Parser Generators
building a parser

• We could build all the functions for a recursive descent parser ourselves
  
• But that’s tedious!
  • Analyzing the grammar to build first/follow/predict sets
  • Writing the recursive functions to do the parsing
  • Dealing with issues in the grammar (need more lookahead, need to rewrite)

```
S → X Y $  
X → a Y q  
X → b  
X → Yq  
Y → λ  
Y → d
```
• **Parser generators** solve this problem
  • *given* a grammar, *produce* a parser
  • Can tell you when your grammar is “broken”
  • Can often fix problems in the grammar automatically

• Common parser generators:
  • **Yacc/bison**: classic parser generators that produce **bottom-up** parsers
  • **ANTLR**: produces **recursive-descent** parsers with some extra magic
    • Automatically fix left-recursion, need for more lookahead
    • Perform backtracking when necessary
ANTLR

- Developed based on parser research done at Purdue!
- Domain specific language for writing parsers
- Lets programmer specify grammar, automatically generates recursive-descent parser that builds the parse tree
- Generates Java code (or can generate C++, Python, etc.)
- Makes it easy to add **semantic actions** to take as the parse tree is processed
ANTLR

• Developed based on parser research done at Purdue!
• Domain specific language for writing parsers
• Lets programmer specify grammar, automatically generates recursive-descent parser that builds the parse tree
• Generates Java code (or can generate C++, Python, etc.)
• Makes it easy to add **semantic actions** to take as the parse tree is processed

```
statements : statement statements | empty
statement : base_stmt ';'
         | if_stmt | while_stmt
while_stmt : 'while' '(' cmp_expr ')' '{' statements '}'
```
• Developed based on parser research done at Purdue!
• Domain specific language for writing parsers
• Lets programmer specify grammar, automatically generates recursive-descent parser that builds the parse tree
• Generates Java code (or can generate C++, Python, etc.)
• Makes it easy to add **semantic actions** to take as the parse tree is processed

```antlr
statements : statement statements
            | empty

statement : base_stmt ';'
            | if_stmt
            | while_stmt

while_stmt : 'while' '(' cmp_expr ')' '{' statements '}'
```

**ANTLR**

Keyword for \(\lambda\)

Define simple tokens inline
next: taking action