Lecture I
FINITE AUTOMATA

Lecture 1: Honors Theory, Spring'02, Yap

We introduce finite automata (deterministic and nondeterministic) and regular languages. Some general concepts and notations are gathered at the end for easy reference.

§1. Regular Sets and DFA

We give an intuitive introduction to state diagrams using the example of an automatic door closer (see figure 1(a)).

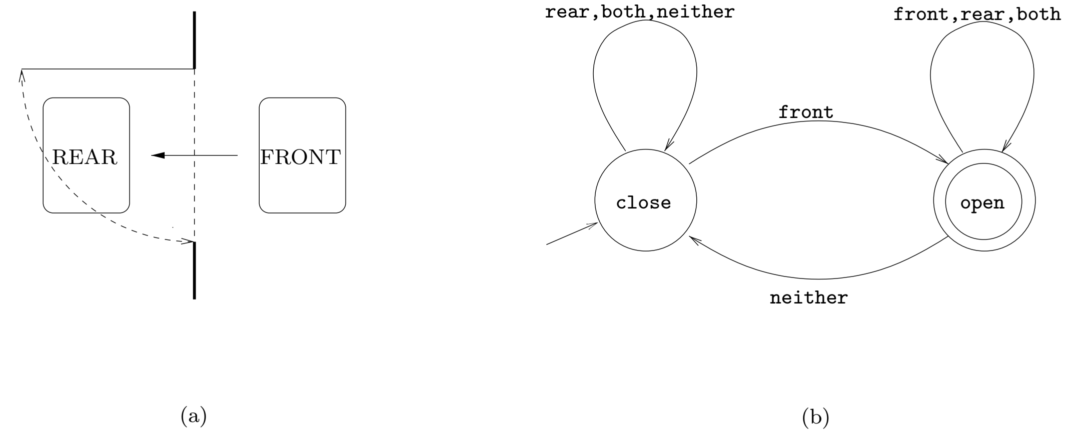

Figure 1: (a) Door Plan, (b) State diagram

The door can be in one of two states: `open` or `close`. There are two sensor pads, one in front and one behind the door. The door opens towards the rear, allowing a person on the front pad to pass through the door. There are 4 sensor inputs, corresponding to the presence or absence of pressure on the pads: `front`, `rear`, `neither` and `both`. Note that `front` means "front pad only", `neither` means "neither pad", etc. The state transitions are triggered by sensor inputs, and fully described in figure 1(b). From either state, there is only one input that can trigger a state change: if in the `close` state, we can only change to `open` if the input is `front`. In particular, we do not open the door if there is someone in the rear, as the door can hit that person. Similarly, from the `open` state, only the `neither` input can trigger a state change. This representation of the logic using a digraph with labels on edges is called a **state diagram**.

Another example of an smart card wallet.

State diagrams can also be formalized as **finite automata**, a term that emphasizes the finiteness of the number of states. A **deterministic finite automata** (dfa) is a 5-tuple

$$M = (Q, \Sigma, \delta, q_0, F)$$

such that

- Q and Σ are finite sets, called the **state set** and the **input alphabet** (respectively).

- $q_0 \in Q$ is the **start state**.
- $F \subseteq Q$ are the **final states**.
- $\delta : Q \times \Sigma \to Q$ is the **transition function**.

We want to associate a language with each dfa. But first we need a useful definition. Given δ, we define the **extended transition function**

$$\delta^* : Q \times \Sigma^* \to Q$$

as follows:

$$\delta^*(q, w) = \begin{cases} q & \text{if } \ w = \epsilon, \\ \delta^*(\delta(q, a), w') & \text{if } \ w = aw', a \in \Sigma, w' \in \Sigma^*. \end{cases}$$

We also write $\delta^*(w)$ for $\delta^*(q_0, w)$. Finally, the **language accepted by** M is defined to be the set

$$L(M) = \{w \in \Sigma^* : \delta^*(w) \cap F \neq \emptyset\}.$$

If $w \in L(M)$, we also say that M **accepts** w. A language is **regular** if it is accepted by some dfa.

Example. For the door closer dfa above, we have $Q = \{\texttt{close}, \texttt{open}\}$ and $\Sigma = \{\texttt{front}, \texttt{rear}, \texttt{both}, \texttt{neither}\}$. We also have $\delta(\texttt{close}, a) = \texttt{open}$ iff $a = \texttt{front}$, and $\delta(\texttt{open}, a) = \texttt{close}$ iff $a = \texttt{neither}$. In this example, the start state and final states are hardly important. Nevertheless, since our formalism requires it, we will arbitrarily let $q_0 = \texttt{close}$ and $F = \{\texttt{open}\}$. In state diagrams (see figure 1(b)), we indicate the start state by an arrow from nowhere that points to q_0, and circle each final state such as `open`.

We leave it as an exercise to formalize the notion of "deterministic state diagrams" that is to equivalent to dfa's.

§2. NFA

There are many possible variations in dfa's: in some applications, we may not be interested in final states but are only interested in the repetitive non-stopping behavior. This is true of the door closer example. Another large class of such examples come from any GUI design in computer window interfaces. Sometimes we are interested in output actions associated with states, or with transitions. The former are called Mealy machines. We may even allow several simultaneous input strings. Perhaps the most important variation is the introduction of nondeterminism. Nondeterminism is the property of machines that may have more than one next course of action.

Formally, a **nondeterministic finite automata** (nfa) is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F\}$ as in a dfa. The only change is that δ is new a multivalued function in the sense that $\delta(q, a)$ has any number of next state $q' \in Q$. If we collect the set of these q', we obtain a set that is possibly empty. Hence,

$$\delta : Q \times \Sigma_\epsilon \to 2^Q$$

where Σ_ϵ is $\Sigma \cup \{\epsilon\}$.

The intuitive idea of nondeterminism is that the finite automata has **choices** in making its transitions. There are two kinds of choices: (I) From state q, it can read the next input symbol (say a) and go to any state in $\delta(q, a)$. (II) From state q, it can choose NOT to read the next symbol, and go to any state in $\delta(q, \epsilon)$. We say that the nfa accepts an input w if it can reach some final state by starting from q_0, after reading all the symbols of w.

To formalize this, we extend the function δ to a new function

$$\delta^* : 2^Q \times \Sigma^* \to 2^Q.$$

For $P \subseteq Q$ and $w \in \Sigma^*$, the set $\delta^*(P, w)$ is, intuitively, the set of all states that we can reach by reading w, starting from any state of P. With such a definition, we can define the language accepted by an nfa M to be

$$L(M) = \{w \in \Sigma^* : \delta^*(\{q_0\}, w) \cap F \neq \emptyset\}.$$

Two preliminary steps will be helpful. First we can easily extend δ to

$$\delta_1 : 2^Q \times \Sigma \to 2^Q$$

where $\delta_1(P, a) = \bigcup_{p \in P} \delta(p, a)$. Next, we write $q \xrightarrow{\epsilon} q'$ if there is a finite sequence of ϵ-transitions from q to q'. Then

$$\delta^*(P, w) = \begin{cases} \{q \in Q : (\exists p \in P)[p \xrightarrow{\epsilon} q]\} & \text{if } \ w = \epsilon, \\ \delta^*(\delta_1(\delta^*(P, w'), a), \epsilon) & \text{if } \ w = w'a, a \in \Sigma, w' \in \Sigma^*. \end{cases}$$

Again, let $\delta^*(w) = \delta^*(q_0, w)$.

Theorem 1 (Rabin-Scott) *NFA's accept precisely the set of regular languages.*

Proof. Clearly every regular language is accepted by an nfa, since a dfa is an nfa. Conversely, given a nfa $M = (Q, \Sigma, \delta, q_0, F)$, we construct a dfa $\overline{M} = (\overline{Q}, \Sigma, \overline{\delta}, \overline{q_0}, \overline{F})$ such that $L(M) = L(\overline{M})$. The idea is quite simple: $\overline{M}$ must simulate all possible transitions of M simultaneously. This means for al $w \in \Sigma^*$, we want the extension of $\overline{\delta}$ to compute the set

$$\overline{\delta}^*(w) = \{ \text{ all the states in } Q \text{ that can be reached from } q_0 \text{ by } M \text{ after reading } w\}. \qquad (1)$$

Once we have this goal, it is natural to define the state set $\overline{Q} = 2^Q$, the start state $\overline{q_0} = \delta^*(q_0, \epsilon)$ and the set of final states to be $\overline{F} = \{P \subseteq Q : P \cap F \neq \emptyset\}$. Now it is easy to see that $L(M) = L(\overline{M})$.

Thus it remains to define the transition function so that (1) is satisfied. For $P \subseteq Q$ and $a \in \Sigma$, we define

$$\overline{\delta}(P, a) = \cup_{q \in P} \delta^*(q, a).$$

Note that $\overline{\delta}(\emptyset, a) = \emptyset$. This completes our description of the dfa $\overline{M}$. **Q.E.D.**

Two nfa's are said to be be **equivalent** if they accept the same language. The preceding lemma proves that nfa's and dfa's are equivalent. This equivalence is very useful, as illustrated in the following proof.

Theorem 2 *Regular languagues are closed under union, intersection, complementation, concatenation and Kleene-star. In particular, if $A, B \subseteq \Sigma^*$ are regular languages then so are*

$$A \cup B, \quad \overline{A} = \Sigma^* \setminus A, \quad A \cap B, \quad AB, \quad A^*.$$

Proof. (a) Let us show that $C = A \cup B$ is regular. Assume that A and B are accepted by the nfa's M_A and M_B that are in the following "nice form": $M_i = (Q_i, \Sigma, \delta_i, q_{0i}, F_i)$ where $i = A, B$. Also, $q_{0i} \neq q_{fi}$. It is easy to put any nfa into this "nice form".

We construct $M_C = (Q_C, \Sigma, \delta_C, q_{0C}, F_C)$ to accept $C = A \cup B$. Let $Q_C = Q_A \cup Q_B \cup \{q_{0C}\}$, where q_{0C} is a new symbol. Let $F_C = F_A \cup F_B$, and define δ_C has all the transitions of δ_A and δ_B as well as the new transition

$$\delta_C(q_{0C}, \epsilon) = \{q_{0A}, q_{0B}\}.$$

It is easy to see that $L(M_C) = A \cup B$.

(b) To show that $C = \overline{A} = \Sigma^* \setminus A$ is regular, assume M_A is a dfa that accepts A. We construct a nice nfa M_C to accept C: start state is a new q_{0C}, final state set is $F_C = \{q_{fC}\}$ where $Q_C = Q_A \cup \{q_{0C}, q_{fC}\}$. The transitions are those of M_A together with $\delta_C(q_{0C}, \epsilon) = \{q_{0A}\}$ and for each $q \in Q_A \setminus F_C$ (i.e., non-final state q of M_A),

$$\delta_C(q, \epsilon) = \{q_{fC}\}.$$

This ensures that we accept the complement of A.

(c) Remaining cases. It follows that the intersection of two regular languages is regular: $A \cap B = \overline{\overline{A} \cup \overline{B}}$. Similar constructions will show AB and A^*. **Q.E.D.**

Note how we exploited the equivalence of nfa and dfa in this proof. When we want to show the "power" of regular languages, it is easier to use nfa's. E.g., the constructed automaton M_C is always an nfa. And in part (a), the automata can be assumed to be in "nice form" only because we assume an nfa. Conversely, when we want to "limit" the power of regular languages, we can use a dfa. E.g., we started out with a dfa M_A in part (b).

REMARK: we can extend the concept of a dfa by defining it to be a nfa in which $|\delta(q,a)| \le 1$ for all q, a. Such a dfa can sometimes be "stuck" with no next state. The advantage of this is that when we draw state diagrams, we can avoid drawing lots of extraneous edges and labels. The idea of "getting stuck" must be distinguished from the idea of staying in the same state. For instance, one could introduce the convention for state diagrams where, if there is no explicit rule for a transition from state q on input a, then the implicit rule is $\delta(q,a) = q$. The problem is that this convention means that we can never get stuck. See the exercise for regular languages that requires fewer states if we allow the nfa to have stuck states.

§3. NOTATIONS AND TERMINOLOGY

This section is a handy reference for common notations.

Numbers. $\mathbb{N}$ is the set of natural numbers $0, 1, 2, \ldots$. $\mathbb{Z}$ is the set of integers $0, \pm 1, \pm 2, \ldots$. $\mathbb{Q}$ is the set of rational numbers (ratio of two integers, where the denominator is non-zero). $\mathbb{R}$ is the set of real numbers.

Sets. If X is a well-defined set, we can introduce a new set by writing $\{x \in X : P(x)\}$ where $P(x)$ is a predicate on X. For instance, $\{x : x \in \mathbb{N}, x \text{ is even}\}$ denotes the set of even natural numbers. The **size** $|X|$ of X is the number of its elements. The **empty set** is denoted $\emptyset$ and thus $|\emptyset| = 0$. If $X = \bigcup_{i \in I} X_i$ and the X_i's are pairwise disjoint, we indicate this fact by writing $X = \biguplus i \in I X_i$, and call $\{X_i : i \in I\}$ a **partition** of X. The set of all subsets of a set X is the **power set** of X, denoted 2^X. We have $\left|2^X\right| = 2^{|X|}$. A subset of X of size k is called a k**-set**. The set of all k-sets of X is denoted $\binom{X}{k}$. This recalls the binomial function $\binom{n}{k} = \frac{n!}{k!(n-k)!}$. In fact, we have $\left|\binom{X}{k}\right| = \binom{|X|}{k}$. The Cartesian product of X and Y is $X \times Y = \{(x,y) : x \in X, y \in Y\}$. This can be generalized to $X_1 \times X_2 \times \cdots X_n$ or $\prod_{i=1}^n X_i$ (this notation exploits the associativity of Cartesian product). In case $X_1 = X_2 = \cdots = X_n$, we write X^n for $\prod_{i=1}^n X$.

Relations. An n**-ary relation** R is a subset of a Cartesian product $\prod_{i=1}^n X_i$ where S_i are sets. We say " the relationship $R(a_1, \ldots, a_n)$ holds" in case $(a_1, \ldots, a_n) \in R$. If $n = 1$, R is also called a **predicate** on S_1. For a binary relation R, we may use an infix notation, and write write $a_1 R a_2$ instead of $R(a_1, a_2)$. A binary relation $R \subseteq S^2$ is **reflexive** if aRa for all $a \in S$; **symmetric** if aRb imples bRa; and **transitive** if aRb and bRc implies aRc. A **partial order** on S is a binary relation $R \subseteq S^2$ that is reflexive, symmetric and transitive. We usually write $a \le b$ for aRb in case R is a partial order. If, in addition, we have either

$a \leq b$ or $b \leq a$ for all $a, b \in S$, then R is called a **total order**. A total order A **well order** on S is a partial ordering $\leq$ such that every subset $A \subseteq S$ has a least element $a \in A$ (i.e., for all $b \in A$, $a \leq b$). Note that a well order is a total order.

Functions. A function f has two associated sets D and R, called its **domain** and **range** respectively. We write $f : D \to R$ to indicate this relationship. The function f specifies for each $x \in D$, a value in R which we denote by $f(x)$. A function f is an **injection** (or **one-one**) if $x \neq y$ implies $f(x) \neq f(y)$; it is a **surjection** (or **onto**) if $\{f(x) : x \in D\} = R$; it is a **bijection** (or **one-one onto**) if it is both an injection and a bijection.

Graphs. There are many varieties of graphs, but two varieties are important for us. A **directed graph** or **digraph** is a pair $G = (V, E)$ where V is any set and $E \subseteq V^2$. A **undirected graph** or **bigraph** ("bi" for bidirectional) is a pair $G = (V, E)$ where V is any set and $E \subseteq \binom{V}{2}$. For both kinds of graphs, the elements of V and E are called **vertices** and **edges**, respectively. We denote an edge by (u, v) where $(u, v) \in E$ (if digraph) or $\{u, v\} \in E$ (if bigraph). A **path** is a sequence $p = (v_1, v_2, \ldots, v_k)$ where (v_i, v_{i+1}) are edges for $i = 1, \ldots, k-1$. We call p a path from v_1 to v_k.

Formal language theory. An **alphabet** Σ is a finite set of symbols, called **letters**. A **word** (or **string**) w over Σ is a finite sequence of letters of Σ,

$$w = a_1 \cdots a_n, \quad (n \geq 0)$$

where $a_i \in \Sigma$. The **length** of w is n, denoted $|w|$. The **empty word** is denoted ϵ; thus $|\epsilon| = 0$. We set of all words over Σ is denoted Σ^*. Also, $\Sigma^+ := \Sigma^* \setminus \{\epsilon\}$ is the set of non-empty words over Σ. If $a \in \Sigma$, then define the counting function $\#_a : \Sigma^* \to \mathbb{N}$ where $\#_a(w)$ is the number of occurences of a in w. The ith symbol in a word w is denoted $w[i]$ where $i = 1, \ldots, |w|$. Assume $w[i]$ is undefined for i outside this range. The reverse of a word w is denoted w^R. Thus $w[i] = w^R[n - i + 1]$ where $n = |w|$. A palindrome is a word w such that $w = w^R$. Let Σ^* denote the set of words over Σ. A **language** is a pair of the form (Σ, A) where $A \subseteq \Sigma^*$. We usually refer to "A" as the language (so the alphabet Σ is implicit). The **complement** of a language (Σ, A) is $(\Sigma, \Sigma^* \setminus A)$, denoted co-$A$. If $\mathcal{C}$ is a class of languages, let co-$\mathcal{C} = \{$co-$\mathcal{A} : \mathcal{A} \in \mathcal{A}\}$. Occasionally, we consider in infinite strings $w = a_0 a_1 a_2 \cdots$ where $a_i \in \Sigma$. Alternatively, $w : \mathbb{N} \to \Sigma$ so $w(n) = a_n$. We call w an ω**-word** (or *omega*-sequence) over Σ. Let Σ^ω denote the set of ω-words over Σ.

Classes and Families. Above, we only allowed new sets to be constructed from known sets. Otherwise, there is some danger in using the concept of "sets" too liberally, giving rise to various logical paradoxes. For instance, the set of all sets will lead to such a paradox. In general paradoxes arise for "very large sets" such as the set of all sets. Logicians are careful to introduce other terms like "classes" or "families" for such large sets. We will use these terms informally (making no true distinction between them except for usage convention) for certain large sets in our book. For instance, consider $\mathcal{L}$, the "set of all languages". This is "large" because each language depends on an underlying alphabet, and there is no fixed[1] set alphabet that we wish to commit ourselves too. Hence, this set is not a mathematical set. It has many of the features of sets – we can recognize members of it. Yet we never get into trouble because each particular use can be justified or suitably circumscribed. Moreover, certain set operations such as power set ought not to be performed. Two usage conventions: we consistently refer to a "large sets of languages" as a **class of languages** (or class for short). Likewise, we refer to a "large set of machines" as a **family of machines**. For example, we talk of the "class of polynomial-time languages" and the "family of deterministic Turing machines".

[1]Later in the book, we will actually establish two acceptable conventions in which these "large sets" can be completely rigorous. These are called conventions α and convention β respectively.

NOTES

Finite automata was first studied in the 1950s. Most elementary books on the theory of computation have a treatment of finite automata and regular languages. For instance, Sipser [4] and Lewis and Papadimitriou [2]. The door closer example is from Sipser. The Rabin-Scott theorem appeared in [3]. An easy introduction to set theory where some paradoxes of large sets are discussed is Halmos' book [1].

Exercises

Exercise 3.1: Consider the following transformation: suppose M is a nfa with start state q_0 and final states $F = \{q_f\}$. We transform it to M' by making $F = \{q_0\}$, and adding the transition $\delta(\epsilon, q_f) = q_0$. This is illustrated in figure 2. Prove or disprove: $L(M') = L(M)^*$. ♢

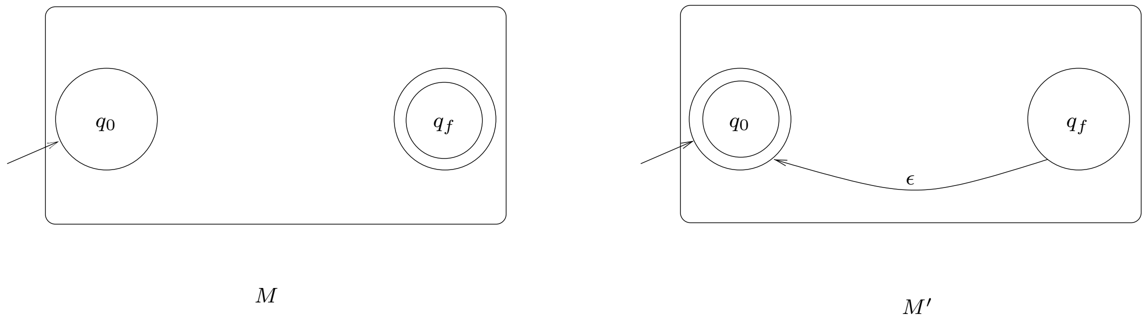

Figure 2: M to M' transformation.

Exercise 3.2: For $n \geq 2$, let $\Sigma_n = \{a_1, \ldots, a_n\}$ be an alphabet with n letters and $L_n = \{w \in \Sigma_n^* : \#_{a_i}(w) = 0 \text{ for some } i = 1, \ldots, n\}$.
(a) Show an nfa that accepts L_n with $n + 1$ states.
(b) Show a dfa that accepts L_n with 2^n states.
(c) Show that every dfa that accepts L_n has at least 2^n states. ♢

Exercise 3.3: Recall that our nfa can "get stuck". Show that you need more than $n + 1$ states if the nfa is not allowed to get stuck in part (a) of the previous question. ♢

Exercise 3.4: (a) Formalize the concept of deterministic state diagram using digraphs, and the language that it accepts.
(b) Prove that your concept is equivalent to dfa's: for each dfa, there is a deterministic state diagram accepting the same language. Conversely, from any deterministic state diagram, there is a dfa for the same language. ♢

End Exercises

References

[1] P. R. Halmos. *Naive Set Theory.* Van Nostrand Reinhold Company, New York, 1960.

[2] H. R. Lewis and C. H. Papadimitriou. *Elements of the Theory of Computation.* Prentice-Hall Inc, Upper Saddle River, New Jersey, second ed. edition, 1998.

[3] M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM J. of Research and Development*, 3:114–125, 1959.

[4] M. Sipser. *Introduction to the Theory of Computation.* PWS Publishing Co, Boston, 1997.