Semantic Analysis

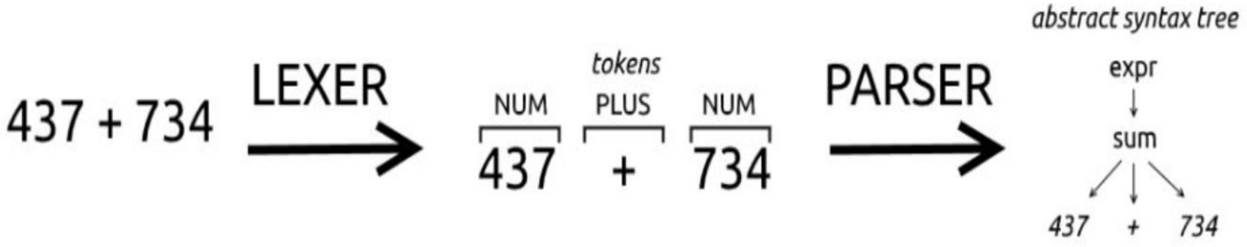
ANTLR

Agenda

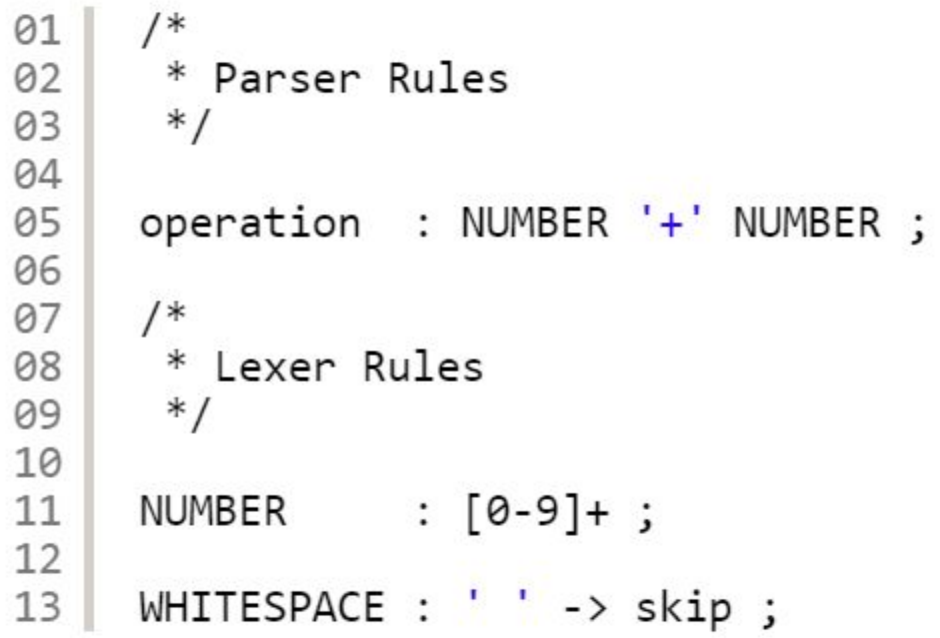
1. Lexer and parser rules in ANTLR
    a. Renaming variables
    b. Evaluating expression
2. Synthesized and inherited attributes
3. Supplementary notes

1. Difference between lexer and parser rules

- Terminal symbols describe the input, while non-terminal symbols describe the tree structure behind the input.
- Terminal symbols are recognized by a lexer and non-terminal symbols are recognized by a parser.
    - lexer rules start with an uppercase letter
    - parser rules start with a lowercase letter



Example of parser and lexer rules



```
/*
 * Parser Rules
 */

operation  : NUMBER '+' NUMBER ;

/*
 * Lexer Rules
 */

NUMBER     : [0-9]+ ;

WHITESPACE : ' ' -> skip ;
```

Example : Calculator

- ANTLR grammar has two building blocks: TOKEN and parser rule.
- As shown in the following code
  - Tokens are written in all uppercase and parser rules are written in all lower case.

```
grammar calc;

start: operation EOF;
operation
    : NUMBER '*' NUMBER
    | NUMBER '/' NUMBER
    | NUMBER '+' NUMBER
    | NUMBER '-' NUMBER      ;

NUMBER  : ('0' .. '9') + ('.' ('0' .. '9') +)?  ;
WS          : [ \r\n\t] + -> skip;
```

- The parser rule operation is an arithmetic addition, subtraction, multiplication or division.
- In this grammar, there are two tokens:
  - WS (spaces or tabs) which are ignored by the ANTLR by adding them to the hidden channel.
  - NUMBER token is represented by a regular expression to match all positive numbers.

a. Renaming variables

The operands are named *left* and *right* to be easily identified.

```
grammar calc;

start  : operation EOF;

operation
                    : left = NUMBER '*' right = NUMBER
                    | left = NUMBER '/' right = NUMBER
                    | left = NUMBER '+' right = NUMBER
                    | left = NUMBER '-' right = NUMBER      ;

NUMBER  : ('0' .. '9') + ('.' ('0' .. '9') +)?     ;
WS           : [ \r\n\t] + -> skip;
```

b. Evaluating expression

```
grammar calc;

/*----------------
* PARSER RULES
*----------------*/

start: operation EOF;
operation :
        left = NUM '*' right = NUM
        {   System.out.println(Float.valueOf($left.text) * Float.valueOf($right.text)); }

        | left=NUM '/' right=NUM
        {System.out.println(Float.parseFloat($left.text) / Float.parseFloat($right.text) );}
```

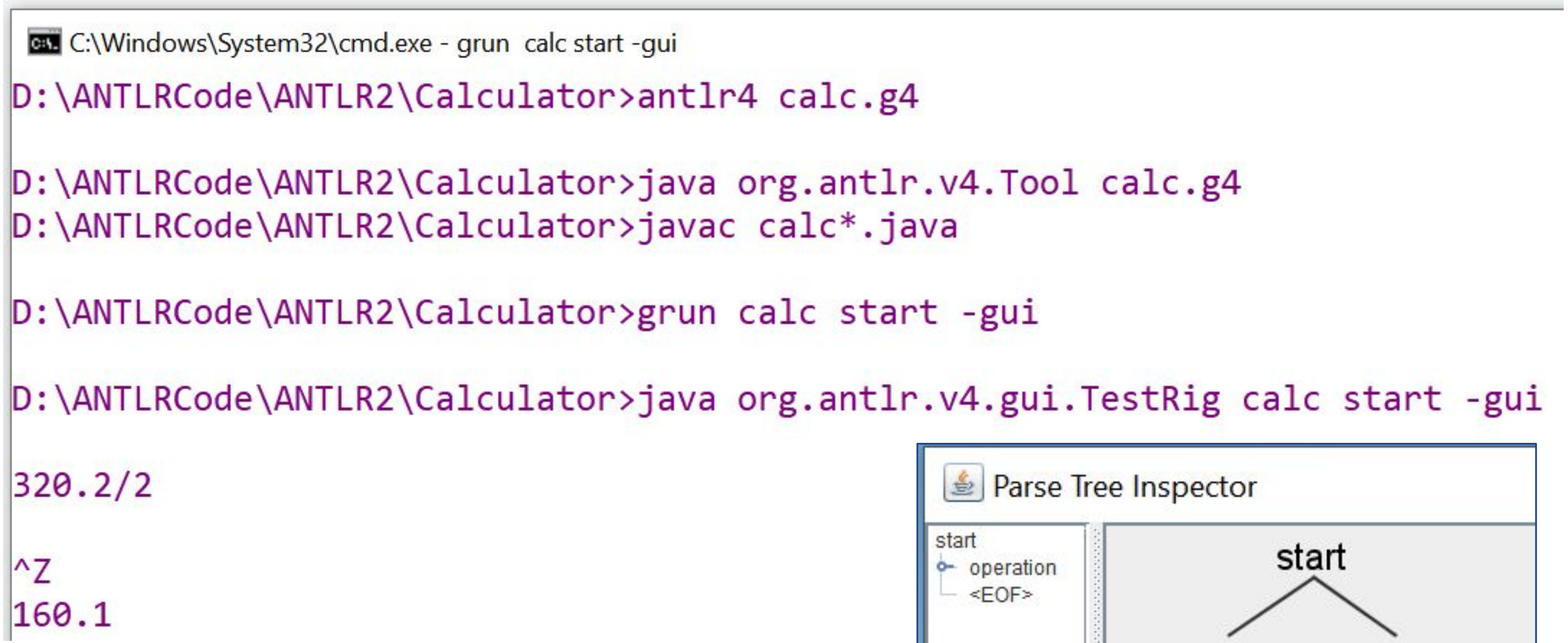
Converting parsed text into float

Embedding java code between { }

```
        | val1=NUM '+' val2=NUM
        { Integer x = Integer.valueOf( $val1.text ).intValue();
          Integer y = Integer.valueOf( $val2.text ).intValue();
          System.out.println(x+y);}

        | n1=NUM '-' n2=NUM
        {System.out.println(Integer.parseInt($n1.text) - Integer.parseInt($n2.text));}
        ;

/*----------------
* LEXER RULES
*----------------*/
NUM : ('0' .. '9') + ('.' ('0' .. '9') +)?     ;

WS : [ \r\n\t] + -> skip;
```

c. Run Antlr

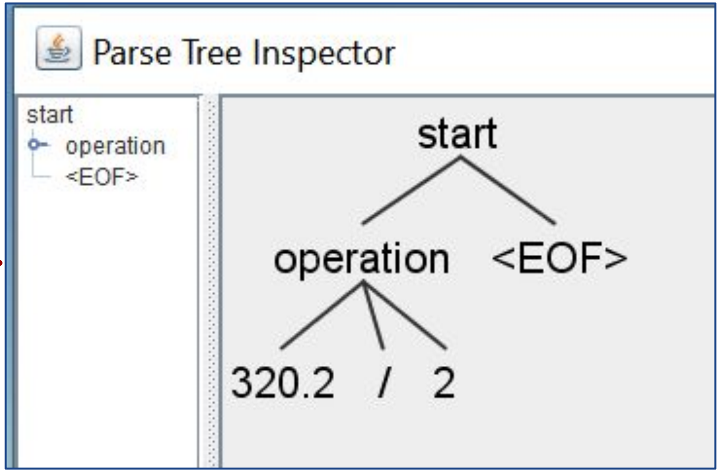
1. Change directory to the folder where you placed your grammar (.g4) file
   - cd D:\ANTLRCode\ANTLR2\Calculator
2. Run antlr on your grammar:
   - antlr4 calc.g4
3. Compile your grammar:
   - javac calc*.java
4. Run:
   - grun calc start –gui
5. Test your grammar:
   - 320.2/2
   - Ctrl+Z to terminate (EOF)


C:\Windows\System32\cmd.exe - grun calc start -gui

```
D:\ANTLRCode\ANTLR2\Calculator>antlr4 calc.g4

D:\ANTLRCode\ANTLR2\Calculator>java org.antlr.v4.Tool calc.g4
D:\ANTLRCode\ANTLR2\Calculator>javac calc*.java

D:\ANTLRCode\ANTLR2\Calculator>grun calc start -gui

D:\ANTLRCode\ANTLR2\Calculator>java org.antlr.v4.gui.TestRig calc start -gui

320.2/2

^Z
160.1
```

Result

GUI



2. Synthesized and inherited attributes

- In Syntax Directed Definition, two attributes are used
  - ❑ Synthesized attribute:
    - ❑ its parse tree node value is determined by the attribute value at child nodes
  - ❑ Inherited attribute:
    - ❑ its parse tree node value is determined by the attribute value at parent and/or siblings node.

Example

E → Term Expr
Expr → + Term Expr | - Term Expr | ε
Term → digit

	Production	Semantic Rule

1	E → Term Expr	E.syn = Expr.syn
		Expr.inh = Term.val

2	Expr → + Term Expr1	Expr.inh = E.inh + Term.val
		E.syn = Expr1.syn

3	Expr → - Term Expr	Expr.inh = E.inh - Term.val
		E.syn = Expr1.syn

4	Expr → ε	Expr.syn = Expr.inh
5	Term → digit	Term.val = digit.lexval

ANTLR

$inh represents the inherited attribute
$val represents the synthesized attribute

Declaration of synthesized attribute

```
grammar calc3;
start : e EOF          {System.out.println("The result is" + $e.val);}  ;

e  returns [int val]
   : term expr [$term.val]          {$val = $expr.val;}  ;

expr [int inh] returns [int val]
   : '+' term E1 = expr [$inh + $term.val]   { $val = $E1.val; }
   | '-' term E1 = expr [$inh - $term.val]   { $val = $E1.val; }
   |                                         { $val = $inh; }     ;
```

Declaration of inherited attribute

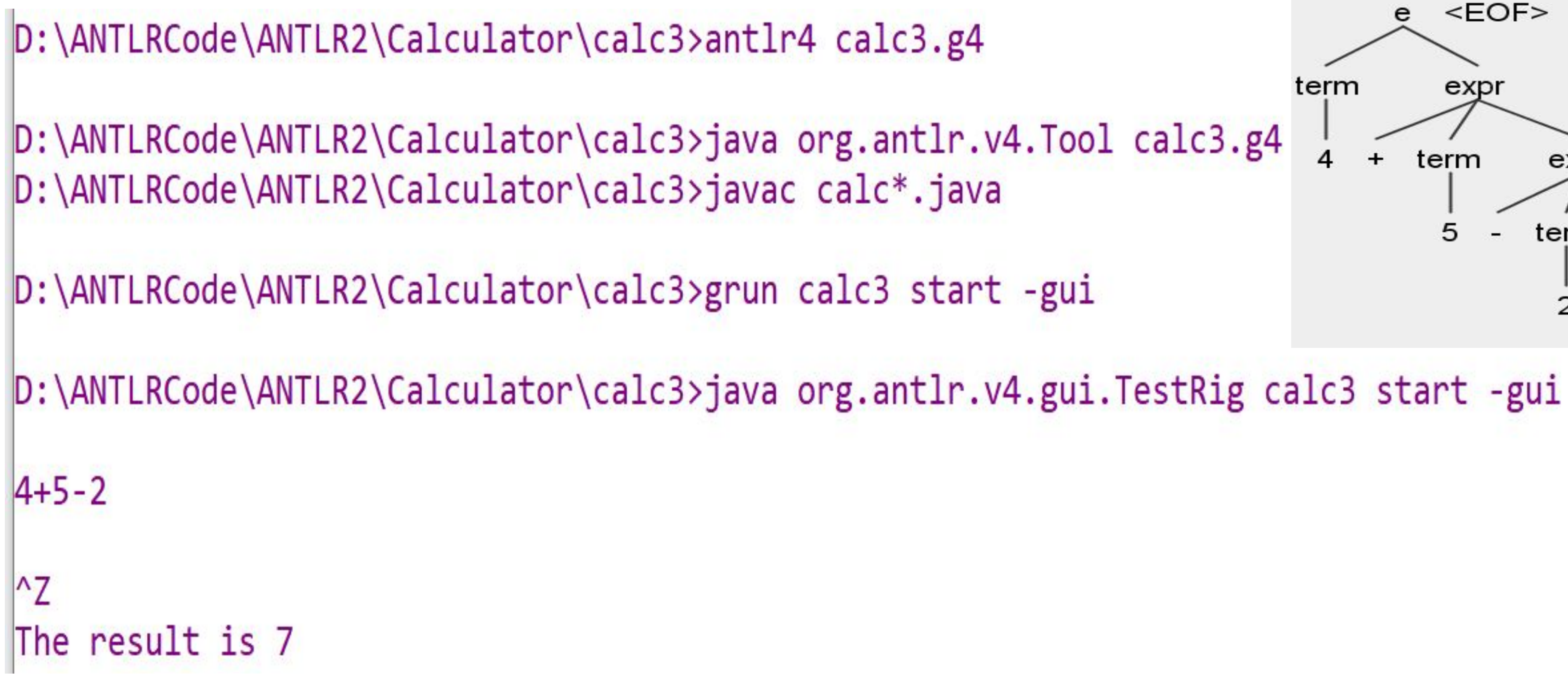
Evaluating synthesized attribute

Passing values of inherited attributes

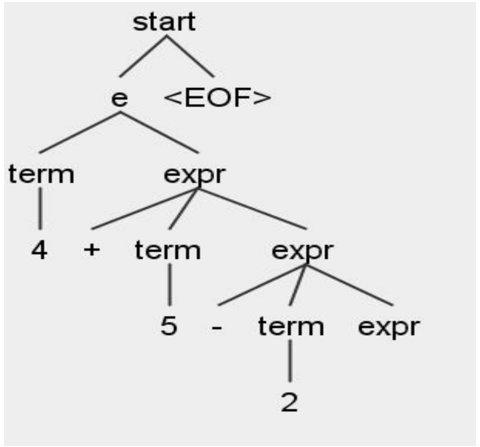
Continue . . .

```
term returns [int val]
    : NUM       {$val = Integer.parseInt($NUM.text);};

NUM : ('0' .. '9') +      ;
```

Evaluating 4 + 5 - 2



```
D:\ANTLRCode\ANTLR2\Calculator\calc3>antlr4 calc3.g4

D:\ANTLRCode\ANTLR2\Calculator\calc3>java org.antlr.v4.Tool calc3.g4
D:\ANTLRCode\ANTLR2\Calculator\calc3>javac calc*.java

D:\ANTLRCode\ANTLR2\Calculator\calc3>grun calc3 start -gui

D:\ANTLRCode\ANTLR2\Calculator\calc3>java org.antlr.v4.gui.TestRig calc3 start -gui

4+5-2

^Z
The result is 7
```



Declaring more than one inherited/synthesized attributes

- Declaring and passing more than one inherited or synthesized attributes can be done by separating them by comma.

Example:

$$\text{term } \underbrace{[\text{double f, double p}]}_{\text{Inherited Attributes}} \text{ returns } \underbrace{[\text{double min, double val, int x}]}_{\text{Synthesized Attributes}}$$

Supplementary Notes

Importing Java library in Antlr

```
grammar g1;

@parser::header

{

   import java.lang.Math;

   import java.util.ArrayList;

}

. . .
```

The extended notations and their meaning

( )	Parentheses. Used to group several elements, so they are treated as one single token
?	Any token followed by ? occurs 0 or 1 times
*	Any token followed by * can occur 0 or more times
+	Any token followed by + can occur 1 or more times
.	Any character/token can occur one time
~	Any character/token following the ~ may not occur at the current place
..	Between two characters .. spans a range which accepts every character between both boundaries inclusive

References

- https://github.com/antlr/antlr4/blob/master/doc/index.md
- https://tomassetti.me/antlr-mega-tutorial/
- https://stackoverflow.com/questions/48094546/making-calculator-with-antlr
- https://www.inf.usi.ch/faculty/soule/teaching/2015-fall/cc/antlr-intro.pdf
- https://theantlrguy.atlassian.net/wiki/spaces/ANTLR3/pages/2687027/Grammars
- https://stackoverflow.com/questions/22744336/antlr-synthesized-and-inherited-attributes?fbclid=IwAR2hQvUBw5ihzdpt7kUTgJQ_BTl9fv3jE9WjcwN7oLbSLAV0lgxIeI5M2B4