CS 3313 Foundations of Computing:

Properties of Regular Languages

http://gw-cs3313-2021.github.io

Properties of Language Classes

- A *language class* is a set of languages.
  - Example: the regular languages.
  - Example: context free 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 closed under union, concatenation, and (Kleene) closure.
  - Use the RE representation of languages.

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?
- Example: Is L(M1) = L(M2) ?
- Example: Does L(M) halt on all inputs w ?

Properties of Regular Languages

- Closure Properties: what happens when we "combine" two regular languages or perform set operations on them ?
  - Ex: Is Intersection of two regular languages still a regular language ?
  - Why is this important ?
    - Construct a larger set from smaller sets
    - Problem decomposition
- Decision Properties: can we provide procedures to determine properties of a language ?
  - Ex: are two machines equivalent? Does a DFA accept an infinite set ?
- How do we determine if a language does not belong to that class of languages ?
  - Ex: How do we show that a language (problem?) cannot be accepted by a DFA ?

Closure Properties

- Theorem: states that if L_1 and L_2 are regular languages, so are the languages that result from the following operations:
  - $L_1 \cup L_2$
  - $L_1 \cap L_2$
  - L_1L_2
  - $\overline{L_1}$
  - L_1^*
- In other words, the family of regular languages is closed under union, intersection, concatenation, complementation, and star-closure.

Proof of the Closure Properties

- Since L_1 and L_2 are regular languages, there exist regular expressions r_1 and r_2 to describe L_1 and L_2, respectively
- The union of L_1 and L_2 can be denoted by the regular expression $r_1 + r_2$
- The concatenation of L_1 and L_2 can be denoted by the regular expression r_1r_2
- The star-closure of L_1 can be denoted by the regular expression r_1*
- Therefore, the union, concatenation, and star-closure of arbitrary regular languages are also regular

Constructive Proofs

- Sometimes we need a constructive proof that will provide the basis for an algorithm to automate the construction
  - Ex: What is the DFA that accepts the union of two regular languages

- We provide constructive proofs for Complement, Reversal, and Intersection

Proof: Closure under Complementation

- Theorem: If L_1 is regular then complement of L_1 is regular.
- Proof: Since L_1 is regular there is a DFA $M=(Q,\mathbf{\Sigma}, \delta,q_0, F)$ such that $L_1= L(M)$.
- Construct DFA $M'= (Q',\mathbf{\Sigma}, \delta',q_0, F')$ such that

  $$L(M')= \{w \mid w \text{ is not in } L(M) \}$$

- From definition of DFA M, a string w is in L(M) (accepted by M) if $\delta(q_0,w)$ is in F and a string *x* is not in L(M) if $\delta(q_0,x)$ is in (Q –F).
- Therefore construct M' where
- $Q' = Q$, $\delta' = \delta$,$q_0 = q_0$, $F' = (Q-F)$
  - M' has the same states, alphabet, transition function, and start state as M
  - The final states in M become non-final states in M', while the non-final states in M become final states in M
- By definition of M', a string x is in L(M') if $\delta(q_0,x)$ is in (Q –F),

  i.e., x is not in L(M).

Therefore L(M') is regular and $L(M')= \overline{L_1}$

Closure under reversal

- Theorem: If L is regular then L^R is regular.
- Proof: Since L_1 is regular there is a DFA $M=(Q,\Sigma, \delta,q_0, F)$ such that $L= L(M)$.
- Construct NFA $N= (Q',\Sigma, \delta',p_0, F')$ such that

$$L(N)= \{w \mid w^R \text{ is in } L(M) \}$$

- Homework 1 !!!
- Key ideas:
  - Start state is a new state p_0 and add empty string transitions to all the final states in M
  - $F' = \{q0\}$ $Q' = Q \cup \{p_0\}$
  - $\delta'(p,a) = q$ where $\delta(q,a) = p$ reverse the direction of the edge!

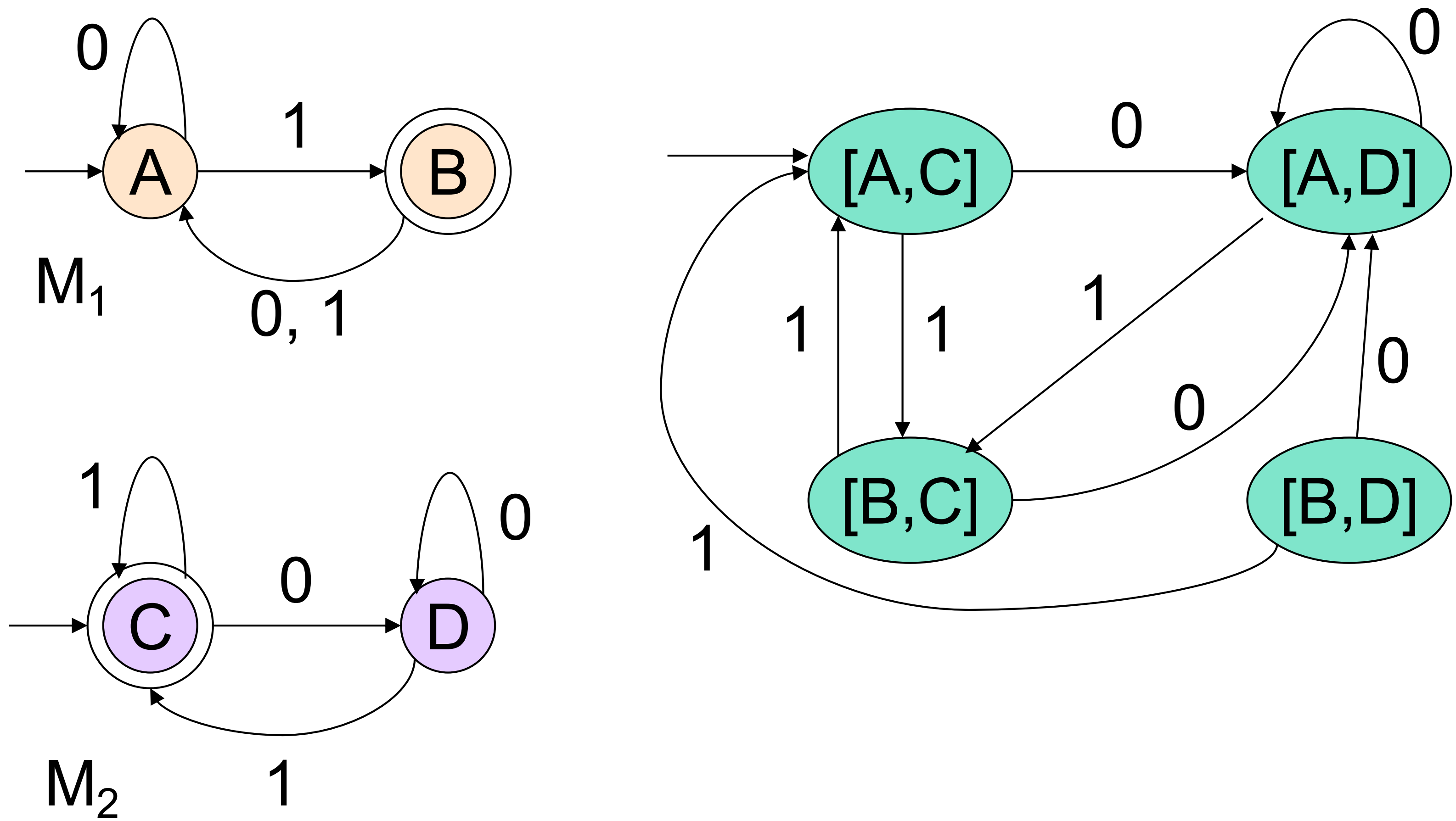
Closure under Intersection

- Theorem: If L_1 and L_2 are regular then $L_1 \cap L_2$ is regular.
- Non-constructive proof: Use closure under complement and union and DeMorgan's laws $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$
- Constructive Proof: Design a DFA that accepts the intersection.
- If L_1 and L_2 are regular, then there are DFAs M_1 and M_2 that accept L_1 and L_2 respectively.
- Use these to construct a "product DFA"

Definition: Product DFA

- "compose" two DFAs using cartesian product of their states
- Let M_1 and M_2 be two DFAs with states Q and R
  - $M_1 = (Q, \Sigma, \delta_1, q_0, F_1)$ and $M_1 = (R, \Sigma, \delta_2, r_0, F_2)$
- Product DFA M_p:
- Product DFA has set of states $Q \times R$
  - i.e., pairs *[q,r]* with q in Q and r in R
- Start state = $[q_0, r_0]$ (the start states of the two DFA's).
- Transitions: $\delta([q,r], a) = [\delta_1(q,a), \delta_2(r,a)]$
  - δ_1, δ_2 are the transition functions for the DFA's of M_1, M_2
  - That is, *we simulate the two DFA's in the two state components of the product DFA.*
- Note: we have not yet defined the final states of the product DFA

Example: Product DFA



Closure under Intersection

- Theorem: If L_1 and L_2 are regular then $L_1 \cap L_2$ is regular.
- Proof: If L_1 and L_2 are regular, then there are DFAs M_1 and M_2 that accept L_1 and L_2 respectively.
- construct the "product DFA" M
- To complete the proof, define the final states of the product DFA
  - How ?
  - Input w is accepted by product DFA M if it is accepted by both M1 and M2
  - So product DFA M is in final state if both M1 and M2 are in a final state

Closure under Difference

- Theorem: If L_1 and L_2 are regular then $L_1 - L_2$ is regular.
- Proof: Construct product DFA M from the two DFAs $M_1 = (Q, \mathbf{\Sigma}, \delta_{1,} q_0, F_1)$ and $M_1 = (R, \mathbf{\Sigma}, \delta_{2,} r_0, F_2)$
- We want a string w to be accepted by M if w is in L_1 and w is not in L_2
- w is in L_1 iff $\delta_1(q_0,w)$ is in F_1
- w is in L_2 iff $\delta_2(r_0,w)$ is not in F_2

- So how would you define F?

Closure under 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) = ϵ.
- Let L be the language of regular expression **01*** + **10***.
- Then h(L) is the language of regular expression

  abϵ* + ϵ(**ab**)*.

Note: use parentheses to enforce the proper grouping.

Alternate Proof: Closure under reversal

- What if the language was specified using a regular expression – can we find a regular expression for the reversal ?
- Given regular expression E for language L, derive regular expression E^R for L^R

Reversal of a Regular Expression

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

Example: Reversal of a RE

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

Examples: Applying closure properties

- L1={ w | w has a's followed by b's}
- L2={ w | w has even length}
- L3 = { w | w has odd number of a's and even number of b's}
- **If L1,L2, L3 are regular then:**
- L1 ∪ L2 = {w | w has a's followed by b's or w has even length} is regular
- L1 ∩ L3 = { w | w has odd number of a's followed by even number of b's} is regular
- $\overline{L1}$ = {w| w does not have a's followed by b's } is regular
- L = L1 ∩ $\overline{L3}$ = {w| w has a's followed by b's and not (a is odd and b is even) } is regular

Summary of Closure Properties

- Regular languages are closed under Union, Concatenation, star closure, complementation, reversal, intersection, homomorphism (and reverse homomorphisms)

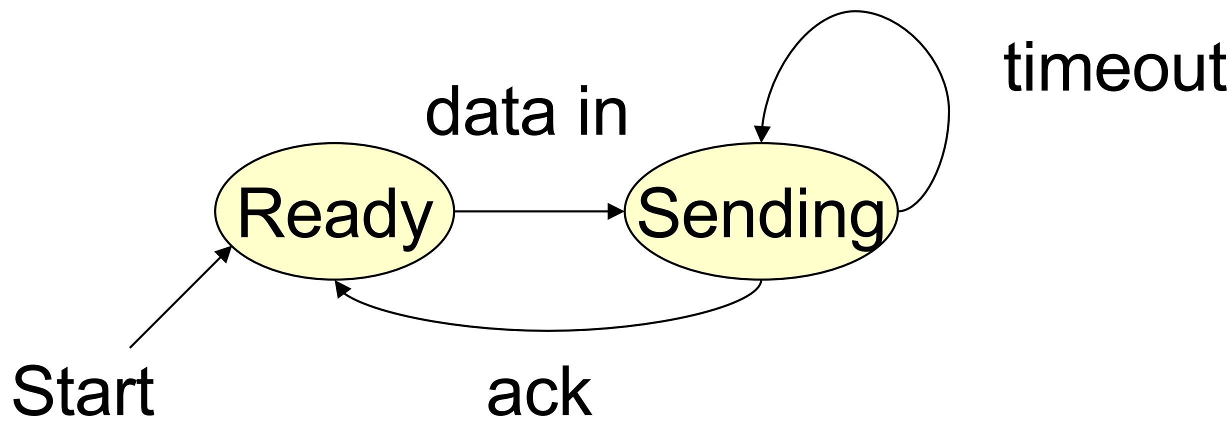
- Where are closure properties used ?
  - Construction a solution (DFA or Reg. Expr.) for a larger language using simpler solutions (machines or languages)
    - Analogy: modular composition of software modules
  - Useful in simplifying proofs to show a language is not regular
  - Useful in constructing "decision algorithms"

Decision Properties of Regular Languages

- 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?

Example: Protocol for Sending Data

Protocols are typically modeled as a DFA



- Protocol is meant to never terminate – i.e, run forever if no errors
- Missing transitions:
  - ack or timeout signal in Ready state…okay to ignore
  - Data-in signal in sending state is an indication of an error
    - So go to an error state (dead state?)

Why Decision Properties?

- Think about DFA's representing network 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?"
- Theorem: Membership in Regular Languages is decidable.
- Proof:
  - Assume L is represented by a DFA A.
  - Simulate the action of A on the sequence of input symbols forming w.
  - DFA makes n moves where n is length of string w.
- Alternate Proof: Consider the transition graph of DFA
  - Is there a path from start state to a final state labeled w
  - If n is length of w, then this takes time O(n)

The Emptiness Problem

- Given a regular language, does the language contain any string at all? i.e., is L(M) = ∅ ?
- Proof: Assume representation is transition graph of the DFA.
  - Compute the set of states reachable from the start state.
  - If at least one final state is reachable, then not empty, else L(M) is empty.

- Note: our proofs/decision algorithmsn use graph algorithms to construct the solution.
  - Finding paths in a graph – breadth first search, all-pairs paths, Djikstra, etc.
    - For now, don’t focus on the time complexity of the algorithm….you will delve into this in the algorithms course

Algorithm to test emptiness of L(M)

- Input: Transition graph of DFA M
- Output: Yes if L(M) is empty, else NO

```
EMPTY := Yes
For each q in F
        { if there is a path from start state q0 to q
                then empty:= NO
        }
return EMPTY
```

Decision Property: Equivalence

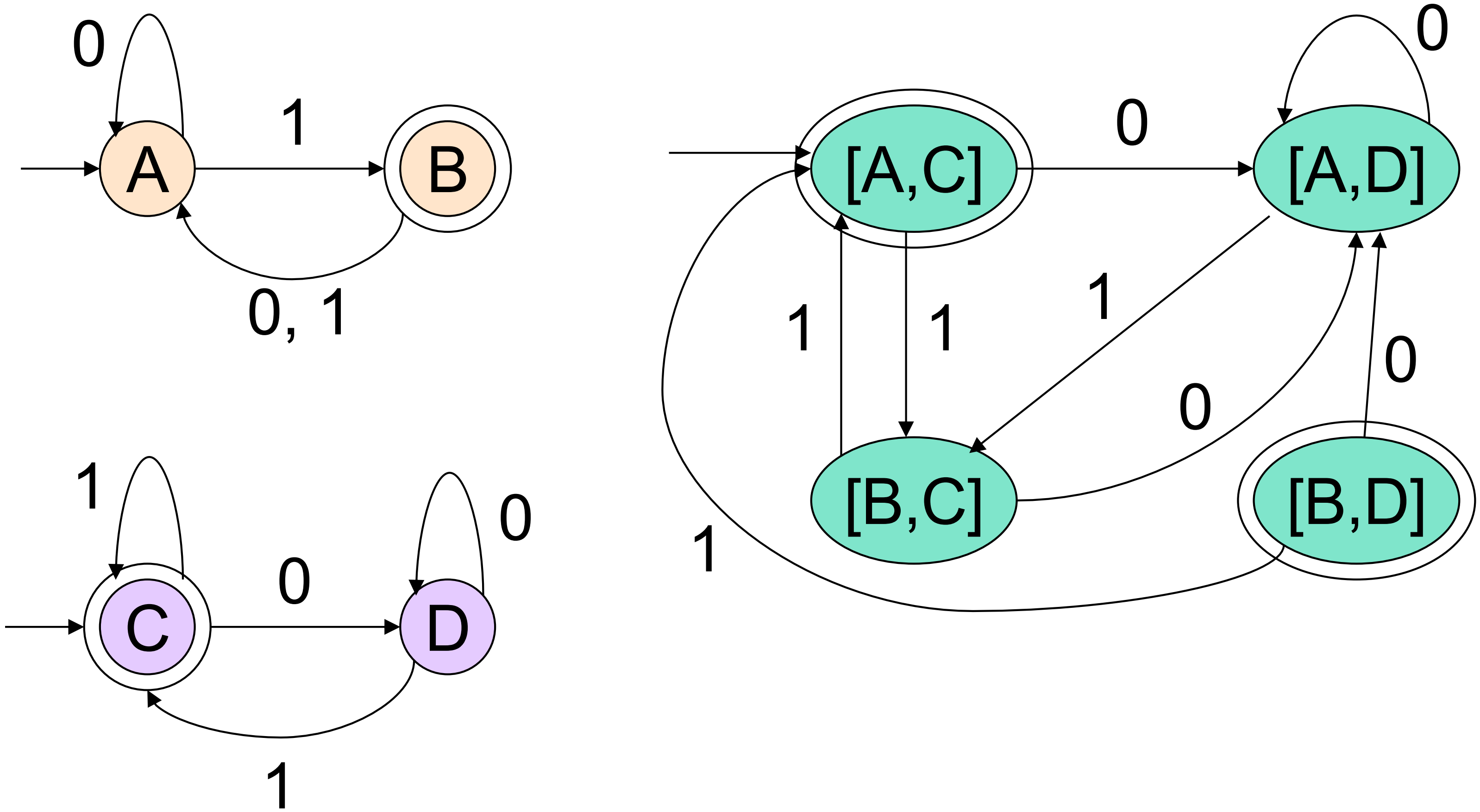
- Given regular languages L_1 and L_2, is $L_1 = L_2$?
- Theorem: Equivalence of regular languages is decidable.
- Proof: Algorithm involves constructing the *product DFA* from DFA's for L_1 and L_2.

- Note: the two languages are not equal if there is a string *w* that is accepted by one language but not the other.
  - $w \in L_1$ and $w \notin L_2$ OR $w \in L_2$ and $w \notin L_1$

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_1 and L_2.
- $L_1 = L_2$ if and only if the product automaton's language is empty.

Example: Equivalence

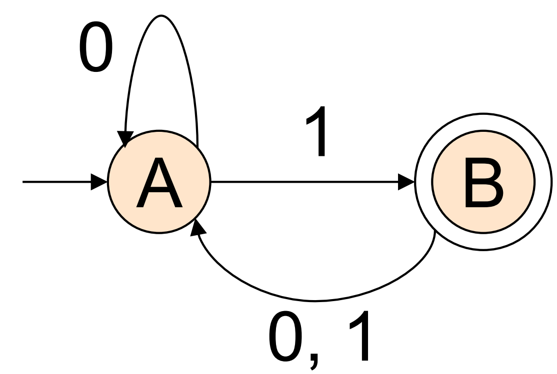


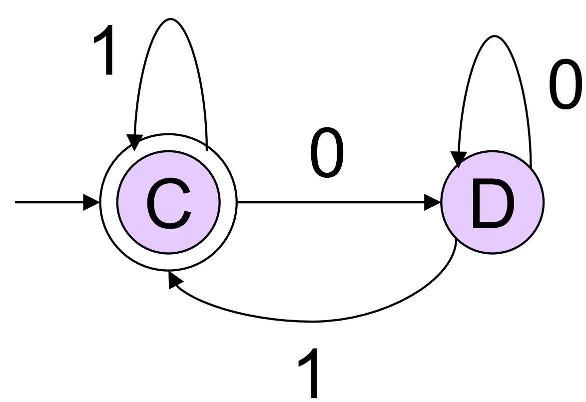
- B is final state of M_1 and C is final state in M_2
  - Therefore [A,C] and [B,D] are final states in product automaton

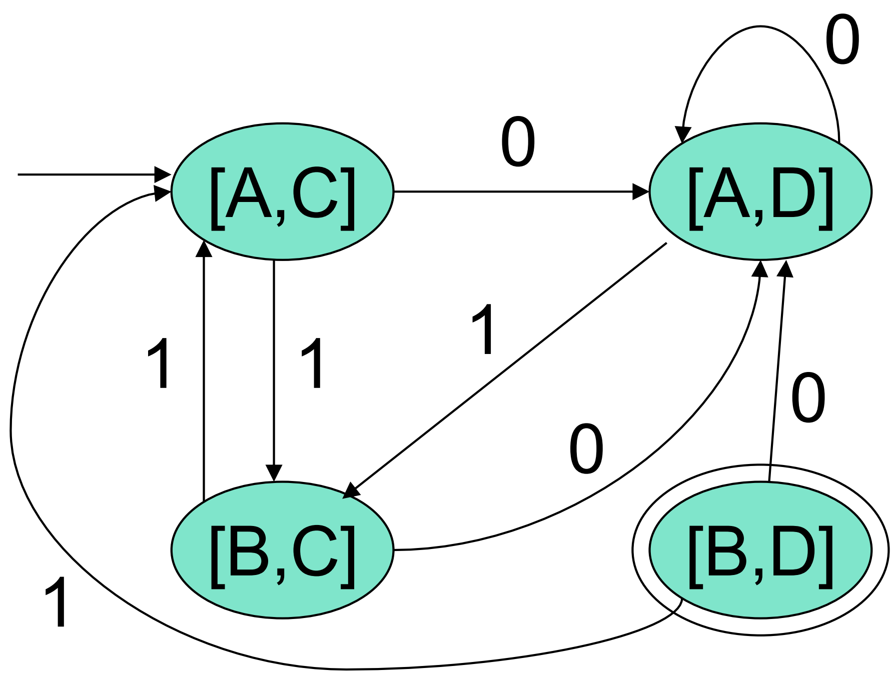
Decision Property: Containment

- Given regular languages L_1 and L_2, is $L_1 \subseteq L_2$?
- Theorem: Containment property is decidable.
- Proof: Algorithm also uses the product automaton.
- How do you define the final states [q, r] of the product so its language is empty iff $L_1 \subseteq L_2$?
  - i.e., there is no string w, such that $w \in L_1$ and $w \notin L_2$
  - [q,r] is final state if q is final and r is not

Example: Containment







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

The Infiniteness Problem

- Is a given regular language infinite?
- Theorem: Testing if L(M) is infinite is a decidable problem.
- 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.

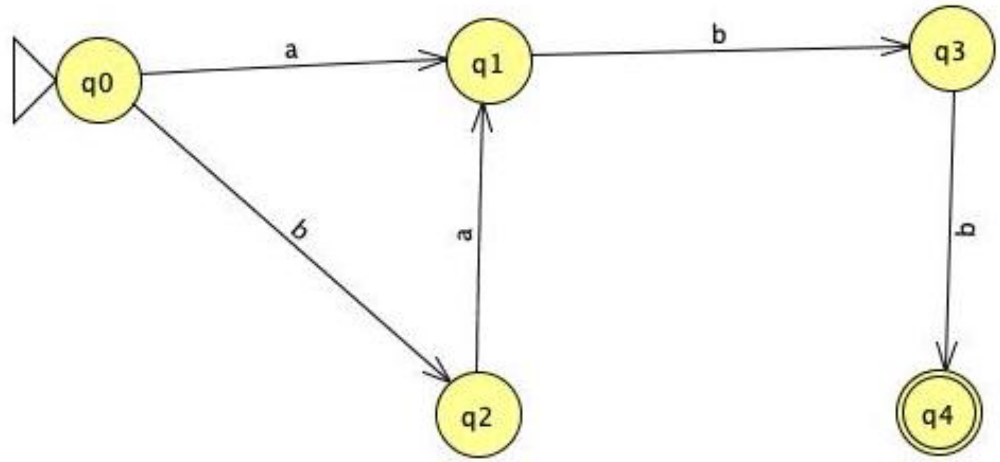
L(M) infinite ?

- Proof: use the graph representation to present the procedure/proof.

Transition graphs for two DFAs

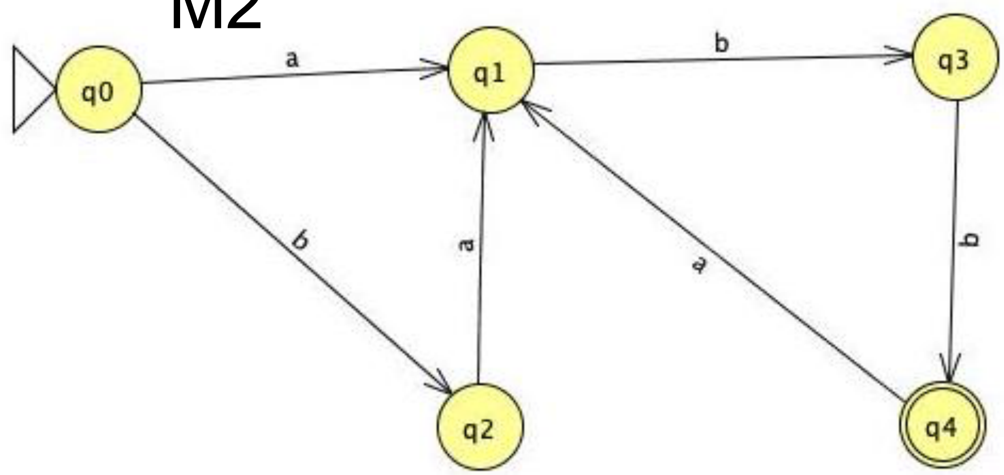
Is L(M1) finite ?

M1



Is L(M2) finite ?

M2



Algorithm to test for L(M) infinite

- Input: Transition graph for DFA M
- Output: Yes if L(M) is infinite, No if L(M) is finite
- Algorithm ?
- Check if graph has a cycle!

So what kinds of languages are not regular and how do we prove they are not ?

- Proof for testing infiniteness of L(M) reveals some properties that can be used to prove that a language is not regular.

- Given any language L, it is either regular or it is not.
  - To prove L is regular, we have to provide a DFA/NFA or Regular expression that accepts L.
  - To prove L is not regular, we need to provide a formal proof using some properties of all regular languages
    - Simply saying “I spent a lot of time and could not find a DFA” is NOT a proof.