CSC236 Week 10

Larry Zhang

Test 2

First hour of next week’s lecture.

- Aid allowed: one 8.5”x11” sheet, double-sided

Coverage

- Divide and conquer and the master theorem
- Correctness of recursive programs
- Correctness of loops
    - loop invariant
    - variant and termination

Test 2

Three Questions:

- Design or given a recursive program
  - analyse runtime and correctness
- Loop invariant
  - Given a piece of code, find or given an invariant, and prove it.
- Loop termination
  - Given a piece of code, find and prove a variant

Test 2

Preparation:

- Lectures, tutorials, problem sets
- Past tests posted on the course web page
- Tutorial this week: test prep exercises
- Office hours
    - Monday and Wednesday as usual
    - Friday 2-4pm

Today’s Topic

Deterministic Finite Automata (DFA)

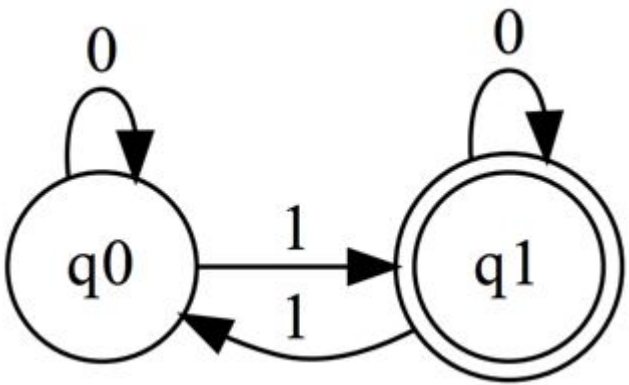
Recap of last week

- We learned a lot of terminologies
  - alphabet
  - string
  - length of string
  - union
  - concatenation
  - Kleene star
  - language
  - **Regular Language and Regular Expression**

Why the fuss about “**regular**”?

- If a language is **regular**, then a computer can **efficiently recognize** it
- By “efficiently recognize” we mean
  - Given a regular language **L** and a string **w**, the computer can determine whether **w** is in **L** by scanning the characters in **w** once, one by one,from left to right.
    - If **L** is not regular, then the computer simply cannot do it.
  - The algorithm used to determine whether **w** is in **L** is modelled using a **Deterministic Finite Automata (DFA)**.

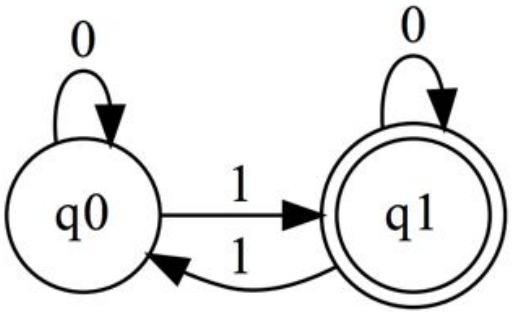
DFA: Example



Say we want to test string w = 0110

- **q0** is the **initial state**
- **q1** is the **accepting (final) state**
- read each character of w from left to right, each character causes a **state transition** in the flowchart, following one of the arrows.
- After scanning the whole string, if the current state is at the **accepting state (q1)** then the string is “**accepted**” by the DFA.
- Otherwise, the string is “**rejected**”.

Let's test a few strings



- 0
  - q0 -> q0, **rejected**
- 1
  - q0 -> q1, **accepted**
- 100
  - q0 -> q1 -> q1 -> q1, **accepted**
- 110
  - q0 -> q1 -> q0 -> q0, **rejected**
- 10101
  - q0 -> q1 -> q1 -> q0 -> q0 -> q1 -> q1, **accepted**
- 10101001
  - q0 -> q1 -> q1 -> q0 -> q0 -> q1 -> q1 -> q1 -> q1 -> q0, **rejected**

Guess what kind of strings are accepted by this DFA?

ANS: Strings with an odd number of 1's!

This DFA determines if a string is a member of the following regular language: {w ∈ {0, 1}* | w consists of an odd number of 1's}

DFA and Regular Language

- For each regular language, there exist a DFA that can be used to determine if a given string is a member of the regular language
- When you use regular expressions in your code, the regular expression is converted into a DFA by the compiler or interpreter. The computer then just run the algorithm by following the transitions in the DFA.
- If a language is not regular, then there does NOT exist a DFA that recognizes it. (You would need an infinitely number of states, but DFA must be finite)

Formal Definition of DFA

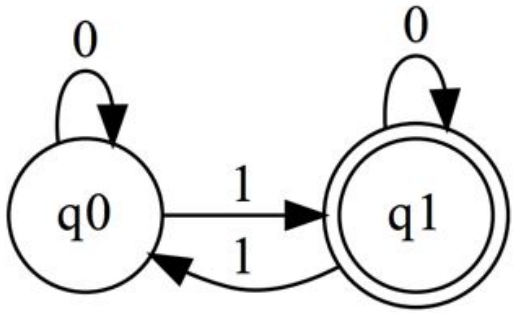
DFA defined as a quintuple: **(Q, ∑, δ, s, F)**

A Deterministic Finite Automaton (DFA) $\mathcal{D}$ has five components.

- Q is the (finite) set of *states*
- Σ is the *alphabet* of symbols
- $\delta : Q \times \Sigma \to Q$ is the *transition function* representing the state transitions
- $s \in Q$ is the *initial state*
- $F \subseteq Q$ is the set of *accepting (final) states*

We write this as $\mathcal{D} = (Q, \Sigma, \delta, s, F)$

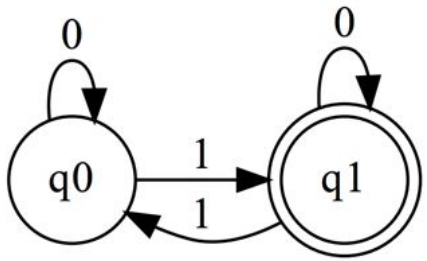
What are the **(Q, ∑, δ, s, F)** of this DFA?



- Q = {q0, q1}
- ∑ = {0, 1}
- s: q0 # always only one initial state
- F: {q1} # can have multiple final states
- δ: can be expressed using a **transition table**

Old State	Symbol	New State
q_0	0	q_0
q_0	1	q_1
q_1	0	q_1
q_1	1	q_0

How DFA works



- DFA reads strings one symbol at a time, from left to right
- DFA cannot “go back” and reread previous symbols
- At a particular state, once you read a symbol, there is only one transition (arrow) that you can follow.
- DFAs have a finite amount of memory, since they have a finite number of states

Exercise

Exercise: Design a DFA

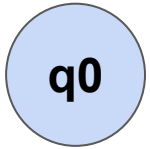
Design a DFA that accepts the following language:

$$L = \{w \in \{a, b\}^* \mid w \text{ starts with } a \text{ and ends with } b\}$$

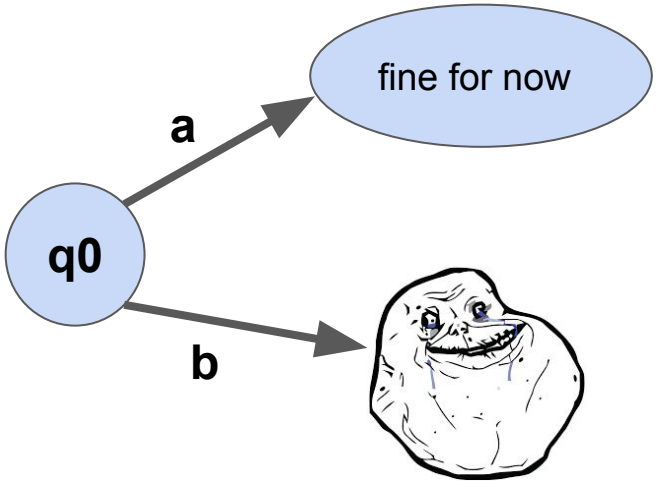
$L = \{w \in \{a, b\}^* \mid w \text{ starts with } a \text{ and ends with } b\}$

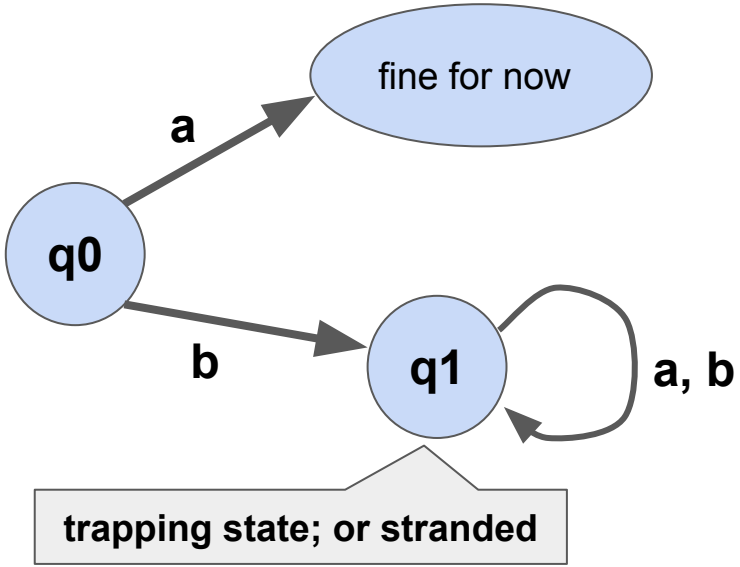
Thinking process ....

Start from an initial state q0 ...



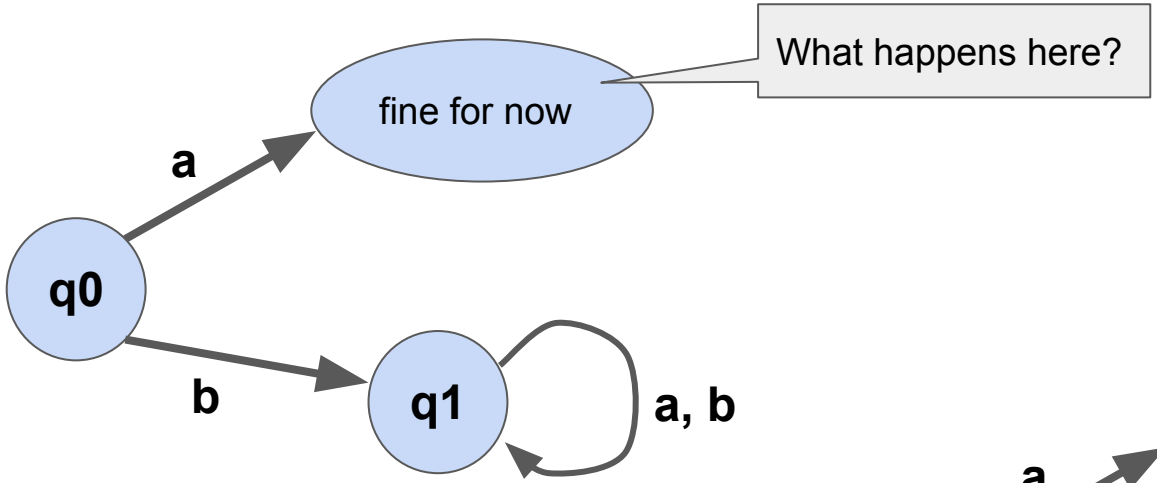
What happens if we get “**a**” or “**b**” when at q0 ?





$L = \{w \in \{a, b\}^* \mid w \text{ starts with } a \text{ and ends with } b\}$

Thinking process, continued

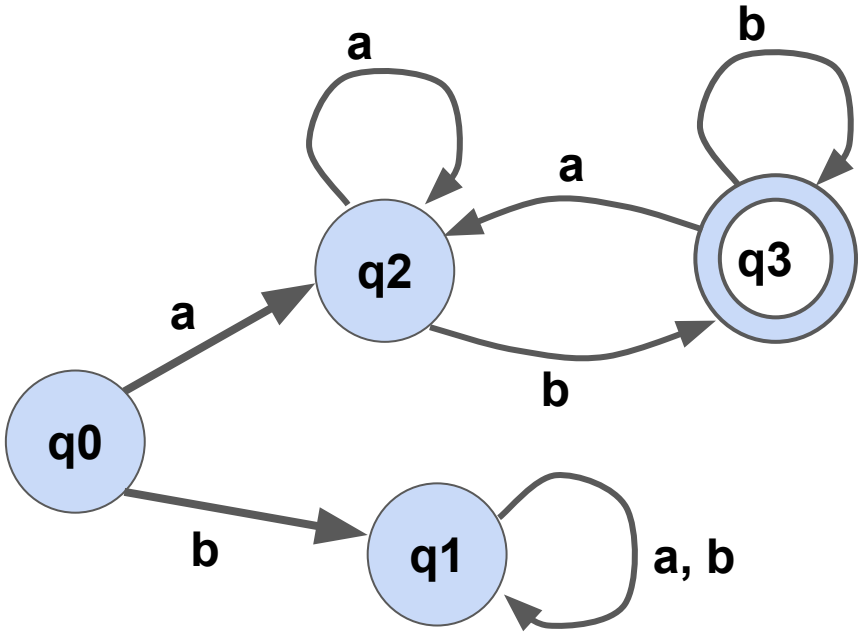


Just after getting a single “**a**”, not ready to accept.

If getting a “**b**”, then ready to accept it.

If getting another “**b**”, still ready to accept.

But if getting a “**a**”, then change back to “not ready to accept” state.



Prove the Correctness of DFAs

Overview

- For regular expressions, it is relatively easy to argue that it is **incorrect**.
  - find a string in the language that is not matched by the regex, or
  - find a string matched by the regex but is not in the language
- To formally prove that a regular expression is **correct** is general hard.
- Proving the **correctness** of a DFA is much easier.
  - With the help from a tool called “**state invariant**”.

Interlude: A notation

- **δ(q0, a) = q1**
  - if at state q0, and reading symbol “a”, transition to state q1
- **δ(q0, abaabb) = q1**
  - if at state q0, and after reading the sequence of symbols “abaabb” (6 transitions happening), the state reaches q1
- **δ(q0, ε) ?**
  - **δ(q0, ε) = q0** (reading an empty string makes no transition)

State Invariant

A **state invariant** gives, for a state **q** of a DFA, what we know to be **true** of **exactly** those strings that reach **q**.

- Let $\mathcal{D} = (Q, \Sigma, \delta, s, F)$
- Let $q \in Q$ be a state of $\mathcal{D}$
- A **state invariant** for q is a predicate $P(x)$ such that for every string $w \in \Sigma^*$, $\delta(s, w) = q$ *if and only if* $P(w)$ is true

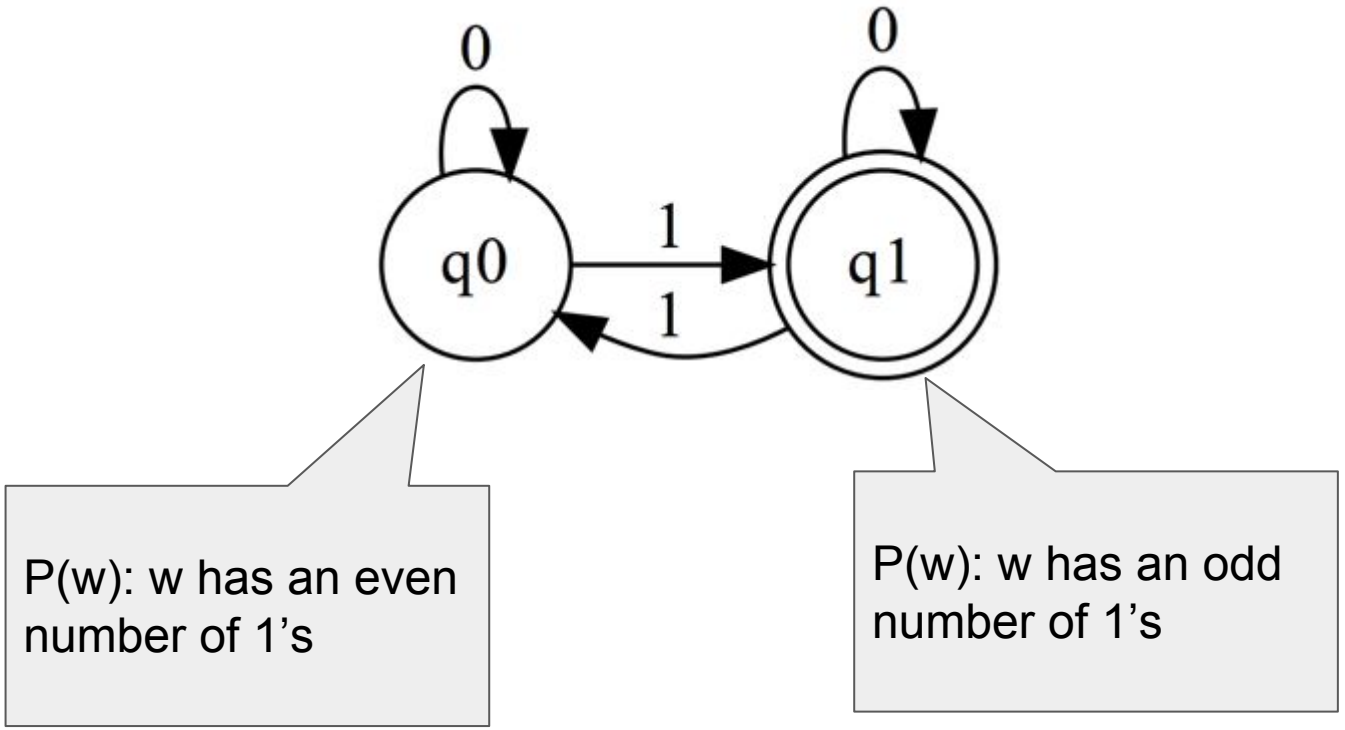
- All strings w reaching q must be satisfy P(w)
- All strings w satisfying P(w) must reach q

Desired Properties of State Invariants

When you design a DFA, you need to design the state invariant for each state you want to have in the DFA. These state invariants must satisfy the following:

- **No** string should satisfy **more than one** invariant
    - e.g., if one state claims that w has at least two 0's, and another state claims that w has at least three 0's, then there are strings that would satisfy both invariants. Bad design!
- **Every** string should satisfy **one** of the invariants
    - e.g., if you have only two states: one state claims that w has exactly two 0's, and another state claims that w has at least three 0's, then strings like 111 does NOT satisfy any invariant. Bad design!

Exercise: What are the state invariants?



Prove the correctness of DFA using State Invariants

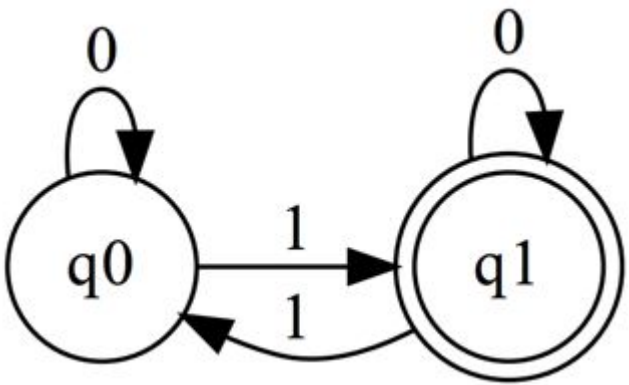
Surprise! We use **induction**.

- **Base case**: Show that **ε** (the empty string) satisfies the state invariant of the initial state.
- **Induction step**: For each transition from state **q** to state **r** on symbol **a**,
  - assume that the invariant of state **q** holds for some string **w** (I.H.)
  - show that the invariant of state **r** holds on string **wa**.
- **“Postcondition”**: Show that the state invariant(s) of the accepting state(s) exactly describe the languages that we want the DFA to accept.

Exercise

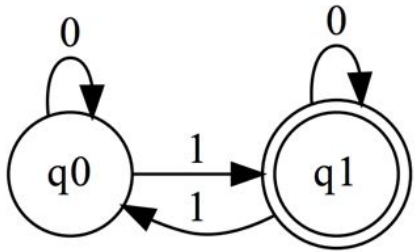
Exercise

Prove that the following DFA accepts exactly strings that have an odd number of 1's. (q0 is initial state, q1 is final state)



Old State	Symbol	New State
q_0	0	q_0
q_0	1	q_1
q_1	0	q_1
q_1	1	q_0

Proof: (identify state invariants)

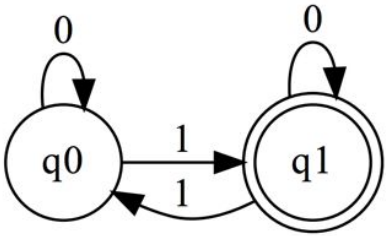


As we discussed earlier, the state invariants are the following

- q0: w has an even number of 1’s
- q1: w has an odd number of 1’s

Proof: (base case)

- q0: w has an even number of 1's
- q1: w has an odd number of 1's



Base case: Show that **ε** (the empty string) satisfies the state invariant of the initial state.

Initial state is q0

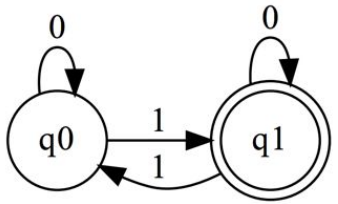
State invariant of q0 is: w has an even number of 1's

ε has zero 1's, so satisfying the invariant

Base case done.

Proof: (Induction Step)

- q0: w has an even number of 1's
- q1: w has an odd number of 1's



Induction step: For each transition from state **q** to state **r** on symbol **a**,

- assume that the invariant of state **q** holds for some string **w** (I.H.)
- show that the invariant of state **r** holds on string **wa**.

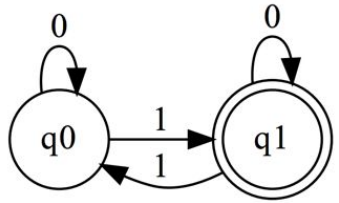
We have **four** transitions to check.

- q0 to q0 on symbol 0
  - assume some string **w** has an even number of 1's, then **w0** still has an even number of 1's
- q0 to q1 on symbol 1
  - assume some string **w** has an even number of 1's, then **w1** has an odd number of 1's
- q1 to q0 on symbol 1
  - assume some string **w** has an odd number of 1's, then **w1** has an even number of 1's
- q1 to q1 on symbol 0
  - assume some string **w** has an odd number of 1's, then **w0** still has an odd number of 1's

Induction step done!

Proof (“Postcondition”)

- q0: w has an even number of 1’s
- q1: w has an odd number of 1’s



“Postcondition”: Show that the state invariant(s) of the accepting state(s) exactly describe the languages that we want the DFA to accept.

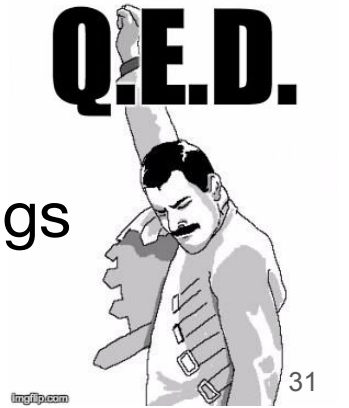
There is one accepting state which is **q1**.

The language we want to match is “strings that have an odd number of 1’s”

This is exactly the state invariant of q1

Done.

All proof steps done. This DFA accepts exactly strings that have an odd number of 1’s.



Review the Steps: Prove the correctness of DFA

Surprise! We use **induction**.

- **Base case**: Show that **ε** (the empty string) satisfies the state invariant of the initial state.
- **Induction step**: For each transition from state **q** to state **r** on symbol **a**,
  - assume that the invariant of state **q** holds for some string **w** (I.H.)
  - show that the invariant of state **r** holds on string **wa**.
- **“Postcondition”**: Show that the state invariant(s) of the accepting state(s) exactly describe the languages that we want the DFA to accept.

Minimum Number of States

Given a language, how many states does a DFA need to have **at least** in order to correctly accept the language?

Minimum Number of States

$$L = \{w \in \{0,1\}^* \mid w \text{ has at least three 1s}\}$$

Prove that any DFA accepting this language has at least 4 states.

$$L = \{w \in \{0,1\}^* \mid w \text{ has at least three 1s}\}$$

Proof sketch

- Suppose for **contradiction** that we can find a correct DFA that has 3 states
- By the Pigeonhole Principle, if we choose 4 strings over $\sum$, then at least two of those strings must end at the same state q.
- Let $\mathbf{w_i}$ and $\mathbf{w_j}$ be these 2 strings that satisfies the above.
- For any string **x**, if $\mathbf{w_i}$ and $\mathbf{w_j}$ end at q, then $\mathbf{w_i x}$ and $\mathbf{w_j x}$ must also end at the same state **r** and hence must both be accepted or rejected by the DFA.
- If we can pick for 4 strings such that, for any pair of them, for some x, that $\mathbf{w_i x}$ **is NOT in L** (rejected) and $\mathbf{w_j x}$ **is in L** (accepted), then we have a contradiction, and what must assumed (we can find a correct DFA with 3 states) must be **FALSE**.

$$L = \{w \in \{0,1\}^* \mid w \text{ has at least three 1s}\}$$

Proof:

We pick the following four strings

- w0 = ε
- w1 = 1
- w2 = 11
- w3 = 111
- For one of the paris of strings (we don't know which pair), the supposed 3-state DFA is forced into the same state for both strings (because of Pigeonhole), and **$w_i x$** and **$w_j x$** must be both accepted or both rejected, for any string **x**.
- We will show, for each possible pair, that this is NOT true.

$$L = \{w \in \{0,1\}^* \mid w \text{ has at least three 1s}\}$$

$$w_0 = \epsilon,\ w_1 = 1,\ w_2 = 11,\ w_3 = 111$$

Check all pairs

- Pair #1: **w_0** and **w_1**, which x to choose?
  - Choose x = 11
  - **w_0x** = 11, rejected; **w_1x** = 111, accepted. That's what we want.
- …
- … **Home exercise: Do the rest!**
- …
- ....
- Pair #6: **w_2** and **w_3,** which x to choose?
  - No need to choose an x
  - **w_2** =11 is rejected by the DFA; **w_3** =111 is accepted
  - We're toast already because no way these two can be in the same state

Reflection: How to approach this kind of proof

The key is to pick the suitable strings w0, w1, w2, …

- They should be “different types” of strings, according to the language in question
    - e.g., the previous example had to do with the number of 1’s, so we varied the number of 1’s in our chosen strings
- Think about the state invariants for the necessary states, then choose one string from each state as your suitable strings, so when you have to assume some of them are in the same state, you’ll get a contradiction.

Regular or Not Regular

That needs a proof

Prove a language is NOT regular

- Not all languages are regular.
- Every regular language has a DFA that accepts it.
- A language that is NOT regular does NOT have a DFA that accepts it
  - because that DFA that accepts it would need an **infinite** number of states!
  - but DFAs must have a **finite** number of states
- So to prove a language is not regular is basically to prove the minimum number of states needed by the DFA is **infinite**.

Prove the following language is NOT regular

$$L = \{0^n 1^n \mid n \in \mathbb{N}\}$$

- This language has the strings that have some number of 0's followed by the same number of 1's.
- Let's start by showing that a DFA accepting L must have at least 3 states.
- Then we crank it up by showing that, for any **k ∈ N**, a DFA accepting **L** must have at least **k** states.
    - This basically means we need **infinite** number of states.

$$L = \{0^n 1^n \mid n \in \mathbb{N}\}$$

The DFA needs at least 3 states

Same deal as last example: For contradiction, assume we have a 2-state DFA, then we pick three strings, and we know two of them must be in the same state; and they reach the same state after concatenating with the same string x.

- Pick 0, 00 and 000, show contradiction for each pair of the three
  - 0 and 00, let x=1, then 01 is accepted while 001 is rejected
  - 0 and 000, let x=1, then 01 is accepted while 0001 is rejected
  - 00 and 000, let x=11, then 0011 is accepted while 00011 is rejected

Use the same technique to show at least **k** states are needed.

$$L = \{0^n 1^n \mid n \in \mathbb{N}\}$$

DFA needs at least k states, for any k ∈ N

- For contradiction, assume we have a DFA with k-1 states
- Then pick k suitable strings
  - 0, 00, 000, …, i.e., 0^1, 0^2, 0^3, …, 0^k
- For any pair **0^i** and **0^j**
  - Let **x=1^i**
  - then **0^i.1^i** is accepted, while **0^j.1^i** is rejected.
  - contradiction
- DFA needs at least k state, where k is an arbitrarily large natural number
- Need an infinitely number of states. **The language is NOT regular.**

Summary

Why do we learn about DFA?

- Really understand the difference regular languages and not regular languages. Know what languages are possible for computers to process.
- Really understand how regular expressions really work in your computer, how string matching thing really happens
- With the power of DFA, you will be able to design crazy string matching algorithms that looks so clever that other people cannot understand how on earth you came up with it.
    - Example: The KMP algorithm http://www.ics.uci.edu/~eppstein/161/960227.html

Next week

- Test 2
- Nondeterministic Finite Automata