COSC252: Programming Languages:

Basic Semantics: Environments, Names, Literals, Scope, …

Jeremy Bolton, PhD
Asst Teaching Professor

GEORGETOWN UNIVERSITY

Outline

I. VERY Basic Semantics
   I. Literals
   II. Operations

Parsing

- Our heartless / soulless parser is simply a recognizer
  - Identifies whether a sentence is in a language
    - Whether a code file abides by the rules of a programming language

- Parse errors can be identified and described at this stage.

- Observe: Many implementations of parsers, e.g. c++ compiler, can identify errors that are more closely related to semantics (as compared to syntax)
  - How is this done?

GEORGETOWN UNIVERSITY

Attribute Grammars

- Attribute Grammars are an extended form of a CFG that can account for "other rules" that can be determined statically, but cannot be accounted for using standard CFGs. (Knuth)
  - Compatibility and Reliability

- Examples (what types of errors can be identified statically, but not with a standard CFG):
  1. ID not in scope, not accessible
  2. Multiple definitions in same scope
  3. Type incompatibility
     - Example: function returns a float but a Node* is expected
  - These are errors that are not syntactic, but can be recognized statically (before runtime).

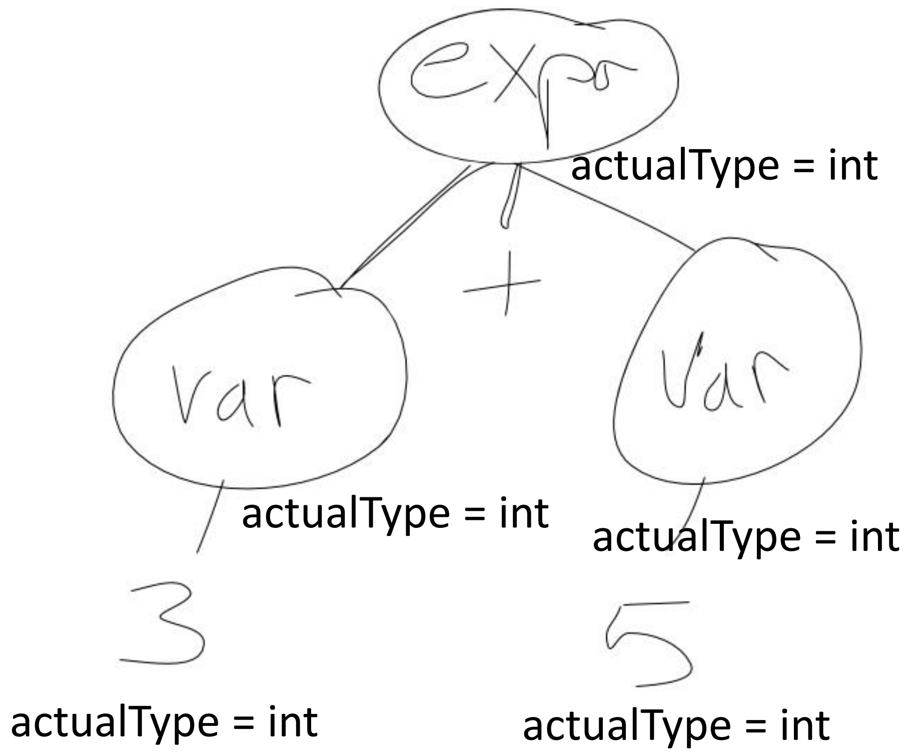
Static Semantics

- Static Semantics are "syntax" rules that are partially related to semantics.
  - "Static" as we can check the rules before runtime, during parsing
- Definitions
    - Attribute is a characteristic of a terminal or non-terminal
    - Semantic Rule Functions are associated with grammar rules
    - Predicate functions: state the static semantic rules associated with a grammar rule
- Attribute Grammar is a CFG with the following:
  - Attributes for a CFG symbol X, A(X)
  - Semantic Rule Function: for each rule in the grammar, $X_0 \rightarrow X_1 X_2 \ldots X_n$ a semantic rule S(X_0) computes the attributes of X_0 given the attributes of $X_1\, X_2 \ldots X_n$, $S(X_0) = f(A(X_0), A(X_1), \ldots, A(X_n))$.
  - Predicate function: is a Boolean expression on the attributes of a grammar. A false value of a predicate function implies that a static semantics rule has been violated.
- A parse tree with an attributed grammar may have attributes, semantic rules, and a predicate function associated with each node.
- If all the attribute values of a parse tree have been computed, the parse tree is said to be fully attributed.
- Intrinsic attributes: are attributes of terminals – leaf nodes in a parse tree

Example: Attribute Grammar

- Attribute grammar to test for compatibility
  - Attributes: expectedType, actualType
  - Grammar:

1. <expr> -> $< num_1 > + < num_2 >$
   1. Semantic Rule:
      if $< num_1 >.actualType == int$ && $< num_2 >.actualType == int$
         then $< expr >.actualType = int$
      else
         then $< expr >.actualType =$ other
   2. Predicate Rule:
      $< expr >.actualType == < expr >.expectedType$
2. <expr> -> <num>
   1. Semantic Rule: $< expr >.actualType = < num >.actualType$
   2. Predicate Rule: $< expr >.actualType == < expr >.expectedType$
3. <num> -> 0 | 1 | …| 9
   1. Semantic Rule: $< var >.actualType = int$

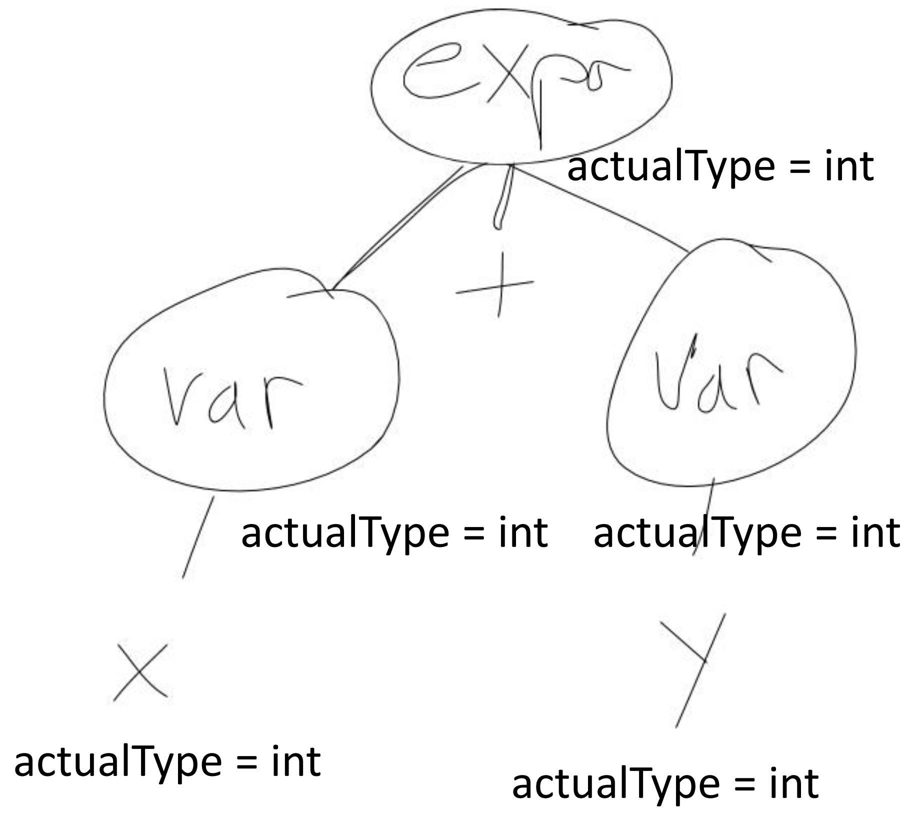


GEORGETOWN UNIVERSITY

Example: Attribute Grammar

- Attribute grammar to test for compatibility
  - Attributes: expectedType, actualType
  - Grammar:

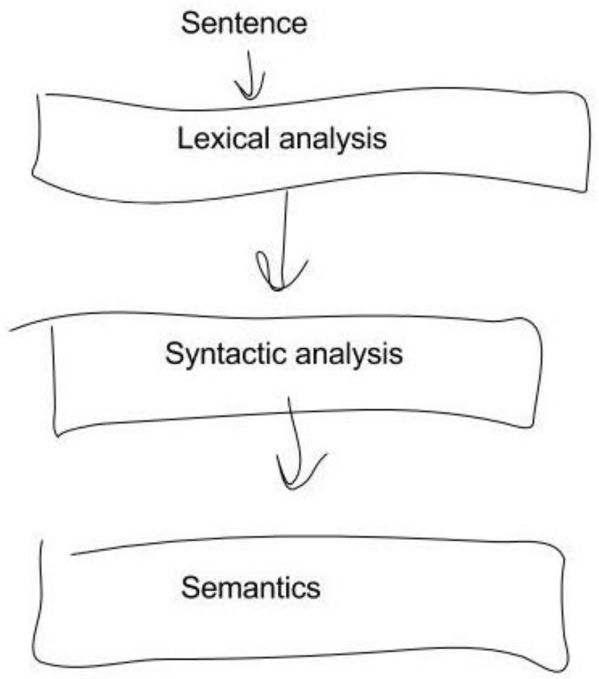
1. <expr> -> $< var_1 > + < var_2 >$
   1. Semantic Rule:
      if $< var_1 >.actualType == int$ && $< var_2 >.actualType == int$
          then $< expr >.actualType = int$
      else
          then $< expr >.actualType =$ other
   2. Predicate Rule:
      $< expr >.actualType == < expr >.expectedType$
2. <expr> -> <var>
   1. Semantic Rule: $< expr >.actualType = < var >.actualType$
   2. Predicate Rule: $< expr >.actualType == < expr >.expectedType$
3. <var> -> x | y | z
   1. Semantic Rule: $< var >.actualType = symbolTableLookup(< var >.lexeme)$



GEORGETOWN UNIVERSITY

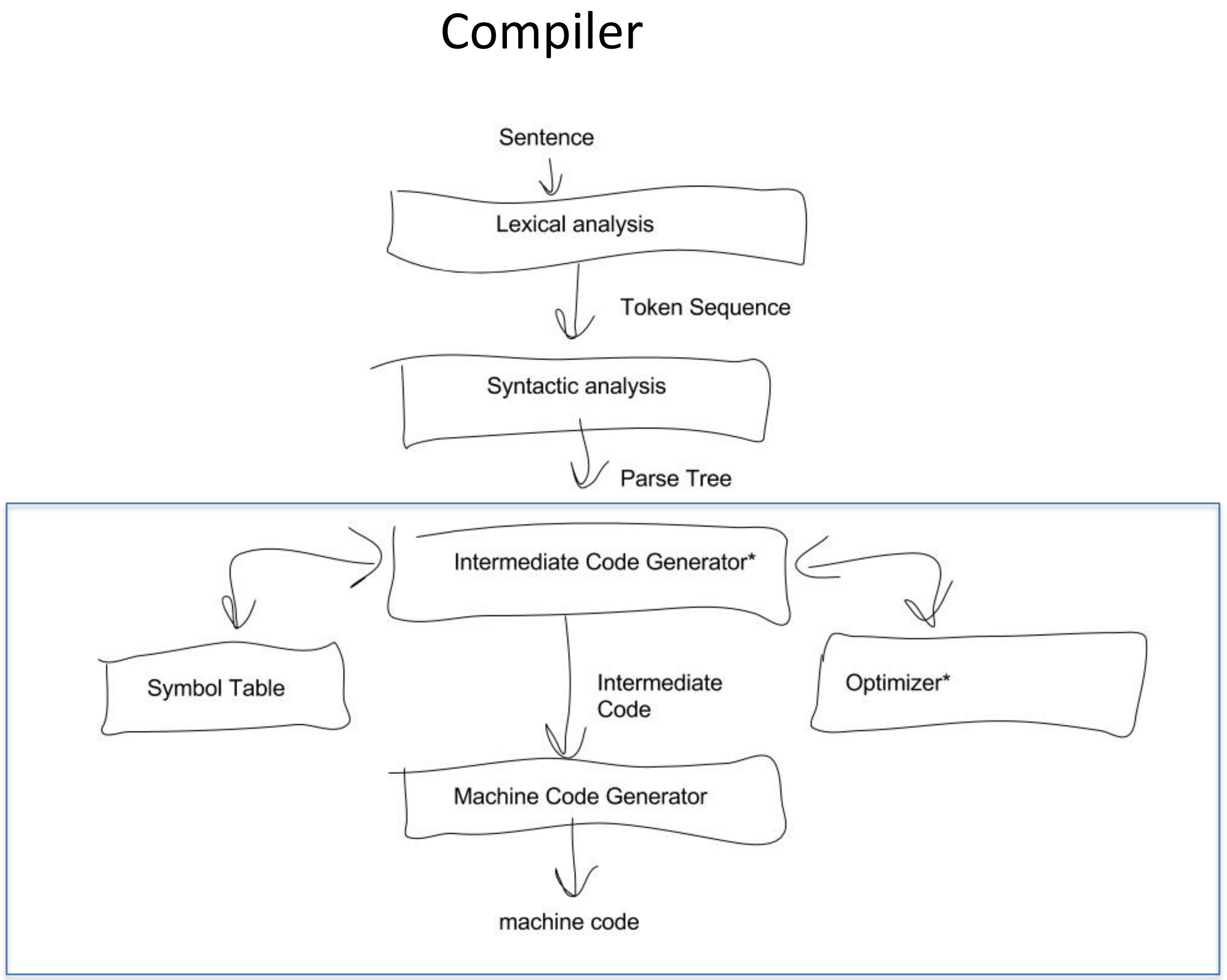
Dynamic Semantics

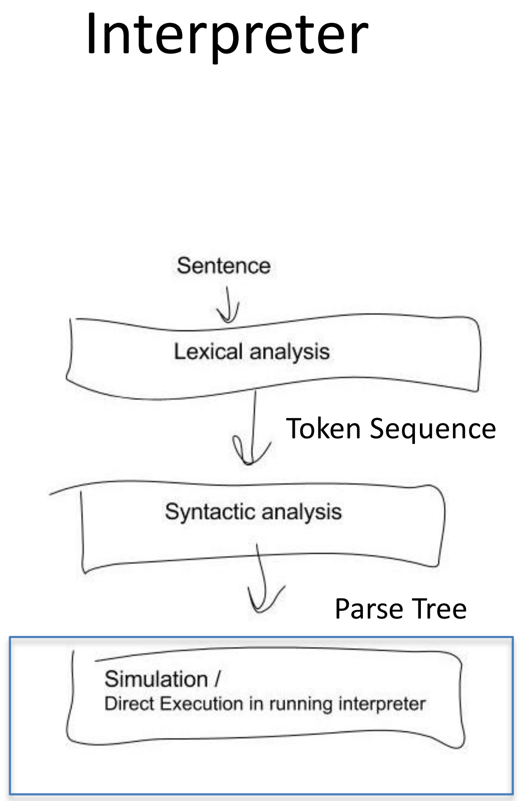
- After lexical and syntactic analysis, semantic analysis is performed
  - Application of *meaning* to an input sentence



GEORGETOWN UNIVERSITY

Semantics: Compiler vs Interpreter



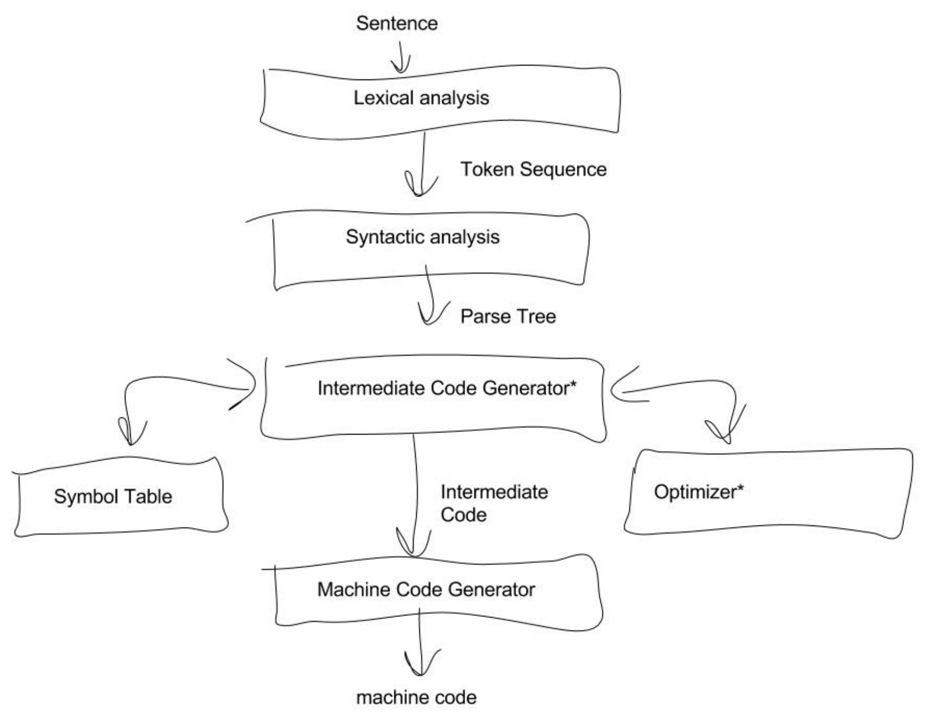


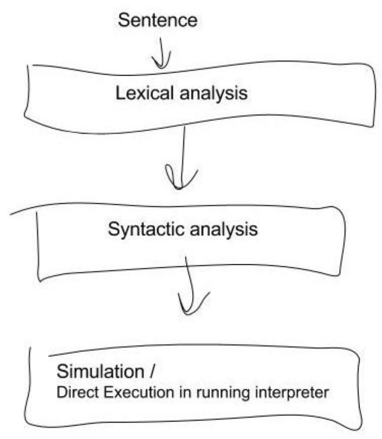
GEORGETOWN UNIVERSITY

Semantics at different levels of the computer abstraction

- Application of Semantics: Compiler vs. Interpreter
  - Compiler
    - A compiler often translates source code to machine code. Thus communicating at the lowest level of the abstraction.
    - Thus the application of semantics is *the creation of machine code* (to be performed by the computer)
  - Interpreter
    - The interpreter is a running program that "simulates" the execution of the source code. It interprets the commands in the source code and then directly performs those commands (or commands similar to those which are available).
    - Thus application of semantics is *the execution of interpreted commands.*

- *Examples to come …*





Application of Semantics

- Most programming languages allow programmers to interact with a computer at a highly abstract level.
  - The underlying machine has various low-level components: CPU, memory, …
  - Most programming languages operate at a higher level of the computing abstraction.
    - For example: the abstraction for a memory location is a variable.
    - Operating at this level of the abstraction conceals low level details from the programmer which eases the programming burden.
  - However, concept of semantics, is intrinsically defined in terms of the state of the machine.
    - Yes – the goal of a programming language is to communicate with a machine – give precise instruction.
      - Intuitively, the instruction must therefore be in
        1. In a format understandable to the machine
        2. Performable by the machine

Semantics of tokens

- Application of semantics is often referred to as *evaluation*.

- Example literal:
  - What is the semantics of “1”?
  - Conceptually the string “1” refers to the quantity 1.
  - Semantic application (evaluation) of “1” in a computer, refers to the binary representation of “1” in a computer.
    - Load Register with 0000 0000 …. 0001

- Example:
  - What is ‘c’?
  - Conceptually: the character c
  - Semantic application (evaluation) in a computer, refers to the binary representation of ‘c’

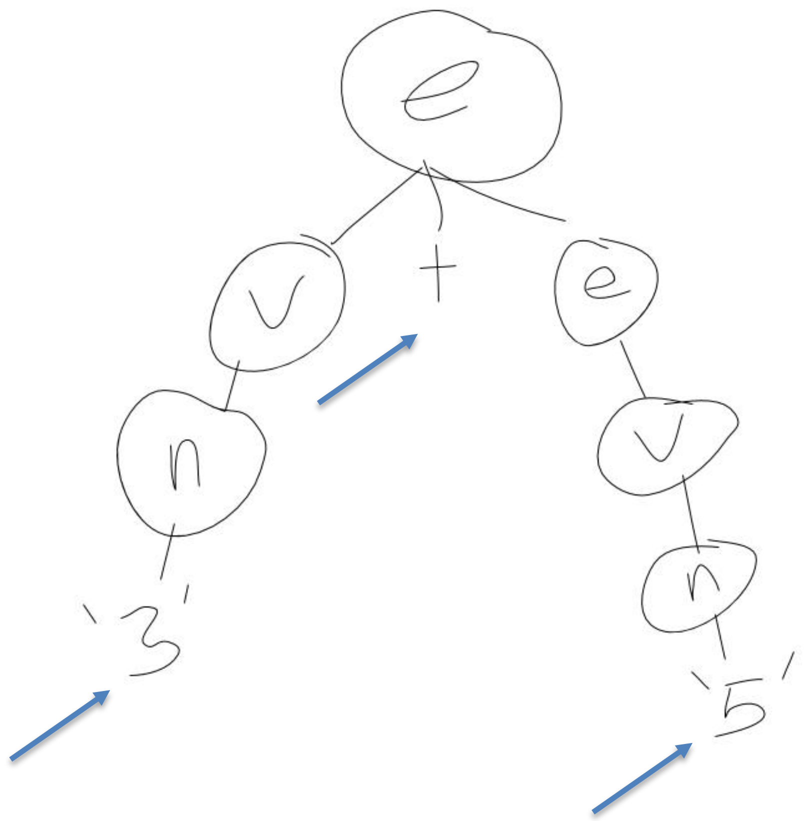
Type Binding

- A literal is *bound* to an attribute at *binding time*
  - Static binding: binding occurs before runtime
  - Dynamic binding: binding occurs during runtime

- C++ Example:
  - 0;
  - type of 0 is bound statically (at compile time).

Semantics: from lexemes to abstractions

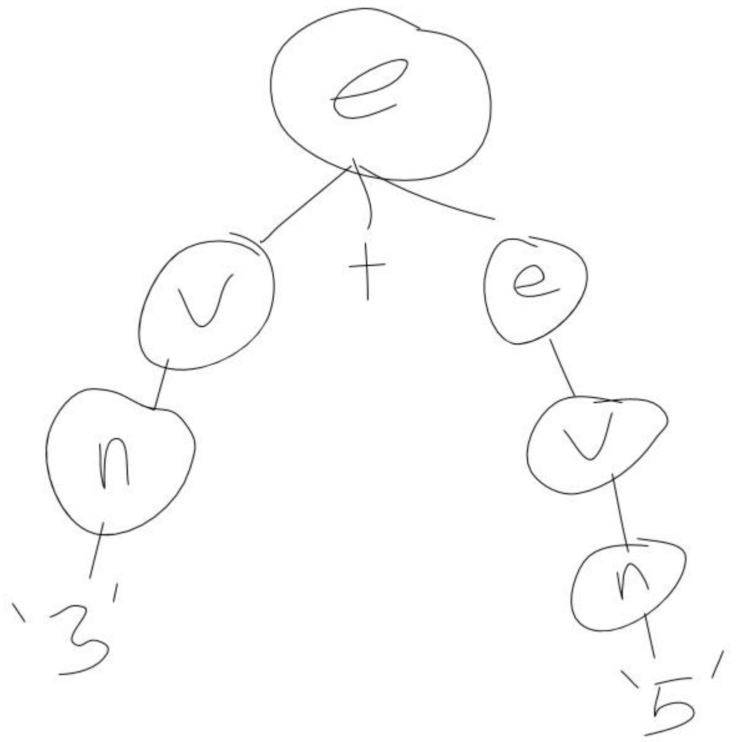
- In most languages
  - Each lexeme, syntactic unit, of a language has intrinsic meaning (semantics)
  - This semantics of an input sentence is generally determined in terms of the semantics of the lexemes of the input.
    - But How?
  - The application of semantics is driven by the BNF productions. This also implies that all characteristics of a language not specified by the BNF, must be specified in the application of semantics.
  - It is intuitive that semantics are applied based on BNF. Meaning is applied to the lexemes directly, and meaning is assigned to non-terminal constructs in terms of the meaning of its constituents
    - Meaning is propagated up the parse tree



GEORGETOWN UNIVERSITY

Semantics Example

- What is the meaning of “3 + 5”?
  - What is the meaning of “3” ?
    - Lexeme “3” is the 3 symbol.
    - Semantics of “3”: the number 3
    - Semantics in the context of
      Computer PL: binary rep of 3
  - What is the meaning of “+” ?
    - Lexeme “+” is the plus symbol.
    - Semantics of “+”: addition operation
    - Semantics in the context of
      Computer PL: a specific ALU operation
  - What is the meaning of “5” ?



GEORGETOWN UNIVERSITY

Semantic Specification

- Semantic Specification determines how meaning is applied to a sentence of a language
  - A universally standardized form of semantic specification does not exist, but there are 3 general categories
    - Operational Semantics: describes the semantics of a language in terms of the state of the underlying machine
    - Denotational Semantics: describes the semantics of a language in terms of functions defined on programs and program constructs
    - Axiomatic Semantics: Uses mathematical logic to formalize characteristics of a program.

- Properties of a good semantic specification
  - It must be complete. Each input program that abides by the syntax should have appropriate semantics as defined by the specification
  - It must be consistent. Each input program must not have two conflicting semantics.

Appendix

1789

GEORGETOWN UNIVERSITY