Automatic Generation of DFA Minimization Problems

Gregor Sönnichsen

Universität Bayreuth

11. 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]

[1] https://www.rindlerwahn.de/zeitdiebe-besiegen-und-mehr-lebenszeit-gewinnen

Outline

Problem definition

Preliminaries (1/2)

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

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

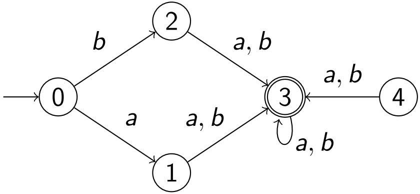
Preliminaries (2/2)

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

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

Example



Theorem

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

Problem definition

Hopcroft's Minimization Algorithm

1. Compute all unreachable states
2. Remove all unreachable states and their transitions
3. Compute all inequivalent state pairs ($\not\sim_A$)

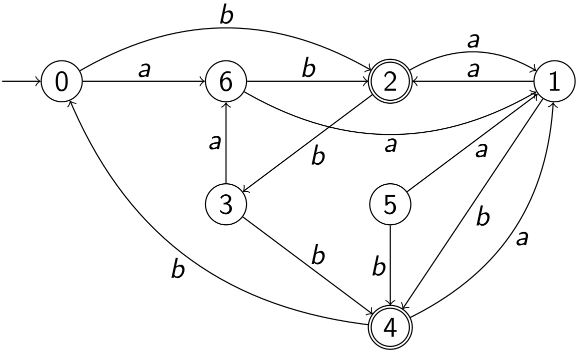
$$
\begin{array}{l}
\textbf{function } \mathrm{FindEquivPairs}(A) \\
\quad i \leftarrow 0 \\
\quad m(0) \leftarrow \{(p,q),(q,p) \mid p \in F, q \notin F\} \\
\quad \textbf{do} \\
\quad\quad i \leftarrow i+1 \\
\quad\quad m(i) \leftarrow \{(p,q),(q,p) \mid (p,q) \notin \bigcup m(\cdot) \wedge \\
\quad\quad\quad\quad\quad\quad\quad\quad \exists \sigma \in \Sigma \colon (\delta(p,\sigma),\delta(q,\sigma)) \in m(i-1)\} \\
\quad \textbf{while } m(i) \neq \emptyset \\
\quad \textbf{return } \bigcup m(\cdot)
\end{array}
$$

4. Merge all equivalent state pairs

Problem definition

A sample DFA minimization task. . .

<u>Task:</u> 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

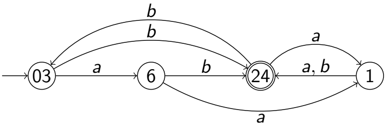
<u>*Solution:*</u>

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.
$\Rightarrow$ modular pipeline architecture

$n_s, k, n_F, d, p_{sol}, p_{task}, n_e, n_u, c$

$\downarrow$

Generate DFA Minimization Problem

- Generate Minimal DFA
  (using a rejection algorithm and randomization/enumeration)
- Extend Minimal DFA
  - Add Equivalent States
  - Add Unreachable States

$\downarrow$

A_{task}

Outline

Generating Minimal DFAs

Rejection Algorithm

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

function GenNewMinDFA (n_s, k, n_F, d, p)

  $I \leftarrow$ all DFAs in DB_{found} matching n_s, k, n_F

  while True **do**

    generate DFA A_{test} with $|Q|, |\Sigma|, |F|$ matching n_s, k, n_F

    if A_{test} not minimal **or** $d \neq \mathfrak{D}(A_{test})$ **then**

      continue

    if $p = 1$ **and** A_{test} is not planar **then**

      continue

    if A_{test} is isomorph to any DFA in I **then**

      continue

    save A_{test} and its respective properties in DB_{found}

    return A_{test}

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:

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

(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.

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

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

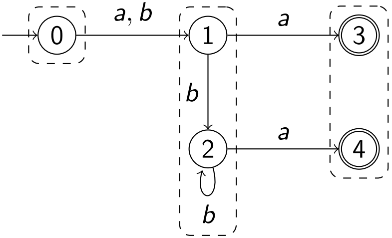
Outline

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

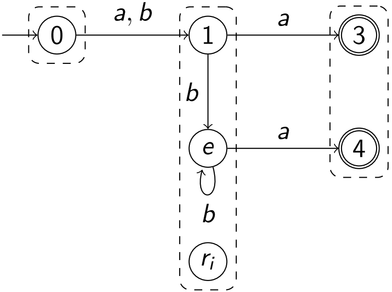


We now want to add states $r_1, \dots, r_{n_e}$ to a DFA, such that every r_i is equivalent to a state e in the original DFA:

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

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

Example



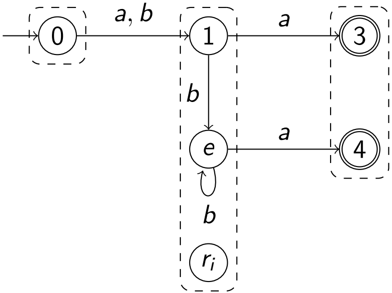
Observation:

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

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

$\Rightarrow$ $\delta(e, \sigma)$ is always defined, so $[\delta(e, \sigma)]_{\sim_A}$ is never empty

Extending Minimal DFAs

Adding Equivalent States (2) - Outgoing Transitions

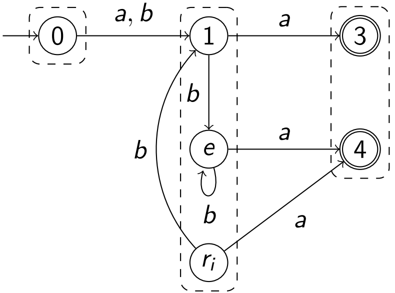
Observation:

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

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

$\Rightarrow$ $\delta(e, \sigma)$ 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, $in(r_i) >= 1$.

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 δ and add $((q, \sigma), r_i)$.

General requirement regarding the choice of a state e for an r_i:

$$duplicatable(q) \Leftrightarrow_{def} (\exists p \in [q]_{\sim_A} \colon in(p) \geq 2)$$

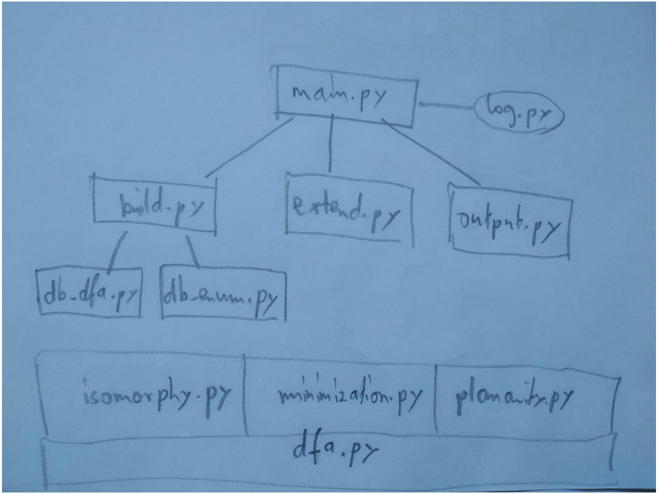
Extending Minimal DFAs

Adding Unreachable States

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

```
function AddUnrStates (A, n_u, c)
    U ← ∅
    for n_u times do
        let q be the new state
        add ingoing tr. from a random subset of U × Σ
        Σ' ← if c = 1 then Σ else random subset of Σ
        add outgoing tr. to |Σ'| random states
        add q to U
    return A
```

Outline


main.py
log.py
build.py
extend.py
output.py
db_dfa.py
db_enum.py
isomorphy.py
minimization.py
planarity.py
dfa.py

Replace

Live Demonstration

Command-Line Options

```
solution DFA:
  -k int              alphabet size of generated DFAs (default: 2)
  -n int              number of states of solution DFA (default: 4)
  -f int              number of final states of solution DFA (default: 1)
  -dmin int           lower bound for D-value (default: 2)
  -dmax int           upper bound for D-value (default: 3)
  -ps {yes,no}        toggle whether solution DFA shall be planar (default: y)
  -b {enum,random}    toggle whether solution DFA shall be build by enumeration or randomization (default: enum)

task DFA:
  -e int              number of distinct equivalent reachable state pairs in task DFA (default: 2)
  -u int              number of unreachable states in task DFA (default: 1)
  -c {yes,no}         toggle whether all unreachable states shall be complete (default: yes)
  -pt {yes,no}        toggle whether task DFA shall be planar (default: yes)

output:
  -out str            working directory; here results will be saved (default: ./output)
  -dfa {yes,no}       toggle whether DFAs shall be printed to .dfa-files (default: no)
  -tex {yes,no}       toggle whether LaTeX-code shall be created from DFAs (default: yes)
  -pdf {yes,no}       toggle whether PDFs shall be created from DFAs (default: yes)
```

Replace

Outline

Introduction

Problem definition and approach

Generating Minimal DFAs

Extending Minimal DFAs

Live Demonstration

Conclusion

Conclusion

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 $deg(q) = d^-(q) + d^+(q)$.
  $\Rightarrow$ capping the max. degree?
- investigate planarity and drawing algorithms deeper
- complexity analysis

References