Lecture 2:
Turing machines and Finite Automata

Valentine Kabanets

September 16, 2016

1 Finite Automaton

Before exploring Turing machines, we consider a very special case: Finite Automata. Basically, it is a Turing machine with the following restrictions:

- the tape is *read-only*, and
- the tape head is always moving to the right.

These restrictions mean that we don't need a separate work-tape alphabet Γ, and that our δ function needn't specify the new symbol to write over (as no writing takes place) or a movement direction (as it's always to the right).

More formally, a *finite automaton (FA)* is $M = (Q, \Sigma, \delta, q_0, F)$, where Q, Σ, q_0 are as above, $F \subseteq Q$ is the set of final (accepting) states, and the transition function δ is of the type $\delta : Q \times \Sigma \to Q$.

The FA doesn't write anything. At each step, it reads the current symbol of the input string, enters a new state (according to δ), and moves its tape head one position to the right. It halts upon reaching the end of the input string (thus, the FA is a linear-time algorithm).

We say that a FA $M = (Q, \Sigma, \delta, q_0, F)$ *accepts* an input string $w = a_1 \dots a_n$ if there is a sequence of states $q_0, q_1, q_2, \dots, q_n$ such that

- $q_{i+1} = \delta(q_i, a_{i+1})$ for $0 \leq i < n$, and
- $q_n \in F$

The *language accepted* by M is $\{w \in \Sigma^* \mid M \text{ accepts } w\}$. We usually denote by $L(M)$ the language accepted by the FA M.

We call a language L *regular* if some FA accepts L.

2 Trivial automata

Here are examples of trivial automata. The first one accepts everything; the second accepts nothing.

1. An automaton T over the alphabet Σ with a single state q_0, which is both initial stata and an accepting state, and a transition $\delta(q_0, a) = q_0$ for every $a \in \Sigma$, will accept every string in Σ^*. That is, $L(T) = \Sigma^*$.
2. An automaton which has no accepting states will not accept any string, and so its language is $\emptyset$.

3 Automaton design

In general, we want to solve the following *design* problem: Given a description of a language L (over some alphabet Σ), design a FA A such that $L(A) = L$. That is, design an automaton that accepts exactly the strings in L and nothing else.

This is similar to Algorithm Design: Given a problem description, write a computer algorithm that solves the problem on all instances. In our case, the "problem" is a decision problem (language), and "algorithm" is a Finite Automaton.

We saw in class that the language

$$Parity = \{w \in \{0,1\}^* \mid w \text{ has an odd number of 1s}\}$$

can be decided (accepted) by a FA. Thus, Parity is a regular language. There are many other regular languages. But, as we'll see soon, there are also many *non-regular* language, i.e., those which no FA can accept.[1]

4 Regular operations

We consider the following operations on languages, called *regular* operations. Let A and B be any languages (i.e., sets of finite-length strings). We define

- Union: $A \cup B = \{w \mid w \in A \text{ or } w \in B\}$,
- Concatenation: $A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$, and
- Star: $A^* = \cup_{k \geq 0} A^k$, where $A^0 = \{\epsilon\}$, and for $k > 0$, A^k is the concatenation of k copies of A. Equivalently, $A^* = \{x_1 \dots x_k \mid k \geq 0, x_i \in A \text{ for each } 1 \leq i \leq k\}$.

We will show the following.

Theorem 1. *The class of regular languages is closed under the regular operations. That is, if languages A and B are any regular languages, then $A \cup B$, $A \circ B$, and A^* (and B^*) are also regular languages.*

To prove this, we need to argue that if A is accepted by some FA $M_A = (Q_A, \Sigma, \delta_A, q_A, F_A)$, and B is accepted by some FA $M_B = (Q_b, \Sigma, \delta_B, q_B, F_B)$, then there are FA for each of the languages $A \cup B$, $A \circ B$, and A^*. We start with the union. The other two operations will have to wait until we introduce *nondeterministic* automata below.

Proof of Theorem 1 for the case of Union. The idea for the automaton accepting $A \cup B$ is simple: Run the two automata M_A and M_B in parallel on the same input string, keeping track of their states. If either of M_A or M_B enters a final state at the end, we accept.

More formally, define the FA $M = (Q, \Sigma, \delta, q_0, F)$, where

- $Q = Q_A \times Q_B$,
- $\delta((q,p),a) = (\delta_A(q,a), \delta_B(p,a))$,

[1] Remember that for a FA A to *accept* a given language L, the FA A must accept every string in L and reject (not accept) every string not in L.

- $q_0 = (q_A, q_B)$, and
- $F = (F_A \times Q_B) \cup (Q_A \times F_B)$.

It is easy to see that the defined automaton M accepts an input string iff at least one of the automata M_A and M_B accepts it. □

5 Nondeterministic Finite Automaton (NFA)

Informally, an NFA has more than one option for the next transition. For example, in state q upon reading 0, it may enter state p_1 or state p_2. We represent this by writing $\delta(q, 0) = \{p_1, p_2\}$. We also allow ϵ-transitions where the automaton doesn't read the current symbol (its tape head stays put) but just changes its state.

More formally, an NFA $N = (Q, \Sigma, \delta, q_0, F)$, where Q, Σ, q_0, F are as for DFA (deterministic finite automata considered before), and

$$\delta : Q \times \Sigma_\epsilon \to \mathcal{P}(Q),$$

where $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$ and $\mathcal{P}(Q)$ is the collection of all subsets of Q (i.e., the *power set* of Q).

We say that the NFA N *accepts* a string $w \in \Sigma^*$ if

- it is possible to write $w = y_1 y_1 \dots y_m$, with each $y_i \in \Sigma_\epsilon$, and
- there is a sequence of states $q_0, q_1, \dots, q_m$ such that
  - each $q_{i+1} \in \delta(q_i, y_{i+1})$, and
  - $q_m \in F$.

The *language accepted* by an NFA N is the set of all strings w accepted by N.

6 Closure under regular operations

We will show later that the collection of languages accepted by NFA is exactly the same as the class of regular languages. For now, we will show that the NFA languages are closed under the regular operations.

Let N_A and N_B be two NFA accepting languages A and B, respectively.

- Union: Define the NFA N by adding a new initial state with ϵ-transitions to the initial states of N_A and N_B. Then N accepts $A \cup B$.
- Concatenation: Define the NFA N by adding an ϵ-transition from each final state of N_A to the initial state of N_B. Make the final states of N_A non-final states of N. Make the initial state of N_A the initial state of N. Make the final states of N_B the final states of N. Clearly, N accepts $A \circ B$.
- Star: Define the NFA N by adding a new initial state q_0'. Make this also a final state of N. Add ϵ-transitions from q_0' to the initial state of N_A. Add ϵ-transitions from each final state of N_A to the initial state of N_A. The NFA N accepts A^*.