Lectures 1: Formal Languages and Complexity
September 9, 1998
Bud Mishra

1 Finite State Automata

An FSA (Finite State Automaton) is the simplest model of computation as it is the model of computation involving *finite amount of memory*. For instance, you may wish to consider the simplest possible such model with just 1-bit of memory. It can have two states: `ON` and `OFF`.

For instance, we can consider the standard on-off model of speech developed in the early days of telephony. There are two states: OFF and ON. The speaker can remain silent (S) or talk (T). S brings the speaker to OFF state (independent what state he is currently in) and T brings him to ON state again independent of the current state.

In picture, this *machine* is depicted by a *digraph* as shown below:

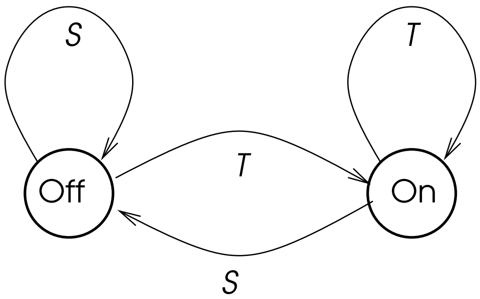

Figure 1: **Talking Machine**

But there is even another way to look at this computation—namely, the *language* recognizable by this computational model. These are all the valid strings of S and T's that keeps the machine in the quiet or OFF state (assuming that the "silence is golden" and it is your natural state). This language is simply the following *set* of finite strings:

$$\{ \quad \epsilon,$$
$$S, SS, \ldots, SSS \cdots S,$$

$$
\begin{aligned}
&\quad TS, TTS, \ldots, TT\cdots S,\\
&\quad STS, STTS, \ldots, STT\cdots S,\\
&\vdots\\
&\quad S\cdots ST\cdots TS\cdots ST\cdots TS\cdots S\\
&\}
\end{aligned}
$$

You may not think that this model is adequate as it cannot describe the possibility of an infinite string in which the machine is silent infinitely often. There are in fact models for these kinds of "reactive processes" described by machines called Büchi automata—but it is not something that we want to talk about in the very first lecture.

Going back to the language recognized by an FSA, there is a *meta-language* to describe such a *regular language*. For instance, the "talking machine" recognizes the regular language

$$S^*(T^+S^+)^*.$$

which simply is saying that a string of zero or more S's followed by zero or more repetitions of substrings consisting of 1 or more T's followed by 1 or more S's keeps the machine in the off state.

A language consisting of 1 K followed by zero or more N's can be described as KN^* (read KN star not Ken Starr).

There are many things we can do with this basic model: we can introduce *non determinism*; we can make the state transitions *probabilistic*; we can make many such individual machines interact with each other by sending and receiving messages; we can endow an FSA with external memory, stacks or infinite tapes where each tape symbol can be read and modified by an i/o (input/output) head, etc. Each such modification will change the computing power and will raise many intriguing questions about how we understand computation. For instance, it is relatively easy to model a brain by modeling each neuron by a 1-bit (ON/OFF) FSA that communicate with each other via axons that inhibit and excite each other. We could also model cells and gene-switches in biology. We could model discrete event dynamical systems and many other objects in nature.

A particularly interesting example occurs when state transitions occur probabilistically. These give rise to classical Markov chains. Let us reconsider our ON-OFF model for speech. Assume that in an OFF state, we can stay silent with probability α or talk with probability $1-\alpha$; in an ON state, we can talk with probability β or go silent with probability $1-\beta$.

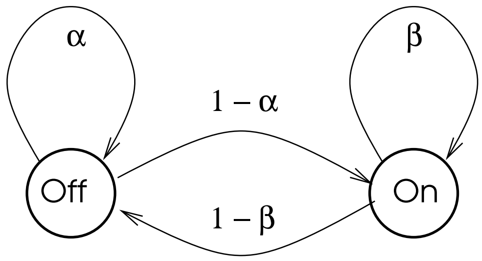

Figure 2: **Markov's Talking Machine**

Let $P_{0,t}$ be the probability that we are in OFF state at time t and $P_{1,t}$, the probability of being in ON state at time t. Then by Baye's rule:

$$\begin{aligned} P_{0,t+1} &= P(S|OFF)P_{0,t} + P(S|ON)P_{1,t} \\ &= \alpha P_{0,t} + (1-\beta)P_{1,t} \\ P_{1,t+1} &= P(T|OFF)P_{0,t} + P(T|ON)P_{1,t} \\ &= (1-\alpha)P_{0,t} + \beta P_{1,t} \end{aligned}$$

In matrix form

$$(P_{0,t+1}\ P_{1,t+1}) = (P_{0,t}\ P_{1,t}) \begin{pmatrix} \alpha & 1-\alpha \\ 1-\beta & \beta \end{pmatrix}.$$

There are many reasons we wish to study these machines. Here are some applications that you will see as computer scientists:

1. Linguistics: Lexical analysis
2. Hardware: HDL, Verification
3. Data Mining
4. Communication protocol
5. Controllers

We need to look at these both as machines and as language. The machine can be thought of as a recognizer or as a generator of the language. While either view would suffice, in order to have total mastery over these concepts we need to look at them in every possible combination. There are still other ways of thinking about them: algebraically (monoids and semigroups) and logically (temporal and modal logics with associated Kripke models). But we will not go into those—we will get ahead of ourselves.

2 Mathematical Concepts

2.1 Strings, Alphabets and Languages

1. *Symbols*: Basic Abstract entities. Examples: Letters, digits, atomic events.

2. *Alphabet*: Set of all symbols. Examples: $\{0, 1\}$, $\{a, b, c, d\}$, $\{\text{on, off}\}$ and $\{A, T, C, G\}$.

3. *String* (Word): Finite sequences of symbols juxtaposed. $GATTACA$, 1010010 and $abcb$.

   Length of a string, $w = |w|$ = number of symbols composing the string.

   Empty string, ϵ = string consisting of zero symbols.

   $|\epsilon| = 0$.

   Prefix of a string = Any number of leading symbols of that string.

   $\text{Pre}(w) = \{x | \exists y.xy = w\}$

   Suffix of a string = Any number of trailing symbols of that string.

   $\text{Suf}(w) = \{y | \exists x.xy = w\}$

   $\text{Pre}(abc) = \{\epsilon, a, ab, abc\}$

   $\text{Suf}(abc) = \{\epsilon, c, bc, abc\}$

Concatenation of two strings = String formed by writing the first followed by the second string. `foo`$\langle\rangle$`bar` = `foobar`

ϵ (the empty string) is the *identity* for the concatenation operation.

2.2 Formal Languages

A set of strings of symbols from some one alphabet.

Examples

$$
\begin{aligned}
&\emptyset \\
&\{\epsilon\} \\
\mathrm{Pal}(\{0,1\}) &= \text{Set of Palindromes over the alphabet } \Sigma = \{0,1\} \\
&= \{\epsilon, 0, 1, 00, 11, 010, 101, \ldots\} \\
\mathrm{RPal}_6(\{0,1\}) &= \text{Set of Reverse Palindromes over the alphabet} \\
&\quad\ \Sigma = \{A, T, C, G\} \text{ with 6 letters} \\
&= \{AAATTT, CCCGGG, TTTAAA, GGGCCC \\
&\quad\ AATATT, CCGCGG, TTATAA, GGCGCC, \\
&\quad\ ATATAT, CGCGCG, TATATA, GCGCGC, \\
&\quad\ TAATTA, GCCGGC, ATTAAT, CGGCCG, \\
&\quad\ \ldots\} \\
\Sigma^* &= \text{Set of all strings over a fixed alphabet } \Sigma \\
\Sigma = \{a\} &\Rightarrow \Sigma^* = \{\epsilon, a, aa, aaa, \ldots\} \\
\Sigma = \{0,1\} &\Rightarrow \Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, \ldots\}
\end{aligned}
$$

2.3 Set Notation

A *Set* is a collection of objects without repetition.

Σ^* = Set of all finite strings over an alphabet Σ. $|\Sigma^*| = \infty$ (countably infinite) = Cardinality of natural numbers $\mathbf{N} = \{0, 1, 2, \ldots\}$.

$\mathcal{L} \subseteq \Sigma^*$ = Subset of Σ^* = *language*

Finite sets ... specified by listing their members delimited by braces: $\{\} = \emptyset$, $\{0,1\}$, ...

Set formers...specified by a statement $P(x)$ holding true of an object in the universe.

$$\mathcal{L} = \{\sigma \in \Sigma^* | P(\sigma)\}.$$

E. g.

$$\mathrm{Pal}(\Sigma) = \{\sigma \in \Sigma^* | \sigma = \sigma^R\}$$

σ^R = Reversal of σ.

$$\mathrm{RPal}_6(\Sigma) = \{\sigma \in \Sigma^* \mid |\sigma| = 6 \wedge \sigma^C = \sigma^R\}$$

σ^C = Complement of σ. ($A^C = T$, $T^C = A$, $C^C = G$ and $G^C = C$)

$$\begin{aligned} (GCCGGC)^C &= CGGCCG \\ (GCCGGC)^R &= CGGCCG \end{aligned}$$

- $\mathcal{L}_1 \subseteq \mathcal{L}_2$, $\mathcal{L}_1$ is a subset or sublanguage of $\mathcal{L}_2$, if every member of $\mathcal{L}_1$ is a member of $\mathcal{L}_2$.

$$\forall \sigma \in \Sigma^*.(\sigma \in \mathcal{L}_1) \Rightarrow (\sigma \in \mathcal{L}_2).$$

- $\mathcal{L}_1 = \mathcal{L}_2 \Leftrightarrow \mathcal{L}_1 \subseteq \mathcal{L}_2$ and $\mathcal{L}_2 \subseteq \mathcal{L}_1$.

$$\forall \sigma \in \Sigma^*.(\sigma \in \mathcal{L}_1) \Leftrightarrow (\sigma \in \mathcal{L}_2).$$

- $\mathcal{L}_1 \underset{\neq}{\subset} \mathcal{L}_2 \Leftrightarrow \mathcal{L}_1$ is a proper subset of $\mathcal{L}_2$.

$$\begin{aligned} &\Leftrightarrow \mathcal{L}_1 \subseteq \mathcal{L}_2 \wedge \mathcal{L}_1 \neq \mathcal{L}_2 \\ &\Leftrightarrow \mathcal{L}_1 \subseteq \mathcal{L}_2 \wedge \neg(\mathcal{L}_2 \subseteq \mathcal{L}_2) \end{aligned}$$

2.4 Operations on Sets

Assume that $\mathcal{L}_1 \subseteq \Sigma_1^*$ and $\mathcal{L}_2 \subseteq \Sigma_2^*$.

- $\mathcal{L}_1 \cup \mathcal{L}_2 \equiv$ Union of $\mathcal{L}_1$ and $\mathcal{L}_2$ = $\{\sigma \mid \sigma \in \mathcal{L}_1 \vee \sigma \in \mathcal{L}_2\}$
- $\mathcal{L}_1 \cap \mathcal{L}_2 \equiv$ Intersection of $\mathcal{L}_1$ and $\mathcal{L}_2$ = $\{\sigma \mid \sigma \in \mathcal{L}_1 \wedge \sigma \in \mathcal{L}_2\}$
- $\mathcal{L}_1 \setminus \mathcal{L}_2 \equiv$ Difference of $\mathcal{L}_1$ and $\mathcal{L}_2$ = $\{\sigma \mid \sigma \in \mathcal{L}_1 \wedge \sigma \notin \mathcal{L}_2\}$
- $\mathcal{L}_1 \times \mathcal{L}_2 \equiv$ Cartesian product of $\mathcal{L}_1$ and $\mathcal{L}_2$ = $\{(\sigma_1, \sigma_2) \mid \sigma_1 \in \mathcal{L}_1 \wedge \sigma_2 \in \mathcal{L}_2\}$
- $\mathcal{P}(\mathcal{L}_1) = 2^{\mathcal{L}_1}$ = Power set of $\mathcal{L}_1$ = $\{\mathcal{L} \subseteq \Sigma^* \mid \mathcal{L} \subseteq \mathcal{L}_1\}$

- $f : \mathcal{L}_1 \to \mathcal{L}_2 =$ Map from a set $\mathcal{L}_1$ to set $\mathcal{L}_2$.

$$\forall \sigma_1 \in \mathcal{L}_1 . \exists \sigma_2 \in \mathcal{L}_2 . f(\sigma_1) = \sigma_2.$$

  - The map is *surjective* if

$$\forall \sigma_2 \in \mathcal{L}_2 . \exists \sigma_1 \in \mathcal{L}_1 . f(\sigma_1) = \sigma_2.$$

  - The map is *injective* if

$$\forall \sigma', \sigma'' \in \mathcal{L}_1 . f(\sigma') = f(\sigma'') \Rightarrow \sigma' = \sigma''.$$

  - The map is *bijective* if it is both surjective and injective

$$\forall \sigma_2 \in \mathcal{L}_2 . \exists ! \sigma_1 \in \mathcal{L}_1 . f(\sigma_1) = \sigma_2.$$

$$f^{-1}(\sigma_2) \equiv \sigma_1 \in \mathcal{L}_1 \text{ s.t. } f(\sigma_1) = \sigma_2.$$

Two sets $\mathcal{L}_1$ and $\mathcal{L}_2$ are said to have same cardinality iff there is a bijection (one to one, 1-1 map) between them.

$$|\mathcal{L}_1| = |\mathcal{L}_2| \quad \text{iff} \quad \exists f : \mathcal{L}_1 \to \mathcal{L}_2 . \mathcal{L}_1 \to^{1-1} \mathcal{L}_2.$$

If there is a bijection between $\mathcal{L}_1$ and the set of natural numbers, $\mathbf{N}$ (or integers, $\mathbf{Z}$) then $\mathcal{L}_1$ is countable or countably infinite.

2.5 A Note

The cardinality of all well-formed Pascal programs is same as that of the set of natural numbers, $\mathbf{N}$. But the cardinality of all functions, $f : \mathbf{N} \to \mathbf{N}$ is same as that of the set of all real numbers, $\mathbf{R}$. Since, it is easily seen by diagonalization that $|\mathbf{N}| < |\mathbf{R}|$, there must be a function $f : \mathbf{N} \to \mathbf{N}$ which cannot be computed by any Pascal program.

It is instructive to look at the diagonalization technique that was used in the proof to show that there are more real numbers than natural numbers. Assume that this is not the case (*i.e.* , $|\mathbf{N}| = |\mathbf{R}|$). Then there is a bijective function, $F : \mathbf{N} \to \mathbf{R} : i \mapsto r_i$. Write $r\langle j \rangle$ to denote the j^{th} bit of r in its binary expansion. Consider the following real number shown in its binary expansion:

$$\tilde{r} = 0.(r_1\langle 1\rangle \oplus 1)(r_2\langle 2\rangle \oplus 1)(r_3\langle 3\rangle \oplus 1)\ldots(r_i\langle i\rangle \oplus 1)\ldots$$

($\oplus : \{0,1\} \times \{0,1\} \to \{0,1\}$ defined as $0 \oplus 0 = 1 \oplus 1 = 0$, $0 \oplus 1 = 1 \oplus 0 = 1$.) Then in fact

$$\neg(\exists i. r_i = \tilde{r}),$$

as $r_i\langle i\rangle \neq \tilde{r}\langle i\rangle$. Since $\tilde{r} \in \mathbf{R}$, we conclude that F is not even surjective!

2.6 Functions on Languages

2.6.1 Homomorphisms

$\mathcal{L}_1 \subseteq \Sigma_1^*$ and $\mathcal{L}_2 \subseteq \Sigma_2^*$.

$$h : \Sigma_1 \to \Sigma_2^* : a_i \mapsto \sigma_i$$

Each letter $a_i \in \Sigma_1$ is mapped to a string $\sigma_i \in \Sigma_2^*$.

Extend h to Σ_1^* as follows:

$$\begin{aligned} h &: \Sigma_1^* \to \Sigma_2^* \\ &: \epsilon \mapsto \epsilon \\ &: a_1 a_2 \cdots a_n \mapsto \sigma_1 \sigma_2 \cdots \sigma_n \end{aligned}$$

$$\mathcal{L}_2 = h(\mathcal{L}_1) = \{h(\sigma) | \sigma \in \mathcal{L}_1\} \subseteq \Sigma_2^*.$$

2.6.2 Substitutions

$\mathcal{L}_1 \subseteq \Sigma_1^*$ and $\mathcal{L}_2 \subseteq \Sigma_2^*$.

$$f : \Sigma_1 \to \mathcal{P}(\Sigma_2^*) : a_i \mapsto \mathcal{K}_i (\subseteq \Sigma_2^*)$$

Each letter $a_i \in \Sigma_1$ is mapped to a set of strings: $\mathcal{K}_i \subseteq \Sigma_2^*$.

Extend f to Σ_1^* as follows:

$$\begin{aligned} f &: \Sigma_1^* \to \mathcal{P}(\Sigma_2^*) \\ &: \epsilon \mapsto \{\epsilon\} \\ &: a_1 a_2 \cdots a_n \mapsto \{\sigma_1 \sigma_2 \cdots \sigma_n | \sigma_i \in \mathcal{K}_i\} \end{aligned}$$

$$\mathcal{L}_2 = f(\mathcal{L}_1) = \bigcup_{\sigma \in \mathcal{L}_1} f(\sigma) \subseteq \Sigma_2^*.$$

3 Relation

A *relation* (or a binary relation) between two sets $\mathcal{L}_1$ and $\mathcal{L}_2$ is a set of pairs from the Cartesian product $\mathcal{L}_1 \times \mathcal{L}_2$:

$$\begin{aligned} R &\subseteq \mathcal{L}_1 \times \mathcal{L}_2 \\ \sigma_1 R \sigma_2 \quad &\Leftrightarrow \quad (\sigma_1, \sigma_2) \in R. \end{aligned}$$

$\mathcal{L}_1$ is the *domain* and $\mathcal{L}_2$ is the *range*.
Binary relation on a set $\mathcal{L}$: Range = Domain = $\mathcal{L}$.

$$\sigma_1 R \sigma_2 \Leftrightarrow (\sigma_1, \sigma_2) \in R \subseteq \mathcal{L} \times \mathcal{L}$$

3.1 Properties of a Relation

$$\begin{aligned} \forall \sigma \in \mathcal{L}. \sigma R \sigma \quad &\rightarrow \quad \text{Reflexive} \\ \forall \sigma \in \mathcal{L}. \neg(\sigma R \sigma) \quad &\rightarrow \quad \text{Irreflexive} \\ \forall \sigma, \sigma' \in \mathcal{L}. \sigma R \sigma' \Rightarrow \sigma' R \sigma \quad &\rightarrow \quad \text{Symmetric} \\ \forall \sigma, \sigma' \in \mathcal{L}. \sigma R \sigma' \Rightarrow \neg(\sigma' R \sigma) \quad &\rightarrow \quad \text{Asymmetric} \\ \forall \sigma, \mu, \lambda \in \mathcal{L}. \sigma R \mu \wedge \mu R \lambda \Rightarrow \sigma R \lambda \quad &\rightarrow \quad \text{Transitive} \end{aligned}$$

E. g. Relation $< \subseteq \mathbf{Z} \times \mathbf{Z}$ defined as $(a, b) \in <$ iff $a < b$, is irreflexive, asymmetric and transitive.

3.2 Equivalence Relation

A reflexive, symmetric and transitive relation.

$$\forall a, b, c. a \equiv a, a \equiv b \Rightarrow b \equiv a, a \equiv b \wedge b \equiv c \Rightarrow a \equiv c.$$

Let R be an equivalence relation on a set $\mathcal{L}$. Then R partitions $\mathcal{L}$ into disjoint nonempty equivalence classes.

$\sigma \in \mathcal{L}$; the equivalence class containing σ,

$$[\sigma]_R = \{\mu \in \mathcal{L} | \mu R \sigma\}$$

This set is nonempty as $\sigma R \sigma$ and $\sigma \in [\sigma]_R$.

$$\sigma R \mu \Rightarrow [\sigma]_R = [\mu]_R.$$

Note that $\lambda \in [\sigma]_R$ implies that $\lambda R \sigma$ and by transitivity $\lambda R \mu$ and $\lambda \in [\mu]_R$. Thus $[\sigma]_R \subseteq [\mu]_R$. But by symmetry, $\mu R \sigma$ and $[\mu]_R \subseteq [\sigma]_R$.

$$\begin{aligned} & [\sigma]_R \cap [\mu]_R \neq \emptyset \\ & \quad \Rightarrow \quad \exists \lambda . \sigma R \lambda \wedge \lambda R \mu \\ & \quad \Rightarrow \quad \sigma R \mu \Rightarrow [\sigma]_R = [\mu]_R. \end{aligned}$$

4 Graph

A graph $G = (V, E)$ consists of a

- Finite set of vertices $= V$
- Finite set of edges $= E \subseteq V \times V =$ Binary relations over V

If the binary relation is symmetric (*i.e.* , $v_1 E v_2 \Rightarrow v_2 E v_1$) then the graph is said to be an undirected graph.

If the binary relation is not symmetric (but not necessarily asymmetric) then the graph is said to be a directed graph (digraph).

A path in a graph (simple path) is a sequence of distinct vertices

$$v_1 v_2 v_3 \cdots v_{k-1} v_k \quad (k \geq 2)$$

such that $\forall i \geq 1 . v_i E v_{i+1}$.

In a digraph G, v is a predecessor of w (w is a successor of v), if $v, w \in V$ and $v E w$.

5 Tree

(Ordered Directed Tree)

A tree is a digraph with the following properties:

- There is one vertex (called the *root*) that has no predecessor and from which there is a path to every vertex.
- Each vertex except the root has exactly one predecessor.
- [The successors of each vertex are ordered from the left.]

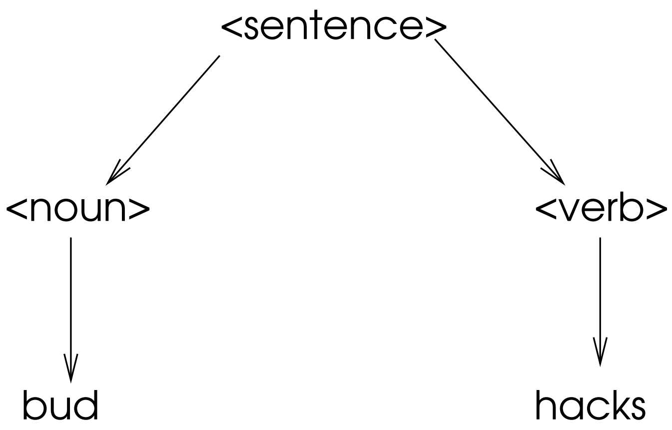

Figure 3: **Tree Example**