CS 3313
Foundations of Computing:

Properties of Regular Languages

http://gw-cs3313.github.io

1

Next….Properties of Regular Languages

- the BIG question = properties of regular languages
- What types of languages are regular?
- What happens when we combine reg. lang. using set and algebraic operations?
- How do we know if the language is not regular ?
  - How can we **prove** that a language/problem is not regular ?
- Why bother ?
  - Algorithmic thinking: we are given a problem to solve (in our case, it is framed as a language with some properties).
  - Question: What is the simplest machine model we can use to solve the problem ?
    - Translates to code efficiency (eventually!)

2

Language Classes and Common Questions on their properties

- A *language class* is a set of languages.
  - Example: the class of regular languages = set of all regular languages
    - All languages accepted by DFAs
  - Example: context free languages.
- Language classes have two important kinds of properties:
  1. Closure properties – what happens when we combine languages using the various (set) operations ?
  2. Decision properties – algorithms that can determine if a language/DFA has a specific property

3

3

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:
  - if we complement a regular language then is the result a regular language ?
  - If we complement a C program then is the result a C program ?
  - If we have a machine model (DFA, PDA, etc.) to solve a problem P, then is there a machine (same machine model) to solve the complement of the problem ?

4

4

Properties of Regular Languages

- **Definition: A language is regular *iff* it is accepted by DFA M (or NFA M or regular expression r)**
- 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 more complex language/machine from simpler languages/machines
    - Problem decomposition
- Decision Problems: can we provide procedures to determine properties of a language ?
  - Ex: are two machines equivalent? Does a DFA accept an infinite set ?
- How to 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 ?

5

Exercise: Closure Properties of Regular Languages

- Question 1: If L_1 and L_2 are *any two* regular languages then prove or disprove the following

1. is $L_1 \cup L_2$ (union) a regular language ?
2. is $L_1 . L_2$ (concatenation) a regular language ?
3. is $(L_1)^*$ (Kleene/star closure) a regular language ?
4. is $(L_1)^R$ (reversal) a regular language ?

- Prove or disprove
  - To prove a language is regular, you must provide a (general) technique to construct a NFA (or DFA or Reg.Expr.) that accepts the language
- You have at your disposal all the results from lectures and homeworks !!

6

Exercise: Closure Properties of Regular Languages

- If L_1 and L_2 are regular languages then the following are regular languages
  - We have $L_1 = L(M_1) = L(r_1)$ and $L_2 = L(M_2) = L(r_2)$

1. $L_1 \cup L_2$ (union) is a regular language: Reg.Expr $r_1 + r_2$
2. $L_1 . L_2$ (concatenation) is a regular language: Reg. expr $(r_1 . r_2)$
3. $(L_1)^*$ (Kleene/star closure) is a regular language: $(r_1)^*$
4. $(L_1)^R$ (reversal) is a regular language: HW2

- Prove or disprove
  - To prove a language is regular, you must provide a (general) technique to construct a NFA (or DFA or Reg.Expr.) that accepts the language
- You have at your disposal all the results from lectures and homeworks !!

7

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 $r_1 r_2$
- 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

8

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 2 !!!
- Key ideas:
  - $F' = \{q0\}$ $Q' = Q \cup \{p_0\}$
  - Start state is a new state p_0 and add empty string transitions to all the final states in M
  - $\delta'(p,a) = q$ where $\delta(q,a) = p$ reverse the direction of the edge!

9

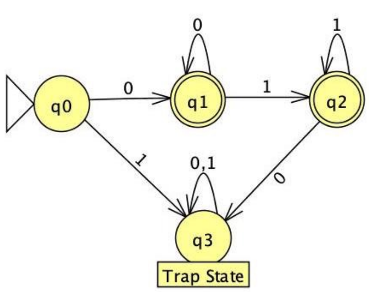
Theorem: Closure under Complementation

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

  $L_1= L(M)$.
- From definition of DFA M:

  a string w is in $L(M)$ (accepted by M) iff $\delta(q_0,w)$ is in F

  and a string x is not in $L(M)$ if $\delta(q_0,x)$ is in $(Q - F)$.



10

Proof: Closure under Complementation

- 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
- $Q' = Q$, $\delta' = \delta$,$q_0 = q_0$, $F' = (Q\text{-}F)$
  - 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* $L(M')$ **iff** $\delta(q_0,x)$ *is in* $(Q-F)$,

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

  Therefore $L(M')$ is regular and $L(M')= L_1$

11

Question: Intersection of Regular Languages

- Theorem: if L_1 and L_2 are regular languages, then the intersection $L_1 \cap L_2$ is a regular language
- Proof: ?

12

Closure under Homomorphisms

- A *homomorphism* $h: \Sigma_1 \rightarrow \Sigma_2^*$ on an alphabet is a function that gives a string for each symbol in that alphabet.
  - Homomorphisms preserve the operations on the algebra
  - $h(w_1 w_2) = h(w_1).h(w_2)$ $\quad$ $h(w_1) + h(w_2) = h(w_1) + h(w_2)$
- Example: $h:\{0,1\} \rightarrow \{a,b\}^*$ and $h(0) = ab$; $h(1) = \lambda$.
- Extend to strings by $h(a_1 \ldots a_n) = h(a_1) \ldots h(a_n)$.
- Example: h(01010) =h(0).h(1).h(0).h(1).h(0) = ababab.

- Example: *h(0) = begin* $\quad$ *h(1) = end*

L = { *w* is a binary string and has equal number of 0's and 1's}

h(L) = { *w* has an equal number of *begin* and *end* }

13

13

Closure Under Homomorphism

- Theorem: If L is a regular language, and h is a homomorphism on its alphabet, then $h(L) = \{h(w) \mid w \text{ is in } L\}$ is also a regular language.
- Proof:
  - Since L is a regular language, it is represented by a regular expression E
  - Since $h(a)$ is a string of symbols, it is a regular expression.
  - We generate regular expression E_h by applying h to each symbol in E.
- Language of resulting RE $E_h = h(L)$.

14

14

Example: Closure under Homomorphism

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

  ab λ * + λ (**ab**)*.

  Note: use parentheses to enforce the proper grouping.

- h(0) = ab   h(1)= bb and let L = (0+1)* 010 (0+1)*

  h(L) = (ab +bb)* ab bb ab (ab+bb)*

15

15

Constructive Proofs

- Sometimes we need a constructive proof that will provide the basis for an algorithm to automate the construction
  - Ex: we had constructive proofs for complementation and reversal
- 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
- Constructive Proof: Design a DFA that accepts the intersection.
- Why ?
- Example of finding disease sequence in DNA of patient – variation "find if patient has disease 1 and disease 2"
  - We want to design a DFA and use this as the algorithm

16

Product DFAs: Simulate both DFAs concurrently

- Key concept: given two DFAs (algorithms), construct a DFA (algorithm) that concurrently simulates both DFAs (algorithms) at each step (i.e., at each input read by the machine)

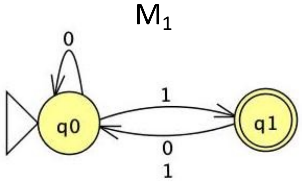
- How?
  - Keep track of the states each DFA is in by creating a corresponding single state

Product DFA

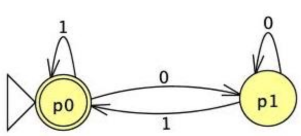
17

Example: Product DFA

M_1



M_2



1. Start both machines
2. Send input to both machines
3. Each examines current state & input
4. Makes transition based on its function & goes to next state specified in its function

18

Definition: Product DFA

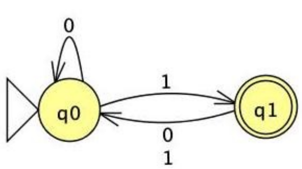
- "compose" two DFAs using cartesian product of their states
- Let M_1 and M_2 be two DFAs with states Q and R
  - $M_1 = (Q, \boldsymbol{\Sigma}, \delta_1, q_0, F_1)$ and $M_1 = (R, \boldsymbol{\Sigma}, \delta_2, r_0, F_2)$
- Product DFA M_p: $(Q_p, \boldsymbol{\Sigma}, \delta_p, p_0, F_p)$
- Product DFA has set of states $Q_p = Q \times R$
  - i.e., ordered pairs $[q,r]$ with q *in* Q and r *in* R
- Start state $p_0 = [q_0, r_0]$ (the start states of the two DFA's).
- Transitions: $\delta_p([q,r], a) = [\delta_1(q,a), \delta_2(r,a)]$
  - 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

19

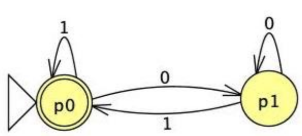
19

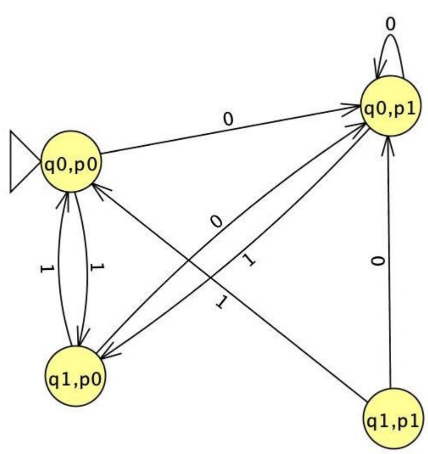
Example: Product DFA

M_1



M_2





20

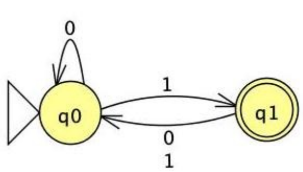
Closure under Intersection

- Theorem: If L_1 and L_2 are regular then $L_1 \cap L_2$ is regular and there is a DFA M that accepts the intersection.
- 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.
  - $M_1 = (Q, \boldsymbol{\Sigma}, \delta_1, q_0, F_1)$ and $M_1 = (R, \boldsymbol{\Sigma}, \delta_2, r_0, F_2)$
- Next, construct the product DFA M_p: $(Q_p, \boldsymbol{\Sigma}, \delta_p, p_0, F_p)$
- 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
  - Therefore M1 and M2 are in a final state
  - Therefore ……

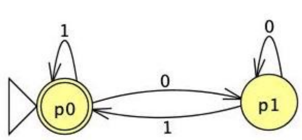
21

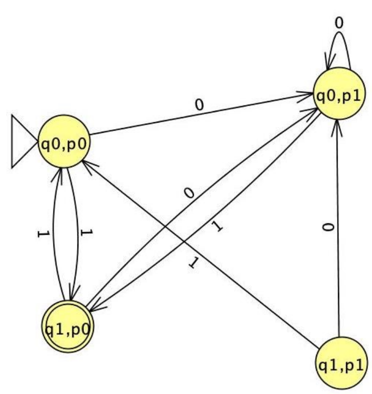
Example: Product DFA for Intersection

M_1



M_2





22

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.
- 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
  - Therefore construct product DFA M_p
  - So product DFA M is in final state if both M1 and M2 are in a final state
  - Therefore $F_p = F_1 \times F_2$

23

Closure under Set Difference

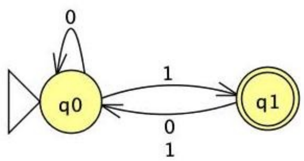
- DNA sequence example: patient has disease L_1 but not disease L_2
- Theorem: If L_1 and L_2 are regular then $L_1 - L_2$ is regular.
- Proof: Construct product DFA M from the two DFAs $M_1 = (Q, \boldsymbol{\Sigma}, \delta_1, q_0, F_1)$ and $M_1 = (R, \boldsymbol{\Sigma}, \delta_2, r_0, F_2)$
- We 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?

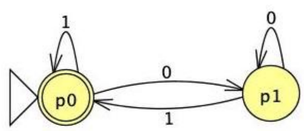
24

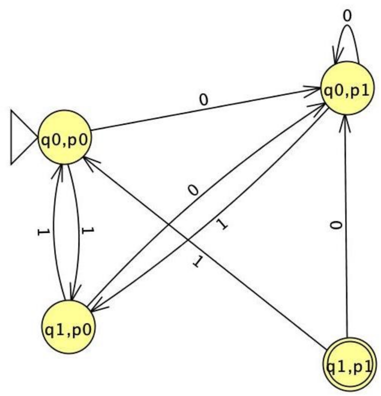
Example: Product DFA for Set Difference

M_1



M_2





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 =
- L1 ∩ L3 =
- $\overline{L1}$ =
- L = L1 ∩ $\overline{L3}$ =

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

27

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”

28

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 determines whether or not some property holds

  a property ***P*** is *decidable* if there is an *algorithm* to check the property

- Examples:
  - Is language L empty?
  - Is L(M1) = L(M2) ? (Are two machines equivalent)
    - If we view M as an algorithm, then "are two programs equivalent"
  - Does L(M) halt on all inputs w ?
    - Is there a bug that causes an infinite loop for some values of inputs ?
  - Is P a valid C program ?
    - This is asking if the syntax is correct…it is not asking for the code to be generated

29

29

Quick Review: Properties of Algorithms

Algorithm must have these properties if the "machine" is to execute it without human intervention:

- **Input specified** (Type of data expected: numbers? Strings? Letters? Alphabet?)
- **Output specified** (Types of data forming the result)
- **Definiteness**: be explicit about how to realize the computation
  - Sequence of commands (steps) that state unambiguously what to do
    - Ex: If (input == 0) then go to step 2
- **Effectiveness** ensures machine can perform operation without human intervention – each step is from primitive operations of the machine
    - Ex: machine code on a computer; *transitions in DFA*,….
- Finiteness – **must terminate and description of algorithm is finite**

- Note: this is still an informal definition of an algorithm…a mathematical equivalent will be defined later – a Turing machine!

30

Decision Problem vs Optimization problem

- Decision Problem: Is there a path of length k from p to q in a graph $G=(V,E)$
  - Answer is always a Yes or No
- Optimization (version) problem: Find the shortest path from p to q in graph $G=(V,E)$
  - Answer is the length of the path (we don't know the answer apriori)

- It may seem like decision problems are "simpler"…..in terms of the difficulty of solving a problem, they are the similar !
  - If you had an algorithm to solve the decision problem (is there a path of length *k*), can you use it to design an algorithm to find shortest path ?

31

Decision Problem vs Optimization problem

- Decision Problem: Is there a path of length k from p to q in a graph $G=(V,E)$
  - Answer is always a Yes or No
- Optimization (version) problem: Find the shortest path from p to q in graph $G=(V,E)$
  - Answer is the length of the path (we don't know the answer apriori)

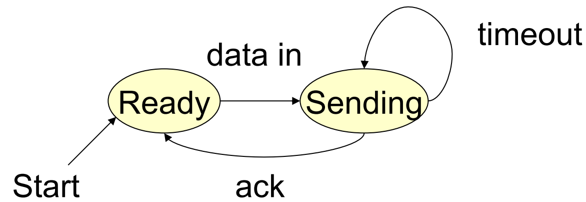
- It may seem like decision problems are "simpler"…..in terms of the difficulty of solving a problem, they are the similar !
  - If you had an algorithm to solve the decision problem (is there a path of length *k*), can you use it to design an algorithm to find shortest path ?

```
    while (  i < N and Found=NO ) /* N is number of vertices in
the graph */
            Found ="Is there a path of length i from p to q"
            i++
```

32

Example: Protocol for Sending Data

(network) 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?)

33

33

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.

34

34

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?" then we cannot check if two DFAs are equivalent

  we cannot check if the minimum state DFA is correct!

35

35

Key concept…Graph Theory

- number of our proofs/decision algorithms use graph theory to construct the solution to the decision problem
  - DFAs can be represented as a transition graph (a directed graph)
- Algorithms for finding paths in a graph
  - Between a specific pair of vertices
  - Between all pairs of vertices
  - Find shortest path
  - Determine if there is a cycle in the graph
- Lab tomorrow will summarize a simple algorithm for answering these questions
  - More efficient (actual!) algorithms covered in algorithms course
  - *We assume for now that these algorithms exist*

36

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 M.
  - Simulate the action of M on the sequence of input symbols forming w.
  - DFA makes *n* moves where *n* is length of string w – *therefore it halts after n steps*
- Alternate Proof: Consider the transition graph of DFA
  - Is there a path from start state q_0 to some final state labeled w
  - Simple algorithm using adjacency matrix to represent a graph

37

37

The Emptiness Problem

- Given a regular language, does the language contain any string at all? i.e., is $L(M) = \emptyset$?
- 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.

38

38

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
```

39

Decision Property: Equivalence

- Given regular languages L_1 and L_2, is $L_1 = L_2$?
  - This is equivalent to testing if two DFAs are equivalent
- Theorem: Equivalence of regular languages is decidable.
- Proof: Algorithm involves constructing the *product DFA* from DFA's for L_1 and L_2.
  - *Combine our proofs from closure properties and decision properties !*
- Note: the two languages are <u>not equal</u> 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 $w \notin L_1$

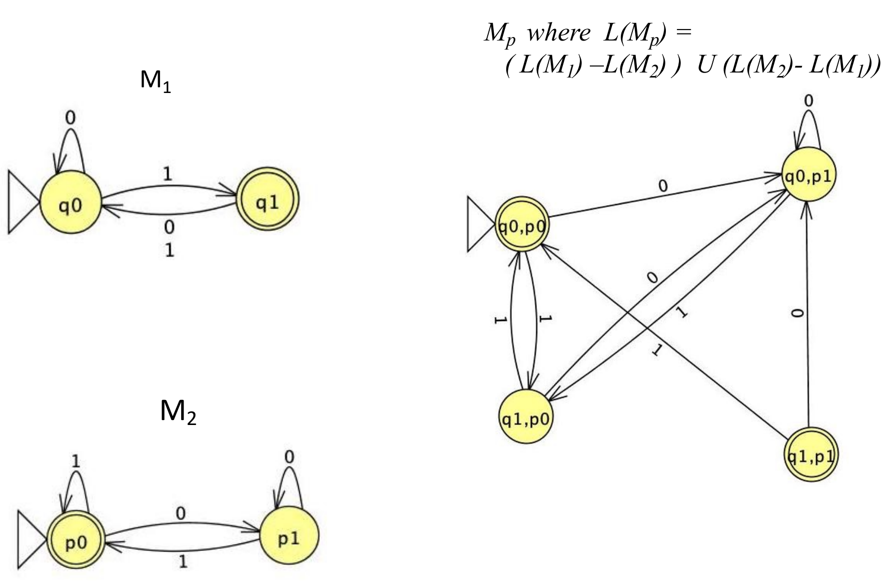
40

40

Equivalence Testing Algorithm

- Construct Product DFA
  - 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
- Call Emptiness testing algorithm with this product DFA as input

Example: Product DFA for Equivalence Testing



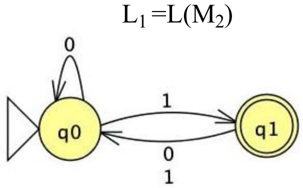
Decision Property: Containment

- 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

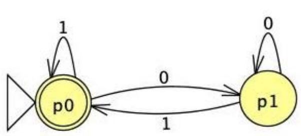
Example: Product DFA for Subset Checking

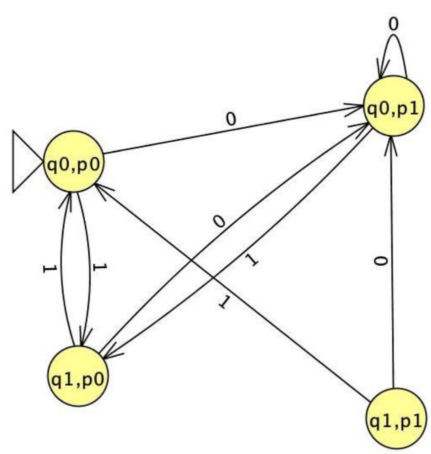
L_1 is subset of L_2 iff no string w such that w accepted by L_1 and not accepted by L_2

L_1 =L(M_2)



L_2 = L(M_2)





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
- Algorithm: Construct this product DFA and call the emptiness testing algorithm

  if product DFA is empty then L_1 is a subset of L_2

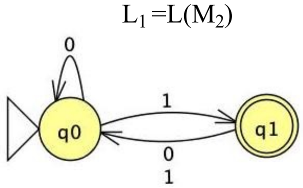
45

45

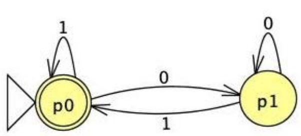
Answer: Product DFA for Subset Checking

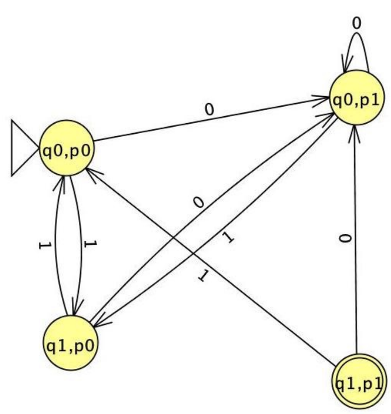
L_1 is subset of L_2 iff no string w such that w accepted by L_1 and not accepted by L_2

L_1 =L(M_2)



L_2 = L(M_2)





46

The Infiniteness Problem

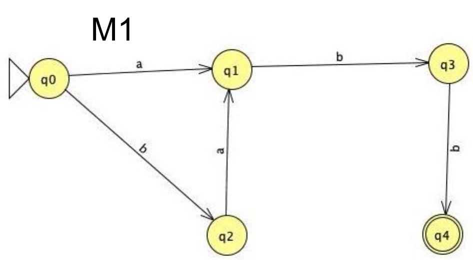
- Is a given regular language infinite?
- Theorem: Testing if L(M) is infinite is a decidable problem.
- Key idea: if the DFA has *n* states, and the language contains any string of length *n* or more, then the language is infinite
  - Proof = Homework 1 !!
  - If there is a path of length *n* or greater (from start to a final state) then there is a cycle in the graph
    - We can repeat the cycle any number of times
- Otherwise, the language is surely finite.
  - Limited to strings of length *n* or less.
- Algorithm: compute all paths length <n, and check if there is a cycle in the graph

47

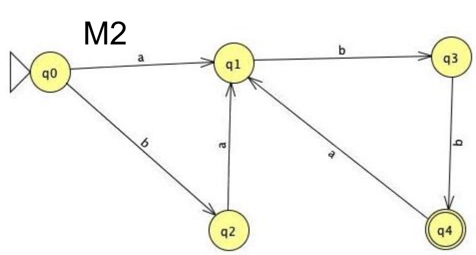
47

Transition graphs for two DFAs

Is L(M1) finite ?



Is L(M2) finite ?



48

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!

49

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.

50