Lecture 11: Algorithms on DFAs

23 February 2010

In this lecture, we will discuss several algorithms for regular languages presented as DFAs.

1 Emptiness

The emptiness problem for DFAs is decidable; i.e. we can write an algorithm to check whether the language of an input DFA A is non-empty. A DFA can be encoded as a string, say by encoding the tuple $(Q, \Sigma, \delta, q_0, F)$. The tuple can be appropriately encoded as it is finite (Q and Σ are finite, and hence $\delta : Q \times \Sigma \to Q$ is representable as a finite table, etc.

Now, consider the DFA as a graph with nodes as states, and (labeled) edges as transitions of the DFA. Then it is easy to see that $L(A)$ is non-empty iff there is a path from the initial state to some final state, in this graph. This problem is simply a reachability problem ($s-t$ reachability) and can be solved in linear time (in the number of nodes and edges), say using depth-first search. Hence the emptiness problem is decidable for DFAs.

A similar algorithm works for NFAs as well.

2 Inclusion, Equivalence, etc.

It now follows that inclusion and equivalence of regular languages, represented as DFAs, is decidable, using closure properties.

Given DFAs A_1 and A_2 over the same alphabet, we can decide if $L(A_1) \subseteq L(A_2)$ by noticing that this is true iff $L(A_1) \cap L(\bar{A_2}) = \emptyset$. Since DFAs are constructively closed under negation and intersection, we can complement A_2, and intersect the resulting automaton with A_1, and check the resulting automaton for emptiness, which is decidable.

Given DFAs A_1 and A_2 over the same alphabet, we can decide if $L(A_1) = L(A_2)$ by checking if $L(A_1) \subseteq L(A_2)$ and $L(A_2) \subseteq L(A_1)$. Hence equivalence is also decidable.

3 Infiniteness

Given a DFA A, can we decide if $L(A)$ is infinite? It turns out that we can. Let us first prove the following.

Theorem 3.1 *Let A be a DFA with n states. Then $L(A)$ is infinite iff A accepts some word of length l, where $n \leq l < 2n$.*

The proof of the above is simple. First, if $L(A)$ is infinite, then for any i, there is some word w in $L(A)$ whose length is at least i. Take a word w in $L(A)$ such that $|w| \geq n$, and let w be a shortest length word of this kind. Then we claim that $|w| < 2n$. Assume not. Then, by an argument similar to the pumping lemma, $w = uvx$ where $|uv| \leq n$ and $|v| > 0$, and we can "pump down" to get a word $w = ux$ in L. But since $|ux| > n$ and $|uv| < |w|$, this contradicts the fact that w was the shortest word longer than n in L. This proves that $w \in L$ and $n \leq |w| < 2n$. Now, turning to the other direction, assume that there is a word $w \in L$ with $n \leq |w| < 2n$. Then, by the pumping lemma, $w = uvx$, with $|v| > 0$ such that for every $i \in \mathbb{N}$, $uv^i w \in L$. Hence L is infinite.

We can now decide whether $L(A)$ is infinite simply by enumerating all words w of length l, where $n \leq l < 2n$, and checking whether A accepts any such w. If it does, then $L(A)$ is infinite, and otherwise, it is not.

4 Minimization algorithm

The Myhill-Nerode theorem gives us a way to minimize DFAs. Given a DFA A, we can simply compute the language L_q accepted by every state q (i.e. the language accepted when q is made the initial state), and we can check which of these languages are equal (using the language equivalence decidability result above), and merge all pairs of states q and q' that have the same language (i.e. if $L_q = L_{q'}$).

However, we now show a more efficient partition refinement algorithm that works in $O(nlogn)$ time.

A *partition* P of states is a set of subsets of Q, $P = \{S_1, \ldots S_k\}$ where each $S_i \subseteq Q$, $\bigcup_{i=1}^{k} S_i = Q$, and for any $i \neq j$, $S_i \cap S_j = \emptyset$. In other words, the sets in P divide Q into disjoint subsets.

A partition $R = \{T_1, \ldots T_m$ is said to refine $P = \{S_1, \ldots S_k\}$ if it is true that for every $q, q' \in Q$, if q and q' are in the same set in the partition R, then they are also in the same set in partition P. In other words, R is obtained by refining the partition P by dividing the sets in P into smaller sets.

We now describe an algorithm where we start with the initial partition $P_0 = \{F, Q \setminus F\}$ and keep refining it till we reach a partition P that can no longer be refined. This final partition P will then tell us which states can be merged to form a minimal DFA. Intuitively, all states that belong to the same set in P will be merged to a single state, and hence the minimal DFA will have only as many states as the number of sets in P.

Assume that we have a partition P. We define the *next* partition P' the unique partition that satisfies the following properties: Let $q, q' \in Q$.

- q and q' are in the same partition in P' iff they are in the same partition in P and for every $a \in \Sigma$, $\delta(q, a)$ and $\delta(q', a)$ are also in the same partition in P.

We keep refining the initial partition P using the above algorithm, till the refinement stabilizes, and does not give a new partition. Using this stabilized partition, we will build our minimal DFA.

Let us describe this on an example. Consider the DFA of Figure 2. It is more complex than it needs to be.

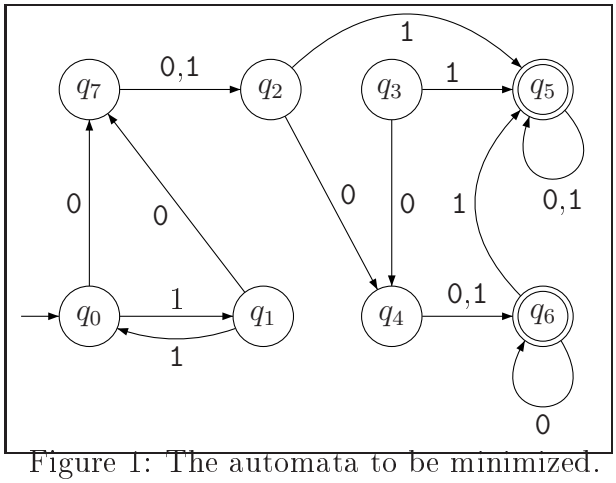

Figure 1: The automata to be minimized.

We start with the partition that groups together the non-final states together and the final states together. Intuitively, we know that a final state and a non-final state definitely have different suffix languages, and hence cannot be merged.

So $P_0 = \{\{q_0, q_1, q_2, q_3, q_4, q_7\}, \{q_5, q_6\}\}$.

Let us name the sets in P_0 as say $A = \{q_0, q_1, q_2, q_3, q_4, q_7\}$ and $B = \{q_5, q_6\}$.

Now, consider the state q_0. On reading 0 it goes to q_7 (which is in A) and on reading 1 it goes to q_1 (which is also in A). However, q_3, when reading 0 goes to q_4 (which is in A) and on reading 1 goes to q_5 (which is in B). Hence we must separate q_0 and q_3 in the next refinement. Also, states q_0 and q_5, since they are already separated in P_0, will continue to be separated in the next refinement.

Let us compute the sets where the states in A go to. q_0 goes to (A, A), q_1 goes to (A, A), q_2 goes to (A, B), q_3 goes to (A, B), q_4 goes to (B, B), and q_7 goes to (A, A). Hence we refine the set A into three sets: $\{q_0, q_1, q_7\}$, $\{q_2, q_3\}$ and $\{q_4\}$. Let us compute the sets where the states in B go to. q_5 goes to (B, B) and q_6 goes to (B, B). Hence we keep q_5 and q_6 in the same partition.

We hence get a new partition of states $P_1 = \{\{q_0, q_1, q_7\}, \{q_2, q_3\}, \{q_4\}, \{q_5, q_6\}\}$ which is a refinement of P_0.

Now we compute the next iteration of refinement. Let us name the sets in P_1. Let $C = \{q_0, q_1, q_7\}$, $D = \{q_2, q_3\}$, $E = \{q_4\}$, and $B = \{q_5, q_6\}$.

First let us examine the states in C. q_0 goes to (C, C), q_1 goes to (C, C), and q_7 goes to (D, D). Hence we refine C into two sets: $\{q_0, q_1\}$ and $\{q_7\}$.

Now let us examine the states in D. q_2 goes to (E, B) and q_3 goes to (E, B). So q_2 and q_3 remain in the same partition.

Examining the states in B, also gives that q_5 and q_6 remain in the same partition.

So we can now form the new partition $P_2 = \{\{q_0, q_1\}, \{q_7\}, \{q_2, q_3\}, \{q_4\}, \{q_5, q_6\}\}$ which is a refinement of P_1.

We can now continue to refine the partition P_2. However, doing so gives the same partition as no other splitting of states happen. We are hence done and we can build the minimal DFA. The idea is to have a single state for each set in the partition of P_2. And have a transition on a letter d from one set S to another set S' provided there is some state $q \in S$ such that $\delta(q, d) \in S'$. Note that if there is some state $q \in S$ such that $\delta(q, d) \in S'$, then for

every state $q' \in S$, $\delta(q', d) \in S'$, as we know otherwise that the partition would have been refined further. We also mark S to be a final state if some state in S (equivalently, all states in S) are final.

Doing so gives the minimal automaton depicted below.

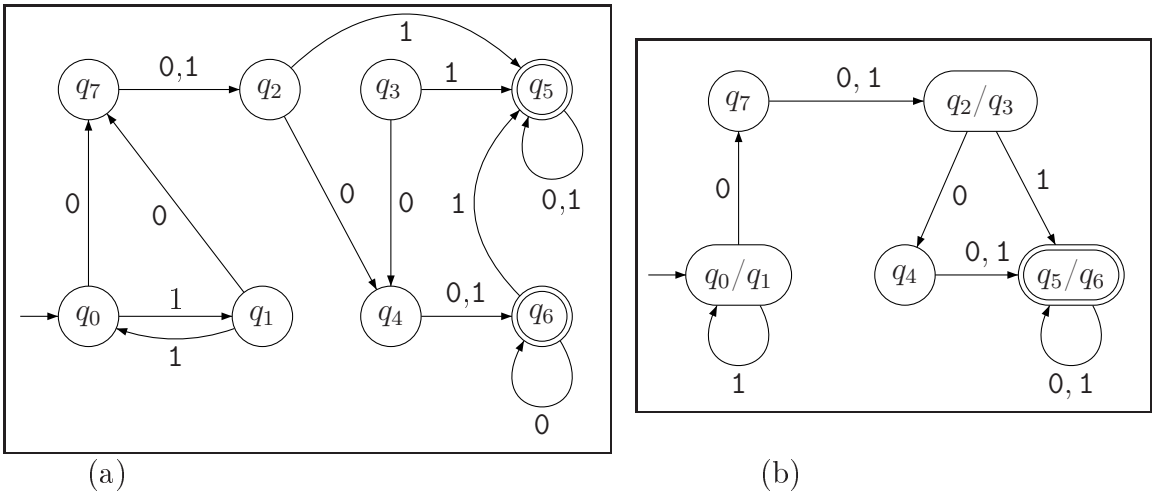

Figure 2: (a) Original automata, (b) minimized autoamta.

.

We omit the correctness argument for minimization. Intuitively, the algorithm ensures that states in different partitions at any time have distinct suffix languages (this is true in the beginning, and is maintained across each refinement). Hence the number of states in the final DFA certainly has no more states than a minimal DFA would have. Furthermore, we can prove that in the final partition, states in the same set have the same suffix languages, and hence indeed can be merged, giving a DFA that accepts the same language as the original DFA. Hence the constructed DFA must be a minimal DFA.