Theory of computation unit

- Languages, finite automata, and regular expressions (Monday)
- Equivalence of models of computation (**today**)
- The lambda calculus (next week Monday)
- Computability and tractability (next week Wednesday)

Today:

- Statement of the big result
- Breakdown into smaller results
- Proof of parts

A **deterministic finite automaton** is a quintuple $M = (Q, \Sigma, \delta, q_0, F)$, where

- Q is a finite set of states
- Σ is an alphabet
- $\delta : (\Sigma, Q) \to Q$ is a state transition function
- $q_0 \in Q$ is the start (initial) state
- $F \subseteq Q$ is the set of accept (final) states

For a DFA M, define the function $\alpha : \Sigma* \to Q$ as the following, where $s \in \Sigma*$ and $n = |s|$:

$$\alpha(s) = \delta(s_{n-1}, \delta(s_{n-2}, \ldots \delta(s_1, \delta(s_0, q_0))))$$

A DFA M **accepts** a language $L \subseteq \Sigma*$ if $\forall s \ \in \Sigma*$, $s \in L$ iff $\alpha(s) \in F$.

For DFA M, let $L(M)$ be the language of strings that are accepted by M.

A **nondeterministic finite automaton** is a quintuple $M = (Q, \Sigma, \delta, q_0, F)$, where

- Q is a finite set of states
- Σ is an alphabet
- Δ is a relation from $(\Sigma \cup \{\varepsilon\}, Q)$ to Q
- $q_0 \in Q$ is the start (initial) state
- $F \subseteq Q$ is the set of accept (final) states

For an NFA M, define the relation β from $\Sigma*$ to Q defined so that for $s \in \Sigma*$, $|s| = n$, and $p_n \in Q$, $(s, p_n) \in \beta$ if there exists a sequence of states $q_0, p_1, p_2, \ldots p_m$ and a sequence of elements $t_0, t_1, \ldots t_{m-1}$ consisting of the characters in s, in order, interspersed with $m - n$ occurrences of ε such that $((t_0, q_0), p_1) \in \Delta$, $((t_1, p_1), p_2) \in \Delta$, $\ldots ((t_{m-1}, p_{m-1}), p_m) \in \Delta$.

An NFA M **accepts** a language $L \subseteq \Sigma*$ if $\forall s \ \in \Sigma*$, $s \in L$ iff $\exists\ p_n \in F$ such that $(s, p_n) \in \beta$.

For NFA M, let $L(M)$ be the language of strings that are accepted by M.

Given an alphabet Σ, a **regular expression** is one of the following:

- $\emptyset$, which denotes the set $\emptyset$, that is, the set/language of no strings.
- ε, which denotes the set $\{\varepsilon\}$, that is, the set/language containing only the empty string.
- $\mathtt{a}$, where $\mathtt{a} \in \Sigma$, which denotes the set $\{\mathtt{a}\}$, that is, the set/language containing only the string with only the symbol $\mathtt{a}$.
- $r|s$, where r and s are regular expressions, which denotes the set/language $r \cup s$ (remember that r and s each denote a set). For example, $\mathtt{a}|\varepsilon$ represents the language $\{\mathtt{a}, \varepsilon\}$, and $\mathtt{a}|\mathtt{b}|\mathtt{c}$ represents the language $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$.
- rs, where r and s are regular expressions, which denotes the set/language of strings each composed of a string from r concatenated with a string from s; formally, $\{RS \mid R \in r \text{ and } S \in s\}$. For example, $(\mathtt{a}|\mathtt{b}|\varepsilon)\mathtt{c}(\mathtt{d}|\mathtt{e})$ represents the language $\{\mathtt{acd}, \mathtt{ace}, \mathtt{bcd}, \mathtt{bce}, \mathtt{cd}, \mathtt{ce}\}$.
- $r*$, where r is a regular expression, denoting the set of strings which are composed of the concatenation of zero or more strings, each from r; formally, $\{R_0 R_1 \ldots R_n \mid n \in \mathbb{W} \text{ and } \forall\ i, 0 \leq i \leq n, R_i \in r\}$. For example, $\mathtt{a}(\mathtt{bc}) * \mathtt{d}$ represents the language $\{\mathtt{ad}, \mathtt{abcd}, \mathtt{abcbcd}, \mathtt{abcbcbcd}, \ldots\}$.

A language is **regular** if there exists a regular expression that generates it.

Big result (informal statement): The set of languages for which there exist DFAs that accepts them is the same as the set of languages for which there exist NFAs that accept them. Morover, this is the set of regular languages.

Lemma 1

For any DFA M_1, there exists an NFA M_2 such that $L(M_1) = L(M_2)$.

Lemma 2

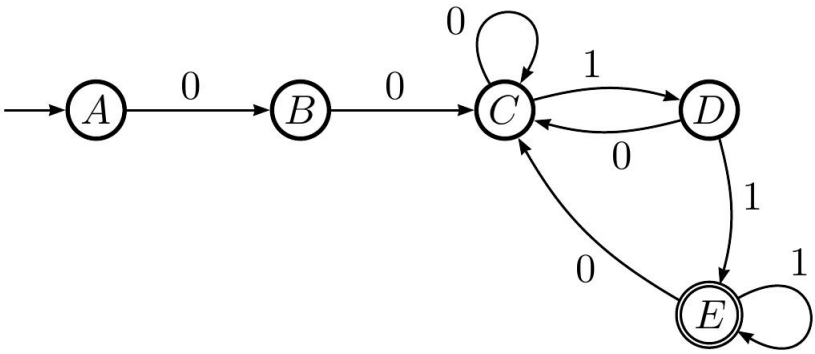
For any NFA M_1, there exists a DFA M_2 such that $L(M_1) = L(M_2)$.

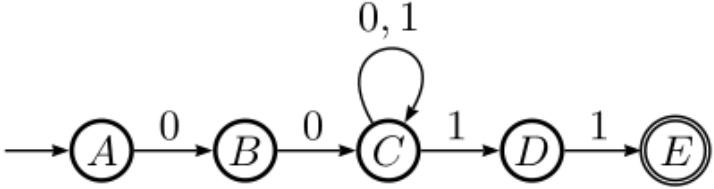
Lemma 3

For any regular expression r, there exists an NFA M such that $L(M)$ is the language generated by r.

Lemma 4

For any NFA M, there exists a regular expression that generates $L(M)$.


A
B
C
D
E
0
0
0
1
0
1
0
1


A
B
C
D
E
0
0
0, 1
1
1

Proof (of Lemma 2). *Suppose $M_1 = (Q, \Sigma, \Delta, q_0, F)$ is an NFA, where Q is a set of states, Σ is an alphabet, Δ is a transition relation from $\Sigma \times Q$ to Q, q_0 is the start state, and F is the set of final states.*

Let $M_2 = (Q_2, \Sigma, \delta_2, \{q_0\}, \{Z \in Q_2 \mid Z \cap F \neq \emptyset\})$ where Q_2 and δ_2 are calculated using this algorithm:

- *$\{q_0\}$, the set containing only the start state of the NFA, is a state in M_2, and let it be the start state of the DFA.*
- *For any set X of NFA states already found to be a state in the DFA (that is, for any $X \in \mathscr{P}(Q)$ such that we know $X \in Q_2$) and for any character a in the alphabet, the set Y of states that any state in X transitions to on a in the NFA is a state in the DFA, and X transitions to it. That is,*

$$\begin{aligned} Y &= \{B \in Q \mid \exists\ A \in X \text{ such that } ((a, A), B) \in \Delta\} \\ Y &\in Q_2 \\ ((X, a), Y) &\in \Delta_2 \end{aligned}$$

- *If X is a state in the DFA and contains an accept state of the NFA, then X is an accept state of the DFA. That is,*

$$F_2 = \{Z \in Q_2 \mid Z \cap F \neq \emptyset\}$$

We claim M_2 is a DFA equivalent to M_1.

That the algorithm terminates: *Each element of $\mathscr{P}(Q)$ could be processed at most once, and $\mathscr{P}(Q)$ is finite. Every time we add something to the worklist, we also add something to the list that becomes Q_2. Either the worklist will become empty (in which case the algorithm terminates) or, in the worst case, Q_2 becomes $\mathscr{P}(Q)$. If the worklist indeed became $\mathscr{P}(Q)$, then from then on we would remove from it without adding, and so eventually the worklist will become empty and terminate.*

That M_2 is deterministic: *The way we have written the algorithm ensures that no DFA state (element of $\mathscr{P}(Q)$) will have more than one transition for a single alphabet symbol. Thus δ_2 will have at most one entry for any $(X \in \mathscr{P}(Q), s \in \Sigma)$ pair.*

That M_2 is equivalent to M_1: *Suppose the string $s_1, s_2, \ldots s_n$ is accepted by M_1. Then M_1 has states $q_0, q_1, \ldots q_n$ such that for all i, $0 < i \leq n$, $((q_{i-1}, s_i), q_i) \in \Delta$.*

Our proof that M_2 accepts this string will be by structural induction on the length of the string. Restating our claim: If $s_1, s_2, \ldots s_n$ can put M_1 into a state q_n, then it puts M_2 into a state X_n such that $q_n \in X_n$.

Suppose $n = 1$. Then $((q_0, s), q1) \in \Delta$ for some q_1, and in our algorithm, $q_1 \in \delta_1(\{q_0\}, s_1)$.

Next, suppose this holds for some $n \geq 1$, that is $s_1, s_2, \ldots s_n$ can put M_1 into state q_n and also puts M_2 into state X_n such that $q_n \in X_n$, and suppose $((q_n, s_{n+1}), q_{n+1}) \in \Delta$ for some q_n. By our algorithm, $q_{n+1} \in \delta_1(X_n, s_{n+1})$.

Then we can apply this to the case where the last NFA state q_n is an accept state.

Conversely, if M_2 accepts a string, a similar process can construct a path in M_1 that accepts the string. $\square$

Proof outline (of Lemma 4). *Suppose $M = (Q, \Sigma, \Delta, q_1, F)$ is an NFA, and let $Q = \{q_1, q_2, \ldots q_n\}$ (this merely gives names to all the states of M; notice we name the start state q_1 this time, not q_0).*

In this proof, we use the following definition:

Let $R(i, j, k)$ be the set of strings which, if M is in state q_i, can bring M into q_j passing through only states $\{q_1, q_2, \ldots q_k\}$ in between. In other words, suppose the front part of a certain string fed into M can get M into state q_i. What substrings can come next that can get M into state q_j without using any state numbered greater than k? (i and j themselves may be greater than k.)

We will prove that a regular expression exists for any such set $R(i, j, k)$ by induction on k.

- Prove that for any i and j where $0 \leq i, j \leq n$, there exists a regular expression defining the set $R(i, j, 0)$. This is our base case, and since there is no state q_0, it means the string must bring M directly from i to j. (Hint: Consider two subcases: $i = j$ and $i \neq j$.)
- Next suppose that $R(i, j, k)$ is regular for some k—that is, for any i and j, there exists a regular expression that defines the set of strings taking M from q_i to q_j without using any state numbered greater than k. Prove, then, that for any i and j, there exists a regular expression defining the set $R(i, j, k + 1)$. (Hint: Consider, among other things, the regular expressions defining $R(i, k + 1, k)$, $R(k + 1, k + 1, k)$, and $R(k + 1, j, k)$. Form a new regular expression from these from the rules of regular expressions.)
- By the two previous exercises, for all accept states $q_a \in F$, there exists a regular expression defining the set of strings $R(1, a, n)$. Show that this implies there exists a regular expression defining the language accepted by M.

For Mon, Apr 27:

Take quiz on Canvas