Closure Properties of Regular Languages

Union, Intersection, Difference, Concatenation, Kleene Closure, Pumping Lemma, Minimal State DFA

Closure Properties

- Recall a closure property is a statement that a certain operation on languages, when applied to languages in a class (e.g., the regular languages), produces a result that is also in that class.
- For regular languages, we can use any of its representations to prove a closure property.

Closure Under Union

- If L and M are regular languages, so is $L \cup M$.
- Proof: Let L and M be the languages of regular expressions R and S, respectively.
- Then $R|S$ is a regular expression whose language is $L \cup M$.

Closure Under Concatenation and Kleene Closure

- Same idea:
  - RS is a regular expression whose language is the concatenation LM.
  - R* is a regular expression whose language is L*.

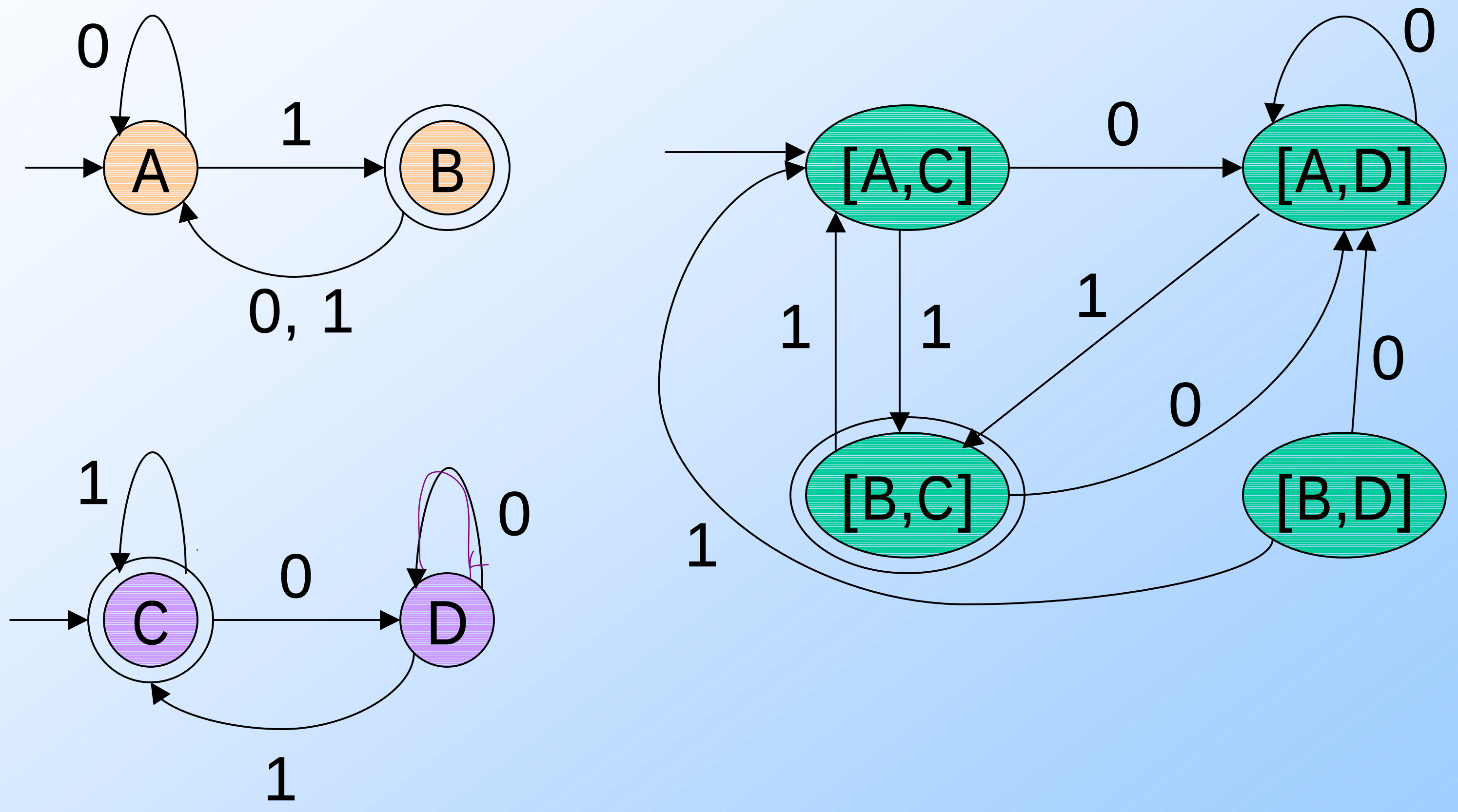
Product Automata

- Given languages L and M construct the *product DFA* from DFA's for L and M.
- Let these DFA's have sets of states Q and R, respectively.
- Product DFA has set of states $Q \times R$.
  - I.e., pairs [q, r] with q in Q, r in R.

Product DFA – Continued

- Start state = $[q_0, r_0]$ (the start states of the DFA's for L, M).
- Transitions: $\delta([q,r], a) = [\delta_L(q,a), \delta_M(r,a)]$
  - δ_L, δ_M are the transition functions for the DFA's of L, M.
  - That is, we simulate the two DFA's in the two state components of the product DFA.

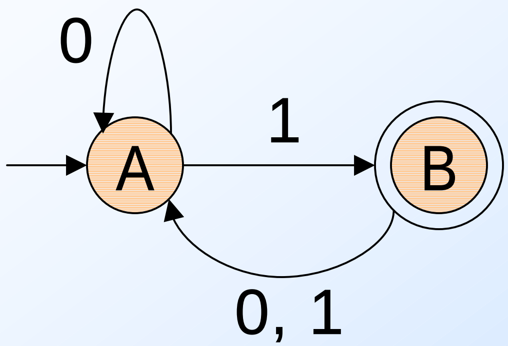
Example: Product DFA for Intersection

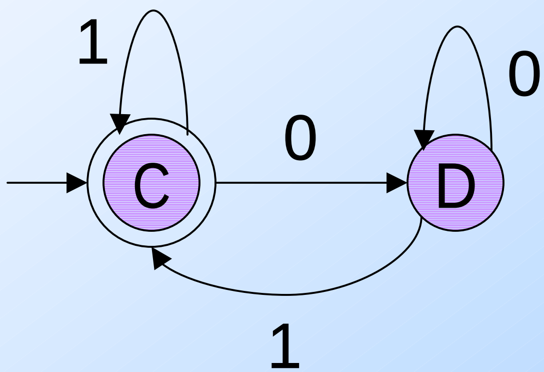


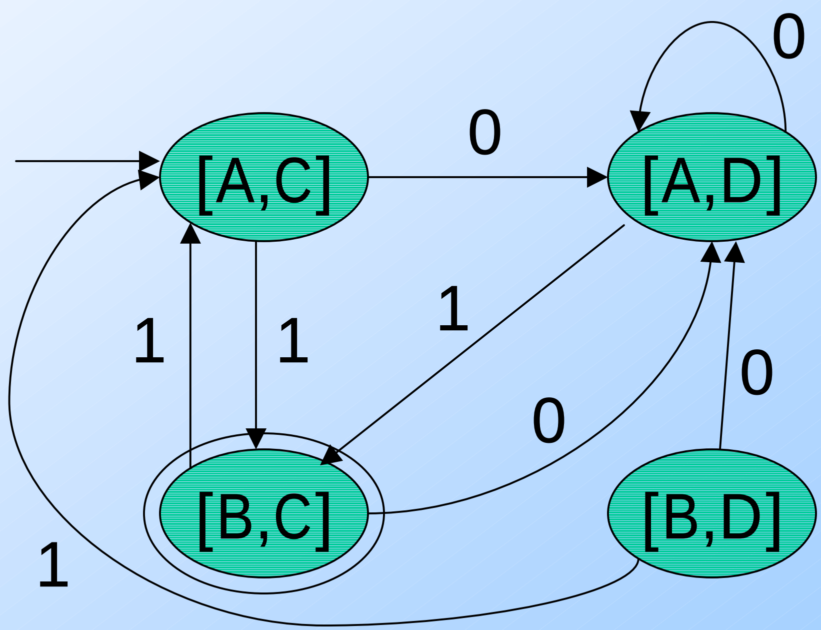
Closure Under Intersection

- If L and M are regular languages, then so is $L \cap M$.
- Proof: Let A and B be DFA's whose languages are L and M, respectively.
- Construct C, the product automaton of A and B.
- Make the final states of C be the pairs consisting of final states of both A and B.

Example: Product DFA for Intersection



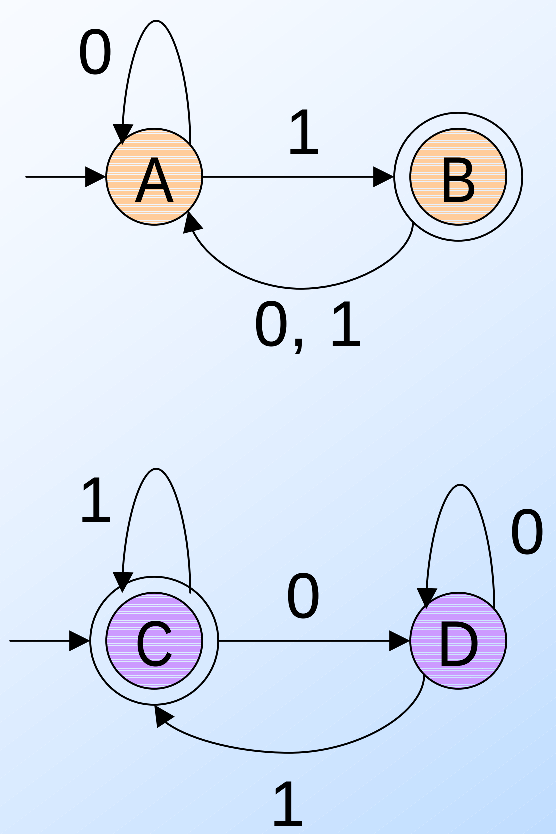


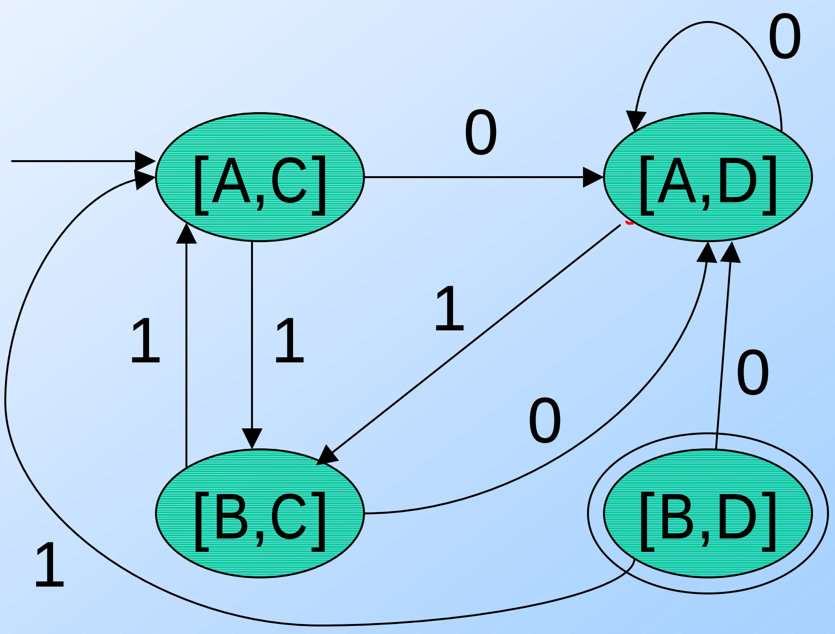


Closure Under Difference

- If L and M are regular languages, then so is $L - M$ = strings in L but not M.
- Proof: Let A and B be DFA's whose languages are L and M, respectively.
- Construct C, the product automaton of A and B.
- Make the final states of C be the pairs where A-state is final but B-state is not.

Example: Product DFA for Difference





Notice: difference
is the empty language

Closure Under Complementation

- The *complement* of a language L (with respect to an alphabet Σ such that Σ* contains L) is Σ* – L.
- Since Σ* is surely regular, the complement of a regular language is always regular.

Closure Under Complementation

Let L be regular and $A_L=(S,V,\delta,s_0,F)$ its DFA

The DFA for the complent language, $\underline{L}$, is:

$$A_{\underline{L}}=(S,V,\delta,s_0,S-F)$$

Decision Properties of Regular Languages

General Discussion of “Properties”

The Pumping Lemma

Membership, Emptiness, Etc.

Decision Properties

- A *decision property* for a class of languages is an algorithm that takes a formal description of a language (e.g., a DFA) and tells whether or not some property holds.
- Example: Is language L empty?

The Membership Question

- Our first decision property is the question: “is string w in regular language L?”
- Assume L is represented by a DFA A.
- Simulate the action of A on the sequence of input symbols forming w.

The Emptiness Problem

- Given a regular language, does the language contain any string at all.
- Assume representation is DFA.
- Construct the transition graph.
- Compute the set of states reachable from the start state.
- If any final state is reachable, then yes, else no.

The Infiniteness Problem

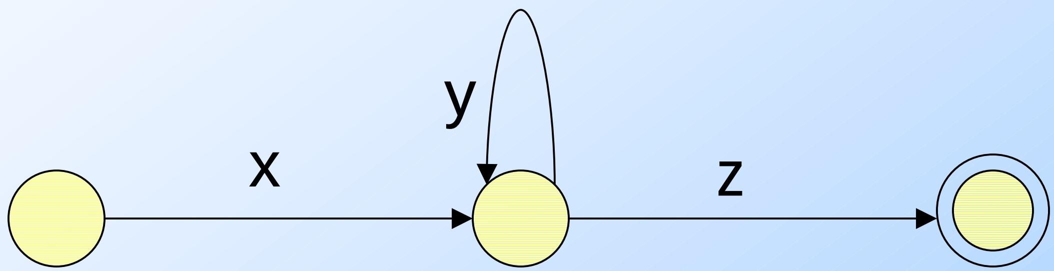
- Is a given regular language infinite?
- Start with a DFA for the language.
- Key idea: if the DFA has n states, and the language contains any string of length n or more, then the language is infinite.
- Otherwise, the language is surely finite.
  - Limited to strings of length n or less.

Proof of Key Idea

- If a DFA with |S|=n accepts a string w of length n or more, then there must be a state that appears twice on the path labeled w from the start state to a final state.
- Because there are at least n+1 states along the path.

Proof – (2)

w = xyz, with y≠ε

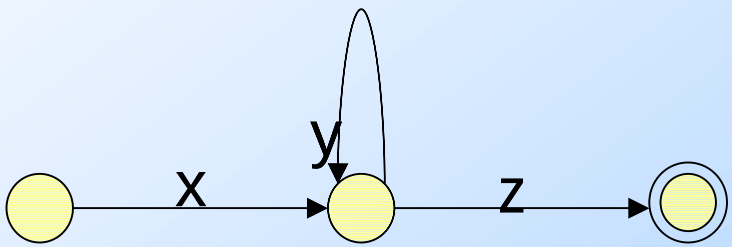


Then xy^iz is in the language for all $i \geq 0$.

Since y is not ϵ, we see an infinite number of strings in L.

Proof of Infiniteness

- Remember:



- We can choose y to be the first cycle on the path.
- So $|xy| \leq n$; in particular, $1 \leq |y| \leq n$.
- Thus, if w is of length n or more, then $w = xyz$, but also xy^iz is recognized.

The Pumping Lemma

- We have, almost accidentally, proved a statement that is quite useful for showing certain languages are not regular.
- Called the *pumping lemma for regular languages*.

Statement of the Pumping Lemma

For every regular language L

There is an integer n, such that

For every string w in L of length $\geq$ n

We can write w = xyz such that:

1. $|xy| \leq n$
2. $|y| \geq 1$
3. For all $i \geq 0$, xy^iz is in L.

Number of states of DFA for L

Labels along first cycle on path labeled w

Example: Use of Pumping Lemma

- We have claimed $\{0^k1^k \mid k \geq 1\}$ is not a regular language.
- Suppose it were. Then there would be an associated n for the pumping lemma.
- Let $w = 0^n1^n$. We can write $w = xyz$, where x and y consist of 0's, and $y \neq \epsilon$.
- But then xyyz would be in L, and this string has more 0's than 1's.

Example: Use of Closure Property

- We proved with Pumping Lemma that $L_1 = \{0^n1^n \mid n \geq 0\}$ is not a regular.
- L_2 = the set of strings with an equal number of 0's and 1's isn't either, but that fact is trickier to prove.
- Regular languages are closed under $\cap$.
- If L_2 were regular, then $L_2 \cap L(\mathbf{0^*1^*}) = L_1$ would be, but it isn't.

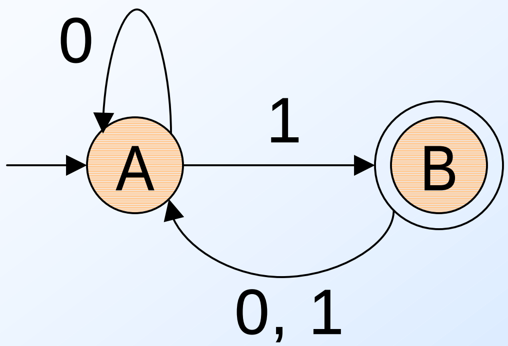
Decision Property: Equivalence

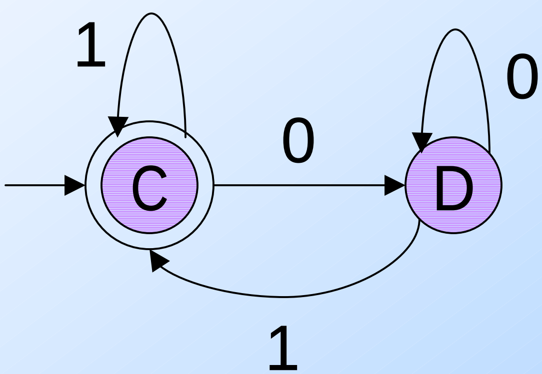
- Given regular languages L and M, is L = M?
- Algorithm involves constructing the *product DFA* from DFA's for L and M.

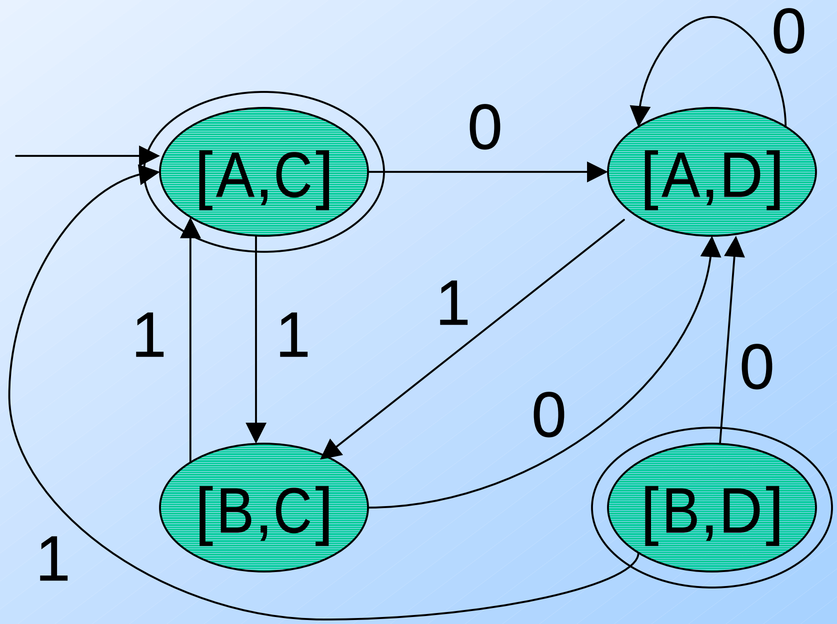
Equivalence Algorithm

- Make the final states of the product DFA be those states [q, r] such that exactly one of q and r is a final state of its own DFA.
- Thus, the product accepts w iff w is in exactly one of L and M.

Example: Equivalence







Equivalence Algorithm – (2)

- The product DFA's language is empty iff L = M.
- But we already have an algorithm to test whether the language of a DFA is empty.

Decision Property: Containment

- Given regular languages L and M, is $L \subseteq M$?
- Algorithm also uses the product automaton.
- How do you define the final states [q, r] of the product so its language is empty iff $L \subseteq M$?

Answer: q is final; r is not.

The Minimum-State DFA for a Regular Language

- In principle, since we can test for equivalence of DFA's we can, given a DFA *A* find the DFA with the fewest states accepting L(A).
- Test all smaller DFA's for equivalence with *A*.
- But that's a terrible algorithm.

Efficient State Minimization

- Construct a table with all pairs of states.
- If you find a string that *distinguishes* two states (takes exactly one to an accepting state), mark that pair.
- Algorithm is a recursion on the length of the shortest distinguishing string.

State Minimization – (2)

- Basis: Mark a pair if exactly one is a final state.
- Induction: mark [q, r] if there is some input symbol *a* such that [δ(q,a), δ(r,a)] is marked.
- After no more marks are possible, the unmarked pairs are equivalent and can be merged into one state.

Constructing the Minimum-State DFA

- Suppose $q_1,\ldots,q_k$ are indistinguishable states.
- Replace them by one state q.
- Then $\delta(q_1, a),\ldots, \delta(q_k, a)$ are all indistinguishable states.
  - Key point: otherwise, we should have marked at least one more pair.
- Let $\delta(q, a)$ = the representative state for that group.

Example: State Minimization

		r	b
→	{1}	{2,4}	{5}
	{2,4}	{2,4,6,8}	{1,3,5,7}
	{5}	{2,4,6,8}	{1,3,7,9}
	{2,4,6,8}	{2,4,6,8}	{1,3,5,7,9}
	{1,3,5,7}	{2,4,6,8}	{1,3,5,7,9}
*	{1,3,7,9}	{2,4,6,8}	{5}
*	{1,3,5,7,9}	{2,4,6,8}	{1,3,5,7,9}

		r	b
→	A	B	C
	B	D	E
	C	D	F
	D	D	G
	E	D	G
*	F	D	C
*	G	D	G

Here it is with more convenient state names

Remember this DFA? It was constructed for the chessboard NFA by the subset construction.

Example – Continued

		r	b
→	A	B	C
	B	D	E
	C	D	F
	D	D	G
	E	D	G
*	F	D	C
*	G	D	G

	G	F	E	D	C	B
A	x	x				
B	x	x				
C	x	x				
D	x	x				
E	x	x				
F						

Start with marks for the pairs with one of the final states F or G.

Example – Continued

		r	b
→	A	B	C
	B	D	E
	C	D	F
	D	D	G
	E	D	G
*	F	D	C
*	G	D	G

	G	F	E	D	C	B
A	x	x				
B	x	x				
C	x	x				
D	x	x				
E	x	x				
F						

Input r gives no help,
because the pair [B, D]
is not marked.

Example – Continued

		r	b
→	A	B	C
	B	D	E
	C	D	F
	D	D	G
	E	D	G
*	F	D	C
*	G	D	G

	G	F	E	D	C	B
A	x	x	x	x	x	
B	x	x	x	x	x	
C	x	x				
D	x	x				
E	x	x				
F	x					

But input b distinguishes {A,B,F} from {C,D,E,G}. For example, [A, C] gets marked because [C, F] is marked.

Example – Continued

		r	b
→	A	B	C
	B	D	E
	C	D	F
	D	D	G
	E	D	G
*	F	D	C
*	G	D	G

	G	F	E	D	C	B
A	X	X	X	X	X	
B	X	X	X	X	X	
C	X	X	X	X		
D	X	X				
E	X	X				
F	X					

[C, D] and [C, E] are marked because of transitions on b to marked pair [F, G].

Example – Continued

		r	b
→	A	B	C
	B	D	E
	C	D	F
	D	D	G
	E	D	G
*	F	D	C
*	G	D	G

	G	F	E	D	C	B
A	X	X	X	X	X	X
B	X	X	X	X	X	
C	X	X	X	X		
D	X	X				
E	X	X				
F	X					

[A, B] is marked because of transitions on r to marked pair [B, D].

[D, E] can never be marked, because on both inputs they go to the same state.

Example – Concluded

		r	b
→	A	B	C
	B	D	E
	C	D	F
	D	D	G
	E	D	G
*	F	D	C
*	G	D	G

		r	b
→	A	B	C
	B	H	H
	C	H	F
	H	H	G
*	F	H	C
*	G	H	G

	G	F	E	D	C	B
A	x	x	x	x	x	x
B	x	x	x	x	x	
C	x	x	x	x		
D	x	x				
E	x	x				
F	x					

Replace D and E by H.

Result is the minimum-state DFA.

Eliminating Unreachable States

- Unfortunately, combining indistinguishable states could leave us with unreachable states in the “minimum-state” DFA.
- Thus, before or after, remove states that are not reachable from the start state.

Clincher

- We have combined states of the given DFA wherever possible.
- Could there be another, completely unrelated DFA with fewer states?
- No. The proof involves minimizing the DFA we derived with the hypothetical better DFA.