



Final year internship report

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

Vincent Trélat

April 14, 2023



Contents

1	Introduction	2
1.1	Original algorithm	2
1.2	Modern formalisation	2
2	Proof of correctness	4
3	Time complexity analysis	4

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: a finite DFA $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{F})$
Result: the equivalence class of \mathcal{Q} under state equivalence

- 1 Construct $\delta^{-1}(q, a) := \{t \in \mathcal{Q} \mid \delta(t, a) = q\}$ for all $q \in \mathcal{Q}$ and $a \in \Sigma$;
- 2 Construct $P_1 := \mathcal{F}$, $P_2 := \mathcal{Q} \setminus \mathcal{F}$ and $s_{a,i} := \{q \in P_i \mid \delta^{-1}(q, a) \neq \emptyset\}$ for all $i \in \{1, 2\}$ and $a \in \Sigma$;
- 3 Let $k := 3$;
- 4 For all $a \in \Sigma$, construct $L_a := \arg \min_{1 \leq i < k} |s_{a,i}|$;
- 5 **while** $\exists a \in \Sigma, L_a \neq \emptyset$ **do**
- 6 Pick $a \in \Sigma$ such that $L_a \neq \emptyset$ and $i \in L_a$;
- 7 $L_a := L_a \setminus \{i\}$;
- 8 **forall** $j < k, \exists q \in P_j, \delta(q, a) \in s_{a,i}$ **do**
- 9 $P'_j := \{t \in P_j \mid \delta(t, a) \in s_{a,i}\}$ and $P''_j := P_j \setminus P'_j$;
- 10 $P_j := P'_j$ and $P_k := P''_j$; construct $s_{a,j}$ and $s_{a,k}$ for all $a \in \Sigma$ accordingly ;
- 11 For all $a \in \Sigma$, $L_a := \begin{cases} L_a \cup \{j\} & \text{if } j \notin L_a \wedge |s_{a,j}| \leq |s_{a,k}| \\ L_a \cup \{k\} & \text{otherwise} \end{cases}$;
- 12 $k := k + 1$;
- 13 **end**
- 14 **end**

1.2 Modern formalisation

The algorithm is now usually given in a more mathematical and formalised way¹, as presented below in Algorithm 2. We justify this transformation.

First, instead of storing indices for the blocks, we directly deal with sets and explicitly work with a partition \mathcal{P} of \mathcal{Q} . Then, we define splitters.

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

¹see for example [EB23]

splitter of B if

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

Remark 1. If (a, C) is a splitter of B , P can be updated to $P \setminus \{B\} \cup \{B', B''\}$, where

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

Definition 2. For a splitter $s \in \Sigma \times P$, we define its size $\|s\|$ as follows. If $s = (a, C)$, then $\|s\| := |C|$.

In Algorithm 1, picking an index $i \in L_a$ corresponds to picking a set $s_{a,i}$, i.e. the subset of P_i of states having at least one predecessor. We cannot directly replace L_a and $s_{a,i}$ by a set of splitters (a, P_i) as is, however the next step in the algorithm is to go over all blocks in the partition that have a successor in $s_{a,i}$. If such a block B is found, it is split into two blocks, namely the set of states having a successor in $s_{a,i}$ and the others. Overall, we look for B such that P_i has a predecessor in B . We add the condition that B must also have at least one successor not in P_i to avoid splitting B into $\{B, \emptyset\}$. Since this is equivalent to splitting B with (a, P_i) , we can replace L_a by a set of splitters (a, P_i) . Since the symbol a is chosen such that $L_a \neq \emptyset$, we can define a workset \mathcal{W} of all splitters, with all symbols of Σ . Therefore, L_a is empty if and only if there is no a -splitter in \mathcal{W} . Because testing emptiness is equivalent to testing membership, and because of the existential quantifier in the definition of splitters, the $s_{a,i}$ are no longer needed.

We can explicitly write the trivial cases where either \mathcal{F} or $\mathcal{Q} \setminus \mathcal{F}$ is empty to improve performance for this specific case. This is not done in Algorithm 1 but this does not produce wrong results. If one of the two sets is empty, one of the $s_{a,1}$ or $s_{a,2}$ will be empty for all $a \in \Sigma$, and the *while* loop will be entered $|\Sigma|$ times but the inner *for* loop will never be entered. This results in a useless $O(|\Sigma|)$ computation. Another difference in that case is that it would compute the partition $\{\min\{\mathcal{F}, \mathcal{Q} \setminus \mathcal{F}\}, \emptyset\}$ – where \min returns the set with the least number of elements – which we avoid in the modern version by just returning \mathcal{Q} .

Algorithm 2: Hopcroft's algorithm in a modern style

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

```
1 if  $\mathcal{F} = \emptyset \vee \mathcal{Q} \setminus \mathcal{F} = \emptyset$  then
2   | return  $\mathcal{Q}$ 
3 else
4   |  $\mathcal{P} := \{\mathcal{F}, \mathcal{Q} \setminus \mathcal{F}\}$  ;
5   |  $\mathcal{W} := \{(a, \min\{\mathcal{F}\}, \mathcal{Q} \setminus \mathcal{F}), a \in \Sigma\}$  ;
6   | while  $\mathcal{W} \neq \emptyset$  do
7     | Pick  $(a, B')$  from  $\mathcal{W}$  and remove it;
8     | forall  $B \in \mathcal{P}$  do
9       | Split  $B$  with  $(a, B')$  into  $B_0$  and  $B_1$  ;
10      |  $\mathcal{P} := (\mathcal{P} \setminus \{B\}) \cup \{B_0, B_1\}$  ;
11      | forall  $b \in \Sigma$  do
12        | if  $(b, B) \in \mathcal{W}$  then
13          | |  $\mathcal{W} := (\mathcal{W} \setminus \{(b, B)\}) \cup \{(b, B_0), (b, B_1)\}$  ;
14          | else
15            | |  $\mathcal{W} := \mathcal{W} \cup \{(b, \min\{B_0, B_1\})\}$  ;
16          | end
17        | end
18      | end
19    | end
20 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”².

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 the representation of a set P_i or $s_{a,i}$, its “position” – in the implemented data structure – must be determined in $O(1)$ time.

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

Lemma 1. In Algorithm 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 while loop in both algorithms for a splitter $s = (a, C)$ takes a time proportional to the number of transitions terminating in C and the number of symbols in the alphabet, i.e. $\Theta(|\Sigma| \cdot |\delta^{-1}(C, a)|)$ time.

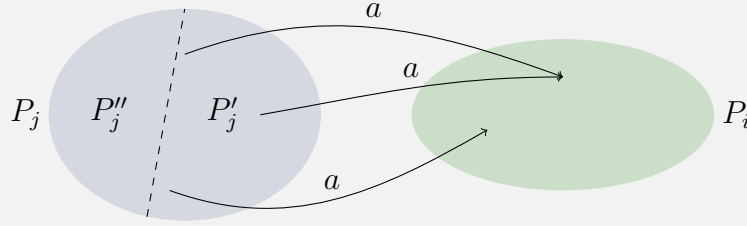
Proof. We start by showing it for Algorithm 1. We pick $a \in \Sigma$ such that $L_a \neq \emptyset$ and an $i \in L_a$. We need to examine all $j < k$ such that $\exists q \in P_j, \delta(q, a) \in s_{a,i}$ to construct the sets corresponding to splitting the block P_j w.r.t. a and P_i .

Let $j < k$. From the definition of $s_{a,i}$, we obtain the following:

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

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:

²This does not make sense for sets and is specific for a data structure. What is meant here is a bit informal and designates a position directly accessible, e.g. the head for a stack, the value of the root for a tree, etc.



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 $P_j'' = P_k$ resulting from the partition can be constructed on the fly without any additional time cost. The construction of the sets $s_{b,j}$ and $s_{b,k}$ 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.

VT: the on-the-fly computation part is a little hand-waving...

Overall, one iteration of the loop runs in time

$$\Theta \left(|\Sigma| \cdot \left| \bigcup_{q \in P_i} \delta^{-1}(q, a) \right| \right)$$

We now show the result for Algorithm 2. A splitter $s =: (a, C)$ is picked from \mathcal{W} . Then, we iterate over all blocks $B \in \mathcal{P}$ that may be split with s . Let $B \in \mathcal{P}$.

$$\begin{aligned} s \text{ splits } B &\iff \exists q_1, q_2 \in B, \delta(q_1, a) \in C \wedge \delta(q_2, a) \notin C \\ &\iff \left(\bigcup_{q \in C} \delta^{-1}(q, a) \right) \cap B \neq \emptyset \wedge \left(\bigcup_{q \in B} \delta(q, a) \right) \setminus C \neq \emptyset \end{aligned}$$

VT: we may express the right-hand conjunct in terms of δ^{-1} in order to prepare the paragraph that follows (But it wouldn't be the same union because you have to start from states outside C)

VT: The second equation may be a little too quick, especially for the right-hand conjunct.

Likewise, instead of examining all blocks $B \in \mathcal{P}$, we go through δ^{-1} and we update the splitters for all symbols in Σ and one iteration of the loop runs in time

$$\Theta \left(|\Sigma| \cdot \left| \bigcup_{q \in C} \delta^{-1}(q, a) \right| \right)$$

■

For the sake of simplicity, we now denote $\bigcup_{q \in C} \delta^{-1}(q, a)$ by $\overset{\leftarrow}{C}$.

We will now justify the logarithmic factor in the time complexity. We briefly explain why a logarithm stands out and prove the statement by induction. The idea is that for each symbol $a \in \Sigma$, a state $q \in \mathcal{Q}$ can be in at most one of the splitters in \mathcal{W} . When the loop iterates over this splitter, it will split the block and keep the smaller one, whose size will be at most the size of the splitter divided by two. This means that a splitter s can be processed at most $\log \|s\|$ times. We now properly state and prove the property by induction.

We see splitters as updatable program variables. This means that $s_{a,i}$ as a set may differ along the loop, however we know there exists some $q \in \mathcal{Q}$ such that $s_{a,i}$ is the unique set containing q throughout the execution. This gives a way to characterize splitters throughout the whole execution.

Lemma 3. Any splitter in the workset $s \in \mathcal{W}$ is processed – i.e. picked at the beginning of the *while* loop – at most $\lfloor \log \|s\| \rfloor$ times.

Proof. We first show the following statement:

During an iteration, the chosen splitter s will either be removed from the set of splitters or its size will be reduced by at least $\frac{\|s\|}{2}$ after the iteration^a.

Let $\{P_1, \dots, P_\ell\}$ be the current partition and let $s = (a, C)$ be the picked splitter. Thus, s is no longer in the set of splitters \mathcal{W} . Since we go over all splittable blocks, C may also be split by s .

- If C is not split by s , then s was already removed from the splitters.

Note that it can be added again later if it is split by another splitter.

- If C is split into C' and C'' , then $(a, \min\{C', C''\})$ is added to the splitters and its size is at most $\frac{\|s\|}{2}$.

Therefore, since a splitter cannot be empty, s can be processed at most m times where m is such that

$$1 \leq \frac{\|s\|}{2^m} < 2$$

which is equivalent to

$$m \leq \log \|s\| < m + 1 \quad \text{i.e.} \quad \lfloor \log \|s\| \rfloor = m$$

■

^aThat is a bit informal because, the original splitter is actually removed and another “similar” splitter (cf the paragraph above the statement of the lemma) of size at most $\frac{\|s\|}{2}$ will be added.

Lemma 4. Any block B resulting from a split and not added as a splitter is processed at most $\lfloor \log \frac{|B|}{2} \rfloor$ times.

Proof. Let $P \in \mathcal{P}$ be a block in the partition and $s =: (a, C)$ be a splitter such that s splits P into S and B so that (a, S) is added to the workset and B is not. In order for B to be processed, some (b, B) must be added to the workset.

Since the only way to create a fresh splitter is to split B , this means that there exists some later step in the loop such that some splitter (b, B') splits B into B_1 and B_2 and $(b, \min\{B_1, B_2\})$ is added to the workset.

From lemma 3, we know that this splitter can be processed at most $\lfloor \log |\min\{B_1, B_2\}| \rfloor$ times. Since $|\min\{B_1, B_2\}| \leq \frac{|B|}{2}$ and since there is no splitter containing B in the workset, we obtain that B can be processed at most $\lfloor \log \frac{|B|}{2} \rfloor$ times.

■

Lemma 5. Let us consider some step in the algorithm such that \mathcal{P} is the current partition. The total time spent in the loop until termination is bounded by

$$T := \theta \left(\sum_{(a,C) \in \mathcal{W}} |\overleftarrow{C}| \log |C| + \sum_{(a,B) \in \Sigma \times \mathcal{P} \setminus \mathcal{W}} |\overleftarrow{B}| \log \frac{|B|}{2} \right)$$

where θ is the constant of proportionality that may be obtained from lemma 2.

Proof. We show the result by induction over the steps.

Base case: the current partition is $\{P_1, P_2\}$ and we may assume w.l.o.g. that P_1 is the smaller set, so that $\mathcal{W} = \{(a, P_1), a \in \Sigma\}$. We have to show that the total time spent in the loop is bounded by

$$T = \theta |\Sigma| \left(|\overleftarrow{P_1}| \log |P_1| + |\overleftarrow{P_2}| \log \frac{|P_2|}{2} \right)$$

We know from lemma 2 that an iteration of the loop for (a, P_i) takes $\theta |\overleftarrow{P_i}|$ time.

- From lemma 3, for all $a \in \Sigma$, (a, P_1) can be processed at most $\log \|(a, P_1)\| = \log |P_1|$ times^a, hence the total time for (a, P_1) is bounded by $\theta |\overleftarrow{P_1}| \log |P_1|$ and thus the total time for splitters with block P_1 is bounded by

$$\theta |\Sigma| |\overleftarrow{P_1}| \log |P_1|$$

- From lemma 4, for all $a \in \Sigma$, (a, P_2) can be processed at most $\log \frac{\|(a, P_2)\|}{2} = \log \frac{|P_2|}{2}$ times. Thus, the total time for non splitters with block P_2 is bounded by

$$\theta |\Sigma| |\overleftarrow{P_2}| \log \frac{|P_2|}{2}$$

Inductive step: Let $\{P_1, \dots, P_\ell\} := \mathcal{P}$ be the current partition. The induction hypothesis states that the total time spent in the loop

until termination for a is bounded by

$$T := \theta \left(\sum_{(a,C) \in \mathcal{W}} |\overleftarrow{C}| \log |C| + \sum_{(a,B) \in \Sigma \times \mathcal{P} \setminus \mathcal{W}} |\overleftarrow{B}| \log \frac{|B|}{2} \right)$$

By going through one more step, some blocks of \mathcal{P} may be split and we define a new time \hat{T} over this new partition and we have to show that $\hat{T} \leq T$.

Suppose some $B \in \mathcal{P}$ is split into B' and B'' by any splitter. We have to consider the cases where (a, B) is a splitter or not for all $a \in \Sigma$. Let $a \in \Sigma$.

- $(a, B) \in \mathcal{W}$: \mathcal{W} is updated as follows:

$$\mathcal{W} := \mathcal{W} \setminus \{(a, B)\} \cup \{(a, B'), (a, B'')\}$$

Thus, instead of taking a time $\theta |\overleftarrow{B}| \log |B|$ time for (a, B) , it now takes a time bounded by:

$$\theta |\overleftarrow{B'}| \log |B'| + \theta |\overleftarrow{B''}| \log |B''|$$

Finally, we show the following:

$$|\overleftarrow{B'}| \log |B'| + |\overleftarrow{B''}| \log |B''| \leq |\overleftarrow{B}| \log |B|$$

From $B = B' \cup B''$, we get $|B| = |B'| + |B''|$ and $|\overleftarrow{B}| = |\overleftarrow{B'}| + |\overleftarrow{B''}|$, hence:

$$\begin{aligned} & |\overleftarrow{B'}| \log |B'| + |\overleftarrow{B''}| \log |B''| - |\overleftarrow{B}| \log |B| \\ &= |\overleftarrow{B'}| \log |B'| + (|\overleftarrow{B}| - |\overleftarrow{B'}|) \log (|B| - |B'|) - |\overleftarrow{B}| \log |B| \\ &= |\overleftarrow{B'}| \log \frac{|B'|}{|B| - |B'|} + |\overleftarrow{B}| \log \frac{|B| - |B'|}{|B|} \\ &\leq |\overleftarrow{B'}| \log \frac{|B|}{|B| - |B'|} + |\overleftarrow{B}| \log \frac{|B| - |B'|}{|B|} \\ &= \underbrace{(|\overleftarrow{B}| - |\overleftarrow{B'}|) \log \frac{|B| - |B'|}{|B|}}_{\leq 1} \leq 0 \end{aligned}$$

- $(a, B) \notin \mathcal{W}$: $(a, \min\{B', B''\})$ is added to the set of splitters. We may assume w.l.o.g. that B' is the smaller set. Thus, the corresponding term in the sum $\overleftarrow{B} \log \frac{|B|}{2}$ is updated as follows:

$$\underbrace{\theta \overleftarrow{B'} \log |B'|}_{\text{lemma 3}} + \underbrace{\theta \overleftarrow{B''} \log \left(\frac{|B''|}{2} \right)}_{\text{lemma 4}}$$

Since $B = B' \cup B''$, the sum above can be written as:

$$\theta \overleftarrow{B'} \log |B'| + \theta (\overleftarrow{B} - \overleftarrow{B'}) \log \left(\frac{|B| - |B'|}{2} \right)$$

By using the fact that $|B'| \leq \frac{|B|}{2}$, we have:

$$\begin{aligned} & \overleftarrow{B'} \log |B'| + (\overleftarrow{B} - \overleftarrow{B'}) \log \left(\frac{|B| - |B'|}{2} \right) - \overleftarrow{B} \log \frac{|B|}{2} \\ & \leq \overleftarrow{B'} \log \frac{|B|}{2} + (\overleftarrow{B} - \overleftarrow{B'}) \log \left(\frac{|B| - |B'|}{2} \right) - \overleftarrow{B} \log \frac{|B|}{2} \\ & \leq (\overleftarrow{B} - \overleftarrow{B'}) \log \left(\frac{|B| - |B'|}{|B|} \right) \leq 0 \end{aligned}$$

Overall, splitting any block of \mathcal{P} does not increase the total time spent over the loop. By iterating over all blocks, we obtain that the total time \hat{T} spent in the loop from this new iteration until termination is such that $\hat{T} \leq T$. ■

^aWe drop the floor operator for simplicity. Note that since we are giving upper bounds, this is completely valid.

Theorem 1. Algorithm 1 runs in $O(|\Sigma| \cdot |\mathcal{Q}| \log |\mathcal{Q}|)$ time.

Proof. From lemma 5, we obtain in particular that for the initial partition the time spent in the loop until termination is bounded by:

$$T = \theta |\Sigma| \left(\overleftarrow{P_1} \log |P_1| + \overleftarrow{P_2} \log \frac{|P_2|}{2} \right)$$

Thus, the following holds:

$$T \leq \theta|\Sigma|(\underbrace{|\vec{P}_1| + |\vec{P}_2|}_{\leq 2|\vec{Q}|}) \log(\underbrace{|P_1| + |P_2|}_{=|\mathcal{Q}|}) \leq 2\theta|\Sigma||\mathcal{Q}| \log |\mathcal{Q}|$$

■

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.