Constructions on Finite Automata

Informatics 2A: Lecture 4

John Longley

School of Informatics
University of Edinburgh
jrl@inf.ed.ac.uk

25 September 2017

Recap of Lecture 3

- A language is a set of strings over an alphabet Σ.
- A language is called regular if it is recognised by some NFA.
- DFAs are an important subclass of NFAs.
- An NFA with n states can be determinized to an equivalent DFA with 2^n states, using the subset construction.
- Therefore the regular languages are exactly the languages recognised by DFAs.

Union of regular languages

Consider the following little theorem:

If L_1 and L_2 are regular languages over Σ, so is $L_1 \cup L_2$.

This is dead easy to prove using NFAs.

Suppose $N_1 = (Q_1, \Delta_1, S_1, F_1)$ is an NFA for L_1, and $N_2 = (Q_2, \Delta_2, S_2, F_2)$ is an NFA for L_2.

We may assume $Q_1 \cap Q_2 = \emptyset$ (just relabel states if not).

Now consider the NFA

$$(Q_1 \cup Q_2,\ \Delta_1 \cup \Delta_2,\ S_1 \cup S_2,\ F_1 \cup F_2)$$

This is just N_1 and N_2 'side by side'. Clearly, this NFA recognizes precisely $L_1 \cup L_2$.

Number of states $= |Q_1| + |Q_2|$ — no state explosion!

Intersection of regular languages

If L_1 and L_2 are regular languages over Σ, so is $L_1 \cap L_2$.

Suppose $N_1 = (Q_1, \Delta_1, S_1, F_1)$ is an NFA for L_1, and $N_2 = (Q_2, \Delta_2, S_2, F_2)$ is an NFA for L_2.

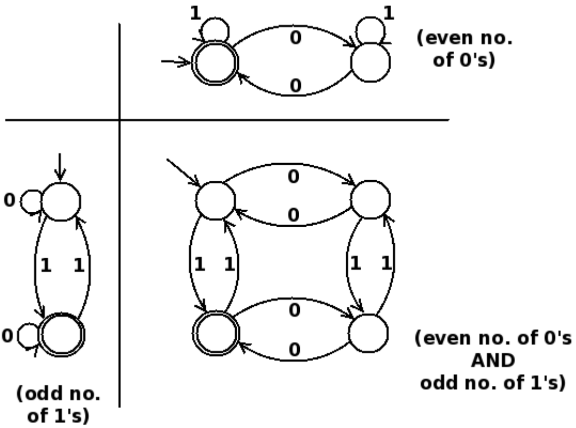
We define a product NFA $(Q',\ \Delta',\ S',\ F')$ by:

$$\begin{array}{rcl} Q' & = & Q_1 \times Q_2 \\ (q,r) \xrightarrow{a} (q',r') \in \Delta' & \iff & q \xrightarrow{a} q' \in \Delta_1 \text{ and } r \xrightarrow{a} r' \in \Delta_2 \\ S' & = & S_1 \times S_2 \\ F' & = & F_1 \times F_2 \end{array}$$

Number of states $= |Q_1| \times |Q_2|$ — a bit more costly than union!

If N_1 and N_2 are DFAs then the product automaton is a DFA too.

Example of language intersection



Complement of a regular language

(Recall the set-difference operation,

$$A - B \ = \ \{x \in A \mid x \notin B\}$$

where A, B are sets.)

If L is a regular language over Σ, then so is $\Sigma^* - L$.

Suppose $N = (Q, \delta, s, F)$ is a DFA for L.

Then $(Q, \delta, s, Q - F)$ is a DFA for $\Sigma^* - L$. (We simply swap the accepting and rejecting states in N.)

Number of states $= |Q|$ — no blow up at all, but we are required to start with a DFA. This in itself has size implications.

The complement construction does not work if N is not deterministic!

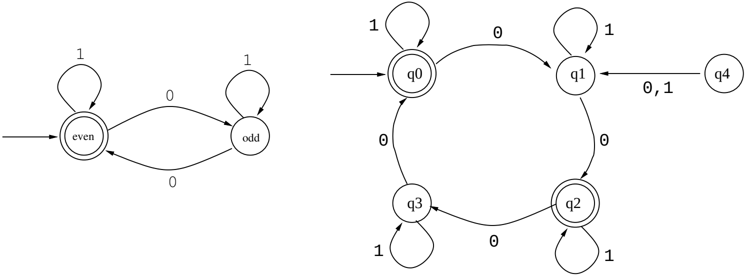
Closure properties of regular languages

- We've seen that if both L_1 and L_2 are regular languages, then so are:
  - $L_1 \cup L_2$ (union)
  - $L_1 \cap L_2$ (intersection)
  - $\Sigma^* - L_1$ (complement)
- We sometimes express this by saying that regular languages are closed under the operations of union, intersection and complementation. ('Closed' used here in the sense of 'self-contained'.)
- Each closure property corresponds to an explicit construction on finite automata. Sometimes this uses NFAs (union), sometimes DFAs (complement), and sometimes the construction works equally well for both NFAs and DFAs (intersection).

The Minimization Problem

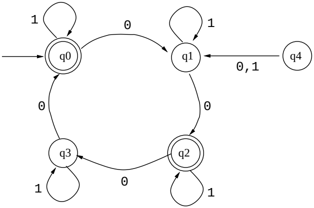
Determinization involves an exponential blow-up in the automaton.
Is it sometimes possible to reduce the size of the resulting DFA?

Many different DFAs can give rise to the same language, e.g.:



We shall see that there is always a unique smallest DFA for a given regular language.

DFA minimization



We perform the following steps to 'reduce' M above:

- Throw away unreachable states (in this case, q4).
- Squish together equivalent states, i.e. states q, q' such that: every string accepted starting from q is accepted starting from q', and vice versa. (In this case, q0 and q2 are equivalent, as are q1 and q3.)

Let's write Min(M) for the resulting reduced DFA. In this case, Min(M) is essentially the two-state machine on the previous slide.

Properties of minimization

The minimization operation on DFAs enjoys the following properties which characterise the construction:

- $\mathcal{L}(\mathrm{Min}(M)) = \mathcal{L}(M)$.
- If $\mathcal{L}(M') = \mathcal{L}(M)$ and $|M'| \leq |\mathrm{Min}(M)|$ then $M' \cong \mathrm{Min}(M)$.

Here $|M|$ is the number of states of the DFA M, and $\cong$ means the two DFAs are isomorphic: that is, identical apart from a possible renaming of states.

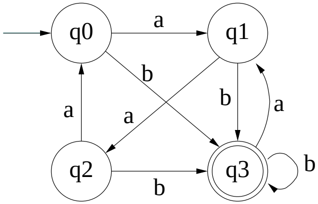
Two consequences of the above are:

- $\mathrm{Min}(M) \cong \mathrm{Min}(M')$ if and only if $\mathcal{L}(M) = \mathcal{L}(M')$.
- $\mathrm{Min}(\mathrm{Min}(M)) \cong \mathrm{Min}(M)$.

For a formal treatment of minimization, see Kozen chapters 13–16.

Challenge question

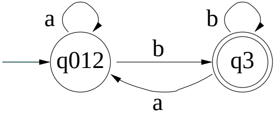
Consider the following DFA over $\{a, b\}$.



How many states does the minimized DFA have?

Solution

The minimized DFA has just 2 states:



The minimized DFA has been obtained by squishing together states q0, q1 and q2. Clearly q3 must be kept distinct.

Note that the corresponding language consists of all strings ending with b.

Minimization in practice

Let's look again at our definition of equivalent states:

> *states q, q' such that: every string accepted starting from q is accepted starting from q', and vice versa.*

This is fine as an abstract mathematical definition of equivalence, but it doesn't seem to give us a way to compute which states are equivalent: we'd have to 'check' infinitely many strings $x \in \Sigma^*$.

Fortunately, there's an actual algorithm for DFA minimization that works in reasonable time.

This is useful in practice: we can specify our DFA in the most convenient way without worrying about its size, then minimize to a more 'compact' DFA to be implemented e.g. in hardware.

An algorithm for minimization

First eliminate any unreachable states (easy).

Then create a table of all possible pairs of states (p, q), initially unmarked. (E.g. a two-dimensional array of booleans, initially set to false.) We mark pairs (p, q) as and when we discover that p and q cannot be equivalent.

1. Start by marking all pairs (p, q) where $p \in F$ and $q \notin F$, or vice versa.
2. Look for unmarked pairs (p, q) such that for some $u \in \Sigma$, the pair $(\delta(p, u), \delta(q, u))$ is marked. Then mark (p, q).
3. Repeat step 2 until no such unmarked pairs remain.

If (p, q) is still unmarked, can collapse p, q to a single state.

Illustration of minimization algorithm

Consider the following DFA over $\{a, b\}$.

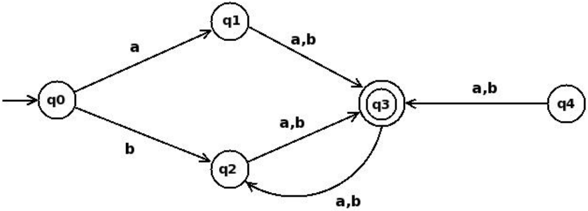


Illustration of minimization algorithm

After eliminating unreachable states:

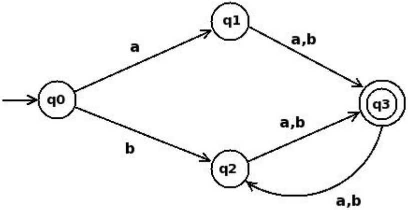
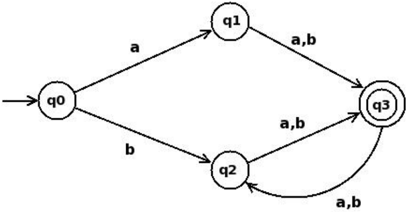


Illustration of minimization algorithm

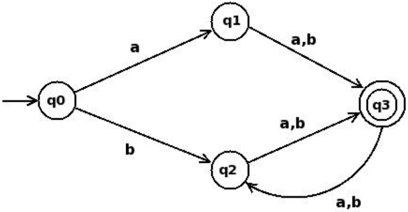
We mark states to be kept distinct using a half matrix:



q0				
q1	.			
q2	.	.		
q3	.	.	.	
	q0	q1	q2	q3

Illustration of minimization algorithm

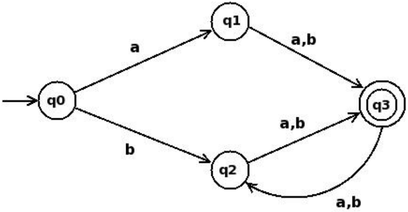
First mark accepting/non-accepting pairs:



q0				
q1	·			
q2	·	·		
q3	✓	✓	✓	
	q0	q1	q2	q3

Illustration of minimization algorithm

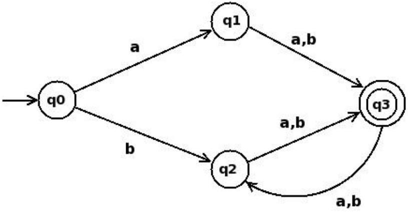
(q0,q1) is unmarked, qo $\xrightarrow{a}$ q1, q1 $\xrightarrow{a}$ q3, and (q1,q3) is marked.



q0				
q1	·			
q2	·	·		
q3	✓	✓	✓	
	q0	q1	q2	q3

Illustration of minimization algorithm

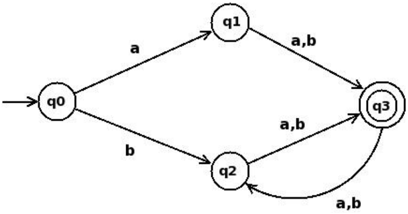
So mark (q0,q1).



q0				
q1	✓			
q2	.	.		
q3	✓	✓	✓	
	q0	q1	q2	q3

Illustration of minimization algorithm

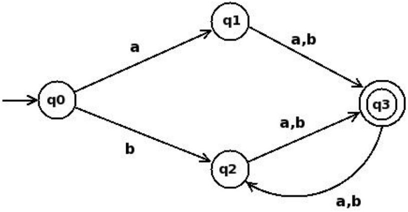
(q0,q2) is unmarked, qo $\xrightarrow{a}$ q1, q2 $\xrightarrow{a}$ q3, and (q1,q3) is marked.



q0				
q1	✓			
q2	·	·		
q3	✓	✓	✓	
	q0	q1	q2	q3

Illustration of minimization algorithm

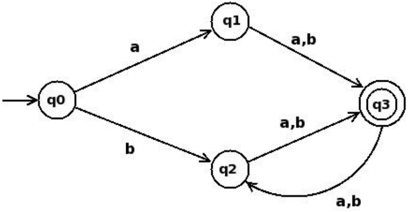
So mark (q0,q2).



q0				
q1	✓			
q2	✓	·		
q3	✓	✓	✓	
	q0	q1	q2	q3

Illustration of minimization algorithm

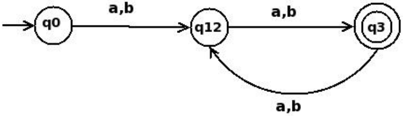
The only remaining unmarked pair (q1,q2) stays unmarked.



	q0	q1	q2	q3
q0				
q1	✓			
q2	✓	·		
q3	✓	✓	✓	

Illustration of minimization algorithm

So obtain minimized DFA by collapsing q1, q2 to a single state.



Why does this algorithm work?

Let's say a string s separates states p, q if s takes us from p to an accepting state and from q to a rejecting state, or *vice versa*.
Such an s is a reason for not merging p, q into a single state.
We mark (p, q) when we find that there's a string separating p, q:

- If $p \in F$ and $q \notin F$, or *vice versa*, then ϵ separates p, q.
- Suppose we mark (p, q) because we've found a previously marked pair (p', q') where $p \xrightarrow{a} p'$ and $q \xrightarrow{a} q'$ for some a.
  If s' is a separating string for p', q', then as' separates p, q.

We stop when there are no more pairs we can mark.
If (p, q) remains unmarked, why are p, q equivalent?

- If $s = a_1 \ldots a_n$ were a string separating p, q, we'd have

$$\begin{aligned} p &= p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} \cdots p_{n-1} \xrightarrow{a_n} p_n \;, \\ q &= q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots q_{n-1} \xrightarrow{a_n} q_n \end{aligned}$$

  with just one of p_n, q_n accepting. So we'd have marked (p_n, q_n) in Round 0, (p_{n-1}, q_{n-1}) by Round 1, ... and (p, q) by Round n.

Alternative: Brzozowski's minimization algorithm

There's a surprising alternative algorithm for minimizing a DFA $M = (Q, \delta, s, F)$ for a language L. Assume no unreachable states.

- Reverse the machine M: flip all the arrows, make F the set of start states, and make s the only accepting state.
  This gives an NFA N (*not* typically a DFA) which accepts $L^{rev} = \{rev(s) \mid s \in L\}$.
- Apply the subset construction to N, omitting unreachable states, to get a DFA P.
  It turns out that P is minimal for L^{rev} (clever)!
- Now apply the same two steps again, starting from P.
  The result is a minimal DFA for $(L^{rev})^{rev} = L$.

Comparing Brzozowski and marking algorithms

- Both algorithms result in the same minimal DFA for a given DFA M (recall that there's a unique minimal DFA up to isomorphism.)
- In the worst case, Brzozowski's algorithm can take time $O(2^n)$ for a DFA with n states. The marking algorithm, as presented, runs within time $O(kn^4)$, where $k = |\Sigma|$. (Can be improved further.)
- There are some practical cases where Brzozowski does better.
- Marking algorithm is probably easier to understand, and illustrates a common pattern (more examples later in course).

Improving determinization

Now we have a minimization algorithm, the following improved determinization procedure is possible.

To determinize an NFA M with n states:

1. Perform the subset construction on M to produce an equivalent DFA N with 2^n states.
2. Perform the minimization algorithm on N to produce a DFA $\mathrm{Min}(N)$ with $\leq 2^n$ states.

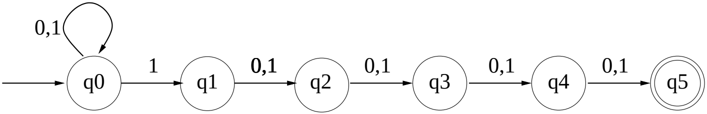
Using this method we are guaranteed to produce the smallest possible DFA equivalent to M.

In many cases this avoids the exponential state-space blow-up.

In some cases, however, an exponential blow-up is unavoidable.

Loose end from last lecture

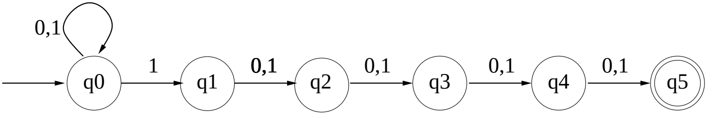
Consider last lecture's example NFA over $\{0, 1\}$:



What is the number of states of the smallest DFA that recognises the same language?

Loose end from last lecture

Consider last lecture's example NFA over $\{0, 1\}$:

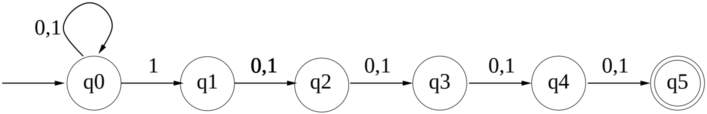


What is the number of states of the smallest DFA that recognises the same language?

Answer: The smallest DFA has 32 states.

Loose end from last lecture

Consider last lecture's example NFA over $\{0, 1\}$:



What is the number of states of the smallest DFA that recognises the same language?

Answer: The smallest DFA has 32 states.

More generally, the smallest DFA for the language:

$$\{x \in \Sigma^* \mid \text{the } n\text{-th symbol from the end of } x \text{ is } 1\}$$

has 2^n states. Whereas, there is an NFA with $n + 1$ states.

Reading

Relevant reading:

- Closure properties of regular languages: Kozen chapter 4.
- Minimization: Kozen chapters 13–14.

Next time:

- Regular expressions and Kleene's Theorem.
  (Kozen chapters 7–9.)