Decision Properties of Regular Languages

General Discussion of “Properties”

The Pumping Lemma

Membership, Emptiness, Etc.

Properties of Language Classes

- A *language class* is a set of languages.
  - Example: the regular languages.
- Language classes have two important kinds of properties:
  1. Decision properties.
  2. Closure properties.

Closure Properties

- A *closure property* of a language class says that given languages in the class, an operation (e.g., union) produces another language in the same class.
- Example: the regular languages are obviously closed under union, concatenation, and (Kleene) closure.
  - Use the RE representation of languages.

Representation of Languages

- Representations can be formal or informal.
- Example (formal): represent a language by a RE or FA defining it.
- Example: (informal): a logical or prose statement about its strings:
  - $\{0^n 1^n \mid n \text{ is a nonnegative integer}\}$
  - "The set of strings consisting of some number of 0's followed by the same number of 1's."

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?

Why Decision Properties?

- Think about DFA's representing protocols.
- Example: "Does the protocol terminate?" = "Is the language finite?"
- Example: "Can the protocol fail?" = "Is the language nonempty?"
  - Make the final state be the "error" state.

Why Decision Properties – (2)

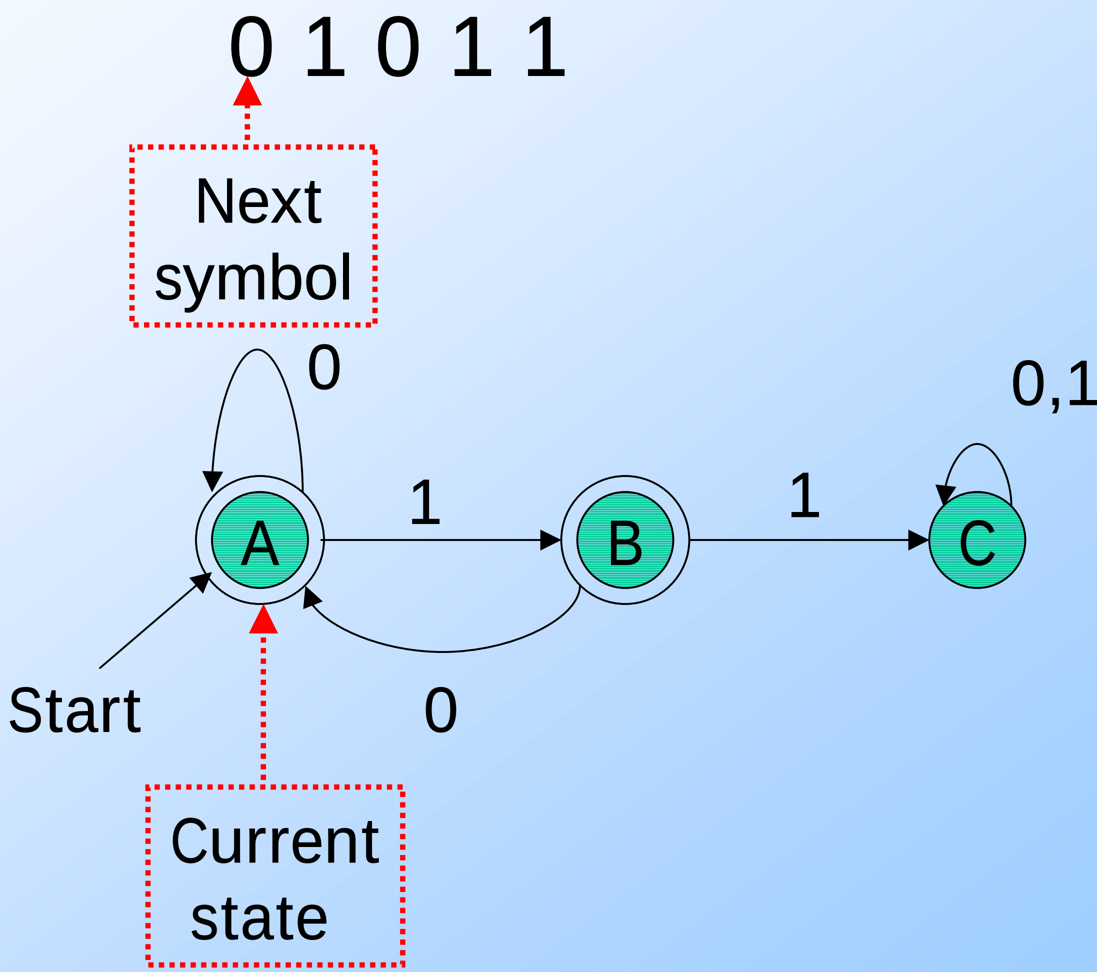
- We might want a "smallest" representation for a language, e.g., a minimum-state DFA or a shortest RE.
- If you can't decide "Are these two languages the same?"
  - I.e., do two DFA's define the same language?

You can't find a "smallest."

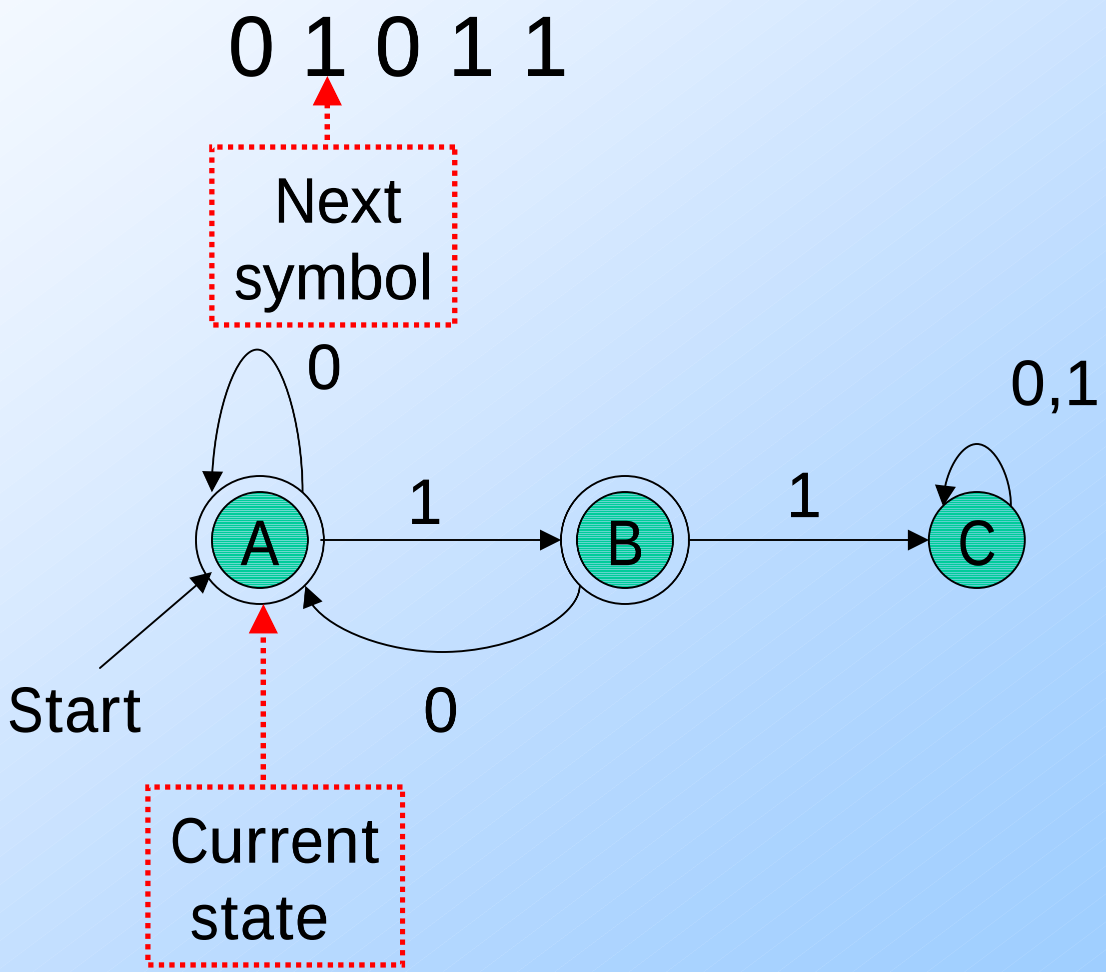
The Membership Problem

- Our first decision property for regular languages 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.

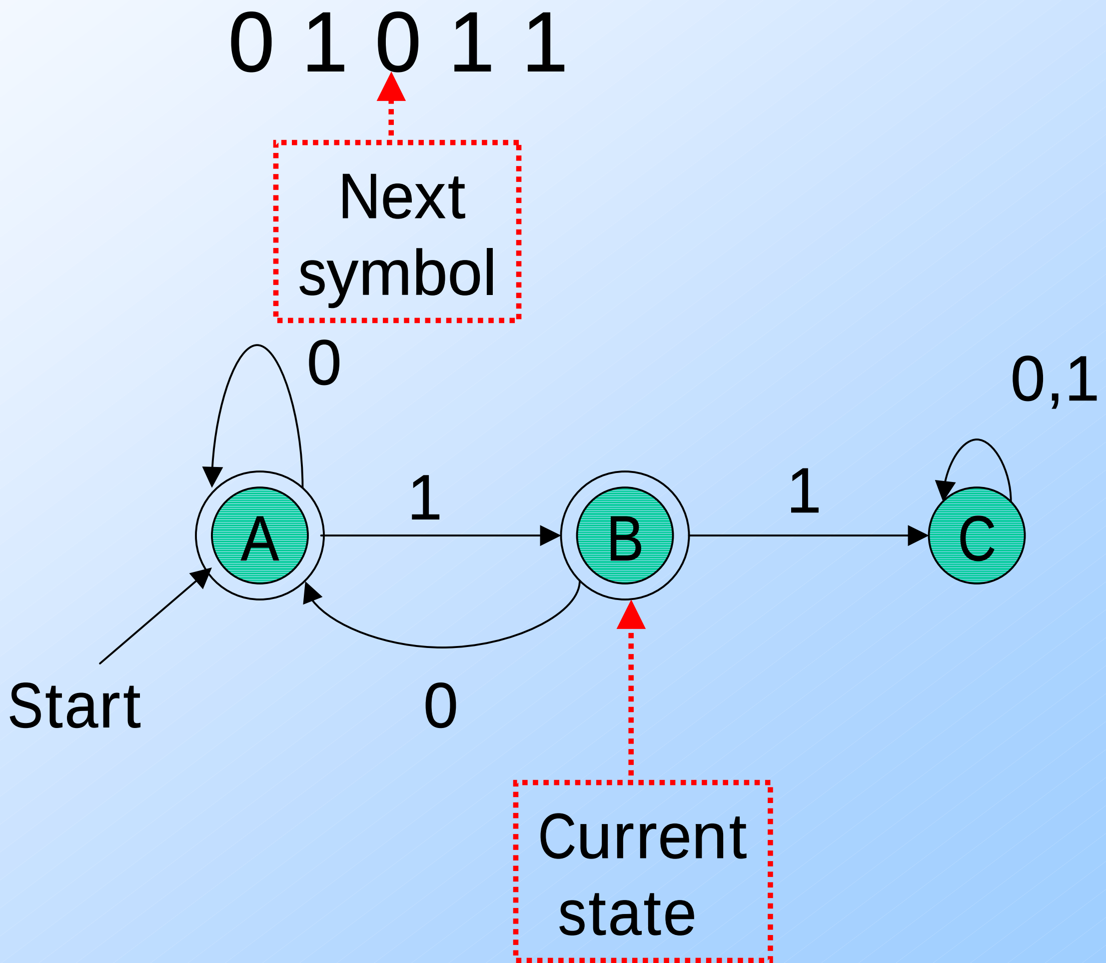
Example: Testing Membership



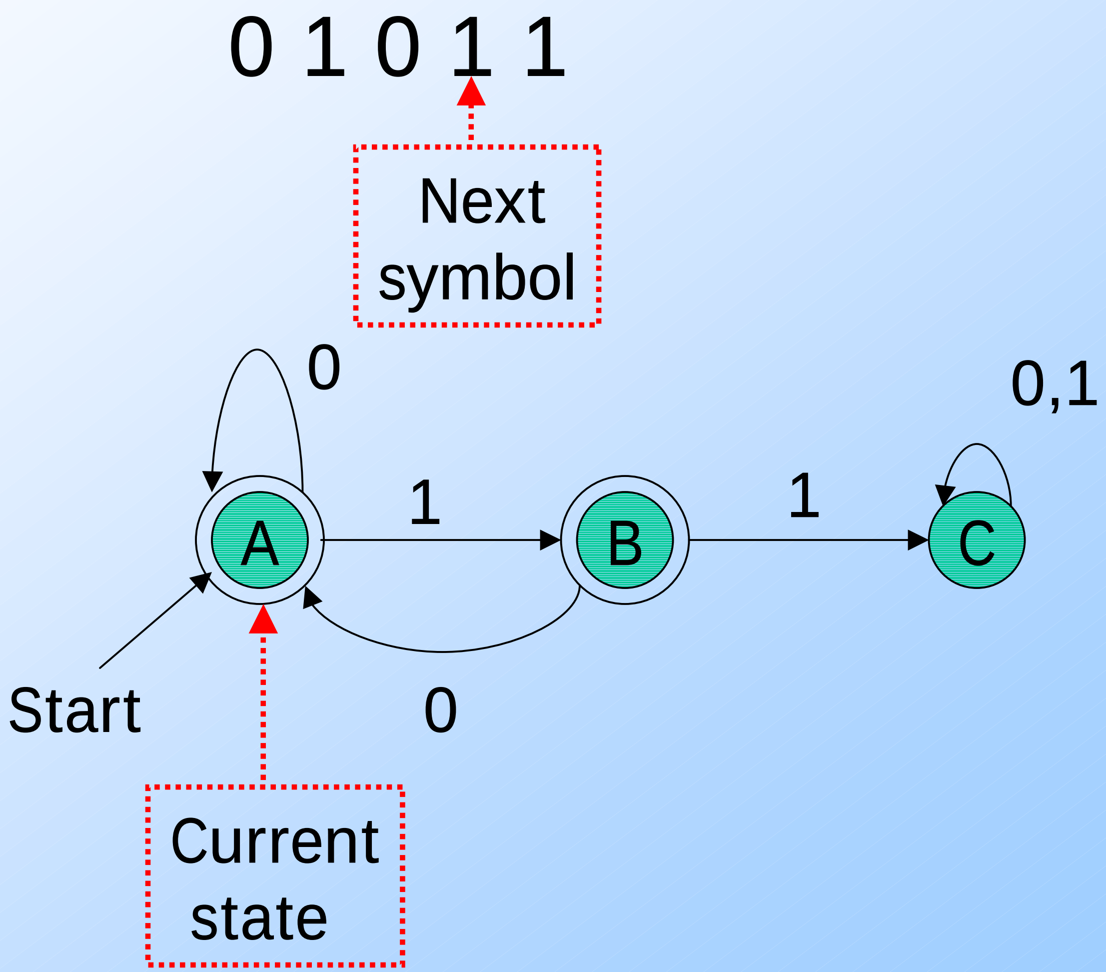
Example: Testing Membership



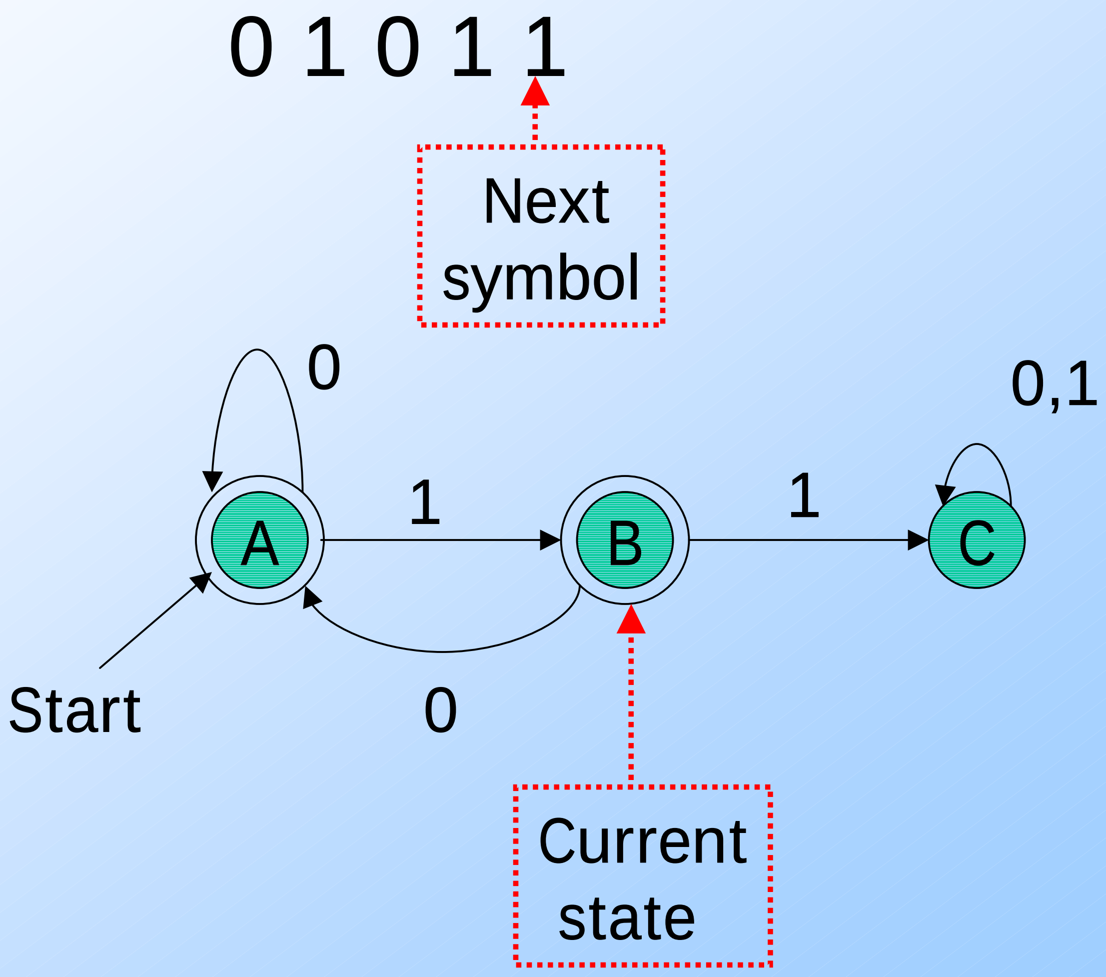
Example: Testing Membership



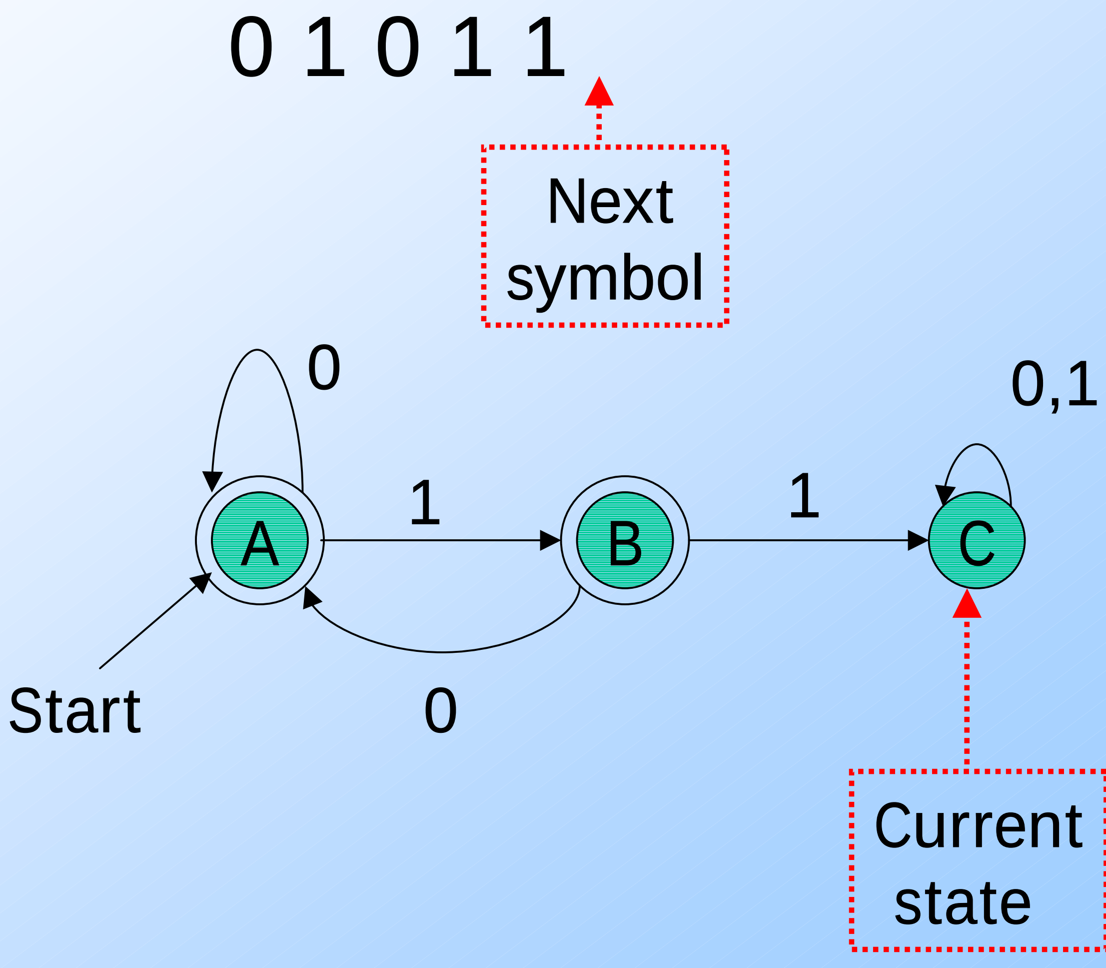
Example: Testing Membership



Example: Testing Membership

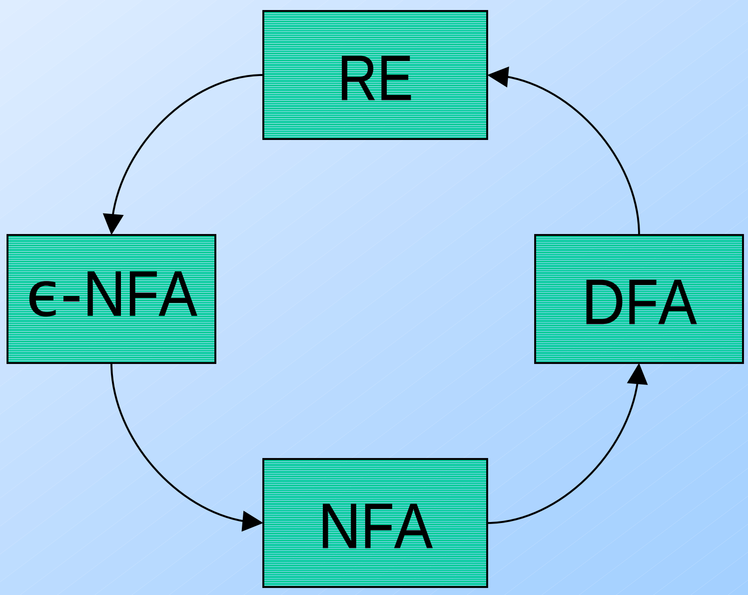


Example: Testing Membership



What if We Have the Wrong Representation?

- There is a circle of conversions from one form to another:



The Emptiness Problem

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

The Infiniteness Problem

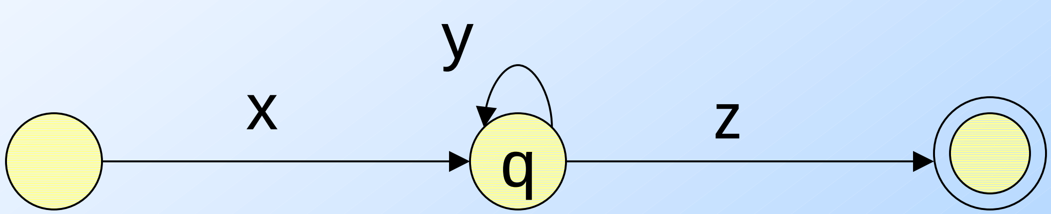
- Is a given regular language infinite?
- Start with a DFA for the language.
- Key idea: if the DFA has n states, and the language contains any string of length n or more, then the language is infinite.
- Otherwise, the language is surely finite.
  - Limited to strings of length n or less.

Proof of Key Idea

- If an n-state DFA 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



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

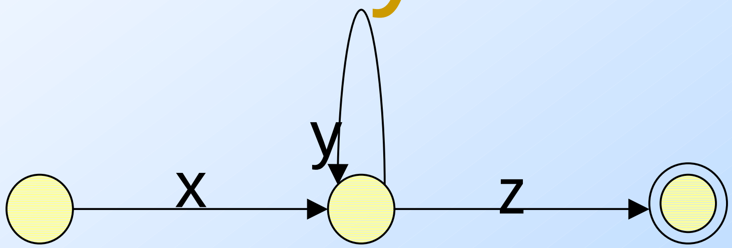
Since y is not ϵ, we see an infinite number of strings in L.

Infiniteness – Continued

- We do not yet have an algorithm.
- There are an infinite number of strings of length > n, and we can't test them all.
- Second key idea: if there is a string of length $\geq$ n (= number of states) in L, then there is a string of length between n and 2n-1.

Proof of 2^{nd} Key Idea

- Remember:



- y is the first cycle on the path.
- So $|xy| \leq n$; in particular, $1 \leq |y| \leq n$.
- Thus, if w is of length 2n or more, there is a shorter string in L that is still of length at least n.
- Keep shortening to reach [n, 2n-1].

Completion of Infiniteness Algorithm

- Test for membership all strings of length between n and 2n-1.
  - If any are accepted, then infinite, else finite.
- A terrible algorithm.
- Better: find cycles between the start state and a final state.

Finding Cycles

1. Eliminate states not reachable from the start state.
2. Eliminate states that do not reach a final state.
3. Test if the remaining transition graph has any cycles.

Finding Cycles – (2)

- But a simple, less efficient way to find cycles is to search forward from a given node N.
- If you can reach N, then there is a cycle.
- Do this starting at each node.

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. $|y| > 0$.
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.

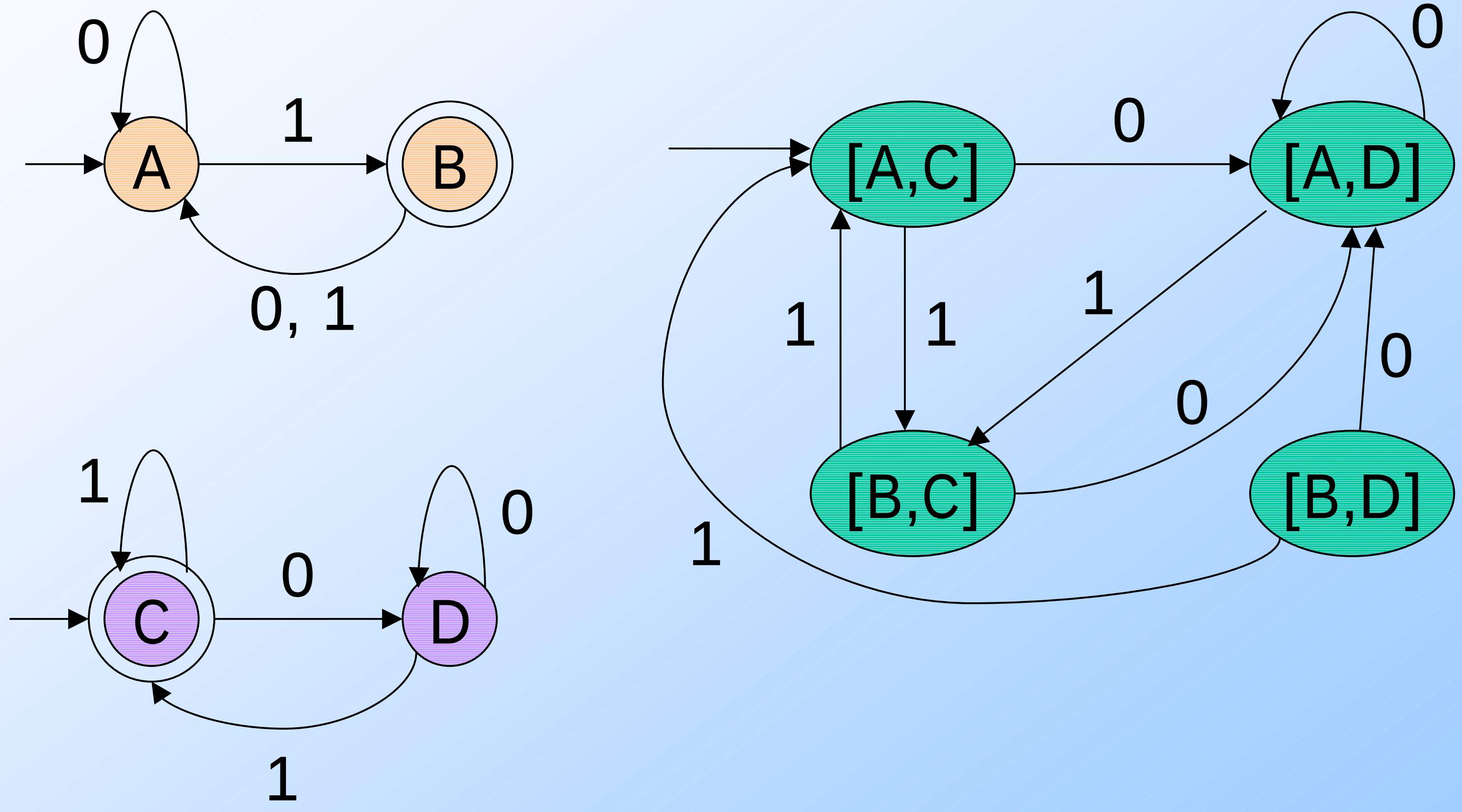
Decision Property: Equivalence

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

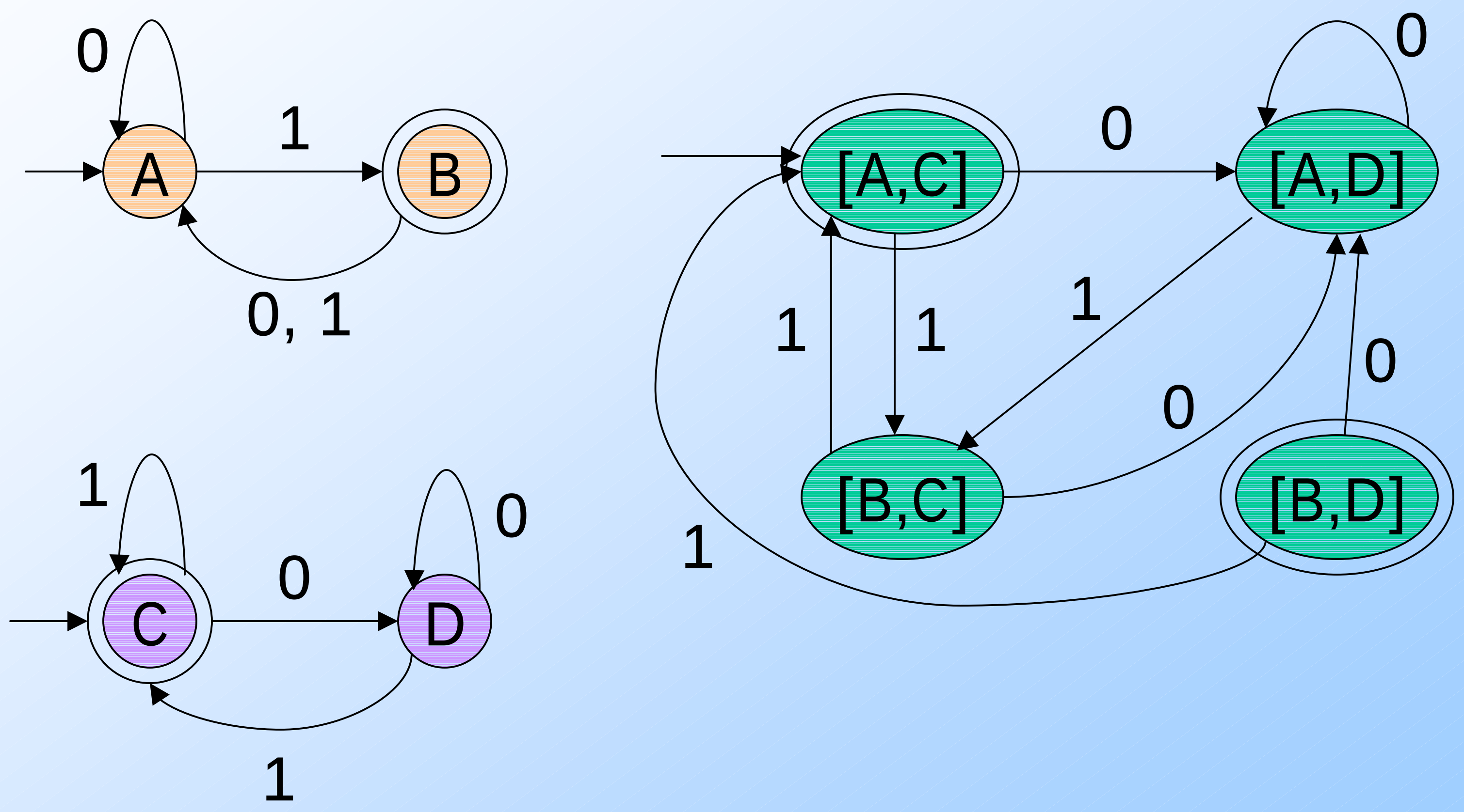
Example: Product DFA



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.
- L = M if and only if the product automaton's language is empty.

Example: Equivalence

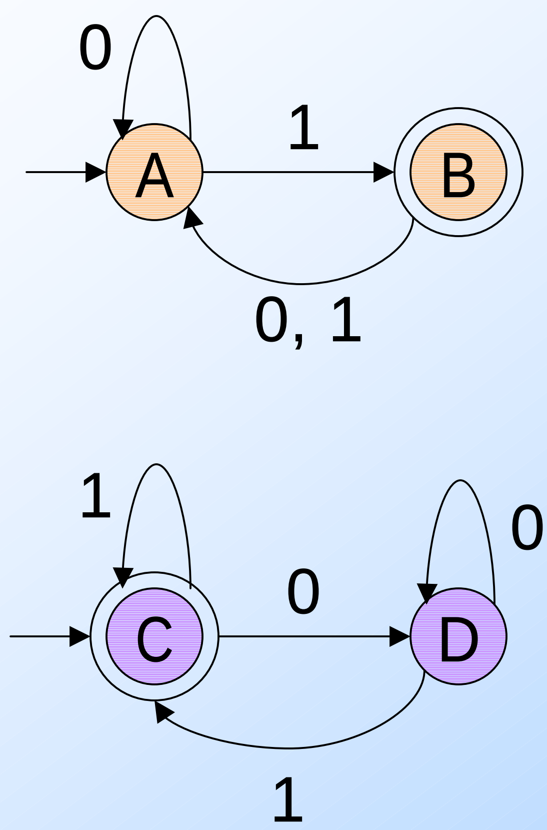


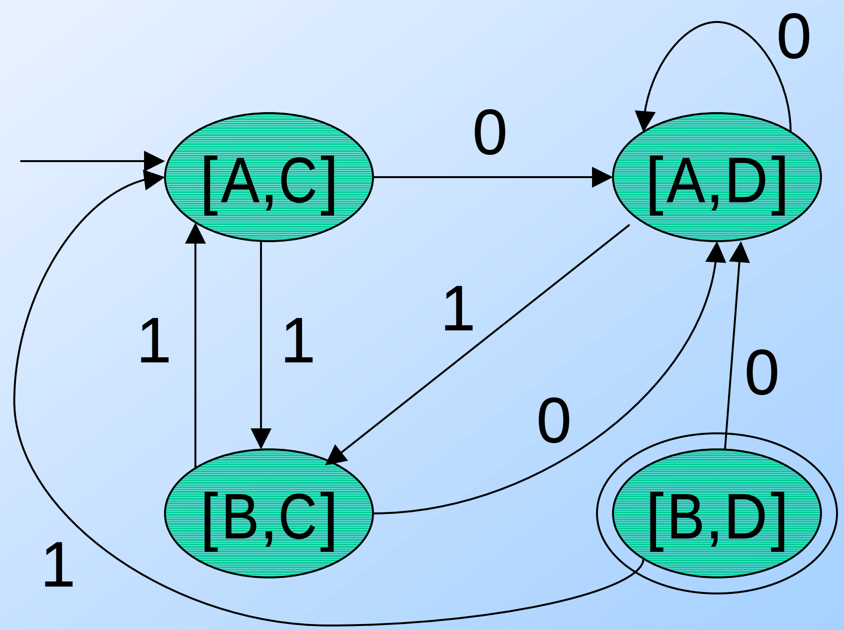
Decision Property: Containment

- 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.

Example: Containment





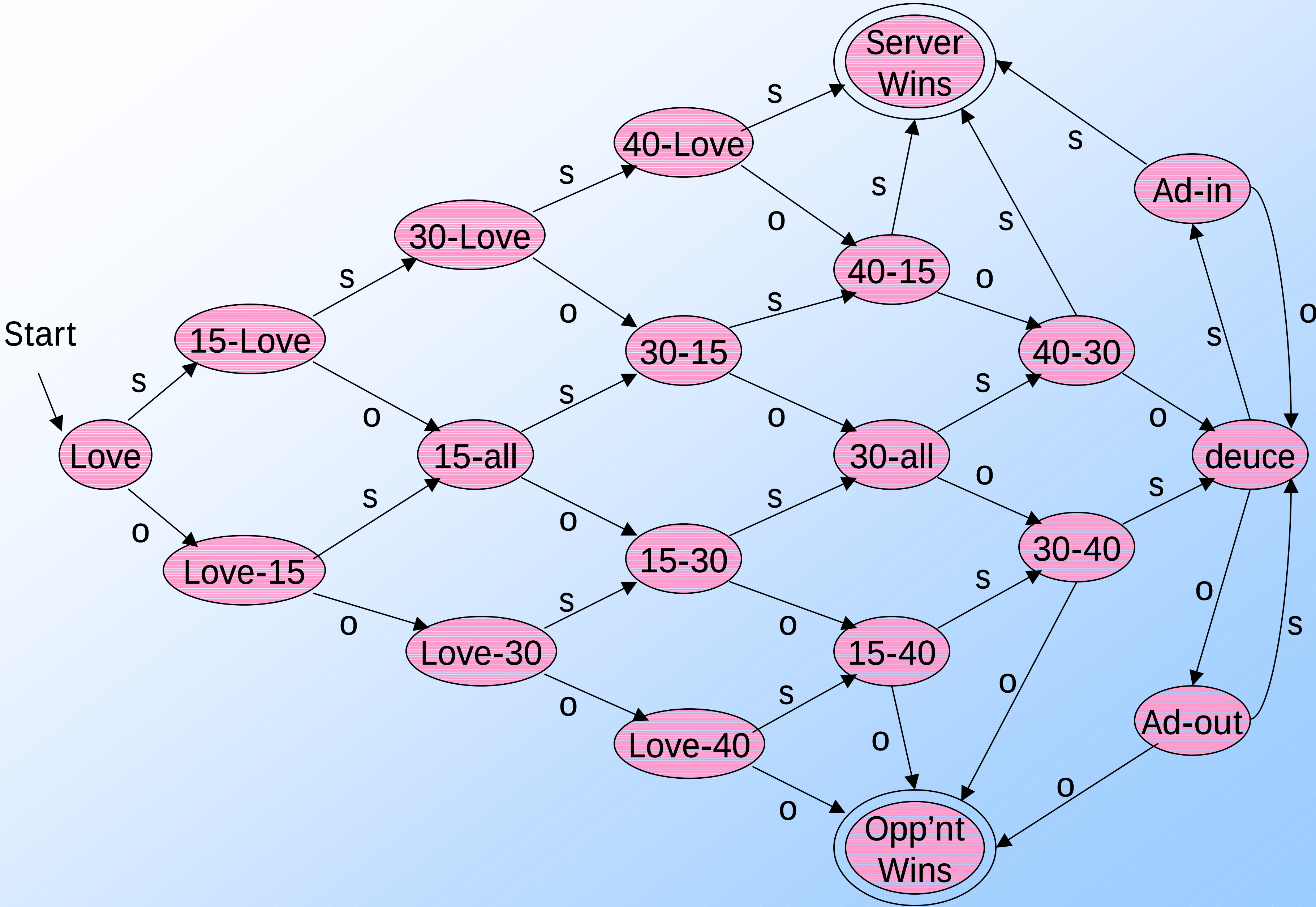
Note: the only final state is unreachable, so containment holds.

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.


Start
Love
15-Love
Love-15
30-Love
15-all
Love-30
40-Love
30-15
15-30
Love-40
Server Wins
40-15
30-all
15-40
Opp'nt Wins
40-30
30-40
Ad-in
deuce
Ad-out

State Minimization – (2)

- Basis: Mark pairs with exactly one final state.
- Induction: mark [q, r] if for some input symbol *a*, [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.

Transitivity of "Indistinguishable"

- If state p is indistinguishable from q, and q is indistinguishable from r, then p is indistinguishable from r.
- Proof: The outcome (accept or don't) of p and q on input w is the same, and the outcome of q and r on w is the same, then likewise the outcome of p and r.

Constructing the Minimum-State DFA

- Suppose $q_1,\dots,q_k$ are indistinguishable states.
- Replace them by one *representative* state q.
- Then $\delta(q_1, a),\dots, \delta(q_k, a)$ 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.

Proof: No Unrelated, Smaller DFA

- Let A be our minimized DFA; let B be a smaller equivalent.
- Consider an automaton with the states of A and B combined.
- Use "distinguishable" in its contrapositive form:
  - If states q and p are indistinguishable, so are $\delta(q, a)$ and $\delta(p, a)$.

Inferring Indistinguishability

$$q_0 \xrightarrow{a} q \xrightarrow{b} r$$

Start states of A and B indistinguishable because L(A) = L(B).

Must be indistinguishable

Must be indistinguishable

$$p_0 \xrightarrow{a} p \xrightarrow{b} s$$

Inductive Hypothesis

- Every state q of A is indistinguishable from some state of B.
- Induction is on the length of the shortest string taking you from the start state of A to q.

Proof – (2)

- Basis: Start states of A and B are indistinguishable, because L(A) = L(B).
- Induction: Suppose w = xa is a shortest string getting A to state q.
- By the IH, x gets A to some state r that is indistinguishable from some state p of B.
- Then $\delta_A(r, a) = q$ is indistinguishable from $\delta_B(p, a)$.

Proof – (3)

- However, two states of A cannot be indistinguishable from the same state of B, or they would be indistinguishable from each other.
  - Violates transitivity of “indistinguishable.”
- Thus, B has at least as many states as A.