Theory of Computation Part 1 - Finite Automata

Ethan Graham

22nd of June 2023

Lecture 1 - Deterministic Finite Automata

Definition DFA

A DFA is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$

- Q: The set of states
- Σ: The alphabet
- $\delta Q \times \Sigma \rightarrow Q$: The transition function
- q_0: The initial state
- $F \subseteq Q$: The set of accepting states

Note that δ assigns exactly one next state to every (q, σ) tuple.

The language $A = L(M)$ is the set of all input strings accepted by the DFA M.

Proving Correctness of a DFA

We usually use induction to do this.

- **Base Case:** Show for the empty string ε. For example $\delta(q_0, \varepsilon) = q_0$ which satisfies constraint
- **Inductive case:** Assume the condition holds for x. Show what happens for $x.\sigma$

These proofs can be quite long since we must show what happens for every tuple (q, σ) to prove correctness in all generality.

We can always use strong induction to prove correctness of an automata. Assuming we have a language L and DFA M. We want to prove $L = L(M)$

- Find a precise description of sets T_i that make the automata go to state q_i for all $i = 0, ..., n$
- Prove by induction that for string w, if $\delta(q_0, w) = q_i$, then $w \in T_i$. Do this for all $i = 0, ..., n$

Definition Regular Language

A language A that is accepted by some DFA is regular. i.e. $\exists M \; s.t. \; L(M) = A$

Definition Complement of a Language

$\overline{L} = \{w \in \Sigma^* | w \notin L\}$

Theorem: Let L be a regular language and M and automaton s.t. $L(M) = L$. Then $\overline{L}$ is regular and accepted by $M' = (Q, \Sigma, \delta, q_0, \bar{F} = Q \backslash F)$

Definition: Union of two Languages

$L_1 \cup L_2 = \{w \in \Sigma^* | w \in L_1 \textbf{ or } w \in L_2\}$

Theorem: Union If L_1, L_2 are two languages accepted by M_1, M_2 then their union is reguar and accepted by the following DFA:

$M = (Q, \Sigma, \delta, q_0, F)$ where $Q = Q_1 \times Q_2$, $F = \{(r_1, r_2) | r_1 \in F_1 \textbf{ or } r_2 \in F_2\}$ $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$

We are basically simulating two machines at the same time on the same input string and seeing whether one ends in an accepting state.

Definition: Intersection of two Languages

($\ldots$ same setup as before) $L_1 \cap L_2 = \{w \in \Sigma^* | w \in L_1 \textbf{ and } w \in L_2\}$ Same tuples and all that as for union except for F. Now: $F = \{(r_1, r_2) | r_1 \in F_1 \textbf{ and } r_2 \in F_2\}$

Definition: Concatenation

(blah blah blah same as before) $L_1 \circ L_2 = \{w \in \Sigma^* | w = w_1 \cdot w_2, w_1 \in L_1 \textbf{ and } w_2 \in L_2\}$

The concatenation of two languages is regular, but it is more convenient to prove this with NFA's since we never know when to "switch" languages.

Lecture 2 - Non-Deterministic Finite Automata

Definition NFA

An NFA is a 5-tuple (like for a DFA) consisting of $(Q, \Sigma, \delta, q_0, F)$

- Q: The set of states
- Σ: The alphabet
- $\delta : \; Q \times (\Sigma \bigcup \{\varepsilon\}) \rightarrow \mathbb{P}(Q) = 2^Q$: The transition function
- q_0: The initial state
- $F \subseteq Q$: The set of accepting states

The main differences between this and a DFA is that we can transition to more than one state at a time on a given symbol, or no symbol at all. The codomain of δ allows us to map to any subset, which could also include $\emptyset$. At any point in the path, if we reach $\emptyset$, then this path is "killed" $\rightarrow$ recall the dead cat thing.

To prove the acceptance of an input string, we just need to draw the branch that goes down to the accepting state. **guess and verify**

NFAs are much smaller than DFAs in general.

NFA $\equiv$ DFA

Theorem: Every NFA has an equivalent DFA. Proof in course slides/Favre notes. **Proof Idea:** The state of the DFA represents the set of states reachable by one of the threads in the NFA. The accepting states $\tilde{F}$ become the sets of states containing an accepting state (in the NFA) $\tilde{F} = \{A \in 2^Q | A \cap F \neq \emptyset$

$\tilde{\delta}(A, a) = \bigcup_{q \in A} \delta(q, a)$ The transition function maps from a set of states into the set of all possible states the threads could reach in one step (from the previous states).

We note that M's state after $x \in \Sigma^*$ is $\tilde{\delta}(\{q_0\}, x)$. We get that $\tilde{\delta}(\{q_0\}, \varepsilon) = \{q_0\}$

Regular $\equiv$ Accepted by NFA

By theorem, a language is regular **iff** it is accepted by some NFA.

Theorem: Concatenation is regular

The idea is to construct an NFA N s.t. every accepting state of M_1 has an ε-transition to the initial state of M_2. This ensures that any $w \in L_1 \circ L_2$ is accepted.

Lecture 3 - Non-regular Languages

A non-regular sometimes has the condition *needs memory depending on the size of the input* but this isn't enough to prove non-regularity.

Pumping Lemma

If A is a regular language, then there is a number p (pumping length) such that for every string $s \in A$ of length at least p, there exists a three-piece division $s = xyz$ such that:

- For any $i \geq 0, xy^iz \in A$
- $|y| \geq 1$
- $|xy| \leq p$

x : Part of the string that brings us to the first state q_i of the loop y : The loop part that goes from q_i to q_i (we can loop any number of times, even 0 times) z : takes us from q_i to an accepting state.

It is better to focus on the proof rather than the lemma itself .

Proof Ideas

- If we have an input string of length at least p, we need to have a loop. $|xy| \leq p$ means that the loop must happen **in the first p characters**. (It must be finished)
- By the pigeonhole principle, since we visit $p + 1$ states for an input of size p

We normally prove non-regularity of a language by showing that it does not satisfy the pumping lemma (since a regular language must satisfy it) The choice of which string to use as counter example is the tricky bit of these proofs.