Finite Automata

Notes on Automata and Theory of Computation

Chia-Ping Chen

Department of Computer Science and Engineering
National Sun Yat-Sen University
Kaohsiung, Taiwan ROC

Finite Automata

- We begin a systematic treatment of the automata theory with the simplest kind: the class of **finite acceptors**.
  - They are characterized by having no temporary storage.
  - A finite amount of information can be retained by using different states.
- We use abbreviation **fa** for finite acceptor.

Deterministic Finite Acceptor

- A deterministic finite acceptor (**dfa**) M is defined by a quintuple (5-tuple)

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

  - Q is a finite set of states
  - Σ is a finite set of symbols called **alphabet**
  - $\delta : Q \times \Sigma \to Q$ is a *total* function called **transition function**
  - $q_0 \in Q$ is the **initial state**
  - $F \subseteq Q$ is the set of **final states**

Operation of dfa

- Initially, M is in state q_0.
- At any instant, it reads an input symbol and transits to a new state according to the transition function.
- It continues until no further changes can be made, i.e, the entire input string has been read.
- Since the transition function of M is a total function, there is always a state to transit to unless there are no more input symbols.

State Transition

- The transition function can be represented by a labeled directed graph, called **transition graph**, where
  - a vertex represents a state
  - an edge from a vertex (state) to another is labeled by the symbol needed for the corresponding state transition
  - conventionally, q_0 is indicated by an incident edge from “space”, and the final states are indicated by double circles.
- It can also be represented by a table, called **transition table**.
- Let’s look at some examples.

Extended Transition Function

- We are interested in the state a dfa is in when an input string has been used.
- This leads to the **extended transition function**

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

  where the second argument is a string rather than a symbol.
- Based on δ, δ^* is defined recursively by

  $$\begin{aligned}\delta^*(q, \lambda) &= q \\ \delta^*(q, wa) &= \delta(\delta^*(q, w), a)\end{aligned}$$

Transition Graph and δ^*

- Let M be a dfa. Let G_M be its transition graph and δ^*_M be its extended transition function.
- For any states $q_i, q_j \in Q$ and $w \in \Sigma^+$, $\delta^*_M(q_i, w) = q_j$ if and only if there is a walk in G_M with label w from vertex q_i to vertex q_j.
  - For $|w| = 1$, it is obvious.
  - Suppose it is true for $|w| = n$. For $|w| = n + 1$, write $w = va$ with $|v| = n$ and $a \in \Sigma$. By definition, $\delta^*_M(q_i, w) = \delta_M(\delta^*_M(q_i, v), a)$, so $\delta^*_M(q_i, w) = q_j$ if and only if there exists q_k such that $q_k = \delta^*_M(q_i, v)$ and $\delta_M(q_k, a) = q_j$. This amounts to a walk in G_M from q_i to q_j (via q_k).

Acceptance by dfa

- Let w be a string. It is said to be **accepted** by M if

$$\delta^*(q_0, w) \in F.$$

  Otherwise it is said to be **rejected**.

- The **language** of M is the set of strings accepted by M.

$$L(M) = \{w : w \text{ accepted by } M\}.$$

- We look at a few examples from text.

Regular Languages

- A language L is **regular** if it is accepted by a dfa.

$$\exists\ M\ s.t.\ L = L(M).$$

- A dfa defines a regular language.
- To show a language is regular, we can construct a dfa for it. We look at a few examples.
- There are other ways to prove a language to be regular. They will be discussed later.
- To show a language is not regular, we often prove it does not satisfy some necessary condition for being regular.

Nondeterministic Finite Automata

- **Nondeterminism** allows a set of possible moves in a given configuration.
- Since determinism is a special case of nondeterminism, nfa is more general than dfa.
- Formally, we define an nfa $N = (Q, \Sigma, \delta, q_0, F)$ with

$$\delta : Q \times (\Sigma \cup \{\lambda\}) \to 2^Q.$$

  - The value of δ is a set of states, including the empty set.
  - The inclusion of λ allows an nfa to make a move without using any input symbol.

δ^* for nfa

- Without λ-transition, the transition function of an nfa can be extended to have a string as the second argument. $\delta^* : Q \times \Sigma^* \to 2^Q$ is defined recursively by

$$\delta^*(q, \lambda) = \{q\}$$

$$\delta^*(q, wa) = \bigcup_{q_k \in \delta^*(q,w)} \delta(q_k, a)$$

- With λ-transition, δ^* is slightly more complicated.

$\delta^*(q, \lambda) = \text{closure}(q) =$ set of states reachable from q by λ

$$\delta^*(q, wa) = \bigcup_{r \in \bigcup_{q_k \in \delta^*(q,w)} \delta(q_k,a)} \text{closure}(r)$$

δ^* and Transition Graph

- The extended transition function is defined such that $\delta^*(q_i, w)$ contains q_j if and only if there is a walk in the transition graph from q_i to q_j labeled w.
- How do we decide whether there is a walk labeled w between q_i and q_j (q_0 and $q \in F$ especially)?
- We can enumerate the walks originating from q_i. If any of these walks ends at q_j and is labelled as w, then the answer is yes.

Enumerating Walks

- The set of walks starting from a particular node, say v, in a graph can be enumerated.
- The idea is to enumerate by the length of walk.
  - Start with length-0, which is v itself.
  - Suppose we have enumerated the walks up to length n. To enumerated the walks of length $n + 1$, we extend the walks of length n in the enumeration order. If there are multiple extensions, then they are enumerated by the order of destination node.

Lengths to be Checked

- *Without* λ-transition, the length of a walk equals the length of the label (string). So the decision can be made within $|w|$ iterations.
- *With* λ-transition, the length of a walk may be larger than the length of the label. Yet, there is still a limit on the length of walk to be inspected. The reason is that for each repeated λ edge, there must be an edge in between with non-λ label. Otherwise we have a cycle with λ label which does not affect acceptance. If the graph has n λ-edges, at most n λ edges can be used without repetition, so every non-λ label is accompanied by at most n λ-edges.

Acceptance by nfa

- Let w be a string. It is said to be accepted by an nfa N if there exists a sequence of possible moves that put N in a final state at the end of w. Otherwise it is said to be rejected.
- The language of N is the set of strings accepted by N.

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

- In other words, the language consists of all strings w for which there is a walk labeled w from the initial vertex to a final vertex.

Why Nondeterminism

- Digital computers are completely deterministic. Why bother with nondeterministic automata if we are modeling real systems?
- Many problems require one to find an optimal solution. At the point of uncertainty, determinism solves this problem by backtracking, which is often inefficient. Nondeterminism allows one to solve such problems without backtracking.
- Nondeterminism also has a theoretical significance. It is used in stating how hard a computable problem is.

Equivalence of nfa and dfa

- Two finite acceptors are **equivalent** if the languages they accept are the same,

$$L(M_1) = L(M_2).$$

- We show that for any nfa N, there is an equivalent dfa for N. So the set of languages accepted by nfa's is a subset of the set of languages accepted by dfa's.
- By definition, a dfa is a special kind of nfa, so the set of languages accepted by dfa's is a subset of the set of languages accepted by nfa's.
- We conclude that dfa and nfa are equivalent, in the sense that they accept the same set of languages.

Constructive Proof

- We will show that for a given nfa, there exists an equivalent dfa, by construction.
- After an nfa has read a string w, while the state it is in is uncertain, the set of states the nfa may be in is certain (dictated by δ^*).
- For a dfa, the state it is in is certain after reading an input string.
- So the idea in establishing equivalence is to link a set of states in nfa to a state in the equivalent dfa.
- Since the nfa has $|Q|$ states, an equivalent dfa has at most $2^{|Q|}$ states, which is finite.

Procedure nfa-to-dfa

- Create a graph G_D with vertex $\{q_0\}$. This is the initial vertex.
- Repeat until nothing further can be done.
  - Take any vertex $\{q_i, q_j, \ldots, q_k\}$ that has no outgoing edges for some $a \in \Sigma$. Let

$$\{q_l, q_m, \ldots, q_n\} = \bigcup_{q \in \{q_i, q_j, \ldots, q_k\}} \delta_N^*(q, a),$$

    add an edge to $\{q_l, q_m, \ldots, q_n\}$ with label a.
- A vertex of G_D containing any $q \in F$ is a final vertex.
- Make $\{q_0\}$ a final vertex if λ is in the language.

Reduction of Finite Automata

- While a given dfa defines a unique language, a given language may be accepted by more than one dfa's.
- We want to reduce the number of states for practical and theoretical reasons.
- We introduce the concept of **indistinguishable states**.

Indistinguishable States

- Two states p and q of a dfa are said to be **indistinguishable** if

$$\delta^*(p, w) \in F \Leftrightarrow \delta^*(q, w) \in F, \quad \forall w \in \Sigma^*.$$

- Otherwise they are said to be **distinguishable**.
- Note that indistinguishability has the properties of an equivalence relation. Thus, we can partition Q into subsets of indistinguishable states.

Procedure: `mark`

1. Remove all inaccessible states, which are the vertices in the transition graph that cannot be reached by the initial vertex. This can be done by enumerating the simple paths of the graph starting from q_0.
2. Consider all pairs (p, q). If $p \in F, q \notin F$, then mark (p, q).
3. Repeat until no previously unmarked pairs are marked:
   - For any $a \in \Sigma$ and pairs (p, q) that are not yet marked, if $p_a = \delta(p, a)$ and $q_a = \delta(q, a)$ has been marked, then mark (p, q).

Properties

- A marked pair (p, q) is indeed distinguishable, since there exists a sequence of symbols leading p to a final state and q to a non-final state.
- A pair (q_i, q_j) is marked at pass n if and only if there exists $a \in \Sigma$ and q_k, q_l marked at pass $n - 1$, such that

$$\delta(q_i, a) = q_k, \;\; \delta(q_j, a) = q_l.$$

  Otherwise (q_i, q_j) would have been marked earlier than pass n.

Theorem

- Procedure `mark` is sure to terminate since there is only a finite number of pairs.
- Procedure `mark`, when applied to a dfa, determines all pairs of distinguishable states.

Proof

- We prove that, at the completion of pass n all pairs distinguishable by a string of length no more than n are marked, by induction on n.
  - The basis case for $n = 0$ is step 2.
  - By the inductive assumption, at the beginning of pass n, all pairs distinguishable by a string of length $n - 1$ or less have been marked. A pair distinguishable by a string of length n will be marked by the properties.
- Suppose the procedure terminates at pass m, and an unmarked pair, say (p, q), cannot be distinguished by a string of length $m - 1$ or less. Then (p, q) cannot be distinguished at pass $m + 1$ since everything would stay the same way.

Procedure: `reduce`

1. Use procedure `mark` to generate equivalence classes of indistinguishable states.
2. For each equivalence class, say $\{q_i q_j \ldots q_k\}$, create a new state labeled $ij \ldots k$ for $\widehat{M}$.
3. For each transition $\delta(q_r, a) = q_p$, find the classes q_r and q_p belong to. Add an edge from the class of q_r to the class of q_p with label a.
4. The initial state $\widehat{q_0}$ of $\widehat{M}$ is the state whose label includes 0.
5. $\widehat{F}$ is the set of states with label containing i such that $q_i \in F$.

Minimization Theorem

- Given any dfa M, application of the procedure `reduce` yields a dfa $\widehat{M}$ such that $L(M) = L(\widehat{M})$.
- $\widehat{M}$ is minimum in the sense that there is no other dfa with a smaller number of states to accept $L(M)$.
- Since all states of $\widehat{M}$ are accessible, there exist strings $w_1, \ldots, w_m$ such that $\widehat{\delta}^*(q_0, w_i) = p_i$. Suppose M_1 has less states, then there must exist k, l such that $\delta_1^*(q_0, w_k) = \delta_1^*(q_0, w_l)$. Since p_k and p_l are distinguishable, there exists x such that $\widehat{\delta}^*(q_k, x) = \widehat{\delta}^*(q_0, w_k x) \in F$ but $\widehat{\delta}^*(q_l, x) = \widehat{\delta}^*(q_0, w_l x) \notin F$. However, for M_1, both $w_k x$ and $w_l x$ end in the same state.