A Partition-Based Minimization Technique for Finite Automata

Rance Cleaveland

Spring 2000

1. Preliminaries

We have seen that it is possible to define a minimum-state automaton for recognizing a given regular language L. The construction of the automaton is somewhat abstract, relying as it does on the *indistinguishability* relation $\overset{L}{\bowtie}$ for the language L. This handout shows how to *construct* the minimum-state automaton for recognizing L, given some finite automaton that accepts L. The book also presents such a procedure; the one described here proves to be more efficient, however.

The algorithm presented in this handout essentially works by determining which states in the automaton can be "merged" into one state. In order to understand how it works, we first introduce the following definitions.

Definition 1.1. Let $M = \langle Q, \Sigma, \delta, q_0, A \rangle$ be a finite automaton, and let $q \in Q$. Then M_q is defined to be the finite automaton $\langle Q, \Sigma, \delta, q, A \rangle$.

In other words, M_q is just M with the start state changed to q. We will sometimes refer to $\mathcal{L}(M_q)$ as the *language of* q. We now define an equivalence relation on states in a finite automaton that reflects a notion of *language equivalence* between states.

Definition 1.2. Let $M = \langle Q, \Sigma, \delta, q_0, A \rangle$ be a finite automaton, and let $p, q \in Q$. Then $p \overset{M}{\sim} q$ is defined to hold if $\mathcal{L}(M_p) = \mathcal{L}(M_q)$.

So $p \overset{M}{\sim} q$ is true if p and q have the same languages. Moreover, although it may not be immediately apparent, there is a close connection between $\overset{M}{\sim}$ and $\overset{\mathcal{L}(M)}{\bowtie}$, the indistinguishability relation on the language of M.

Lemma 1.3. *Let $M = \langle Q, \Sigma, \delta, q_0, A \rangle$ be a finite automaton. Then for any $x, y \in \Sigma^*$, $\delta^*(q_0, x) \overset{M}{\sim} \delta^*(q_0, y)$ if and only if $x \overset{\mathcal{L}(M)}{\bowtie} y$.*

Based on this lemma, it is possible to describe the minimum-state finite automaton recognizing the language $\mathcal{L}(M)$ of a given finite automaton M in terms of $\overset{M}{\sim}$ rather than $\overset{\mathcal{L}(M)}{\bowtie}$. The importance

of this characterization stems from the fact that it is easier to imagine computing $\overset{M}{\sim}$, which is a relation over a finite set (the states of M), than it is to imagine computing $\overset{L(M)}{\bowtie}$, which is a relation over an infinite set (Σ^*).

Theorem 1.4. *Let $M = \langle Q, \Sigma, \delta, q_0, A \rangle$ be a finite automaton accepting language L. Then the minimum-state finite automaton $M_L = \langle Q_L, \Sigma, \delta_L, q_L, A_L \rangle$ for L may be given as follows:*

$$\begin{aligned} Q_L &= \{\, [q]_{\underset{\sim}{M}} \mid q \in Q \,\} \\ q_L &= [q_0]_{\underset{\sim}{M}} \\ A_L &= \{\, [q]_{\underset{\sim}{M}} \mid q \in A \,\} \\ \delta([q]_{\underset{\sim}{M}}, a) &= [\delta(q,a)]_{\underset{\sim}{M}} \end{aligned}$$

Note that Q_L is is the set of equivalence classes of $\overset{M}{\sim}$.

On the basis of this theorem, if we can develop a mechanism for computing the equivalence classes of $\overset{M}{\sim}$ then we effectively have a method for building minimum-state finite automata.

2. Computing $\overset{M}{\sim}$

We now turn to the task of developing an algorithm for computing the equivalence classes of $\overset{M}{\sim}$. The procedure we give is *partition-based*, meaning that it operates on equivalence classes of states. In a nutshell, it works by first taking equivalence classes that are too large (that is, that include states that are not necessarily language equivalent) and then *splitting* them into smaller and smaller classes until we reach the equivalence classes of $\overset{M}{\sim}$.

Given this somewhat vague overview, the first question that one might ask is, "Where do we begin?" In other words, what are the largest equivalence classes that we can reasonably begin working on? Of course, the largest possible equivalence class would include every state; but one can also observe the following.

Lemma 2.5. *Let $M = \langle Q, \Sigma, \delta, q_0, A \rangle$ be a finite automaton. If $q_1 \in A$ and $q_2 \in Q - A$ then $q_1 \overset{M}{\not\sim} q_2$.*

Therefore, we can in fact begin with two equivalence classes: A, and $Q - A$.

The second question that one might ask is the following. Suppose we have a collection P (for "partition") of equivalence classes, and suppose we know that if two states, q_1 and q_2, are in different equivalence classes in P then $q_1 \overset{M}{\not\sim} q_2$. Note that this is certainly true of the initial partition (A and $Q - A$) given above. Now, it may be the case that some of the equivalence classes of P are still too large (that is, include states that aren't equivalent to each other) and therefore may need to be split. How do we determine if this is the case, and if it is, how do we split equivalence classes? The answer to this question can be found in the next lemma.

Lemma 2.6. *Let $M = \langle Q, \Sigma, \delta, q_0, A \rangle$, and suppose $q_1' \overset{M}{\not\sim} q_2'$. Also suppose that $\delta(q_1, a) = q_1'$ and $\delta(q_2, a) = q_2'$. Then $q_1 \overset{M}{\not\sim} q_2$.*

algorithm *equiv*
Input: Finite automaton $M = \langle Q, \Sigma, \delta, q_0, A \rangle$.
Output: Equivalence classes of $\overset{M}{\sim}$ (in variable P).
Method:

```
P := {A, Q − A};
check := true;
while check do begin
  check := false;
  foreach B ∈ P do
    if ∃a ∈ Σ, q1, q2 ∈ B, B′ ∈ P. δ(q1, a) ∈ B′ ∧ δ(q2, a) ∉ B′
    then begin
      B1 := { q ∈ B | δ(q, a) ∈ B′ };
      B2 := B − B1;
      P := (P − {B}) ∪ {B1, B2}
      check := true;
    end
end
```

Figure 1: The algorithm for computing equivalence classes of $\overset{M}{\sim}$.

Now, suppose we have two equivalence classes $B, B' \in P$ and a symbol $a \in \Sigma$ with the following property: there exist $q_1, q_2 \in B$ with $\delta(q_1, a) \in B'$ and $\delta(q_2, a) \notin B'$. From the property we are assuming of P, this means that $\delta(q_1, a) \overset{M}{\not\sim} \delta(q_2, a)$. But from the lemma, this means that $q_1 \overset{M}{\not\sim} q_2$! Therefore, we can split B into two new equivalence classes: one containing the states (like q_1) containing an a-transition into B', and one containing those that do not. (Note that B and B' do not have to be different; they can be the same equivalence class.) Conversely, if no such B, B' and a exist, then no more partitions need splitting, and the equivalence classes of P are the equivalence classes of $\overset{M}{\sim}$.

With these observations in hand, we can now give a formal "pseudo-code" account of the algorithm for computing the equivalence classes of $\overset{M}{\sim}$. The code appears in Figure 1.

3. Examples

We close the handout with a couple of examples illustrating how the algorithm works. When executing *equiv* by hand, it helps to maintain a couple of data structures:

1. a table recording, for each state, what equivalence class it is presently in; and
2. a "tree" that records how equivalence classes are split.

Intuitively, the leaves in the tree represent the current equivalence classes in P. When an equivalence class B is split, we will make the two new equivalence classes children of it in this tree;

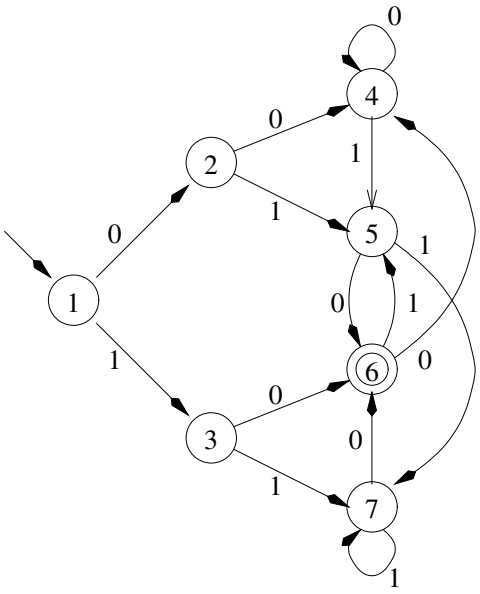


Figure 2: Problem 1.

moreover, we will label the left branch with the symbol a and the equivalence class B' that caused the split, and we will put the set $\{ q \in B_1 \mid \delta(q,a) \in B' \}$ as the left child of B, with the other set being the right child.

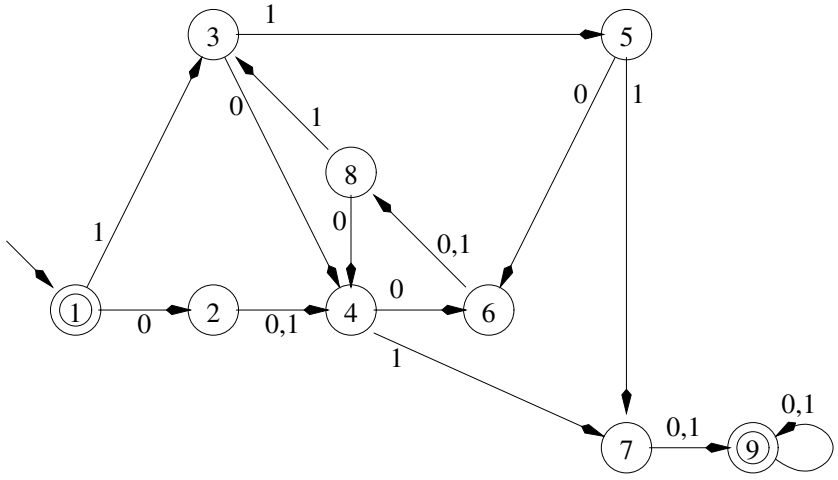
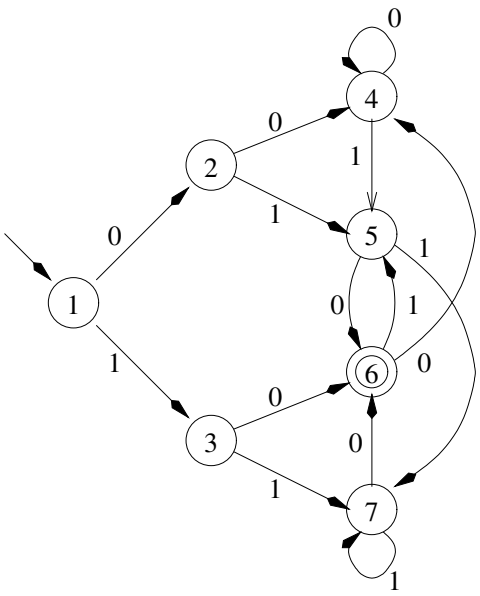
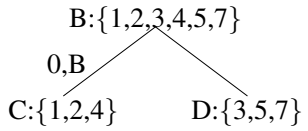


Figure 3: Problem 2.



state	equivalence class
1	B C
2	B C
3	B D
4	B C
5	B D
6	A
7	B D

A:{6}



Equivalence classes: A, C, D
Minimized automaton:

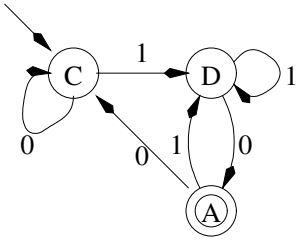
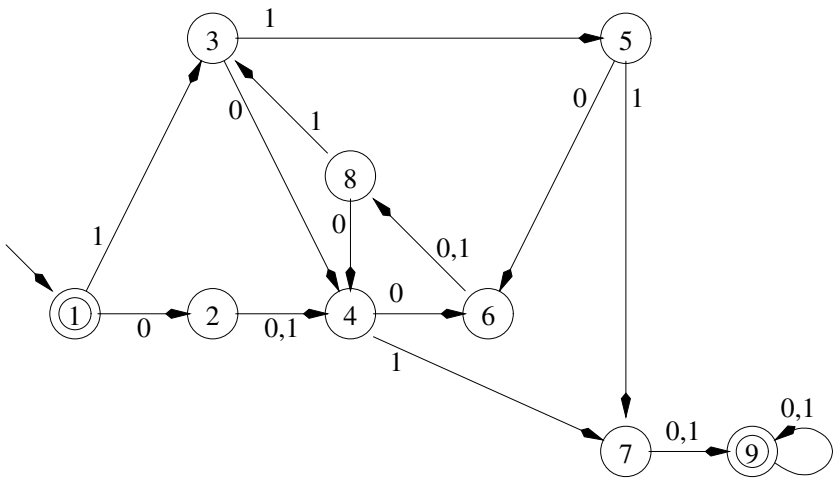
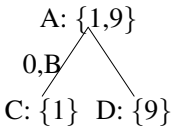
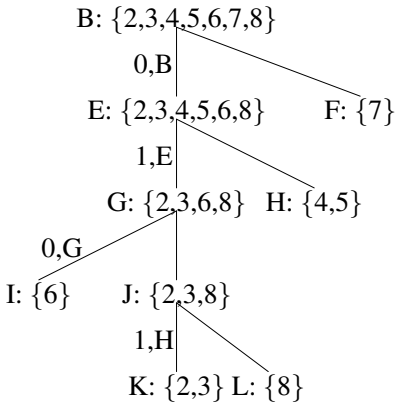


Figure 4: Problem 1 solution.



state	equivalence class
1	A C
2	B E G J K
3	B E G J K
4	B E H
5	B E H
6	B E G I
7	B F
8	B E G J L
9	A D





Equivalence classes: C, D, F, H, I, K, L
Minimized automaton:

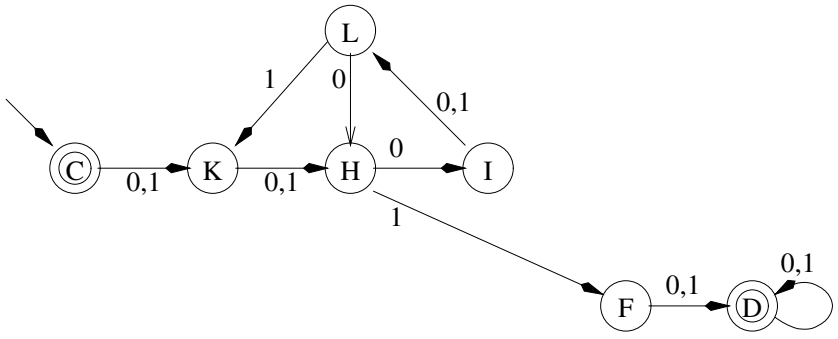


Figure 5: Problem 2 solution.