# Foundations of Computer Science

**Notes from 02-26-2015**

Kenny Roffo

March 10, 2015

## Contents

# 1 Formal Languages

## 1.1 Definitions

1. A symbol is the basic indivisible entity

   Natural Languages: words, not letters

2. An alphabet is a finite, nonempty set of symbols

   $\Sigma$ is typically used as the name for the alphabet

   Natural languages: $\Sigma = \{\textit{all words in English}\}$ - Lexicon

3. A string (over $\Sigma$) is a finite sequence of symbols (over $\Sigma$)

   Properties:

   - length
   - empty string denoted $\lambda$
   - concatenation
   - $\lambda$ identity of concatenation

4. $\Sigma^*$ is the set of all finite strings over $\Sigma$

   e.q. $\Sigma = \{0, 1\}$, $\Sigma^* = \{$all strings of 0 and 1$\}$

   $\Sigma$ is an alphabet
   $\Sigma^*$ is defined recursively

   $\lambda$ is an element of $\Sigma^*$, and if $w \in \Sigma^*, a \in \Sigma$, then $wa \in \Sigma^*$

5. A language $L$ is any set of strings formed from a given alphabet $\Sigma$
   $\phi$ is a language over *any* alphabet
   $\Sigma$ is a language over $\Sigma$
   $\Sigma^*$ is a language over $\Sigma$

## 1.2 Examples

1. $\Sigma = \{b\}$
   $\Sigma^* = \{\lambda, b, bb, bbb, ...\}$

2. $\Sigma = \{a, b\}$
   $\Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, ...\}$
   L is a language over $\Sigma$ that is defined recursively: ex: $L = \{w | w = a^n b^n, n \geq 1\}$
   ($a^n$ means $aaa...a$ n times)
   Thus $L = \{ab, aabb, aaabbb, aaaabbbb, ...\}$, if $w \in L$, then $awb \in L$

3. Let $L_n = a^n b^n$. Then $L = L_0 \cup L_1 \cup L_2 \cup ...$

4. Let $\Sigma_i$ be the language of $i$ length strings in the alphabet $\Sigma = \{a, b\}$.
   Then $\Sigma_0 = \{\lambda\}$, $\Sigma_1 = \{a, b\}$, $\Sigma_2 = \{aa, ab, ba, bb\}$, ...
   It follows that the language $\Sigma^* = \Sigma_0 \cup \Sigma_1 \cup \Sigma_2 \cup ... = \bigcup_{i=1}^{\infty} \Sigma_i$

### 1.3 What is $\Sigma^*$

Concatenation - Making new strings from existing strings. We can also concatenate strings with languages and languages with languages. If $L_1$ and $L_2$ are languages, then $L_1 L_2 = \{w_1 w_2 | w_1 \in L_1, w_2 \in L_2\}$

ex: Let $L_1 = \{\text{in,out}\}$ and $L_2 = \{\text{law,door,ward}\}$.
Then $L_1 L_2 = \{\text{inlaw,outlaw,indoor,outdoor,inward,outward}\}$

$\Sigma^*$ is the set of all strings made from the alphabet $\Sigma$. But why $\Sigma^*$?
$\Sigma^*$ is the result of concatenating $\Sigma$ with itself zero or more times.
$\Sigma^+$ is the result of concatenating $\Sigma$ with itself one or more times.
This is called the positive closure of $\Sigma$.

## 2 Regular Expressions

### 2.1 What is a Regular Expression?

A *regular expression* (regex) is a way to specify patterns for strings using union (or), concatenation, and $^*$

A regex over $\Sigma$ is defined:
*Basis*: Every $a \in \Sigma$ is a regex over $\Sigma$
*Recursive*: If $u$ and $v$ are regex over $\Sigma$ then $u|v$, $uv$, and $u^*$ are all regex over $\Sigma$

Here, | means *or* and $^*$ means 0 or more. When in doubt, use parentheses.

grep - general regular expression parser - a Unix command which searches a file for a pattern defined by a regex.

Let $X = \{a, ab, aba\}$ and $Y = \{b, bb\}$. Then

- $XY = \{ab, abb, abb, abbb, abab, ababb\}$ (Concatenation)
- $X|Y = \{a, ab, aba, b, bb\}$ (Like union)
- $X^* = \{a, aa, aaa, ..., aab, ab, abab, ababab, ..., aab, aaba, aaab, ababa, ...\}$
  (All possible strings from 0 or more concatenations)
- $ababa \in X^*$
- $ababa \in XY^*$
- $ab(ab)^*a$ is a regex that matches $ababa$

## 3 Finite State Machines

### 3.1 The Vending Machine

Consider a vending machine which contains Jelly beans and Gum. The Machine has inputs

- N - 5 cents
- D - 10 cents
- J - Jelly Bean (Costs 20 cents)
- G - Gum (Costs 15 cents)

These can be represented by $\Sigma = \{N, D, J, G\}$. The machine also has outputs

- b - beep when money is added
- j - jelly bean dispensed
- g - gum dispensed

Design a Finite State machine - a machine with a finite number of "things to remember" This vending machine has to "remember":

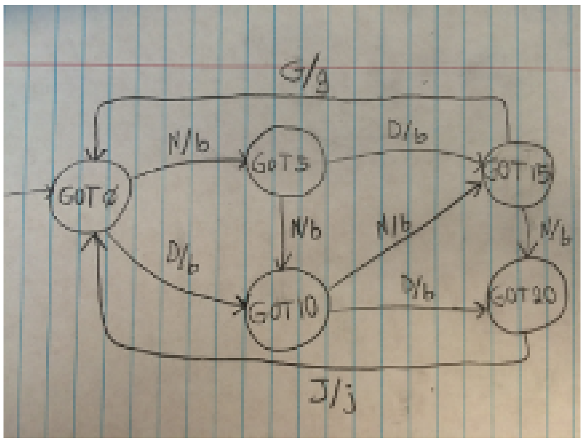
- total money deposited (but not he order in which coins are desposited)
- which product is selected

States are drawn with circles and named on the inside:

For our Vending Machine the states are described by how much money is in the machine, and the transitions represent money input, or a purchase or gum or a jelly bean. The states live in the set $Q = \{GOT\emptyset, GOT5, GOT10, GOT15, GOT20\}$. State transitions are defined by a function $\delta : Q \times \Sigma \to Q$. As an example,

$$(GOT5)(N) \to GOT10$$

Here is an example of a state transition diagram.



## 3.2 Finite State machine

A finite state machine has one or more states. For the vending machine in the previous section the inital and final states happen to be the same. Any path in the state transition diagram is called a computation. Any path in a finite state machine which starts in the initial state and ends in the final state is called an *acceptable* path.

A finite state machine must have:

- An input mechanism
- A computing mechanism
- An output mechanism

We use ordered pairs to reference what has happened in the fsm. Referring back to the vending machine, here is an example:

$(GOT5, NNNDG) \to (GOT10, NNDG) \to (GOT15, NDG) \to (GOT20, DG) \to (GOT20, G) \to (GOT\emptyset, \lambda)$

The first part of the pair represents the current state while the second represents the input. Each pair is called an *instantaneous configuration.*

A finite state machine is defined by a 5-tuple: $M = (Q, \Sigma, \delta, q_0, F)$:

- $Q$: finite, nonempty set of states
- $\Sigma$: alphabet (finite, nonempty set of symbols
- $\delta$: $Q \times \Sigma \to Q$ is the state transition function
- $q_0$: initial state
- $F \subset Q$ : set of final states