Part 1: Introduction

By:
Morteza Zakeri
PhD Student
Iran University of Science and Technology
Winter 2020

Agenda

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

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**.
- **ANTLR** is recursive descent parser Generator! (See Appendix)

Runtime Libraries and Code Generation Targets

- There is no language specific code generators
- There is **only one tool, written in Java**, which is able to generate Lexer and Parser code for all targets, through command line options.
- The available targets are the following (2020):
  - Java, C#, C++, Swift, Python (2 and 3), Go, PHP, and JavaScript.
- Read more:
  - https://github.com/antlr/antlr4/blob/master/doc/targets.md

Runtime Libraries and Code Generation Targets

- $ java -jar antlr4-4.8.jar **-Dlanguage=CSharp** MyGrammar.g4
  - https://github.com/antlr/antlr4/tree/master/runtime/CSharp
  - https://github.com/tunnelvisionlabs/antlr4cs
- $ java -jar antlr4-4.8.jar **-Dlanguage=Cpp** MyGrammar.g4
  - https://github.com/antlr/antlr4/blob/master/doc/cpp-target.md
- $ java -jar antlr4-4.8.jar **-Dlanguage=Python3** MyGrammar.g4
  - https://github.com/antlr/antlr4/blob/master/doc/python-target.md

History

- Initial release:
  - February **1992**; 24 years ago.
- The latest version
  - **4.8, released January 16, 2020.**
- ANTLR creator and maintainer
  - **Dr. Terence Parr**
  - University of San Francisco.

Motivation

- In my experience, almost no one uses parser generators to build commercial compilers.
- People use ANTLR for their everyday work
  - building everything from configuration files to little scripting languages.

Motivation

- ANTLR is 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.

Motivation

- In IUST Reverse Engineering Research Laboratory
  - We use ANTLR for **software refactoring** and **software testing.**

What is New in ANTLR v4?

- v4 is the culmination of **25 years of research into parsers and parser generators.** I think I finally know what I want to build. :)

What is New in ANTLR v4?

- 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?

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

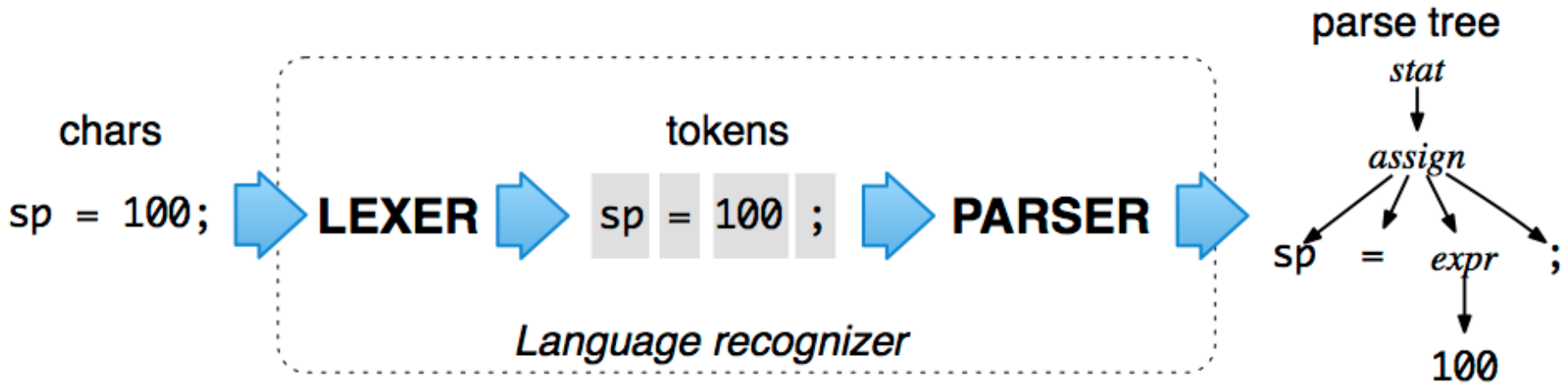
```
expr : expr '*' expr  // match subexpressions joined with '*' operator
     | expr '+' expr  // match subexpressions joined with '+' operator
     | INT            // matches simple integer atom
     ;
```

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

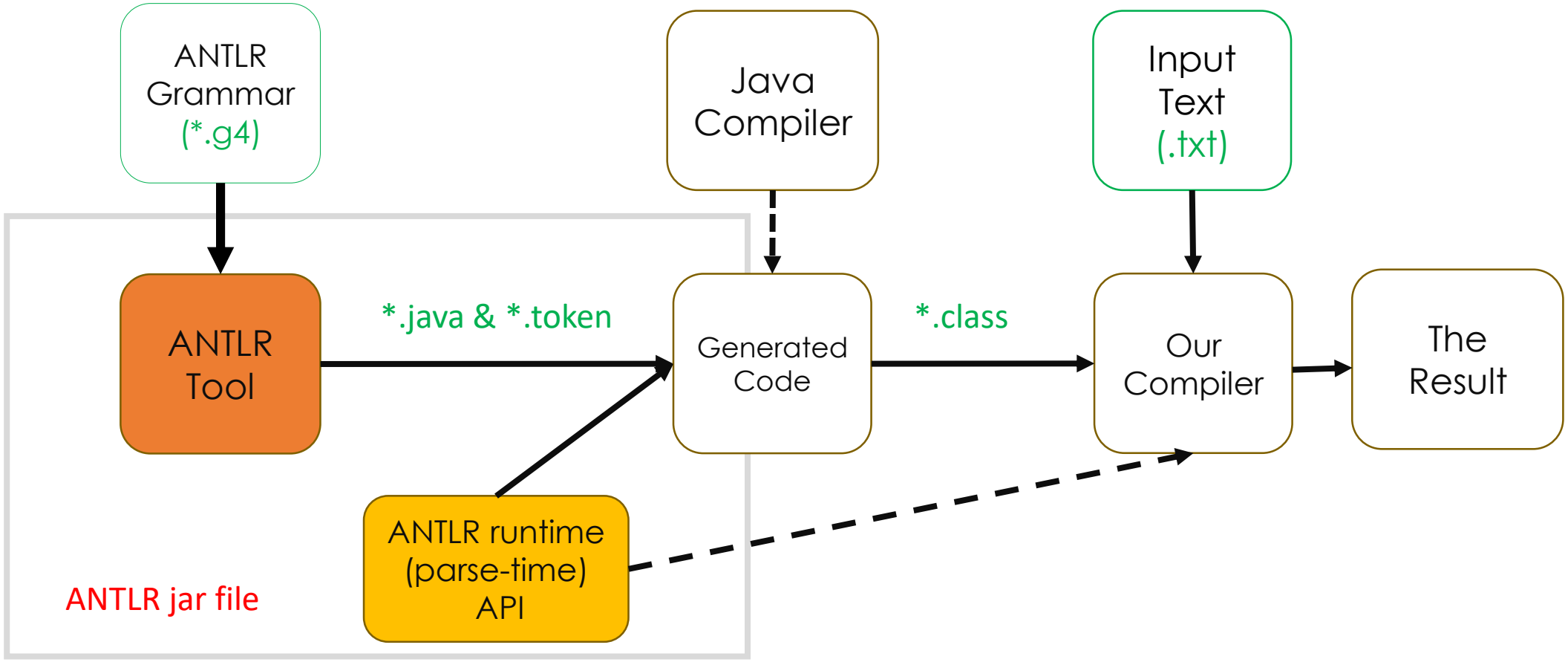
What is New in ANTLR v4?

- 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?



ANTLR Components: How it Works?



Getting Started 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 Started 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 %*
```

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

Part 2: Getting Started with ANTLR in JAVA

Next Session


Thank you for your attention!
Do you have any question?
m-zakeri@live.com

Appendix

LL(K) Grammars

LL(K) Parsers

LL(*) Parsers

LL(K) Grammars

LL(K) Parsers

- 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(*) Parsers

- 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!