Nonregular Languages

Recap from Last Time

Theorem: The following are all equivalent:

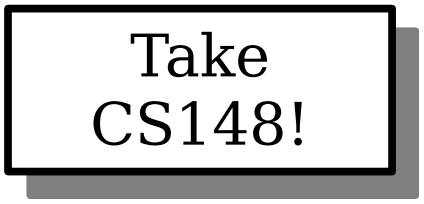
- L is a regular language.
- There is a DFA D such that $\mathscr{L}(D) = L$.
- There is an NFA N such that $\mathscr{L}(N) = L$.
- There is a regular expression R such that $\mathscr{L}(R) = L$.

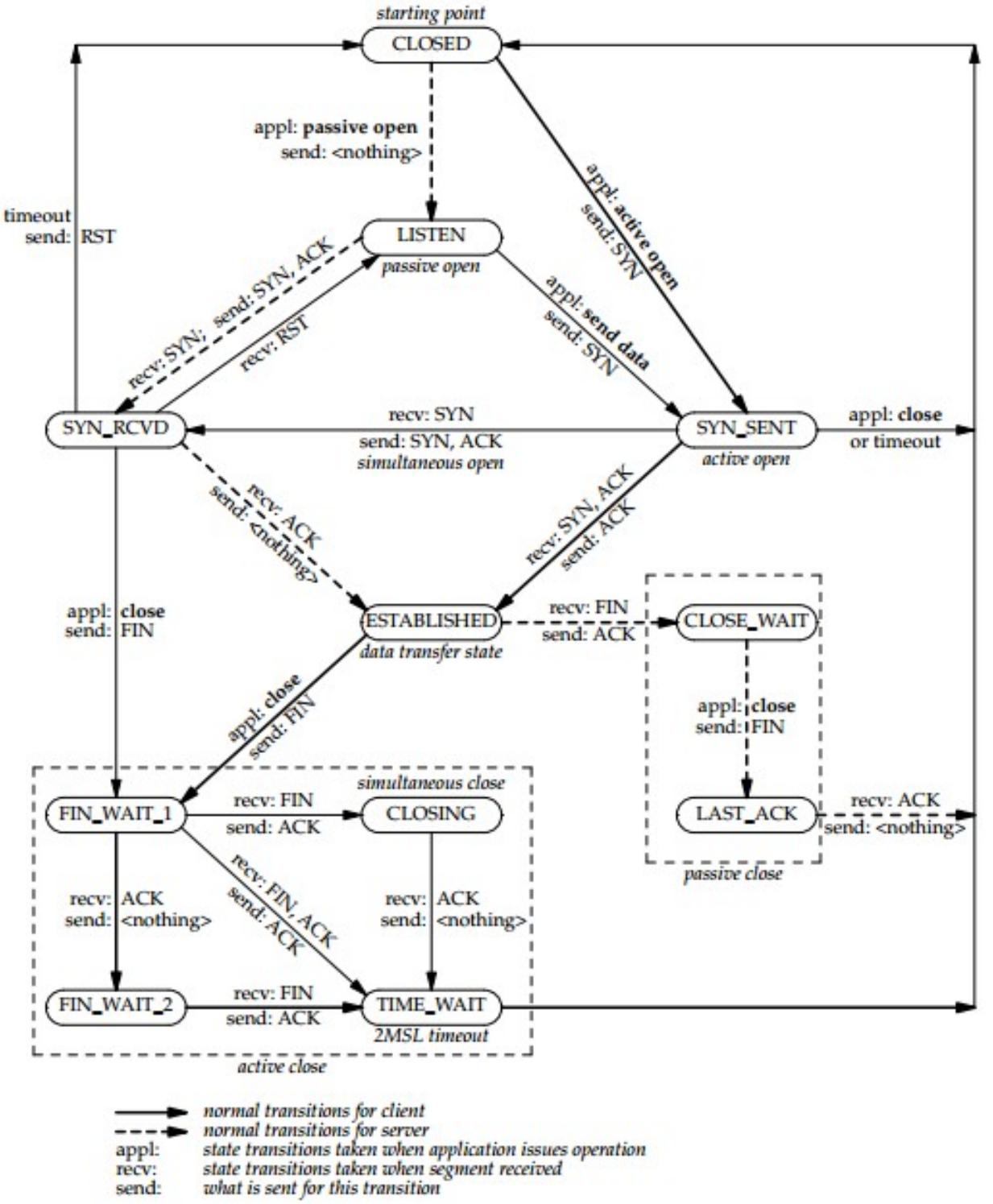
New Stuff!

Why does this matter?

Buttons as Finite-State Machines:

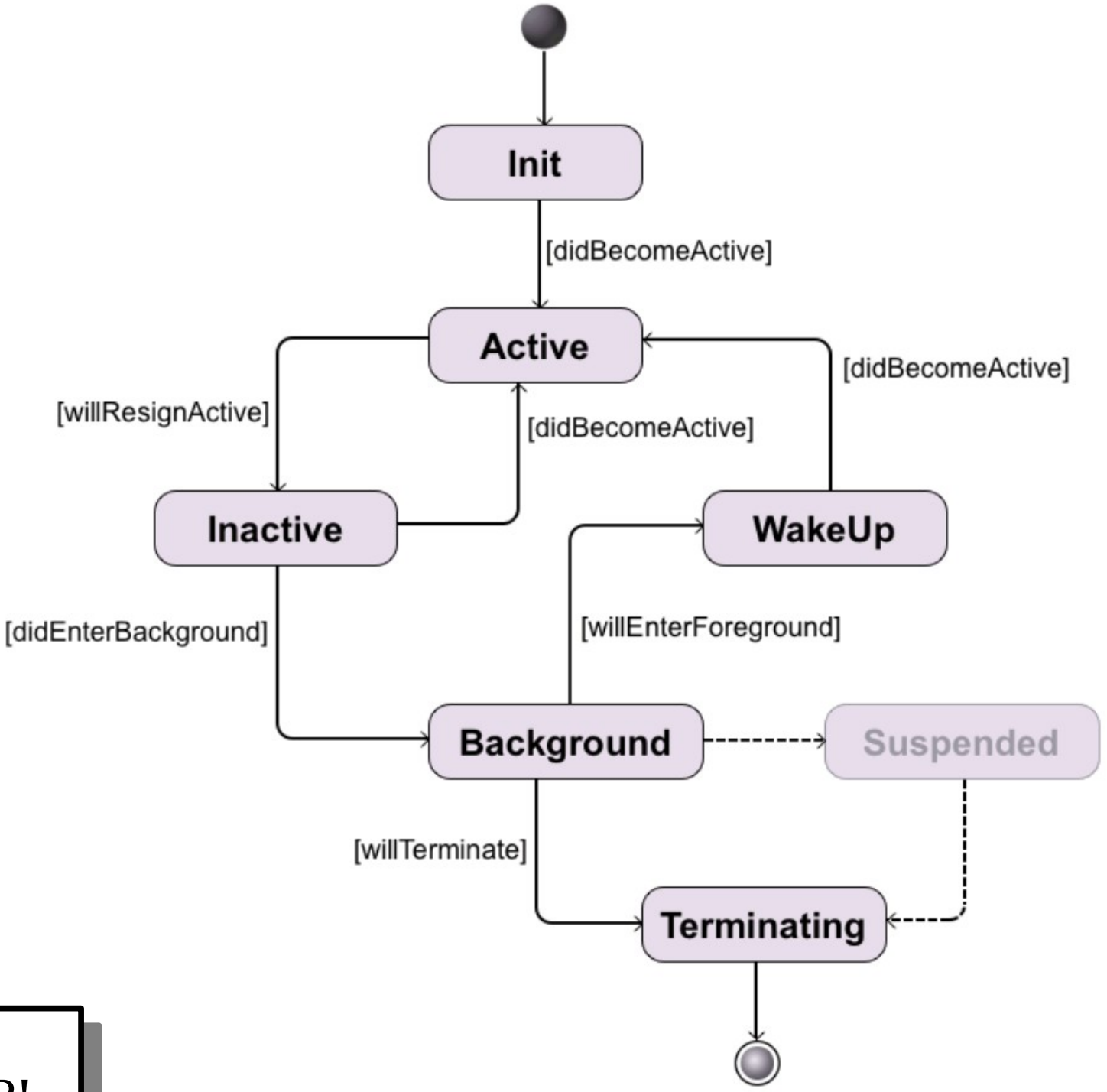
http://cs103.stanford.edu/tools/button-fsm/





Take CS144!

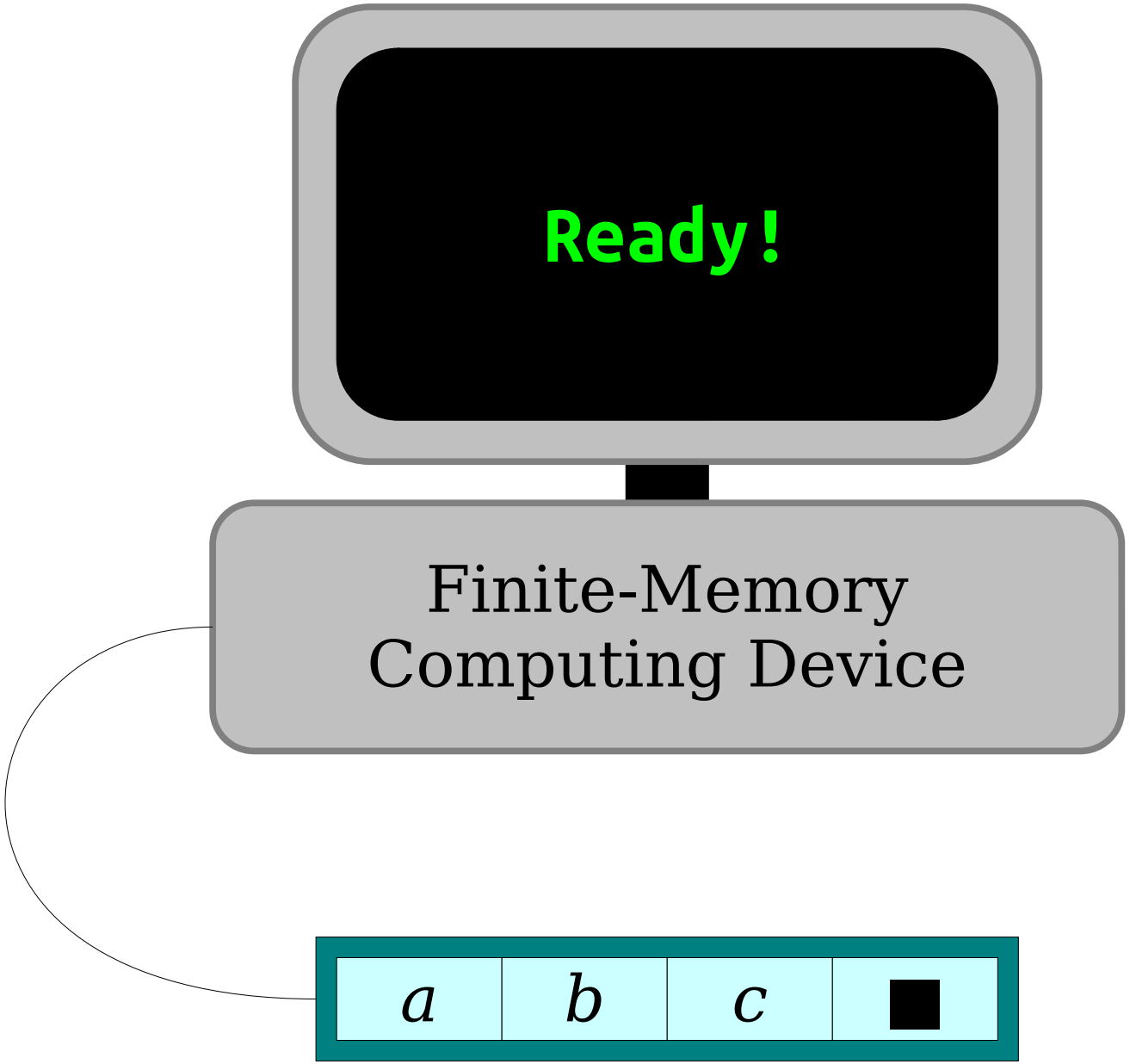
http://www.tti.unipa.it/~gneglia/ip_networks06/slides/TCPIP_State_Transition_Diagram.pdf



Take CS193P!

https://blog.codecentric.de/en/2016/07/handling-ios-app-states-state-machine/

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 about 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 $\Sigma = \{\texttt{a}, \texttt{b}\}$ and consider the following language:

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

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

$$\{\ \varepsilon,\ \texttt{ab},\ \texttt{aabb},\ \texttt{aaabbb},\ \texttt{aaaabbbb},\ \ldots\ \}$$

A Simple Language

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

How many of the following are regular expressions for the language E defined above?

a*b*

(ab)*

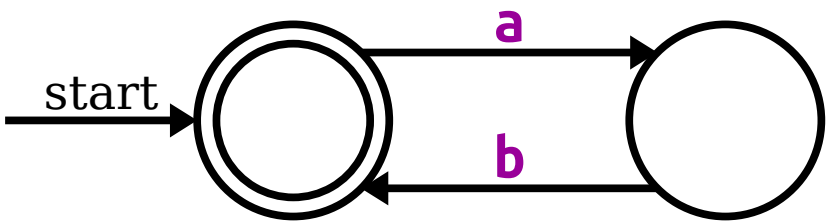
ε U ab U $\mathtt{a}^2\mathtt{b}^2$ U $\mathtt{a}^3\mathtt{b}^3$

Another Attempt

- Let's try to design an NFA for

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

- Does this machine work?

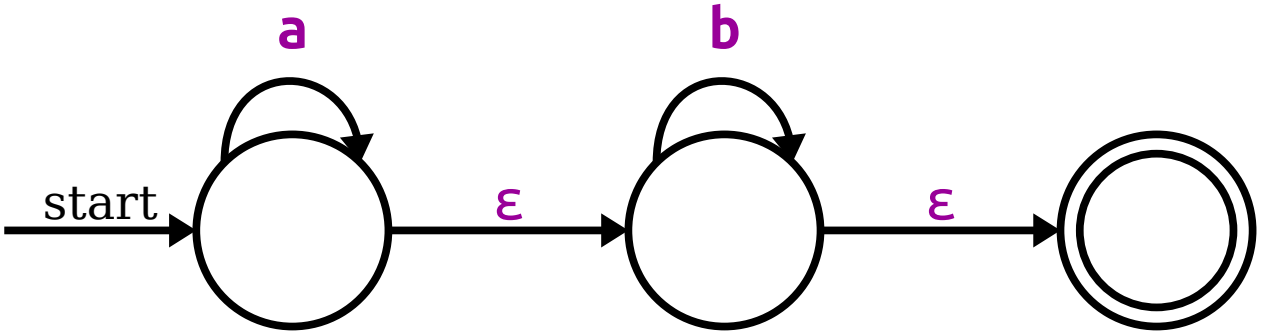


Another Attempt

- Let's try to design an NFA for

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

- How about this one?

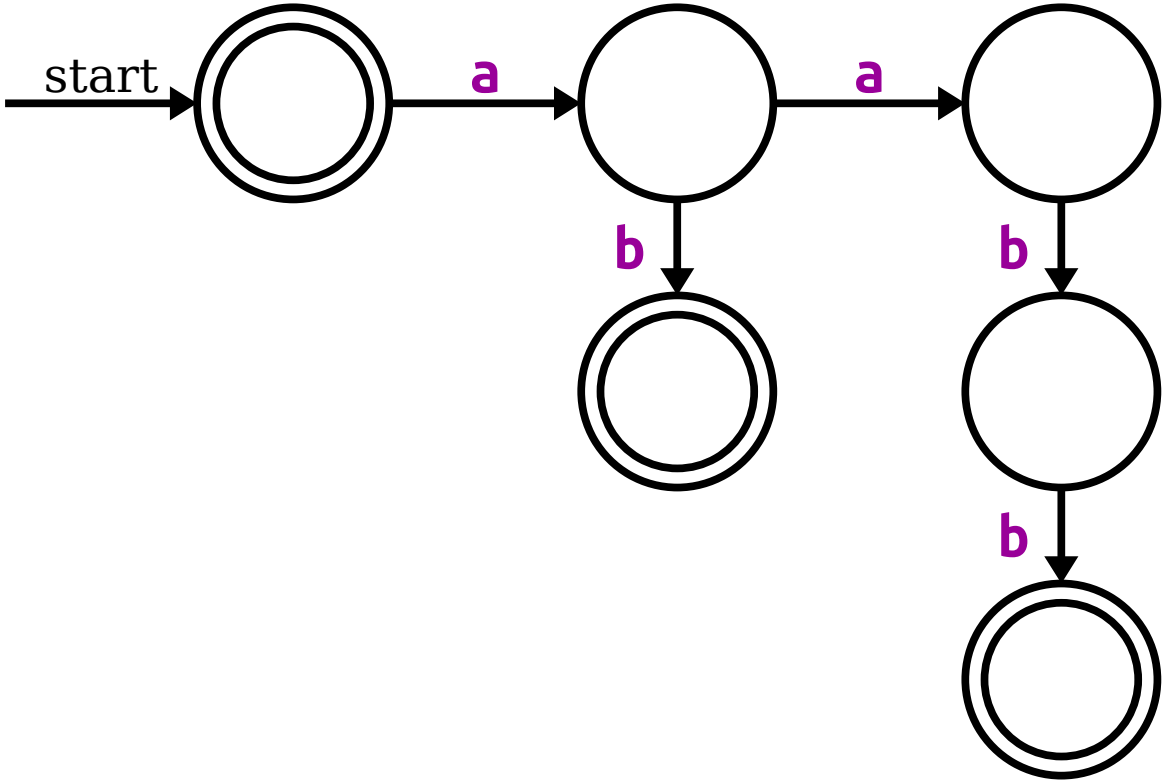


Another Attempt

- Let’s try to design an NFA for

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

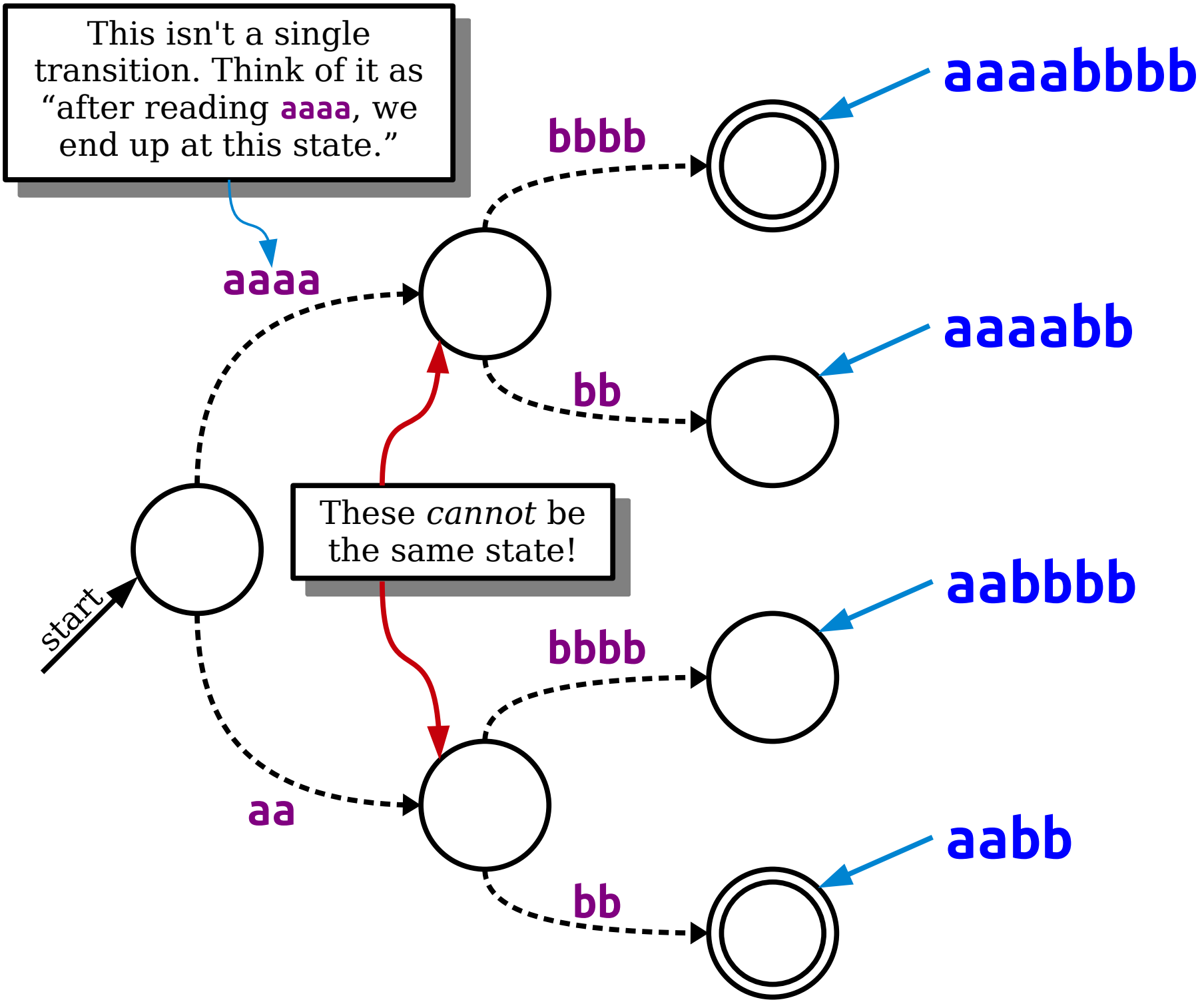
- What about this?



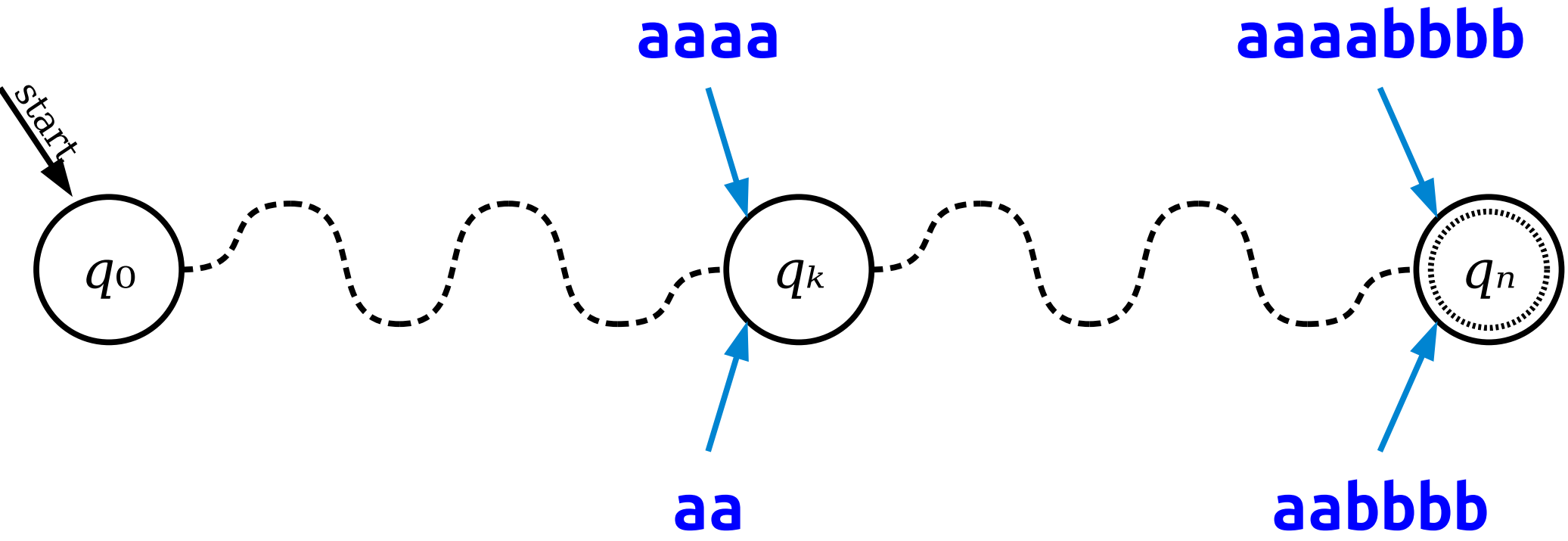
We seem to be running into some trouble.
Why is that?

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

Can we say anything about it?


This isn't a single transition. Think of it as "after reading aaaa, we end up at this state."
aaaa
bbbb
aaaabbbb
bb
aaaabb
These *cannot* be the same state!
start
aa
bbbb
aabbbb
bb
aabb

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 $\mathtt{a}^4$ and $\mathtt{a}^2$ can't end up in the same state in *any* DFA for $E = \{\mathtt{a}^n\mathtt{b}^n \mid n \in \mathbb{N}\}$.
- Two proof routes:
  - *Direct:* The states you reach for $\mathtt{a}^4$ and $\mathtt{a}^2$ have to behave differently when reading $\mathtt{b}^4$ – 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 $\mathtt{a}^4\mathtt{b}^4$ and $\mathtt{a}^2\mathtt{b}^4$ end up in the same state, so we either reject $\mathtt{a}^4\mathtt{b}^4$ (oops) or accept $\mathtt{a}^2\mathtt{b}^4$ (oops).
- ***Powerful intuition:*** Any DFA for E must keep $\mathtt{a}^4$ and $\mathtt{a}^2$ 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!

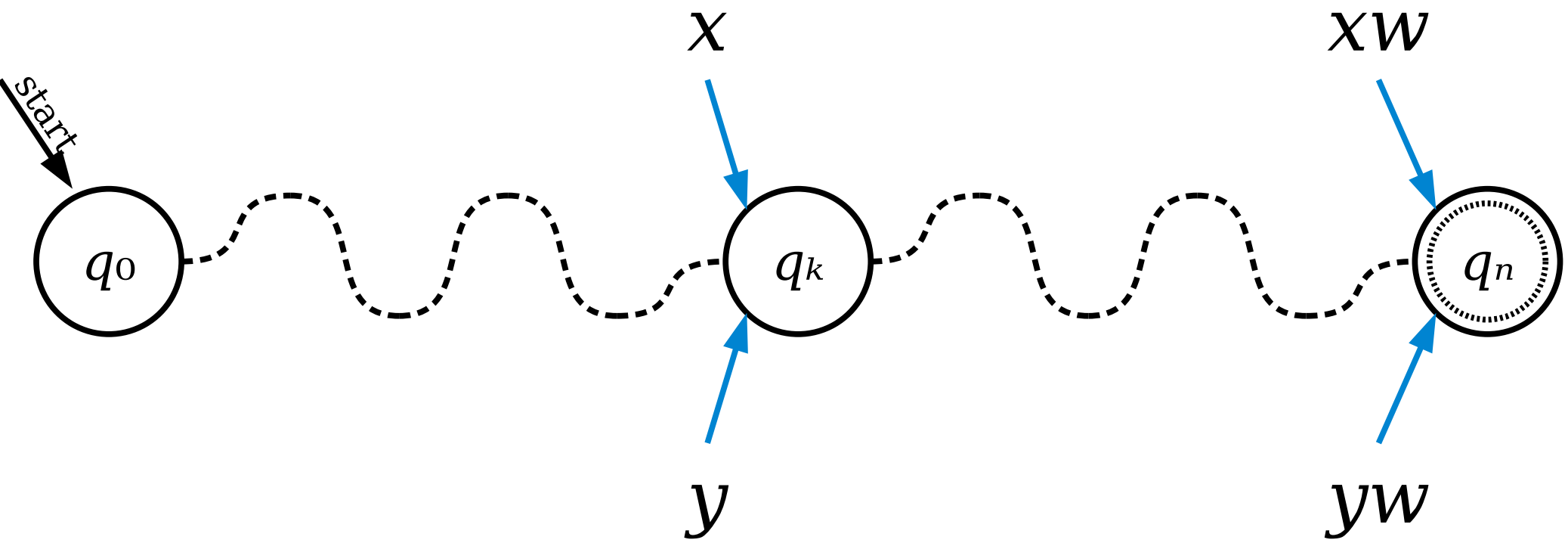
Distinguishability

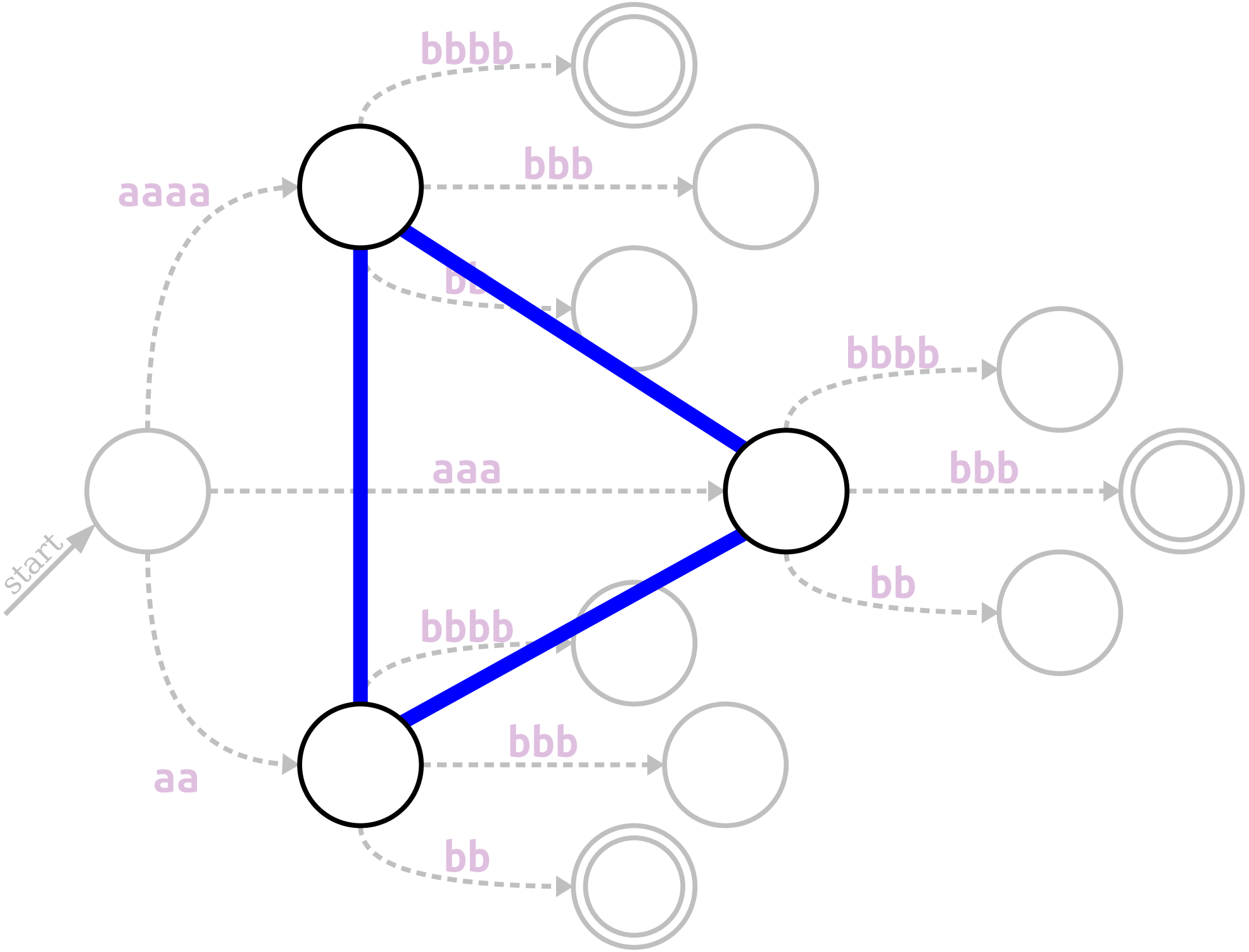
- Let L be an arbitrary language over Σ.
- Two strings $x \in \Sigma^*$ and $y \in \Sigma^*$ are called ***distinguishable relative to L*** if there is a string $w \in \Sigma^*$ such that exactly one of xw and yw is in L.
- We denote this by writing $\boldsymbol{x \not\equiv_L y}$.
- In our previous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

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

Distinguishability

- ***Theorem:*** Let L be an arbitrary language over Σ. 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:***




bbbb
aaaa
bbb
bbbb
aaa
bbb
start
bb
bbbb
bbb
aa
bb

Distinguishability

- Let's focus on this language for now:

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

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

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

A Bad Combination

- Suppose there is a DFA D for the language $E = \{ \mathtt{a}^n\mathtt{b}^n \mid n \in \mathbb{N} \}$.
- We know the following:
  - Any two strings of the form $\mathtt{a}^m$ and $\mathtt{a}^n$, where $m \neq n$, cannot end in the same state when run through D.
  - There are infinitely many strings of the form $\mathtt{a}^m$.
  - 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 $E = \{ \mathtt{a}^n\mathtt{b}^n \mid n \in \mathbb{N} \}$ 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 $\mathtt{a}^0$, $\mathtt{a}^1$, $\mathtt{a}^2$, …, $\mathtt{a}^k$. 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 $\mathtt{a}^m$ and $\mathtt{a}^n$ that end in the same state when run through D.

Our lemma tells us that $\mathtt{a}^m \not\equiv_E \mathtt{a}^n$, so by our earlier theorem we know that $\mathtt{a}^m$ and $\mathtt{a}^n$ cannot end in the same state when run through D. But this is impossible, since we know that $\mathtt{a}^m$ and $\mathtt{a}^n$ 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 $E = \{ \mathtt{a}^n\mathtt{b}^n \mid n \in \mathbb{N} \}$, we need to have different memory configurations (states) for all possible strings of the form $\mathtt{a}^n$.
- 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 $\Sigma = \{\texttt{a}, \texttt{b}, \stackrel{?}{=}\}$:

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

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

$\texttt{ab} \stackrel{?}{=} \texttt{ab} \in EQ$ $\qquad$ $\texttt{bbb} \stackrel{?}{=} \texttt{bbb} \in EQ$ $\qquad$ $\stackrel{?}{=} \in EQ$

$\texttt{ab} \stackrel{?}{=} \texttt{ba} \notin EQ$ $\qquad$ $\texttt{bbb} \stackrel{?}{=} \texttt{aaa} \notin EQ$ $\qquad$ $\texttt{b} \stackrel{?}{=} \notin EQ$

Another Language

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

- This language corresponds to the following problem:

Given strings x, $y \in \{\texttt{a}, \texttt{b}\}^*$, does $x = y$?

- We can think of things this way because

$x = y$ if and only if $x \stackrel{?}{=} y \in EQ$.

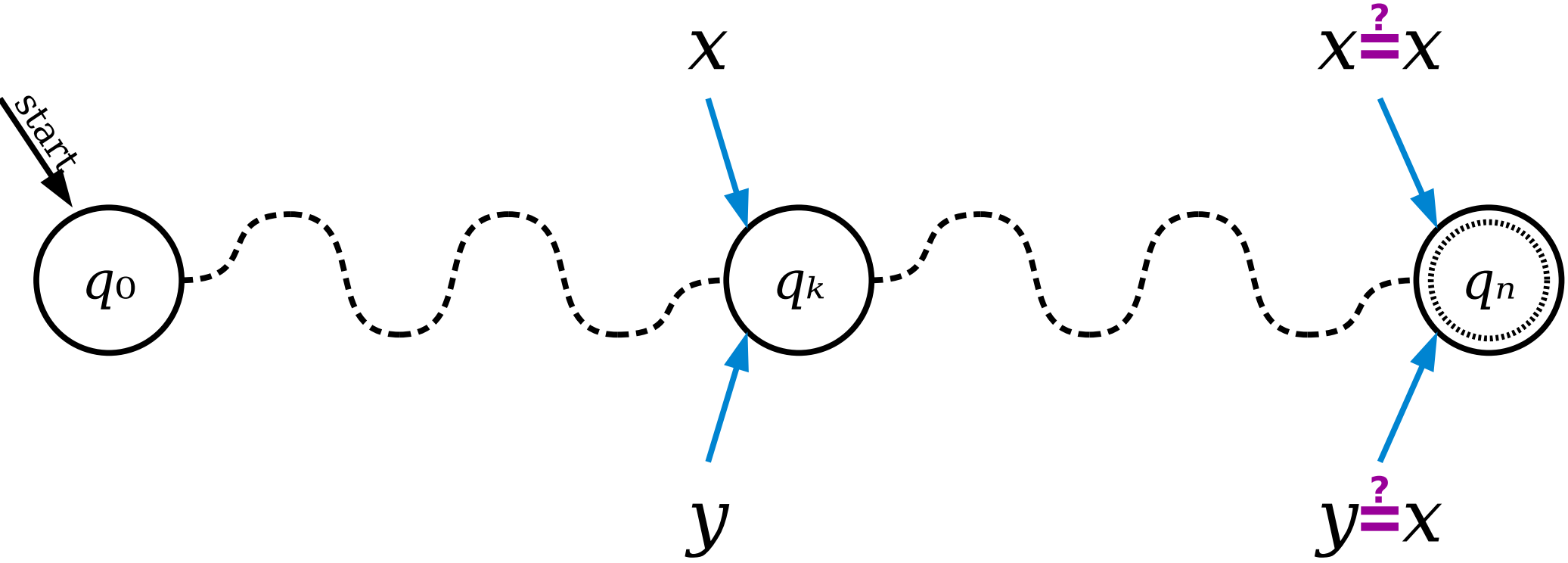
- Is this language regular?

The Intuition

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

- 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 $y \stackrel{?}{=} x \notin EQ$!

...a rejecting state? We reject $x \stackrel{?}{=} x \in EQ$!

Distinguishability

- Let's focus on this language for now:

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

Lemma: If $x, y \in \{\texttt{a}, \texttt{b}\}^*$ and $x \neq y$, then $x \not\equiv_{EQ} y$.

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

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

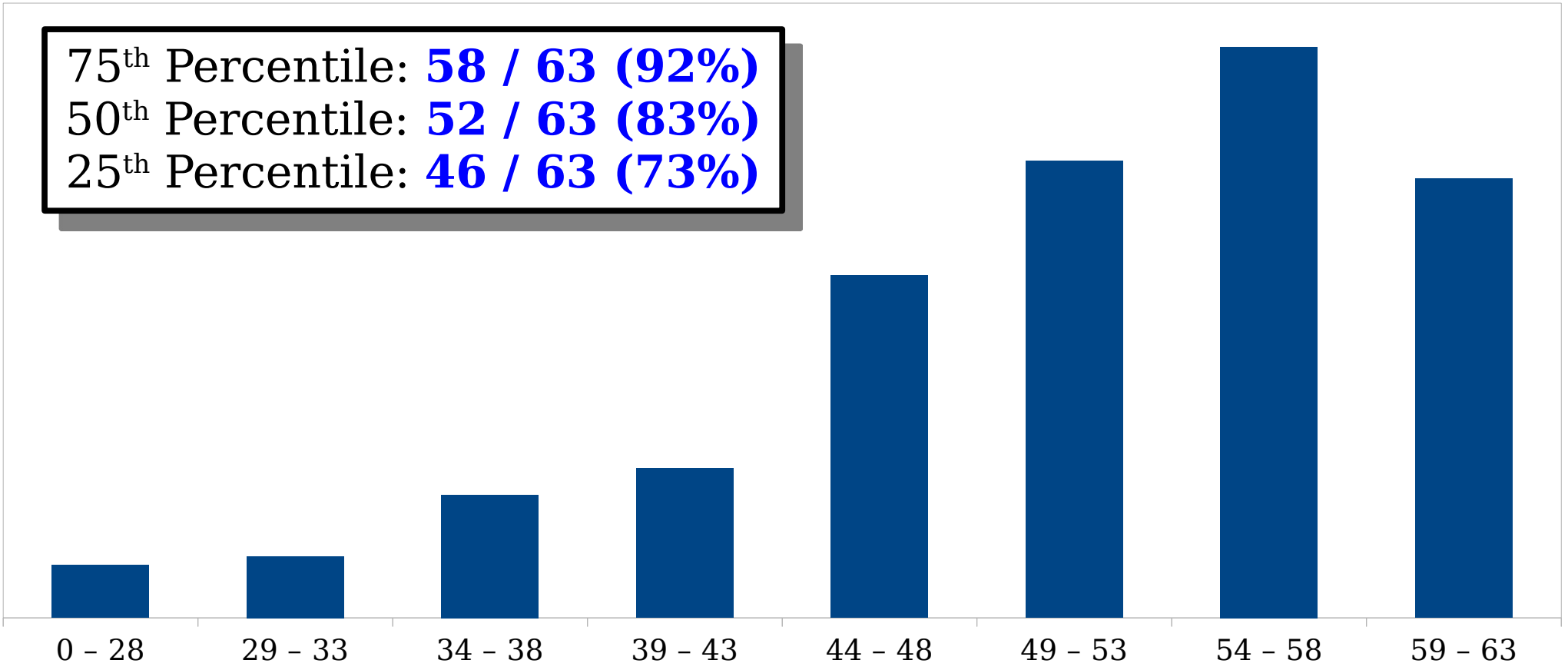
Proof: Suppose for the sake of contradiction that EQ is regular. Let D be a DFA for EQ and let k be the number of states in D. Consider any $k+1$ distinct strings in $\{\texttt{a}, \texttt{b}\}^*$. Because D only has k states, by the pigeonhole principle there must be at least two strings x and y that,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

Problem Set Five Scores



Midterm Exam Logistics

- Our next midterm runs this Friday, November 5th at 2:30PM through this Sunday, November 7th at 2:30PM, Pacific time.
  - That's 49 hours rather than the normal 48. Huzzah!
- Topic coverage is primarily lectures 06 – 13 (functions through induction) and PS3 – PS5. Finite automata and onward won't be tested here.
  - Because the material is cumulative, topics from PS1 – PS2 and Lectures 00 – 05 are also fair game.
- Extra Practice Problems 2 is available on the course website if you want to get more practice with these topics.
- ***We want you to do well on this exam***. Keep in touch and let us know what we can do to help make that happen!

Don't be afraid to take a deep breath & smile!
Cuz you got this :)

try

Your Questions

“My best friend is a philosophy enthusiast and often tells me "CS is just a branch of philosophy." How would you respond to the statement?”

The boundaries between different disciplines are often blurry. Our CS faculty spans the range of folks working on improving the design of buildings using computing (architecture, social psychology, urban design, etc.) and people working on the theoretical limits of computing machines (mathematics, probability, etc.). This is a good thing - it means that there's a lot of cross-pollination of ideas. It also means it's hard to say something is "just" a branch of something.

"Could you talk about the biomedical computation major and some advantages to following that path rather than the traditional CS major (and possibly in contrast to the biocomputation track)?"

There's a decent amount of overlap between the two programs. BMC has a bit more flexibility than biocomputation (there are tracks within BMC; biocomputation is a track within CS). Biocomputation has more of a coding component and does a bit more with AI. If you're interested in exploring this space, it might make sense to work backwards from the program sheets to figure out what works best for you. (That's good general advice with any similar majors.)

Back to CS103!

Comparing Proofs

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

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 $\mathtt{a}^0$, $\mathtt{a}^1$, $\mathtt{a}^2$, …, $\mathtt{a}^k$. 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 $\mathtt{a}^m$ and $\mathtt{a}^n$ that end in the same state when run through D.

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

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

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 $\{\texttt{a}, \texttt{b}\}^*$.These are $k+1$ strings and there are only k states in D. By the pigeonhole principle, there must be two distinct strings x and y from this group that end in the same state when run through D.

Our lemma tells us that $x \not\equiv_{EQ} y$. 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 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. Therefore, EQ is not regular. ■

Theorem: The language $L = [$ regular language.

Proof: Suppose for the sake of [illegible] Let D be a DFA for L and let k be the number of states in 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, s[illegible] have been wrong. Therefore, L is n[illegible]

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…

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

Distinguishing Sets

- Let L be a language over Σ.
- A ***distinguishing set*** for L is a set $S \subseteq \Sigma^*$ where the following is true:

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

If you pick any two strings in S that aren't equal to one another…

… then they're distinguishable relative to L.

Distinguishing Sets

- Let L be a language over Σ.
- A ***distinguishing set*** for L is a set $S \subseteq \Sigma^*$ where the following is true:

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

- As an example, here's a distinguishing set for $E = \{\ \mathtt{a}^n\mathtt{b}^n \mid n \in \mathbb{N}\ \}$:

$$S = \{\ \mathtt{a}^n \mid n \in \mathbb{N}\ \}$$

Distinguishing Sets

- Let L be a language over Σ.
- A ***distinguishing set*** for L is a set $S \subseteq \Sigma^*$ where the following is true:

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

- As an example, here's a distinguishing set for $EQ = \{\ w \stackrel{?}{=} w \mid w \in \{\texttt{a}, \texttt{b}\}^*\}$:

$$S = \{\texttt{a}, \texttt{b}\}^*$$

Theorem (Myhill-Nerode): If L is a language and S is a distinguishing set for L that contains infinitely many strings, then L is not regular.

Proof: Let L be an arbitrary language over Σ and let S be a distinguishing set for L that contains infinitely many strings. 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 $x \neq y$ and S is a distinguishing set for L, 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 $E = \{ \mathtt{a}^n\mathtt{b}^n \mid n \in \mathbb{N} \}$ is not regular.

Proof: Let $S = \{ \mathtt{a}^n \mid n \in \mathbb{N} \}$. We will prove that S is infinite and that S is a distinguishing set for E.

To see that S is infinite, note that S contains one string for each natural number.

To see that S is a distinguishing set for E, consider any strings strings $\mathtt{a}^m$, $\mathtt{a}^n \in S$ where $m \neq n$. Note that $\mathtt{a}^m\mathtt{b}^m \in E$ and that $\mathtt{a}^n\mathtt{b}^m \notin E$. Therefore, we see that $\mathtt{a}^m \not\equiv_E \mathtt{a}^n$, as required.

Since S is infinite and is a distinguishing set for E, by the Myhill-Nerode theorem we see that E is not regular. ■

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

Proof: Let $S = \{\texttt{a}, \texttt{b}\}^*$. We will prove that S is infinite and that S is a distinguishing set for EQ.

To see that S is infinite, note that, for any $n \in \mathbb{N}$, we have $\texttt{a}^n \in S$. Therefore, S contains at least one string for each natural number, so S is infinite.

To see that S is a distinguishing set for EQ, consider any strings $x, y \in S$ where $x \neq y$. Then $x \stackrel{?}{=} x \in EQ$ and $y \stackrel{?}{=} x \notin EQ$. Therefore, $x \not\equiv_{EQ} y$, as required.

Since S is infinite and a distinguishing set for EQ, by the Myhill-Nerode theorem we see that EQ is not regular, as required. ■

Approaching Myhill-Nerode

- The challenge in using the Myhill-Nerode theorem is finding the right set of strings.
- ***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 you have infinitely many strings an that the group of strings is a distinguishing set.

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.
  - This can be made rigorous! 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