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.
  - We have one example: the regular languages.
  - We'll see many more in this class.
- Language classes have two important kinds of properties:
  1. Decision properties.
  2. Closure properties.

Representation of Languages

- Representations can be formal or informal.
- Example (formal): represent a language by a RE or DFA defining it.
- Example: (informal): a logical or prose statement about its strings:
  - $\{0^n1^n \mid n \text{ is a nonnegative integer}\}$
  - "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?

Subtle Point: Representation Matters

- You might imagine that the language is described informally, so if my description is “the empty language” then yes, otherwise no.
- But the representation is a DFA (or a RE that you will convert to a DFA).
- Can you tell if $L(A) = \emptyset$ for DFA A?

Why Decision Properties?

- When we talked about protocols represented as DFA's, we noted that important properties of a good protocol were related to the language of the DFA.
- Example: "Does the protocol terminate?" = "Is the language finite?"
- Example: "Can the protocol fail?" = "Is the language nonempty?"

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."

Closure Properties

- A *closure property* of a language class says that given languages in the class, an operator (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.

Why Closure Properties?

1. Helps construct representations.
2. Helps show (informally described) languages not to be in the class.

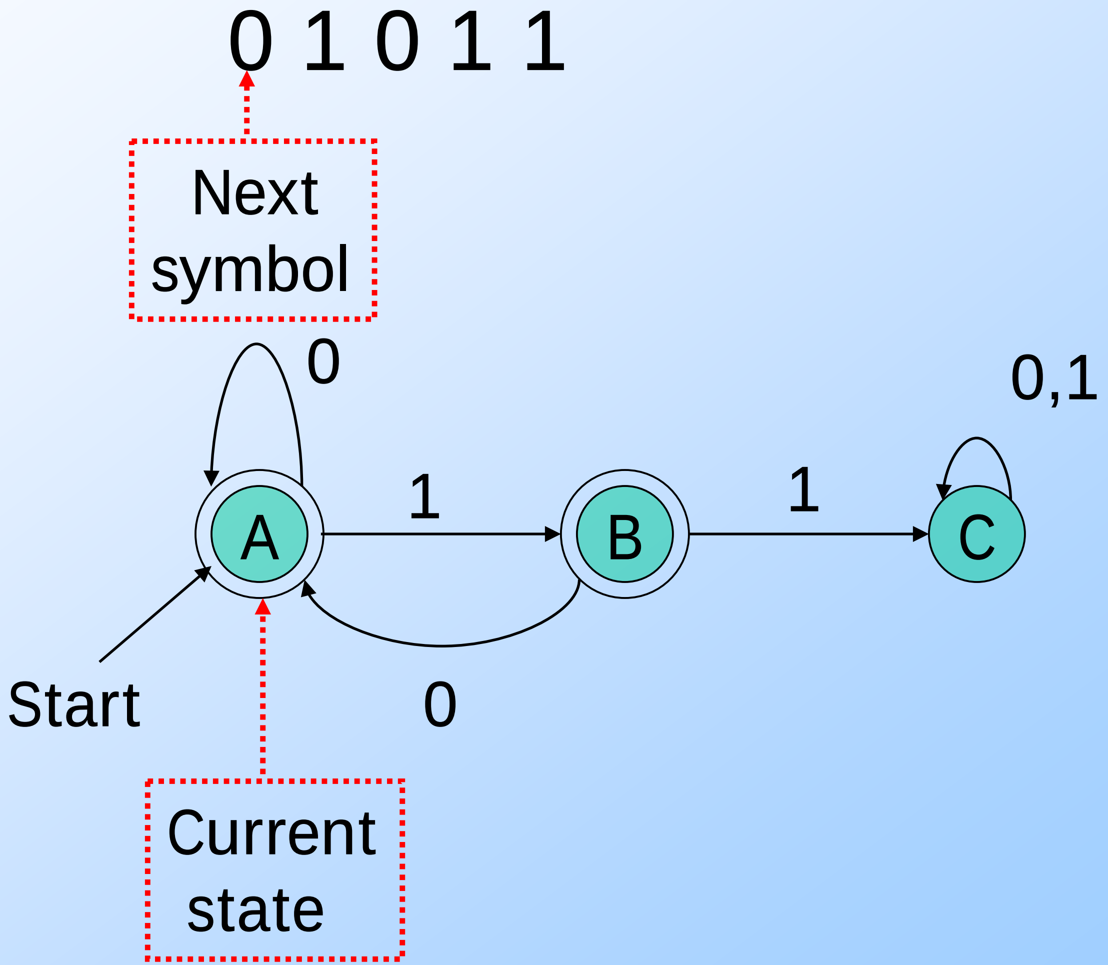
Example: Use of Closure Property

- We can easily prove $L_1 = \{0^n1^n \mid n \geq 0\}$ is not a regular language.
- L_2 = the set of strings with an = 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.

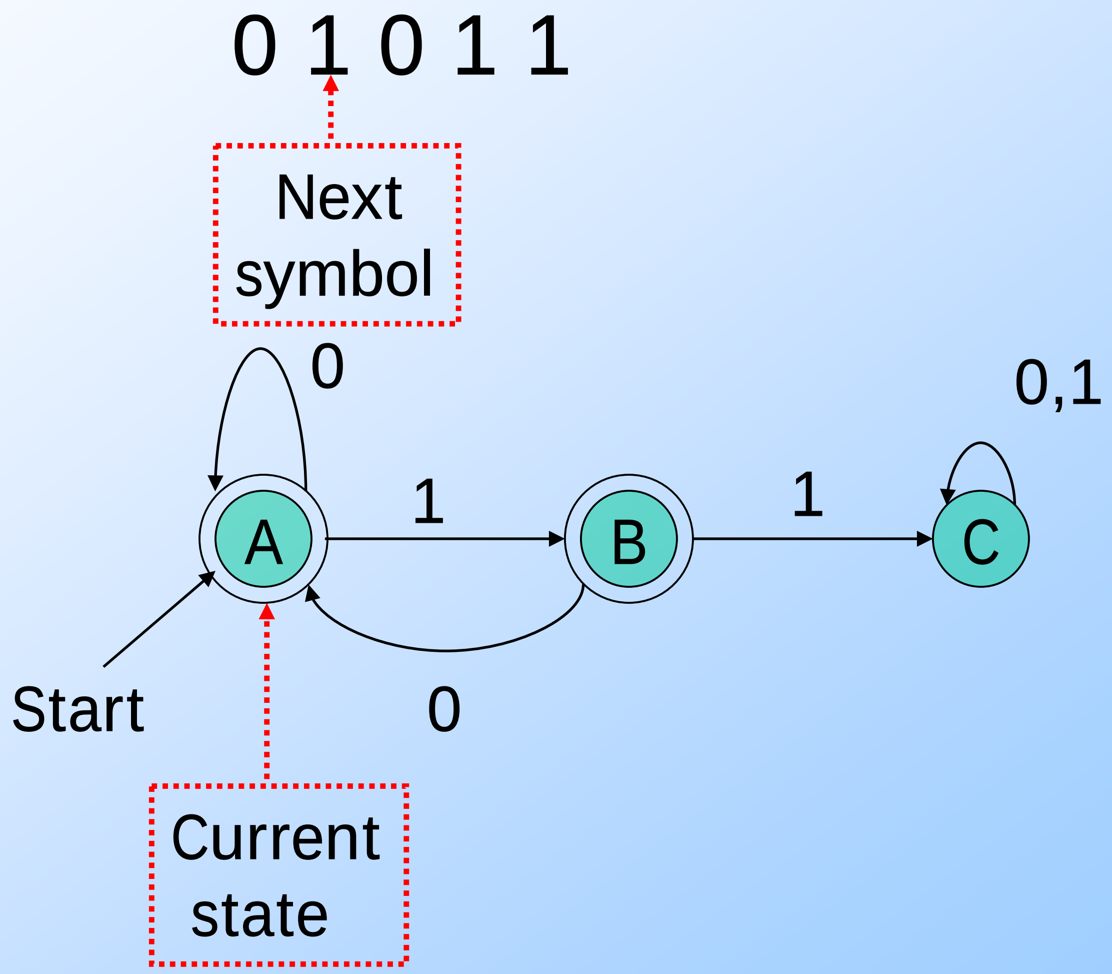
The Membership Question

- 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.

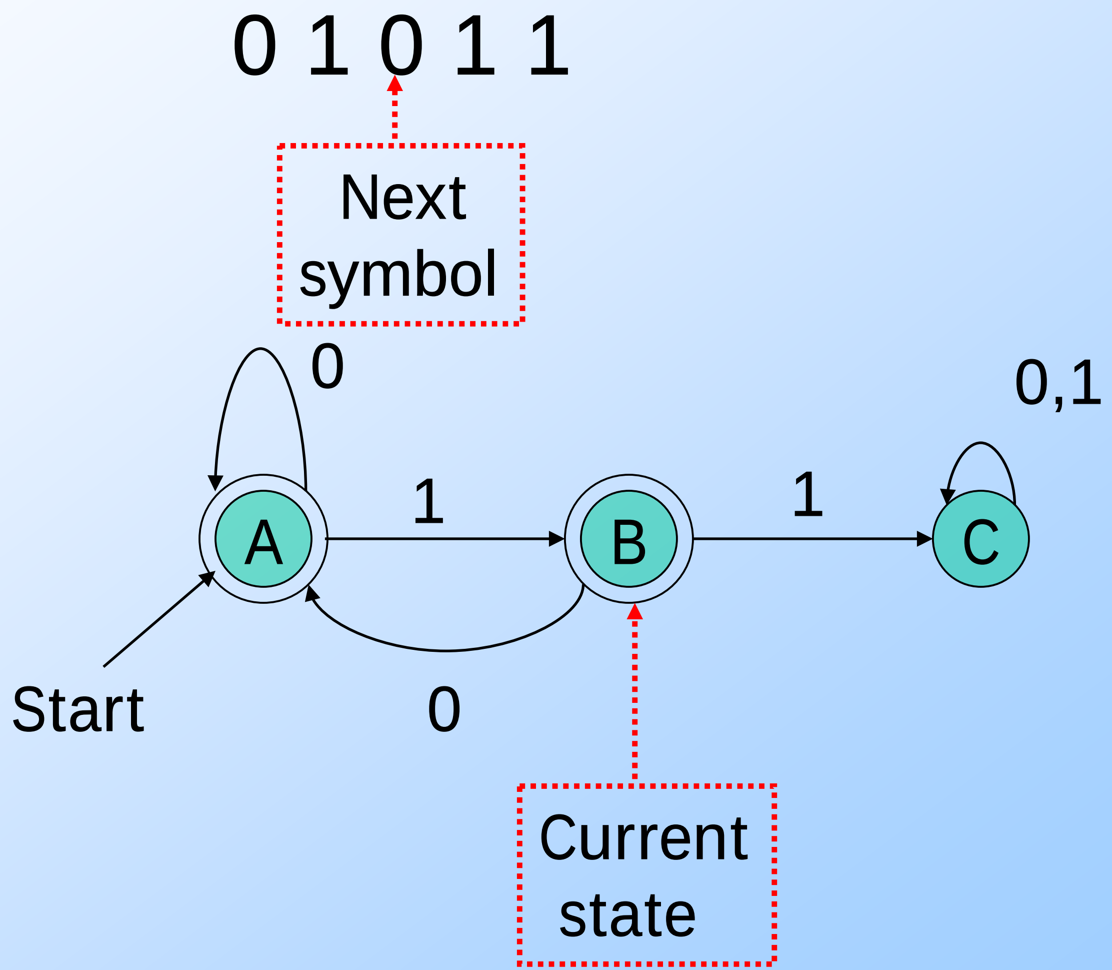
Example: Testing Membership



Example: Testing Membership

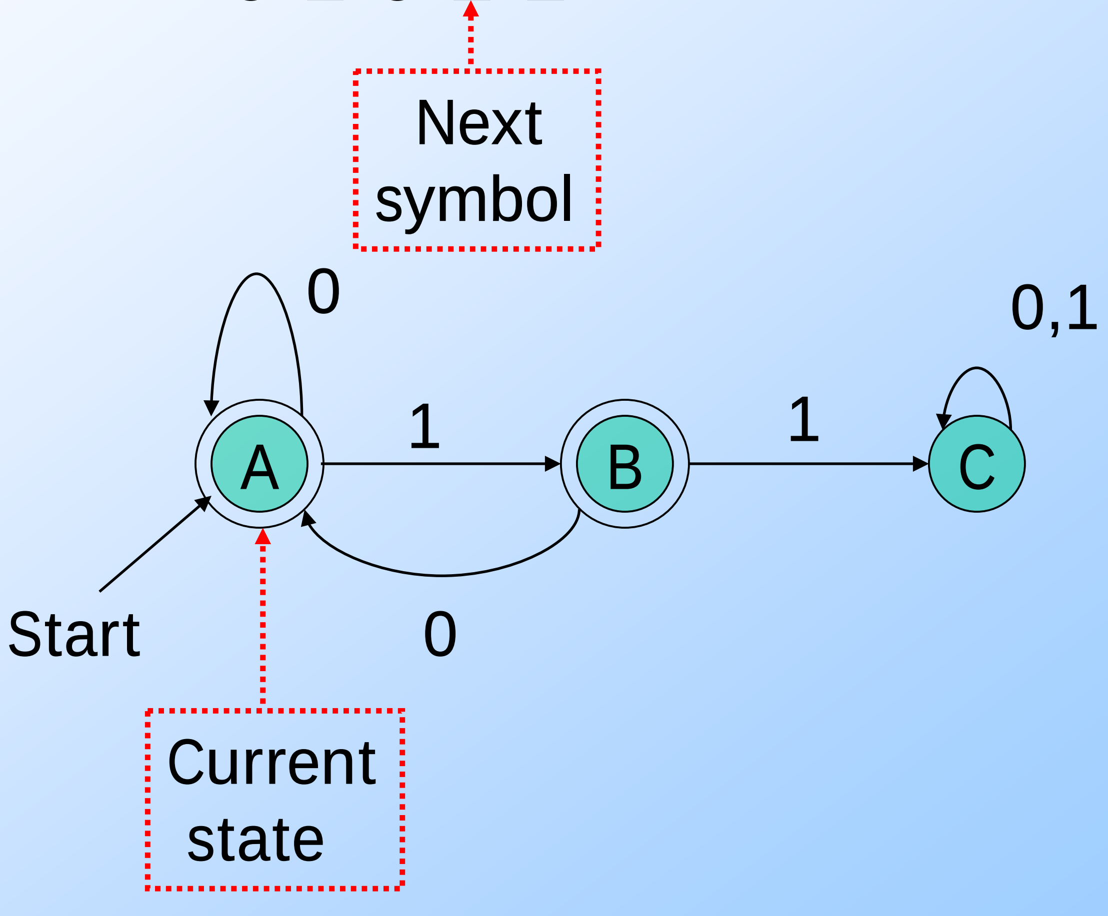


Example: Testing Membership

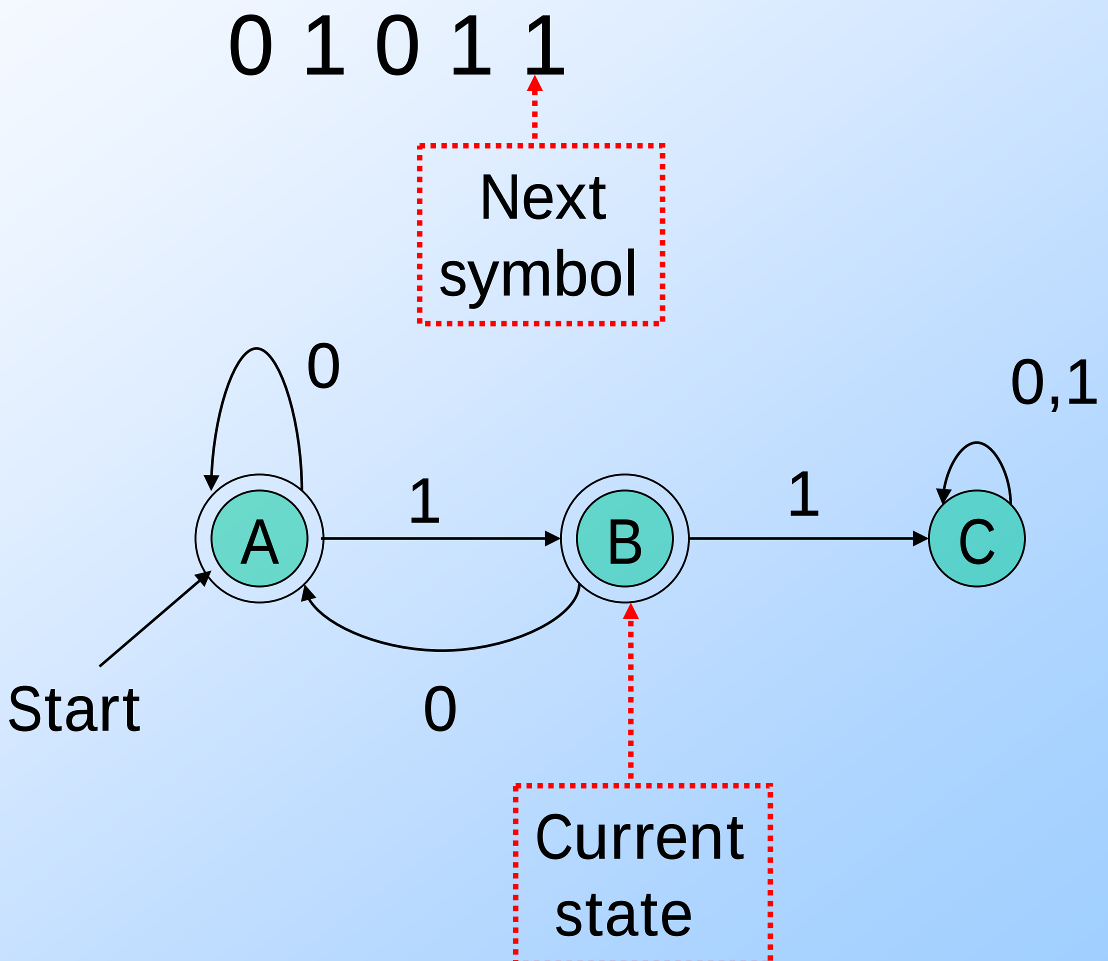


Example: Testing Membership

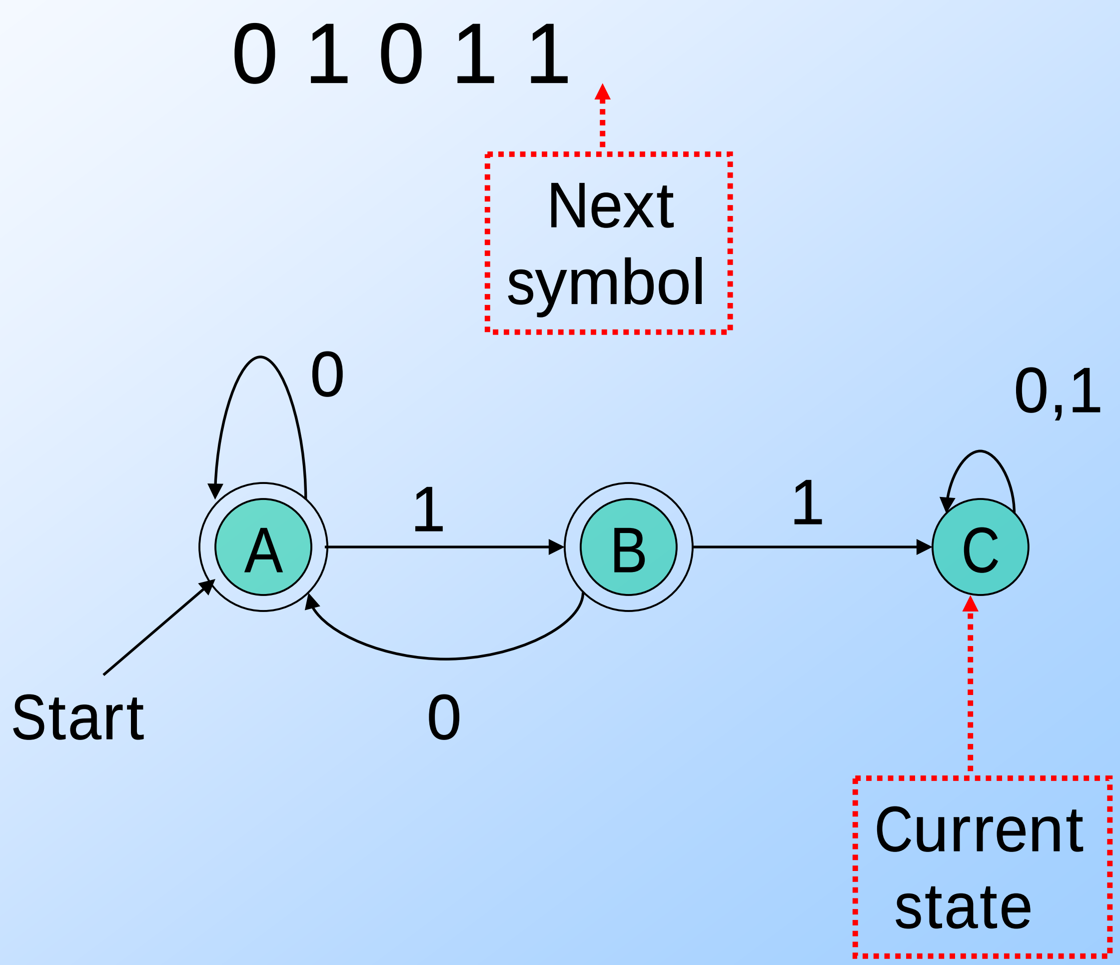
0 1 0 1 1



Example: Testing Membership

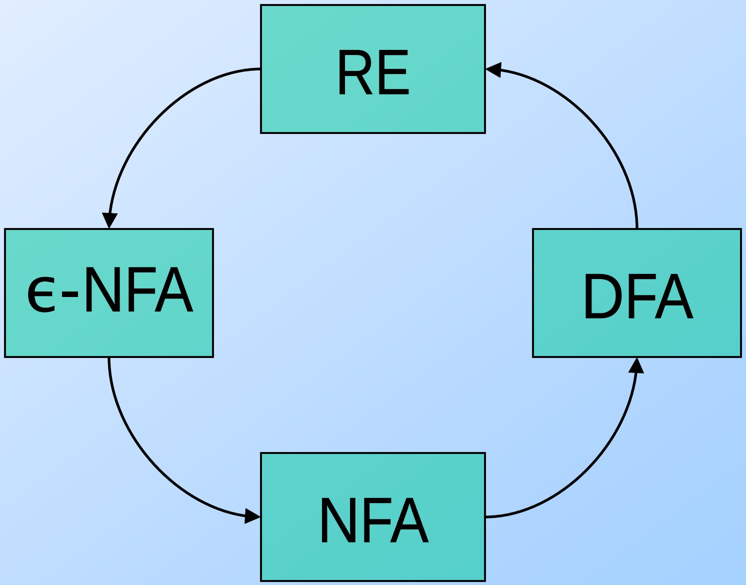


Example: Testing Membership



What if the Regular Language Is not Represented by a DFA?

- 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.
- Construct the transition graph.
- Compute the set of states reachable from the start state.
- If any final state is reachable, then yes, else no.

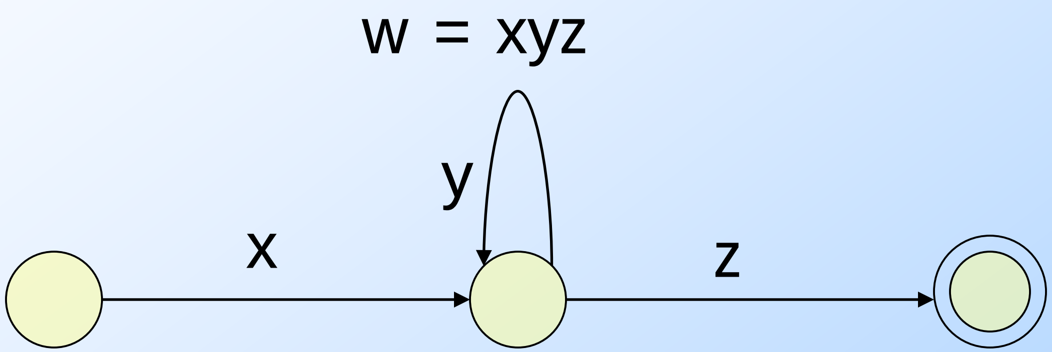
The Infiniteness Problem

- 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 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)



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

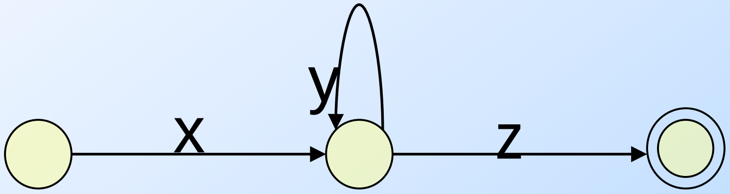
Since y is not ϵ, we see an infinite number of strings in L.

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 2nd Key Idea

- 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 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.

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.

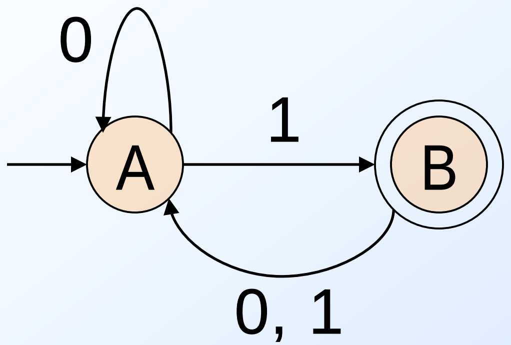
Decision Property: Equivalence

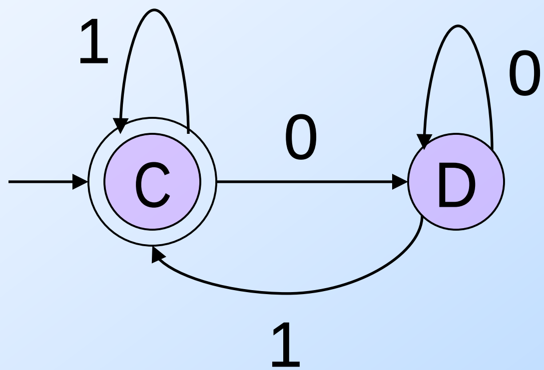
- Given regular languages L and M, is L = M?
- Algorithm involves constructing the *product DFA* from DFA's for L and M.
- Let these DFA's have sets of states Q and R, respectively.
- Product DFA has set of states $Q \times R$.
  - I.e., pairs [q, r] with q in Q, r in R.

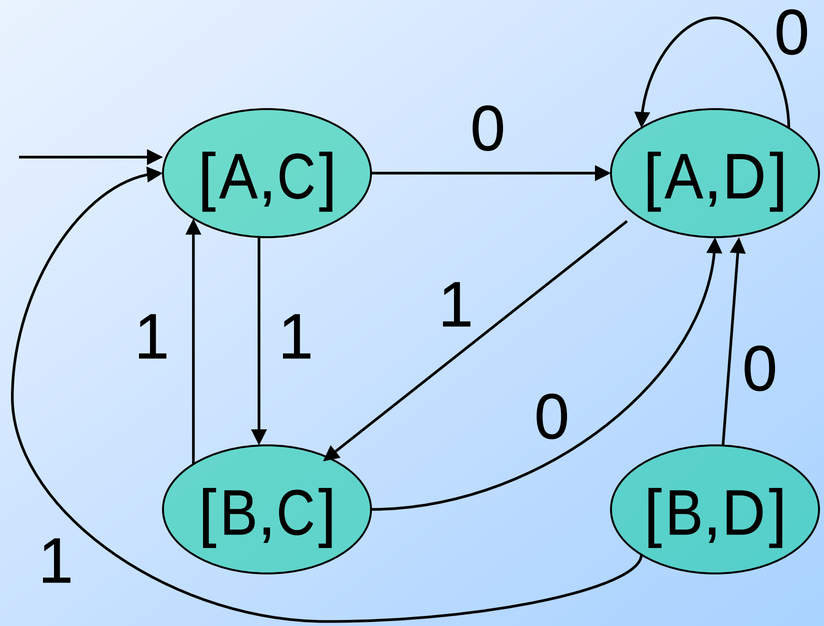
Product DFA – Continued

- 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



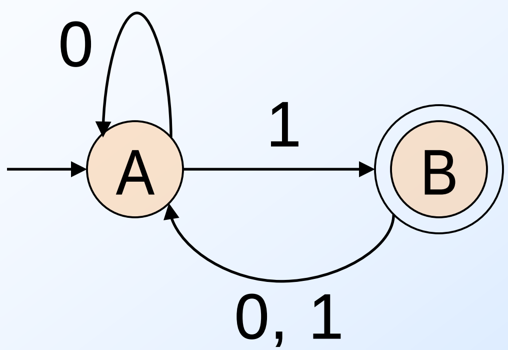


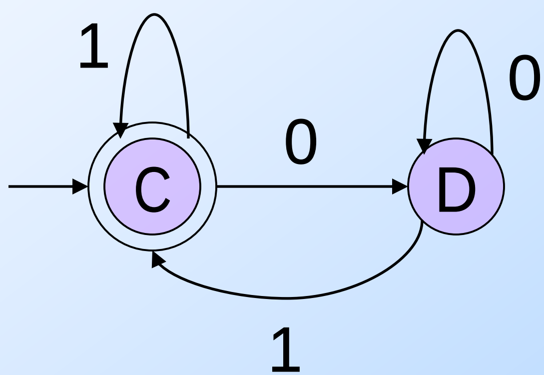


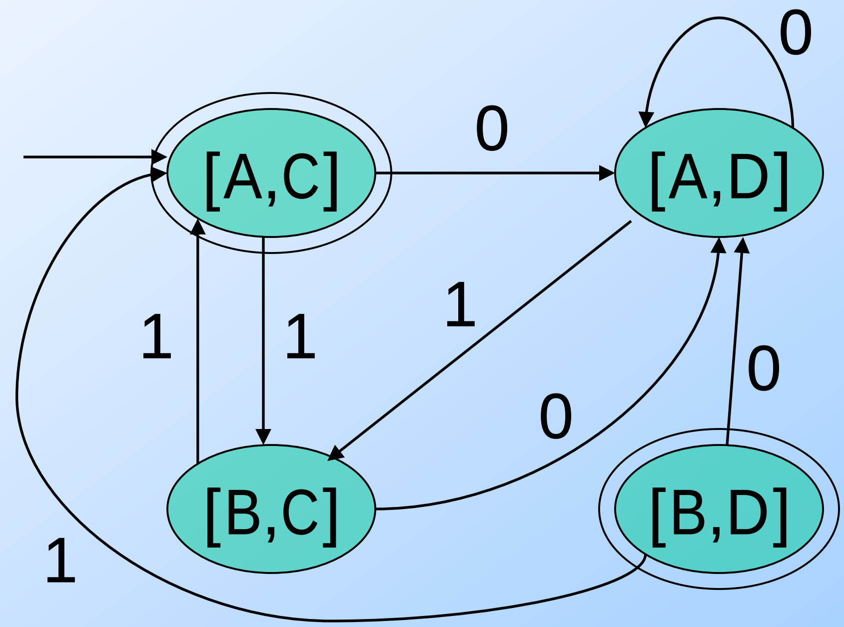
Equivalence Algorithm

- 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.

Example: Equivalence







Equivalence Algorithm – (2)

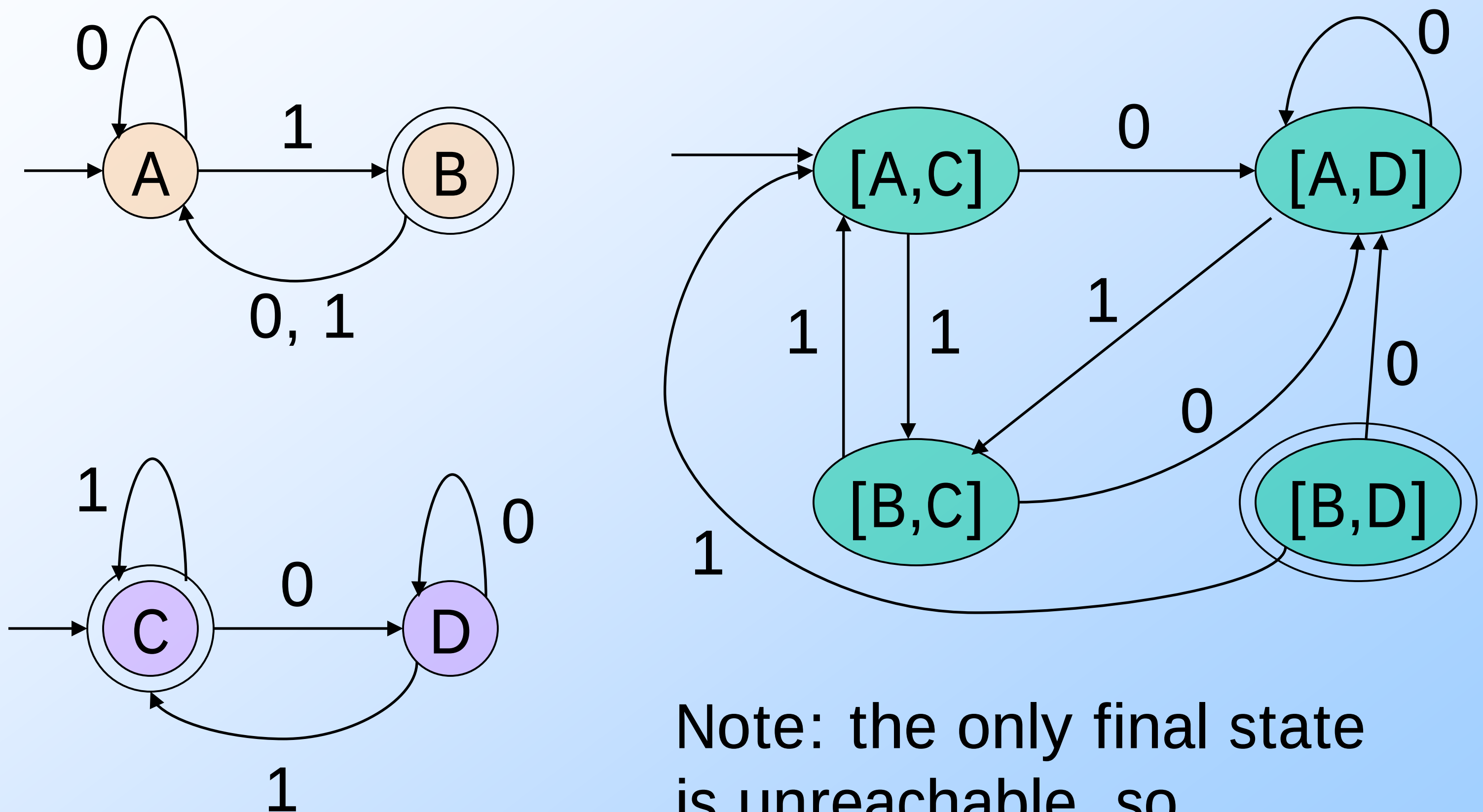
- The product DFA's language is empty iff L = M.
- But we already have an algorithm to test whether the language of a DFA is empty.

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.

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.

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.

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 $q_1,\ldots,q_k$ 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.

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 $\delta(r, a) = q$ is indistinguishable from $\delta(p, a)$.

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.

Closure Properties of Regular Languages

Union, Intersection, Difference, Concatenation, Kleene Closure, Reversal, Homomorphism, Inverse Homomorphism

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 $L \cup M$.

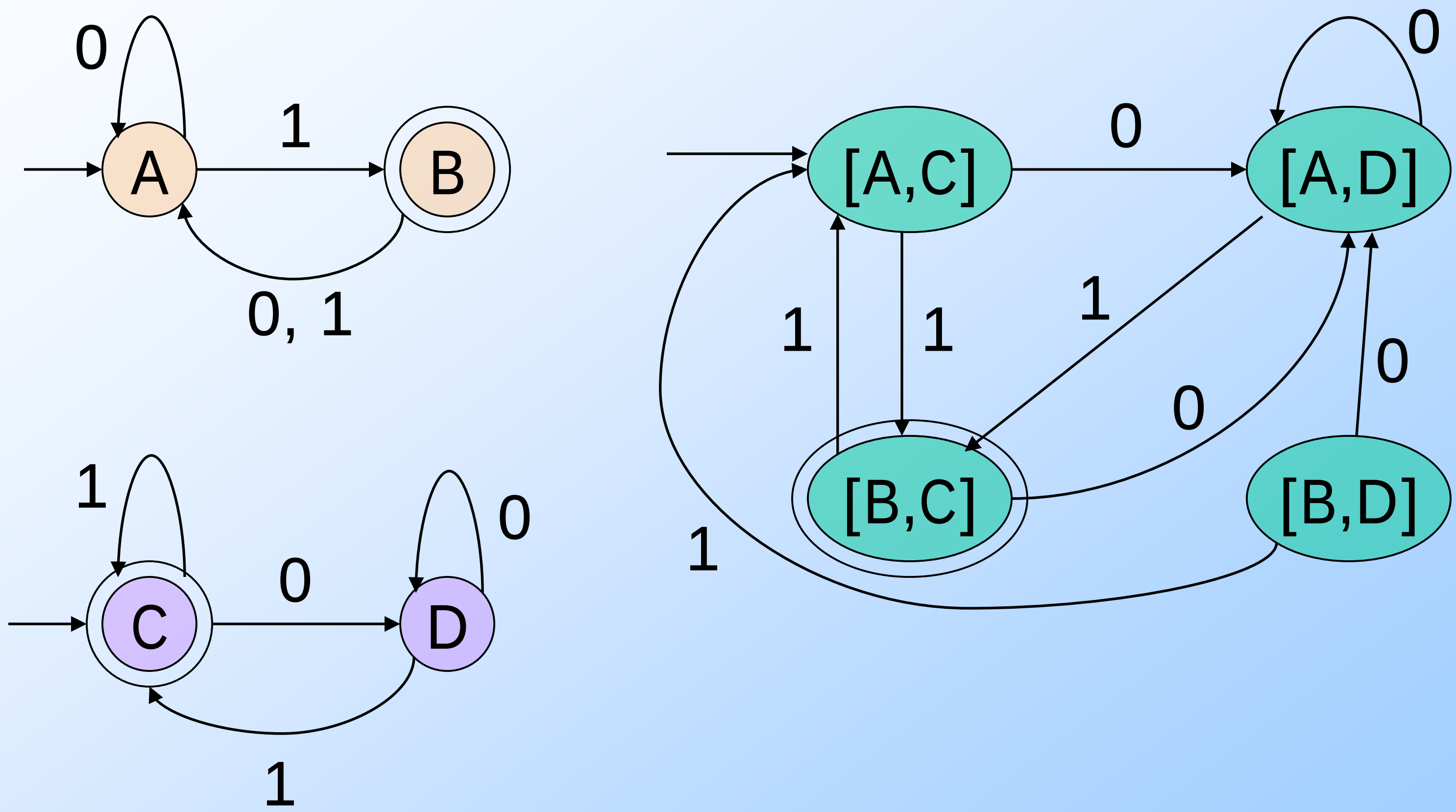
Closure Under Concatenation and Kleene Closure

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

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.

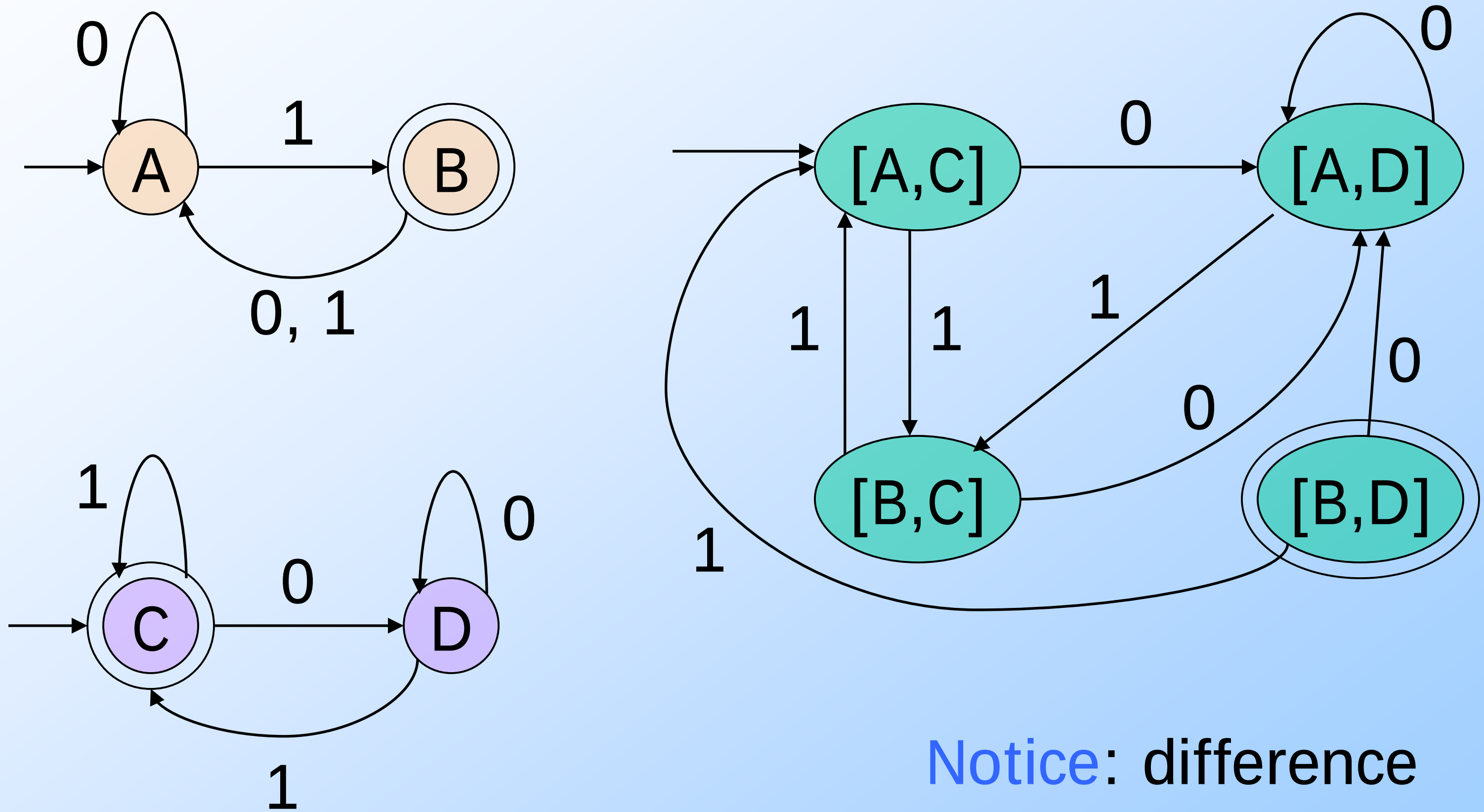
Example: Product DFA for Intersection



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 Reversal

- Recall example of a DFA that accepted the binary strings that, as integers were divisible by 23.
- We said that the language of binary strings whose reversal was divisible by 23 was also regular, but the DFA construction was very tricky.
- Good application of reversal-closure.

Closure Under Reversal – (2)

- Given language L, L^R is the set of strings whose reversal is in L.
- Example: L = {0, 01, 100};
  L^R = {0, 10, 001}.
- Proof: Let E be a regular expression for L.
- We show how to reverse E, to provide a regular expression E^R for L^R.

Reversal of a Regular Expression

- Basis: If E is a symbol a, ϵ, or ∅, then $E^R = E$.
- Induction: If E is
  - F+G, then $E^R = F^R + G^R$.
  - FG, then $E^R = G^RF^R$
  - F*, then $E^R = (F^R)^*$.

Example: Reversal of a RE

- Let E = $\mathbf{01^*} + \mathbf{10^*}$.
- $E^R = (\mathbf{01^*} + \mathbf{10^*})^R = (\mathbf{01^*})^R + (\mathbf{10^*})^R$
- $= (\mathbf{1^*})^R\mathbf{0}^R + (\mathbf{0^*})^R\mathbf{1}^R$
- $= (\mathbf{1}^R)^*\mathbf{0} + (\mathbf{0}^R)^*\mathbf{1}$
- $= \mathbf{1^*0} + \mathbf{0^*1}$.

Homomorphisms

- A *homomorphism* on an alphabet is a function that gives a string for each symbol in that alphabet.
- Example: $h(0) = ab$; $h(1) = \epsilon$.
- Extend to strings by $h(a_1 \ldots a_n) = h(a_1) \ldots h(a_n)$.
- Example: $h(01010) = ababab$.

Closure Under Homomorphism

- If L is a regular language, and h is a homomorphism on its alphabet, then h(L) = {h(w) | w is in L} is also a regular language.
- Proof: Let E be a regular expression for L.
- Apply h to each symbol in E.
- Language of resulting RE is h(L).

Example: Closure under Homomorphism

- Let $h(0) = ab$; $h(1) = \epsilon$.
- Let L be the language of regular expression $\mathbf{01}^* + \mathbf{10}^*$.
- Then h(L) is the language of regular expression $\mathbf{ab}\epsilon^* + \epsilon(\mathbf{ab})^*$.

Note: use parentheses to enforce the proper grouping.

Example – Continued

- $\mathbf{ab}\epsilon^* + \epsilon(\mathbf{ab})^*$ can be simplified.
- $\epsilon^* = \epsilon$, so $\mathbf{ab}\epsilon^* = \mathbf{ab}\epsilon$.
- ϵ is the identity under concatenation.
  - That is, $\epsilon E = E\epsilon = E$ for any RE *E*.
- Thus, $\mathbf{ab}\epsilon^* + \epsilon(\mathbf{ab})^* = \mathbf{ab}\epsilon + \epsilon(\mathbf{ab})^* = \mathbf{ab} + (\mathbf{ab})^*$.
- Finally, $L(\mathbf{ab})$ is contained in $L((\mathbf{ab})^*)$, so a RE for $h(L)$ is $(\mathbf{ab})^*$.

Inverse Homomorphisms

- Let h be a homomorphism and L a language whose alphabet is the output language of h.
- $h^{-1}(L) = \{w \mid h(w) \text{ is in } L\}$.

Example: Inverse Homomorphism

- Let $h(0) = ab$; $h(1) = \epsilon$.
- Let $L = \{abab, baba\}$.
- $h^{-1}(L)$ = the language with two 0's and any number of 1's = L(**1*01*01***).

Notice: no string maps to baba; any string with exactly two 0's maps to abab.

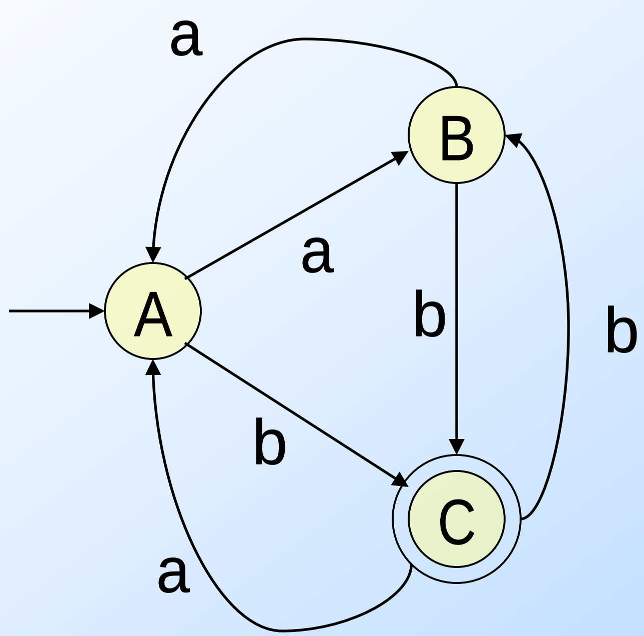
Closure Proof for Inverse Homomorphism

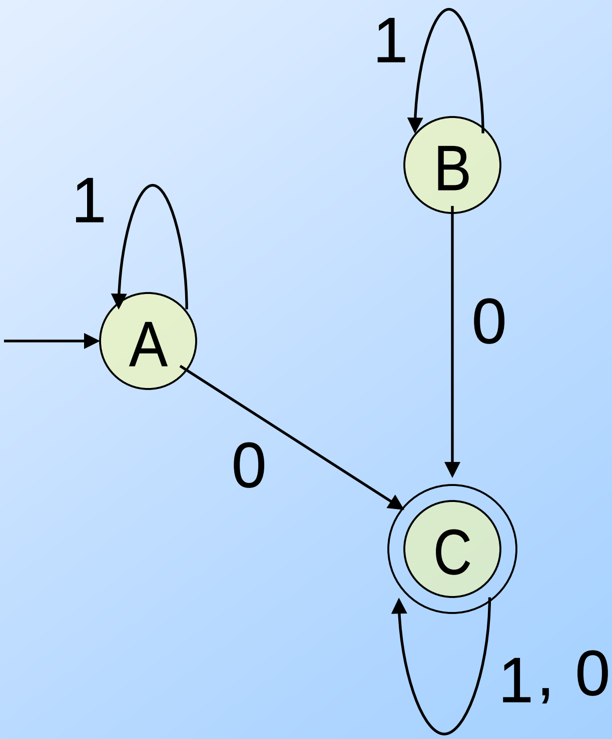
- Start with a DFA A for L.
- Construct a DFA B for $h^{-1}(L)$ with:
  - The same set of states.
  - The same start state.
  - The same final states.
  - Input alphabet = the symbols to which homomorphism h applies.

Proof – (2)

- The transitions for B are computed by applying h to an input symbol *a* and seeing where A would go on sequence of input symbols h(a).
- Formally, $\delta_B(q, a) = \delta_A(q, h(a))$.

Example: Inverse Homomorphism Construction





Since
$h(1) = \epsilon$

Since
$h(0) = ab$

$h(0) = ab$
$h(1) = \epsilon$

Proof – (3)

- Induction on |w| shows that $\delta_B(q_0, w) = \delta_A(q_0, h(w))$.
- Basis: w = ϵ.
- $\delta_B(q_0, \epsilon) = q_0$, and $\delta_A(q_0, h(\epsilon)) = \delta_A(q_0, \epsilon) = q_0$.

Proof – (4)

- Induction: Let w = xa; assume IH for x.
- $\delta_B(q_0, w) = \delta_B(\delta_B(q_0, x), a)$.
- $= \delta_B(\delta_A(q_0, h(x)), a)$ by the IH.
- $= \delta_A(\delta_A(q_0, h(x)), h(a))$ by definition of the DFA B.
- $= \delta_A(q_0, h(x)h(a))$ by definition of the extended delta.
- $= \delta_A(q_0, h(w))$ by def. of homomorphism.