ANTLR

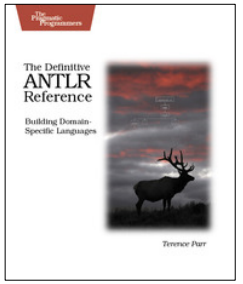
A Lexer/Parser/TreeWalker Generator

Monday, December 10, 2012

CS235 Languages and Automata

Department of Computer Science
Wellesley College

What is ANTLR?



- ANother Tool for Language Recognition (http://www.antlr.org/)
- Developed by Terence Parr @ University of San Francisco
- Automatically constructs lexers, parsers, and tree walkers from regexp, grammar, and tree-walking specifications
- Can create readable code for these in many target languages: Java, C/C++, JavaScript, Python, PHP, Ruby, etc.
- Uses LL(*) technology to create predictive (recursive descent) parsers with arbitrary lookahead.
- ANTLRWorks GUI development for ANTLR facilitates creation and testing of grammars.

Specifying Tokens with Lexer Rules

Tokens are specified by lexer rules of the form:

TOKEN_NAME : *regular-expression* ;

All upper case

```
// Lexer Rules for Intex
ADD    :    '+';
SUB    :    '-';
MUL    :    '*';
DIV    :    '/';
LPAREN :    '(';
RPAREN :    ')';
INT :  '0'..'9'+ ;
WS  :  ( ' ' | '\t' | '\r' | '\n') {$channel=HIDDEN;};
```

Unwanted tokens can go on hidden channel

Intex Token Example

```
(235+ 16    )
```

LPAREN INT ADD WS INT WS RPAREN

```
// Lexer Rules for Intex
ADD    :    '+';
SUB    :    '-';
MUL    :    '*';
DIV    :    '/';
LPAREN :    '(';
RPAREN :    ')';
INT :  '0'..'9'+ ;
WS  :  ( ' ' | '\t' | '\r' | '\n')+ {$channel=HIDDEN;};
```

Specifying ParseTrees with Parser Rules

Concrete parse trees are specified by parser rules of the form

variable : RHS_1 | RHS_2 | ... | RHS_n ;

All lower case

where RHS_1 is a sequence of tokens and variables

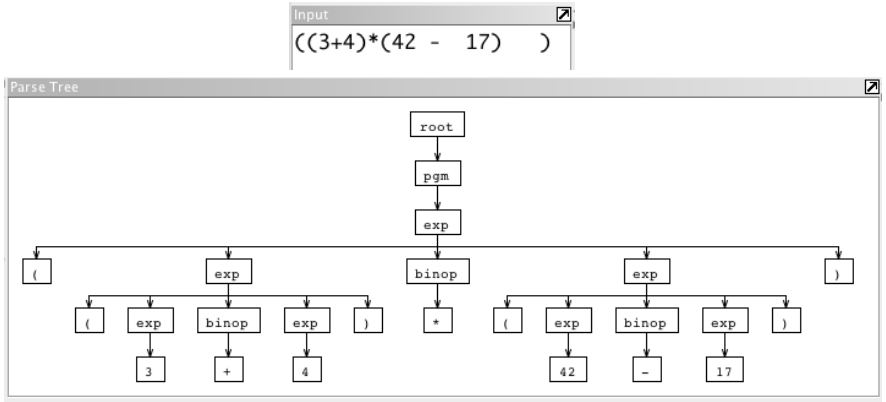
```
// Parser Rules for Intex
pgm : exp EOF ;

exp : LPAREN exp binop exp RPAREN
    | INT ;

binop : ADD | SUB | MUL | DIV ;
```


ANTLRWorks Parse Tree

In ANTLRWorks GUI, Debug>Run allows us to specify input and interactively view construction of concrete parse tree.

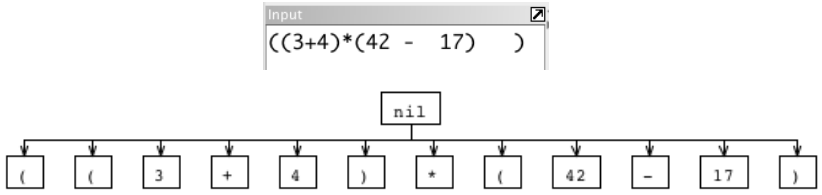


Default ASTs

In grammar (.g) file, can specify that ASTs should be generated.

```
options {
  language = Java; // ANTLR will generate java lexer and parser
  output = AST; // generated parser creates abstract syntax tree
  ASTLabelType=CommonTree; // use default ANTLR trees
}
```

By default, AST will contain flat list of all tokens rooted at nil node:



User-Specified ASTs

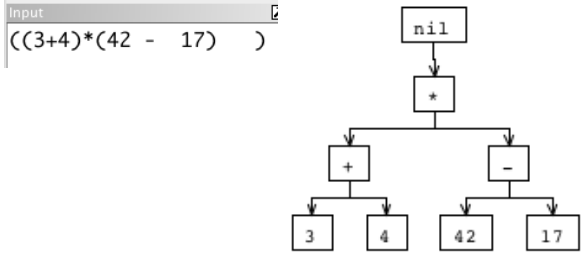
In parser rules, can specify how AST nodes should be constructed.

```
// Parser Rules for Intex with AST spec
pgm : exp EOF  ->  exp  ;

exp : LPAREN exp binop exp RPAREN  ->  ^(binop exp exp)

    | INT  ->  INT ;

binop : ADD | SUB | MUL | DIV ;
```



Full ANTLR IntexParens.g file, Part 1

```
grammar IntexParens;

options {
    language = Java; // ANTLR will generate Java lexer and parser
    output = AST; // generated parser creates abstract syntax tree
    ASTLabelType=CommonTree; // use default ANTLR trees
}

@header{
import org.antlr.runtime.debug.*;
import java.util.HashMap;
import org.antlr.stringtemplate.*;
}

@members {
  public static void main(String[] args) throws Exception {
    IntexParensLexer lex = new IntexParensLexer(new ANTLRInputStream(System.in));
    CommonTokenStream tokens = new CommonTokenStream(lex);
    IntexParensParser parser = new IntexParensParser(tokens);
    CommonTree tree = (CommonTree) parser.pgm().getTree();
  }
}
```


Full ANTLR IntexParens.g file, Part 2

```
// Parser Rules

pgm : exp EOF -> exp ;

exp : LPAREN exp binop exp RPAREN -> ^(binop exp exp)
    | INT -> INT ;

binop: ADD | SUB | MUL | DIV ;

// Lexer Rules
ADD    :        '+';
SUB    :        '-';
MUL    :        '*';
DIV    :        '/';
LPAREN :        '(';
RPAREN :        ')';

INT : '0'..'9'+ ;

WS :  ( ' ' | '\t' | '\r' | '\n')+ {$channel=HIDDEN;};
```


Processing ANTLR from shell

```
[fturbak@new-host Intex]$ java org.antlr.Tool IntexParens.g
  // Generate IntexParensParser.java, IntexParensLexer.java, etc.
[fturbak@new-host Intex]$ javac IntexParensParser.java
  // Compile IntexParensParser.java, IntexParensLexer.java, etc.
[fturbak@new-host Intex]$ java IntexParensParser
  // Run IntexParensParser on user input
((3+4)*(42  - 17)  ) // program entered by user
(* (+ 3 4) (- 42 17)) // displayed AST

// Version with default (flat) AST
[fturbak@new-host Intex]$ java org.antlr.Tool IntexParensFlat.g
[fturbak@new-host Intex]$ javac IntexParensFlatParser.java
[fturbak@new-host Intex]$ java IntexParensFlatParser
((3+4)*(42  - 17)  )
( ( 3 + 4 ) * ( 42 - 17 ) ) <EOF>
```