CS 350: Programming Language Design

Lecture 4:

Languages are structured systems of communication by use of sounds and gestures

Programming Languages are sets of instructions used for computers

Syntax and Semantics

Syntax: How we write our statements and sentences in languages

Semantics: How we assign meaning to parts of our statements and keywords used

It is important to describe languages since it is the identity of the language

ALGOL

* Concise
* Not easy to understand

These factors impacted level of acceptance

Challenges: diversity of users

* Initial evaluators, other language designers
* Implementers
* Users

Language Definitions

* Recognizers
  + A recognition device
  + Reads input strings over the alphabet of the language
  + Decides whether the input strings belong to the language
    - Accept/Reject
* Generators
  + A device which generates sentences of a language
  + Used to enumerate all the sentences of a language
    - Like a button
  + Compare a sentence to the generated structure to know if accepted

Formal Methods for Describing Syntax

Context-Free Grammars

* Developed by Noam Chomsky in mid-1950s
* Language generators, meant to describe syntax of natural languages
* Two grammar classes; context-free and regular

Backus-Naur Form (BNF)

* Invented by John Backus to describe ALGOL 58
* Revised by Peter Naur for 60 version
* Not accepted readily, but became standard
* BNF is equivalent to context-free grammars

Grammars are formal language-generation mechanisms

Grammar Fundamentals

Meta-Languages are languages which describe other languages

* BNF or grammar is a meta-language for programming languages, i.e the language of languages

Single words correspond to terminals

* Keywords: class, public, while, for; in languages like Java or C++
* Literals: 23, ‘d’, “String”
* Separators and delimiters: semicolons, commas, brackets, braces

All structures built on terminals correspond to non-terminals

Rules of Grammars

* Rules / Production has a left-hand side which is nonterminal and right-hand side which is terminals and/or non-terminals
* <Assign> -> <var> = <expression>
* Highlighted is RHS
* Nonterminal symbols
  + Enclosed in angle brackets
  + Act like syntactic variables (We can make var “x” or “apple”)
* Grammar: a finite non-empty set of rules

Multiple Rules

Some statements can have two similar starting methods, such as if-statements, when we describe them we can use the single line ( | ) to differentiate them yet keep them as related behavior.

<if\_stmt> -> if <statements>; |

If <statements> else <statements>

Variable-length lists in math are written using an ellipsis (…)

Syntactic lists are described using recursion

<ident\_list> -> identifier |

Identifier; <ident\_list>

This dual meaning is good for referencing the first item in the list or creating a list of identifiers for a set of declarations.

Derivations for Grammars

* A start symbol is a special element of the non-terminals of a grammar
  + Complete programming language -> <program>
* A derivation is a repeated application of rules, starting with the start symbol
  + The derivation continues until the sentential form contains no non-terminals
* A derivation may be either leftmost or rightmost
  + Leftmost derivations are those in which the leftmost non-terminal in each sentential form is the one that is expanded

Symbol ( => ) is read a “derives”

Sentential form: every string of symbols in a derivation

Generated sentence: a sentential form, consisting of only terminals or lexemes