CHAPTER 1 Regular Languages

Outline

- Finite Automata (FA or DFA)
 - definitions, examples, designing, regular operations
- Non-deterministic Finite Automata (NFA)

definitions, equivalence of NFAs and DFAs, closure under regular operations

- Regular expressions
 - definitions, equivalence with finite automata
- Non-regular Languages
 - the pumping lemma for regular languages

Non-determinism

• So far in our discussion, every step of a computations follows in a unique way from the preceding step.

• When the machine is in a given state and reads the next input symbol, we know what the next state will be – it is determined. We call this **deterministic** computation.

• In a **non-deterministic** machine, several choices may exist for the next state at any point.



• Non-determinism is a generalization of determinism, so every DFA is automatically a NFA.

- The difference: every state of a DFA has exactly one exiting arrow for each symbol
 - in a NFA a state may have 0, 1, or many exiting arrows for each symbol
 - a NFA may have arrows with the label ε

How does an NFA compute?

- After reading the symbol, the machine splits into multiple copies of itself and follows all the possibilities in parallel.
- Each copy takes one of the possible ways to proceed and continues as before.
- If there are subsequent choices, the machine splits again.
- If the next input symbol doesn't appear on any of the arrows exiting the state occupied by a copy of the machine, that copy dies.
- If any one of these copies of the machine is in an accept state at the end of the input, the NFA accepts the input string.

• If a state with an \mathcal{E} symbol on an exiting arrow is encountered, the machine (w/o reading any input) splits into multiple copies, one following each of the exiting arrow with \mathcal{E} and one staying at current state.



Non-determinism may be viewed as a kind of parallel computation wherein several "processes" can be running concurrently.

A non-deterministic FA (**NFA**) N_1 (Run for inputs 11, 101)

Tree of Possibilities

• A way to think of a non-deterministic computation is as a **tree of possibilities.**

- The root corresponds to the start of the computation
- Every branching point in the tree corresponds to a point in the computation at which the machine has multiple choices



NFA vs. DFA

- NFAs are useful in several aspects.
 - Every NFA can be converted into an equivalent DFA (construction later).
 - Constructing NFAs is sometimes easier than directly constructing DFAs.
 - An NFA may be much smaller than its deterministic counterpart.
 - Its functioning may be easier to understand.
 - We will use non-determinism in more powerful computational models.

• Example.



They recognize the same language A={all strings over {0,1} containing a 1 in the third position from the end}

Formal Definition of NFAs

- A non-deterministic finite automaton (NFA) is specified by a 5tuple $(Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states, Σ is a finite alphabet, Is a collection of all subsets of Q $\delta: Q \times \Sigma_{\varepsilon} \rightarrow \mathbf{P}(Q)$ is the transition function, $q_0 \in Q$ is the initial state, $\Sigma_{\varepsilon} = \Sigma \cup \{\varepsilon\}$ is the set of final states.
- For NFA N_1 we have

 $Q = \{q1, q2, q3, q4\}$ $\Sigma = \{0, 1\}$ $q_0 = q1$ $F = \{q4\}$



Acceptance of Strings and the Language of NFA

• Let $N = (Q, \Sigma, \delta, q_0, F)$ be a NFA

• *N* accepts *w* if we can write *w* as $w = w_1, w_2, ..., w_n$, where each w_i is a member of Σ_{ε} and a sequence of states $r_0, r_1, r_2, ..., r_n$ exists in *Q* with the following three conditions:

1.
$$r_0 = q_0$$
,
2. $r_{i+1} \in \delta(r_i, w_{i+1})$ for $i = 0, ..., n-1$, and
3. $r_n \in F$

• If L is a set of strings that N accepts, we say that L is the *language* of N and write L=L(N).

• We say N recognizes L or N accepts L.

 In this example, N₄ recognizes the strings *a, baba, baa,* but doesn't accept the strings *b, bb, babba.*



Subset Construction

- For every NFA there is an *equivalent* (accepts the same language) DFA.
- But the DFA can have exponentially many states.
- Let $N = (Q_N, \Sigma, \delta_N, q_0, F_N)$ be an NFA.
- The equivalent DFA constructed by the *subset construction* is

$$D = (Q_D, \Sigma, \delta_D, q_{0D}, F_D).$$

- For $R \subseteq Q_N$, we define $E(R) = \{ q: q \text{ can be reached from } R \text{ by traveling along } 0 \text{ or more } \mathcal{E} \text{ arrows} \}.$
- Then,

1. $Q_D = P(Q_N)$, (= the set of subsets of Q_N),

- 2. For $R \in Q_D$ and $a \in \Sigma$ let $\delta_D(R, a) = E(\bigcup_{r \in R} \delta_N(r, a))$,
- 3. $q_{0D} = E(\{q_0\}),$
- 4. $F_D = \{R \in Q_D : R \text{ contains an accept state of } N\}.$

Example And The Theorem



Theorem. Every NFA has an equivalent DFA.

Corollary. A language is regular if and only if some NFA recognizes it.

We have seen that for any NFA there exists an equivalent DFA. Hence

A language is regular if and only if some NFA recognizes it.

We will show today that regular languages are closed under regular operations.

Regular Operations (again)

- Let L1 and L2 be languages. We defined the regular operations *union, concatenation, and star* as follows.
 - Union: $L1 \cup L2 = \{w : w \in L1 \text{ or } w \in L2\}.$
 - Concatenation: $L1 \circ L2 = \{wv : w \in L1 \text{ and } v \in L2\}$.
 - Star: $L1^* = \{w_1 w_2 ... w_k : k \ge 0 \text{ and each } w_i \in L1\}.$
- Example: Let the alphabet Σ be the standard 26 letters {a,b,...,z}.
 - If L1={good, bad} and L2= {boy, girl}, then

 $L1 \cup L2 = \{\text{good, bad, boy, girl}\}.$

 $L_1 \circ L_2 = \{\text{goodboy, badboy, goodgirl, badgirl}\}.$

 $L1^* = \begin{cases} \mathcal{E}, \text{ good, bad, goodgood, badgood, badbad, goodbad, goodgoodgood, goodgoodbad, goodbadbad, ...} \end{cases}$

Th.1 The class of regular languages is closed under the union operation.

- We have regular languages L1 and L2 and want to prove that $L1 \cup L2$ is regular.
- The idea is to take two NFAs *N1* and *N2* for *L1* and *L2*, and combine them into one new NFA *N*.
- N must accept its input if either N1 or N2 accepts this input
- N will have a new state that branches to the start states of the old machines N1, N2 with ε arrows
- In this way *N* non-deterministically guesses which of the two machines accepts the input
- If one of them accepts the input then N will accept it, too



Th.2 The class of regular languages is closed under the concatenation operation.

- We have regular languages L1 and L2 and want to prove that L1 L2 is regular.
- The idea is to take two NFAs N1 and N2 for L1 and L2, and combine them into a new NFA N.
- *N* accepts when the input can be split into two parts, the first accepted by *N1* and the second by *N2*
- We can think of *N* as non-deterministically guessing where to make the split



Th.3 <u>The class of regular languages is closed under the</u> star operation.

- We have regular language L1 and want to prove that L1 * is regular.
- We take an NFA N1 for L1, and modify it to recognize L1*.
- The resulting NFA *N* accepts its input if it can be broken into several pieces and *N1* accepts each piece.
- *N* is like *N1* with additional ε arrows returning to the start state from the accept state.
- In addition we must modify N so that it accepts ε , which always is a member of $L1^*$.

$$Q = \{q_0\} \cup Q_1$$

$$F = \{q_0\} \cup F_1$$

$$F = \{q_0\} \cup F_1$$

$$P = \{q_0\} \cup F_1$$

$$Q_0 \text{ is the start state}$$

$$N = (Q, \Sigma, \delta, q_0, F) \text{ recognizes } LI^*$$

$$Q_0 \text{ is the start state}$$

$$M = (Q, \Sigma, \delta, q_0, F) \text{ recognizes } LI^*$$

$$Q_0 \text{ is the start state}$$

$$M = (Q, \Sigma, \delta, q_0, F) \text{ recognizes } LI^*$$

$$Q_0 \text{ is the start state}$$

$$M = (Q, \Sigma, \delta, q_0, F) \text{ recognizes } LI^*$$

$$Q_0 \text{ is the start state}$$

$$M = (Q, \Sigma, \delta, q_0, F) \text{ recognizes } LI^*$$