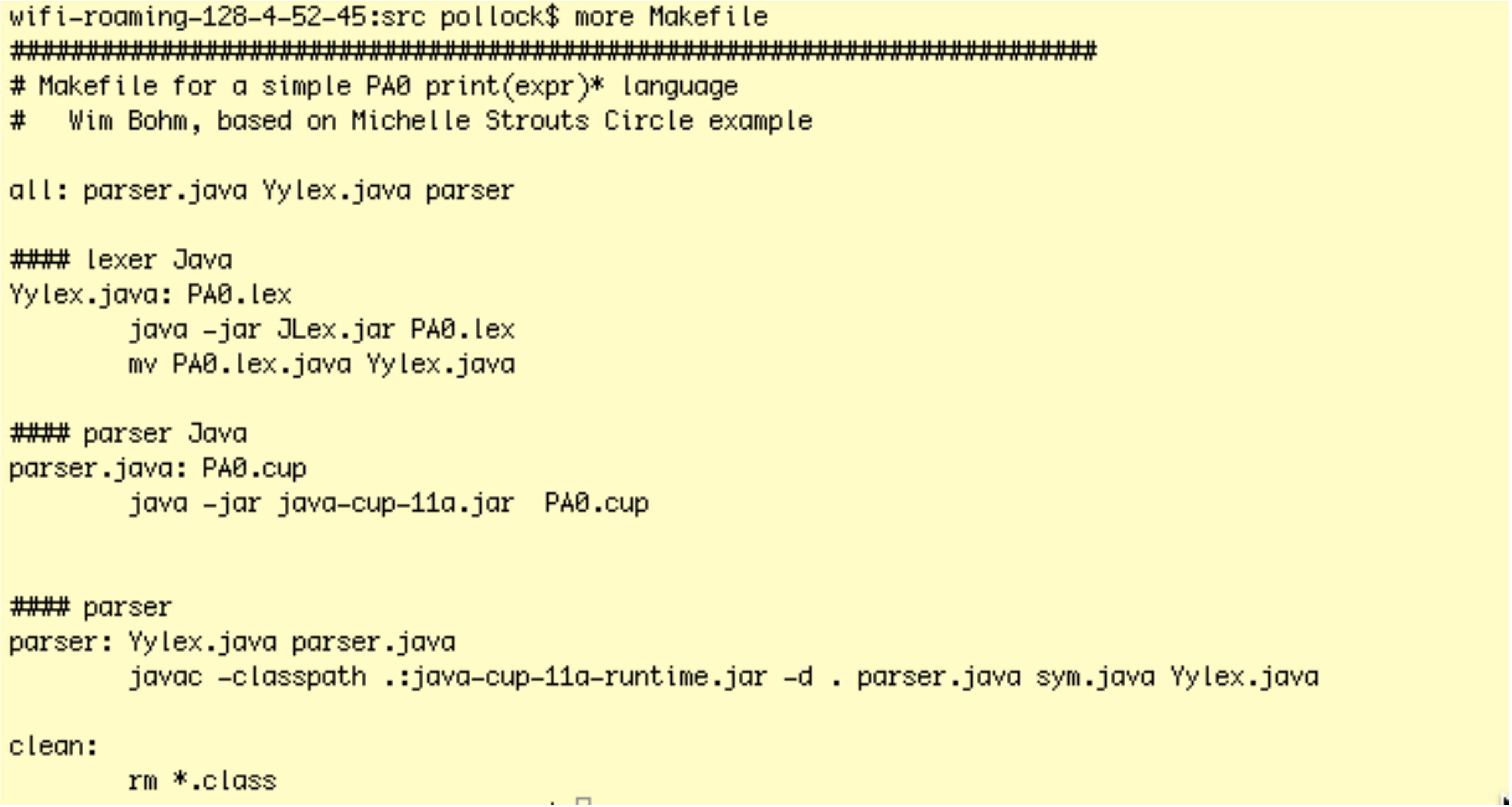
```
/* Precedence List */
precedence nonassoc LES, LEQ, EQU, NEQ, GRE,
GEQ; precedence left ADD, MIN, OR;
precedence left MUL, DIV, AND, MOD; precedence
left UMIN;
precedence right NOT;
precedence right DOT; precedence right PTR;;
```

Grammar (context-free)

```
/* Grammar */
start with
program;
program ::= PROGRAM IDE:n block:b    {: RESULT = new Program(b,n);
                                          parser.root=RESULT; :}
                                          ;
block ::= LC stat_seq:s RC                  {: RESULT = new Block(s); :}
          | decl_list:d LC stat_seq:s RC    {: RESULT = new Block(d,s); :} ;

decl_list ::= decl:d                {: RESULT = new DeclarationList(d); :}
              | decl:d decl_list:dl {: RESULT = new DeclarationList(dl,d); :} ;

decl ::= var_decl:vd                          {: RESULT = vd; :}
                  | proc_decl:pd                      {: RESULT = pd; :}
                  | func_decl:fd                      {: RESULT = fd; :} ;

…


assign ::= var:v ASSIGN expr:e      {: RESULT = new Assign(e,v); :} ;

cond_stat ::= IF expr:e THEN stat_seq:ss FI   {: RESULT = new
                                           ConditionalStatement(e,ss); :}
```

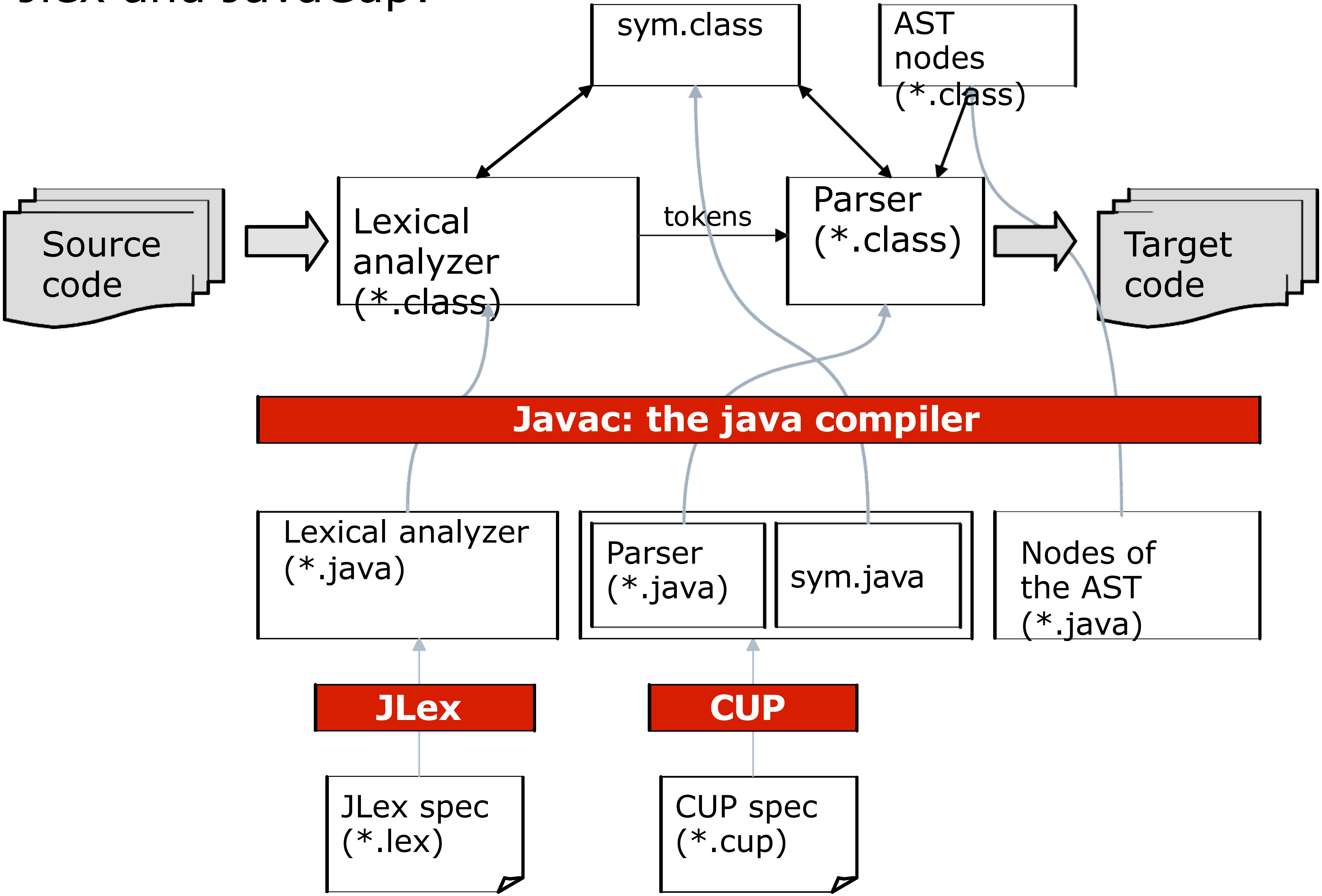
Note the labels on symbols in the productions. Refer to values on parse stack.

A Makefile using Jlex and JavaCup



```
wifi-roaming-128-4-52-45:src pollock$ more Makefile
#########################################################################
# Makefile for a simple PA0 print(expr)* language
#   Wim Bohm, based on Michelle Strouts Circle example

all: parser.java Yylex.java parser

#### lexer Java
Yylex.java: PA0.lex
        java -jar JLex.jar PA0.lex
        mv PA0.lex.java Yylex.java

#### parser Java
parser.java: PA0.cup
        java -jar java-cup-11a.jar  PA0.cup


#### parser
parser: Yylex.java parser.java
        javac -classpath .:java-cup-11a-runtime.jar -d . parser.java sym.java Yylex.java

clean:
        rm *.class
```

Interoperability of Jlex and JavaCup:



Building an Interpreter with jLex and JavaCup

The Source Language to be Interpreted:

```
 program ::= stmts | ε
   stmts   ::= stmts stmt | stmt
   stmt    ::= PRINT exp SEMI
   exp     ::= exp TIMES exp | exp PLUS exp | exp MINUS exp | NUMBER
```

With tokens and lexemes:

  Keywords: print ; * + -

  Number: [0-9]+

Some example programs:

The (incomplete) jLex specification

```
import java_cup.runtime.Symbol;
%%
%cup

%eofval{
  return new Symbol(sym.EOF, null);
%eofval}

%%
";" { return new Symbol(sym.SEMI, null); }
"*" { return new Symbol(sym.TIMES, null); }
"print" { return new Symbol(sym.PRINT, null); }
[ \t\r\n\f] { /* ignore white space. */ }
[0-9]+ {return new Symbol(sym.NUMBER, new Integer(yytext())); }
. { System.err.println("Illegal character: "+yytext()); }
```

To do: Update this so the scanner produces tokens for PLUS and MINUS

The (incomplete) JavaCup specification

```
import java_cup.runtime.*;
import java.io.FileInputStream;

parser code {:
    public static void main(String args[]) throws Exception {
        new parser(new Yylex(new FileInputStream(args[0]))).parse();
    }
:}

terminal PRINT;
terminal Integer NUMBER;
terminal PLUS, MINUS, TIMES;
terminal SEMI;

non terminal program;
non terminal stmts;
non terminal stmt;
non terminal Integer exp;

precedence left PLUS, MINUS;
precedence left TIMES;
```

The (incomplete) JavaCup spec continued

```
program ::=
        stmts
    |   /* Empty */
        ;

stmts   ::=
        stmts stmt
    |   stmt
    ;

stmt    ::=
        PRINT exp:e SEMI
        {: System.out.println("exp val"); :}

    ;

exp ::=
        exp:a TIMES:op exp:b        {: RESULT = new Integer(0); :}
    |   NUMBER:n                   {: RESULT = new Integer(0); :}

    ;
```

What happens for the program:

```
print 5;
print 6*7;
```

The (incomplete) JavaCup spec continued

```
program ::=
        stmts
    |   /* Empty */
        ;

stmts   ::=
        stmts stmt
    |   stmt
    ;

stmt    ::=
        PRINT exp:e SEMI
        {: System.out.println("exp val"); :}

    ;

exp ::=
        exp:a TIMES:op exp:b          {: RESULT = new Integer(0); :}
    |   NUMBER:n                    {: RESULT = new Integer(0); :}

    ;
```

To do: Update this so that:

- When print statement is interpreted, it outputs the value of the expression
- Number, times, plus, and minus parse and evaluate correctly

The generated sym.java file

```
/** CUP generated class containing
symbol constants. */
public class sym {
  /* terminals */
  public static final int PRINT = 2;
  public static final int error = 1;
  public static final int PLUS = 4;
  public static final int NUMBER = 3;
  public static final int SEMI = 7;
  public static final int MINUS = 5;
  public static final int TIMES = 6;
  public static final int EOF = 0;
}
```

PA2 Assignment Overview

On to how the parser actually works under the hood…