

# CmpE 585 - Course Notes

October 13, 2009

## Lecture 1

**Date:** Sept. 28, 2009. **Instructor:** C. Say. **Scribe:** B. Gökçe. **Book Name:** Sequential Machines: Selected Papers ed. Moore.

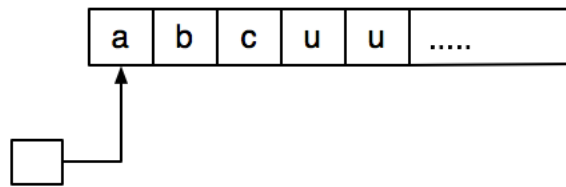
3 important papers:

1. Finite Automata and Their Decision Problems (Rabin & Scott)
2. Reduction of Two-way Automata to One-way Automata (Shepherdson)
3. Probabilistic Automata (Rabin)

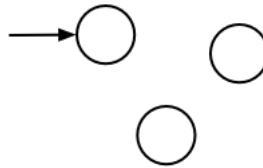
### Introduction

- The Turing Machine (T.M.) was initially “used” for computability theory.
- Finite Automaton Approach is one of the first to concentrate on resource limitations

Single-tape T.M.:



States:

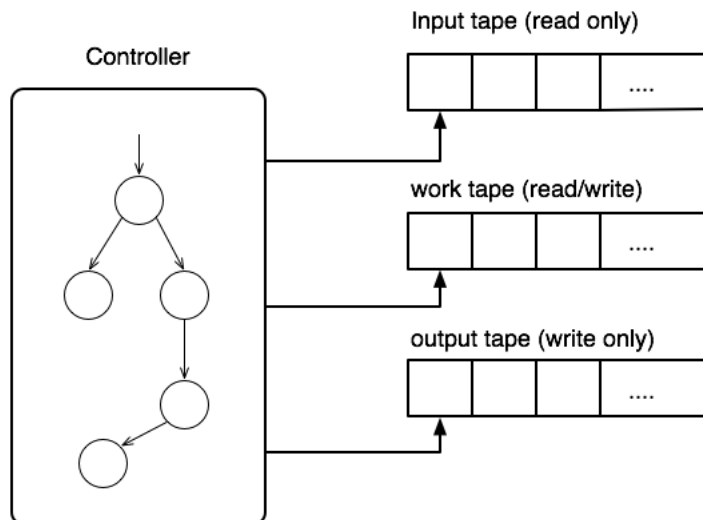


Program: Constructed by “if-then-else” type rules.

$Q \times \Gamma \rightarrow Q \times \Gamma \times \{-1, 0, 1\}$  where  $Q$  represents the state space and  $\Gamma$  represents the input alphabet.

**Sublinear Complexity:** In sublinear complexity, the boundaries are less than  $n$ , length of the input. In time complexity, sublinear bounds are insufficient for reading the entire input, so we don’t consider them here. In sublinear space complexity the machine is able to read the entire input but it doesn’t have enough space to store the input.

For studying sublinear space complexity, we use the following kind of T.M.:

3 tapes:

Rules:

1. Read from the input tape
2. Read from the work tape
3. Determine the next state according to the read data and the current state
4. Move the pointer of input tape in a direction
5. Write to the work tape
6. Move the pointer of work tape in a direction
7. Write to the output tape
8. Move the pointer of work tape to the right

It is shown that the power of DTM's with  $\log\log$  space is more than that of DTM's with no space.

We mostly concentrate on constant space complexity ( $O(1)$ ).

**Theorem:** Power of automata with  $O(1)$  space equals to that of automata with zero space.

**Proof:** Reduction of TM with no space to TM using  $O(1)$  space is trivial. For the opposite reduction, following procedure is applied: For each state in the automata with  $O(1)$  space, a state where the information about the work-tape is embedded is defined for the construction of TM using no work-tape. By this way, the information about the

work-tape which is finite is stored in the states.

## The Myhill-Nerode Theorem

### IDFA:

From Sipser's 2<sup>nd</sup> edition, problems 1.51 and 1.52.

$x, y \in \Sigma^*$

$L \subseteq \Sigma^*$

$x$  and  $y$  are distinguishable by L if there exists a string  $z$  such that exactly one of the strings  $xz$  &  $yz$  is a member of  $L$ .

Example:

$L : a^*b^*$

If for every string  $z$ , we have  $xz \in L \leftrightarrow yz \in L$ , then we say  $x$  and  $y$  are indistinguishable by L.

In this situation, we write  $x \equiv_L y$  (equivalence relation).

Number of equivalence classes can be finite (ex: equivalence mod  $k$ ) or infinite (ex: equality over the real numbers).

Let  $L$  be a language and  $X$  be a set of strings. Say that  $X$  is pair-wise distinguishable by  $L$  if every two distinct strings in  $X$  are distinguishable by  $L$ .

Example:

$L : a^*b^*$

$X = \{ab, a, aba\}$

$L : \{a^n b^n \mid n \geq 0\}$

$X = \{a, ab, aaab, aaaab, \dots\}$

Define the index of  $L$  to be the maximum number of elements in any set that is pairwise distinguishable by  $L$ .

i) if  $L$  is regular (recognized by a 1DFA with  $k$  states) then  $L$  has index at most  $k$ .  
If index were more than  $k$ , then at least two of members in  $X$  should finish with the same state. Therefore, these two strings cannot be distinguishable.

ii) if index of  $L$  is a finite number  $k$ ,  $L$  is recognized by a 1DFA with  $k$  states.  
Let  $X = \{s_1, s_2, \dots, s_k\}$  be pairwise distinguishable by  $L$ . Let's construct 1DFA  $M = \{Q, \Sigma, \delta, q_0, F\}$  with  $k$  states.  
 $Q = \{q_1, q_2, \dots, q_k\}$   
 $\delta(q_i, a) = q_j$ , where  $s_j \equiv_L s_i a$

Two strings from the class of  $s_i$  will be in the same class when the same symbol is appended. Otherwise,  $s_{i1}az$  and  $s_{i2}az$  would be distinguishable by  $L$  and this means that  $s_{i1}$  and  $s_{i2}$  should be distinguishable by  $L$  with the string  $z$  of "az". This is a contradiction with the initial assumption.

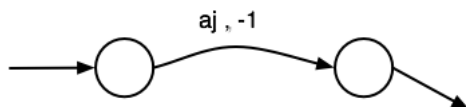
$F = \{q_i \mid s_i \in L\}$

Let the start state be the state  $q_i$  such that  $q_i \equiv_L \epsilon$ .

The index of  $L$  (regular) is the size of the smallest DFA recognizing it.

**2DFA:**

Direction of the head on the tape can be 'left', 'stay put' and 'right'.



$$L_m = \{1^x | x \equiv 0(\text{mod } m)\}$$

**Theorem:**

For any  $m$ , the index of  $L_m$  is  $m$ .

**Proof:**

Let's say a DFA with  $k \leq m$  states recognizes  $L_m$ . What happens when that DFA reads  $1^m$

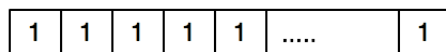
We will call the steps of the machine while recognizing the string  $1^m$  as  $q_1, q_2 \dots q_m, q_{m+1}$ . Machine should accept at  $m + 1^{\text{st}}$  step. If  $k < m$  then there exists  $q_i$  and  $q_j$  where  $i < j$  and  $q_i$  and  $q_j$  are the same state.

$$L_m = \{1^x | x \equiv 0(\text{mod } m)\}$$

But much smaller 2DFA's exist for some values of  $m$ .

Example:

$$m = 2 \times 3 \times 5 \times 7 \times 11 \times 13 \times 17$$



→

In the first pass, check for  $L_2 \rightarrow 2$  states

←

In the second pass, check for  $L_3 \rightarrow 3$  states

→

In the third pass, check for  $L_5 \rightarrow 5$  states

←

In the fourth pass, check for  $L_7 \rightarrow 7$  states

→

In the fifth pass, check for  $L_{11} \rightarrow 11$  states

←

In the sixth pass, check for  $L_{13} \rightarrow 13$  states

→

In the seventh pass, check for  $L_{17} \rightarrow 17$  states

We assume left and right end-markers on the tape.

***Shepherdson's Theorem:***

2DFA's can only recognize regular languages.

***Proof:***

Given any 2DFA M recognizing L, we can construct an equivalent 1DFA M'.

Let S be the state set of M.

For each  $t \in \Sigma^*$ , define a function  $T_t : (\bar{s}_0 \cup S) \rightarrow (\{0\} \cup S)$

For  $s \in S$ ,  $\tau_t(s)$  describes the ultimate result of the motion of M when started in state S on the rightmost symbol of t, i.e. if with these initial conditions M ultimately leaves t from the right in the state S' then  $\tau_t(s) = s'$ , if M never leaves t, or leaves t from the left, then  $\tau_t(s) = 0$ .

$\tau_t(\bar{s}_0)$  describes the result of the motion when M is started in the initial state  $s_0$  on the leftmost symbol of t.

It is easy to see that if  $\tau_{t_1} = \tau_{t_2}$  then  $t_1 \equiv_L t_2$ .

## Lecture 2

*Date:* Oct. 5, 2009. *Instructor:* C. Say. *Scribe:* Barış Kurt.

### Constructing 1DFA from 2DFA

In the first lecture, we have seen that 2DFA can be converted to 1DFA according. Here how we construct a 1DFA, given a 2DFA.

- First we make a start state which is equal to the equivalence class of strings which includes the empty string.

$$\begin{aligned}\tau_\epsilon(\overline{s_0}) &= s_0 \\ \tau_\epsilon(s) &= 0, \text{ for all } s \in S\end{aligned}$$

- We make a table for each  $\tau_t$  by simulating the 2DFA. We start the machine from each state on the leftmost symbol of  $t$ , and learn the  $\tau_t$  function for string  $t$ . Here

is the table for  $\tau_\epsilon$ :

$\overline{s_0}$	$s_0$
$s_0$	0
$s_1$	0
..	0
$s_n$	0

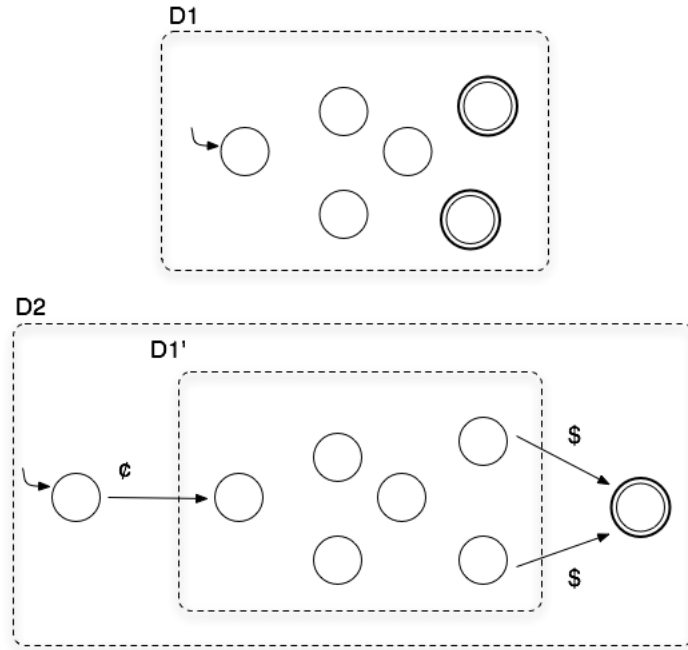
- For each new table we add a new state to the 1DFA, and if the table we calculated has already been created, we direct the arrow to the previously added state that represents the table.
- We know that a regular language has in index, which means that it has finite number of equivalence classes and each string on the language falls into one of those classes. So there are finite number of  $\tau$  functions.
- As we simulate the 2DFA, we detect the infinite loops by counting the different configurations of the 2DFA. There should be less than the *number of states X input string length* number of configurations, else the machine will start to loop.
- We decide accept states of the 1DFA by simulating the 2DFA with a member string from each equivalence class. If the string is accepted, the state representing the class on 1DFA becomes the accept state.

### Using End Markers

Originally 2DFA's are defined so that we have only the input string on the tape, and acceptance is by moving off the right end. We can define two symbols, ( $\$$ ,  $\epsilon$ ) for the start and the end of the input string. These end markers give extra information about the input string, because whenever the machine reads one of them, it will not move off the tape from the right or left side, and will have to ability to turn back and re-scan the tape. Although it seems that this property improves the power of the machine, it

only simplify algorithms. The language that can be recognized by the machine is not affected.

Proof:  $\phi L \$$  is regular if and only if  $L$  is regular. Given a DFA  $D1$  recognizing  $L$ , we can construct another DFA  $D2$  recognizing  $\phi L \$$ . We add a new initial state and go to the former initial state by reading input  $\phi$ . We change accept states by making them non-accepting states, and adding a new accept states such that we reach the new accept state from the former accept states by reading input  $\$$ .



## Probabilistic Finite Automata

**Definition:** A probabilistic automaton over the alphabet  $\Sigma$  is a system  $U = \langle S, M, s_0, F \rangle$  where  $S = \{s_0, \dots, s_n\}$  is the finite set of states,  $M$  is a function from  $S \times \Sigma$  into  $[0, 1]^{n+1}$  such that for  $(s, \rho) \in S \times \Sigma$ ,

$$M(s, \rho) = (p_0(s, \rho), \dots, p_n(s, \rho)) \quad (1)$$

$s_0$  is the initial state and  $F \subseteq S$  is the set of final (accept) states.

**Definition:** Let  $M$  be a PFA and  $\lambda$  be a real number  $0 \leq \lambda < 1$ . We define the language  $L$  as

$$L(M, \lambda) = \{x \mid x \in \Sigma^*, \lambda < p(x)\} \quad (2)$$

where  $p(x)$  is the sum of the probabilities of the accept states. All such languages are called *stochastic languages*.  $\lambda$  is called the cut-point.

**Computation As Matrix Multiplication:**

The transitions of a DFA can be represented as binary matrices, such that each matrix shows the transition from every state to every other state for a given input symbol. For example, transitions of a DFA with two input symbols  $\{a, b\}$  and three states  $\{q_0, q_1, q_2\}$  can be written as:

$$M_a = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \quad M_b = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

Where  $M_a$  is the transition matrix for input symbol  $a$  and  $M_b$  is the transition matrix for input symbol  $b$ . We show the current state of the DFA with the vector  $[q_0 q_1 q_2]$ . The vector  $[100]$  shows initial configuration, meaning that the machine is at state  $q_0$ .

When the machine reads input symbol  $a$  at the initial state  $q_0$ , it goes to the state  $q_1$ . This transition can be represented as a matrix multiplication such as:

$$[1 \ 0 \ 0] \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} = [0 \ 1 \ 0]$$

Similarly, for input string  $ababbba$  we multiply the state vector by the matrix

$$M_{ababbba} = M_a M_b M_a M_b M_b M_b M_a$$

The transitions of a PFA can also be written as matrices, such that the entries in the matrix are not necessarily 1 and 0. An example of transition matrix for a PFA with three states  $\{q_0, q_1, q_2\}$  for input symbol  $a$  can be like this:

$$M_a = \begin{bmatrix} 0.3 & 0.5 & 0.2 \\ 1 & 0 & 0 \\ 0 & 0.7 & 0.3 \end{bmatrix}$$

$M_a$  shows that whenever the input symbol  $a$  is read, and the machine is at state  $q_0$ , it stays at  $q_0$  with probability 0.3, it jumps to state  $q_1$  with probability 0.5 and to state  $q_2$  with probability 0.2. The matrix  $M_a$  is called *stochastic matrix* and the sum of its raw entires add up to 1.

**PFA's can recognize non-regular languages:**

Let  $\Sigma = 0, 1$ ,  $S = s_0, s_1$  and  $F = s_1$ . Define the matrices

$$M_0 = \begin{bmatrix} 1 & 0 \\ 0.5 & 0.5 \end{bmatrix} \quad M_1 = \begin{bmatrix} 0.5 & 0.5 \\ 0 & 1 \end{bmatrix}$$

For input string  $x = x_1 x_2 x_3 \dots x_n$ , the probability of the accept state can be calculated as:

$$M_{x_1} M_{x_2} M_{x_3} \dots M_{x_n} = \begin{bmatrix} m & p \\ q & r \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} m & p \\ q & r \end{bmatrix} = \begin{bmatrix} m & p \end{bmatrix}$$

then  $p = 0.x_n x_{n-1} x_{n-2} \dots x_1$  where  $p$  is written in binary expression.

- If  $0 \leq \lambda < \lambda_1 < 1$  then  $L(M, \lambda_1) \subsetneq L(M, \lambda)$ .
- There always exists a rational number between  $\lambda$  and  $\lambda_1$ . The input string that generates that rational number is in one language but not in the other language.
- There are uncountably many cut-points, hence PFAs can recognize uncountably many languages.
- There are only countably infinite regular languages, therefore, PFA's recognize non-regular languages.
- Moreover, PFA's recognize non-Turing recognizable languages, because there are countably infinite Turing recognizable languages.

**Theorem:** Given  $L(M, \lambda)$  and any  $\lambda' \in (0, 1)$ , there exists a PFA  $M'$  such that  $L(M', \lambda') = L(M, \lambda)$ .

$M'$  is constructed as follows: For each symbol  $a$ , construct the matrix  $M'_a$  such that

$$M'_a = \begin{bmatrix} M_a & 0 \\ 0 & 1 \end{bmatrix}$$

- If  $\lambda' < \lambda$ , then  $M'$  jumps to the new state with probability  $1 - \lambda'/\lambda$  on the left end marker, and does what  $M$  normally does with the remaining probability. The accept states are the same as the accept states of  $M$  (The new state is not an accept state).
- If  $\lambda' > \lambda$ , then  $M'$  does what  $M$  does with probability  $p = 1 - \lambda'/1 - \lambda$  on the left end marker, and jumps to the new state with probability  $1-p$ . The new a state is also an accept state.