Describing an $n \log n$ algorithm for minimizing states in deterministic finite automaton

Yingjie XU

January 3, 2009

Abstract

There are several well known algorithms to minimize deterministic finite automata. In this paper, an algorithm is given for minimizing the number of states in a finite automaton or for determining if two finite automata are equivalent. The asymptotic running time of the algorithm is bounded by $kn \log n$ where k is some constant depends linearly on the size of the input alphabet and n is the number of states.

Contents

1 Introduction

The problem of writing efficient algorithms to find the minimal deterministic finite automaton equivalent to a given automaton can be traced back to 1950's with the works of Huffman [Huf55] and Moore [Moo58]. Over the years several alternative algorithms were proposed. However, a worst case analysis of these algorithms indicate that they are n^2 processes where n is the number of states. Hopcroft [Hop71] introduced in 1971 an $O(n \log n)$ algorithm for minimizing a finite deterministic automaton of n states. The constant of proportionality depends linearly on the number of input symbols. Clearly the same algorithm can be used to determine if two finite automata are equivalent. Recent researches [BC04, BBC08] show this bound is tight.

The text is organized as follows. In Section 2 we present some definitions and notation used throughout the paper. In Section 3 we describe Hopcroft's algorithm, explain how it works.

2 Preliminaries

2.1 DFA, word and language

Definition 2.1. *A* deterministic finite automaton *(DFA)* $\mathcal{D}$ *is a tuple* $(Q, \Sigma, \delta, q_0, F)$ *where:*

- Q *is a finite set of* states*;*
- Σ *is the* input alphabet*;*
- $\delta : Q \times \Sigma \to Q$ *is the* transition function*;*
- q_0 *is the* initial state*;*
- $F \subseteq Q$ *is the set of* final states.

When the transition function is total, the automaton $\mathcal{D}$ is said to be *complete*.

Definition 2.2. *Any finite sequence of alphabet symbols* $a \in \Sigma$ *is a* word.

Let Σ^* denote the set of all words over the alphabet Σ and ϵ denote the *empty word*. We define the *extended transition function* $\hat{\delta} : Q \times \Sigma^* \to Q$ in the following way:

$$\begin{cases} \hat{\delta}(q, \epsilon) & = q \\ \hat{\delta}(q, xa) & = \delta(\hat{\delta}(q, x), a) \end{cases}$$

Definition 2.3. *The* language *accepted by* $\mathcal{D}$*,* $L(\mathcal{D})$*, is the set of all words* $w \in \Sigma^*$ *such that* $\hat{\delta}(q_0, w) \in F$.

2.2 Equivalent states and Minimal DFA

Given a DFA $\mathcal{D} = (Q, \Sigma, \delta, q_0, F)$, two states $q_1, q_2 \in Q$ are said to be *equivalent*, denoted $q_1 \approx q_2$, if for every $w \in \Sigma^*$, $\hat{\delta}(q_1, w) \in F \Leftrightarrow \hat{\delta}(q_2, w) \in F$. Two states that are not equivalent are called *distinguishable*. The equivalent minimal automaton $\mathcal{D}/\approx$ is called *quotient automaton*, and its states correspond to the equivalence class of $\approx$. It is proved to be unique up to isomorphism.

Definition 2.4. *Two DFAs* $\mathcal{D}$ *and* $\tilde{\mathcal{D}}$ *are* equivalent *if and only if* $L(\mathcal{D}) = L(\tilde{\mathcal{D}})$.

Definition 2.5. *A DFA is called* minimal *if there is no other equivalent DFA with fewer states.*

3 Minimization

3.1 The classical algorithm

Definition 3.1. *A state* $q \in Q$ *of a DFA* $\mathcal{D} = (Q, \Sigma, \delta, q_0, F)$ *is called* accessible *if* $\hat{\delta}(q_0, w) = q$ *for some* $w \in \Sigma^*$.

Definition 3.2. *If all states in* Q *are accessible, a complete DFA* $\mathcal{D}$ *is called* initially-connected *(ICDFA).*

Lemma 3.1. *Suppose that* $\mathcal{D}$ *is a ICDFA. Then* $\mathcal{D}$ *is minimal if and only if all pairs of states are distinguishable.*

Proof. ($\Rightarrow$) This implication is easy, for if $\mathcal{D}$ has two equivalent states, one of them can be eliminated, and the transitions into this state can be changed to go to the other one. This will not affect the accepted language.

($\Leftarrow$) Assume that in $\mathcal{D}$ all states are pairwise distinguishable. Let $\mathcal{D}$ have k states. Consider any other $\tilde{\mathcal{D}}$ with $l < k$ states. We need to prove that $L(\tilde{\mathcal{D}}) \neq L(\mathcal{D})$.

For each state q of $\mathcal{D}$, choose arbitrarily one string w_q such that $\hat{\delta}(q_0, w_q) = q$. That such w_q exists for each q follows from the assumption that all states are accessible. Since $l < k$, there are two states $p \neq q$ of $\mathcal{D}$ such that in $\tilde{\mathcal{D}}$ we have $\hat{\delta}(q_0, w_p) = \hat{\delta}(q_0, w_q)$. Since p, q are distinguishable in $\mathcal{D}$, there is a string w such that $\hat{\delta}(q_w) \in F$ but $\hat{\delta}(p_w) \notin F$ (or vice verse, but if so, we can always swap p and q). This means that $\mathcal{D}$ accepts $w_p w$ but not $w_q w$. But in $\tilde{\mathcal{D}}$, we have $\hat{\delta}(q_0, w_p w) = \hat{\delta}(q_0, w_q w)$, so $\tilde{\mathcal{D}}$ either accepts both $w_p w$ and $w_q w$ or rejects both. Therefore $L(\tilde{\mathcal{D}}) \neq L(\mathcal{D})$, as claimed. □

Lemma 3.2. *State indistinguishability is an equivalence relation.*

Lemma 3.3. *Let $\delta(p, a) = p'$ and $\delta(q, a) = q'$. Then, if p', q' are distinguishable then so are p, q.*

To minimize automaton $\mathcal{A}$, after removing unreachable states, we will find all equivalence class of the indistinguishability relation and join all states in each class into one state of the new automaton $\tilde{\mathcal{A}}$.

To determine which states are equivalent, we will use the following algorithm. Instead of trying to figure out which states are indistinguishable, we will try to figure out which states are distinguishable. The complete algorithm is given below.

Algorithm 1 The classical algorithm

Require: Deterministic finite automaton $\mathcal{A}$
Ensure: Minimal automaton $\tilde{\mathcal{A}}$ equivalent to $\mathcal{A}$

```
1: Remove inaccessible states
2: Mark all pairs p, q, where p ∈ F and q ∉ F
3: repeat
4:     for all non-marked pairs p, q do
5:         for all symbol a do
6:             if the pair δ(p, a), δ(q, a) is marked then
7:                 mark p, q
8:             end if
9:         end for
10:    end for
11: until no new pairs are marked
12: Construct the reduced automaton Ã
```

Theorem 3.1. *Minimization algorithm 3.1 is correct, that is $L(\tilde{\mathcal{A}}) = L(\mathcal{A})$ and $\tilde{\mathcal{A}}$ is minimal.*

Proof. We prove all the required conditions, one by one.

The equivalence are correct The proof is quite easy, by induction on the length of the shortest word that distinguishes p, q, using Lemma 3.3.

$\tilde{\mathcal{A}}$ is well define The fact that indistinguishability is an equivalence relation from Lemma 3.2 implies that the states of $\tilde{\mathcal{A}}$ are well-defined.

$L(\tilde{\mathcal{A}}) = L(\mathcal{A})$ For each w we have $\hat{\delta}(q_0, w) \in \hat{\delta}(\hat{q}_0, w)$. Thus, by the definition of the final states of $\tilde{\mathcal{A}}$, $\tilde{\mathcal{A}}$ accepts w if and only if $\mathcal{A}$ accepts w.

$\tilde{\mathcal{A}}$ is minimal This is where Lemma 3.1 is helpful. This is quite obvious from the construction that all states are distinguishable in $\tilde{\mathcal{A}}$. □

Theorem 3.2. *The running time of Algorithm 3.1 is $O(n^2)$.*

3.2 The Hopcroft's algorithm

Lemma 3.4. *Let $\mathcal{D} = (Q, \Sigma, \delta, q_0, F)$ and a series ρ_i $(i \geq 0)$ of equivalence relations on Q be defined as follows:*

$$\rho_0 = \{(p, q) | p, q \in F\} \cup \{(p, q) | p, q \in Q - F\},$$
$$\rho_{i+1} = \{(p, q) \in \rho_i | (\forall a \in \Sigma)(\delta(p, a), \delta(q, a)) \in \rho_i\}.$$

Then the following holds:

- $\rho_0 \supseteq \rho_1 \supseteq \cdots$.
- *If $\rho_i = \rho_{i+1}$ then $\rho_i = \rho_j$ for all $j > i$.*
- *There exists $0 \leq k \leq |Q|$ such that $\rho_k = \rho_{k+1}$.*

Now consider the situation where $\rho_i \neq \rho_{i+1}$,

$$\begin{array}{rcll} \rho_i \neq \rho_{i+1} & \Leftrightarrow & (\exists p, q \in Q, a \in \Sigma) & (p, q) \in \rho_i \texttt{ and } (\delta(p, a), \delta(q, a)) \notin \rho_i \\ & \Leftrightarrow & (\exists U \in Q/\rho_i, a \in \Sigma) & p, q \in U \texttt{ and } (\delta(p, a), \delta(q, a)) \notin \rho_i \\ & \Leftrightarrow & (\exists U, V \in Q/\rho_i, a \in \Sigma) & p, q \in U \texttt{ and } \delta(p, a) \in V \texttt{ and } \delta(q, a) \notin V \\ & \Leftrightarrow & (\exists U, V \in Q/\rho_i, a \in \Sigma) & \delta(U, a) \cap V \neq \emptyset \texttt{ and } \delta(U, a) \not\subseteq V \end{array} \tag{1}$$

Using Lemma 3.4, equivalent relation on $\mathcal{D}$ can be computed by the algorithm below:

Algorithm 2 Computing equivalent relation using refinements

```
Q/θ ← {F, Q − F}
while (∃U, V ∈ Q/θ, a ∈ Σ) s.t. Equation 1 holds do
    Q/θ ← (Q/θ − {U}) ∪ {U ∩ δ^{-1}(V, a), U − U ∩ δ^{-1}(V, a)}
end while
```

As such, it is yet rather abstract, and in particular we have to decide how to efficiently find some triple U, V, a for which Equation 1 holds.

Lemma 3.5. *Let $\mathcal{D} = (Q, \Sigma, \delta, q_0, F)$ and $U \in Q/\theta$. Suppose we refine U into U' and U''. Then, for any $a \in \Sigma$, refining all the classes of θ with respect to (U', a) and (U'', a) yields the same result as refining θ with respect to (U, a), (U', a) and (U'', a).*

Lemma 3.6. *Let $\mathcal{D} = (Q, \Sigma, \delta, q_0, F)$, $Q/\theta = \{F, Q - F\}$, and $a \in \Sigma$. Then refining θ with respect to either (F, a) or $(Q - F, a)$ yields the same result as refining θ with respect to both of them.*

Algorithm 3 Hopcroft's algorithm

```
1: W ← {F, Q − F}
2: P ← {F, Q − F}
3: while W is not empty do
4:     select and remove S from W
5:     for all a ∈ Σ do
6:         l_a ← δ^{-1}(S, a)
7:         for all R in P such that R ∩ l_a ≠ ∅ and R ⊈ l_a do
8:             partition R into R_1 and R_2: R_1 ← R ∩ l_a and R_2 ← R − R_1
9:             replace R in P with R_1 and R_2
10:            if R ∈ W then
11:                replace R in W with R_1 and R_2
12:            else
13:                if |R_1| ≤ |R_2| then
14:                    add R_1 to W
15:                else
16:                    add R_2 to W
17:                end if
18:            end if
19:        end for
20:    end for
21: end while
```

Lemmas 3.5 and 3.6 provide a possibility to reduce calculation. Algorithm 3.2 is the Hopcroft's algorithm:

Theorem 3.3. *Let $\mathcal{U} = \{U_1, U_2, \ldots, U_m\}$ $(1 \leq m \leq |Q|)$ be the set of all built classes created in the minimization algorithm. Then*

$$|\mathcal{U}| = \sum_{i=1}^{m} |U_i| \leq |Q| \log |Q|$$

Theorem 3.4. *The running time of Algorithm 3.2 is $O(n \log n)$.*

References

[BBC08] Jean Berstel, Luc Boasson, and Olivier Carton. Hopcroft's automaton minimization algorithm and sturmian words. In *DMTCS'2008*, volume AI, pages 355–366, 2008.

[BC04] Jean Berstel and Olivier Carton. On the complexity of hopcroft's state minimization algorithm. In *CIAA'2004*, volume 3317, pages 35–44, 2004.

[Hop71] J. E. Hopcroft. An $n \log n$ algorithm for minimizing states in a finite automaton. Technical Report CS-71-190, Stanford University, January 1971.

[Huf55] D. A. Huffman. The synthesis of sequential switching circuits. *The Journal of Symbolic Logic*, 20(1), 1955.

[Moo58] E. F. Moore. Gedanken-experiments on sequential machines. *The Journal of Symbolic Logic*, 23(1), 1958.