Formal Language Theory

Motivation

Languages are a powerful abstraction: everything from logical formulas to compilation of programs can be studied using languages. The study of properties of languages is an important aspect of theoretical computer science and some of its applications, particularly the abstract problem of language recognition:

Given language L and string s, does s belong to L?

This comes up in applications such as compiler design, where source code must go through lexical analysis (to break up source code into tokens that represent identifiers, function names, operators, etc.) and parsing (to determine whether source code has correct syntax), something you can study in CSC488. It is also central to the study of computational complexity and computability, which you can study in CSC463.

We will look at two ways to express languages: descriptive ways (regular expressions, context-free grammars), and procedural ways (finite state automata, pushdown automata).

Basic Definitions

Alphabet: Any finite, non-empty set Σ of atomic symbols is considered as alphabet

Example 1. *The following sets define an alphabet.* $\{a, b, c\}$*,* $\{0, 1\}$*,* $\{+\}$*, etc.*

Remark. *Compound symbols like "ab" are not allowed (to prevent ambiguity) unless explicitly said otherwise.*

String: Any finite sequence of symbols is called a string. Empty sequence is also allowed and denoted ϵ or Λ (called "empty" or "null" string).

Example 2. *Consider the following examples of string.*

- a*, ab, cccc are strings over* $\{a, b, c\}$
- ϵ*,* $++++$ *are strings over* $\{+\}$
- $a + 00$ *is* **not** *a string over* $\{a, b, c\}$ *but it* **is** *over* $\{0, 1, +, a, b, c\}$*.*

Remark. *The set of all strings over alphabet* Σ *is denoted by* Σ^**.*

Remark. *Neither* ϵ *nor* Λ *are allowed as symbols in an alphabet to avoid confusion with empty string, unless explicitly said otherwise.*

Language: Any set of strings (empty or not empty, finite or infinite) is called a language.

Example 3. *Consider the following examples of language:*

- $\{bab, bbabb, bbbabbb, \cdots\}$ *is a language over* $\{a, b, c\}$
- $\{+, ++\}$ *is a language over* $\{+\}$
- $\{\epsilon\}$ *is a language over any alphabet*
- $\{\}$ *is a language over any alphabet*

Note: $\{\}$ is different from $\{\epsilon\}$. $\{\}$ contains NO string, but $\{\epsilon\}$ contains ONE string (i.e., the empty string ϵ).

Remark. *A string on an alphabet* Σ *is a subset of* Σ^**.*

We can define a set of operations that can be done on strings. These operations are operations that are applicable to any sequence:

- **Length** of string s, denoted by $|s|$, is the number of symbols in s. For example, $|bba| = 3$, $|+| = 1$, and $|\epsilon| = 0$.
- Strings s and t are **equal** iff $|s| = |t|$ and $s_i = t_i$, $\forall 1 \leq i \leq n$ where $n = |s|$ and v_i denotes i^{th} symbol in string v.
- **Reversal** of string s (denoted by s^R) is a string obtained by reversing the order of symbols in s. For example, $1011^R = 1101$, $+++^R = +++$, and $\epsilon^R = \epsilon$.
- **Concatenation** of strings s and t (denoted by st or $s \circ t$) consists of every symbol of s followed by every symbol of t. For example, $bba \circ bb = bbabb$, $\epsilon \circ +++ = +++$.

Remark. *For string s, and natural number k, s^k denotes k times concatenation of s with itself. E.g., $aba^2 = abaaba$, $++^0 = \epsilon$, and $\epsilon^3 = \epsilon$.*

Remark. *For alphabet Σ, Σ^n denotes set of all strings of length n over Σ, and Σ^* denotes the set of all strings over Σ. For example, $\{a,b,c\}^0 = \{\epsilon\}$, $\{0,1\}^3 = \{000, 001, 010, 011, 100, 101, 111\}$, $\{+\}^* = \{\epsilon, +, ++, +++, ++++, \cdots\} = \{+^k : k \in \mathbb{N}\}$*

Prefix: A string x is a prefix of string y if there exist a string x' (possibly ϵ) such that $xx' = y$.

Suffix: A string x is a suffix of string y if there exist a string x' (possibly ϵ) such that $x'x = y$.

Now that we have defined operations on strings, let's define some operations on languages. For all languages L, L' over alphabet Σ:

- **Complementation:** $\bar{L} = \Sigma^* - L$. E.g., if $L = \{0x : x \in \{0,1\}^*\} = \{0, 00, 01, 000, 001, \cdots\}$, then $\bar{L} = \{\epsilon\} \cup \{1x : x \in \{0,1\}^*\}$.
- **Union:** $L \cup L' = \{x : x \in L \text{ or } x \in L'\}$
- **Intersection:** $L \cap L' = \{x : x \in L \text{ and } x \in L'\}$
- **Set Difference:** $L - L' = L \cap \bar{L'}$
- **Reversal:** $Rev(L) = \{x^R : x \in L\}$. E.g., $Rev\{a, ab, abb\} = \{a, ba, bba\}$
- **Concatenation:** $L \circ L' = \{s \in \Sigma^* : s = r \circ t \text{ for } r \in L, t \in L'\}$. E.g.,

  $\{a, bc\} \circ \{bb, c\} = \{abb, ac, bcbb, bcc\}$
  $\forall L,\ L \circ \{\epsilon\} = L = \{\epsilon\} \circ L$ and $L \circ \{\} = \{\} = \{\} \circ L$
  $\{a, aa, aaa, \cdots\} \circ \{b, bb, bbb, \cdots\} = \{ab, abb, abbb, \cdots, aab, aabb, \cdots\} =$
  $\{s \in \{a,b\}^* : \text{s contains some number of a's followed by soe number of b's, with at least one of each}\}$

- **Exponentiation:** $L^k = \underbrace{L \circ L \circ \cdots \circ L}_{ktimes}$. E.g., $\{+, ++, +++\}^0 = \{\epsilon\}$, $\{\epsilon\}^5 = \{\epsilon\}$, $\{\}^5 = \{\}$ but $\{\}^0 = \{\epsilon\}$.
- **Kleene star:** $L^{\circledast} = L^0 \cup L^1 \cup L^2 \cup \cdots = \{s : s \in L^k \text{ for some } k\}$. E.g., $\{ab\}^{\circledast} = \{\epsilon, ab, abab, ababab, \cdots\}$, $\{\epsilon\}^{\circledast} = \{\epsilon\}$, and $\{\}^{\circledast} = \{\epsilon\}$.

Finite State Automata

Finite State Automatas (FSA) are simple models of computing devices used to analyze strings. A FSA has a fixed, finite set of **states**, one of which is the **initial state** and some of which are **accepting** (or final) states. There are **transitions** from one state to another for each possible symbol of a string. The FSA starts in its initial state and processes a string one symbol at a time from left to right. For each symbol processed, the FSA switches states based on the latest input symbol and its current state, as specified by the transitions. Once the entire string has been processed, the FSA will either be in an **accepting** state (in which case the string is **accepted**) or not (in which case the string is **rejected**).

Example 4. *Let's look at a simplified control mechanism for a vending machine that accepts only nickels (5¢), dimes (10¢) and quarters (25¢), where everything costs exactly 30¢and no change is ever given:*

Alphabet: $\Sigma = \{n, d, q\}$ (for nickel, dime, quarter)

States: set of states $= \{0, 5, 10, 15, 20, 25, 30+\}$ (for amount of money put in so far; no need to keep track of excess since no change will be provided)

Transitions: are defined by following table (state across the top, input symbol down the side), with the initial state being 0 and the only accepting state being 30+ – representing amounts ≥ 30:

	0	5	10	15	20	25	30+
n	5	10	15	20	25	30+	30+
d	10	15	20	25	30+	30+	30+
q	25	30+	30+	30+	30+	30+	30+