

Solution: Forward State Merging

Part A: State Merging

Here are the three stages in the forward state merging algorithm.

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

Hence we obtain a machine with 6 states and transition matrix

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

1 is initial and {3, 5} are final.

Part B: Language

It is easy to check that (ignoring the sink) the diagram of this machine is acyclic and hence the language is finite: {aa, ba, aaa, abb, baa, bbb}

Part C: Quotients

From part B. we can construct the minimal DFA M_1 by computing all the quotients. The table shows x and $x^{-1}L$.

ε	{aa, ba, aaa, abb, baa, bbb}
a	{a, aa, bb}
aa	{ ε , a}
bb	{b}
b	{ ε }
aaa	\emptyset

Part D: Isomorphism

For the quotient machine as constructed above the isomorphism is

1	$\{aa, ba, aaa, abb, baa, bbb\}$
2	$\{a, aa, bb\}$
3	$\{\varepsilon, a\}$
4	$\{b\}$
5	$\{\varepsilon\}$
6	\emptyset

Solution: More Forward State Merging

Part A: Example Machines

The perhaps easiest solution is to have a machine on n states $\{1, 2, \dots, n\}$ where 1 is initial, n is final, $\Sigma = \{a\}$ and the transitions are

$$\delta(p, a) = \begin{cases} p + 1 & \text{if } p < n, \\ n & \text{otherwise.} \end{cases}$$

It is easy to see that after k steps the partition produced by the algorithm has the form

$$\{1, 2, \dots, n - k - 1\}, \{n - k\}, \{n - k + 1\}, \dots, \{n\}$$

Since the machine is already minimal the process continues for $n - 2$ steps till only singletons are left.

Part B: Table vs. Hash

Initialize an n by n array P to 0 (we assume 1-indexing here). If we are given two vectors T_1 and T_2 representing the canonical selector function for the relations we can compute the canonical selector for their meet as follows:

```

for  $i = 1$  to  $n$  do
  if  $P(T_1(i), T_2(i)) == 0$ 
  then  $P(T_1(i), T_2(i)) = T(i) = i;$ 
  else  $T(i) = P(T_1(i), T_2(i));$ 
for  $i = 1$  to  $n$  do
   $P(T_1(i), T_2(i)) = 0;$ 

```

Note that the table P is reset after each round, so it can be used in the next round. Running time is clearly $\Theta(n^2)$ because of the initialization of P . Note, though, that there are tricks to avoid initialization, so one can sometimes achieve sub-quadratic running time even with this approach.

Solution: Primitivity and Minimality

Part A: Minimality

The diagram of $M(u)$ is just a cycle $\{0, 1, \dots, n\}$. Think of $u \in \mathbf{2}^n$ as a circular word and write ρ for the operation of rotating this word by one place counterclockwise. In other words, $\rho(u) = u_1 \dots u_{n-1}u_0$. But then the behavior of state p , $0 \leq p < n$, is just $\mathcal{L}(M(\rho^p(u)))$. Since primitivity is invariant under rotation there are two distinct states with the same behavior if, and only if, there is some $0 \leq p < n$ such that $L = \mathcal{L}(M(u)) = \mathcal{L}(M(\rho^p(u)))$.

Assume such a p exists. Then $a^i \in \mathcal{L}(M(\rho^p(u)))$ if, and only if, $(\rho^p(u))_{i \bmod n} = 1$, for all $i \geq 0$. It follows that $u = \rho^p(u)$. But then $u = xy = yx$ where x is the prefix of length p and u is not primitive.

The opposite direction is entirely similar.

Part B: Algorithmic Argument

The standard forward state-merging algorithm originally distinguishes only p such that $u_p = 0$ and q such that $u_q = 1$. After one round two states p and q are equivalent if $u_p u_{p+1} = u_q u_{q+1}$ (where indices are supposed to be taken mod n), and so forth. If p and q are still equivalent after k rounds then at least $u_p u_{p+1} \dots u_{p+k-1} = u_q u_{q+1} \dots u_{q+k-1}$.

Now, if $u = v^k$ the algorithm cannot distinguish between states p and $p + k$ and will produce a machine of at most $|v| = n/k$ states.

On the other hand, if u is primitive, then for every p and q there exists some i such that $u_{p+k} \neq u_{q+k}$ and the algorithm will place u_p and u_q into separate classes after at most k rounds.

Part C: Counting

Any DFA over a one-letter alphabet is characterized by $(t, p; u, v)$ where $t \geq 0$ is the transient part, p the period and $u \in \mathbf{2}^t, v \in \mathbf{2}^p$ are bit-vectors indicating the final states. The automaton is minimal iff v is primitive and $u_{-1} \neq v_{-1}$. Hence the number of minimal DFAs on n states is

$$C(n) = \sum_{t+p=n} \pi(p) 2^{\max(t-1, 0)}$$

where $\pi(p) = \sum_{d|n} \mu(n/d)2^d$ denotes the number of primitive words of length p (μ is the Möbius function). This is not particularly pretty, but it suffices to compute a few values:

n	1	2	3	4	5	6	7	8	9	10	11	12
$C(n)$	2	4	12	30	78	180	432	978	2220	4926	10908	23790

It is not hard to see that C grows exponentially.