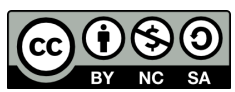
CDM

Minimization

Klaus Sutner

Carnegie Mellon University

Fall 2025



We'll take a closer look at the quotient machine from last time and in interpret it as the minimal DFA, and we'll explain why this idea is critical to the understanding of DFAs.

Then we will massage the result a little bit to extract to a perfectly reasonable quadratic algorithm to compute the minimal DFA. There are better n-$\log n$ algorithms, but those are substantially more complicated.

Definition

The state complexity of a recognizable language L is the minimum state complexity of any DFA that recognizes L.

In symbols, $\mathrm{scp}(L)$.

The definition is a bit weak, there could be several smallest DFAs that all have size $\mathrm{scp}(L)$.

Existence is trivial, but for the wrong reasons: the naturals are well-ordered.

We know how to build a DFA out of the quotients $Q = \mathcal{Q}(L)$ of some language.

Construct a DFA

$$\mathfrak{Q}_L = \langle Q, \Sigma, \delta; q_0, F \rangle$$

$$\delta(K, a) = a^{-1} K$$

$$q_0 = L$$

$$F = \{ K \in Q \mid \varepsilon \in K \}$$

The behaviors of the states of $\mathfrak{Q}_L$ are exactly all the quotients.

The reason this construction works is the following trivial observation:

$$L = \mathsf{chr}_L \cup \bigcup_{a \in \Sigma} a \cdot (a^{-1} L)$$

where $\mathsf{chr}_L = \varepsilon$ if $\varepsilon \in L$ and $\emptyset$ otherwise.

The transitions in $\mathfrak{Q}_L$ directly express this decomposition, done.

Dire Warning: This fails miserably for nondeterministic machines. Behaviors are subsets of quotients, and that's all we can say.

For deterministic machines, though, everything works out perfectly.

Lemma

Let $\mathcal{A}$ be an arbitrary DFA recognizing L, p a state and $x \in \Sigma^\star$. Then

$$[\![\delta(p, x)]\!] = x^{-1} [\![p]\!]$$

This is straightforward induction on x: $(xa)^{-1}L = a^{-1}(x^{-1}L)$

In particular

$$[\![\delta(q_0, x)]\!] = x^{-1}L$$

Consequences

Corollary

Every recognizable language has only finitely many left quotients.

Corollary

Every DFA accepting a recognizable language has at least as many states as the number of quotients of the language.

Corollary

The state complexity of a recognizable language is the number of its quotients:

$$\mathsf{scp}(L) = \#\ \textit{quotients of}\ L$$

As long as we can figure out how to compute $\mathcal{Q}(L)$ we can construct a DFA of minimum size.

Question: Is the quotient machine the only minimum size DFA?

Suppose $\mathcal{A}$ is some other DFA with size scp(L).

Then each quotient corresponds to exactly one state in $\mathcal{A}$.

By the decomposition result, it follows that there is no other way to define the transitions than in the quotient automaton.

Suppose we have two DFAs $\mathcal{A}_1$ and $\mathcal{A}_2$ and a map $f : Q_1 \to Q_2$.

We want f to map runs in $\mathcal{A}_1$ to runs in $\mathcal{A}_2$. For this to work, we need additional properties:

f must preserve transitions:

$$p \xrightarrow{a} q \qquad \text{implies} \qquad f(p) \xrightarrow{a} f(q)$$

f must preserve in inital and final states

$$f(q_{10}) = q_{20} \qquad f(F_1) = F_2$$

Definition

A map with these properties is a DFA homomorphism.

In other words, a homomorphism maps transitions to transitions:

$$\begin{array}{lcll} p & \xrightarrow{a} & \delta_1(p,a) & \mathcal{A}_1 \\ \downarrow f & & \downarrow f & \\ f(p) & \xrightarrow{a} & \delta_2(f(p),a) & \mathcal{A}_2 \end{array}$$

By chaining together boxes of this kind we can see how whole computations in $\mathcal{A}_1$ are mapped to computations in $\mathcal{A}_2$.

Lemma

If there is a homomorphism from $\mathcal{A}_1$ to $\mathcal{A}_2$, then $\mathcal{L}(\mathcal{A}_1) \subseteq \mathcal{L}(\mathcal{A}_2)$.

Dire Warning: It may still be the case that $\mathcal{L}(\mathcal{A}_1) \neq \mathcal{L}(\mathcal{A}_2)$.

To make sure that the languages are the same we need to strengthen the conditions a bit:

$$f^{-1}(F_2) = F_1$$

Definition

A covers or covering map is a surjective homomorphism such that $f^{-1}(F_2) = F_1$.

Needless to say, the classical example of a cover is the behavioral map:

$$f : Q \to \mathcal{Q}(L)$$
$$f(p) = [\![p]\!]$$

Hence we have the following lemma which shows that an arbitrary DFA for a given recognizable language is always an "inflated" version of the minimal DFA. There always is a close connection between an arbitrary DFA and the minimal automaton.

Lemma

Let L be a recognizable language and $\mathcal{A}$ an arbitrary accessible DFA for L. Then there is covering map from $\mathcal{A}$ onto $\mathfrak{Q}_L$.

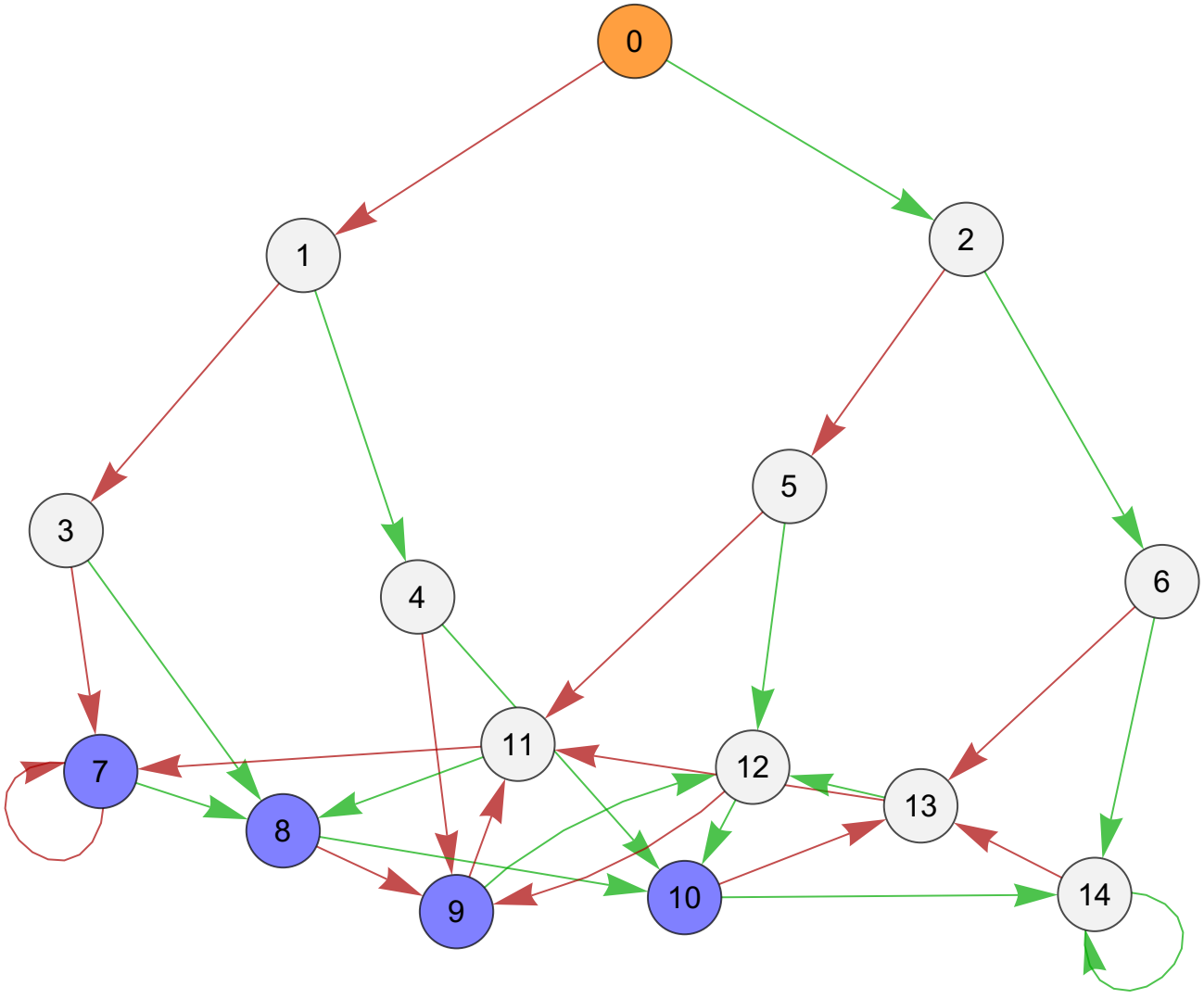
Example

Let's return to our old example $\mathsf{Pos}_{a,-3} = \Sigma^\star a \Sigma \Sigma$.

Perhaps the most natural way to construct a DFA for the language is to start with ε and then remember letters until we get to 3. From then on, we drop and add one letter.

$$\delta(w, s) = \begin{cases} ws & \text{if } |w| < 3, \\ w_2 w_3 s & \text{otherwise.} \end{cases}$$

The initial state is ε and the final states are $\{aaa, aab, aba, abb\}$.


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14

We already know that the states Σ^3 suffices, we only need the de Bruijn graph but not the tree (the initial state moves to bbb). Here is the corresponding cover:

$$\begin{array}{llll} aaa & \mapsto & aaa & aa, baa & \mapsto & baa \\ aab & \mapsto & aab & ab, bab & \mapsto & bab \\ aba & \mapsto & aba & a, ba, bba & \mapsto & bba \\ abb & \mapsto & abb & \varepsilon, b, bb, bbb & \mapsto & bbb \end{array}$$

Let us call a DFA reduced iff no two states have the same behavior.

Theorem

A DFA for a recognizable language is minimal with respect to the number of states if, and only if, it is accessible and reduced.

Moreover, up to isomorphism, there is only one such minimal DFA: the quotient automaton of the language.

All DFAs recognizing L cover the minimal automaton.

From the last section, we can directly derive a minimization algorithm.

Suppose we have some accessible DFA $\mathcal{A}$ for a language L.

We know that the behavioral map $p \mapsto [\![p]\!]_{\mathcal{A}}$ is a covering map onto the minimal DFA for L. So we compute behavioral equivalence.

Then we do state merging: We collapse all the blocks into single states (and inherit tranistions, initial state and final states). This produces a DFA by the definition of behavioral equivalence.

That's it!

We could compute behavioral equivalence by brute force and perform a quadratic test on quadratically many pairs of states, leading to a $O(n^4)$ algorithm.

A moment's thought shows that we can save some work.

> Compute the full product automaton $\mathcal{A}^2 = \mathcal{A} \times \mathcal{A}$.
>
> Determine all pairs $(p, q) \in Q \times Q$ that can only reach points in
>
> $$F \times F \cup \overline{F} \times \overline{F}$$

These pairs represent behavioral equivalence, done.
This method is strictly quadratic, in time as well as space.

Not bad, but not great either.

We will use partitions alongside equivalence relations.

Equivalence Relations

A relation $\rho \subseteq A \times A$ that is reflexive, symmetric and transitive.

Partition

A collection B_1, B_2, ..., B_k of pairwise disjoint, non-empty subsets of A such that $\bigcup B_i = A$ (the blocks of the partition).

k is the index of ρ.

Definition

Given a map $f : A \to B$ the kernel relation induced by f is the relation

$$x \; K_f \; y \iff f(x) = f(y).$$

Clearly, K_f is an equivalence relation for any f.

Every equivalence relation is a kernel relation: just let $f(x) = [x]$. The codomain here is $\mathfrak{P}(A)$ which is not attractive computationally.

But, we can use a function $f : A \to A$: just choose a fixed representative in each class $[x]$. This requires the axiom of choice in general, but we don't need big guns.

We may safely assume that $A = [n]$, so we can store f as a simple array: this requires only $O(n)$ space and equivalence testing is $O(1)$ with very small constants.

Definition

The canonical choice function (CCF) for an equivalence relation R on A is

$$\mathsf{can}_R(x) = \min\big(\, z \in A \mid x \,\rho\, z \,\big) = \min [x]$$

So each equivalence class is represented by its least element.

To test whether $a, b \in A$ are equivalent we only have to compute $f(a)$ and $f(b)$ and test for equality. If the values of f are stored in an array this is $O(1)$, with very small constants.

Here are some basic ideas involving equivalence relations.

Definition

Let ρ and σ be two equivalence relations on A.
ρ is finer than σ (or: σ is coarser than ρ), if $x\,\rho\,y$ implies $x\,\sigma\,y$.

In symbols $\rho \sqsubseteq \sigma$.

Hence every block of ρ is included in a block of σ (does not cut across boundaries) and the index of σ is at most the index of ρ.

To avoid linguistic dislocations, we mean this to include the case where ρ and σ are the same (we will say that ρ is strictly finer otherwise).

If we think of equivalence relations as sets of pairs then

$$\rho \sqsubseteq \sigma \iff \rho \subseteq \sigma.$$

We need algorithms to manipulate equivalence relations.

Definition (Meet of Equivalence Relations)

Let ρ and σ be two equivalence relations on A.
$\rho \sqcap \sigma$ denotes the coarsest equivalence relation finer than both ρ and σ.

This is just logical *and*:

$$x \, (\rho \sqcap \sigma) \, y \iff x \, \rho \, y \wedge x \, \sigma \, y$$

Meet is sometimes written $\rho \cap \sigma$; which is fine if we think of the relations as sets of pairs, but a bit misleading otherwise.

The dual notion of meet is join.

Definition (Join of Equivalence Relations)

Let ρ and σ be two equivalence relations on A.
$\rho \sqcup \sigma$ denotes the finest equivalence relation coarser than both ρ and σ.

Note that $\rho \sqcup \sigma$ is required to be an equivalence relation, so we cannot in general expect $\rho \sqcup \sigma = \rho \cup \sigma$ in the sets-of-pairs model: the union typically fails to be transitive. Hence, we have to take the transitive closure:

$$\rho \sqcup \sigma = \mathsf{tcl}(\rho \cup \sigma)$$

Here is how to compute the meet $\tau = \rho \sqcap \sigma$ of two equivalence relations:

$$p \; \tau \; q \iff \mathsf{can}_\rho(p) = \mathsf{can}_\rho(q) \wedge \mathsf{can}_\sigma(p) = \mathsf{can}_\sigma(q)$$

We may safely assume that the carrier set is $A = [n]$ and that all the relations are represented by their CCFs:

$$\rho \rightsquigarrow \mathsf{can}_\rho = (r_1, r_2, \ldots, r_n)$$

But then we are really looking for identical pairs in the table

1	2	3	$\ldots$	p	$\ldots$	n
r_1	r_2	r_3	$\ldots$	r_p	$\ldots$	r_n
s_1	s_2	s_3	$\ldots$	s_p	$\ldots$	s_n

```
// meet τ of ρ and σ
for p = 1, ..., n do
    i = can_ρ(p)                    // table lookup
    j = can_σ(p)                    // table lookup
    if (i, j) is new
        t_p = seen(i, j) = p        // hash table
    else
        t_p = seen(i, j)
return (t_1, ..., t_n)
```

A hash table is the canonical choice, but any fast container type will do.

Example

	1	2	3	4	5	6	7	8
ρ	1	1	1	1	5	5	1	5
σ	1	2	2	2	1	1	1	2
τ	1	2	2	2	5	5	1	8

Problem: Characterize canonical choice functions.

Proposition

Using array representations, we can compute the meet of two equivalence relations in (expected) linear time.

The "expected" hedge is just for the hash table, not the logical structure of the algorithm. In practice it can be ignored.

Exercise

Show how to implement the algorithm in linear time (not just expected) using a quadratic precomputation.

This method goes back to a paper in famous early volume on automata theory.

> E. F. Moore
> Gedanken-Experiments on Sequential Machines
> Automata Studies, C. Shannon, J. McCarthy's *eds.*, 1956

Moore's approach is iterative:

> Start with the partition $F, \overline{F}$,
>
> keep refining the partition,
>
> stop when behavioral equivalence is reached.

Definition

Let $f : A \to A$ be an endofunction and $\mathcal{F}$ a family of such functions.

An equivalence relation ρ on A is f-compatible if $x \,\rho\, y$ implies $f(x) \,\rho\, f(y)$.

ρ is $\mathcal{F}$-compatible if it is f-compatible for all $f \in \mathcal{F}$.

Let ρ be some equivalence relation and write $\rho^{\mathcal{F}}$ for the coarsest refinement of ρ that is $\mathcal{F}$-compatible. From the definitions

$$\rho^{\mathcal{F}} = \bigsqcup \{\, \sigma \sqsubseteq \rho \mid \sigma \,\mathcal{F}\text{-compatible} \,\}$$

Alas, this set-theoretic description is useless from a computational perspective.

Fix some DFA $\mathcal{A}$ and consider the state transition maps $\mathcal{F} = \{\, \delta_a : Q \to Q \mid a \in \Sigma \,\}$.

We compute a sequence of finer and finer partitions starting with $F, \overline{F}$, maintaining the invariant

> $B \neq B'$ blocks, $p \in B$, $q \in B'$ implies
> p and q have different behavior

by making the partitions more and more compatible with $\mathcal{F}$.

We stop when compatibility is reached, obtaining the coarsest compatible partition.

To compute $\rho^{\mathcal{F}}$ first define for any $f \in \mathcal{F}$ and any equivalence relation σ:

$$p \,\sigma_f\, q \Leftrightarrow f(p)\,\sigma\, f(p)$$

$$\mathsf{ref}_f(\sigma) = \sigma \sqcap \sigma_f$$

It is easy to see that $\mathsf{ref}_f(\sigma)$ is indeed an equivalence relation and is a refinement of σ. The following lemma shows that we cannot make a mistake by applying these refinement operations.

Lemma

Suppose $\rho^{\mathcal{F}} \sqsubseteq \sigma \sqsubseteq \rho$. *Then*

$\rho^{\mathcal{F}} \sqsubseteq \mathsf{ref}_f(\sigma) \sqsubseteq \sigma$ *for all* $f \in \mathcal{F}$.

If σ *not* $\mathcal{F}$*-compatible, then* $\mathsf{ref}_f(\sigma) \sqsubsetneq \sigma$ *for some* $f \in \mathcal{F}$.

Proof

Let $\tau \sqsubseteq \rho$ be $\mathcal{F}$-compatible and assume $x\ \tau\ y$. By assumption, $\tau \sqsubseteq \sigma$. By compatibility, $f(x)\ \tau\ f(y)$, whence $f(x)\ \sigma\ f(y)$. But then $x\ \mathrm{ref}_f(\sigma)\ y$.

Since σ fails to be $\mathcal{F}$-compatible there must be some $f \in \mathcal{F}$ such that $x\ \sigma\ y$ but not $f(x)\ \sigma\ f(y)$. Hence $\mathrm{ref}_f(\sigma) \neq \sigma$.

□

Since our carrier sets are finite, the lemma guarantees that we can just apply the operations ref_f repeatedly until we get down to $\rho^{\mathcal{F}}$. The exact order of refinement steps does not matter (at least logically, efficiency is another matter).

Once we have behavioral equivalence relation E, we can determine the quotient structure: we replace Q by Q/E, and q_0 and F by the corresponding equivalence classes.

Define

$$\delta'([p]_E, a) = [\delta(p, a)]_E$$

Proposition

This produces a new DFA that is equivalent to the old one, and reduced.

Each refinement step is $O(n)$, so a big step is $O(kn)$ where $k = |\Sigma|$.

Total running time is $O(knr)$ where r is the number of refinement rounds. In many cases r is quite small, but one can force $r = n - 2$.

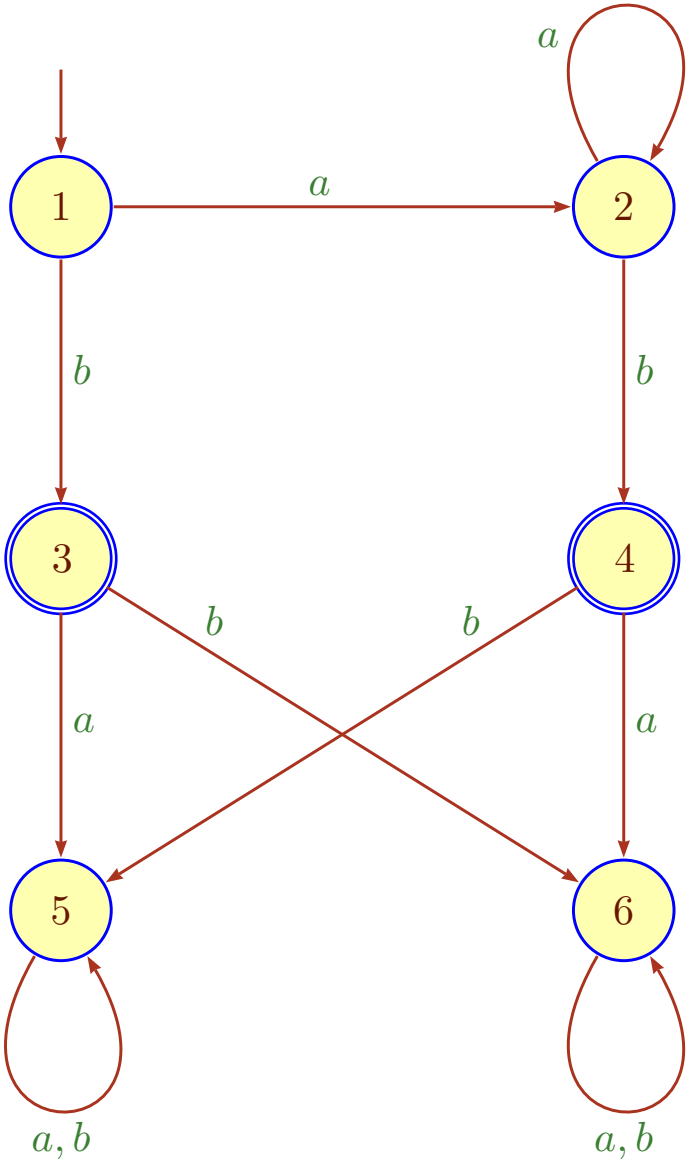
Lemma

Moore's minimization algorithm runs in (expected) time $O(kn^2)$.

Expected because of the hash table, not because of something inherently probabilistic about the algorithm.

Merging Example

The 6-state DFA for $a^\star b$.



The last DFA has transition matrix

	1	2	3	4	5	6
a	2	2	5	6	5	6
b	3	4	6	5	5	6

and final states $\{3, 4\}$.

Hence, the original approximation E_0 and the two child relations E_{0,δ_a} and E_{0,δ_b} look like so:

	1	2	3	4	5	6
E_0	1	1	3	3	1	1
E_{0,δ_a}	1	1	1	1	1	1
E_{0,δ_b}	3	3	1	1	1	1

We perform a double table lookup to get the kids: first in the transition matrix and then in the CCF for E_0.

	1	2	3	4	5	6
E_0	1	1	3	3	1	1
a	1	1	1	1	1	1
b	3	3	1	1	1	1
E_1	1	1	3	3	5	5
a	1	1	5	5	5	5
b	3	3	5	5	5	5
E_2	1	1	3	3	5	5

Hence $E_2 = E_1$ and the algorithm terminates.

Merged states are $\{1, 2\}$, $\{3, 4\}$, $\{5, 6\}$.

To save space, we have performed giant refinement steps using

$$\mathsf{ref}(\rho) = \bigsqcap_{f \in \mathcal{F}} \mathsf{ref}_f(\rho)$$

Another Example

Consider the DFA with final states $\{1, 4\}$ and transition table

	1	2	3	4	5	6	7	8
a	2	4	5	2	6	8	4	6
b	3	5	4	3	7	4	8	7

produces the trace:

	1	2	3	4	5	6	7	8
E_0	1	2	2	1	2	2	2	2
a	2	1	2	2	2	2	1	2
b	2	2	1	2	2	1	2	2
E_1	1	2	3	1	5	3	2	5
a	2	1	5	2	3	5	1	3
b	3	5	1	3	2	1	5	2
E_2	1	2	3	1	5	3	2	5

The last minimization method may be the most canonical, but there are others. Noteworthy is in particular a method by Brzozowski that uses reversal and Rabin-Scott determinization to construct the minimal automaton.

Write

rev($\mathcal{A}$) for the reversal of any finite state machine, and

pow($\mathcal{A}$) for the accessible part obtained by determinization.

Thus pow preserves the acceptance language but rev reverses it.

Lemma

If $\mathcal{A}$ is an accessible DFA, then $\mathcal{A}' = \mathsf{pow}(\mathsf{rev}(\mathcal{A}))$ is reduced.

Proof.

Let $\mathcal{A} = \langle Q, \Sigma, \delta; q_0, F\rangle$.

$\mathcal{A}'$ is accessible by construction, so we only need to show that any two states have different behavior.

Let $P = \delta_x^{-1}(F) \neq P' = \delta_y^{-1}(F)$ in $\mathcal{A}'$ for some $x, y \in \Sigma^\star$.

We may safely assume that $p \in P - P'$.

Since $\mathcal{A}$ is accessible, there is a word z such that $p = \delta_z(q_0)$.

Since $\mathcal{A}$ is deterministic, z^{op} is in the $\mathcal{A}'$-behavior of P but not of P'.

□

On some (but rare) occasions the last lemma can be used to determine that an automaton is minimal.

For example, if $\mathcal{A}$ is the canonical NFA for the language "kth symbol from the end is a", then rev(pow(rev($\mathcal{A}$))) is $\mathcal{A}$ plus a sink. Hence pow($\mathcal{A}$) must be the minimal automaton.

Similarly, let $\mathcal{A}$ be the DFA for all words over $\mathbf{2}$ whose numerical values are congruent 0 modulo some prime p. Then rev($\mathcal{A}$) is again an accessible DFA and pow(rev(pow(rev($\mathcal{A}$)))) is isomorphic to $\mathcal{A}$.

More generally, we can use the lemma to establish the following surprising minimization algorithm.

Theorem (Brzozowski 1963)

Let $\mathcal{A}$ be a finite state machine. Then the automaton $\mathsf{pow}(\mathsf{rev}(\mathsf{pow}(\mathsf{rev}(\mathcal{A}))))$ *is (isomorphic to) the minimal automaton of $\mathcal{A}$.*

Proof.

$\widehat{\mathcal{A}} = \mathsf{pow}(\mathsf{rev}(\mathcal{A}))$ is an accessible DFA accepting $\mathcal{L}(\mathcal{A})^{\mathrm{op}}$.

By the lemma, $\mathcal{A}' = \mathsf{pow}(\mathsf{rev}(\widehat{\mathcal{A}}))$ is the minimal automaton accepting $\mathcal{L}(\mathcal{A})^{\mathrm{op}\,\mathrm{op}} = \mathcal{L}(\mathcal{A})$.

□

Which is Better?

One might ask whether Moore or Brzozowski is better in the real world. Somewhat surprisingly, given a good implementation of Rabin-Scott determinization, there are some examples where Brzozowski's method is faster.

Theorem (David 2012)

Moore's algorithm has expected running time $O(n \log n)$.

Theorem (Felice, Nicaud, 2013)

Brzozowski's algorithm has exponential expected running time.

These results assume a uniform distribution, it is not clear whether this properly represents "typical" inputs in any practical sense (say, for pattern matching algorithms).

Mathematical Thinking: Behavioral Equivalence. Once the concept of behavior is clear, there is a straightforward algorithm for minimization. With a little effort, it's even quadratic.

Algorithmic Thinking: Iterative Refinement. We get a better algorithm by adjusting the logic and using a good representation of equivalence relations. Produces a clean, worst case quadratic algorithm.

Smart Algo Thinking: Controlling Refinement. To get a sub-quadratic algorithm the logic of refinement needs to be improved. Some creativity and insight is needed to get down to log-linear.

Total Recall

Recall: we have an equivalence relation $\rho \subseteq Q \times Q$ and a map $f : Q \to Q$.

We want to find the coarsest refinement $\widehat{\rho}$ of ρ that is compatible with f:

$$p \,\widehat{\rho}\, q \quad \Rightarrow \quad f(p) \,\widehat{\rho}\, f(q)$$

This is accomplished by repeated application of a refinement operator ref_f:

$$p \,\rho_f\, q \Leftrightarrow f(p) \,\rho\, f(q)$$

$$\mathsf{ref}_f(\rho) = \rho \sqcap \rho_f$$

In other words: $\widehat{\rho}$ is the fixed point of ρ under ref_f.

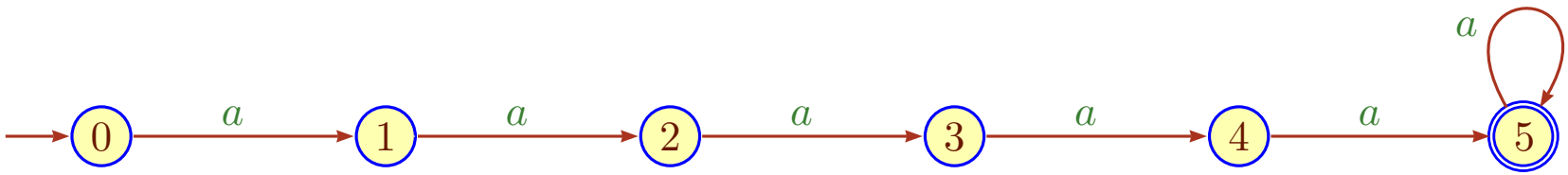
The refinement operator in Moore's algorithm works by representing all relations as canonical selector functions (aka int arrays), each round requires a scan of the whole array.

So each refinement step is $\Theta(n)$—with good constants but still linear in n.

The good news is that, quite often, the algorithm uses fewer than n rounds, so the total time complexity may well be sub-quadratic.

Alas, there are cases when Moore requires $\Theta(n)$ rounds, producing quadratic running time overall.

Here is the standard example that demonstrates that Moore's algorithm may be quadratic: the minimal DFA for $\{a\}^{\geq k}$.



For this automaton, a single Moore round will split off only one state from the right end of block $D = \{0, 1, \ldots, r\}$, at a cost of $\Theta(n)$ steps.

The split occurs only because of some block $B = \{p\}$, nothing else matters.

Critical Idea:

Maybe we could get mileage out of trying to guide the refinement by single blocks, instead of blindly hitting the whole carrier set.

We only want to touch the the blocks in the partition that are currently "relevant," not just blindly every state in the machine.

Consider two blocks C and B in partition ρ.

We say that C splits B (wrto f) iff

$$B \cap f^{-1}(C) \neq \emptyset \quad \text{and} \quad B - f^{-1}(C) \neq \emptyset.$$

In other words, $f(B)$ intersects both C and $Q - C$ and is not f-compatible yet: stopping the refinement process at this point would produce a nondeterministic machine. We need further refinement.

Define a more fine-grained refinement operator $\rho' = \mathsf{ref}_f(\rho, B, C)$ as follows:

$$p \; \rho' \; q \iff (p, q \notin B \wedge p \; \rho \; q) \vee$$
$$\Big(p, q \in B \wedge (f(p) \in C \Leftrightarrow f(q) \in C)\Big)$$

In other words: outside of block B we keep the old ρ.
Inside of B we check for C-equivalence of children.

So $\mathsf{ref}_f(\rho, B, C)$ is a indeed a refinement of ρ: block B is potentially split in two (or may be unchanged).

Proposition

$\mathsf{ref}_f(\rho) \sqsubseteq \mathsf{ref}_f(\rho, B, C)$.

$\mathsf{ref}_f(\rho) \neq \rho$ *implies that* $\mathsf{ref}_f(\rho, B, C) \neq \rho$ *for some* B *and* C.

In other words, we make no mistakes and we can't get stuck.

Proof.

$\mathsf{ref}_f(\rho)$ is $\bigsqcap_{C,B} \mathsf{ref}_f(\rho, B, C)$ and thus finer than each part.

If $\mathsf{ref}_f(\rho) \neq \rho$ there must be some block B and $p, q \in B$ such that $\neg(f(p)\ \rho\ f(q))$.

Let C be the block containing $f(p)$, done. □

Really?

One might worry that the fine-grained approach is arguably more elegant, but the extra bureaucracy needed to make it work might just overwhelm the gains: we spend more work on finding out how to avoid extra work.

It takes a bit of effort to design the right datastructures, but if everything is done properly, we do get a significant speedup.

Suppose we have $\Sigma = \{a\}$ and we want to insure compatibility with δ_a (see below for the general case). In addition, we have an initial partition $(F, Q - F)$. The algorithm maintains two data structures, both are initialized by the given partition.

- a partition P of Q, representing the equivalence relation,
- a split list S with entries some of the blocks in the partition.

We refer to the blocks C in S as active: those are the blocks that we will use at some point to try to refine P by $a^{-1}C = \delta_a^{-1}(C)$.

The algorithm extracts an active block from the split list and tries to refine the blocks in the partition accordingly.

It then updates the split list in a clever way, and stops when the split list becomes empty.

Initializing P is straightforward, just the given partition.

The split list S only gets the smaller of the two blocks (we are dealing with the single-function case here, see below on how to handle larger alphabets).

Incidentally, for some application one needs to allow initial partitions with more than 2 blocks; everything we do here easily generalizes.

Hopcroft's Algorithm

initialize partition P and split list S

while S not empty **do**
    extract C from S
    compute $\widehat{C} = a^{-1}C$
    foreach block B split by C **do**
        $B^+ = B \cap \widehat{C}$
        $B^- = B - B^+$
        replace B by B^+ and B^- in P // update partition
        if B is in S // update split list
        then replace B by B^+, B^- in S
        else place the smaller of B^+, B^- into S
end

Theorem (Hopcroft 1971)

Hopcroft's algorithm minimizes a DFA in $O(kn \log n)$ steps, where n is the state complexity of the DFA and k the size of the alphabet.

J. Hopcroft

A N log N Algorithm for Minimizing States in a Finite Automaton

STAN-CS-71-190

This is a seminal paper that will bring tears to your eyes.

The algorithm is quite messy to implement correctly, as can be seen from the following papers:

D. Gries
Describing an algorithm by Hopcroft
Acta Informatica, 2 (1973) 97–109.

T. Knuutila
Re-describing an algorithm by Hopcroft
Theoretical Computer Science, 250 (2001) 333–363.

Exercise

Implement Hopcroft's algorithm, correctly.

Note that Hopcroft's algorithm is nondeterministic in several ways.

- We can extract any element from the split list (e.g., could use a stack, queue, random choice, ...).

- Likewise we can place the new entries anywhere in the split list.

- When B^+ and B^- have the same size, we can pick either one.

None of these choices effect correctness, but they may well influence running time. As a consequence, any detailed analysis taking into account possible strategies is quite complicated.

It should be noted that the algorithm often takes far fewer than $n \log n$ steps.

Given a reasonable implementation, every round is linear. It turns out to be quite difficult to construct inputs where the algorithm requires $\log n$ many rounds.

The best result known today is that there are some DFAs such that the algorithm takes $n \log n$ steps for a certain choice of active blocks in the main loop.

Alas, for these machines a different choice of active blocks results in linear running time.

When is the running time $\Omega(n \log n)$ regardless of the chosen split list protocol? (A unary example is known where the execution sequence is essentially unique and reaches the log-lin bound.)

What is the average complexity of Hopcroft's algorithm (average with respect to input automaton and/or split list protocol)?

Is Hopcroft faster than Moore on average? For the uniform distribution, Moore has expected behavior $O(n \log n)$ and it may be that the constants are smaller (Bassino, David, Nicaud 2009).