

# Automatic Generation of DFA Minimization Problems

Gregor Sönnichsen

Universität Bayreuth

30. Juni 2020

# Introduction

Automata theory is a classical topic in computer science curricula.  
Minimization of DFAs is a typical task for students:

- ▶ sufficiently easy to understand
- ▶ practical applications
- ▶ understanding can be tested easily

Consequently, studying automatized generation of DFA minimization problems is interesting because it could...

- ▶ ... free up precious time for exercise constructors (if a generator is implemented)
- ▶ ... yield a deeper insight in the nature of such problems



1

---

<sup>1</sup><https://www.rindlerwahn.de/zeitdiebe-besiegen-und-mehr-lebenszeit-gewinnen>

# Outline

Introduction

Problem definition and approach

Generating Minimal DFAs

Extending Minimal DFAs

Live Demonstration and Conclusion

# Problem definition

## Preliminaries (1/2)

A tuple  $A = (Q, \Sigma, \delta, s, F)$  with  $Q, \Sigma$  being a finite,  $\delta: Q \times \Sigma \rightarrow Q$ ,  $s \in Q$  and  $F \subseteq Q$  is called *deterministic finite automaton*.

We define the *extended transition function*  $\delta^*: Q \times \Sigma^* \rightarrow Q$  as:

- ▶  $\delta^*(q, \varepsilon) = q$
- ▶  $\delta^*(q, w\sigma) = \delta(\delta^*(q, w), \sigma)$  for all  $q \in Q$ ,  $w \in \Sigma^*$ ,  $\sigma \in \Sigma$

The *language* of DFA is defined as  $L(A) = \{ w \mid \delta^*(w) \in F \}$ .

We call a DFA *minimal*, if there exists no other DFA with the same language having less states.

# Problem definition

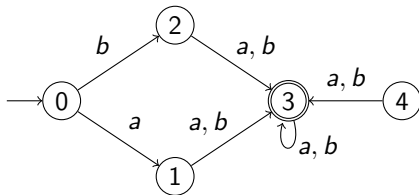
## Preliminaries (2/2)

We say a state  $q$  is *unreachable*, iff there is no word  $w \in \Sigma^*$  such that  $\delta^*(s, w) = q$ .

A state pair  $q_1, q_2 \in Q$  is called *equivalent*, iff  $\sim_A(q_1, q_2)$  is true, where

$$q_1 \sim_A q_2 \Leftrightarrow_{\text{def}} \forall z \in \Sigma^*: (\delta^*(q_1, z) \in F \Leftrightarrow \delta^*(q_2, z) \in F)$$

## Example



## Theorem

A DFA is minimal, iff it has neither unreachable nor equivalent states.

## Hopcroft's Minimization Algorithm

- 
- A portrait of a man with dark hair, wearing glasses and a light-colored, possibly striped, shirt. He is looking slightly to the right of the camera with a neutral expression. The background is dark and out of focus.

2:  $i \leftarrow 0$ 4: **do**6:  $m(i) \leftarrow \{(p, q), (q, p) \mid (p, q) \notin \bigcup m(\cdot) \wedge$ 

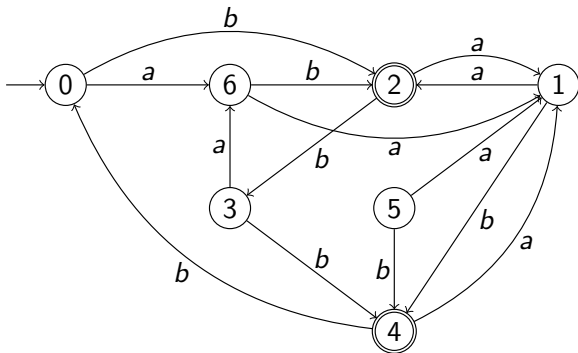
```
8:   while  $m(i) \neq \emptyset$ 
```

4. Merge all equivalent state pairs

# Problem definition

A sample DFA minimization task. . .

Task: Consider the below shown deterministic finite automaton A:



Apply the minimization algorithm and illustrate for each state pair of A during which FindEquivPairs-iteration it was marked. Draw the resulting automaton.

# Problem definition

... and its solution

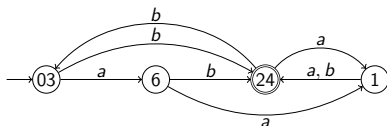
Solution:

*Step 1: Detect and eliminate unreachable states.*

State 5 is unreachable.

*Step 2: Apply FindEquivPairs to A and merge equivalent state pairs:*

	0	1	2	3	4	6
0	■	1	0		0	2
1	■	■	0	1	0	1
2	■	■	■	0		0
3	■	■	■	■	0	2
4	■	■	■	■	■	0
6	■	■	■	■	■	■



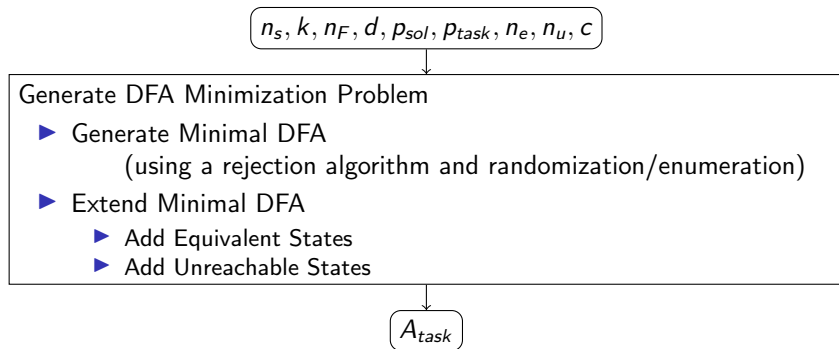


# Problem Definition and Approach

Problem: How to generate a *DFA Minimization Problem*  $A_{task}$ ?

Idea: First generate  $A_{sol}$ , then add equivalent, then unreachable states.

⇒ modular pipeline architecture



# Outline

Introduction

Problem definition and approach

Generating Minimal DFAs

Extending Minimal DFAs

Live Demonstration and Conclusion

# Generating Minimal DFAs

## Rejection Algorithm

Approach: Generate test DFAs until they match the demanded properties.

```
1: function GenNewMinDFA ( $n_s, k, n_F, d, p$ )  
2:    $I \leftarrow$  all DFAs in  $DB_{found}$  matching  $n_s, k, n_F$   
3:   while True do  
4:     generate DFA  $A_{test}$  with  $|Q|, |\Sigma|, |F|$  matching  $n_s, k, n_F$   
5:     if  $A_{test}$  not minimal or  $d \neq \mathfrak{D}(A_{test})$  then  
6:       continue  
7:     if  $p = 1$  and  $A_{test}$  is not planar then  
8:       continue  
9:     if  $A_{test}$  is isomorph to any DFA in  $I$  then  
10:      continue  
11:    save  $A_{test}$  and its respective properties in  $DB_{found}$   
12:    return  $A_{test}$ 
```



# Generating Minimal DFAs

## Test DFA Generation

We will restrict ourselves to the following DFAs:

$$Q = \mathbb{N}_0^{n_s-1}, \Sigma = \mathbb{N}_0^{k-1}, s = 0$$

(a) Generation by Randomization:

$$\begin{aligned} F &= \text{random\_subset}(Q) \\ \delta(q, \sigma) &= \text{choose\_one}(Q) \quad \forall q \in Q, \sigma \in \Sigma \end{aligned}$$

(b) Generation by Enumeration:

An *enumeration state*  $s_{n_s, k, n_F} = (F_F, F_\delta)$  consists of two arrays of length  $n_s, n_s * k$ , respectively.

$$\begin{aligned} F_F[i] &= 1 \Leftrightarrow_{\text{def}} i \in F \\ F_\delta[i * k + j] &= q \Leftrightarrow_{\text{def}} \delta(i, j) = q \end{aligned}$$

Example state:  $s_{4,2,2} = (0110)_2 \ (10 \ 13 \ 22 \ 03)_4$

# Outline

Introduction

Problem definition and approach

Generating Minimal DFAs

Extending Minimal DFAs

Live Demonstration and Conclusion

# Extending Minimal DFAs

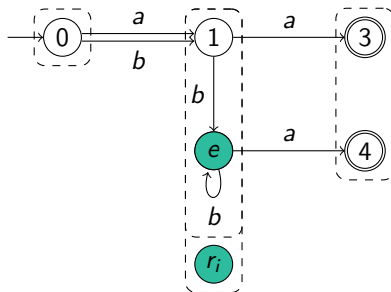
## Adding Equivalent States (1)

We now want to add states  $r_1, \dots, r_{n_e}$  to a DFA, such that every  $r_i$  is equivalent to a state  $e$  in the original DFA:

$$\forall i \in [1, n_e]: \exists e \in Q_{sol}: r_i \sim_A e$$

We will first choose an  $e \in Q_{sol}$  for each state  $r_i$  we create, then we add its transitions.

### Example



# Extending Minimal DFAs

## Adding Equivalent States (2) - Outgoing Transitions

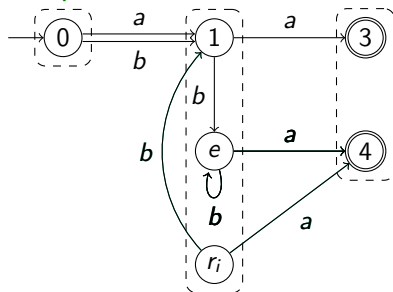
Observation:

$$r_i \sim_A e \implies \forall \sigma \in \Sigma: [\delta(r_i, \sigma)]_{\sim_A} = [\delta(e, \sigma)]_{\sim_A}$$

Consequently:

**R1:** For each symbol  $\sigma \in \Sigma$  choose exactly one state  $q \in [\delta(e, \sigma)]_{\sim_A}$  and set  $\delta(r_i, \sigma) = q$ .

Example



The rule is always fulfillable:

- ▶  $A_{sol}$  is complete
  - ▶ every  $r_i$  gets an out. transition for every alphabet symbol
- $\implies \delta(e, \sigma)$  is always defined, so  $[\delta(e, \sigma)]_{\sim_A}$  is never empty

# Extending Minimal DFAs

## Adding Equivalent States (3) - Ingoing Transitions

First observation: Since  $r_i$  must be reachable,  $in(r_i) \geq 1$ .

Let  $q$  be a state s.t.  $\delta(q, \sigma) = p$  and we want  $\delta(q, \sigma) = r_i$ .

$q$  must remain in its equivalence class

$\Rightarrow p$  must be in  $[r_i]_{\sim_A} = [e]_{\sim_A}$

$\Rightarrow q$  has to have a transition to some state in  $[r_i]_{\sim_A} = [e]_{\sim_A}$

We see that  $p$  must have at least 2 ingoing transitions.

**R2:** Choose at least one  $((q, \sigma), p) \in \delta$  with  $[p] = [e]$  and  $in(p) \geq 2$ .  
Remove  $((q, \sigma), p)$  from  $\delta$  and add  $((q, \sigma), r_i)$ .

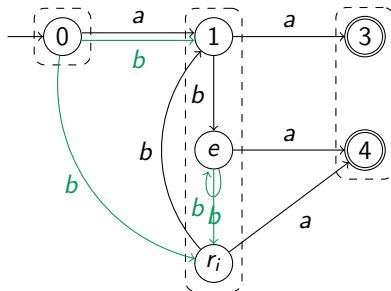


# Extending Minimal DFAs

## Adding Equivalent States (3) - Ingoing Transitions

**R2:** Choose at least one  $((q, \sigma), p) \in \delta$  with  $[p] = [e]$  and  $in(p) \geq 2$ .  
Remove  $((q, \sigma), p)$  from  $\delta$  and add  $((q, \sigma), r_i)$ .

### Example



# Extending Minimal DFAs

## Adding Unreachable States

Reminder: We say a state  $q$  is *unreachable*, iff there is no word  $w \in \Sigma^*$  such that  $\delta^*(s, w) = q$ .

```
1: function AddUnrStates ( $A, n_u, c$ )  
2:    $U \leftarrow \emptyset$   
3:   for  $n_u$  times do  
4:     let  $q$  be the new state  
5:     add ingoing tr. from a random subset of  $U \times \Sigma$   
6:      $\Sigma' \leftarrow$  if  $c = 1$  then  $\Sigma$  else random subset of  $\Sigma$   
7:     add outgoing tr. to  $|\Sigma'|$  random states  
8:     add  $q$  to  $U$   
9:   return  $A$ 
```

# Outline

Introduction

Problem definition and approach

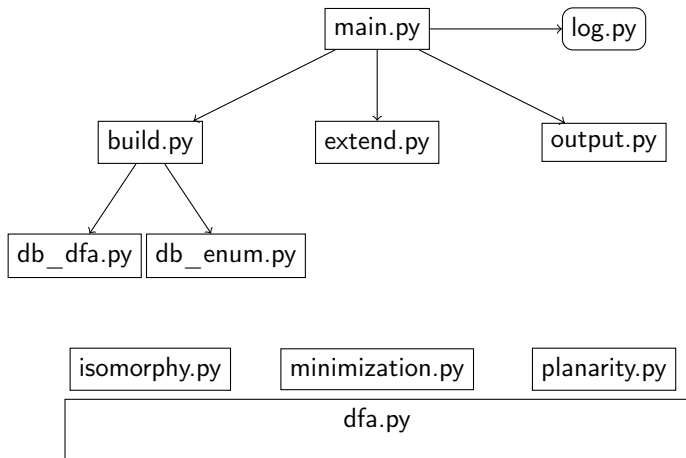
Generating Minimal DFAs

Extending Minimal DFAs

Live Demonstration and Conclusion

# Live Demonstration

## Program Architecture



This presentation has...

- ▶ introduced the problem of DFA Minimization Problem Generation
- ▶ given an overview over a possible solution
- ▶ shown that the theoretic results might be useful in praxis

Lookout:

- ▶ more parameters, ranged parameters  
The *degree* of a state  $q$  is defined as  $\deg(q) = |\text{in}(q)| + |\text{out}(q)|$ .  
 $\Rightarrow$  capping the max. degree?
- ▶ investigate planarity and drawing algorithms
- ▶ complexity analysis