**Week 1: Lexical Analyzer**

Nate Bachmeier

May 5th, 2019

TIM-8101: Principals of Computer Science

Northcentral University

# Lexical Parsers

A lexical parser (or Lex) is a component that tokenizes input based on a language’s grammar definition. This tokenization does not contain any context and relies on external semantic and logical parsers to perform those additional transformations.

## Example: English Lex

An English lexical parser might be passed the text Hello, good day cofveve! and return the token stream [{Word: Hello}, {Punctuation: ,}, {Word: good}, {Word: day}, {Word: cofveve}, {Punctuation: !}].

The Lex is not responsible for understanding that (1) “good” was an adjective nor (2) if “cofveve” is a real word. It only knows to emit a Word token when the reader sees a continuous series of alphabetic characters.

It is the responsibility of more specialized layers to derive semantic and logical insights from the token stream. This separation of duties improves the design efficiency as later layers can combine multiple generalized tokens into more specialized units.

For example, our document parser might want to optimize common phrases into a single token. It would be easier to reduce the tokens [{Word: good}, {Word: day}] into [{Phrase: good day}] than the deriving meaning from the raw 8-characters good day . Consider the subtle differences between text fragments she shouted “Good! Day!” and … Good! Day or night he… The first matches with superficial punctuation while the second cannot match as it spans sentences.

## Semantic Parsing

Semantic parsing attempts to go one step further than tokenization and provide structural information. Perhaps a layer is added to the parser chain to return an Abstract Syntax Tree (AST) with nodes representing paragraphs, sentences, and words. Understanding where a sentence begins and ends makes the phrase optimizer easier to implement and avoids the spanning sentence scenario.

The semantic layer is responsible for finding structure and proposing one or more potential versions of the tree. It is not until the logical parsing layer takes place that (1) contextual sensitive decisions and (2) final ambiguity can be reasoned.

## Logical Parsing

First consider the extracted fragment … useful as a baseball bat and … useful as a soccer bat. Both are semantically valid statements with soccer and baseball specialized sports. However, there is no such thing as a ‘soccer bat’ a detail not discoverable until logical parsing.

Then consider the definition class foo{foo(foo foo){...}}. This class ‘foo’ has a copy constructor that takes an argument named foo. The semantic parser might be able to determine that ‘foo’ is an identifier but what does it point too? That answer is very contextually sensitive and cannot be reliably answered until the logical parsing layer.

# BNF Grammar

Backus-Naur Form is a standardized method for describing tokens and their expected sequence. A traditional for-loop might be represented in BNF as figure 1.

## Tokens and Reusable Segments

Each <Token> maps to a collection of regular expressions that can stand for its presence. Tokens within the sequence are can be annotated as required, optional, or repeating.

Reusable segments can be encapsulated inside of methods and called by the compiler’s compiler. This reduces the complexity for adding new statement types and promotes better consistency across the language.

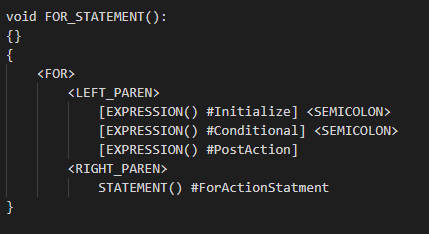


Figure 1: For-Loop