Final year intership report

Verification in Isabelle/HOL of Hopcroft's algorithm for minimizing DFAs including runtime analysis

Vincent Trélat

April 5, 2023

Contents

1 Introduction **2**

1.1 Original algorithm 2

1.2 Modern formalisation 2

2 Proof of correctness **3**

3 Time complexity analysis **3**

1 Introduction

1.1 Original algorithm

John E. Hopcroft's algorithm for minimizing DFAs was first presented in his original 1971 paper [Hop71] as a formal algorithm. Algorithm 1 is a direct translation of the original algorithm with only slight changes in the notations.

Algorithm 1: Hopcroft's original formal algorithm

Data: Input: a finite DFA $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{F})$

Result: Output: the equivalence class of $\mathcal{Q}$ under state equivalence

Construct $\delta^{-1}(q, a) := \{t \in \mathcal{Q} \mid \delta(t, a) = q\}$ for all $q \in \mathcal{Q}$ and $a \in \Sigma$;

Construct $P_1 := \mathcal{F}$, $P_2 := \mathcal{Q} \setminus \mathcal{F}$ and $a_i := \{q \in P_i \mid \delta^{-1}(q, a) \neq \varnothing\}$ for all $i \in \{1, 2\}$ and $a \in \Sigma$;

Let $k := 3$;

For all $a \in \Sigma$, construct $L_a := \underset{0 \leq i < k}{\arg\min}\ |a_i|$;

while $\exists a \in \Sigma, L_a \neq \varnothing$ **do**

    Pick $a \in \Sigma$ such that $L_a \neq \varnothing$ and $i \in L_a$;

    $L_a := L_a \setminus \{i\}$;

    forall $j < k, \exists q \in P_j, \delta(q, a) \in a_i$ **do**

        $P'_j := \{t \in P_j \mid \delta(t, a) \in a_i\}$ and $P''_j := P_j \setminus P'_j$;

        $P_j := P'_j$ and $P_k := P''_j$; construct a_j and a_k for all $a \in \Sigma$ accordingly ;

        For all $a \in \Sigma$, $L_a := \begin{cases} L_a \cup \{j\} & \text{if } j \notin L_a \wedge |a_j| \leq |a_k| \\ L_a \cup \{k\} & \text{otherwise} \end{cases}$;

        $k := k + 1$;

    end

end

1.2 Modern formalisation

Today, the algorithm is usually given in a more mathematical and formalised way[1], as presented below in Algorithm 2.

Definition 1. Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, I, F)$ be a DFA. Let P be a partition of Q. Let $B \in P$ and $a \in \Sigma$. We say for $C \in P$ that (a, C) splits B if

$$\exists q_1, q_2 \in B \quad \delta(q_1, a) \in C \wedge \delta(q_2, a) \notin C.$$

[1] see for example [EB23]

If (a, C) is a splitter of B, P can be updated to $P \setminus \{B\} \cup \{B', B''\}$, where

$$B' := \{q \in B \mid \delta(q, a) \in C\} \text{ and } B'' := B \setminus B'.$$

Algorithm 2: Hopcroft's algorithm in a modern style

Data: Input: a finite DFA $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{F})$
Result: Output: the language partition P_ℓ

if $\mathcal{F} = \varnothing \vee \mathcal{Q} \setminus \mathcal{F} = \varnothing$ **then**
  return $\mathcal{Q}$
else
  $P := \{\mathcal{F}, \mathcal{Q} \setminus \mathcal{F}\}$;
  $\mathcal{W} := \{(a, \min\{\mathcal{F}, \mathcal{Q} \setminus \mathcal{F}\}, a \in \Sigma\}$;
  while $\mathcal{W} \neq \varnothing$ **do**
    Pick (a, B') from $\mathcal{W}$;
    forall $B \in P$ **do**
      Split B with (a, B') into B_0 and B_1 ;
      $P := (P \setminus \{B\}) \cup \{B_0, B_1\}$;
      forall $b \in \Sigma$ **do**
        if $(b, B \in \mathcal{W})$ **then**
          $\mathcal{W} := (\mathcal{W} \setminus \{(b, B)\}) \cup \{(b, B_0), (b, B_1)\}$;
        else
          $\mathcal{W} := \mathcal{W} \cup \{(b, \min\{B_0, B_1\})\}$;
        end
      end
    end
  end
end

2 Proof of correctness

3 Time complexity analysis

We focus on the original algorithm presented in Algorithm 1 in order to work on the arguments given in [Hop71]. The data structures used at that time were mostly linked lists, but let us rather give some requirements for the data structures instead of actual implementations. The goal is to show that the algorithm can be executed in $O(m \cdot n \log n)$ time, where m is the number of symbols in the alphabet and n is the number of states in the DFA.

The following requirements come directly from [Hop71] and are specific to the algorithm presented in Algorithm 1.

Requirement 1. Sets such as $\delta^{-1}(q,a)$ and L_a must be represented in a way that allows $O(1)$ time for addition and deletion in front position.

Requirement 2. Vectors must be maintained to indicate whether a state is in a given set.

Requirement 3. Sets such as P_i must be represented in a way that allows $O(1)$ time for addition and deletion at any given position.

Requirement 4. For a state q in a set P_i or a_i, its position must be determined in $O(1)$ time.

VT: Maybe not necessary? This should be provable from Req. 2 and Req. 3.

Lemma 1. Lines 1 to 4 can be executed in $O(|\Sigma| \cdot |\mathcal{Q}|)$ time.

Proof. The non trivial part is the computation of the inverse transition function $\delta^{-1}(q,a)$, for all $q \in \mathcal{Q}$ and $a \in \Sigma$. This can be done in $O(|\Sigma| \cdot |\mathcal{Q}|)$ time by iterating over Σ and traversing the automaton (e.g. with a DFS) while keeping track of the predecessor at each step. ∎

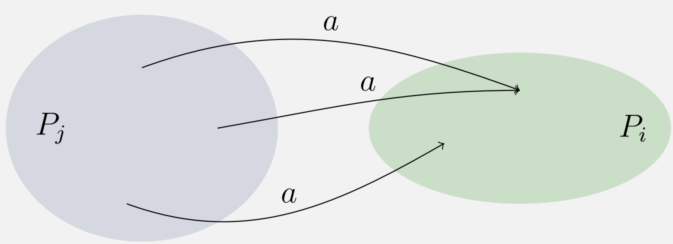
Lemma 2. An iteration of the loop at line 5 taken for a letter a and an index $i \in L_a$ takes a time proportional to the number of transitions terminating in P_i and the number of symbols in the alphabet, i.e. $\Theta\left(|\Sigma| \cdot \left|\bigcup_{q \in P_i} \delta^{-1}(q,a)\right|\right)$ time.

Proof. We pick an $a \in \Sigma$ such that $L_a \neq \varnothing$ and an $i \in L_a$. We need to examine all $j < k$ such that $\exists q \in P_j, \delta(q,a) \in a_i$ to construct the sets corresponding to splitting the block P_j w.r.t. a and P_i.

Let $j < k$. From the definition of a_i, we obtain the following:

$$\begin{aligned} \exists q \in P_j, \delta(q,a) \in a_i &\iff \exists q \in P_j, \delta(q,a) \in P_i \wedge \underbrace{\delta^{-1}(\delta(q,a),a) \neq \varnothing}_{\text{true}} \\ &\iff \exists q \in P_j, \delta(q,a) \in P_i \end{aligned}$$

Which corresponds to finding states in P_j having an outgoing a-transition to a state in P_i, as represented in the following scheme:



This set of states can be expressed via the inverse transition function:

$$\{q \in P_j \mid \delta(q,a) \in P_i\} = \left(\bigcup_{q \in P_i} \delta^{-1}(q,a) \right) \cap P_j$$

Since δ^{-1} was already computed in the first step of the algorithm, we can determine using req. 3 whether a state of $\bigcup_{q \in P_i} \delta^{-1}(q,a)$ is also in P_j in $\Theta(1)$ time.

Thus, instead of examining P_j for all $j < k$, we rather go through the table of δ^{-1} and for each state q such that $\delta(q,a) \in P_i$, we know from req. 4 that we can determine the index $j < k$ (because there are k blocks) of the block P_j containing q in $\Theta(1)$ time. The sets P'_j and $P''_j = P_k$ can be constructed on the fly without any additional time cost. The construction of the sets b_j and b_k as well as the update of L_b for all $b \in \Sigma$ can also be done on the fly but require $\Theta(1)$ time for each symbol $b \in \Sigma$ and thus add up to a total of $\Theta(|\Sigma|)$ time.

Overall, an iteration of the loop takes $\Theta\left(|\Sigma| \cdot \left|\bigcup_{q \in P_i} \delta^{-1}(q,a)\right|\right)$ time.

■

References

[EB23] Javier Esparza and Michael Blondin. *Automata Theory: An Algorithmic Approach.* 2023.

[Hop71] John E. Hopcroft. *An n Log n Algorithm for Minimizing States in a Finite Automaton.* Stanford University, Stanford, CA, USA, 1971.