Notes on Finite Automata

Deterministic Finite Automata

Definition: A *deterministic finite automaton* (DFA) is a 5-tuple (Q, Σ, δ, q_0, F):

1. Q — a *finite* set (the states)
2. Σ — a *finite* set (the alphabet)
3. δ: $Q \times \Sigma \to Q$ — a function from a state and alphabet symbol to a state (the transition function)
4. $q_0 \in Q$ — the start state
5. $F \subseteq Q$ — the set of accept states

Language Recognition: $\mathcal{L}(A)$ represents the language recognized by DFA A.

Informally (which we define more formally below):

$w \in \mathcal{L}(A) \Leftrightarrow$ running A on input w ends in an accept state

More formally, we use the definition below.

Computation Model for DFA

Define δ^* as the *extended transition function*, so δ^*_A is the extended transition function for DFA $A = (Q, \Sigma, \delta, q_0, F)$.

For any input, $w \in \Sigma^*$, $w \in \mathcal{L}(A) \iff \delta^*_A(q_0, w) \in F$.

The extended transition function δ^* is defined recursively on the input string (its second input, w), $\forall q \in Q$:

Basis: $w = \epsilon$ (empty string)

$$\delta^*(q, \epsilon) = q$$

Inductive Case: $w = ax$, $a \in \Sigma, x \in \Sigma^*$

$$\delta^*(q, ax) = \delta^*(\delta(q, a), x)$$

Finally, we define regular languages as follows.

Definition: A *regular language* is a language that can be recognized by some deterministic finite automaton.

Note that in class we defined regular languages based on regular expressions, but we also mentioned that regular expressions can describe a language if and only if that language can be recognized by a DFA. So, we can use either of the models to define regular languages.

Defining the NFA Computation Model

Definition: A *nondeterministic finite automaton* (NFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where Q, Σ, q_0, and F are defined as they are for a DFA, but the transition function δ can move into *potentially more than one, or even zero* states as the next possible state. More formally, it is defined as:

> δ**:** $Q \times \Sigma \rightarrow pow(Q)$ — which is a (total) function from a state and alphabet symbol to **a set of states** that is a member of $pow(Q)$, the power set of Q.

Computation Model for an NFA
The NFA $A = (Q, \Sigma, \delta, q_0, F)$ accepts the language $\mathcal{L}(A) = \{w \mid \delta^*(q_0, w) \cap F \neq \emptyset\}$ where the *extended transition function* δ^*: $Q \times \Sigma^* \rightarrow pow(Q)$ is defined by:

> Basis: $w = \epsilon$:
>
> $\delta^*(q, \epsilon) = \{q\}$
>
> Inductive case: $w = ax$, $a \in \Sigma$, $x \in \Sigma^*$
>
> $\delta^*(q, ax) = \bigcup_{q_i \in \delta(q,a)} \delta^*(q_i, x)$

Equivalence of NFAs and DFAs

Show that the set of languages that can be recognized by some NFA is equal to the set of languages that can be recognized by some DFA.

1. Every DFA has an equivalent NFA ($\mathcal{L}(NFA) \subseteq \mathcal{L}(DFA)$). (Proof by construction — trivial enough to be left as exercise.)
2. Every NFA has an equivalent DFA ($\mathcal{L}(DFA) \subseteq \mathcal{L}(NFA)$). (Proof by construction below and done less formally in class.)

Given $N = (Q, \Sigma, \delta, q_0, F)$, an NFA recognizing some language A. We prove that every NFA has an equivalent DFA by showing how to construct a DFA N' from N that recognizes the same language A.

$N' = (Q', \Sigma', \delta', q_0', F')$ defined as:

1. $Q' = pow(Q)$ — we have a state in Q' to represent each possible subset of states in Q. The label for each state in Q' is a set.
2. $\Sigma' = \Sigma$ — the alphabet is the same
3. $\delta' : Q' \times \Sigma' \rightarrow Q'$ is defined to capture all possible states resulting from δ transitioning from the input state:

$$\delta'(R, a) = \bigcup_{r \in R} \delta(r, a)$$

   where R is a state in Q', but interpreted as a set of the elements in its label.

4. $q_0' = \{q_0\}$
5. $F' = \{$states in Q' that correspond to subsets of states that include a state in $F\}$

