

# Back to School and learn some (more) grammar...

Dave Karetnyk

*Premise: a 'reasonable understanding' of modern parsing techniques and tools can lead to sensible and elegant solutions to a class of problems that otherwise remain out of reach. For a number of reasons the ANTLR tool is prescribed as the obvious front-runner in such scenarios.*

## General Problem

Many programming tasks are ultimately about manipulating data. That is, some combination of reading data, translating or modifying data, interpreting data, or writing data. For instance:

- reading an XML file;
- writing a JSON file;
- compiling a C++ program;
- interpreting a Python script.

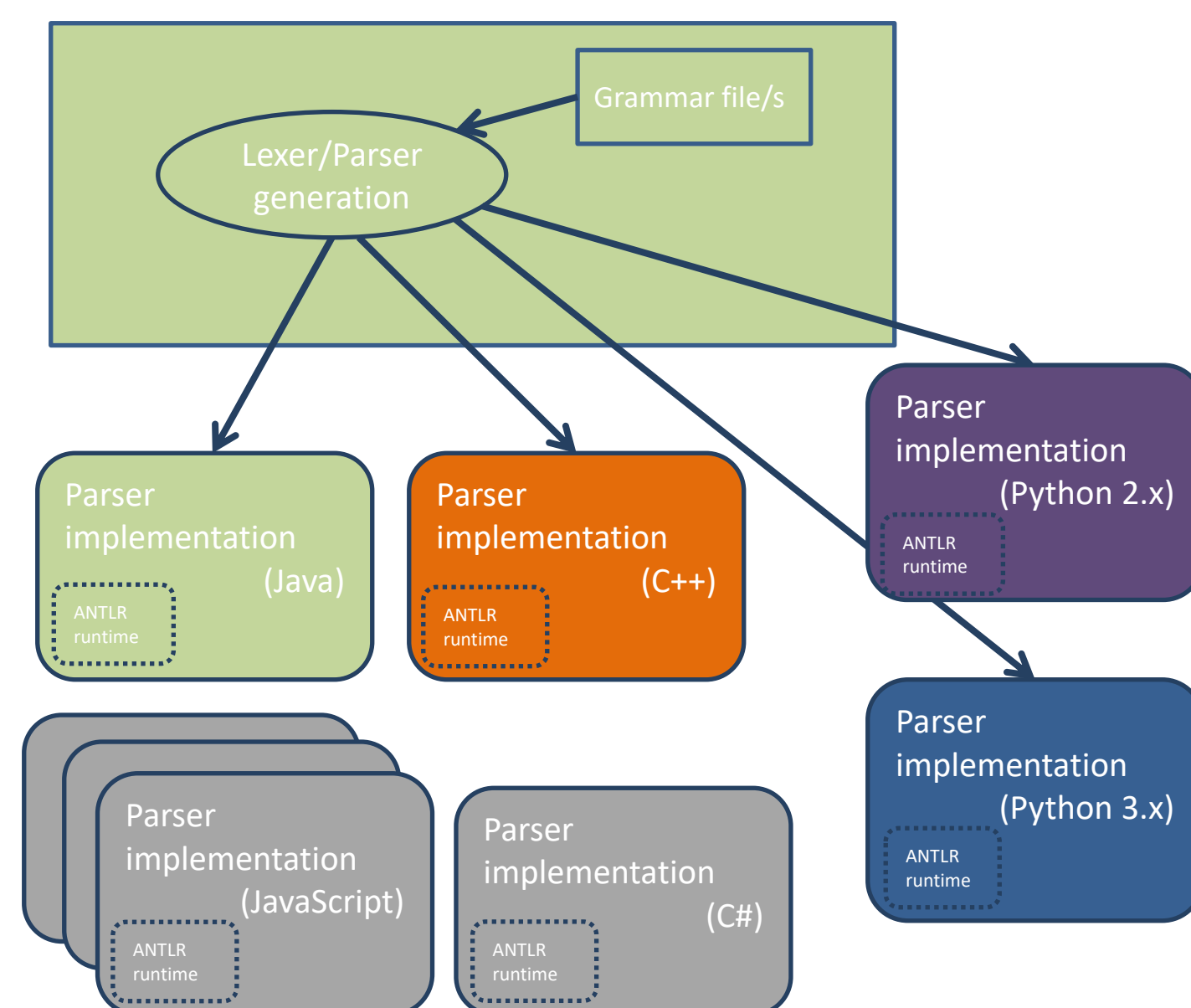
'Under the hood' there is commonality in how these problems are tackled, regardless of whether it is 'data' or 'code' that is being processed. In essence **well structure content** is being manipulated.

## A parsing approach

Assuming there is some degree of structure to the content then a parsing approach might be fruitful. 20-30 years ago this would be the domain for the parsing experts. At that time you would either write your own parser or use the tooling available at the time:

- UNIX Lex and Yacc.
- Or more recently the GNU variants Flex and Bison.

Well that was a long time ago and computer language parsing theory and the associated tooling has progressed considerably since then. Enter some of the modern day alternatives: for example: ANTLR; Boost Spirit; Xtext. The author is particularly interested in ANTLR since its developer (Terence Parr) has work in the field of language parsing for the last ~25 years and ANTLR is the tangible outcome of that effort. Digging further into the alternatives typically points back to ANTLR as a common reference. And Xtext even makes direct use of an older ANTLR version (3.X) internally.

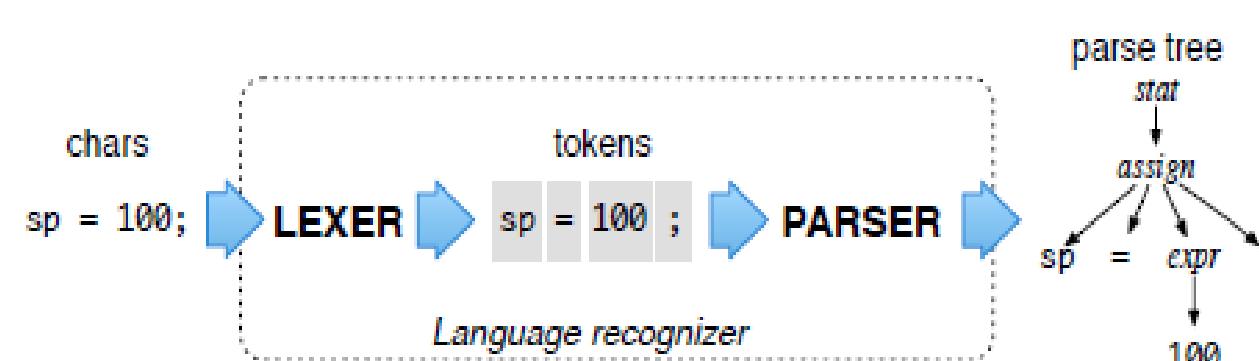


### ANTLR: core and infrastructure

- Grammars for many languages.
- Deployment in multiple languages.
- Active maintenance and online user community.
- Interactive plugin for IntelliJ IDEA.

## Parsing via ANTLR

ANTLR supports deployment in different target languages including Java, C++, and Python allowing it to be embedded in a custom application. There are also ANTLR grammars defined for many languages: Java; Python; C; C++; JSON; xml; mysql; idl etc.



## ANTLR 4.X and Xtext

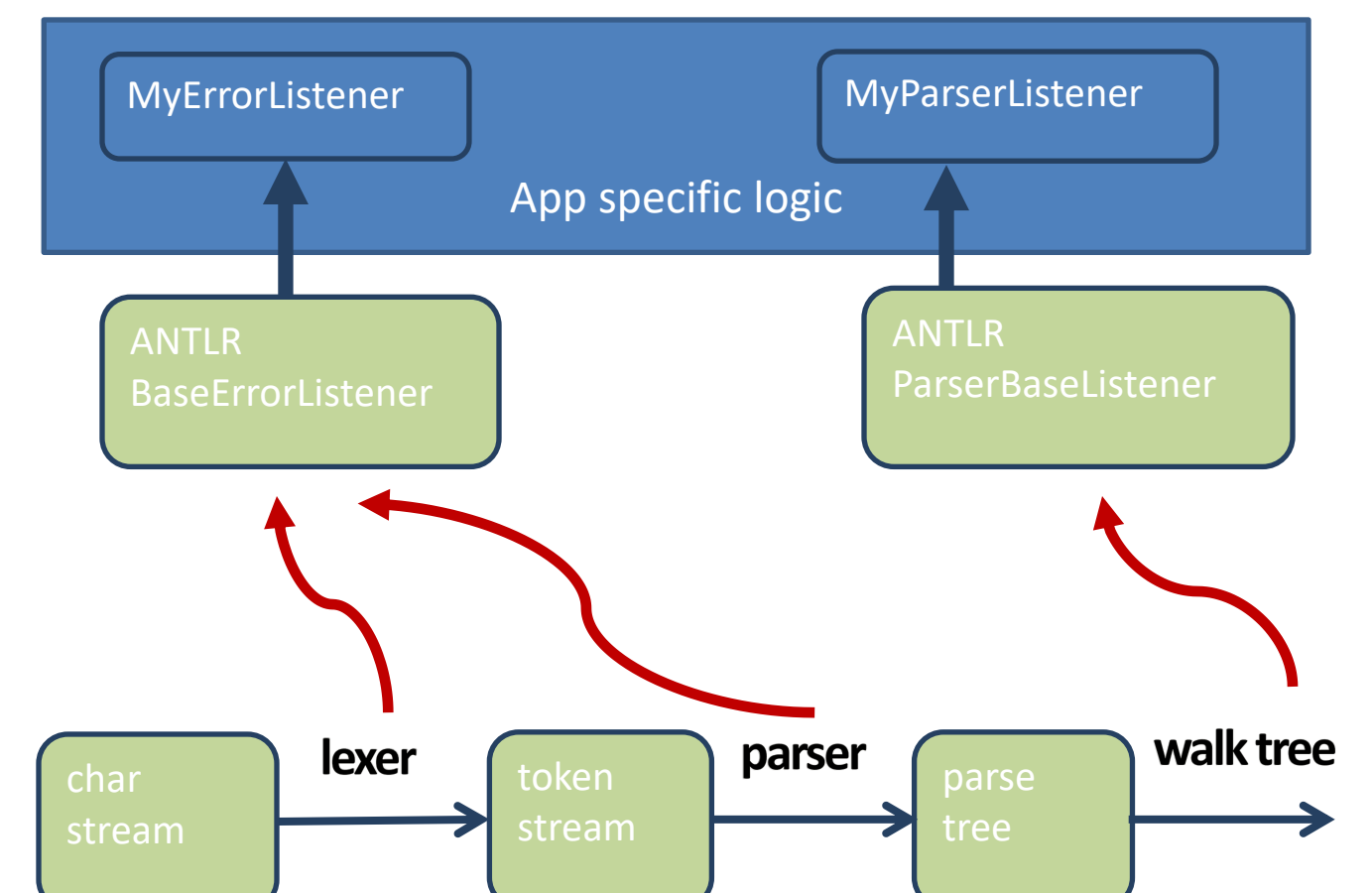
Some colleagues have successfully used Xtext. Great! The reason this route was/is not pursued by the author?

- Xtext uses ANTLR 3.X which does not support direct left recursion (Java grammar is 172 rules in 3.X, 91 in 4.X).
- ANTLR has a larger and more active community.

## Clean Design

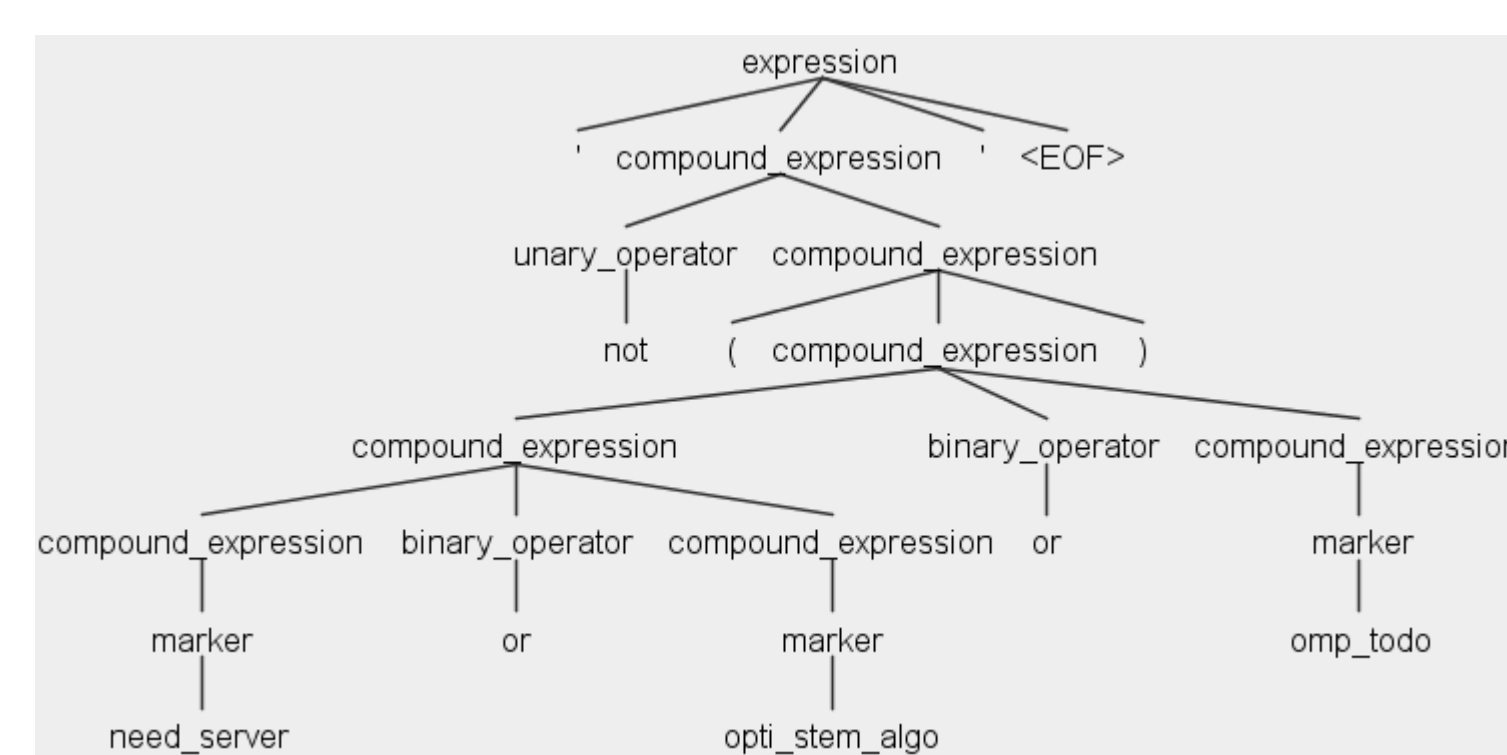
4.X provides a clean separation of the 'parsing code' and the 'application code' that uses it. The generated implementation for a particular grammar allows code for both the

Listener and Visitor design styles to be adopted. This is an option that is selected during code generation.



## Usages and Examples

- Python 2.X syntax checker (used manually but can be easily integrated into Jenkins overnight build/test runs).
- Python Unit test code template generation (insight into how refactoring tools work).
- FDT log file analyzer (helps clarify what to do in the lexer and what to do in the parser).
- pytest expression parser. An example to show the basic principles, including implementation in Java; Python 2.X; and C++ 11.



## Conclusions

Some familiarity with modern parsing techniques and their associated tooling opens the door to solving a class of problems that might otherwise be out of reach. It also removes the mystery behind such trends as DSLs and projectional editing.

As a Software Engineer these are skills that some may find useful.