CS 3210
Principles of PL

Lesson 02

Agenda (1st half)

- Review
- PL Abstraction Levels
- PL Evaluation

Agenda (2nd half)

- The Compilation Process
- The Linking Process
- The Interpretation Process
- The Hybrid Process
- Describing Syntax

PL Abstraction Levels

- Machine
- Assembly
- System
- High-level
- Visual

PL Evaluation

- What makes a good PL?

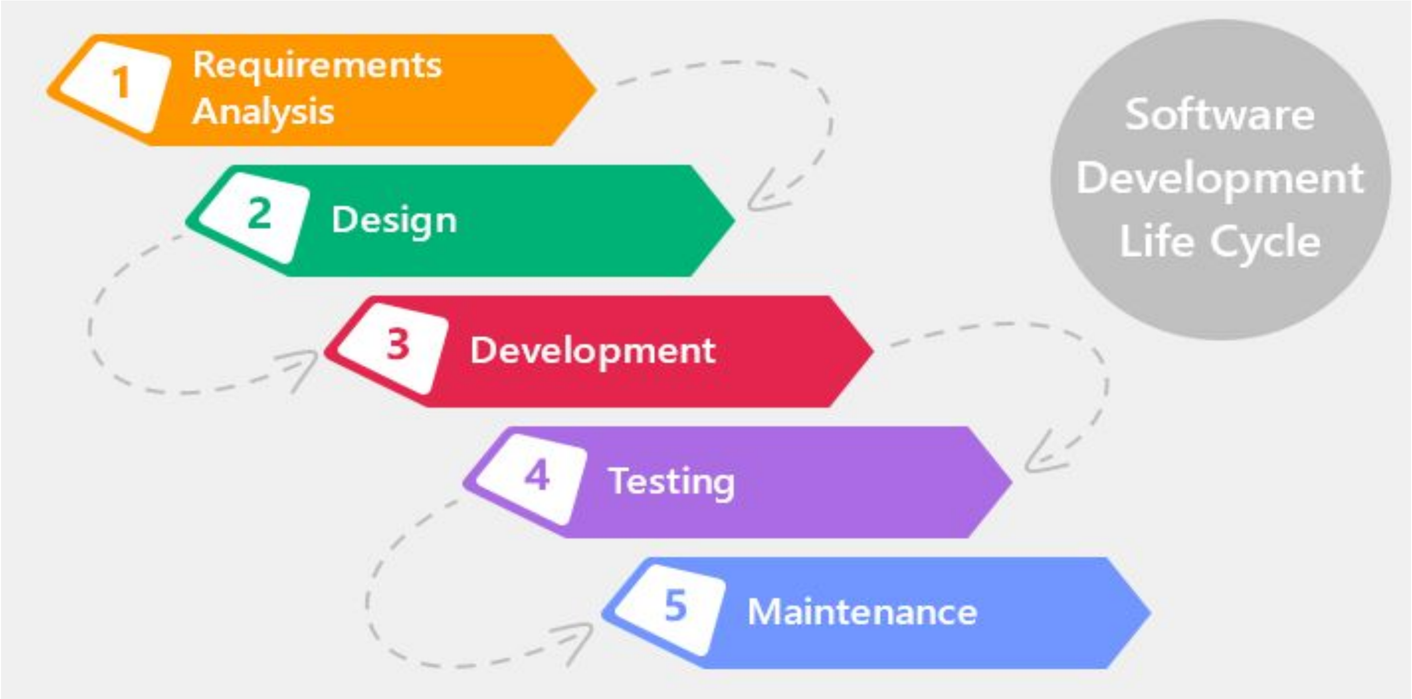
PL Evaluation

Table 1.1 Language evaluation criteria and the characteristics that affect them

Characteristic	CRITERIA: READABILITY	WRITABILITY	RELIABILITY
Simplicity	•	•	•
Orthogonality	•	•	•
Data types	•	•	•
Syntax design	•	•	•
Support for abstraction		•	•
Expressivity		•	•
Type checking			•
Exception handling			•
Restricted aliasing			•

PL Evaluation

- Readability and the software crisis of the 70s



PL Evaluation

- Writability:
  - measures how easy a PL can be used to create programs for a chosen problem domain

PL Evaluation

- Reliability:
  - a program is said to be reliable if it performs to its specifications under different conditions (e.g., different platforms, inputs, etc.)
  - the easier a program is to write, the more likely it is to be correct
  - programs that are difficult to read are difficult both to write and modify

PL Evaluation

Table 1.1 Language evaluation criteria and the characteristics that affect them

Characteristic	CRITERIA		
	READABILITY	WRITABILITY	RELIABILITY
Simplicity	•	•	•
Orthogonality	•	•	•
Data types	•	•	•
Syntax design	•	•	•
Support for abstraction		•	•
Expressivity		•	•
Type checking			•
Exception handling			•
Restricted aliasing			•

PL Evaluation

- Simplicity:
  - # of constructs
    - Large PLs often end up having overlooked features
      - Example:
        - Java’s “Double Brace Initialization” is a feature that allows writing expressions to create and initialize collections

PL Evaluation

```
@Test
public void whenInitializeSetWithDoubleBraces_containsElements() {
    Set<String> countries = new HashSet<String>() {
        {
            add("India");
            add("USSR");
            add("USA");
        }
    };

    assertTrue(countries.contains("India"));
}
```

PL Evaluation

- Simplicity:
  - # of constructs
    - Large PLs often end up having overlooked features
      - Example:
        - Naming slices in Python

```python
a = [0, 1, 2, 3, 4, 5]
last3 = slice(-3, None)
print(a[last3])
```


METROPOLITAN STATE UNIVERSITY OF DENVER
LIVES TRANSFORMED

PL Evaluation

- Simplicity:
  - Feature Multiplicity
    - having more than one way to accomplish a particular operation

```
count = count + 1
count += 1
count++
++count
```

PL Evaluation

- Simplicity:
  - Operator Overloading
    - When a single operator has more than one meaning
    - Example:

      “+” being used for both addition and concatenation

PL Evaluation

- Simplicity:
  - Operator Overloading
    - How “+” works in Python when operands are lists?
    - How about when one operand is a list and the other is a scalar value?

```python
a = [0, 1, 2]
b = [3, 4, 5]
c = a + b
print(c)

d = a + 1
print(d)
```

METROPOLITAN STATE UNIVERSITY℠ OF DENVER
LIVES TRANSFORMED

PL Evaluation

- Simplicity:
  - Simple languages are easier to learn
  - It should not be understood as less powerful
  - Be careful not to carry the concept too far
  - Example:
    - Most forms of assembly language are very simple and hard to read at the same time

PL Evaluation

- Orthogonality:
  - a language is orthogonal if its features are built upon a small, mutually independent set of primitive operations
  - in other words, language features can be used in any combinations <u>that make sense</u>
  - orthogonal languages are conceptually simple, having fewer exceptional rules

PL Evaluation

- Orthogonality:
  - Examples of lack of orthogonality in C:
    - **arrays** and **structs** are the only two structured data types available in C
    - **structs** can be returned from functions
    - **arrays** cannot

PL Evaluation

- Orthogonality:
  - Examples of lack of orthogonality in C:
    - **void** is a type in C but you cannot use **void** in all contexts like the other types
    - for example, you cannot define a pointer to **void** or use **void** to define a field in a **struct**

PL Evaluation

- Orthogonality:
  - Examples of lack of orthogonality in C:
    - **array** variables cannot be assigned to other **array** variables

PL Evaluation

- Orthogonality:
  - Examples of lack of orthogonality in Java:
    - there are two type categories in Java: *primitive* and *user-defined* (class types)
    - there is no mechanism that allow the creation of primitive types

PL Evaluation

- Orthogonality:
  - Most of the times PL designers trade-off orthogonality for efficiency
  - Also, too much orthogonality can negatively influence writability if unforeseen combinations of constructs lead to code absurdities hard to be detected by the compiler

PL Evaluation

- Data Types:
  - As mentioned before, a data type defines a collection of data values and a set of predefined operations on those values
  - Lack of types for commonly used values certainly impacts a PL readability, writability, and reliability

PL Evaluation

- Syntax Design:
  - syntax is the form while semantic is the meaning
  - in a well designed PL, semantics should follow directly from syntax

PL Evaluation

- Syntax Design:
  - Example:
    - **static** in C has different semantics depending on the context of its appearance
      - if used inside a function it means that a variable is to be defined at compile time (in contrast with **auto** variables)
      - if used outside all functions it means that a variable shall not be exported from the file where it was created

PL Evaluation

- Syntax Design:
  - the words chosen to be included in a PL grammar directly affect its readability
  - for example, most PL use the same set of words to represent common statements, such as **if**, **while**, **for**, etc.

PL Evaluation

- Syntax Design:
  - another important PL design choice refers to the mechanisms used to define compound statements
  - some PLs use matching pair of special words or symbols to form groups
  - others use braces (Java) or tabs (python)

PL Evaluation

- Expressivity:
  - computations can be expressed using powerful operators
  - it measures how simple it is to express an idea (write a program) using a PL

PL Evaluation

- Expressivity:

Java

```java
public class Main {
  public static void main(String[] args) {
      System.out.println("hello world");
    }
}
```

Python

```python
print("hello world");
```

PL Evaluation

- Type Checking:
  - testing for type errors in a given program, either by the compiler or during program execution

PL Evaluation

- Type Checking:
  - Example:
    - early versions of the original C language didn’t require parameters to functions to be type checked
    - for example, an `int` could be passed to a function that expected a `float`, with unpredictable results (see Ariane 5 explosion)

PL Evaluation

Metropolitan State University of Denver
Lives Transformed

PL Evaluation

- Exception Handling:
  - the ability of a program to intercept run-time errors and take corrective measures
  - this features is common in most languages today, but older languages, such as C, don't provide explicit mechanism for exception handling

PL Evaluation

- Aliasing:
  - the ability to use distinct names to refer to the same memory location
  - it can be dangerous because it can lead to errors difficult to trace, making the PL less reliable
  - Example: > 1 reference to an object

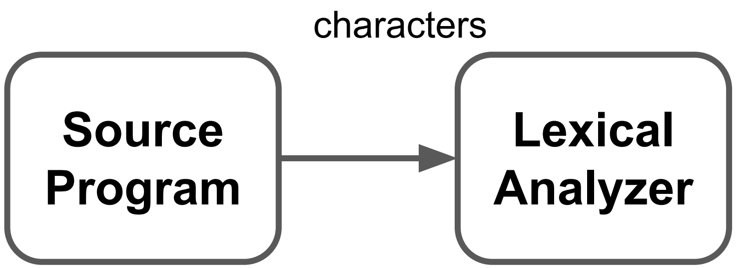
Agenda (2nd half)

- The Compilation Process
- The Linking Process
- The Interpretation Process
- The Hybrid Process
- Describing Syntax

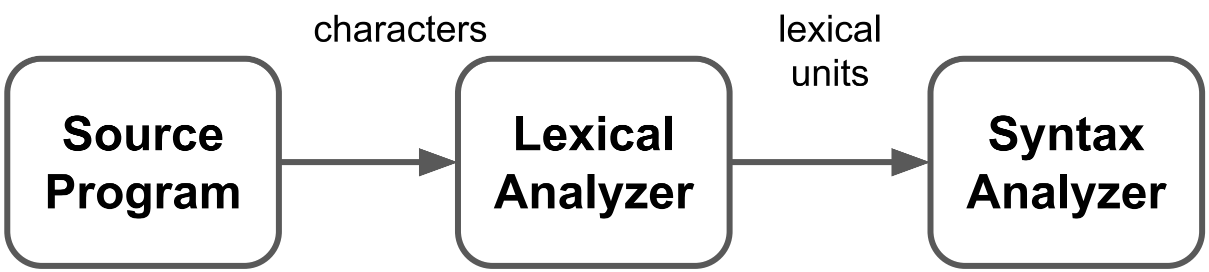
The Compilation Process

Source Program

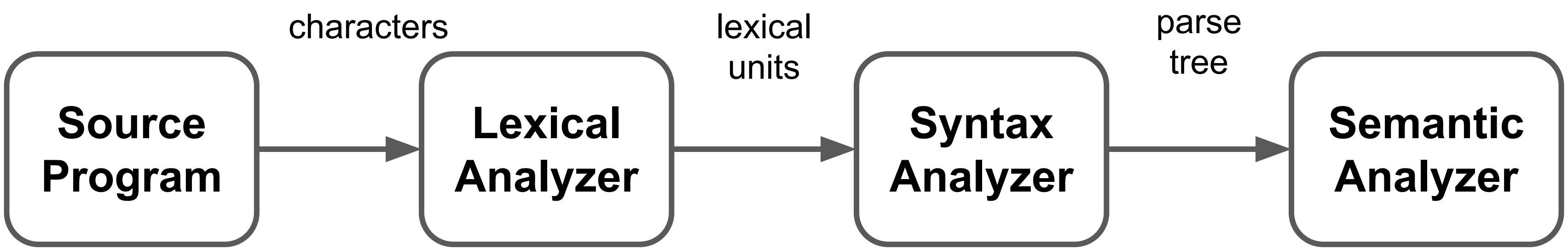
The Compilation Process



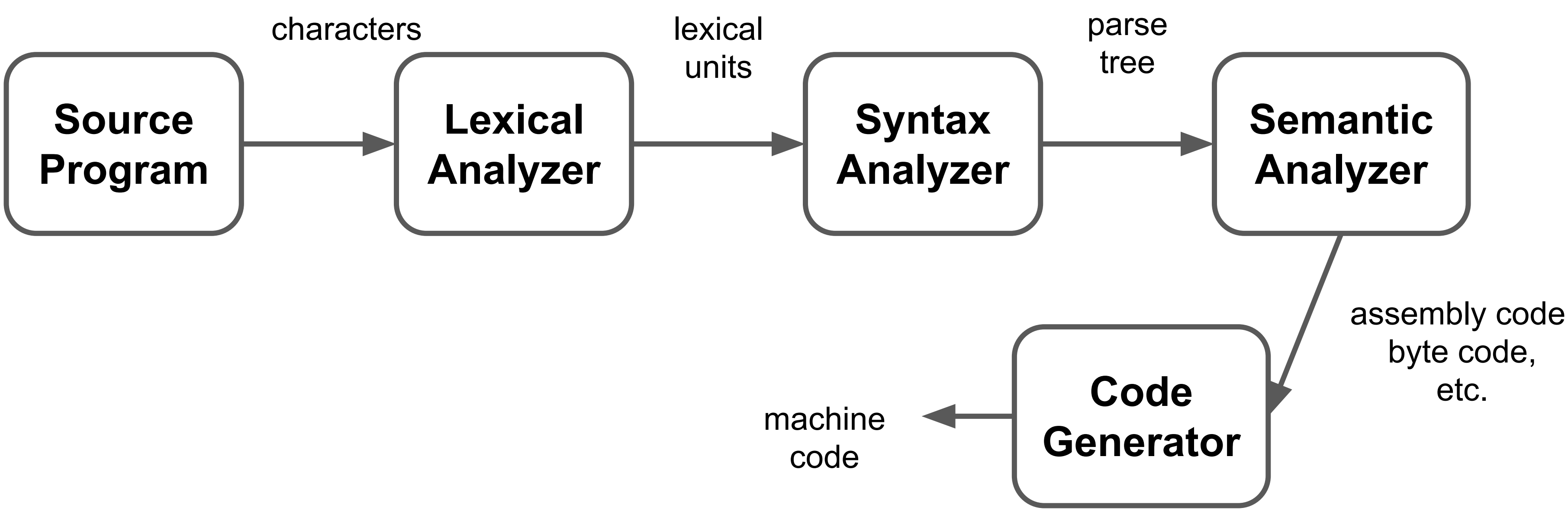
The Compilation Process



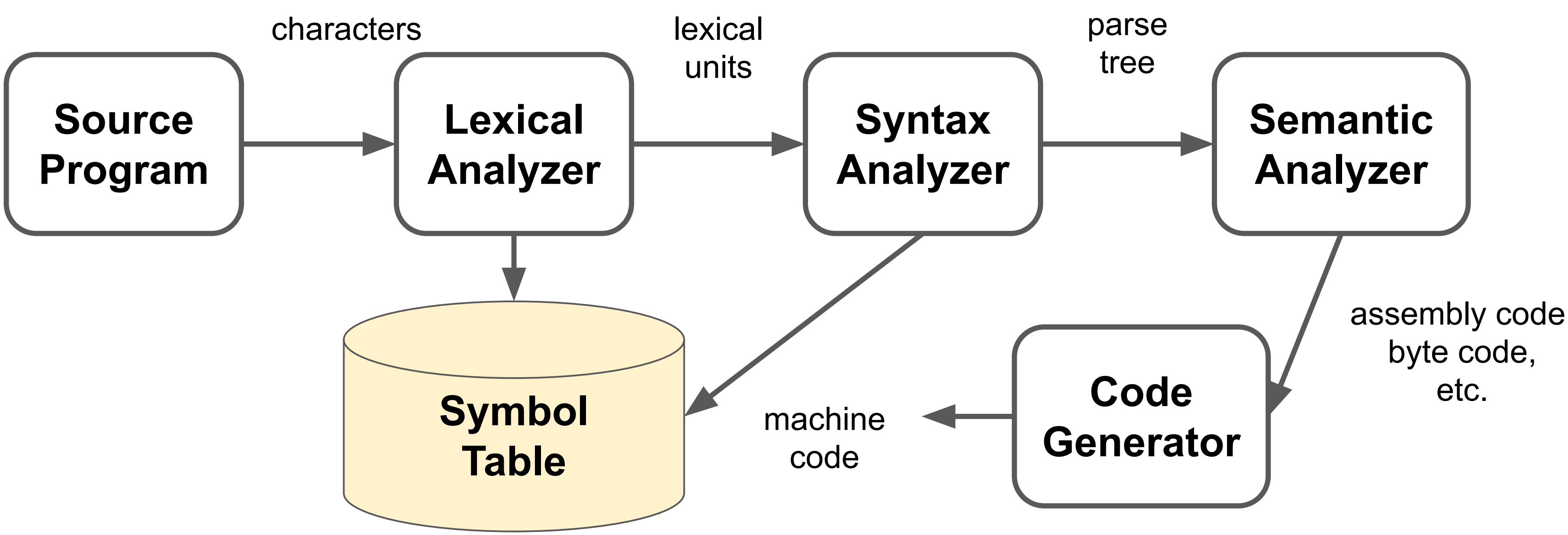
The Compilation Process



The Compilation Process



The Compilation Process



The Linking Process

- Programs depend heavily on low-level system calls
- They also require other external programs from supporting libraries
- Those extra pieces of code need to be linked to the user program

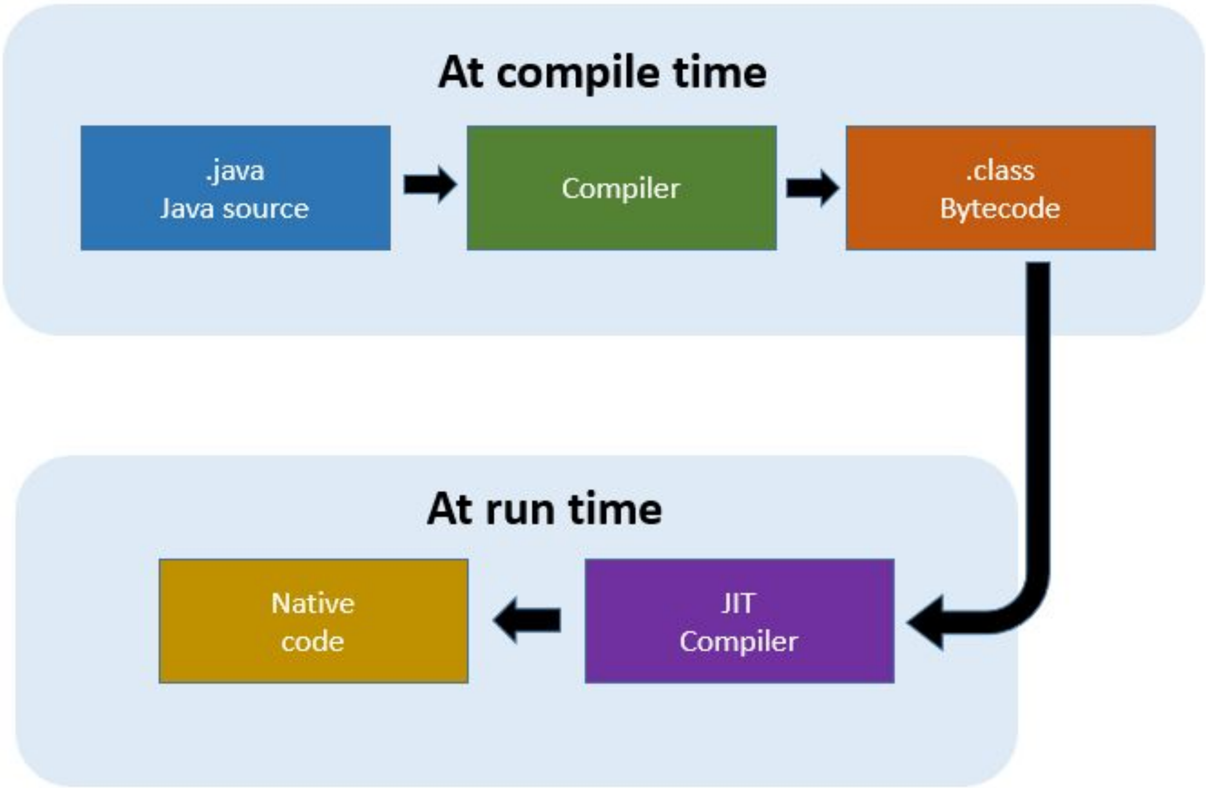
The Interpretation Process

- Each statement is decoded while the program runs
- The interpreter acts like a virtual machine
- It runs 10 to 100 times slower compared to compilation

The Hybrid Process

- The intermediate code (not the source code) is interpreted
- In other words, the source code is partially compiled
- The code generator is replaced by an interpreter
- Examples: Perl, Java, Python, and others

The Hybrid Process



Describing Syntax

- Alphabet:
  - A finite set of symbols
  - We like to use ∑ to represent alphabets
- Word:
  - A finite sequence of symbols from a given alphabet
  - If w is a word, |w| denotes its length
  - ε denotes the empty symbol

Describing Syntax

- Example:

$\sum = \{a, b\}$

a is a word from $\sum$ and $|a| = 1$

ab is another word from $\sum$ and $|ab| = 2$

bbbb is another word from $\sum$ and $|bbbb| = 4$

Describing Syntax

- Formal Language:
  - A set of words from an alphabet
  - Example:

$\sum = \{a, b\}$

L = {a, ab, bbbb} is a formal language from $\sum$

Describing Syntax

- Grammar:
  - G = (V, T, P, S), where:
    - V is a finite set of variables
    - T is a finite set of terminal symbols
    - P is a finite set of production rules
    - S is the start variable

Describing Syntax

- Example:

$G = (V, T, P, S)$

$V = \{S, D\}$

$T = \{0, 1, 2, \ldots, 9\}$

$P = \{S \rightarrow D, S \rightarrow DS, D \rightarrow 0|1|\ldots|9\}$

Describing Syntax

- Generated Language:
  - Given a grammar G, the generated language from G, denoted as L(G), is the set of all words that can be derived from S (start symbol) using only G's productions

Describing Syntax

- Example:

$G = (V, T, P, S)$

$V = \{S, D\}$

$T = \{0, 1, 2, \ldots, 9\}$

$P = \{S \rightarrow D, S \rightarrow DS, D \rightarrow 0|1|\ldots|9\}$

$L(G) = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11, \ldots\}$

Describing Syntax

- Left Linear Grammar:
  - G = (V, T, P, S)
  - A ϵ V, B ϵ V
  - w ϵ T*
  - G is left linear if all productions P have the form:
    - A → Bw | w

Describing Syntax

- Example:

$G = (V, T, P, S)$

$V = \{S, A\}$

$T = \{a, b\}$

$P = \{S \rightarrow Aaa \mid Abb,\ A \rightarrow Aa \mid Ab \mid \varepsilon\}$

Describing Syntax

- Right Linear Grammar:
  - $G = (V, T, P, S)$
  - $A \in V$, $B \in V$
  - $w \in T^*$
  - G is right linear if all productions P have the form:
    - $A \rightarrow wB \mid w$

Describing Syntax

- Example:

$G = (V, T, P, S)$

$V = \{S, A\}$

$T = \{a, b\}$

$P = \{S \rightarrow aS \mid bS \mid A, A \rightarrow aa \mid bb\}$

Describing Syntax

- Regular Grammar:
  - A left (or right) linear Grammar
  - If G is regular then we say that L(G) is regular

For Next Class

- Finish reading chapter 1
- Do Homework 1