Programming Language Principles

Programming Language Theory

In This Week

- 2 Lectures + 1 Assignment
- This week’s assignment is not just for attendance.
- It is also assessed and reflected on your score (5%).
- The assignment will be posted on **Sep 18 Fri**, and due date will be on **Sep 29 Tue**, which is before the holidays.
- You need to answer several questions about Turing machine and context-free grammar.

This Week's Topics

- **Syntax, Semantics, Pragmatics**
- **How to define a language mathematically?**
  - **Formal Language Basics**
  - Backus-Naur Form (BNF)
  - Context-free Grammar (Syntax)
  - Parsing and Ambiguity

Syntax vs. Semantics vs. Pragmatics

- ***Syntax*** is about the ***form*** of programs.
- ***Semantics*** is about the ***meaning*** of programs.
- ***Pragmatics*** is the meaning of programs in a certain context.

Syntax vs. Semantics vs. Pragmatics

- ***Syntax***:
  - A mouse is kicking a cat. → **OK!**
  - mouse a cat is a kicking. → **Wrong!**
- ***Semantics***:
  - A mouse is kicking a cat. → Ah, wait…
- ***Pragmatics***:
  - The mouse is Jerry and the cat is Tom. → Aha! it's possible.

Focus on Syntax

- Among the three, we're more interested in ***Syntax*** in this course.

- Before discuss about the others, we need to know how to define a programming language first.

- To talk about the meaning of a program, we first need to say what is a correctly written program.

Formal Language

- At this point, we're more interested in how to define the form of a program.
- How can we determine what is right or wrong for a PL?
- How can we decide whether given code is correctly written in a PL?

Formal Language

- A ***formal language*** is an abstraction of the general characteristics of programming languages.

- A formal language consists of a *set of symbols* and some *rules of formation* by which these symbols can be combined.

Terms

- **alphabet** Σ is a finite, nonempty set of symbols.
- A **string** is a finite sequence of symbols from the alphabet.
- e.g.) if the alphabet Σ = { a, b },

  then *aabb*, *abab*, *aabbb* are strings on Σ.
- Or we can say a string *w* = abba.

Terms

- **Concatenation** of two strings *w* and *v*.
    - $w = a_1a_2a_3\ldots a_n$
    - $v = b_1b_2b_3\ldots b_m$
    - concatenation of *w* and *v*:
        - $wv = a_1a_2\ldots a_nb_1b_2\ldots b_m$
- a^n: aaa...a, n times

Terms

- An empty string:
  - ε
- Length of a string w:
  - $|w| = n$
- Reverse of a string w:
  - $w^R = a_n \ldots a_2 a_1$

Terms

- Kleene Star *: >= 0
  - V* is a set of all strings represented by concatenating alphabets or strings in V zero or more times.
  - Σ = {a, b}, then Σ^* = { ε, a, b, aa, bb, ab, aaa, bbb, …}
- Kleene Plus $^+$: >= 1
  - $V^+ = V^* - \{ \varepsilon \}$

Grammars

- A grammar G is defined as a quadruple,
    - $G = (V, T, S, P)$,
    - where V is a finite set of objects called **variables**,
    - T is a finite set of objects called **terminal symbols**,
    - $S \in V$ is the **start** variable,
    - P is a finite set of ***productions***.

V, T, S, P

- ***Start symbol (S)*** is the starting point.
- ***Productions (P)*** start from ***S***, and convert it to strings defined by the grammar *G*.
- Variables and Terminal Symbols are used in Productions.
- ***Variables (V)*** are the things we can replace.
- ***Terminal Symbols (T)*** are the terminals of replacement, we cannot change it further.

Example

- $G = (V, T, S, P) = (\{S, A\}, \{a, b\}, S, P)$
- $P = \{S \rightarrow Ab, A \rightarrow Sa, A \rightarrow b\}$
- $\{S, A\}$ are variables.
- $\{a, b\}$ are terminal symbols.
- S is the start variable in $\{S, A\}$.
- P is a set of productions.

Production Rules

- The core of a grammar.
- Production rules specify how the grammar transforms a string to another.
- We will represent a production like this.
  - $x \rightarrow y$
  - where $x \in (V \cup T)^+$ and $y \in (V \cup T)^*$

How to Apply?

- The Key is ***Replacement***!
- Production x → y, applied to w = uxv, and we get z = uyv
- Find x in a string, and replace it with y.
- e.g.)
    - production: $\underbrace{\text{don't}}_{\textbf{x}} \rightarrow \underbrace{\text{do}}_{\textbf{y}}$
    - a string w = $\underbrace{\text{we}}_{\textbf{u}}\ \underbrace{\textbf{don't}}_{\textbf{x}}\ \underbrace{\text{know PL}}_{\textbf{v}}$
    - a new string z = $\underbrace{\text{we}}_{\textbf{u}}\ \underbrace{\textbf{do}}_{\textbf{y}}\ \underbrace{\text{know PL}}_{\textbf{v}}$

Derivation

- In the previous example, we converted **w** to **z** by applying the production.
- We can write such conversion as,
    - $w \Rightarrow z$
- and we say w **derives** z, or z is derived from w.

Derivation

- We can apply any productions whenever they're applicable.
- It is also possible to convert w_1 to w_n,
    - $w_1 \Rightarrow w_2 \Rightarrow \ldots \Rightarrow w_n$
    - $w_1 \Rightarrow^* w_n$
    - The whole process can be said as a derivation of a string w_n.

Example

- $G = (V, T, S, P) = (\{S, A\}, \{a, b\}, S, P)$
- P: $S \rightarrow Ab$, $A \rightarrow Sa$, $A \rightarrow b$
- Derive bbab.
  - $S \Rightarrow Ab$ $S \rightarrow Ab$
  - $Ab \Rightarrow Sab$ $A \rightarrow Sa$
  - $Sab \Rightarrow Abab$ $S \rightarrow Ab$
  - $Abab \Rightarrow bbab$ $A \rightarrow b$

Langauge L

- Now we can define the language L generated by the grammar G.

- Let $G = (V, T, S, P)$ be a grammar.

- Then the set

  - $L(G) = \{ w \in T^* : S \Rightarrow^* w \}$

  - is the language generated by G.

Langauge L

- $G = (V, T, S, P)$ → Grammar generating L.
- w ∈ T*
    - T* indicates a set of strings generated by concatenating terminal symbols T.
    - w belongs to T*.
- $S \Rightarrow^* w$
    - S is the start variable, which indicates the start of derivation.
    - w is derived by applying productions in P, starting from S.

Examples

- $G = (\{S\}, \{a,b\}, S, P)$, with P as follows.
  - $S \rightarrow aSb$,
  - $S \rightarrow \varepsilon$
- Then, $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$
- $S \Rightarrow^* aabb$
- We can conjecture that L(G) can be defined as follows.
  - $L(G) = \{ a^n b^n : n >= 0\}$
- What if a = ‘(‘ and b = ‘)’ ? ⟶ ab = ( ), aabb = (( ))

Examples

- How about another language like this?
  - $L = \{ a^n b^{n+1} : n >= 0\}$
  - e.g.) b, abb, aabbb, aaabbbb $\in$ L
- Can we find a grammar G that generates L?

Examples

- We can easily guess that terminal symbols are {a, b}.
- For variables, we need the start variable S.
- Can we do it with something similar to before?
  - S → aSbb
  - S → ε
  - S ⇒ aSbb ⇒ aaSbbbb **Fail!**

We need just one more b!

- Productions are very similar, but we just want to add one more b to ab or aabb.
- Add another variable, and use it to simulate $a^n b^n$.
- At the beginning with S, we simply add one b to it.
    - $S \rightarrow Ab$
    - $A \rightarrow aAb$
    - $A \rightarrow \varepsilon$

Let's Try

- Productions P
  - $S \rightarrow Ab$
  - $A \rightarrow aAb$
  - $A \rightarrow \varepsilon$
- $S \Rightarrow Ab \Rightarrow aAbb \Rightarrow aaAbbb \Rightarrow aabbb$ Successful!
- So the grammar G = ({S, A}, {a, b}, S, P).

You're expected to…

- Define a language L(G) when the grammar G is given.
  - Apply some production rules, and find common characteristics to define L(G).
- Find the grammar G when a language L(G) is given.
  - Take a few example strings of L, and find production rules.

Summary

- Syntax, Semantics, and Pragmatics
- Formal Language
  - Grammar
  - Language