**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 an input based on a language’s grammar definition. This tokenization does not contain any context and relies on an external semantic parser to perform the next layer of transformation.

## Example: English Lex

An English lexical parser might be passed the text Hello, good day! and return tokens [{Word: Hello}, {Punctuation: ,}, {Word: good}, {Word: day}, {Punctuation: !}]. It is not the responsibility for Lex to know that “good” is an adjective nor the legitimacy of “cofveve”. The tokenization only understands that a word is a series of alphabetic characters.

Having a clear separation of these responsibilities allows for more specialized layers to operate on high-level token instead of raw characters. A consuming layer that looks for common phrases and can reduce the tokens [{Word: good}, {Word: day}] into [{Phrase: good day}].

The complexity required to reduce these 2 tokens is significantly less than the deriving meaning from 8 raw characters. If the logic was placed inside of the Lex, then it would quickly become unmaintainable due to the number of possible combinations. Consider the subtle differences between she shouted “Good! Day!” and … Good! Day or night he…. The first is match with superficial punctuation while the second spans different sentences. Yet both are the exact same 8-character sequence.

## Semantic Parsing

Semantic parsing attempts to go one step further and provide structural information about the token set. Perhaps the English Lex is modified to return a tree of paragraphs, sentences, and words. This handles the scenario where our words spanned phrases and need to be inspected as separate isolated units. Within the tree light contextual information can be derived such as nouns, verbs, and adjectives but there is no logical analysis yet.

An example within the tree might be the extracted segment … useful as a soccer bat versus … useful as a baseball bat. Both are semantically valid statements with soccer and baseball belonging to the same category and used as adjectives. The term bat could be replaced with ball and again both stay valid statements. However, there is no such thing as a ‘soccer bat’ a detail only logical parsing could discover.

## Logical Parsing

After the semantical parsing the logical parser can review the document structure, called an Abstract Syntax Tree (AST). Within the AST there can be branches that do not make logical sense or are ambiguous.

Consider the definition class foo{foo(foo foo){...}}. This valid class called foo with 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? The answer is very contextually sensitive and rich context is only available starting at this layer.

# BNF Grammar

Backus-Naur Form is a standardized method for describing Lex tokens and their permitted 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 must match for the token to be considered present. 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.

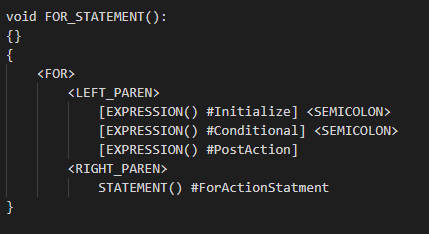


Figure 1: For-Loop