CS 3313
Foundations of Computing:

Deterministic Finite Automata (DFA)

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

1

Today…..

- Recap algorithmic thinking, and introduce DNA sequence matching problem
- Introduce Deterministic Finite Automata
  - First introduce the mathematical model and notations
  - Formally define ”acceptance” by a DFA
  - Understand how the “machine” works
  - Apply algorithmic thinking to design solutions for problems that have to be solved using DFAs
    - ”Code” the algorithm as a DFA
    - “simulate” the machine (i.e., DFA) using JFLAP
- Next topic will then introduce concept of Non-determinism

2

Our Approach to studying automata models: Algorithmic Thinking

To understand how each automata (machine) model works, we take the approach of developing "algorithms" that work on that machine

3

Computational Thinking and Algorithmic Thinking – an important skill for Computer Scientists!

- "Computational Thinking is the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent" – Cuny, Snyder, Wing, 2010
- Invaluable to methodically approach a complex (unknown) problem, break it down into smaller/easier problems and quickly build a robust solution
  - Not just in CS…..everyday tasks!!

4

Algorithmic Thinking

1. Understanding the problem
   - Define it “precisely”
     - What are the inputs? What are the outputs ? What constraints?
   - Can you describe the problem
2. Devising a plan
   - Identify the level of problem solving..be methodical in your steps
     - *Do you apply a known solution*? Do you generate a new solution?
   - Create a series of steps to solve the problem
     - Each step is precise and unambiguous
   - Determine effectiveness (efficiency) of solution
3. Program the solution…..

5

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
  - Unambiguous commands
- Effectiveness: reduce task to the primitive operations of the machine
  - Ex: machine code on a computer
- Finiteness – must terminate

6

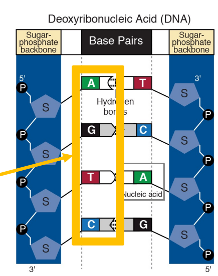
An important problem – DNA sequence detection

- Problem: Detect if a disease (captured by a DNA subsequence) appears in a patient's DNA
- Why discuss this – example of algorithmic thinking and solution design using methods we currently know

- Start with a simple solution and then return to the problem next week to see if we can design a better solution by using other solutions – i.e., different way of constructing an algorithm

7

A Well-Known Problem



NIH, Base Pair
[https://www.genome.gov/genetics-glossary/Base-Pair]

- *Problem*: Suppose a certain genetic disease *D* is characterized by the pattern *p* = **ATCCTG** presented in a strand of DNA sequence.
- p is consisted of *M* base pairs (bps) of 4 kinds.
  - For *p* = **ATCCTG**, *M* = 6 bps.
- *Question*: given a DNA sequence sample, does the patient have *D?*
  - *Assume* only need to find the pattern once.
  - DNA Sample: … AACGACCAATCCTGGA …
  - However, the DNA sample has a length of *N* bps, where *N* >> *M*.

8

Understanding the problem

- Input is a DNA sequence of patient and disease sequence
  - DNA sequence is a string of length N
  - Disease sequence D is a string of length M
  - Alphabet $\Sigma = \{ A,C,G,T \}$
- Output = {Yes, No}
  - Yes if sequence contains the disease sequence else No
- At each step we read one symbol (character) from input sequence
  - Machine/computer capability: We can check if two characters are equal
    - `If (x == y) then { ….}`
- Efficiency: how many steps, i.e., time complexity

9

How do we proceed? – Naïve

[1]: https://www.nature.com/articles/37551

- *Naïve Method*: for each* of the first *(N-M+1)* positions in the sample, try to identify the pattern by checking the current and the next* *(M-1)* positions.

DNA Sample: **A A C G A C C A A T C C T G G A …**

A T C C T G **A T C C T G**

✓ Does it work? **Yes!!**

❖ Is it efficient? Let's check!

DNA Sample: … **A T C C T C C A A T C C T G G A …**

A T C C T G

- What if the pattern is not there at all?
  - Have to check all *(N-M+1)* positions, & each time check *M* times.
  - ❖ In total, $(MN\text{-}M^2\text{+}M)$ times.

❖ **Time Complexity**: $O(MN)$

❖ *N* ~ billions & *M* ~ millions bps long;
- Lyme disease has 910,725 bps [1]

10

Summary – DNA sequence detection problem

- We designed an algorithm to detect if the disease sequence D occurs in a patient
  - D has length M and DNA sequence of patient has length N
- Algorithm efficiency (time complexity): O(NM)
  - N > M
- Next week's question: can we design a better solution using concepts we learn today (and next week).
  - Hint: yes, of course – else why would we spend time on this !!
  - Use understanding of DFAs and apply algorithmic thinking to devise a more efficient solution

11

Deterministic Finite Automata aka Finite State Machines in Sequential Circuits

- Define Deterministic Finite Automata (DFA) Model
  - Formal definition
  - Model as a graph
  - Acceptance by DFA
  - Examples
- Deterministic: at every step, and for every input, there is exactly one next state (i.e, one decision)
- Non-deterministic Automata
  - "choice" of moves the machine can make
  - View this as exploring several "parallel" options concurrently
  - Machine eventually follows one sequence of options/choices
  - Question: does adding non-determinism add to their "power"?

12

DFAs and Finite State Machines (FSM)

- How are they different ? Are they different ?...NO
  - DFAs are mathematical model of FSMs, but defined as "acceptors"
    - Final output is a "yes" or "no"
  - FSMs are an implementation of a DFA and can generate different outputs from each state
- Why DFAs (/FSMs)?
  - Control unit of a processor.....yes, they are DFAs!
  - Network protocols, switching circuits,
  - Search (in editors)...RegEx search
  - Sometimes an algorithm can be modeled as a DFA
    - Ex: searching for substrings
- Focus of this course = theoretical model of DFAs
  - We won't discuss the implementation of a DFA in hardware
    - You know this already!!

13

Recall Definitions and Notations:

- *Alphabet*: set of symbols, i.e. $\Sigma = \{a, b\}$
- *String*: finite sequence of symbols from Σ
  - Empty string: denoted λ or ϵ
- *Operations on strings*: *Concatenation, Reverse, ..*
- *Length of a string: number of symbols*
- Σ^* = set of all strings formed by concatenating zero or more symbols in Σ
- Σ^+ = set of all non-empty strings formed by concatenating symbols in Σ, i.e., $\Sigma^+ = \Sigma^* - \{ \lambda \}$
- A *formal language* L is any subset of Σ^*
- Convention: we use w,x,y to denote strings and a,b,c to denote symbols from the alphabet

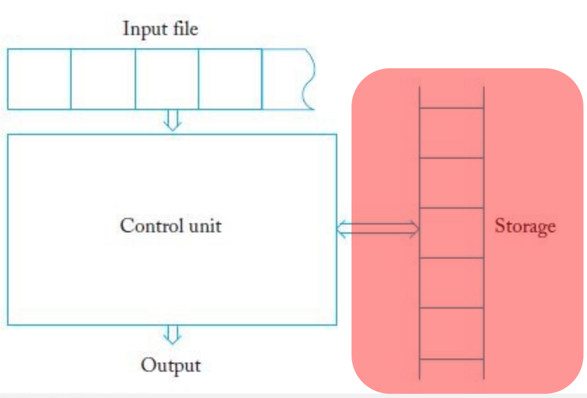
14

Recall: Automata Definition

- An ***automaton*** is an abstract model of a digital computing device
- An automaton consists of
  - An input mechanism
  - A control unit
  - Possibly, a storage mechanism
  - Possibly, an output mechanism
- Control unit can be in any number of internal *states*, as determined by a *next-state* or *transition* function.
- There are a finite number of states

15

Illustration of a General Automaton



Note: In DFA model, there is no storage device

16

Deterministic Finite Automata (Accepters)

- **Definition:** A *deterministic finite automaton (DFA)* is defined as $M = (Q, \Sigma, \delta, q_0, F)$ where:
  1. Q: a finite set of ***states***
  2. Σ: a set of symbols called the ***input alphabet***
  3. δ: a ***transition function*** from Q X Σ to Q
  4. q_0 : the ***start (initial) state***
  5. F: a subset of Q representing the ***final states***

  Final state also called "accepting" state;

  Start state also called "initial" state
- Example dfa M:

$Q = \{ q_0, q_1, q_2 \}$ $\Sigma = \{ 0, 1 \}$ $F = \{q_0\ q_1 \}$

where the transition function is given by

$\delta(q_0, 0) = q_0$ $\delta(q_0, 1) = q_1$ $\delta(q_1, 0) = q_0$

$\delta(q_1, 1) = q_2$ $\delta(q_2, 0) = q_2$ $\delta(q_2, 1) = q_2$

17

The Transition Function δ

- Takes two arguments: a state and an input symbol.
- $\delta(q, a)$ = the state that the DFA goes to when it is in state q and input a is received.
- Note: always a next state – add a trap state (also called *dead state*) if no transition
- *Yes, this is the same as the 'next state' function that you saw in the design of finite state machines*

18

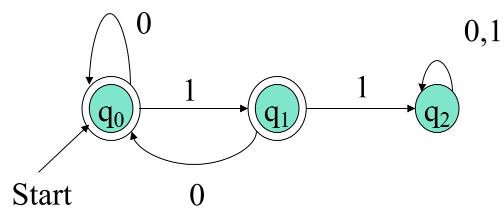
18

Graph Representation of DFA's

- Nodes = states.
- Edges (arcs) represent transition function.
  - Edge from state p to state q labeled by all those input symbols that have transitions from p to q.
- Arrow labeled "Start" to the start state.
- Final states indicated by double circles.

$\delta(q_0, 0) = q_0$ $\delta(q_0, 1) = q_1$

$\delta(q_1, 0) = q_0$ $\delta(q_1, 1) = q_2$ $\delta(q_2, 0) = q_2$ $\delta(q_2, 1) = q_2$



19

19

Processing Input with a DFA

- A DFA starts by processing the leftmost input symbol with its control in state q_0. The transition function determines the next state, based on current state and input symbol
- The DFA continues processing input symbols until the end of the input string is reached
- The input string is *accepted* if the automaton is in a final state after the last symbol is processed. Otherwise, the string is *rejected*.
- For example, the dfa in example accepts the string 100 but rejects the string 110

20

Alternative Representation: Transition Table

Final states starred

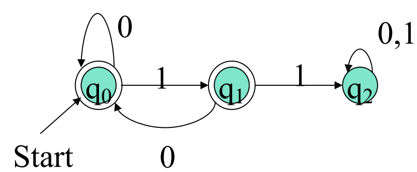
Columns = input symbols

	0	1
→ * q_0	q_0	q_1
* q_1	q_0	q_2
q_2	q_2	q_2

Arrow for start state

Rows = states

Each entry is δ of the row and column.



21

21

Transition Table vs Truth Table

Yes, transition table is similar to a truth table for next state function

Recall: for truth table all entries are binary

need to encode each state in binary using two bits s_1 s_0

q_0 encoded as 00, q_1 is 01, q_2 is 10

	0	1
q_0	q_0	q_1
q_1	q_0	q_2
q_2	q_2	q_2

Input	**s_1**	**s_0**	**s_1***	**s_0***
0	0	0	0	0
1	0	0	0	1
0	0	1	0	0
1	0	1	1	0
0	1	0	1	0
1	1	0	1	0

22

22

States in a DFA – what do they convey ?

- Finite number of states Q
- What does a state denote – i.e., design process
  - State summarizes a finite amount of information
  - Summary based on past events
    - In DFA, the past events are inputs read by the automaton until this point in time
    - Ex: input= **bba**abb
    - After reading bb, DFA is in some state *p* and then reads **a**
      - The current state *p* depends on the input **bb**
      - The next state *q* depends on current state *p* and the input *a*
- *In context of "algorithm design": think of a state as a specific "step" or point in the program*

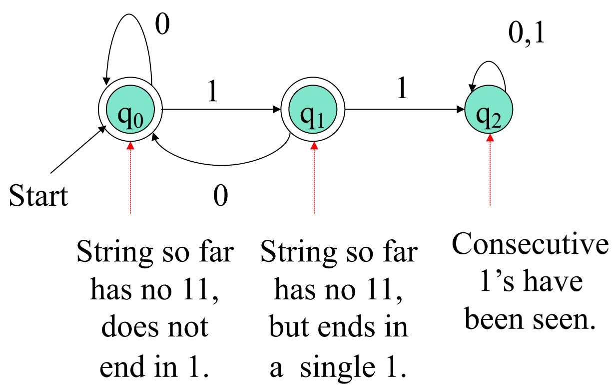
23

Example: A Vending Machine

- Accept user input (coins) when total is at least 50 cents
  - Real machine: dispense output (candy) when total is 50 or more
- Input valid coins: alphabet = { 5,10,25}
  - Q (25cents) D (10) or N (5)
- What should it keep track of ?
  - current total
- When it reaches 50 or more: Final state
  - Generate output
- States of the machine ?
  - What should each state capture ?
  - How many states ?

24

Example 1: Strings With no consecutive 1's (no 11 in input)



25

Extended Transition Function

- The machine/DFA reads an input string of n symbols, therefore need to describe the effect of entire input string on a DFA by extending δ to a state and a string.
- Intuition: Extended δ is computed for state q and inputs $a_1a_2\ldots a_n$ by following a path in the transition graph, starting at q and selecting the arcs with labels $a_1, a_2, \ldots, a_n$ in turn.

- Notation: we start by denoting the extended function as δ^* and will drop the * after we define it

26

Inductive Definition of δ^* – Extended δ

- Induction on length of string.
- Basis: $\delta^*(q, \epsilon) = q$
- Induction: $\delta^*(q,wa) = \delta(\delta(q,w),a)$
  - Remember: w is a string; a is an input symbol, by convention.

27

27

Example: Extended Delta

	0	1
q_0	q_0	q_1
q_1	q_1	q_2
q_2	q_2	q_2

$$\delta(q_1,011) = \delta(\delta(q_1,01),1) = \delta(\delta(\delta(q_1,0),1),1) =$$

$$\delta(\delta(q_0,1),1) = \delta(q_1,1) = q_2$$

28

28

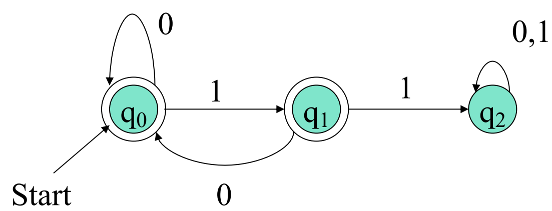
Example1: String accepted by DFA M

A string w is "accepted" by the DFA if the machine is in a final state after reading w

String 101 is in the language of the DFA below.

Starts at q_0

Ends in state q_1 which is a final state.



29

29

Definition: Language accepted by a DFA

- If M is an automaton/machine, L(M) is its language: set of input strings accepted by the machine.
  - L(M) is the set of strings that take the machine from the start state to a final state
- Formally:

L(M) = the set of strings w such that $\delta(q_0, w)$ is in F.

$L(M) = \{ w \mid \delta(q_0, w) \in F \}$

- In terms of the graph: string w is accepted by DFA M if there is a path labelled w from node q_0 to some node q which is a final state

30

30

Definition: Regular Languages

- DFAs accept a family of languages collectively known as *regular languages*.
- A language L is *regular* if and only if there is a DFA M that accepts L.
  - Therefore, to show that a language is regular, one must construct a DFA to accept it, i.e., L=L(M) for some DFA A.
- Regular languages have wide applicability in problems that involve scanning input strings in search of specific patterns.
- Some languages are not regular, i.e., there is no DFA M that accepts the language
  - Intuitively, regular languages cannot "count" to arbitrarily high integers.
  - *To show that a language is not regular we need to prove that there is no DFA M that accepts the language*

31

Designing DFAs…and applying algorithmic thinking

- Recall properties of an "algorithm" – sequence of steps, each step can be carried out by the primitive operations of the machine,…
- In DFA: each step reads symbol from input and transition function determines next state
- DFA "accepts" (i.e., output = "Yes") if it ends up in a final state
- Each state summarizes events that have taken place thus far
  - It captures some property of the string that has been read thus far
  - Ex: 0110<u>1</u>10 after reading 0110 it is in state p, now reads 1

32

Example 2:

- L = { w | w is a string in $\{0,1\}^*$ and w contains the substring 101}
  - Input is binary string, and DFA M is in final state if input contains 101.
- What should the states keep track of ?

  Pattern of the input read 'thus far'

  - All of the input or the relevant part, i.e., the properties of the substring ?
- Algorithm: reads input, one input symbol at a time, determines next state/step
- Properties/constraints:
  - If we read 0 then this cannot be end or beginning of the substring
    - It can only be after a 1 was read in the previous step
  - If we read a 1 then it could be the start of the substring OR it could be end of substring if previous two inputs were 10
  - Once you read 101, then continue reading all inputs and stay in final state

33

Example 2: Algorithm

- L = { w | w is a string in $\{0,1\}^*$ and w contains the substring 101}

1. (Start) If (input ==0) , i.e., read 0, stay in step 1 – because this is the first 0 or is a 0 with no 1 before it.

   else If (input == 1) then go to step 2 – this could be the first symbol in the substring 101

2. If (input ==0) then go to step 3 – this means we read 10 as last two inputs.

   else If (input ==1) then stay in step 2 – this recent 1 is start of substring

3. IF (input==1) then go to step 4 – we have seen substring 101

   else if (input ==0) then go to step 1 – we have to start all over again since 100 cannot be part of the substring 101

4. IF (input==0) or (input==1) stay in step 4.

   If no input then halt and accept the input string.

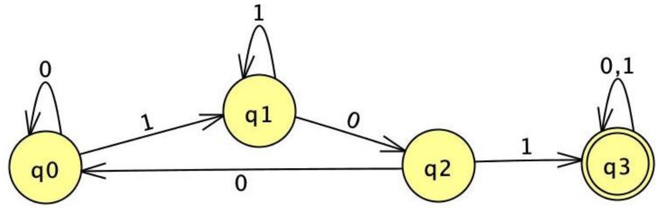
34

Example 2: Encoding the states

- From the algorithm, identifying the states is reasonably straightforward (in this example):
- q_0 – start state, corresponds to step 1 of algorithm
  - State summarizes “machine has not seen the first symbol of the substring 101”
- q_1 –step 2
  - State summary: last input was 1, so we just read first symbol of 101
- q_2 –step 3
  - State summary: last two inputs are 10 which is substring of 101
- q_3 – step 4…final state
  - Machine has read 101 in the input string
- Next – write out the formal DFA (the “machine language”)

35

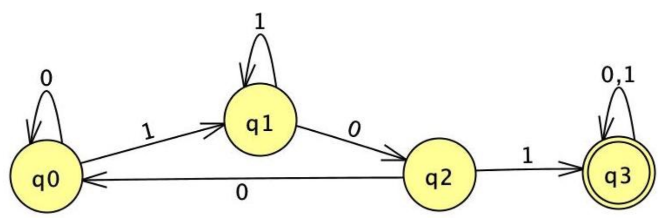
Example 2: DFA that recognizes Substring 101



q0: not read first 1 in substring 101
q1: last input read was a 1, could be start of substring 101
q2: last two inputs read were 10 which is part of substring 101
q3: last three inputs read were 101 which means substring 101 is in input

36

Example 2: DFA that recognizes Substring 101



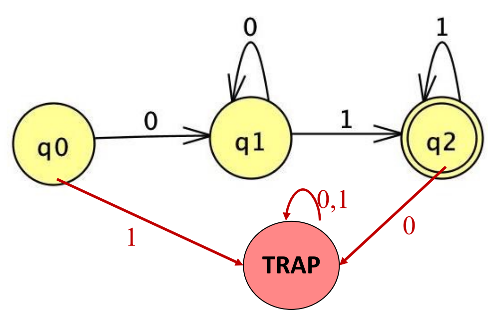
Transition Table:

	0	1
q_0	q_0	q_1
q_1	q_2	q_1
q_2	q_0	q_3
q_3	q_3	q_3

37

Trap States

- Typical convention is to define transition function for “valid” moves of the machine – implication is that undefined moves will take the machine to a trap state
  - In trap state, machine keeps reading input symbols till end of input
  - ***Assumption: If transition $\delta(q, w)$ is not defined then it goes to Trap state***
- Example: L = { w | w {0,1}* and w has one or more 0’s followed by pne or more 1’s}



38

Exercise DFA 1 : Groups

- L = { w | w is a string in $\{0,1\}^*$ and w has an even number (at least 2) of 1's followed by an odd number of 0's}
  - Ex: 11000 is in L
  - Ex: 11100 is not in L

- Algorithm ?
  - Hint: the conditions that the algo should check are (a) 1's followed by 0's and (b) 1's should be even and 0's should be odd
    - From start state (step 1), read the input string one symbol at a time
    - At each state "capture" if we have even 1's or odd 1's or after you have finished reading only 1's check even 0's/odd 0's (

39

Summary and Next….

- Machine M specified as a 5-tuple
  - Alphabet, Set of States Q, Final states F, Start State , Transition function δ
- Language accepted by M is set of strings that take M from start to final state: $L(M) = \{ w \mid \delta(q_0, w) \in F \}$
- State: summarizes events that have taken place thus far
  - Until current input is read
- Next question: How do we show that a DFA M accepts exactly a language L ?
- Your to-do list: Install JFLAP, read notes/text
- Next week labs will go over examples using JFLAP

40

Deterministic Finite Automata - Summary

- **Definition:** A *deterministic finite automaton (DFA)* is defined as $M = (Q, \Sigma, \delta, q_0, F)$ where:
  1. Q: a finite set of ***states***
  2. Σ: a set of symbols called the ***input alphabet***
  3. δ: a ***transition function*** from Q X Σ to Q – can be represented as a graph
  4. q_0 : the ***start (initial) state***
  5. F: a subset of Q representing the ***final (/accepting) states***

  "Algorithm" model for the machine:
- At each step: M reads one symbol from input and goes to next state – a state summarizes events that have occurred thus far
- If at end of input, M is in final state it accepts

41

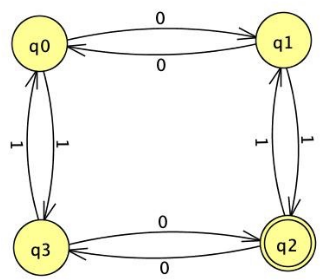
Example DFA 3

- Give a DFA for the language

  $L_3 = \{ w \mid w \in \{0,1\}^*$ and w has an odd number of 0's and an odd number of 1's $\}$
  - Ex: 010110 is in L_3 1010 is not in L_3
- For any binary string, we have n 0's and m 1's
- What are the cases we can have:
  - a) Both even
  - b) Even number of 0's and Odd 1's
  - c) Odd 0's and Even 1's
  - d) Odd 0's and Odd 1's
- Question: If M has read a substring v thus far and has case (b) – even 0 and odd 1 – then if it reads 0 next, what is the next case/state?

42

Example DFA 3



q_0: even 0 and even 1
q_1: odd 0 and even 1
q_2: odd 0 and odd 1
q_3: even 0 and odd 1

43

Exercise 2: Work in breakout groups and submit

- Provide a DFA for L = { w | w is a string in $\{0,1\}^*$ and w contains (a) the substring 101 or (b) substring 010 }
  - Ex: 00101011 is in L, 10110 is in L,
  - Ex:1110 is not in L, 0001100 is not in L

44

Next: Proving Correctness of your DFA

- If M is an automaton/machine, L(M) is its language: set of input strings accepted by the machine.

L(M) = the set of strings w such that $\delta(q_0, w)$ is in F.

$L(M) = \{ w \mid \delta(q_0, w) \in F \}$

- If you are designing a DFA M to accept a language L, then how do we show that your design is correct ?

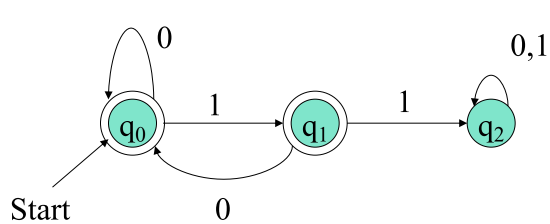
Prove L(M) = L

Example – Proving Property of language accepted by DFA Example 1

- The language of our example DFA is:

$\{w \mid$ w is in $\{0,1\}^*$ and w does not have two consecutive 1's$\}$

These conditions about w are true.



Proofs of Set Equivalence

- Often, we need to prove that two descriptions of sets are in fact the same set.
- Here, one set is "the language of this DFA," and the other is "the set of strings of 0's and 1's with no consecutive 1's."

47

47

Proofs – (2)

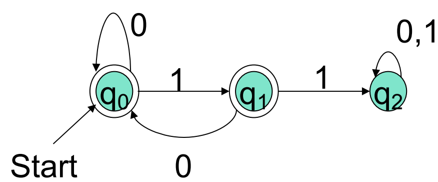
- In general, to prove S = T, we need to prove two parts:
  $S \subseteq T$ and $T \subseteq S$. That is:
  1. If w is in S, then w is in T.
  2. If w is in T, then w is in S.
- Here, S = the language of our running DFA, and T = "no consecutive 1's."

48

48

Part 1: S ⊆ T

- To prove: if w is accepted by DFA M then
  w has no consecutive 1's.
- Proof is an induction on length of w.
- *Important trick*: Expand the inductive hypothesis to be more detailed than the statement you are trying to prove.



The Inductive Hypothesis

1. If $\delta(q_0, w) = q_0$, then w has no consecutive 1's and does not end in 1.
2. If $\delta(q_0, w) = q_1$, then w has no consecutive 1's and ends in a single 1.

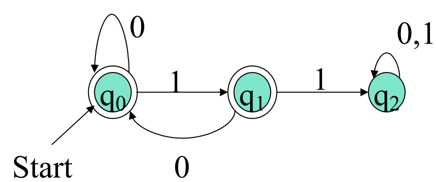
- Basis: $|w| = 0$; i.e., $w = \varepsilon$.
  - (1) holds since ϵ has no 1's at all.
  - (2) holds *vacuously*, since $\delta(q_0, \varepsilon)$ is not q_1.

"length of"

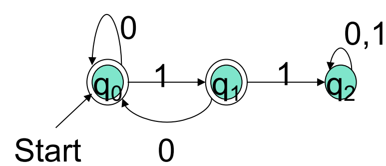
Important concept:
If the "if" part of "if..then" is false, the statement is true.

Inductive Step

- Assume (1) and (2) are true for strings shorter than w, where |w| is at least 1.
- Because w is not empty, we can write w = xa, where *a* is the last symbol of w, and x is the string that precedes.
- IH is true for x.

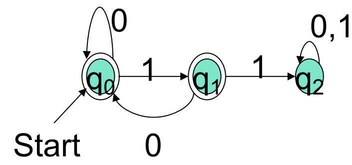


Inductive Step – (2)



- Need to prove (1) and (2) for w = xa.
- (1) for w is: If $\delta(q_0, w) = q_0$, then w has no consecutive 1's and does not end in 1.
- Since $\delta(q_0, w) = q_0$, $\delta(q_0, x)$ must be q_0 or q_1, and *a* must be 0 (look at the DFA).
- By the IH, x has no 11's.
- Thus, w has no 11's and does not end in 1.

Inductive Step – (3)



- Now, prove (2) for $w = xa$: If $\delta(q_0, w) = q_1$, then w has no 11's and ends in 1.
- Since $\delta(q_0, w) = q_1$, $\delta(q_0, x)$ must be q_0, and a must be 1 (look at the DFA).
- By the IH, x has no 11's and does not end in 1.
- Thus, w has no 11's and ends in 1.

Part 2: T ⊆ S

X

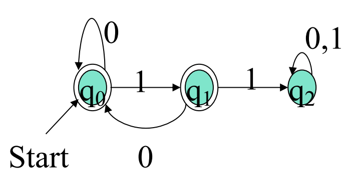
- Now, we must prove: if w has no 11's,

  then w is accepted by DFA M

Y

- *Contrapositive* : If w is **not** accepted by M then

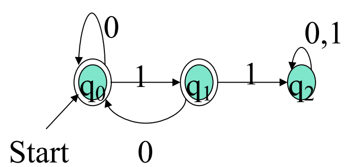
  string w has 11



Key idea: contrapositive of "if X then Y" is the equivalent statement "if not Y then not X."

Using the Contrapositive

- Because there is a unique transition from every state on every input symbol, each w gets the DFA to exactly one state.
- The only way w is not accepted is if it gets to q_2.

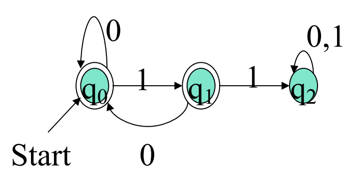


55

55

Using the Contrapositive – (2)

- The only way to get to q_2 [formally: $\delta(q_0,w) = q_2$] is if $w = x1y$, and x gets to q_1, and y is the tail of w that follows what gets to q_2 for the first time.
- If $\delta(q_0,x) = q_1$ then surely $x = z1$ for some z.
- Thus, $w = z11y$ and has 11.



56

56

Summary: Language accepted by a DFA M

- In general, to prove the correctness of your design (of a DFA M to accept a language L):

  Prove they are equal !

- Typically: prove "property" of each state (or at least the state that is connected to a final state)
  - This is essentially proving the correctness of your algorithm !!

- Proving correctness of an algorithm is an important step in the design of algorithms

57

Question

- I have a DFA M_1 and another DFA M_2, and want to check if $L(M_1)=L(M_2)$
- How ?...
  - A proof like we just provided ?

- Why ask this question.....Algorithmic thinking !!!!
  - Will return to this question when we discuss "Decision Properties" for Regular Languages/DFAs

58

DFAs - Summary

- Simple model of machines
  - Finite number of states
  - Transition from one state to another based only on the input symbol
- Algorithms using DFA Model:
  - Write out steps, each step machine is in a state and reads an input and computes next state
  - This can be implemented as a DFA or can be implemented as a program!

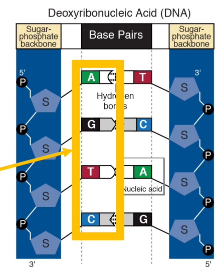
59

Next……Non-determinism

- We want to add features to the base machine (DFA) in an attempt to increase its "power"
  - Power = more problems it can solve ? Or is it efficiency (time) ?
- Next topic: non-deterministic machines and non-deterministic finite automaton
  - Non-determinism adds more expressive power to the algorithm
    - Can be viewed as machine executing several parallel paths
- Now on to another exercise….

60

Exercise 3 - Recall: DNA Sequence Matching Problem



NIH, Base Pair
[https://www.genome.gov/genetics-glossary/Base-Pair]

- *Problem*: Suppose a certain genetic disease *D* is characterized by the pattern *p* = **ATCCTG** presented in a strand of DNA sequence.
- p is consisted of *M* base pairs (bps) of 4 kinds.
  - For *p* = **ATCCTG**, *M* = 6 bps.
- *Question*: given a DNA sequence sample, does the patient have *D?*
  - *Assume* only need to find the pattern once.
  - DNA Sample: … AACGACCAATCCTGGA …
  - However, the DNA sample has a length of *N* bps, where *N* >> *M*.
- Naïve Algorithm: Given sequence of length N and disease sequence of length M, we have an O(MN) algorithm.

61

Exercise 3: Apply DFA Algorithmic thinking to the DNA sequence matching problem

- Why study DFA model of 'computing' (processing)?
- Now that you know how a DFA works, can you construct a more efficient solution for the DNA sequence matching problem ?
  - Design an algorithm that works like a DFA
  - What is the time complexity of your solution ?
- Work in groups and submit……

62