# CS 230
# Programming Languages

09 / 13 / 2022

Instructor: Michael Eckmann

# Today's Topics

- Questions? / Comments? --- from the reading?
- Describing the Syntax of a programming language
- Context Free Grammars
- BNF
- Generators vs. Recognizers
- Ambiguity
- Extended BNF

# Now Chapter 3 ...

- Describing a language
  - How to give it a clear and precise definition so that implementers (compiler writers) will get it right
  - How to describe it to users (programmers) of the language

# Syntax and Semantics

- Describing a language involves both its syntax and semantics
- Syntax is the form, semantics is the meaning
  - e.g. English language example:
    - Time flies like an arrow.
      - Syntactically correct but has 3 different meanings (semantics)
- Easier to describe syntax formally than it is to describe semantics formally

# Syntax

- A **language** is a set of strings (or sentences or statements) of characters from an alphabet.
  - e.g. the set of all the valid English *sentences* make up the English *language*

# Syntax

- Lexemes vs. tokens
  - tokens (more general) are categories of lexemes (more specific)
  - e.g. Some tokens might be: identifier, integerLiteral, additionOperator
  - e.g. Some lexemes might be: idx, 42, +

# Syntax

idx = 42 + count;

| *Lexemes* | *Tokens* |
| --- | --- |
| idx | identifier |
| = | assignmentOperator |
| 42 | integerLiteral |
| + | additionOperator |
| count | identifier |
| ; | statementTerminator |

# Syntax

- Recognizers and generators are used to define languages.
- Generators generate valid programs in a language.
- Recognizers determine whether or not a program is in the language (that is, is it valid syntactically?)

- Generators are studied in Chapter 3 (stuff coming up next in this lecture) and recognizers (parsers) in Chapter 4.

- How many valid programs are there for some particular language, say Java?

# Syntax

- Context Free Grammars (CFGs) developed by Noam Chomsky are essentially equivalent to Backus-Naur Form (BNF) by Backus and Naur.
- They are used to describe syntax.
- These are termed metalanguages (languages used to describe languages.)

# Syntax

- A Context Free Grammar is a four-tuple

(T, N, S, P) where

  - T is the set of terminal symbols
  - N is the set of non-terminal symbols
  - S is the start symbol (which is one of the non-terminals)
  - P is the set of productions of the form:
    - A -> $X_1$ . . . $X_m$ where
      - A element of N
      - $X_i$ element of N U T, 1 <= i <= m, m>=0

# Syntax

- How are CFGs used to describe the syntax of a programming language?
- The nonterminals are abstractions (they represent larger structures)
- The terminals are tokens and lexemes
- The productions are used to describe programs, individual statements, expressions etc. They describe how a nonterminal (abstraction) can be expanded.

# Syntax

- Example production:

**<while_stmt> → `while` ( <logic_expr> ) <stmt>**

- Everything to the left of the arrow is considered the left-hand side, LHS, and to the right the RHS.
- The only thing that can appear on the LHS is one nonterminal.
- Multiple RHS's for a LHS are separated by the | or symbol, e.g.

**<compound_stmt> → <single_stmt> ;**

**| { <stmt_list> }**

# Syntax

- Recursion is allowed in productions, e.g.

<ident_list> → `ident`
| `ident`, <ident_list>

# Syntax

- **An example grammar:**

<program> → <stmts>

<stmts> → <stmt>

| <stmt> ; <stmts>

<stmt> → <var> = <expr>

<var> → `a` | `b` | `c` | `d`

<expr> → <term> + <term>

| <term> - <term>

<term> → <var> | `const`

# Syntax

- **Derivations are repeated applications of production rules.**
- **An example derivation:**

<program> => <stmts>

=> <stmt>

=> <var> = <expr>

=> a = <expr>

=> a = <term> + <term>

=> a = <var> + <term>

=> a = b + <term>

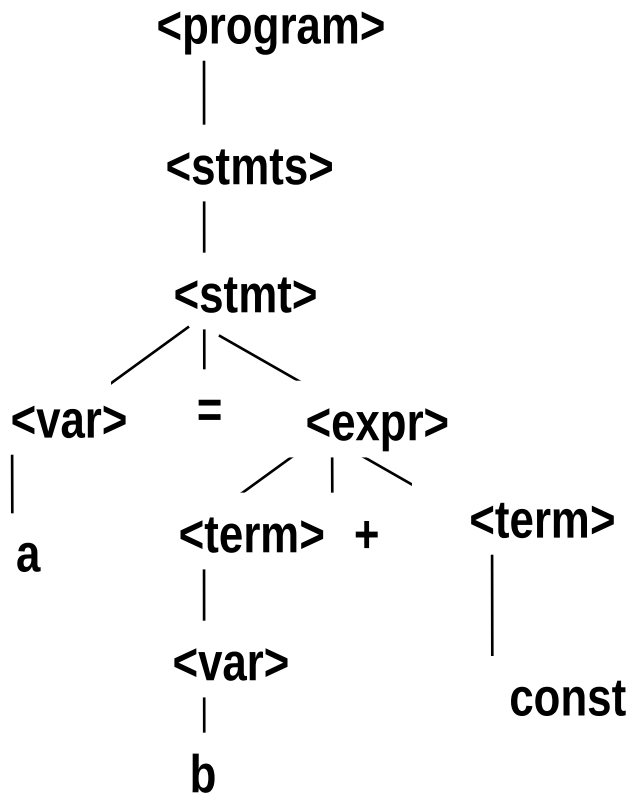
=> a = b + const

# Syntax

- Every string of symbols in the derivation is a *sentential form*
- A *sentence* is a sentential form that has only terminal symbols
- A *leftmost derivation* is one in which the leftmost nonterminal in each sentential form is the one that is expanded in each step of the derivation.
- A *rightmost derivation* is what?

# Syntax / Parse Trees

- A hierarchical representation of a derivation



# Syntax / Parse Trees

- What are at the leaves?
- What are at the internal nodes?
- What's at the root?
- Does a parse tree imply a particular order of a derivation (eg. Leftmost, rightmost, niether)?

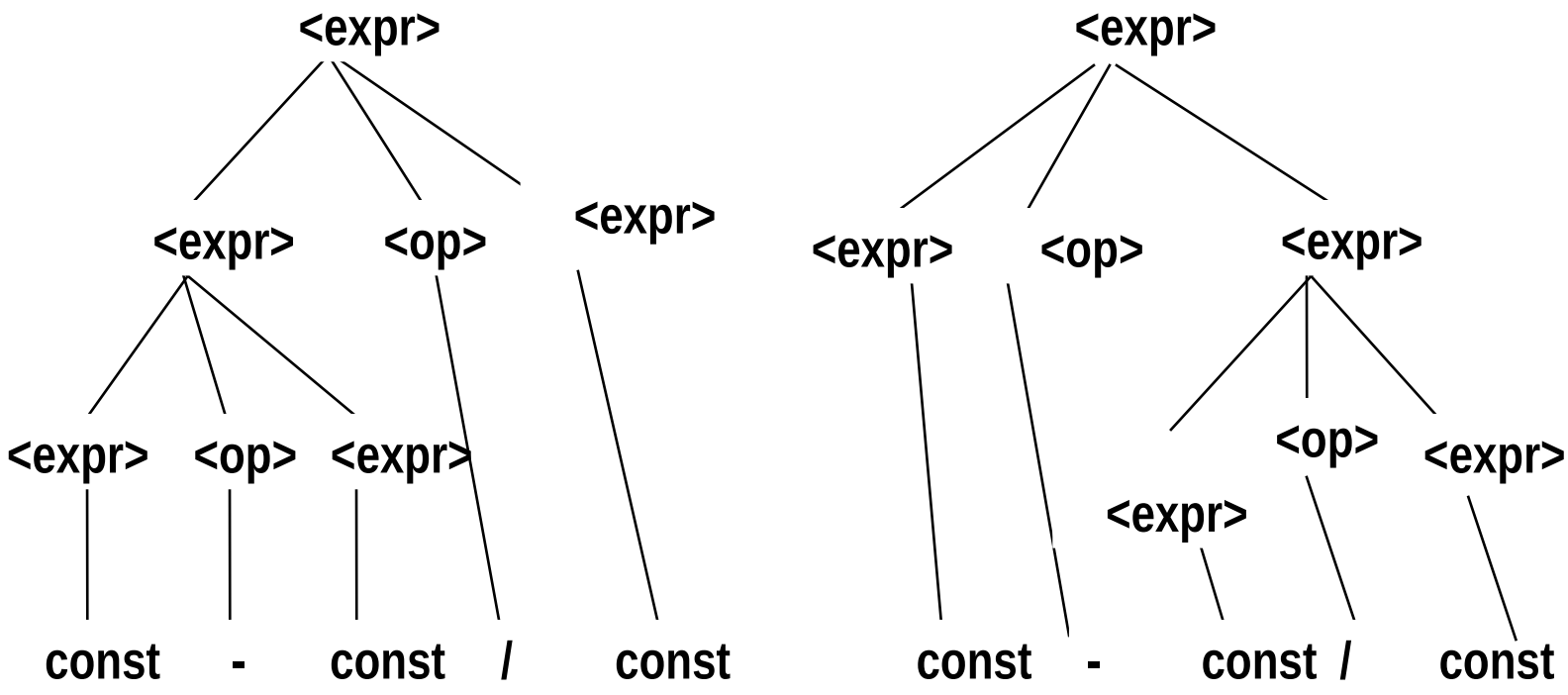
# Ambiguity

- A grammar is ambiguous if one can generate a sentential form for which there is more than one parse tree.
- Compilers typically base the semantics of the language on their syntactic form. A compiler chooses the code to be generated for a statement from its parse tree.

# An Ambiguous Expression Grammar

<expr> → <expr> <op> <expr> | const

<op> → / | -



# Ambiguous

- Ambiguity is not able to be handled by compilers, so the language description must be unambiguous so the compiler can be written

- The compiler examines the parse tree to determine the code to generate. Two parse trees for the same syntax will cause the meaning (semantics) of the code to not be unique.

# Ambiguous?

- Look at the if statement rules below

**<if_stmt> → if <logic_expr> then <stmt>**

**| if <logic_expr> then <stmt> else <stmt>**

**<stmt> → <if_stmt> | ...**

- Do you think this is ambiguous? That is, can more than one parse tree be generated from the same code?
- if (a==b) then if (c==d) then print_something() else print_something_else()

# Ambiguous?

```
if (a==b) then if (c==d) then print_something() else
    print_something_else()
```

```
if (a==b) then
      if (c==d) then
           print_something()
      else
          print_something_else()
```

```
if (a==b) then
    if (c==d) then
        print_something()
else
    print_something_else()
```

# Ambiguous?

- To make it unambiguous take a look at page 125 in our text.
- New Nonterminals are added and the other productions are changed in order to remove the ambiguity.

# Extended BNF

- Common extensions to BNF include:

- Use of square brackets [ ] to enclose optional parts of RHS's. That is, the part in the brackets is used 0 or 1 time.
- Use of braces { } to enclose parts of RHS's that can be repeated indefinitely or left out. That is, the part in the braces is repeated 0 or more times.
- Use of parentheses ( ) around a group of items of which one is chosen. The items are separated by |.

# Extended BNF (EBNF)

- It should be obvious that these new symbols are not terminal symbols in the language being described nor are they non-terminals. These new symbols are part of the EBNF meta language.
- If the language being described by the grammar does require brackets, braces or parentheses as terminal symbols (as many languages do) they have to be denoted in some way like underlining them to differentiate them from the EBNF symbols.

- What good are these extensions?

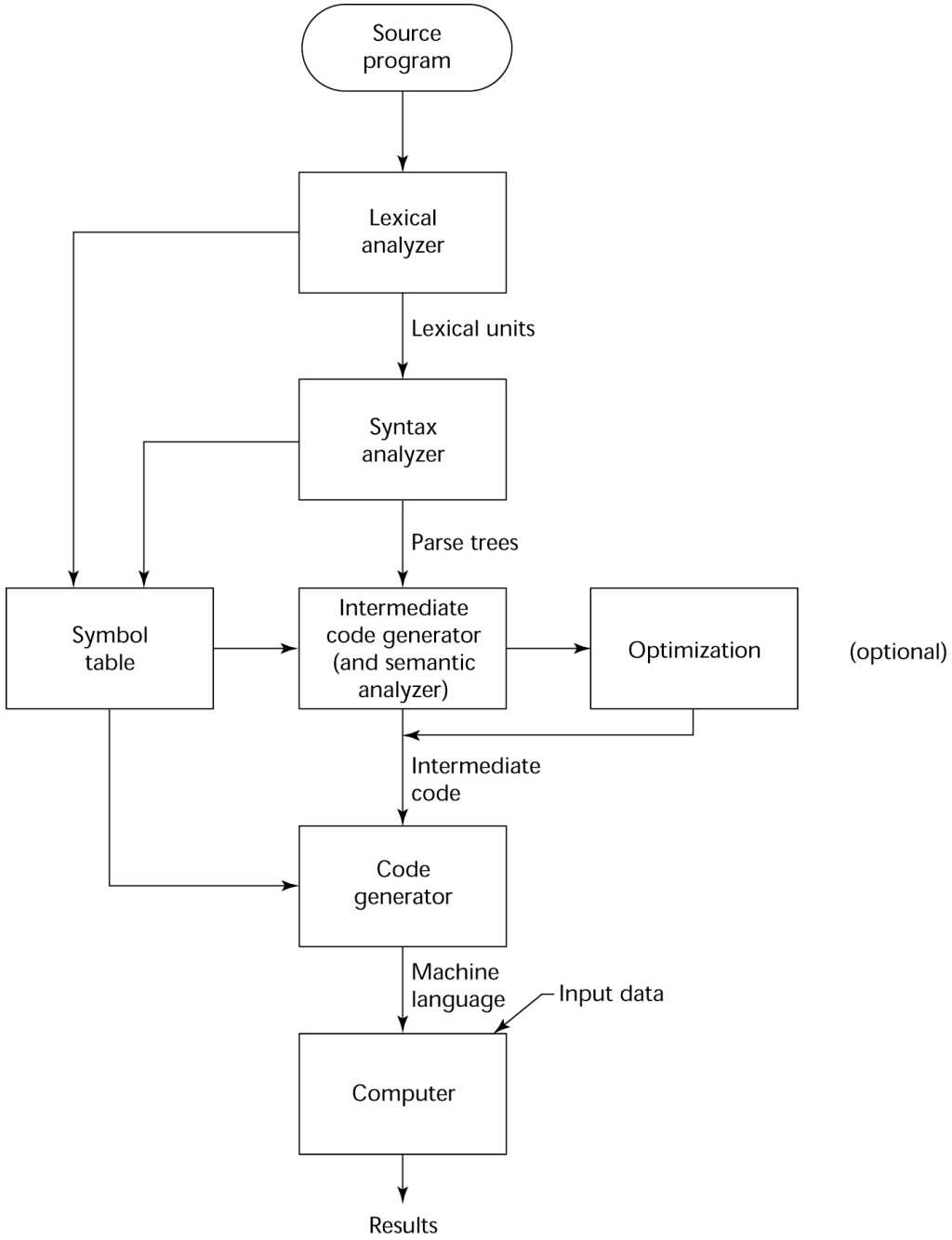
# Extended BNF

- What good are these extensions?
  - They increase writability and readability of the BNF metalanguage. Often lead to fewer non-terminals and fewer productions.
  - They DO NOT enhance the descriptive power of BNF --- that is, they do not allow languages to be described that couldn't have been described without them.

# CFG's and Recognizers

- Given a formal description of a language, a recognizer (syntax analyzer, aka parser) for that language can be algorithmically constructed. Therefore a program can be written to do this. yacc is an example of one.

# Let's Revisit the Compilation Process




Michael Eckmann - Skidmore College - CS 230 - Fall 2022

# Compilation Process

- Lexical analyzer – groups the code into variable names, reserved words, punctuation, etc.
- Syntax analyzer – constructs parse trees to determine if the lexical units in order are syntactically correct
- Both create items in the **symbol table** to refer to these units
- Intermediate code generator (and semantic analyzer) generates assembly-like code from the symbol table and parse trees.

# Chapter 3 so far

- Generator / recognizer
- Is a grammar a generator or recognizer, do you think?
- CFG and BNF are what?
- CFG and BNF are used for what?
- Derivation for a sentence (program)
- Parse tree for a sentence (program)
- What is ambiguity and why is it bad?