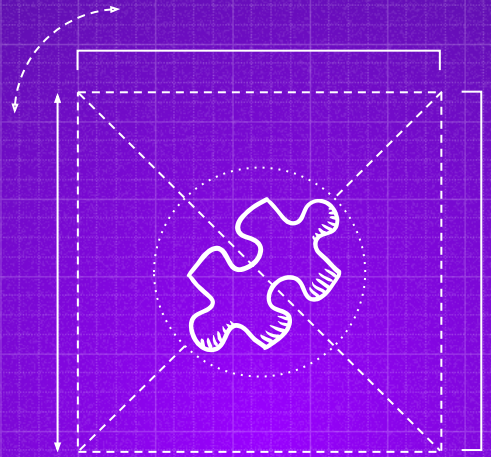
ANTLR 4, Language Parsing, Filter Engine

1

Nature of Language

Tokenization, Lexers, Parsers and Parse trees



Tokenization

Lexical Analysis

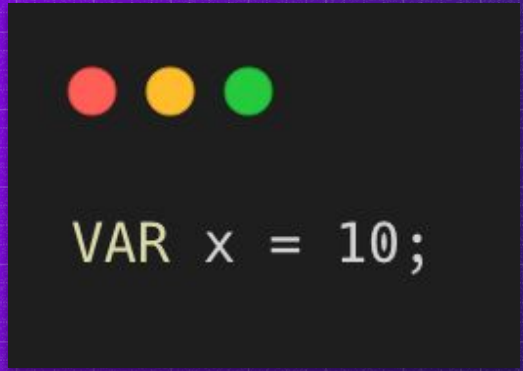
Breaking input into meaningful pieces

Tokenization

Breaking input text into meaningful pieces called “tokens”

Tokens:

- “VAR” keyword
- “x” (identifier)
- “=” (operator)
- “10” (Integer Literal)
- “;” (delimiter)
- “ “ (whitespace)

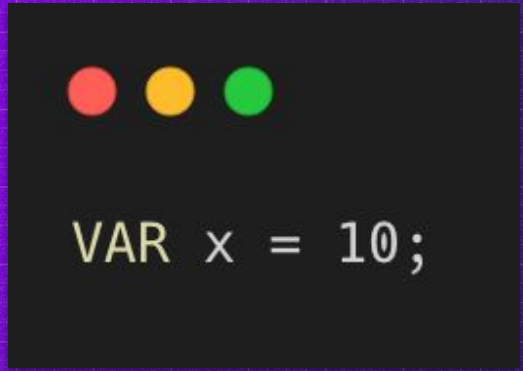


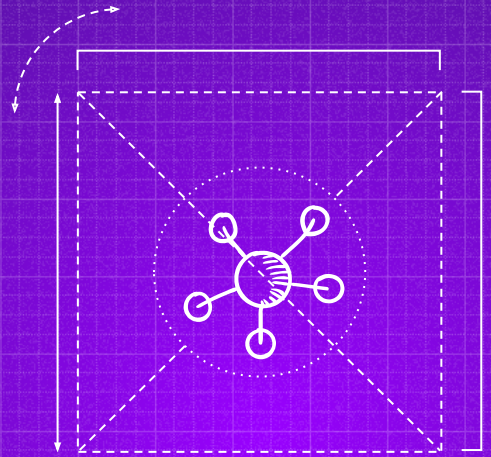
Tokenization

Breaking input text into meaningful pieces called “tokens”

Lexer Rules:

- VAR : 'VAR' ;
- IDENTIFIER : [a-zA-Z][a-zA-Z0-9_]* ;
- INT : [0-9]+ ;
- WS : [\t\r\n\f]+ -> skip ;





Parse Trees

Syntactic Analysis

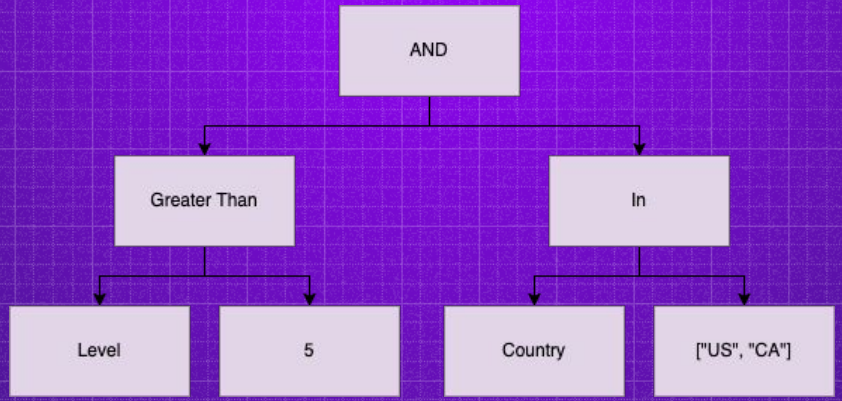
Hierarchical structure of tokens

Parse Trees

Organizing tokens into hierarchical structures using grammar rules

```
LEVEL > 5 AND COUNTRY IN ("US", "CA")
```

Simplified AST (Abstract Syntax Tree)



Parse Trees

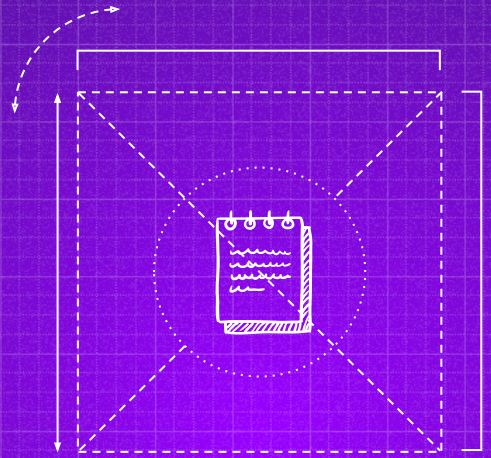
```
LEVEL > 5 AND COUNTRY IN ("US", "CA")
```

```
orExpression    : andExpression (OR andExpression)* ;
andExpression   : notExpression (AND notExpression)* ;
notExpression   : NOT notExpression | primaryExpression ;

primaryExpression :
    filter
  | '(' expression ')'
  ;
```

2

ANTLR and Language Parsers

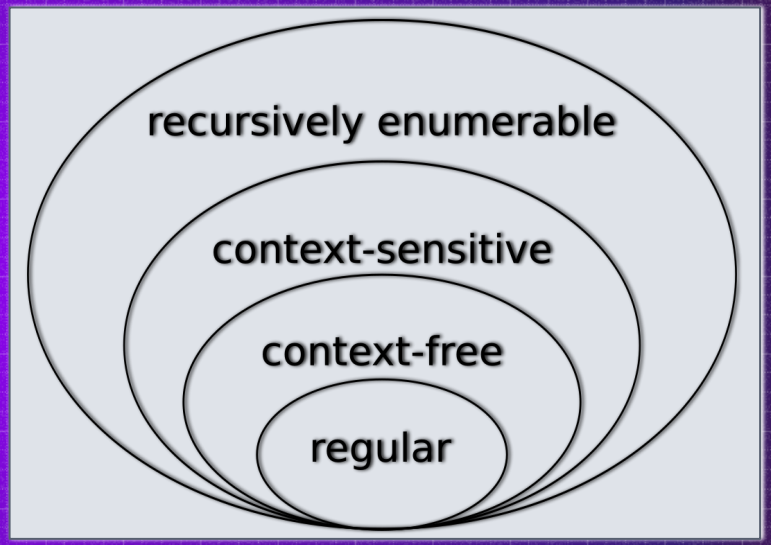
Choosing the tool,
Building the parser

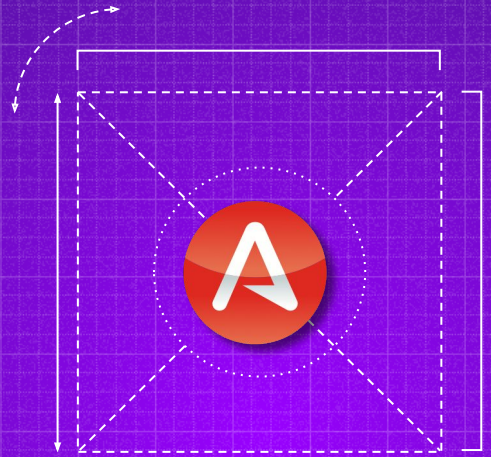


Language Parsers

Language Parsers

- **Two Most common Types of Parsers:**
  - “Regular” Parsers
    - "A follows B", "Either A or B"
    - File Paths, regex identifiers.
    - Flex, Ragel, Lex,...
  - “Context-Free” Parsers
    - **"every A is eventually followed by a matching B"**
    - Most programming languages, recursive structures.
    - Antlr, Parsec, GOLD, GNU Bison





ANTLR

ANother **T**ool for **L**anguage **R**ecognition

Antlr

```
VAR x = 10;
```

Antlr

```
VAR x = 10;
```

Grammar file declaration

‘Script’ type structure

‘Assignment’ type structure

Value and numeric value types

Lexer Token for ‘Var’

INT Token and variable ‘name’ token

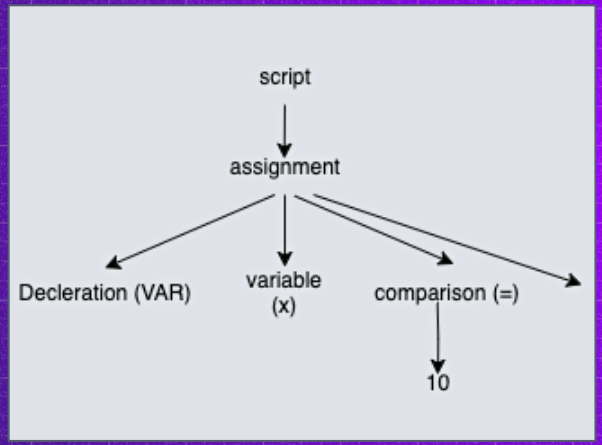
White space and ‘comment’ tokens

```
grammar SimpleVars;

// Parser Rules
script: assignment* EOF;

assignment: VAR IDENTIFIER '=' value ';';

value: numericValue ;
numericValue: INT ;

// Lexer Rules
VAR: 'VAR';

INT: [0-9]+;
IDENTIFIER: [a-zA-Z][a-zA-Z0-9_]*;

WS: [ \t\r\n\f]+ -> skip;
COMMENT: '//' ~[\r\n]* -> skip;
```

Antlr: Listener Events

Listener Events are how the parser connects to its surrounding application.

```
VAR x = 10;
```



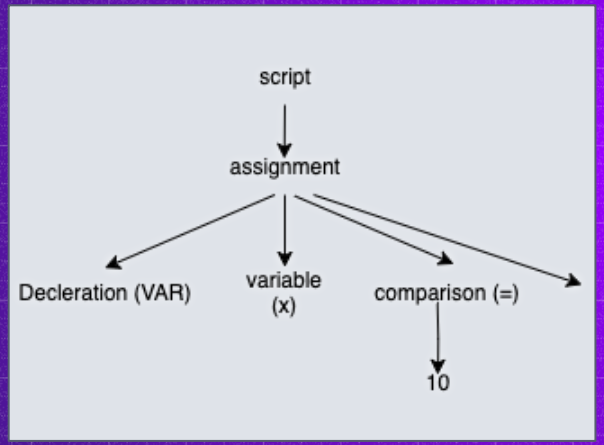
Generates Listener API Methods:

- enterScript(ScriptContext)
- enterAssignment(AssignmentContext)
- visitTerminal(TerminalNode)
- enterComparison(ComparisonContext)
- visitTerminal(TerminalNode)
- exitComparison(ComparisonContext)
- enterVariable(VariableContext)
- exitVariable(VariableContext)
- enterDecleration(AssignmentContext)
- exitDecleration(AssignmentContext)
- exitAssignment(AssignmentContext)
- exitScript(ScriptContext)

Antlr: Visitor Events

Listener Events are how the parser connects to its surrounding application.

```
VAR x = 10;
```



Generates Visitor API Methods:

- visitScript(ScriptContext)
- visitAssignment(AssignmentContext)
- visitDecleration(DeclerationContext)
- visitVariable(VariableContext)
- visitComparison(ComparisonContext)
- visitNumericValue(NumericValueContext)

Antlr: Visitor Events Usage Example

```
public class SimpleVarsVisitor extends SimpleVarsBaseVisitor<Object> {

    ...

}
```

```
    @Override
    public Object visitScript(SimpleVarsParser.ScriptContext ctx) {
        for (SimpleVarsParser.AssignmentContext assignmentCtx : ctx.assignment()) {
            visit(assignmentCtx);
        }
        return variables;
    }
```

Antlr: Visitor Events Usage Example

```java
@Override
public Object visitAssignment(SimpleVarsParser.AssignmentContext ctx) {
    String variableName = ctx.IDENTIFIER().getText();
    Object value = visit(ctx.value());
    variables.put(variableName, value);
    return value;
}

@Override
public Object visitValue(SimpleVarsParser.ValueContext ctx) {
    return visit(ctx.numericValue());
}

@Override
public Object visitNumericValue(SimpleVarsParser.NumericValueContext ctx) {
    if (ctx.INT() != null) {
        return Integer.parseInt(ctx.INT().getText());
    }
    throw new IllegalStateException("Unexpected numeric value type: " + ctx.getText());
}

public Map<String, Object> getVariables() {
    return variables;
}
```