Strings

Definitions

- **Alphabet (Σ)**
  A a **finite set** consiting of symbols or characters.
- **String**
  A **sequence** of symbols drawn from some alphabet Σ.
  A string can be empty, denoted by ϵ
- **Σ***
  Denotes the set of **all possible strings** over an alphabet Σ

Functions on a string

- **length**
  $|\epsilon| = 0$
  $|100111| = 6$
- **character occurences**
  $\#_a(\texttt{abbaa}) = 3$
- **concatenation**
  $x = good$, $y = bye$
  $xy = goodbye$, $yx = byegood$
  Also, $x||y \ = goodbye$ (Another notation)
  - The empty string ϵ is the identity of the concatenation.

$$\forall x(x\epsilon = \epsilon x = x)$$

  - Concatentation is also associative.

$$\forall s, t, w\ ((st)w = s(tw))$$

- **replication**

$$w^0 = \epsilon$$
$$w^i = w^{i-1}w$$

  $a^3 = aaa$, $(bye)^2 = byebye$, $a^0b^3 = bbb$

- **reversal**
  For a string w, the reversal w^R is defined as:

$$\text{if } |w| = 0 \text{ then } w^R = w = \epsilon$$

$$\text{if } |w| \geq 1 \text{ then } \exists a \in \Sigma(\exists u \in \Sigma^*(w = ua))$$
$$\text{then } w^R = au^R$$

Theorem 2.1

If w and x are strings, then $(wx)^R = x^R w^r$

Relations on Strings

- **substring**
  A string s is a substring of a string t iff s occurs contiguously as a part of t
  A string s is a **proper substring** of a string t iff s is a substring of t and $\mathbf{s \neq t}$
- **prefix**
  A string s is a prefix of t iff $\exists x \in \Sigma^*(t = sx)$
  proper prefix again same as above ($\mathbf{s \neq t}$)
- **suffix**
  A string s is a suffix of t iff $\exists x \in \Sigma^*(t = xs)$
  proper suffix again same as above ($\mathbf{s \neq t}$)

Empty string ϵ is a suffix, prefix and a substring of every string

Languages

A **language** is a **finite or infinite set of strings** over a fine alphabet Σ.
For a language L defined over alphabet Σ, $L \subset \Sigma^*$
Σ_L is used to denote the alphabet from which the strings in the language L are formed.

Ways of defining languages

- We can have a **language generator** that **enumerates** the elements of the language or have **language recognizer** which **decides** whether or not a candidate string is in the language and returns *True* if it is and *False* if it isn't.
- **Go through examples on page 11-13 to see different ways languages are defined**
- Empty language $L = \{\} = \varnothing$ is a language that contains no strings but a language $L = \{\epsilon\}$ is not an empty language as it consists of one element i.e., an empty string.
- **Lexigographic order**
  - Sometimes we may care about the order in which the elements of a language are generated in.
  - If there exists a total order D then we can use D to define on L a useful total order called **lexicographic order** (written $<_L$):

$$\forall x(\forall y((\ |x| < |y|\ ) \rightarrow (x <_L y)))$$

    and of the strings of the same length, sort them in dictionary order using D.

Cardinality of a language

- The smallest language over any alphabet is $\varnothing$ with cardinality of 0
- The largest language over any alphabet Σ is Σ^*
- For $\Sigma = \varnothing,\ \Sigma^* = \{\epsilon\}$ and $|\Sigma^*| = 1$ theorem 2.2 is for non-empty alphabets.

Theorem 2.2

If $\Sigma \neq \varnothing$ then Σ^* is countably infinite ($\aleph_0$) - Therefore, the cardinality of every language is at least 0 and at most $\aleph_0$

How many languages are there?

- The set of languages defined on Σ is $P(\Sigma^*)$
- For $\Sigma = \varnothing,\ \Sigma^* = \{\epsilon\}$ and $P(\Sigma^*)\ is\ \{\varnothing, \epsilon\}$
- But when $\Sigma \neq \varnothing$ theorem 2.3 applies.

Theorem 2.3

If $\Sigma \neq \varnothing$ then the set of languages over Σ is uncountably infinite.

Functions on a language

- Since languages are sets, all of the standard set operations are well-defined on languages. (refer Ex2.12 from page 15)
- **concatenation**
  Let L_1 and L_2 be two languages defined over Σ then:

$$L_1L_2 = \{w \in \Sigma^* : \exists s \in L_1(\exists t \in Ł_2(w = st))\}$$

Example
Let $L_1 = \{$`cat, dog, mouse`$\}$ and $L_2 = \{$`bone, food`$\}$

$$L_1L_2 = \{\texttt{catbone, catfood, dogbone, dogfood, mousebone, mousefood}\}$$

- The language $\{\epsilon\}$ is the identity for concatenation of languages. For all languages L:

$$L\{\epsilon\} = \{\epsilon\}L = L$$

- The language $\varnothing$ is a zero for concatenation of languages. For all languages L:

$$L\varnothing = \varnothing L = \varnothing$$

There are no ways of selecting a string from an empty set.
- It's also associative

$$(L_1L_2)L_3 = L_1(L_2L_3)$$

- **Kleene star**
Let L be a language defined over Σ then:

$$L^* = \{\epsilon\} \cup \{w \in \Sigma^* : \exists k \geq 1(\exists w_1, w_2, \dots w_k \in L \\ (w = w_1w_2 \dots w_k))\}$$

note[1]
Example:

Let $L =$\{`dog, cat, fish`\}. Then:

$$L^* = \{\epsilon, \texttt{dog, cat, fish, dogdog, dogcat,...,} \\ \texttt{fishdog, fishcat, fishfish,...,} \\ \texttt{fishcatfish,fishdogfishcat,...}\}$$

- L^* always contains an **infinite number** of strings as long as L is not $\varnothing$ or $\{\epsilon\}$. i.e., as long as there is one nonempty string in L.
- If $L = \varnothing$, then $L^* = \{\epsilon\}$ as there are no strings that could be concatenated to ϵ to make it longer. This means that the Kleene star of any language must have at least ϵ in it.
- If $L = \{\epsilon\}$, then $L = \{\epsilon\}$ as well.

- **reverse** Reverse of a language L defined over Σ, written as L^R is:

$$L^R = \{w \in \Sigma^* : w = x^R \text{ for some } x \in L\}$$

Theorem 2.4

If L_1 and L_2 are languages, then $(L_1L_2)^R = L_2^R L_1^R$

Assigning Meaning to Languages

To be able to use the language and form the framework for it's applications we need to assigning meaning to its strings. Working with formal languages require a precise way to assign meaning to strings (also called its **semantics**). - A function that maps strings to its meanings is called a **semantic interpreation function**. But since languages are infinite, its not, in general possible to map each string with its meaning.
- Instead we define a function in such a way that it identifies units and can combine those units based on rules to build meanings for larger expression. We call such a function, **compositional semantic interpretation function**
- So this function "composes" the meanings of simpler constituents into a single meaning for a larger expression.
- When we define a formal function to fulfill a specific purpose, we design it so that there exists a compositional semantic interpretation function.
- For example there exists a compositional semantic interpretation function for the C programming language, a C compiler.
- These functions are generally *not* one-to-one. For example:

- "Chocolate, please", "I'd like chocolate", "I'll have chocolate" all mean the same
- These also have to same meaning

```
int x = 4;     int x = 4;     int x = 4;
x++;           x = x --1;     x = x + 1;
```

[1]The part $\exists w_1, w_2, \dots w_k \in L$ should mean 'select k elements from L'. Each time selecting k elements in a particular permutation and concatenating them.