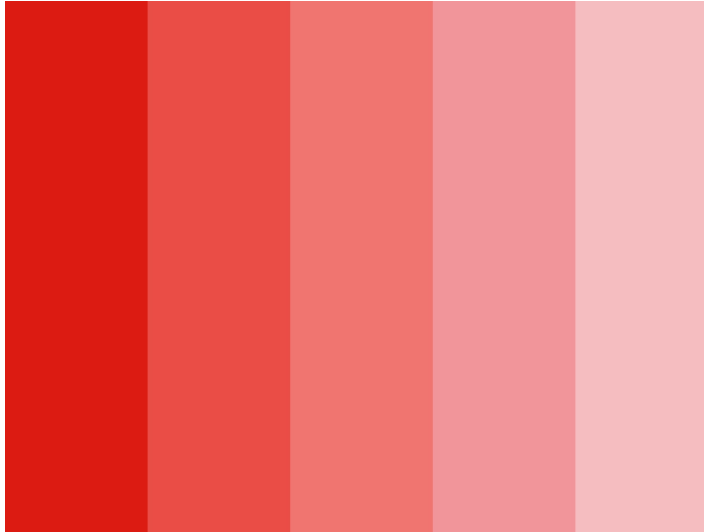
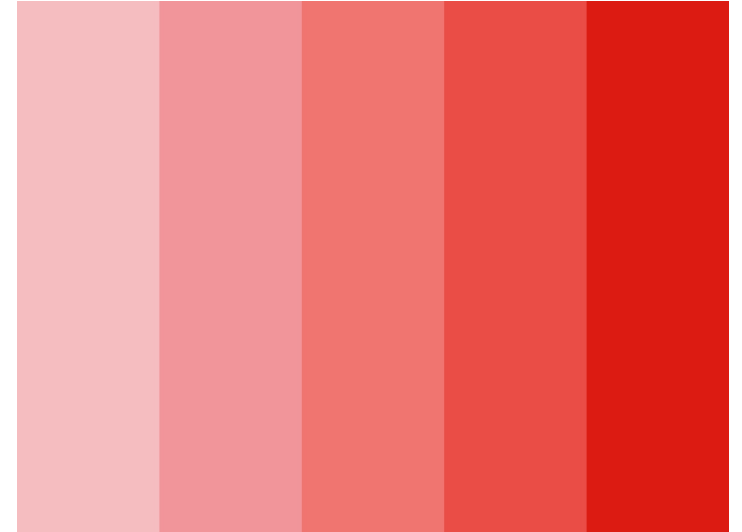


Semantic Analysis

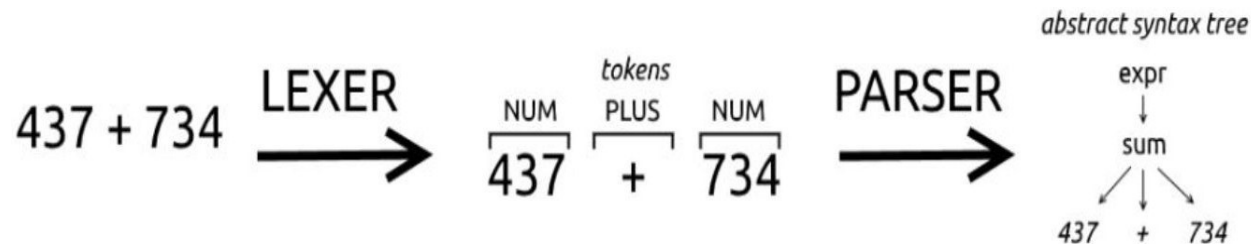


ANTLR



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 lexerrules

```
01  /*
02   * Parser Rules
03   */
04
05  operation : NUMBER '+' NUMBER ;
06
07  /*
08   * Lexer Rules
09   */
10
11  NUMBER : [0-9]+ ;
12
13  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

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;

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

b. Evaluating expression

grammar calc;

```
/*-----  
*  PARSE RULES  
*-----*/
```

```
start: operation EOF;  
operation :
```

```
left = NUMBER '*' right = NUMBER
```

```
{ System.out.println(Float.valueOf($left.text) * Float.valueOf($right.text)); }
```

```
| left = NUMBER '/' right = NUMBER
```

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

Converting parsed text
into float

Embedding java
code between { }

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

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

```
/*-----  
* LEXER RULES  
*-----*/
```

```
NUMBER : ('0' .. '9') + ('.' ('0' .. '9') +)?    ;
```

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

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.

Row --> Digit | Row

Row returns (int sum):

Digit { \$sum = \$Digit.val }

| Digit Row { \$sum = Digit.val +



Example

$E \rightarrow \text{Term Expr}$

$\text{Expr} \rightarrow + \text{Term Expr} \mid - \text{Term Expr} \mid \varepsilon$

$\text{Term} \rightarrow \text{digit}$



Ex: $2 + 3 - 1$

ANTLR

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

Declaration of
synthesized
attribute

```
grammar calc2;  
start : e EOF {System.out.println("The result is" + $e.val);} ;
```

e --> term expr

Declaration
of inherited
attribute

```
e returns [int val]  
: term expr [$term.val]
```

```
{ $val = $expr.val; } ;
```

Evaluating
synthesized
attribute

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

```
{ $val = $E1.val; }  
{ $val = $E1.val; }  
{ $val = $inh; } ;
```

expr --> + term expr
expr --> - term expr

Passing values of
inherited attributes

Continue ...

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


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

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:

term [double f, double p] returns [double min, double val, int x]



The diagram illustrates the declaration of multiple attributes. A blue curly brace under the parameters [double f, double p] is labeled "Inherited Attributes". A red curly brace under the return values [double min, double val, int x] is labeled "Synthesized Attributes".

Inherited Attributes

Synthesized Attributes

Supplementary Notes



Importing Java library in Antlr

```
grammar g1;
```

```
@header
```

```
{
```

```
    import java.lang.Math;
```

```
    import java.util.ArrayList;
```

```
}
```

```
...
```

expr [int inh] returns [int val]

locals [int i=0] // initialize variables if needed

@init { // run before the rule
}

@after { // run after the rule
}

: '+' term E1 = expr [\$inh + \$term.val]{ \$val = \$E1.val; }

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_BTl9fv3jE9WjcwN7oLbSLAV0lgxlel5M2B4