Introduction to

By:
Morteza Zakeri
Iran University of Science and Technology
Fall 2016

Contents at a Glance I

- What is ANTLR?
- LL Grammars
- History
- Motivation
- What is New in ANTLR v4?
- ANTLR Components: How it Works?
- Getting Start with ANTLR v4

Contents at a Glance II

- ANTLR Hello World!
- Integrating a Generated Parser into a Java Program
- A Starter ANTLR Project: ArrayInit
- Building a Language Application
- Customizing BaseListener Methods
- Congratulation!
- References

What is ANTLR?

- ANTLR (pronounced Antler), or **An**other **T**ool For **L**anguage **R**ecognition, is a parser generator that uses LL(*) for parsing.
- ANTLR takes as input a **grammar** that specifies a language and generates as output **source code** for a **recognizer** for that language.
  - supported generating code in Java, C#, JavaScript, Python2 and Python3.

LL(K) Grammars

- An LL parser is a **top-down** parser for a subset of context-free languages.
  - It parses the input from **Left to right**, performing **Leftmost derivation** of the sentence.
- An LL parser is called an LL(k) parser if it uses **k** tokens of look-ahead when parsing a sentence.
- The LL(K) parser is a **deterministic pushdown automaton** with the ability to peek on the next **k** input symbols without reading.

LL(*) Grammars

- An LL parser is called an LL(*) parser **(an LL-regular parser)** if it is not restricted to a finite ***k*** tokens of look-ahead, but can make parsing decisions by recognizing whether the following tokens belong to a **regular language.**
- LL (LL(1), LL(k), LL(*)) grammars can be parsed by **recursive descent parsers**.
- In fact **ANTLR** is recursive descent parser Generator!

History

- Initial release:
  - February **1992**; 24 years ago.
- Stable release:
  - 4.5.1 / July 15, 2015; 14 months ago
- Its maintainer is:
  - **Professor Terence Parr**
  - **University of San Francisco**.

Motivation

- It's widely used in **academia** and **industry** to build all sorts of languages, tools, and frameworks.
  - **Twitter search** uses ANTLR for query parsing, with more than 2 billion queries a day.
  - **Oracle** uses ANTLR within the SQL Developer IDE and its migration tools.
  - The **NetBeans IDE** parses C++ with ANTLR.
  - The **HQL language** in the Hibernate object-relational mapping framework is built with ANTLR.

What is New in ANTLR v4? I

- The most important new feature is:
  - ANTLR v4 gladly accepts every grammar you give it!
  - with one exception regarding **indirect left recursion,** i.e. grammars rules ***x*** which refer to ***y*** which refer to ***x***.
- ANTLR v4 automatically **rewrites left-recursive** rules such as expr into **non left-recursive** equivalents.
  - The only constraint is that the left recursion must be **direct**, where rules immediately reference themselves.

What is New in ANTLR v4? II

- ANTLR v4 dramatically simplifies the grammar rules used to match syntactic structures.
  - like programming language arithmetic expressions.

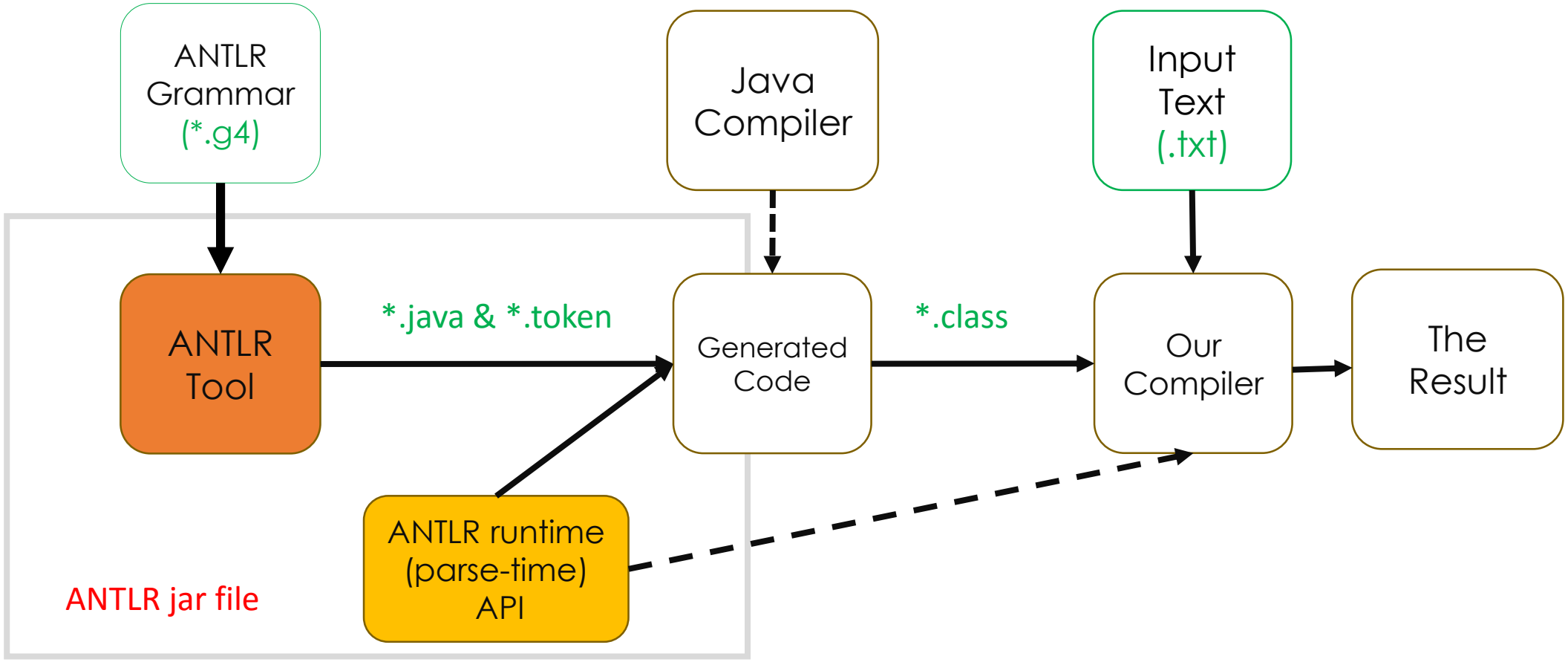
```
expr : expr '*' expr  // match subexpressions joined with '*' operator
     | expr '+' expr  // match subexpressions joined with '+' operator
     | INT            // matches simple integer atom
     ;
```

- ANTLR v4 also automatically generates **parse-tree walkers** in the form of *listener* and *visitor* pattern implementations.

What is New in ANTLR v4? III

- ANTLR v4 de-emphasizes embedding actions (code) in the grammar, favoring *listeners* and *visitors* instead.
  - *Listeners* and *visitors* are the familiar design patterns.
- ANTLR parsers use a new parsing technology called ***Adaptive LL(*)*** or ***ALL(*)*** ("all star").
  - ANTLR v3's **LL(*)** parsing strategy is **weaker than** v4's **ALL(*)**.

ANTLR Components: How it Works?



Getting Start with ANTLR v4: Linux

LINUX

```
$ cd /usr/local/lib
$ wget http://www.antlr.org/download/antlr-4.5.3-complete.jar
$ export CLASSPATH=".:/usr/local/lib/antlr-4.5.3-
complete.jar:$CLASSPATH"
$ alias antlr4='java -jar /usr/local/lib/antlr-4.5.3-complete.jar'
$ alias grun='java org.antlr.v4.gui.TestRig'
```

Getting Start with ANTLR v4: Windows

Windows

1. Download http://antlr.org/download/antlr-4.5.3-complete.jar.
2. Add antlr4-complete.jar to CLASSPATH, either:
    1. Permanently: Using System Properties dialog > Environment variables > Create or append to CLASSPATH variable
    2. Temporarily, at command line:

    ```
    SET CLASSPATH=.;C:\Javalib\antlr4-
    complete.jar;%CLASSPATH%
    ```
3. Create batch commands for ANTLR Tool, TestRig in dir in PATH

    ```
    antlr4.bat: java org.antlr.v4.Tool %*
    grun.bat:   java org.antlr.v4.gui.TestRig %*
    ```

ANTLR Hello World!

```
// Define a grammar called Hello
grammar Hello;
s   : 'hello' ID ;          // match keyword hello followed by an identifier
ID : [a-zA-Z]+ ;            // match lower case identifiers
WS : [ \t\r\n]+ -> skip; // skip spaces, tabs, newlines
```

```
$ antlr4 Hello.g4
$ javac Hello*.java
```

```
$ grun Hello r -tree
  hello World
  ^Z
```

ANTLR Hello World!

Do all with my own bat file!

```
2  java -jar C:\Javalib\antlr-4.5.3-complete.jar *.g4
3  javac -cp C:\Javalib\antlr-4.5.3-complete.jar *.java
4
5  set /P id=Enter Grammar Name:
6  set /P id2=Enter Start rule name:
7  java -cp .;C:\Javalib\antlr-4.5.3-complete.jar org.antlr.v4.gui.TestRig %id% %id2% in.txt -tree -gui
8  set /P id2=Press any key ...
```

Integrating a Generated Parser into a Java Program

- We can integrate the ANTLR generated code into a larger application.
- We'll see simple example in next slides for recognition structures like {1,{2,3},4} in C or JAVA.
- Then we'll look at a simple Java main() that invokes our initializer parser and prints out the parse tree like TestRig's -tree option.

A Starter ANTLR Project: ArrayInit

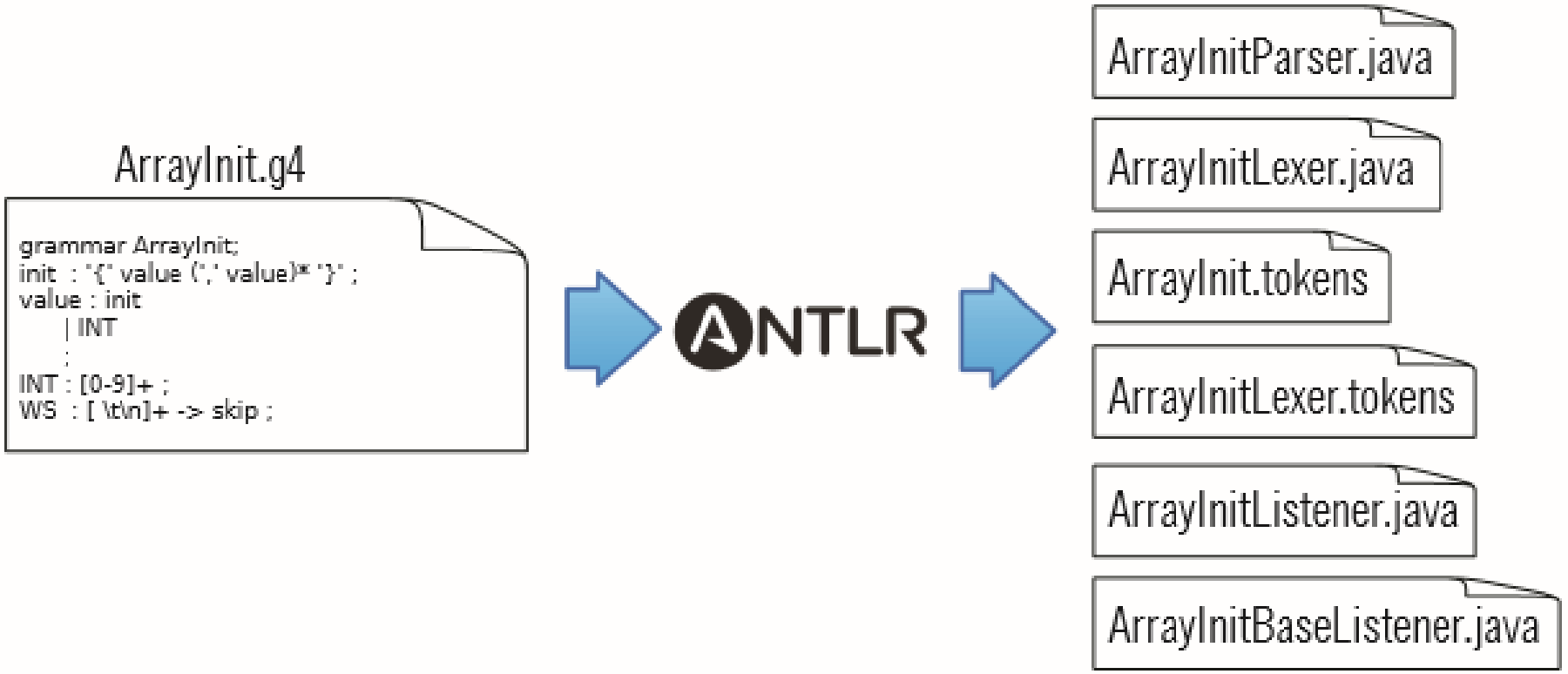
```
/** Grammars always start with a grammar header. This grammar is called
 *  ArrayInit and must match the filename: ArrayInit.g4
 */
grammar ArrayInit;

/** A rule called init that matches comma-separated values between {...}. */
init  : '{' value (',' value)* '}' ;  // must match at least one value

/** A value can be either a nested array/struct or a simple integer (INT) */
value : init
      | INT
      ;

// parser rules start with lowercase letters, lexer rules with uppercase
INT :   [0-9]+ ;              // Define token INT as one or more digits
WS  :   [ \t\r\n]+ -> skip ; // Define whitespace rule, toss it out
```

A Starter ANTLR Project: ArrayInit



A Starter ANTLR Project: ArrayInit

```
// import ANTLR's runtime libraries
import org.antlr.v4.runtime.*;
import org.antlr.v4.runtime.tree.*;

public class Test {
    public static void main(String[] args) throws Exception {
        // create a CharStream that reads from standard input
        ANTLRInputStream input = new ANTLRInputStream(System.in);

        // create a lexer that feeds off of input CharStream
        ArrayInitLexer lexer = new ArrayInitLexer(input);

        // create a buffer of tokens pulled from the lexer
        CommonTokenStream tokens = new CommonTokenStream(lexer);

        // create a parser that feeds off the tokens buffer
        ArrayInitParser parser = new ArrayInitParser(tokens);

        ParseTree tree = parser.init(); // begin parsing at init rule
        System.out.println(tree.toStringTree(parser)); // print LISP-style tree
    }
}
```

A Starter ANTLR Project: ArrayInit

- Here's how to compile everything and run Test:

```
⇛ $ javac ArrayInit*.java Test.java
⇛ $ java Test
⇛ {1,{2,3},4}
⇛ EOF
❮ (init { (value 1) , (value (init { (value 2) , (value 3) })) , (value 4) })
```

A Starter ANTLR Project: ArrayInit

- ANTLR parsers also automatically report and recover from syntax errors.
- For example, here's what happens if we enter an initializer that's missing the final curly brace:

```
⇛ $ java Test
⇛ {1,2
⇛ EOF
❮ line 2:0 missing '}' at '<EOF>'
  (init { (value 1) , (value 2) <missing '}'>)
```

Building a Language Application I

- An application that merely checks syntax is not that impressive!
- Continuing with our array initializer example, our next goal is to **translate** not just **recognize** initializers.
- For example, let's translate Java short arrays like {99,3,451} to "\u0063\u0003\u01c3" where 63 is the hexadecimal representation of the 99 decimal.

Building a Language Application II

- To move beyond recognition, an application has to extract data from the parse tree.
  - ANTLR automatically generates a **listener infrastructure** for us.
  - These **listeners** are like the **callbacks** on GUI widgets (for example, a button would notify us upon a button press) or like SAX events in an XML parser.

Building a Language Application III

- To write a program that reacts to the input, all we have to do is **implement a few methods** in a **subclass** of *ArrayInitBaseListener*.
  - The basic strategy is to have each **listener method** print out a translated piece of the input when called to do so by the **tree walker**.
  - All we know is that our listener gets notified at the beginning and end of phrases associated with rules in the grammar and we don't even have to know that the **runtime** is walking a tree to call our methods.

Customizing BaseListener Methods

```
/** Convert short array inits like {1,2,3} to "\u0001\u0002\u0003" */
public class ShortToUnicodeString extends ArrayInitBaseListener {
    /** Translate { to " */
    @Override
    public void enterInit(ArrayInitParser.InitContext ctx) {
        System.out.print('"');
    }

    /** Translate } to " */
    @Override
    public void exitInit(ArrayInitParser.InitContext ctx) {
        System.out.print('"');
    }
    /** Translate integers to 4-digit hexadecimal strings prefixed with \\u */
    @Override
    public void enterValue(ArrayInitParser.ValueContext ctx) {
        // Assumes no nested array initializers
        int value = Integer.valueOf(ctx.INT().getText());
        System.out.printf("\\u%04x", value);
    }
}
```

Language Application Main Class

```
// import ANTLR's runtime libraries
import org.antlr.v4.runtime.*;
import org.antlr.v4.runtime.tree.*;

public class Translate {
    public static void main(String[] args) throws Exception {
        // create a CharStream that reads from standard input
        ANTLRInputStream input = new ANTLRInputStream(System.in);
        // create a lexer that feeds off of input CharStream
        ArrayInitLexer lexer = new ArrayInitLexer(input);
        // create a buffer of tokens pulled from the lexer
        CommonTokenStream tokens = new CommonTokenStream(lexer);
        // create a parser that feeds off the tokens buffer
        ArrayInitParser parser = new ArrayInitParser(tokens);
        ParseTree tree = parser.init(); // begin parsing at init rule

        // Create a generic parse tree walker that can trigger callbacks
        ParseTreeWalker walker = new ParseTreeWalker();
        // Walk the tree created during the parse, trigger callbacks
        walker.walk(new ShortToUnicodeString(), tree);
        System.out.println(); // print a \n after translation
    }
}
```

Run and Test!

- Let's build the translator and try it on our sample input:

```
⇛ $ javac ArrayInit*.java Translate.java
⇛ $ java Translate
⇛ {99, 3, 451}
⇛ EOF
❮ "\u0063\u0003\u01c3"
```

Congratulation!

- It works! We've just built **our first translator**, without even touching the grammar!
- All we had to do was implement a few methods that printed the appropriate phrase translations.
- **Listeners** effectively **isolate** the language application from the grammar, making the grammar **reusable** for other applications.

References

1. The Definitive ANTLR 4 Reference
   - Terence Parr, The Pragmatic Programmers, LLC; 2012.
2. ANTLR 4 Official Website:
   - http://www.antlr.org/
3. ANTLR page on Wikipedia
   - https://en.wikipedia.org/wiki/ANTLR


Thank you for your attention!
Slides will available on:
www.micropedia.ir
Do you have any question?
m-zakeri@live.com