

# Nonregular Languages

Recap from Last Time

**Theorem:** The following are all equivalent:

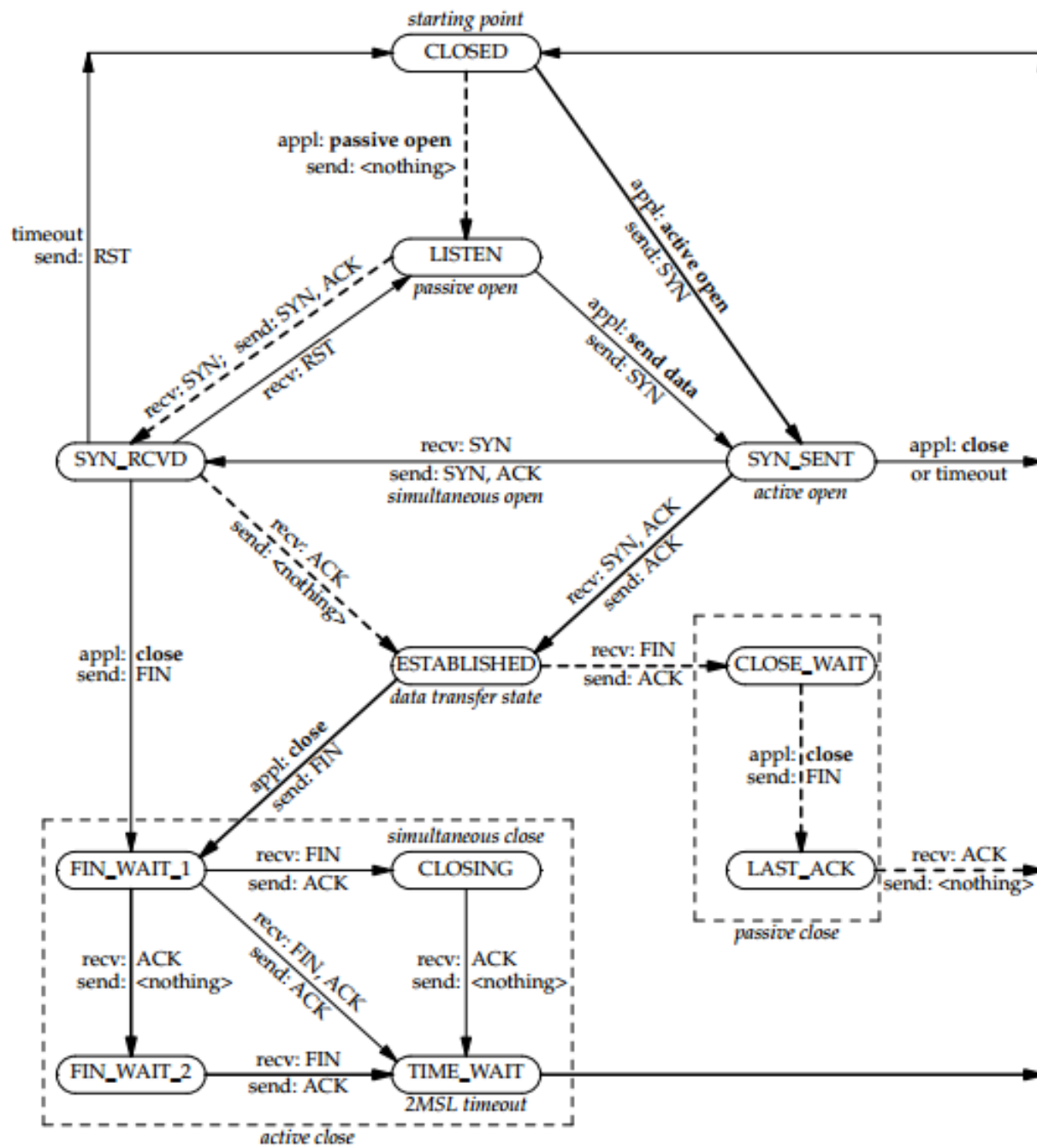
- $L$  is a regular language.
- There is a DFA  $D$  such that  $\mathcal{L}(D) = L$ .
- There is an NFA  $N$  such that  $\mathcal{L}(N) = L$ .
- There is a regular expression  $R$  such that  $\mathcal{L}(R) = L$ .

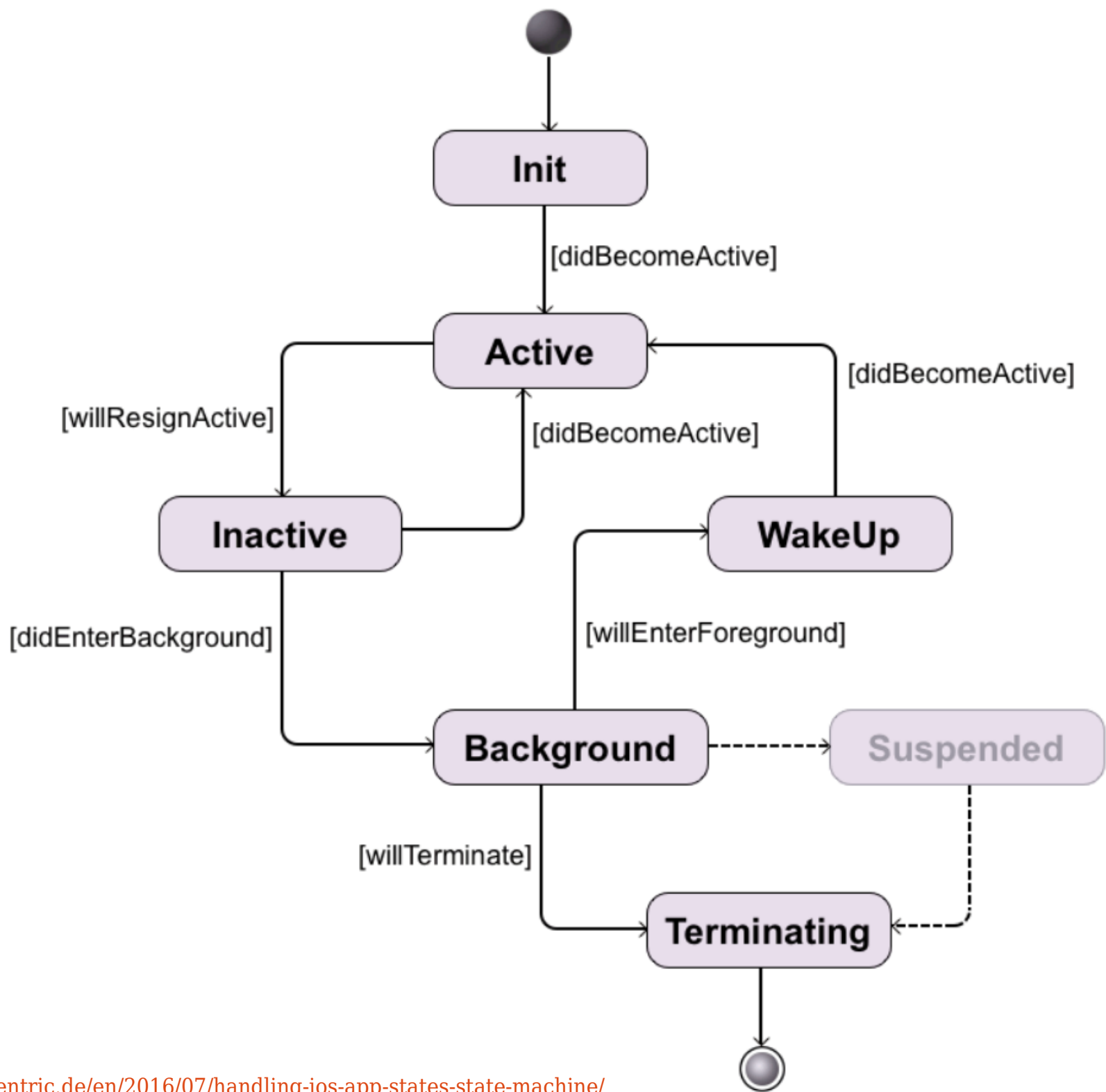
New Stuff!

Why does this matter?

## Buttons as Finite-State Machines:

<http://cs103.stanford.edu/tools/button-fsm/>







What exactly is a finite-state machine?

Ready!

Finite-Memory  
Computing Device

*a*

*b*

*c*



# The Model

- The computing device has internal workings that can be in one of finitely many possible configurations.
  - Each **state** in a DFA corresponds to some possible configuration of the internal workings.
- After each button press, the computing device does some amount of processing, then gets to a configuration where it's ready to receive more input.
  - Each **transition** abstracts away how the computation is done and just indicates what the ultimate configuration looks like.
- After the user presses the “done” button, the computer outputs either YES or NO.
  - The **accepting** and **rejecting** states of the machine model what happens when that button is pressed.

# Computers as Finite Automata

- My computer has 12GB of RAM and 150GB of hard disk space.
- That's a total of 162GB of memory, which is 1,391,569,403,904 bits.
- There are “only”  $2^{1,391,569,403,904}$  possible configurations of the memory in my computer.
- You could in principle build a DFA representing my computer, where there's one symbol per type of input the computer can receive.

# A Powerful Intuition

- ***Regular languages correspond to problems that can be solved with finite memory.***
  - At each point in time, we only need to store one of finitely many pieces of information.
- Nonregular languages, in a sense, correspond to problems that cannot be solved with finite memory.
- Since every computer ever built has finite memory, in a sense, nonregular languages correspond to problems that cannot be solved by physical computers!

# Finding Nonregular Languages

# Finding Nonregular Languages

- To prove that a language is regular, we can just find a DFA, NFA, or regex for it.
- To prove that a language is not regular, we need to prove that there are no possible DFAs, NFAs, or regexes for it.
  - **Claim:** We can actually just prove that there's no DFA for it. Why is this?
- ***This sort of argument will be challenging.*** Our arguments will be somewhat technical in nature, since we need to rigorously establish that no amount of creativity could produce a DFA for a given language.
- Let's see an example of how to do this.

# A Simple Language

- Let  $\Sigma = \{\mathbf{a}, \mathbf{b}\}$  and consider the following language:

$$E = \{\mathbf{a}^n\mathbf{b}^n \mid n \in \mathbb{N}\}$$

- $E$  is the language of all strings of  $n$  **a**'s followed by  $n$  **b**'s:

$$\{\varepsilon, \mathbf{ab}, \mathbf{aabb}, \mathbf{aaabbb}, \mathbf{aaaabbbb}, \dots\}$$

- Is this language regular? Let's see!



# An Attempt

- Let's try to write a regex for

$$E = \{ \mathbf{a^n b^n} \mid n \in \mathbb{N} \}.$$

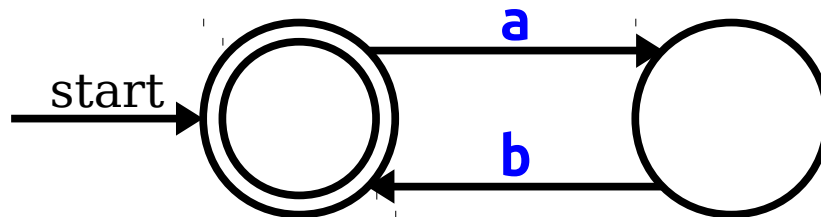
- Does  $\mathbf{a^*b^*}$  work?
- How about  $\mathbf{(ab)^*}$ ?
- How about  $\mathbf{\epsilon \cup ab \cup a^2b^2 \cup a^3b^3}$ ?

# Another Attempt

- Let's try to design an NFA for

$$E = \{ \mathbf{a}^n \mathbf{b}^n \mid n \in \mathbb{N} \}.$$

- Does this machine work?

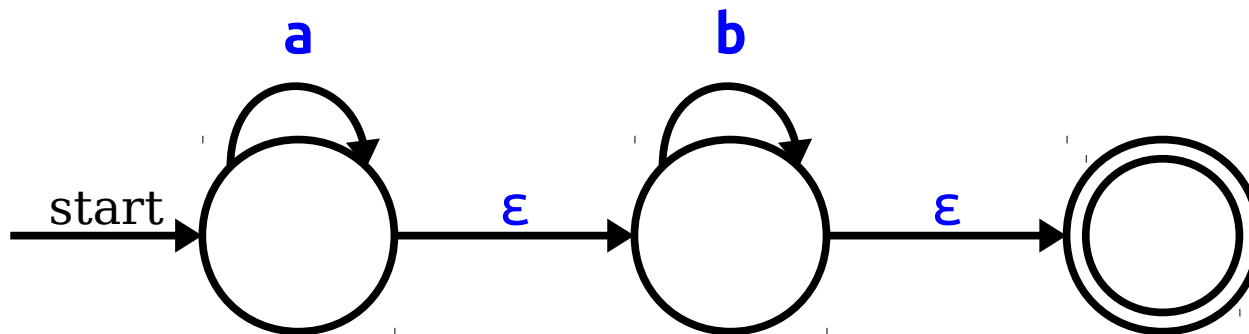


# Another Attempt

- Let's try to design an NFA for

$$E = \{ \mathbf{a}^n \mathbf{b}^n \mid n \in \mathbb{N} \}.$$

- How about this one?

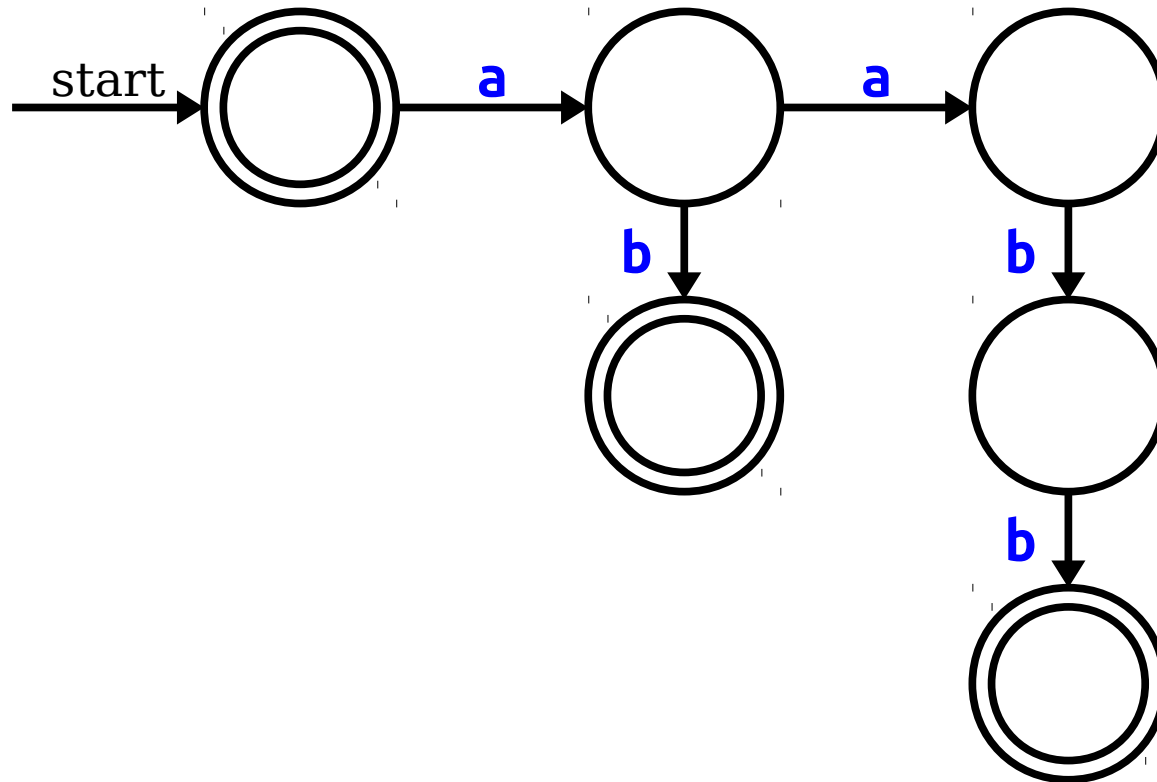


# Another Attempt

- Let's try to design an NFA for

$$E = \{ \mathbf{a}^n \mathbf{b}^n \mid n \in \mathbb{N} \}.$$

- What about this?

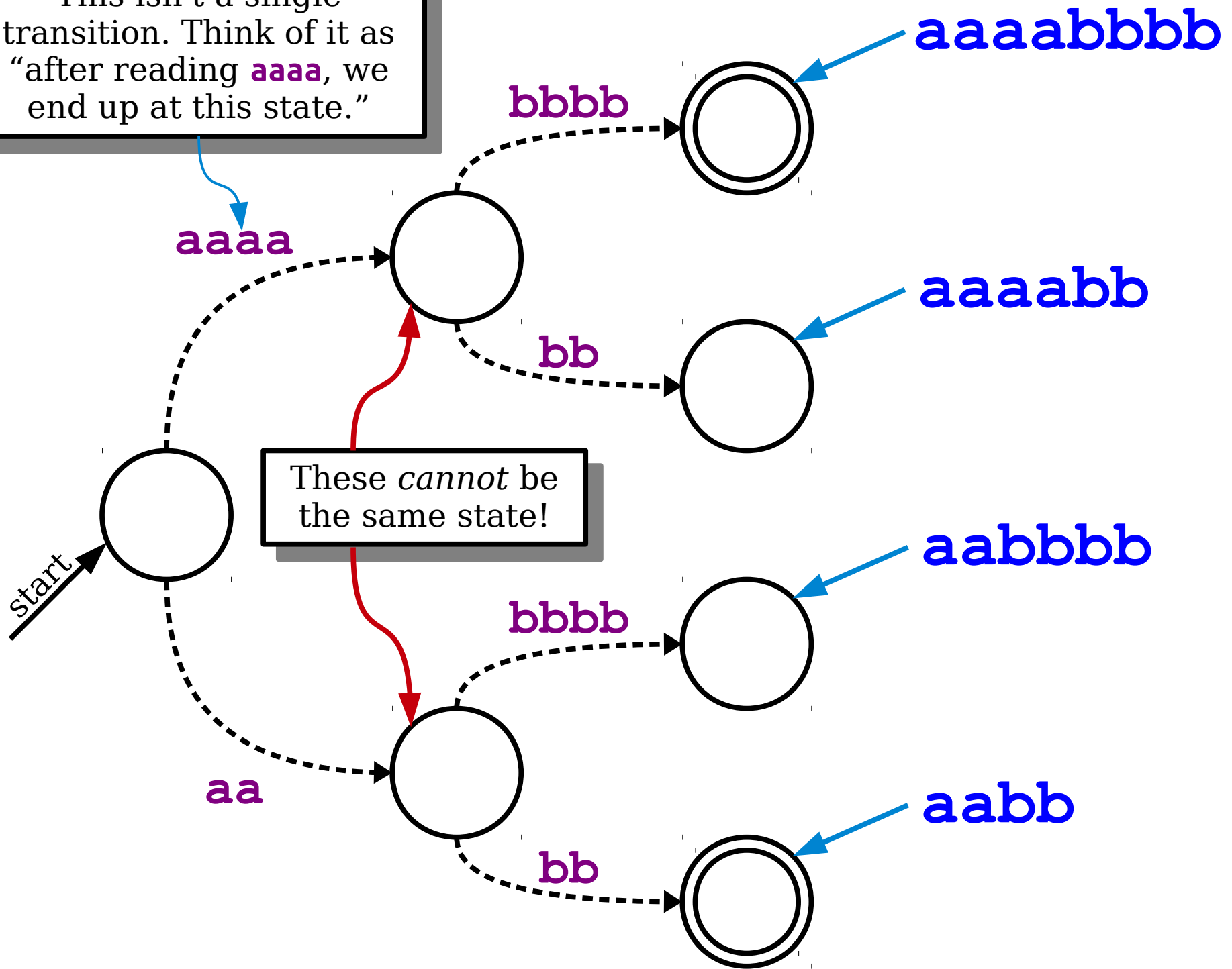


We seem to be running into some trouble.  
Why is that?

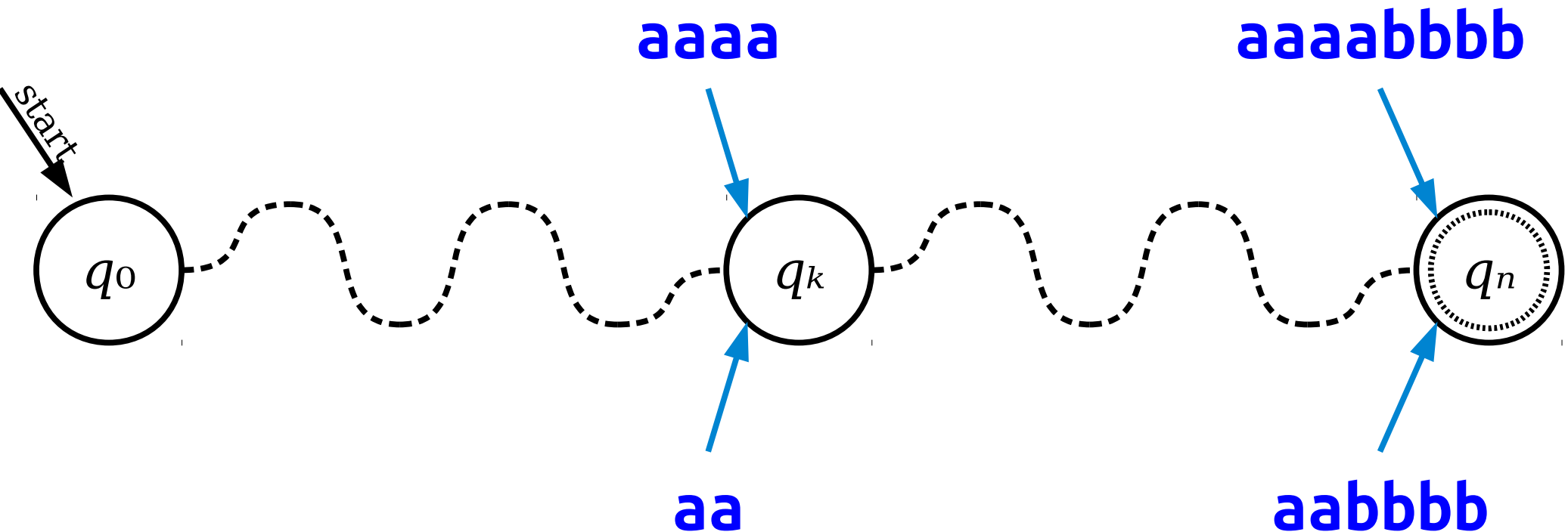
Let's imagine what a DFA for the language  
 $\{ \mathbf{a}^n \mathbf{b}^n \mid n \in \mathbb{N} \}$  would have to look like.

Can we say anything about it?

This isn't a single transition. Think of it as "after reading **aaa**, we end up at this state."



# A Different Perspective



What happens if  $q_n$  is...

...an accepting state?

We accept  **$aabbbb \notin E!$**

...a rejecting state?

We reject  **$aaaabbbb \in E!$**



# What's Going On?

- As you just saw, the strings  $a^4$  and  $a^2$  can't end up in the same state in *any* DFA for  $E = \{a^n b^n \mid n \in \mathbb{N}\}$ .
- Two proof routes:
  - *Direct*: The states you reach for  $a^4$  and  $a^2$  have to behave differently when reading  $b^4$  – in one case it should lead to an accept state, in the other it should lead to a reject state. Therefore, they must be different states.
  - *Contradiction*: Suppose you do end up in the same state. Then  $a^4 b^4$  and  $a^2 b^4$  end up in the same state, so we either reject  $a^4 b^4$  (oops) or accept  $a^2 b^4$  (oops).
- ***Powerful intuition***: Any DFA for  $E$  must keep  $a^4$  and  $a^2$  separated. It needs to remember something fundamentally different after reading those strings.

This idea - that two strings shouldn't end up in the same DFA state - is fundamental to discovering nonregular languages.

Let's go formalize this!

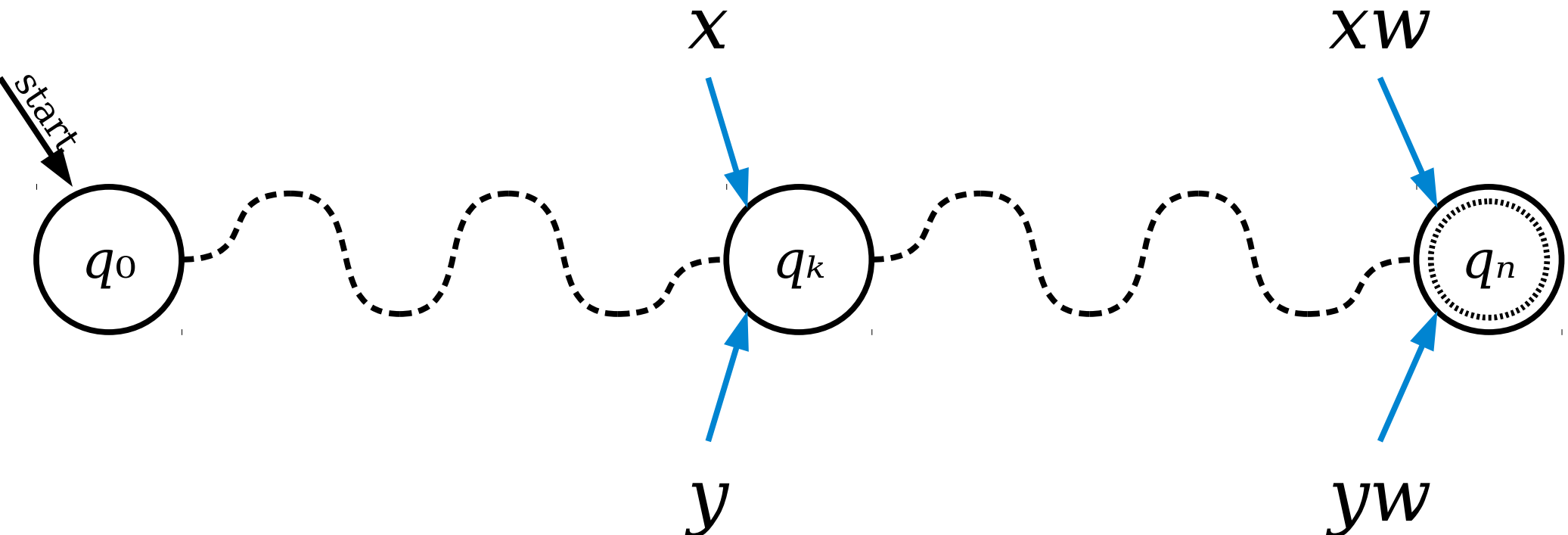
# Distinguishability

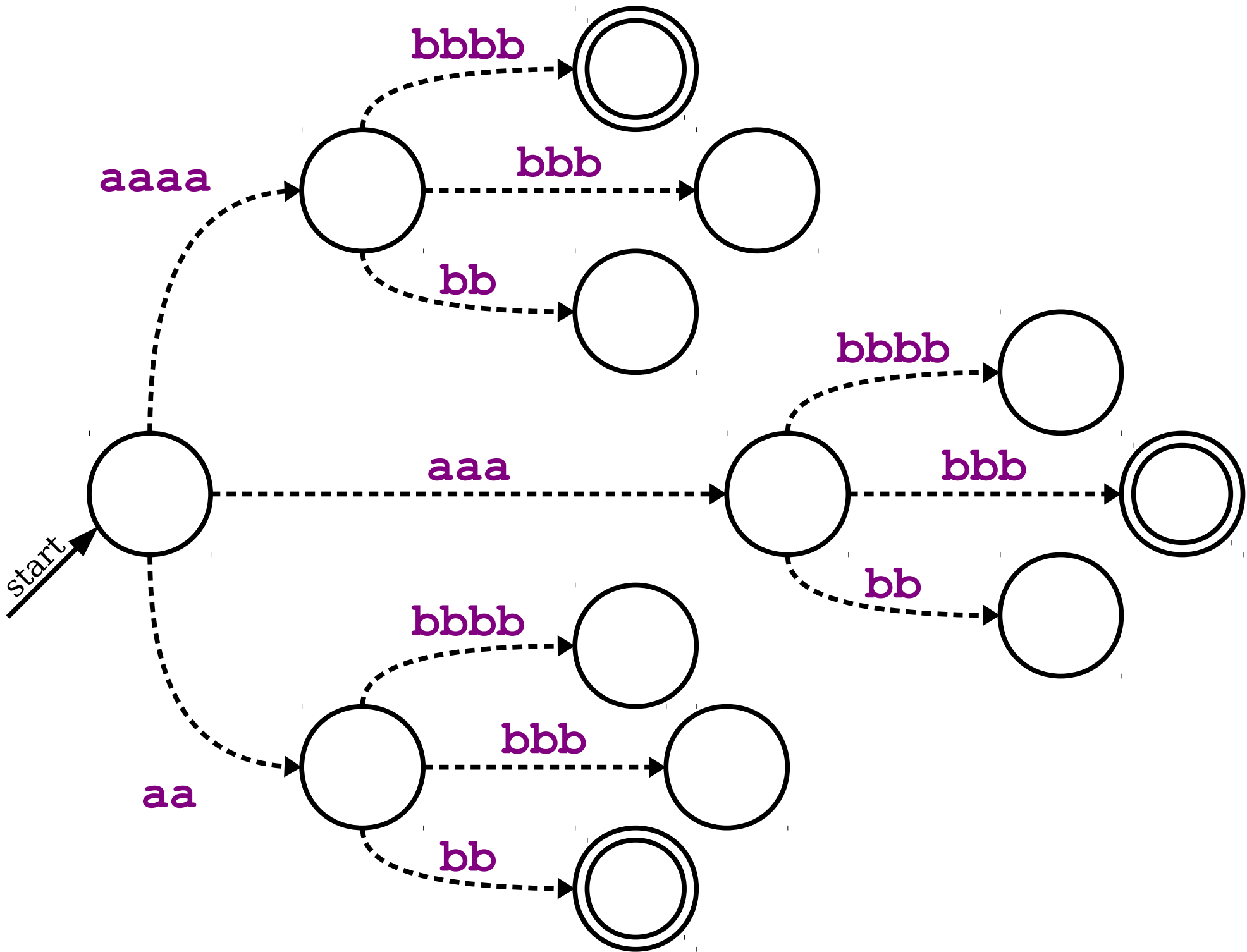
- Let  $L$  be an arbitrary language over  $\Sigma$ .
- Two strings  $x \in \Sigma^*$  and  $y \in \Sigma^*$  are called ***distinguishable relative to  $L$***  if there is a string  $w \in \Sigma^*$  such that exactly one of  $xw$  and  $yw$  is in  $L$ .
- We denote this by writing  $x \not\equiv_L y$ .
- In our previous example, we saw that  $a^2 \not\equiv_E a^4$ .
  - Try appending  $b^4$  to both of them.
- Formally, we say that  $x \not\equiv_L y$  if the following is true:

$$\exists w \in \Sigma^*. (xw \in L \leftrightarrow yw \notin L)$$

# Distinguishability

- **Theorem:** Let  $L$  be an arbitrary language over  $\Sigma$ . Let  $x \in \Sigma^*$  and  $y \in \Sigma^*$  be strings where  $x \not\equiv_L y$ . Then if  $D$  is **any** DFA for  $L$ , then  $D$  must end in different states when run on inputs  $x$  and  $y$ .
- **Proof sketch:**





# Distinguishability

- Let's focus on this language for now:

$$E = \{ \mathbf{a}^n \mathbf{b}^n \mid n \in \mathbb{N} \}$$

**Lemma:** If  $m, n \in \mathbb{N}$  and  $m \neq n$ , then  $\mathbf{a}^m \not\equiv_E \mathbf{a}^n$ .

**Proof:** Let  $\mathbf{a}^m$  and  $\mathbf{a}^n$  be strings where  $m \neq n$ . Then  $\mathbf{a}^m \mathbf{b}^m \in E$  and  $\mathbf{a}^n \mathbf{b}^m \notin E$ . Therefore, we see that  $\mathbf{a}^m \not\equiv_E \mathbf{a}^n$ , as required. ■

# A Bad Combination

- Suppose there is a DFA  $D$  for the language  $E = \{ \mathbf{a}^n \mathbf{b}^n \mid n \in \mathbb{N} \}$ .
- We know the following:
  - Any two strings of the form  $\mathbf{a}^m$  and  $\mathbf{a}^n$ , where  $m \neq n$ , cannot end in the same state when run through  $D$ .
  - There are infinitely many pairs of strings of the form  $\mathbf{a}^m$  and  $\mathbf{a}^n$ .
  - However, there are only *finitely many* states they can end up in, since  $D$  is a deterministic ***finite*** automaton!
- What happens if we put these pieces together?

**Theorem:** The language  $E = \{ \mathbf{a}^n \mathbf{b}^n \mid n \in \mathbb{N} \}$  is not regular.

**Proof:** Suppose for the sake of contradiction that  $E$  is regular. Let  $D$  be a DFA for  $E$ , and let  $k$  be the number of states in  $D$ . Consider the strings  $\mathbf{a}^0, \mathbf{a}^1, \mathbf{a}^2, \dots, \mathbf{a}^k$ . This is a collection of  $k+1$  strings and there are only  $k$  states in  $D$ . Therefore, by the pigeonhole principle, there must be two distinct strings  $\mathbf{a}^m$  and  $\mathbf{a}^n$  that end in the same state when run through  $D$ .

Our lemma tells us that  $\mathbf{a}^m \not\equiv_E \mathbf{a}^n$ , so by our earlier theorem we know that  $\mathbf{a}^m$  and  $\mathbf{a}^n$  cannot end in the same state when run through  $D$ . But this is impossible, since we know that  $\mathbf{a}^m$  and  $\mathbf{a}^n$  do end in the same state when run through  $D$ .

We have reached a contradiction, so our assumption must have been wrong. Therefore,  $E$  is not regular. ■

*We're going to see a simpler proof of this result later on once we've built more machinery. If (hypothetically speaking) you want to prove something like this in the future, we'd recommend not using this proof as a template.*



# What Just Happened?

- ***We've just hit the limit of finite-memory computation.***
- To build a DFA for  $E = \{ \mathbf{a}^n \mathbf{b}^n \mid n \in \mathbb{N} \}$ , we need to have different memory configurations (states) for all possible strings of the form  $\mathbf{a}^n$ .
- There's no way to do this with finitely many possible states!

# Where We're Going

- We just used the idea of *distinguishability* to show that no possible DFA can exist for some language.
- This technique turns out to be pretty powerful.
- We're going to see one more example of this technique in action, then generalize it to an extremely powerful theorem for finding nonregular languages.

# More Nonregular Languages

# Another Language

- Consider the following language  $L$  over the alphabet  $\Sigma = \{a, b, \underline{?}\}$ :

$$EQ = \{ w\underline{?}w \mid w \in \{a, b\}^* \}$$

- $EQ$  is the language all strings consisting of the same string of  $a$ 's and  $b$ 's twice, with a  $\underline{?}$  symbol in-between.
- Examples:

$$ab\underline{?}ab \in EQ$$

$$bbb\underline{?}bbb \in EQ$$

$$\underline{?} \in EQ$$

$$ab\underline{?}ba \notin EQ$$

$$bbb\underline{?}aaa \notin EQ$$

$$b\underline{?} \notin EQ$$

# Another Language

$$EQ = \{ w \stackrel{?}{=} w \mid w \in \{a, b\}^* \}$$

- This language corresponds to the following problem:

**Given strings  $x$  and  $y$ , does  $x = y$ ?**

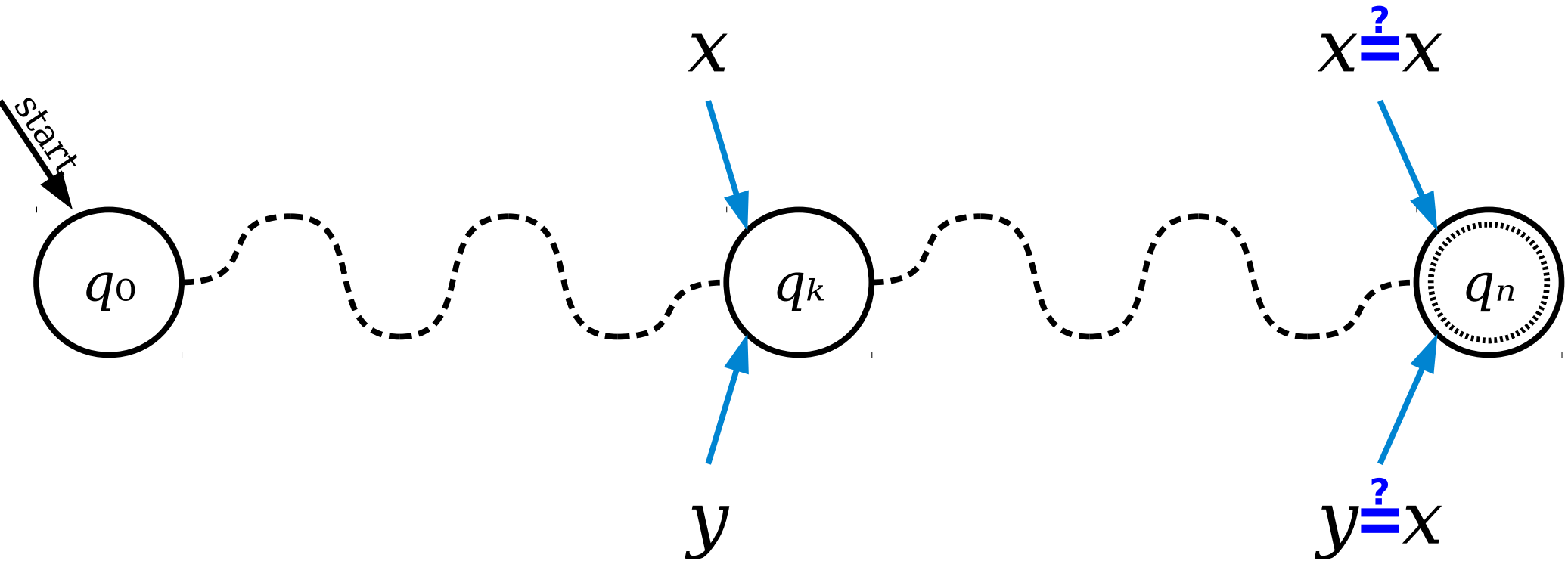
- Justification:  $x = y$  iff  $x \stackrel{?}{=} y \in EQ$ .
- Is this language regular?

# The Intuition

$$EQ = \{ w \stackrel{?}{=} w \mid w \in \{a, b\}^* \}$$

- Intuitively, any machine for  $EQ$  has to be able to remember the contents of everything to the left of the  $\stackrel{?}{=}$  so that it can match them against the contents of the string to the right of the  $\stackrel{?}{=}$ .
- There are infinitely many possible strings we can see, but we only have finite memory to store which string we saw.
- That's a problem... can we formalize this?

# The Intuition



What happens if  $q_n$  is...

...an accepting state?

We accept  $y?x \notin EQ!$

...a rejecting state?

We reject  $x?x \in EQ!$

# Distinguishability

- Let's focus on this language for now:

$$EQ = \{ w \stackrel{?}{=} w \mid w \in \{a, b\}^* \}$$

**Lemma:** If  $x, y \in \{a, b\}^*$  and  $x \neq y$ , then  $x \not\stackrel{?}{=}_{EQ} y$ .

**Proof:** Let  $x$  and  $y$  be two distinct, arbitrary strings from  $\{a, b\}^*$ . Then we see that  $x \stackrel{?}{=} x \in EQ$  and  $y \stackrel{?}{=} x \notin EQ$ , so we conclude that  $x \not\stackrel{?}{=}_{EQ} y$ , as required. ■



**Theorem:** The language  $EQ = \{ w^?w \mid w \in \{a, b\}^* \}$  is not regular.

**Proof:** Suppose for the sake of contradiction that  $EQ$  is regular. Let  $D$  be a DFA for  $EQ$  and let  $k$  be the number of states in  $D$ . Consider any  $k+1$  distinct strings in  $\{a, b\}^*$ . Because  $D$  only has  $k$  states, by the pigeonhole principle there must be at least two strings  $x$  and  $y$  that, when run through  $D$ , end in the same state.

Our lemma tells us that  $x \not\equiv_{EQ} y$ . By our earlier theorem, this means that  $x$  and  $y$  cannot end in the same state when run through  $D$ . But this is impossible, since specifically chose  $x$  and  $y$  to end in the same state when run through  $D$ .

We have reached a contradiction, so our assumption must have been wrong. Thus  $EQ$  is not regular. ■

**Time-Out for Announcements!**

# Midterm Exam Logistics

- The next midterm is **Monday, November 13<sup>th</sup>** from **7:00PM - 10:00PM**. Locations are divvied up by last (family) name:
  - Abb - Hal: Go to **Hewlett 201**.
  - Han - Zwa: Go to **Hewlett 200**.
- The exam focuses on Lecture 06 – 13 (binary relations through induction) and PS3 – PS5. Finite automata onward is *not* tested.
  - Topics from earlier in the quarter (proofwriting, first-order logic, set theory, etc.) are also fair game, but that's primarily because the later material builds on this earlier material.
- The exam is closed-book, closed-computer, and limited-note. You can bring a double-sided, 8.5" × 11" sheet of notes with you to the exam, decorated however you'd like.

# Practice Midterm Exam

- To help you prepare for the midterm, we'll be holding a practice midterm exam **tonight** from **7PM - 10PM** in **Bishop Auditorium**.
- The practice midterm exam is composed of what we think is a good representative sample of older midterm questions from across the years. It's probably the best indicator of what you should expect to see.
- Course staff will be on hand to answer your questions.
- Can't make it? We'll release the practice exam and solutions online. Set up your own practice exam time with a small group and work through it under realistic conditions!

Your Questions

# “If we did really badly on Midterm 1 what should we do differently between today and Monday?”

I've uploaded a new handout to the course website with some specific advice about how to get ready. It's titled "How to Improve in CS103" and is the sequel to "How to Succeed in CS103." Briefly:

1. Get targeted practice on your weak spots. Revisit problem set or exam questions that you didn't do as well on as you'd like. Redo those problems and stop by office hours (or post privately on Piazza) to get feedback on whether you've now got it or whether there are still areas to patch up. Iterate this a few times for full effect.
2. Keep the TAs in the loop when studying. If you solve a practice problem, write up a full solution and have the TAs look over it. We're happy to offer input and suggestions for how to improve, and since there are no "stakes" on those problems we can give extremely detailed feedback.

Back to CS103!

# Comparing Proofs



**Theorem:** The language  $L = [ \text{fill in the blank} ]$  is not a regular language.

**Proof:** Suppose for the sake of contradiction that  $L$  is regular. Let  $D$  be a DFA for  $L$  and let  $k$  be the number of states in  $D$ .

Consider  $[ \text{some } k+1 \text{ specific strings.} ]$  This is a collection of  $k+1$  strings and there are only  $k$  states in  $D$ . Therefore, by the pigeonhole principle, there must be two distinct strings  $x$  and  $y$  that end in the same state when run through  $D$ .

$[ \text{Somehow we know} ]$  that  $x \not\equiv_L y$ . By our earlier theorem we know that  $x$  and  $y$  cannot end in the same state when run through  $D$ . But this is impossible, since we know that  $x$  and  $y$  must end in the same state when run through  $D$ .

We have reached a contradiction, so our assumption must have been wrong. Therefore,  $L$  is not regular. ■

Imagine we have an infinite set of strings  $S$  with the following property:

$$\forall x \in S. \forall y \in S. (x \neq y \rightarrow x \not\equiv_L y)$$

What happens?

For any number of states  $k$ , we need a way to find  $k+1$  strings so that two of them get into the same state...

$D$ .

Consider [ some  $k+1$  specific strings. ] This is a collection of  $k+1$  strings and there are only  $k$  states in  $D$ . Therefore, by the pigeonhole principle, there must be two distinct strings  $x$  and  $y$  that end in the same state when run through  $D$ .

[ Somehow we know ] that  $x \not\equiv_L y$ . By our earlier theorem we know that  $x$  and  $y$  cannot end in the same state when run through  $D$ . But this is impossible, since we know that  $x$  and  $y$  must end in the same state when run through  $D$ .

We have reached a contradiction, so our assumption that  $L$  is not regular must have been wrong. Therefore,  $L$  is not regular.

... and all those strings need to be distinguishable so that we get a contradiction.

# The Myhill-Nerode Theorem

***Theorem:*** Let  $L$  be a language over  $\Sigma$ . If there is a set  $S \subseteq \Sigma^*$  with the following properties, then  $L$  is not regular:

- $S$  is infinite (that is,  $S$  contains infinitely many strings).
- The strings in  $S$  are *pairwise distinguishable relative to  $L$* . That is,

$$\forall x \in S. \forall y \in S. (x \neq y \rightarrow x \not\equiv_L y).$$

**Proof:** Let  $L$  be an arbitrary language over  $\Sigma$ . Let  $S \subseteq \Sigma^*$  be an infinite set of strings with the following property: if  $x, y \in S$  and  $x \neq y$ , then  $x \not\equiv_L y$ . We will show that  $L$  is not regular.

Suppose for the sake of contradiction that  $L$  is regular. This means that there must be some DFA  $D$  for  $L$ . Let  $k$  be the number of states in  $D$ . Since there are infinitely many strings in  $S$ , we can choose  $k+1$  distinct strings from  $S$  and consider what happens when we run  $D$  on all of those strings. Because there are only  $k$  states in  $D$  and we've chosen  $k+1$  strings from  $S$ , by the pigeonhole principle we know that at least two strings from  $S$  must end in the same state in  $D$ . Choose any two such strings and call them  $x$  and  $y$ .

Because  $x$  and  $y$  are distinct strings in  $S$ , we know that  $x \not\equiv_L y$ . Our earlier theorem therefore tells us that when we run  $D$  on inputs  $x$  and  $y$ , they must end up in different states. But this is impossible – we chose  $x$  and  $y$  precisely because they end in the same state when run through  $D$ .

We have reached a contradiction, so our assumption must have been wrong. Thus  $L$  is not a regular language. ■

# Using the Myhill-Nerode Theorem

**Theorem:** The language  $E = \{ \mathbf{a}^n \mathbf{b}^n \mid n \in \mathbb{N} \}$  is not regular.

To use the Myhill-Nerode theorem, we need to find an infinite set of strings that are pairwise distinguishable relative to  $E$ .

We know that any two strings of the form  $\mathbf{a}^n$  and  $\mathbf{a}^m$ , where  $n \neq m$ , are distinguishable.

So pick the set  $S = \{ \mathbf{a}^n \mid n \in \mathbb{N} \}$ .

Notice that  $S$  isn't a subset of  $E$ . That's okay: we never said that  $S$  needs to be a subset of  $E$ !

**Theorem:** The language  $E = \{ \mathbf{a}^n \mathbf{b}^n \mid n \in \mathbb{N} \}$  is not regular.

**Proof:** Let  $S = \{ \mathbf{a}^n \mid n \in \mathbb{N} \}$ . This set is infinite because it contains one string for each natural number. Now, consider any strings  $\mathbf{a}^n, \mathbf{a}^m \in S$  where  $\mathbf{a}^n \neq \mathbf{a}^m$ . Then  $\mathbf{a}^n \mathbf{b}^n \in E$  and  $\mathbf{a}^m \mathbf{b}^n \notin E$ . Consequently,  $\mathbf{a}^n \not\equiv_E \mathbf{a}^m$ . Therefore, by the Myhill-Nerode theorem,  $L$  is not regular. ■

**Theorem:** The language  $EQ = \{ w \stackrel{?}{=} w \mid w \in \{a, b\}^* \}$  is not regular.

To use the Myhill-Nerode theorem, we need to find an infinite set of strings that are pairwise distinguishable relative to  $EQ$ .

We know that any two distinct strings over the alphabet  $\{a, b\}$  are distinguishable.

So pick the set  $S = \{a, b\}^*$ .

Notice that  $S$  isn't a subset of  $EQ$ . That's okay: we never said that  $S$  needs to be a subset of  $EQ$ !



**Theorem:** The language  $EQ = \{ w \stackrel{?}{=} w \mid w \in \{a, b\}^* \}$  is not regular.

**Proof:** Let  $S = \{a, b\}^*$ . This set contains infinitely many strings. Now, consider any  $x, y \in S$  where  $x \neq y$ . Then  $x \stackrel{?}{=} x \in EQ$  and  $y \stackrel{?}{=} x \notin EQ$ . Consequently,  $x \not\equiv_{EQ} y$ . Therefore, by the Myhill-Nerode theorem,  $EQ$  is not regular. ■

# Approaching Myhill-Nerode

- The challenge in using the Myhill-Nerode theorem is finding the right set of strings to use.
- ***General intuition:***
  - Start by thinking about what information a computer “must” remember in order to answer correctly.
  - Choose a group of strings that all require different information.
  - Prove that those strings are distinguishable relative to the language in question.

# Tying Everything Together

- One of the intuitions we hope you develop for DFAs is to have each state in a DFA represent some key piece of information the automaton has to remember.
- If you only need to remember one of finitely many pieces of information, that gives you a DFA.
  - ***You can formalize this!*** Take CS154 for details.
- If you need to remember one of infinitely many pieces of information, you can use the Myhill-Nerode theorem to prove that the language has no DFA.

Where We Stand

# Where We Stand

- We've ended up where we are now by trying to answer the question “what problems can you solve with a computer?”
- We defined a computer to be DFA, which means that the problems we can solve are precisely the regular languages.
- We've discovered several equivalent ways to think about regular languages (DFAs, NFAs, and regular expressions) and used that to reason about the regular languages.
- We now have a powerful intuition for where we ended up: DFAs are finite-memory computers, and regular languages correspond to problems solvable with finite memory.
- Putting all of this together, we have a much deeper sense for what finite memory computation looks like – *and what it doesn't look like!*

# Where We're Going

- What does computation look like with unbounded memory?
- What problems can you solve with unbounded-memory computers?
- What does it even mean to “solve” such a problem?
- And how do we know the answers to any of these questions?

# Next Time

- ***Context-Free Languages***
  - Context-Free Grammars
  - Generating Languages from Scratch