0.1 English language?

- Grammar, syntax - defines rules, the structure of the statement, not the meaning
- Meaning (semantics) - even if structure is correct, meaning may not be direct or correct
- We need to follow the rules, and get the correct meaning

1 Introduction

- Syntax and semantics provide a language's definition
- Syntax: the form or structure of the expressions, statements, and program units
- Semantics: the meaning of the expressions, statements, and program units
- A *sentence (or statement)* is a string of characters over some alphabet
- A *language* is a set of sentences
- A *lexeme* is the lowest level syntactic unit of a language (e.g., *, sum, begin)
- A *token* is a category of lexemes (e.g., identifier)
- Programs are string of lexemes rather than characters. E.g., *sum+=2;*

Lexemes and Tokens are closely related:

Lexemes	*Token*
sum	identifier
+	arthmetic operator
=	equal_sign
2	int_literal
;	semicolon

1.1 Recognizers

- A recognition device reads input strings over the alphabet of the language and decides whether the input string belong to the language
- Example: syntax analysis part of a compiler

1.2 Generators

- A device that generates sentences of a language
- One can determine if the syntax of a particular sentence is syntactically correct by comparing it to the structure of the generator

2 BNF and Context-Free Grammers

2.1 Context-Free Grammars

- Developed by Noam Chomsky in the mid-1950s
- Language generators, meant to describe the syntax of natural languages
- Define a class of languages called context-free languages

2.2 Backus-Naur Form

- Inveted by John Backus to describe the syntax of Algol58, later modified by Peter Naur for Algol 60
- BNF (Backus-Naur Form) is equivalent to context-free forms
- In BNF, abstractions are used to represent classes of syntactic structures;they act like syntactic variables, including nonterminal symbols or terminals
- *Terminals* are lexemes or tokens
- A **rule or production** has a left-hand side (LHS) which is a nonterminal, and a right hand side (RHS), which is a string of terminals and/or nonterminals
- Example:

$$< assign > \rightarrow < var > = < expression >$$

- Examples of BNF Rules:
  - $< ident\_list > \rightarrow identifier | identifier, < ident\_list >$
  - $< if\_stmt > \rightarrow if < logic\_expr > then < stmt >$
- A *start symbol* is a special element of the nonterminals of a grammar
- Rules can be recursive

3 Derivation

- A derivation is a repeated application application of rules, starting with the start symbol, repeat till ending with a sentence (all terminal symbols)
- Application of rules:
  - Pick a non-terminal symbol on the right, and replace the non-terminal symbol using a RHS of rule for the non-terminal symbol
  - Example:

$< start\_symbol > \rightarrow < program >$

$< program > \rightarrow$ **begin** $< stmt\_list >$ **end**

$< stmt\_list > \rightarrow < stmt > \mid < stmt >; < stmt\_list >$

$< stmt > \rightarrow < var > = < expression >$

$< var > \rightarrow A|B|C$

$< expression > \rightarrow < var > + < var > \mid < var > - < var > \mid < var >$

$$
\begin{aligned}
< program > &\rightarrow begin < stmt\_list > end \\
&\rightarrow begin < stmt >; < stmt\_list > end \\
&\rightarrow begin < var > = < expression >; < stmt\_list > end \\
&\rightarrow beginA = < expression >; < stmt\_list > end \\
&\rightarrow beginA = < var > + < var >; < stmt\_list > end \\
&\rightarrow beginA = B + < var >; < stmt\_list > end \\
&\rightarrow beginA = B + C; < stmt\_list > end \\
&\rightarrow beginA = B + C; B = Cend
\end{aligned}
$$

3.0.1 Example

Build a sentence using the following rules:

$$
\begin{array}{lcl}
< assign > & \rightarrow & < id >=< expression > \\
< id > & \rightarrow & \text{A} \parallel \text{B} \parallel \text{C} \\
< expression > & \rightarrow & < id > + < expression > \\
 & & < id > * < expression > \\
 & & (< expression >) \\
 & & < id >
\end{array}
$$

4 The General Problem of Describing Syntax

-

5 Formal Methods of Describing Language

-

6 Introduction

-

7 Introduction

-

8 Introduction

-