

Regular Expressions

Recap from Last Time

Regular Languages

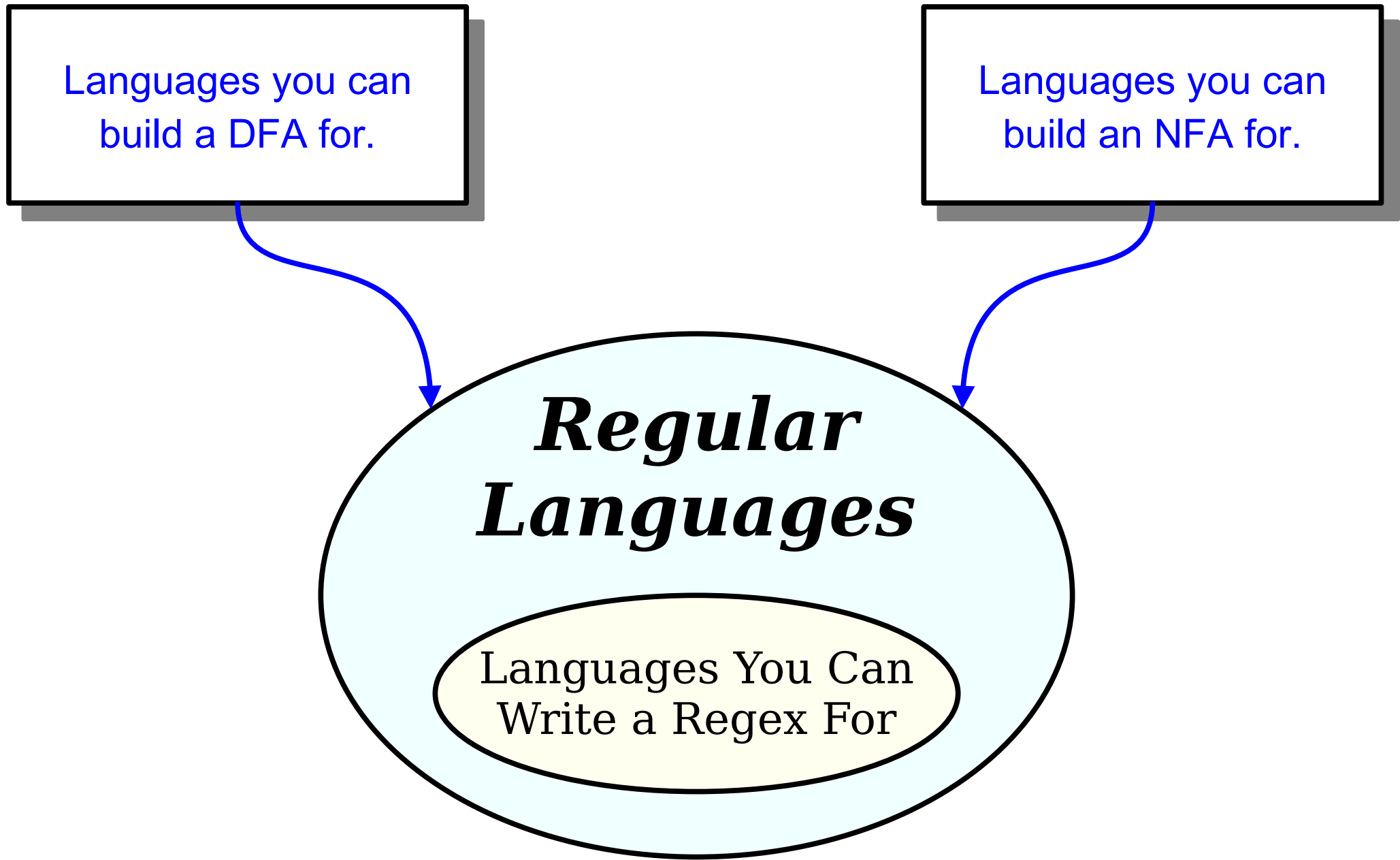
- A language L is a ***regular language*** if there is a DFA D such that $\mathcal{L}(D) = L$.
- ***Theorem:*** The following are equivalent:
 - L is a regular language.
 - There is a DFA for L .
 - There is an NFA for L .

Languages you can
build a DFA for.

Languages you can
build an NFA for.

Regular Languages

Languages You Can
Write a Regex For

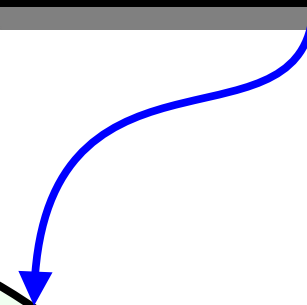
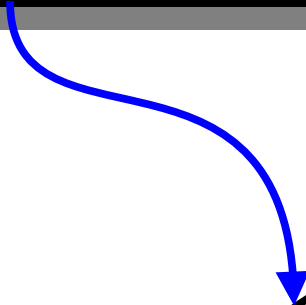


Languages you can
build a DFA for.

Languages you can
build an NFA for.

Regular Languages

Languages You Can
Write a Regex For



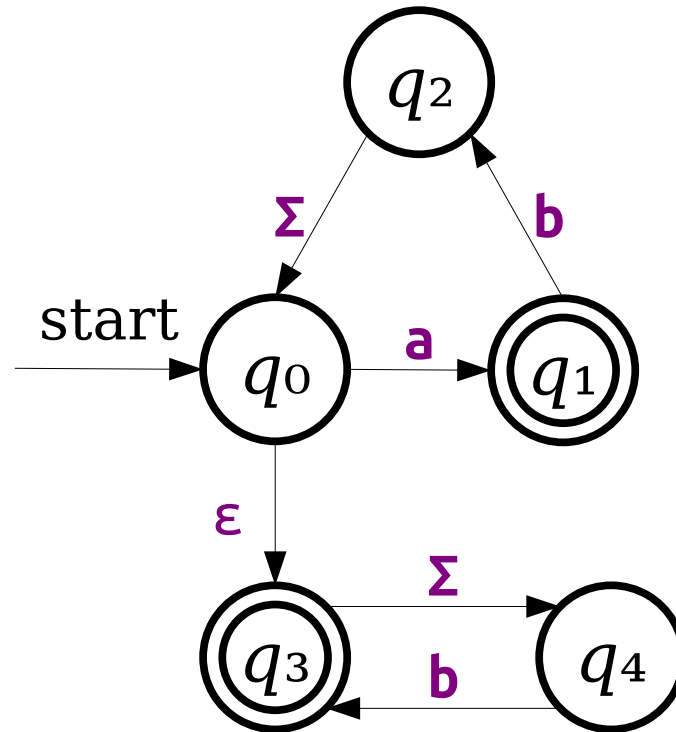
The Power of Regular Expressions

Theorem: If L is a regular language, then there is a regular expression for L .

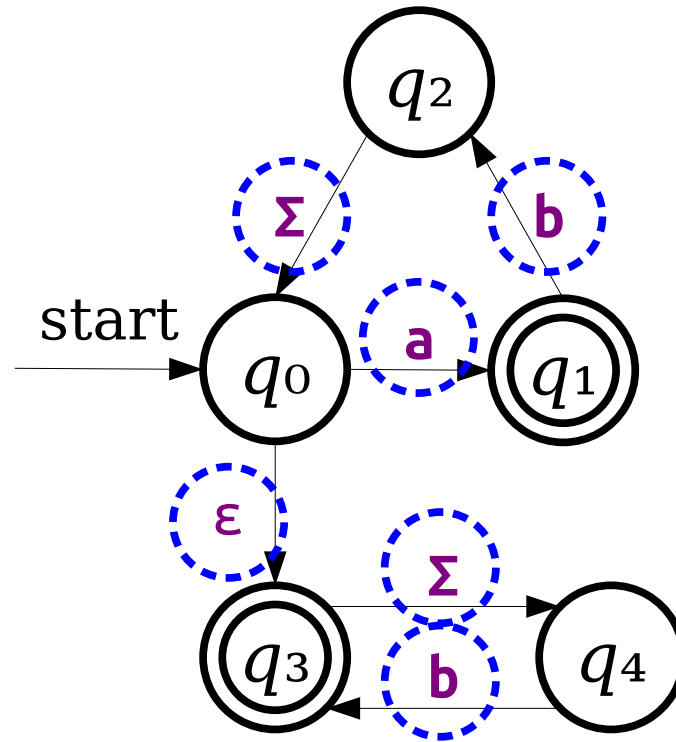
This is not obvious!

Proof idea: Show how to convert an arbitrary NFA into a regular expression.

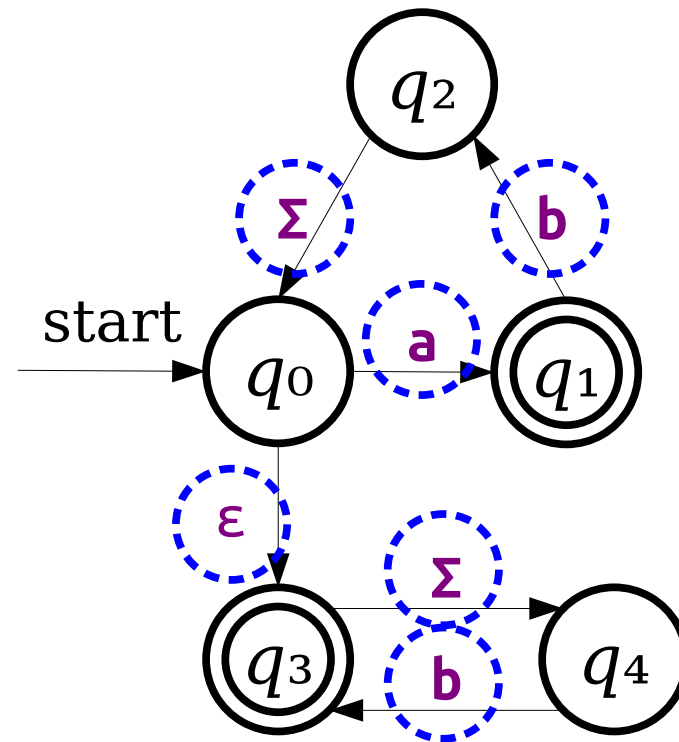
Generalizing NFAs



Generalizing NFAs

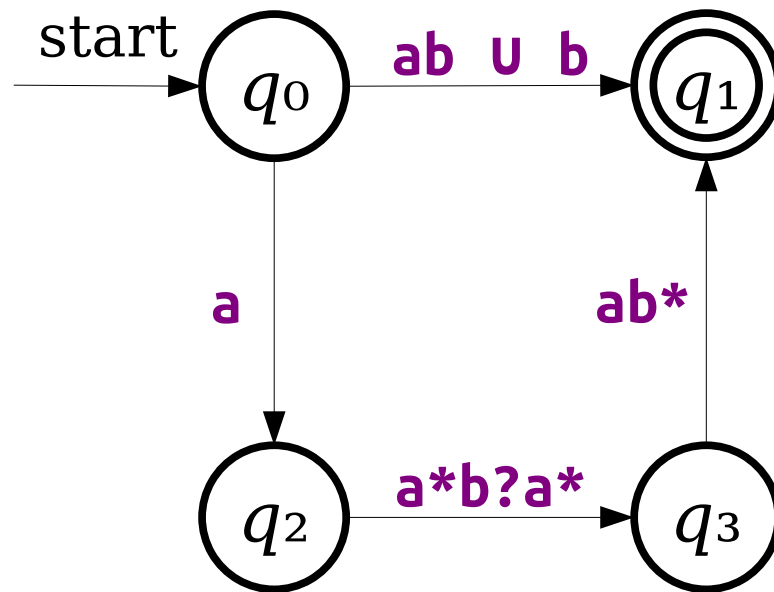


Generalizing NFAs

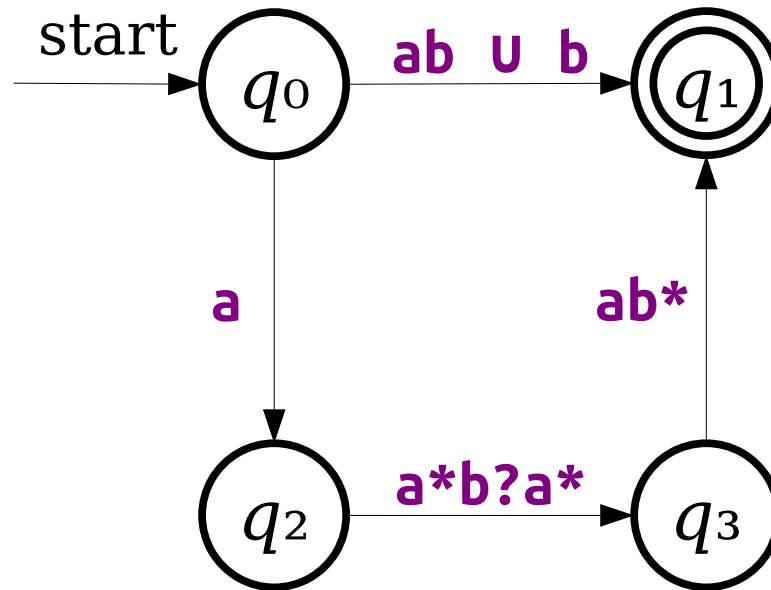


These are all regular expressions!

Generalizing NFAs

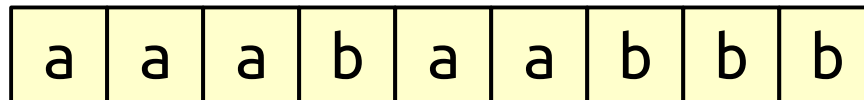
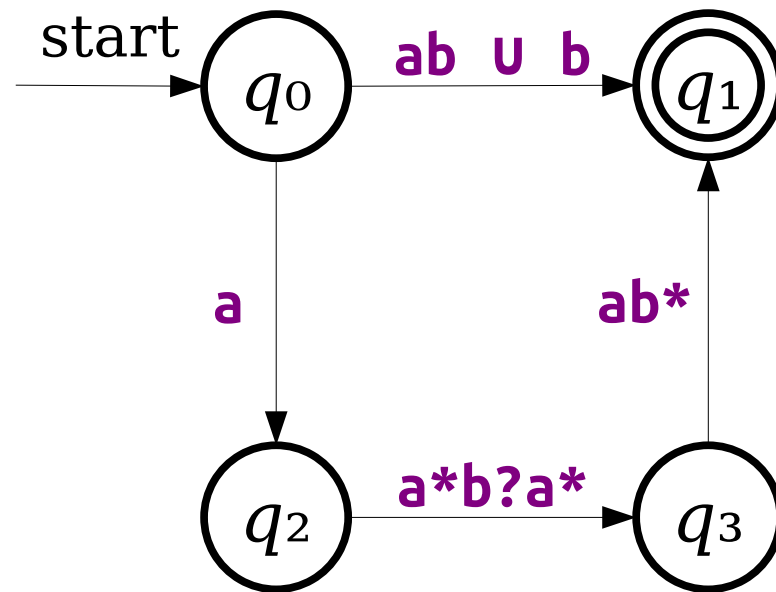


Generalizing NFAs

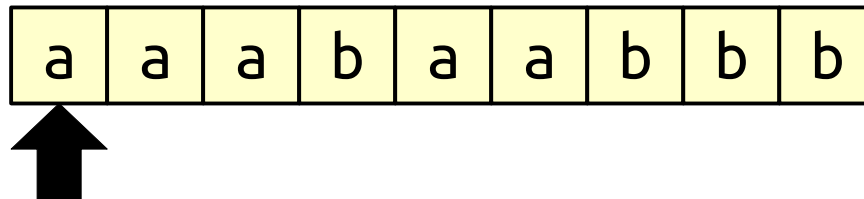
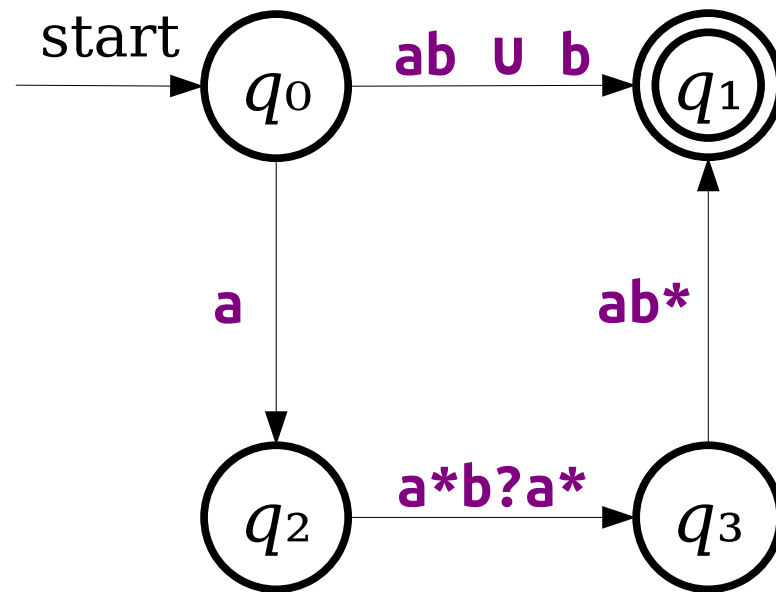


Note: Actual NFAs aren't allowed to have transitions like these. This is just a thought experiment.

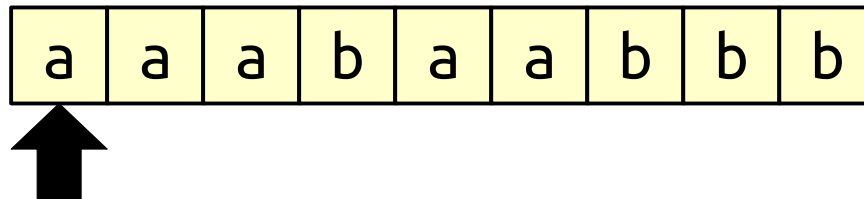
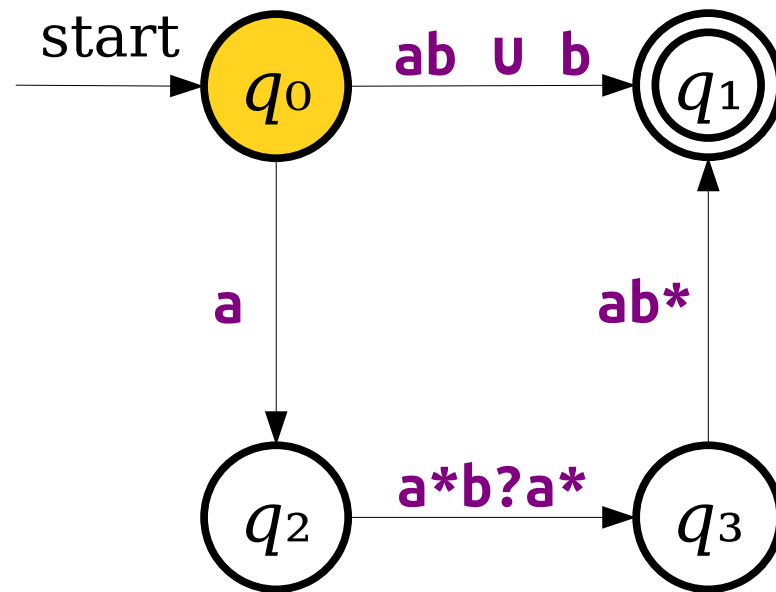
Generalizing NFAs



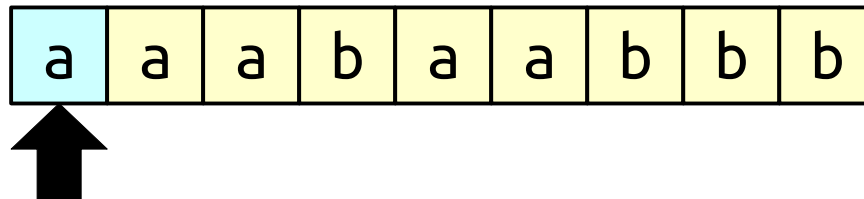
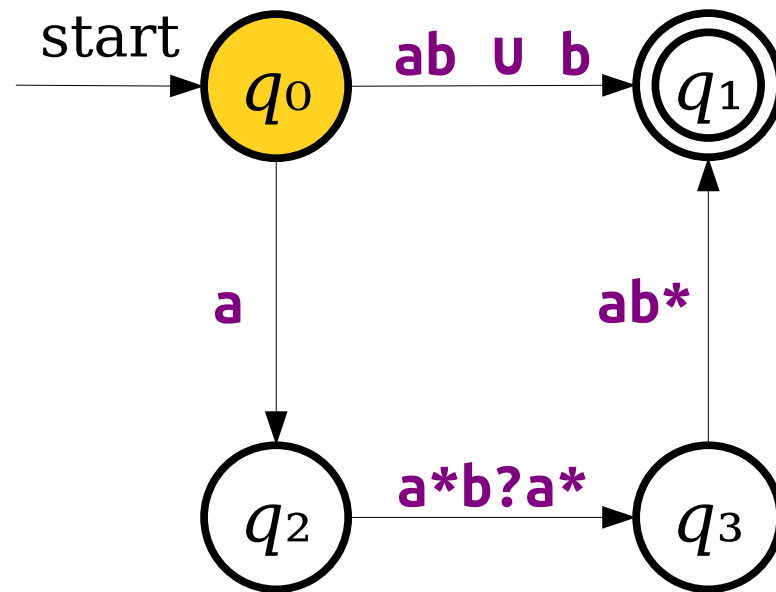
Generalizing NFAs



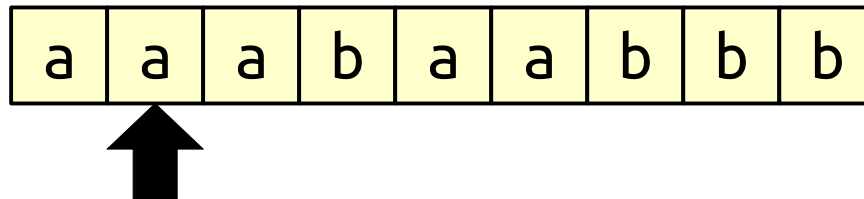
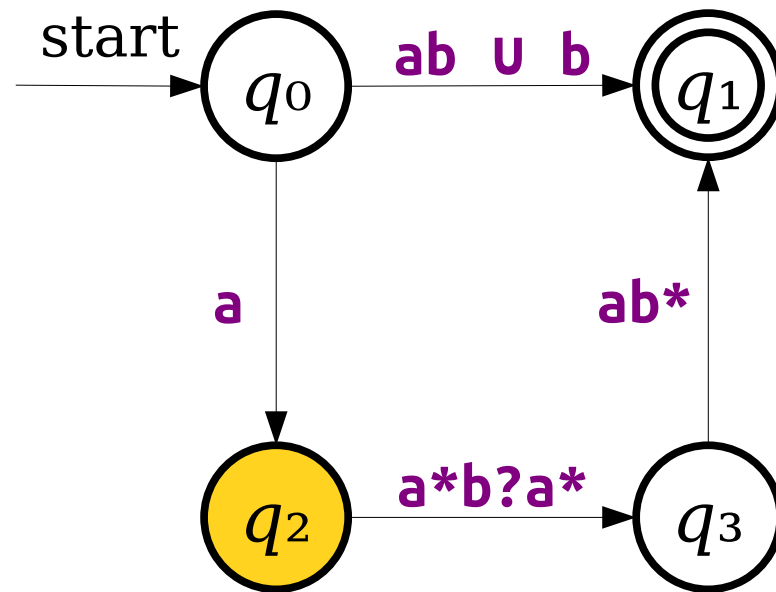
Generalizing NFAs



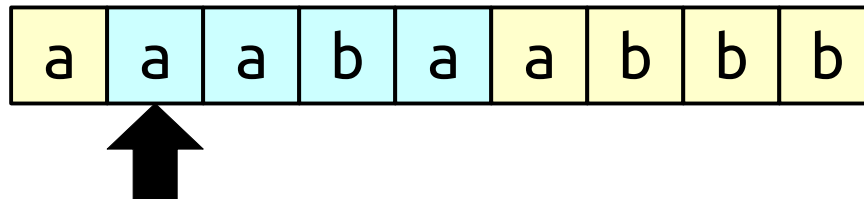
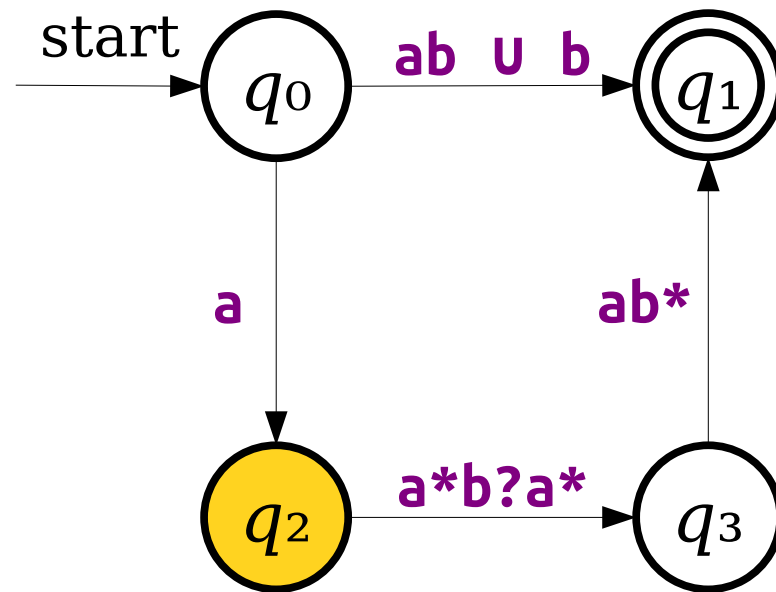
Generalizing NFAs



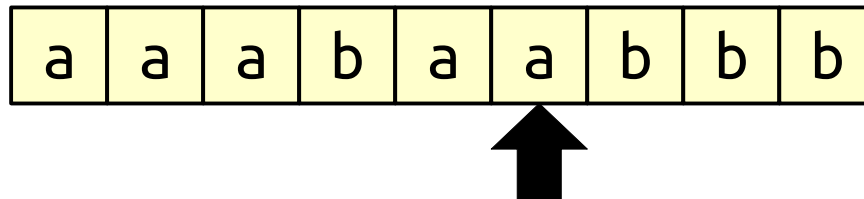
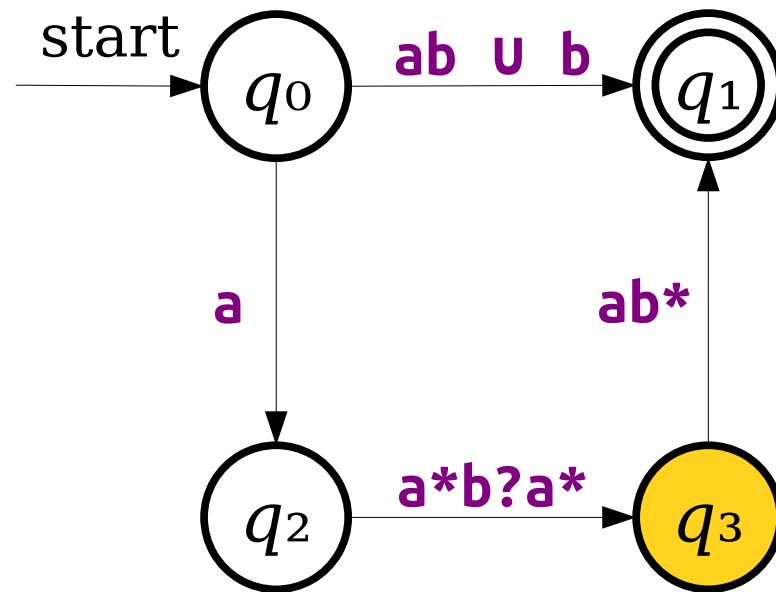
Generalizing NFAs



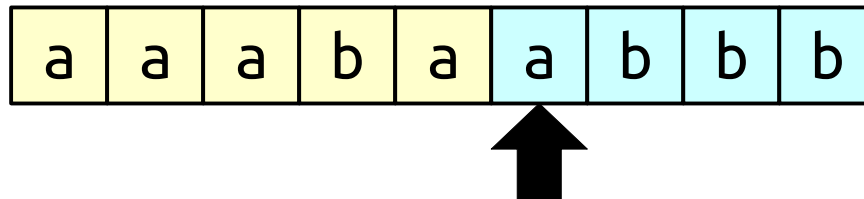
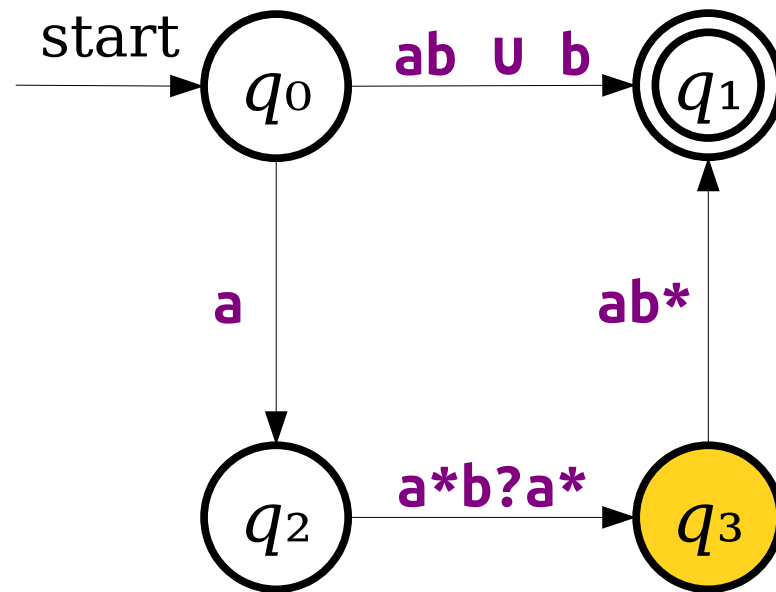
Generalizing NFAs



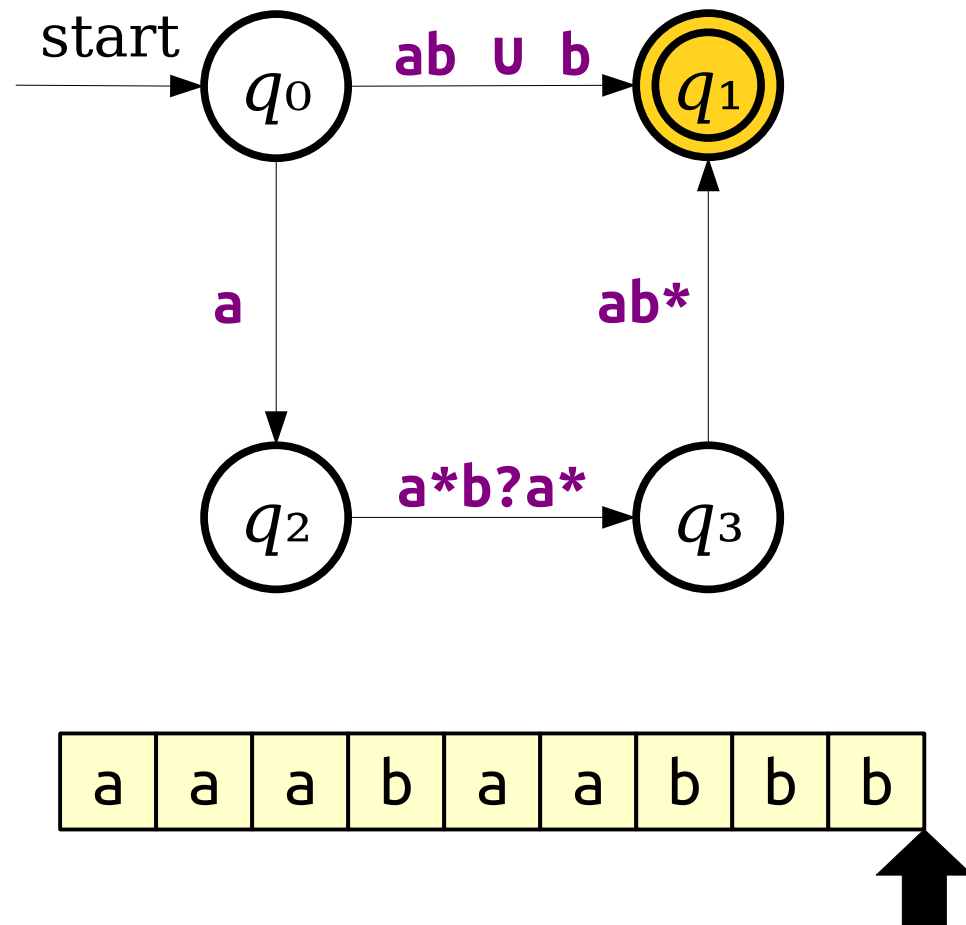
Generalizing NFAs



Generalizing NFAs



Generalizing NFAs



Key Idea 1: Imagine that we can label transitions in an NFA with arbitrary regular expressions.

Generalizing NFAs

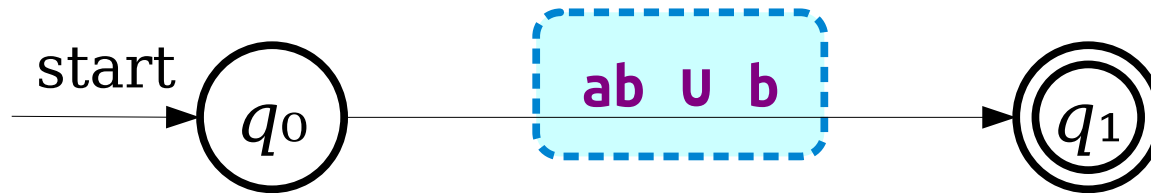


Generalizing NFAs



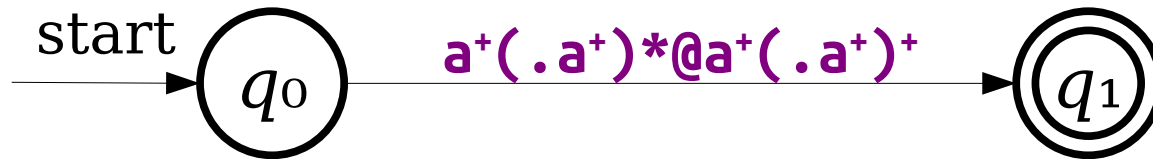
Is there a simple regular expression for the language of this generalized NFA?

Generalizing NFAs

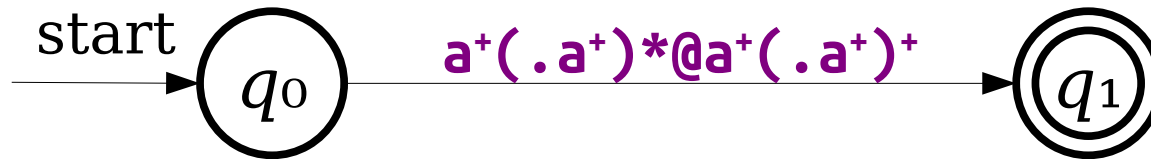


Is there a simple regular expression for the language of this generalized NFA?

Generalizing NFAs

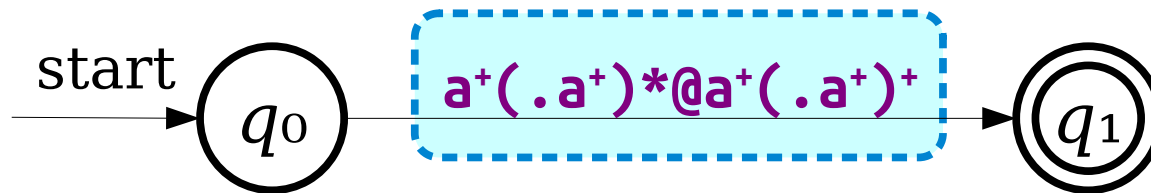


Generalizing NFAs



Is there a simple regular expression for the language of this generalized NFA?

Generalizing NFAs



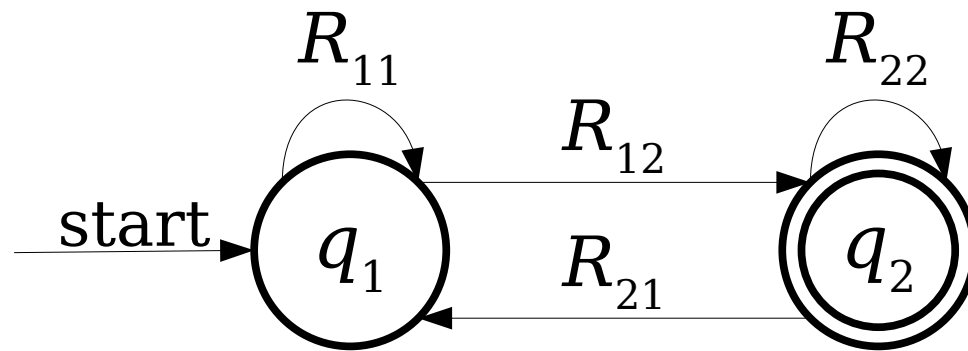
Is there a simple regular expression for the language of this generalized NFA?

Key Idea 2: If we can convert an NFA into a generalized NFA that looks like this...

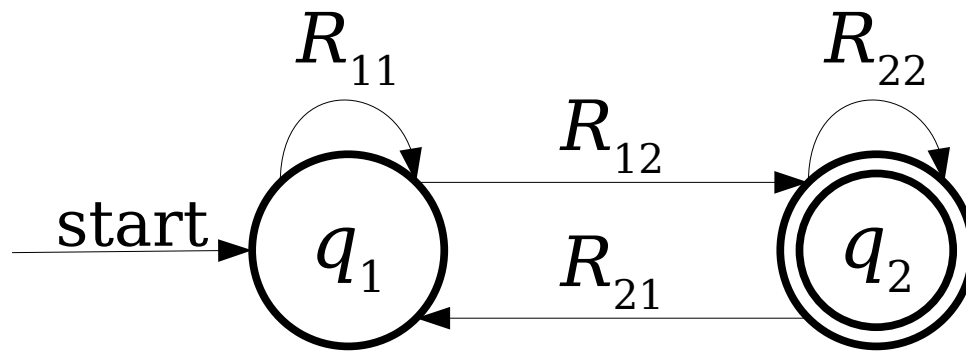


...then we can easily read off a regular expression for the original NFA.

From NFAs to Regular Expressions

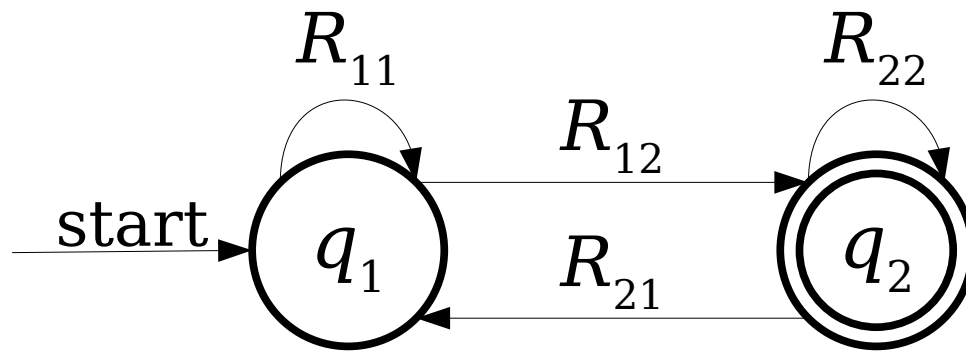


From NFAs to Regular Expressions



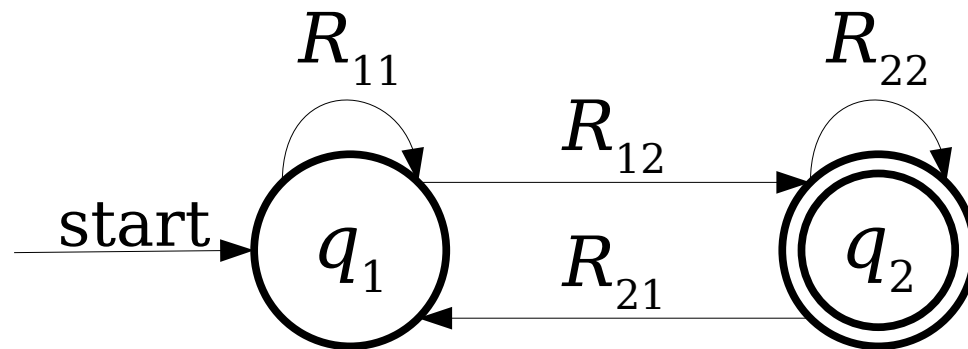
Here, R_{11} , R_{12} , R_{21} , and R_{22} are arbitrary regular expressions.

From NFAs to Regular Expressions

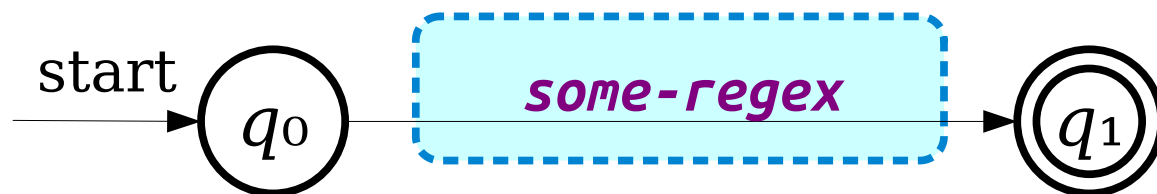


Question: Can we get a clean regular expression from this NFA?

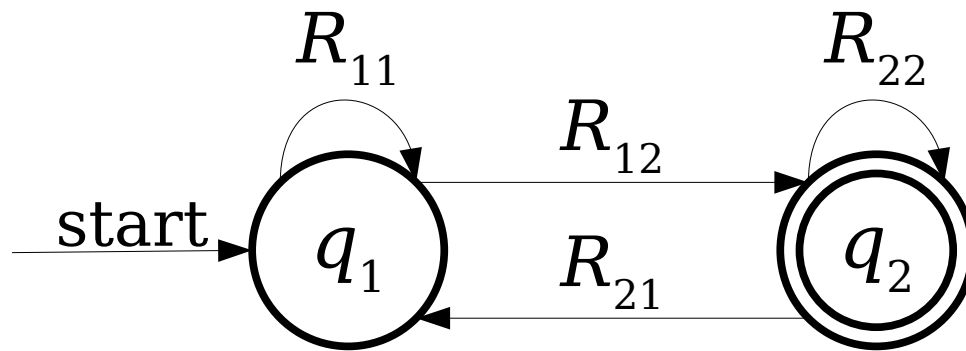
From NFAs to Regular Expressions



Key Idea 3: Somehow transform this NFA so that it looks like this:

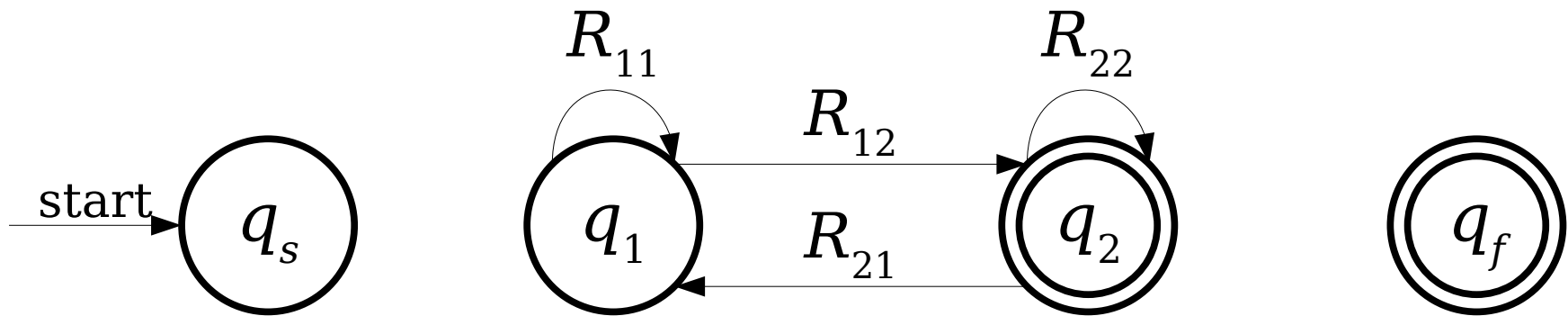


From NFAs to Regular Expressions

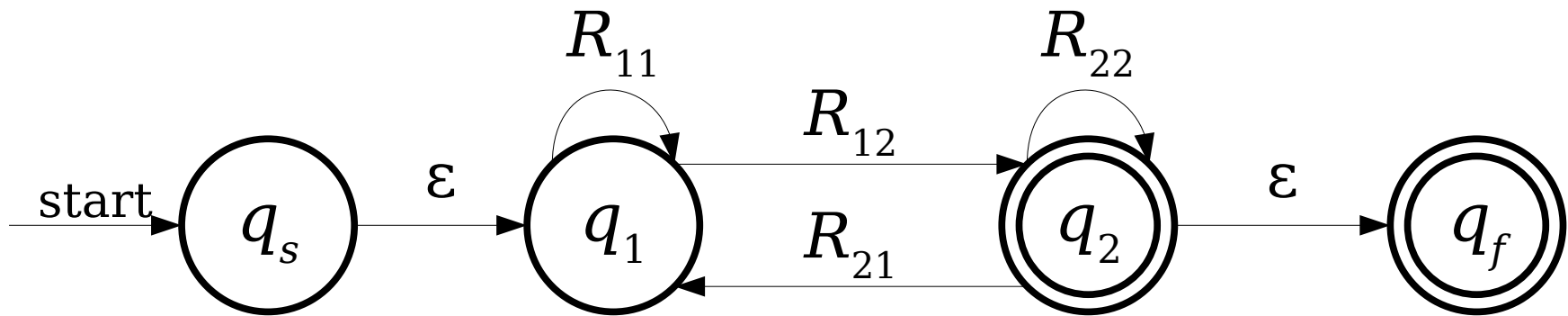


The first step is going to be a
bit weird...

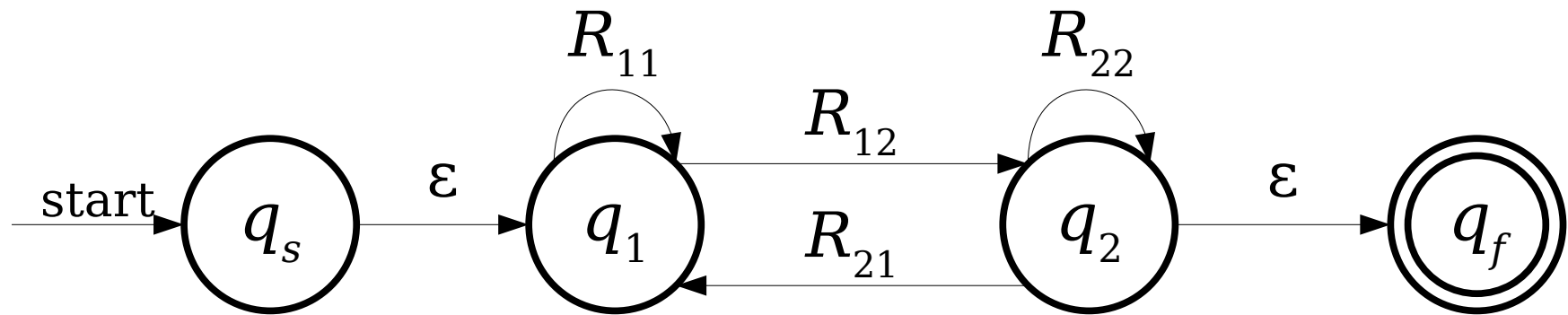
From NFAs to Regular Expressions



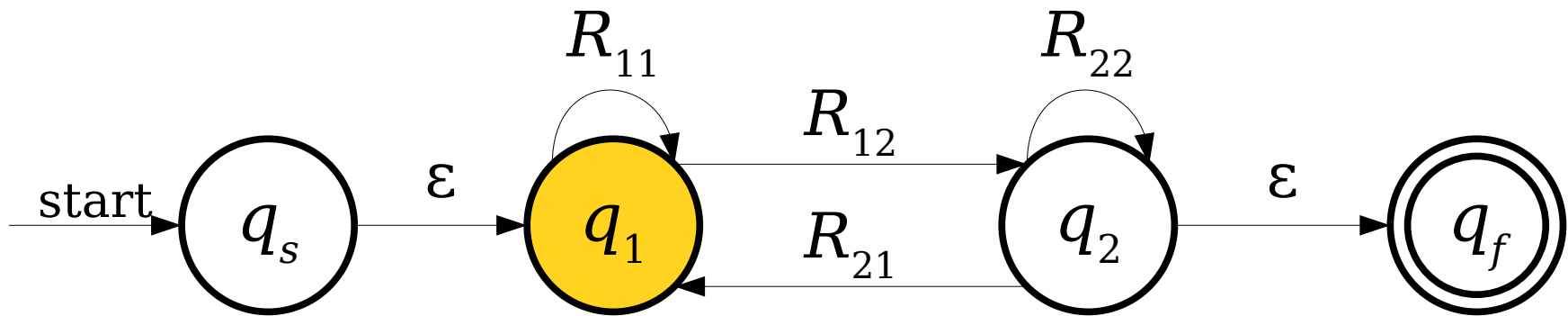
From NFAs to Regular Expressions



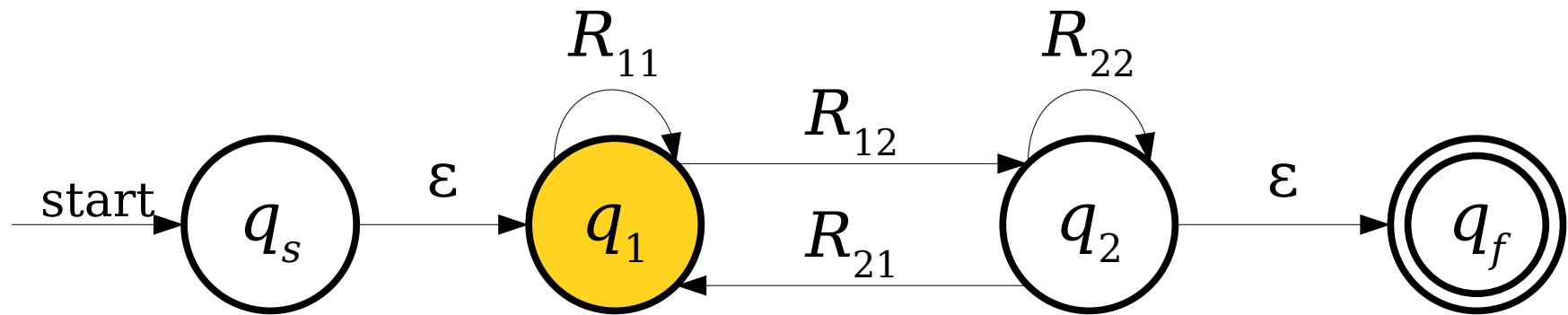
From NFAs to Regular Expressions



From NFAs to Regular Expressions

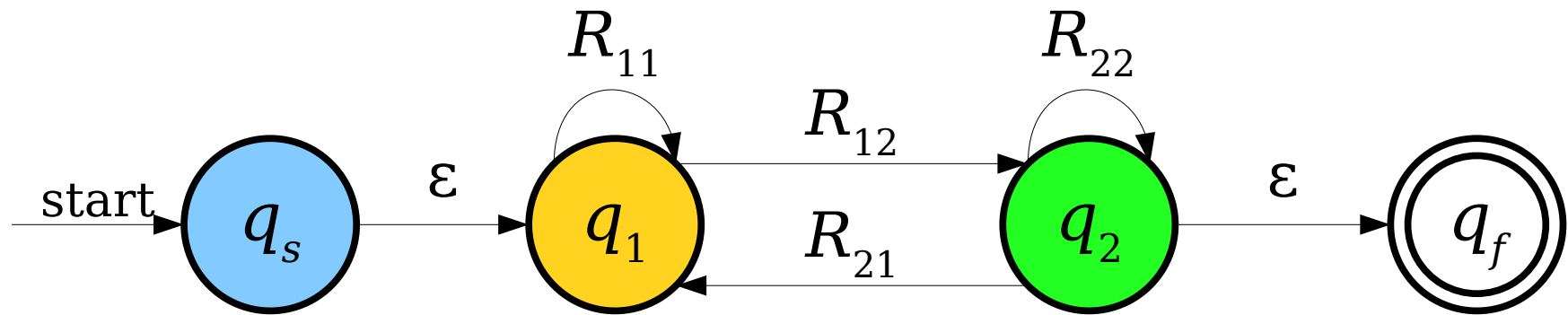


From NFAs to Regular Expressions

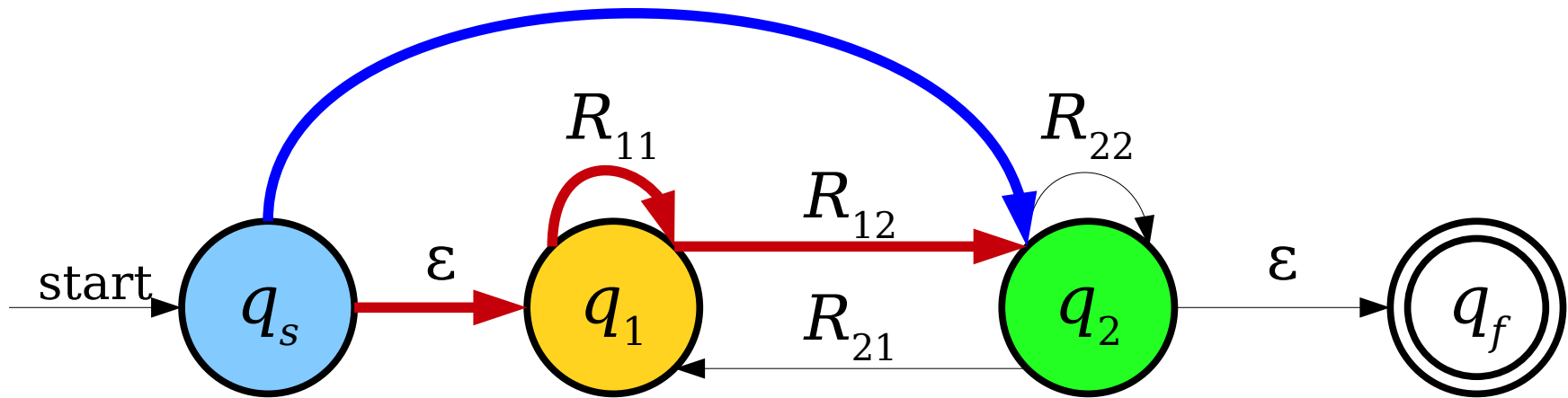


Could we eliminate this state from the NFA?

From NFAs to Regular Expressions

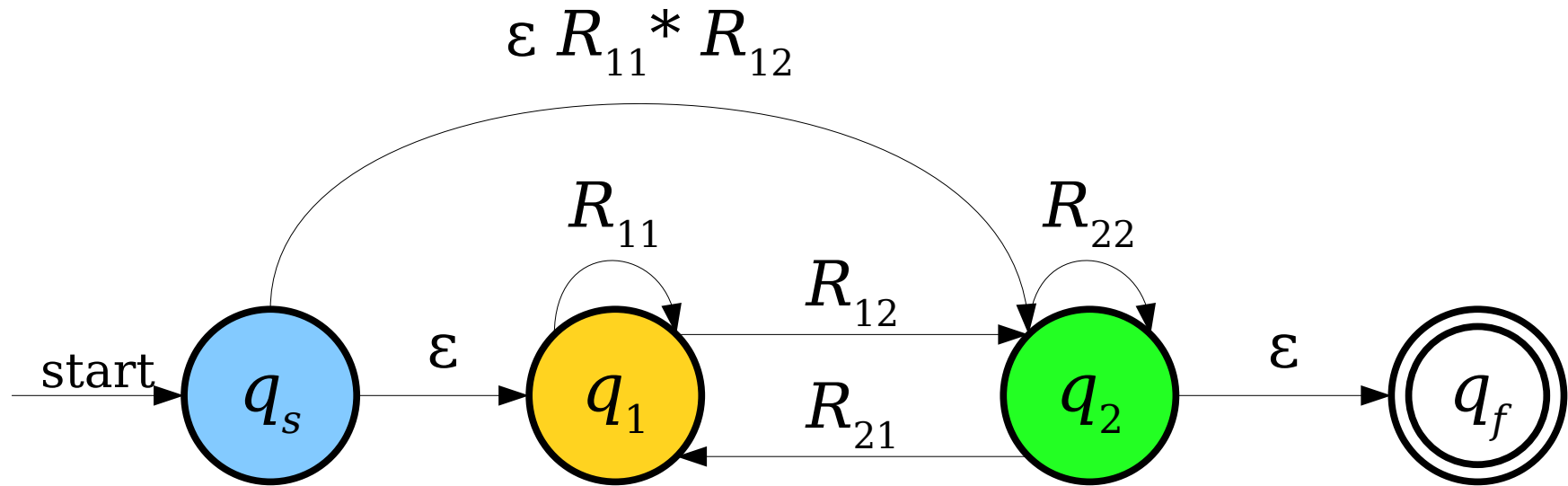


From NFAs to Regular Expressions



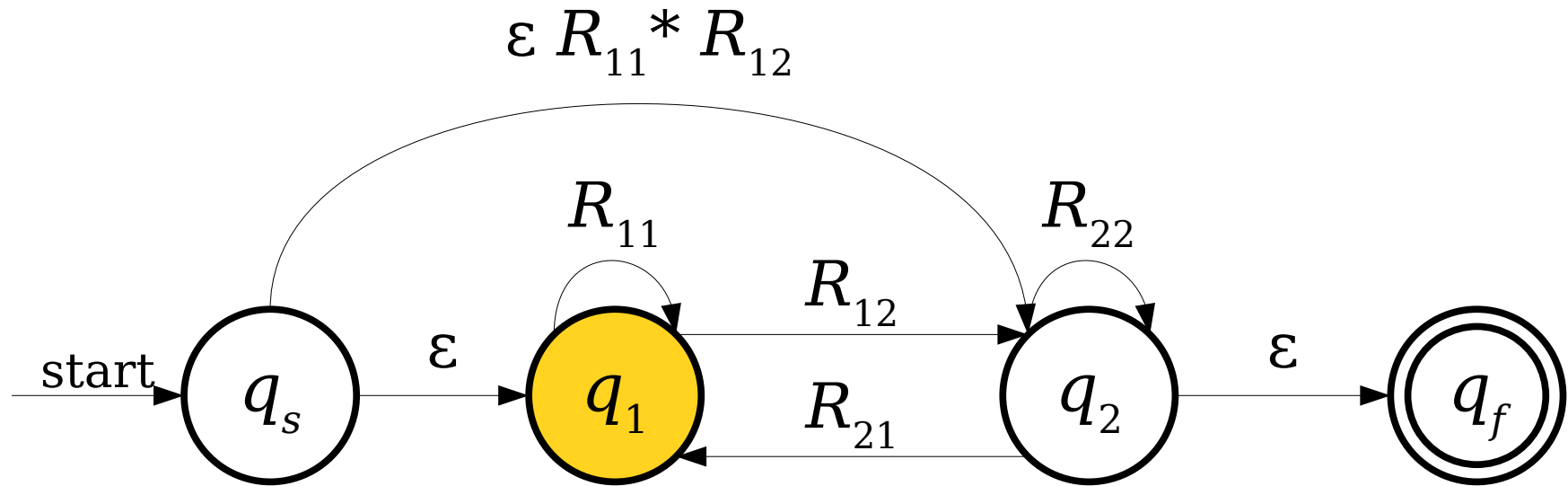
I'd like to replace the red transitions with this new blue transition that skips q_1 .

From NFAs to Regular Expressions

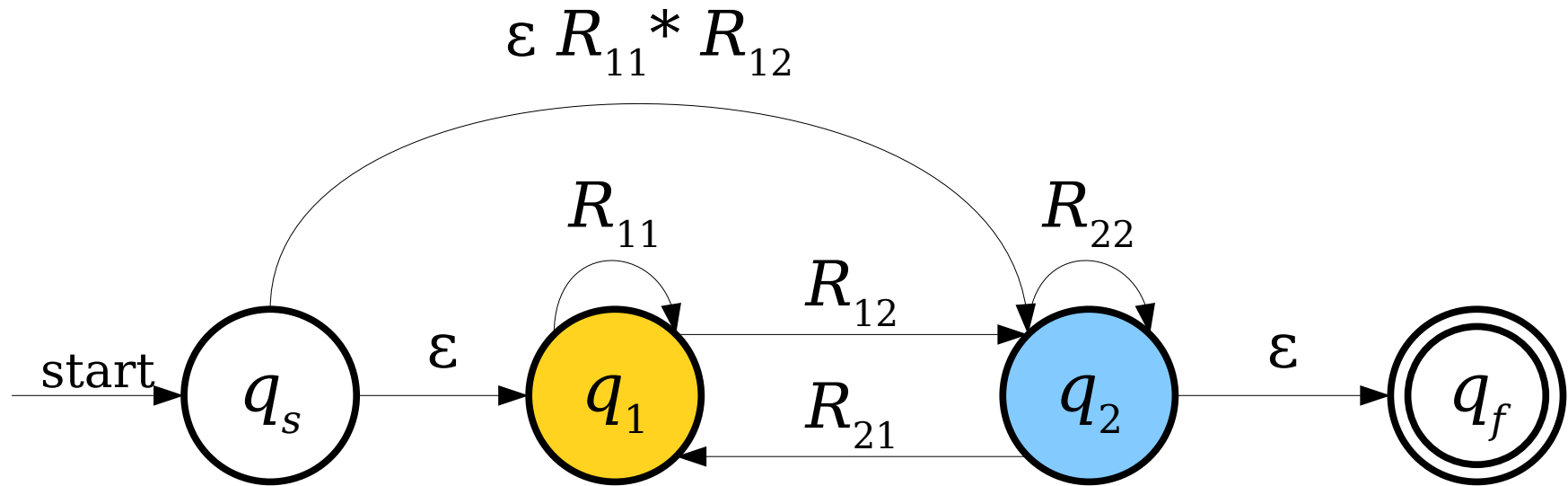


Note: We're using **concatenation** and **Kleene closure** in order to skip this state.

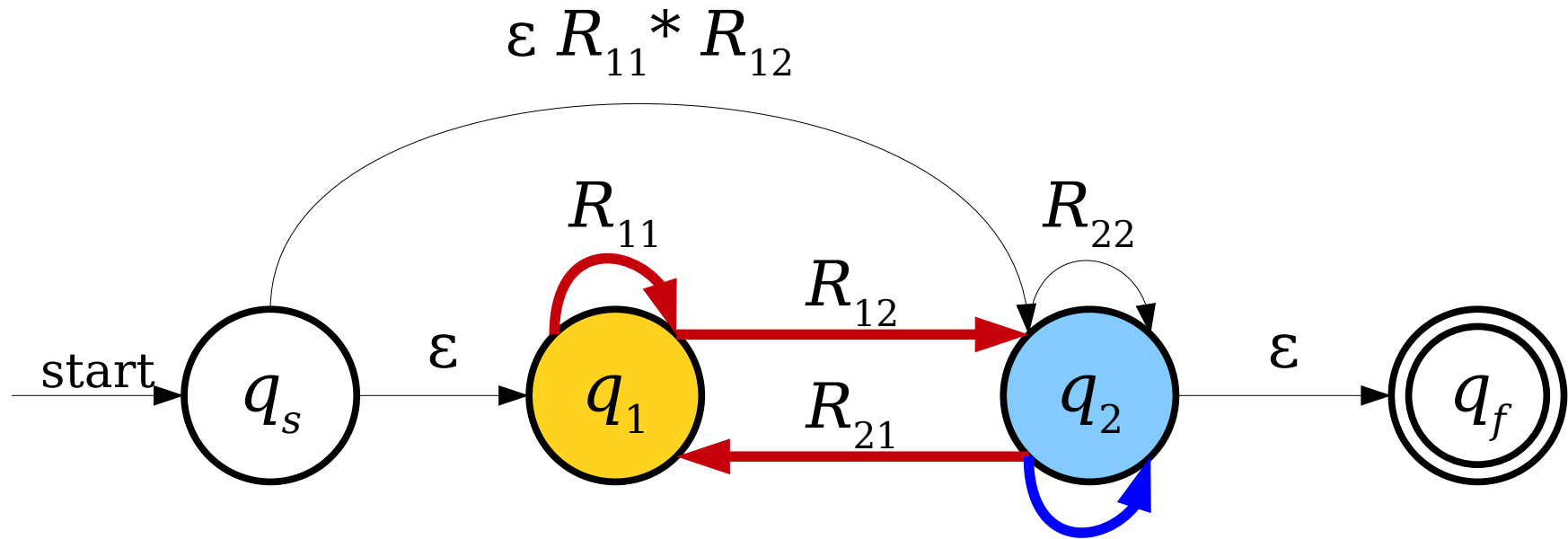
From NFAs to Regular Expressions



From NFAs to Regular Expressions

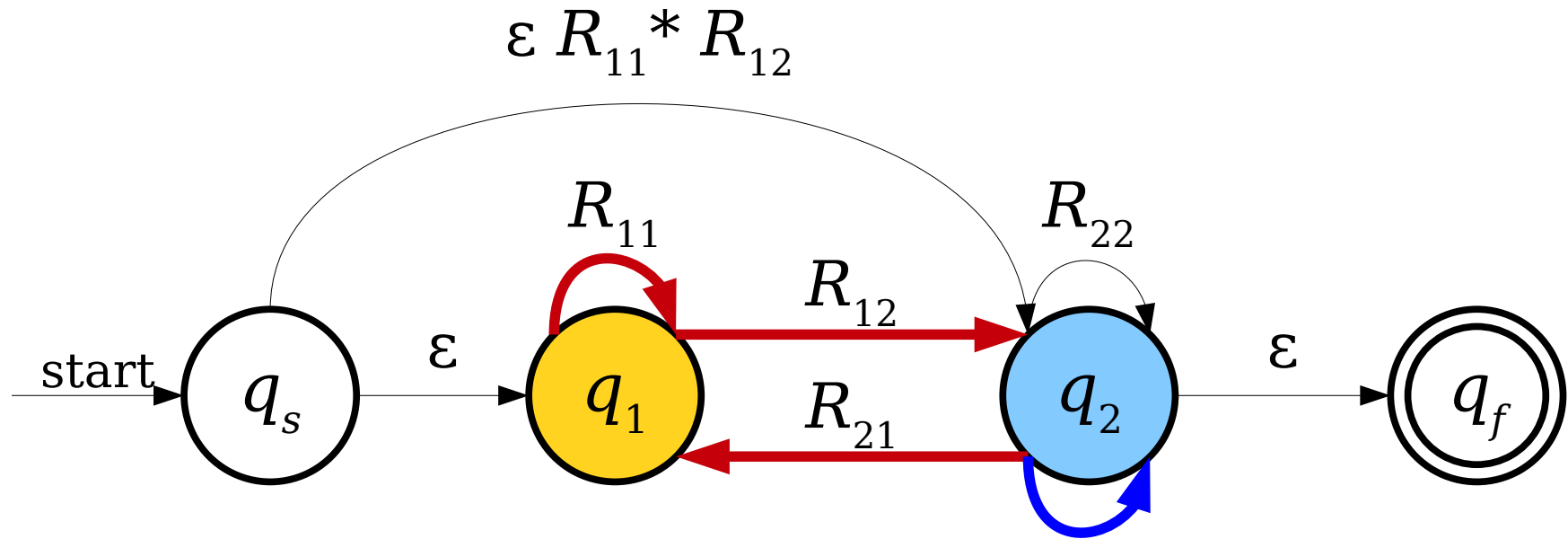


From NFAs to Regular Expressions



Again, I want to replace these red transitions with this new blue transition that skips q_1 .

From NFAs to Regular Expressions



What regex should go on this edge?

A) $R_{12}R_{21}$

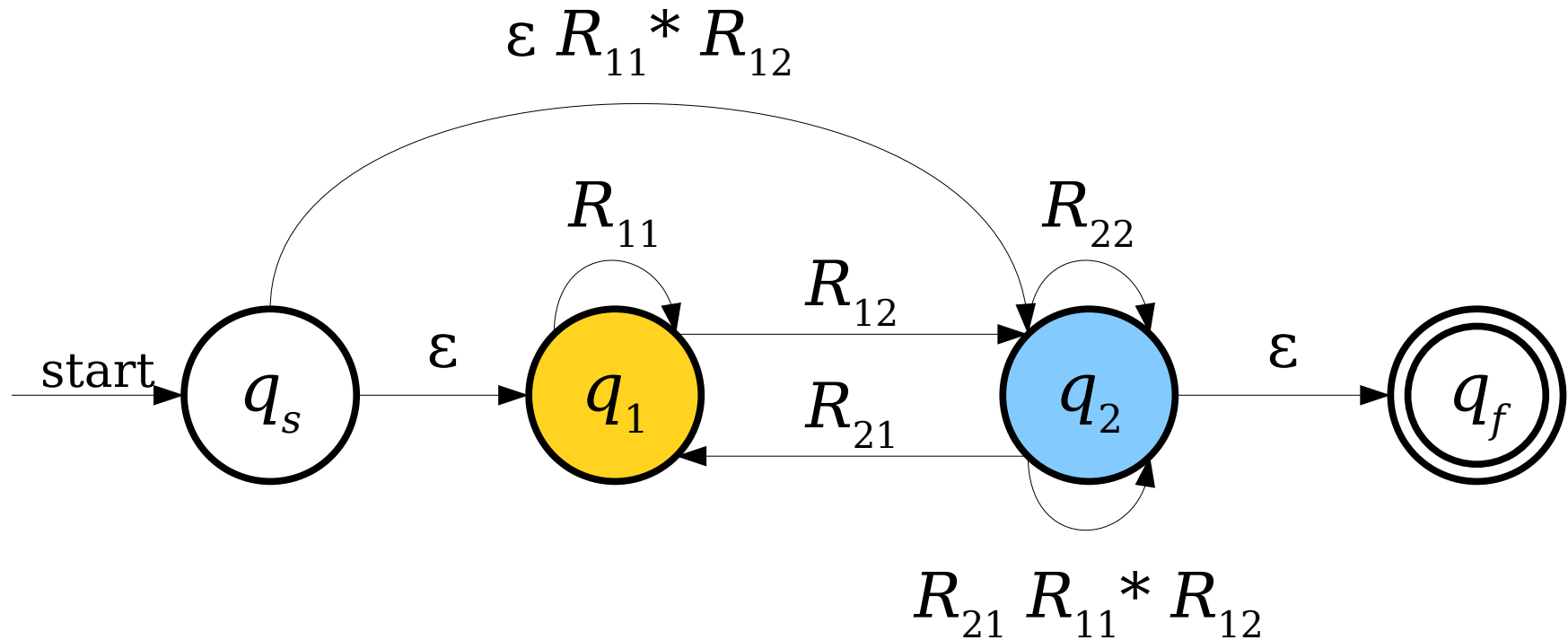
C) $R_{21}R_{12}$

B) $R_{12}R_{11}^*R_{21}$

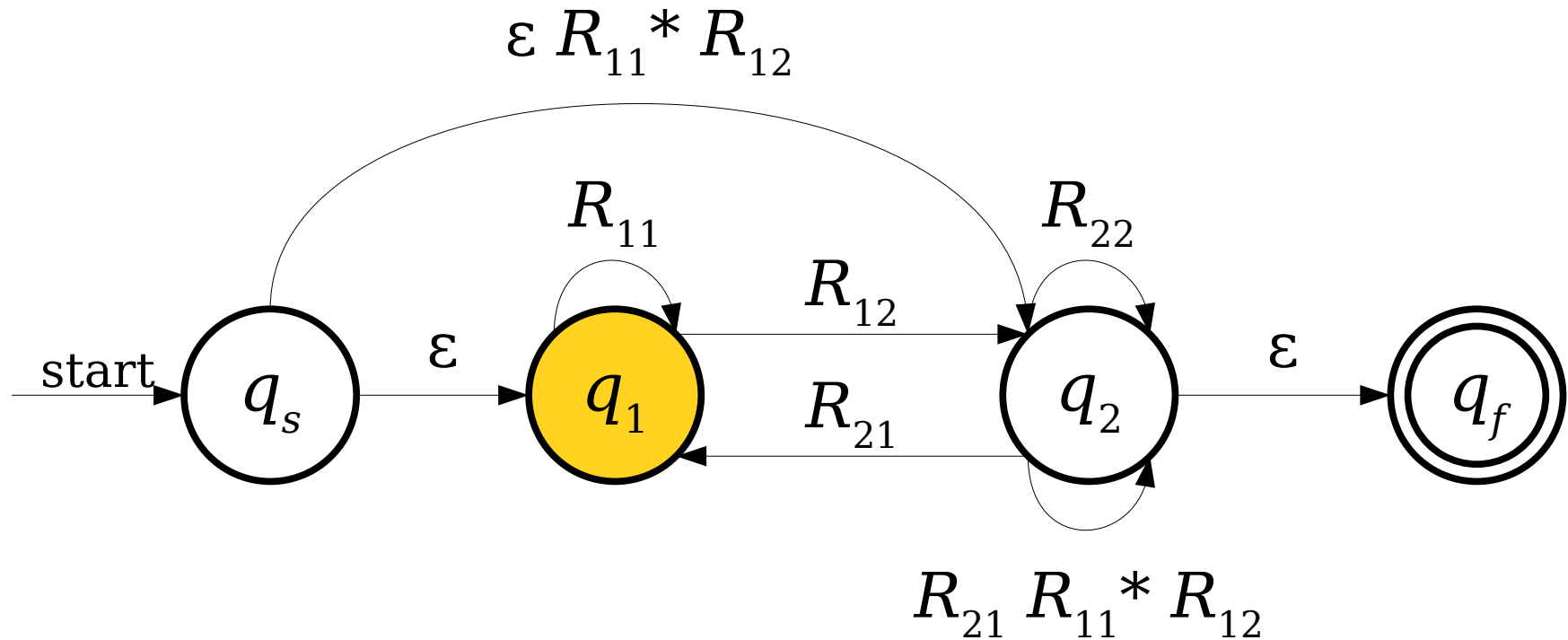
D) $R_{21}R_{11}^*R_{12}$

Respond at pollev.com/zhenglian

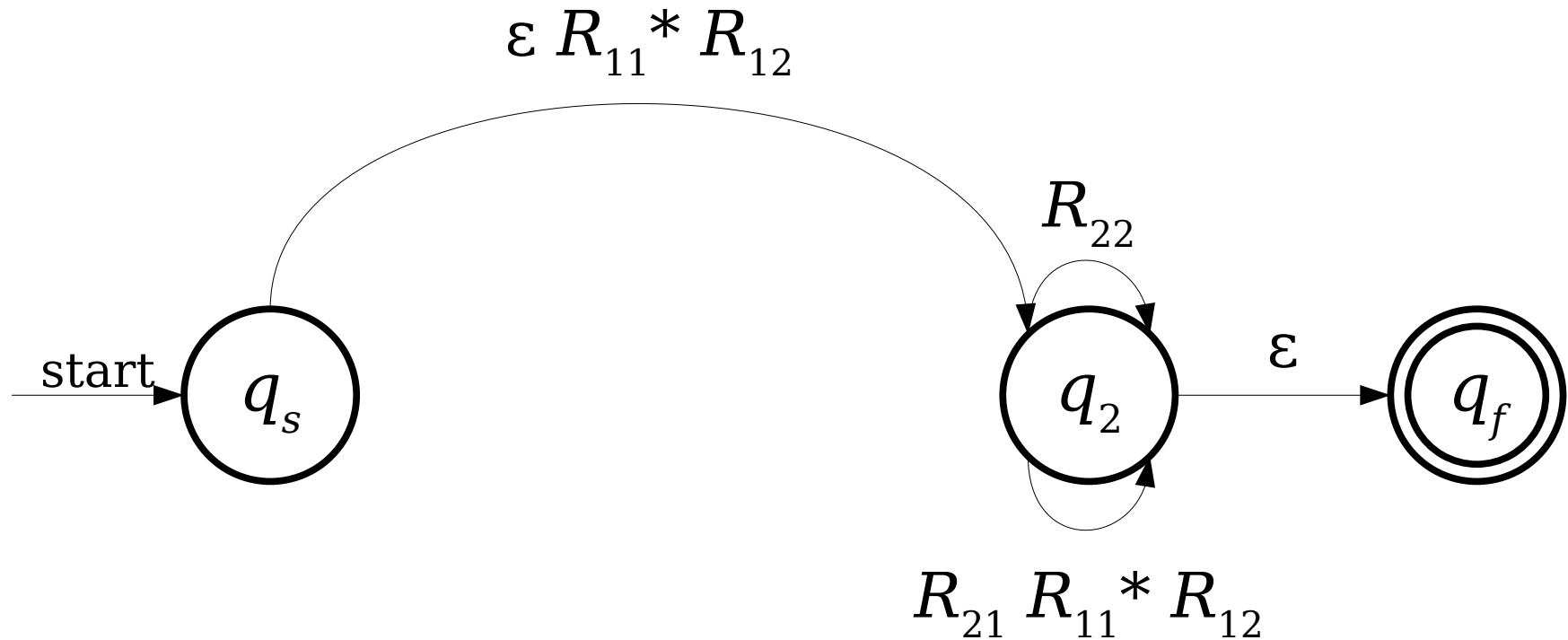
From NFAs to Regular Expressions



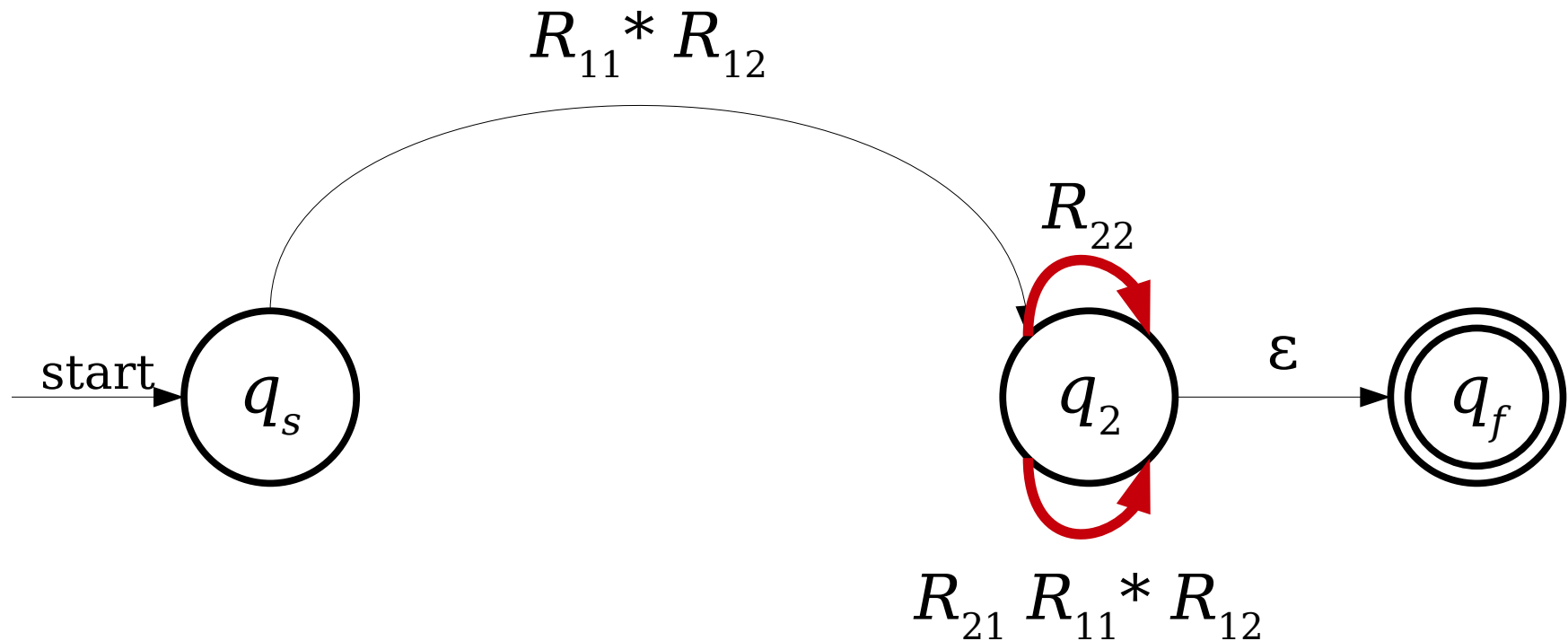
From NFAs to Regular Expressions



From NFAs to Regular Expressions

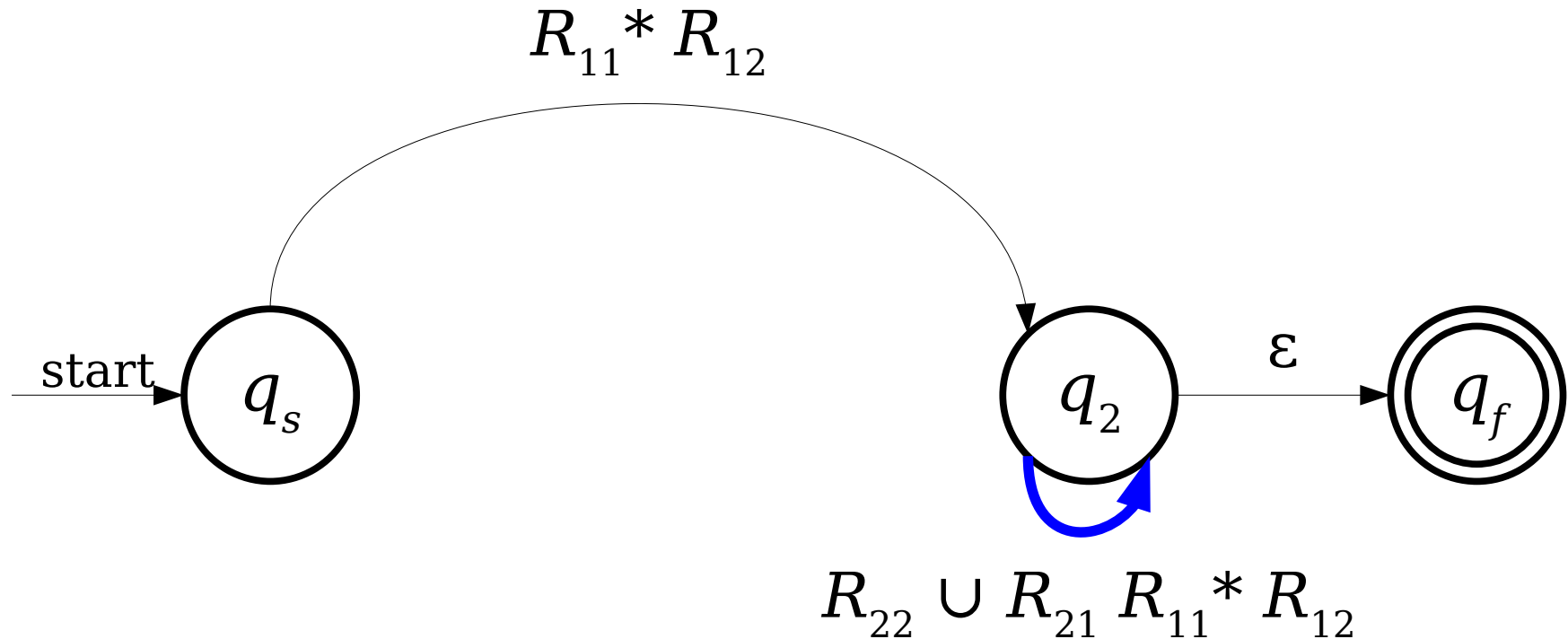


From NFAs to Regular Expressions



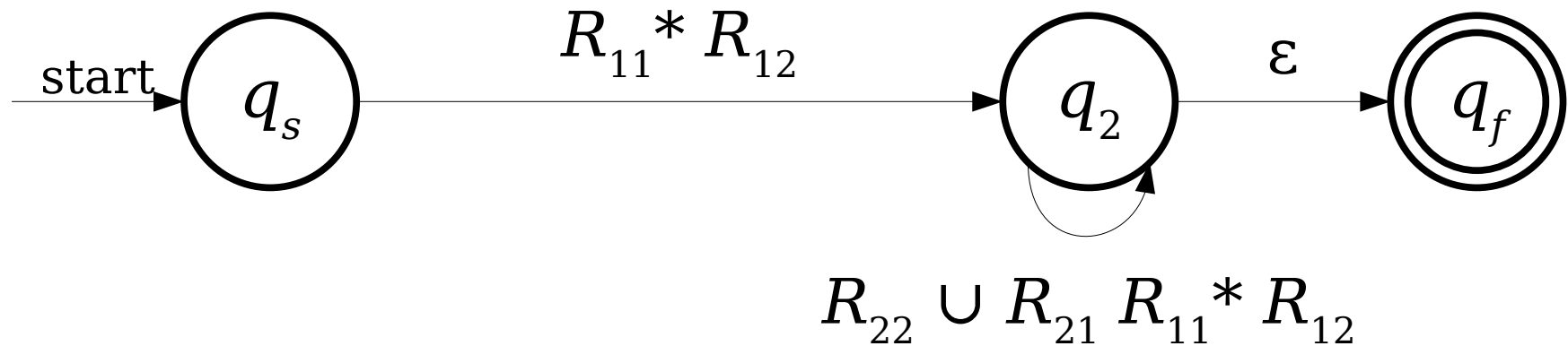
Can I combine these two red transitions into a single one?

From NFAs to Regular Expressions

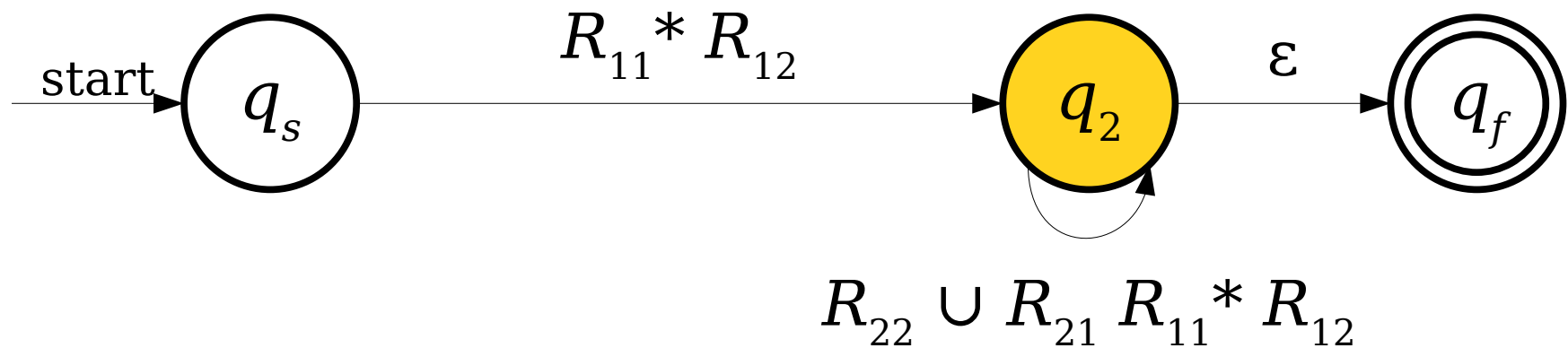


Note: We're using **union** to combine these transitions together.

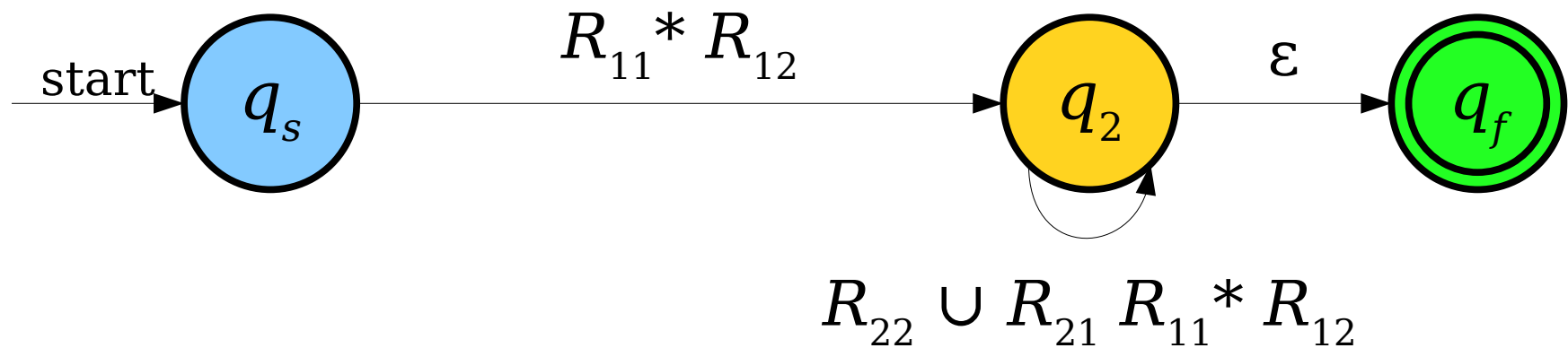
From NFAs to Regular Expressions



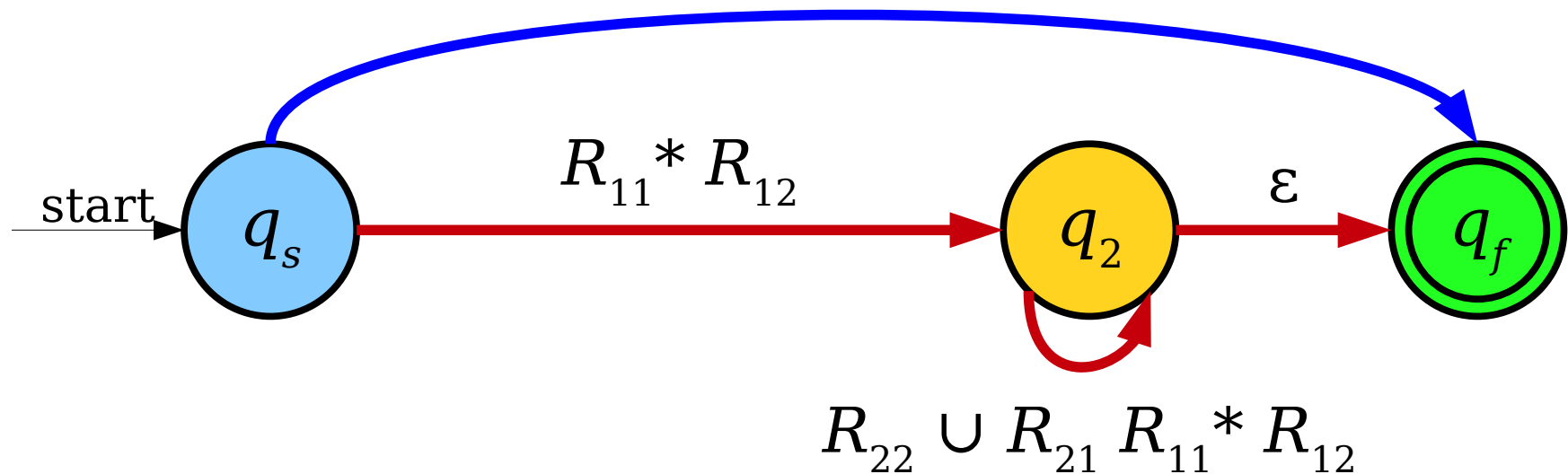
From NFAs to Regular Expressions



From NFAs to Regular Expressions



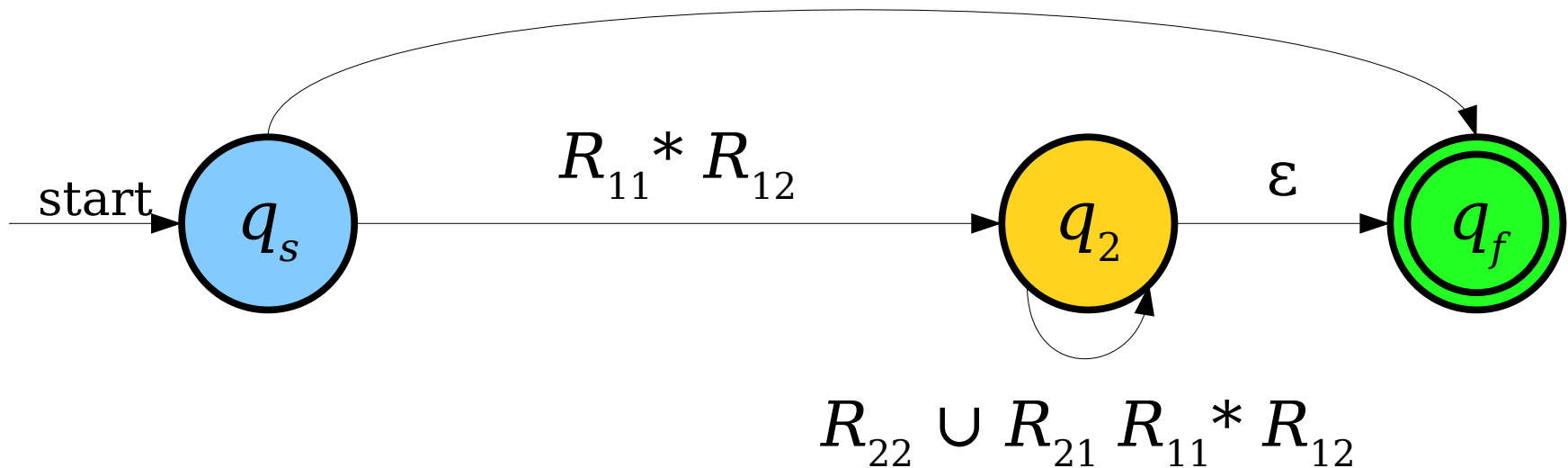
From NFAs to Regular Expressions



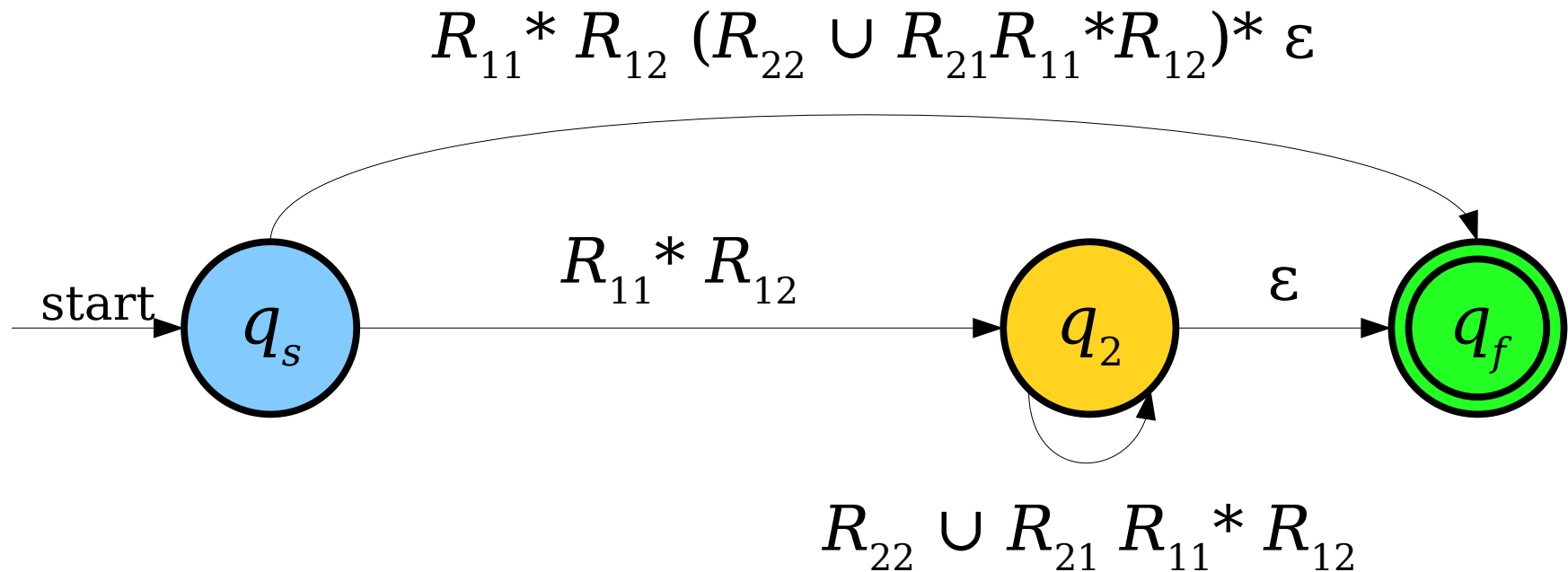
One last time: I want to replace these red transitions with this new blue transition that skips q_2 .

From NFAs to Regular Expressions

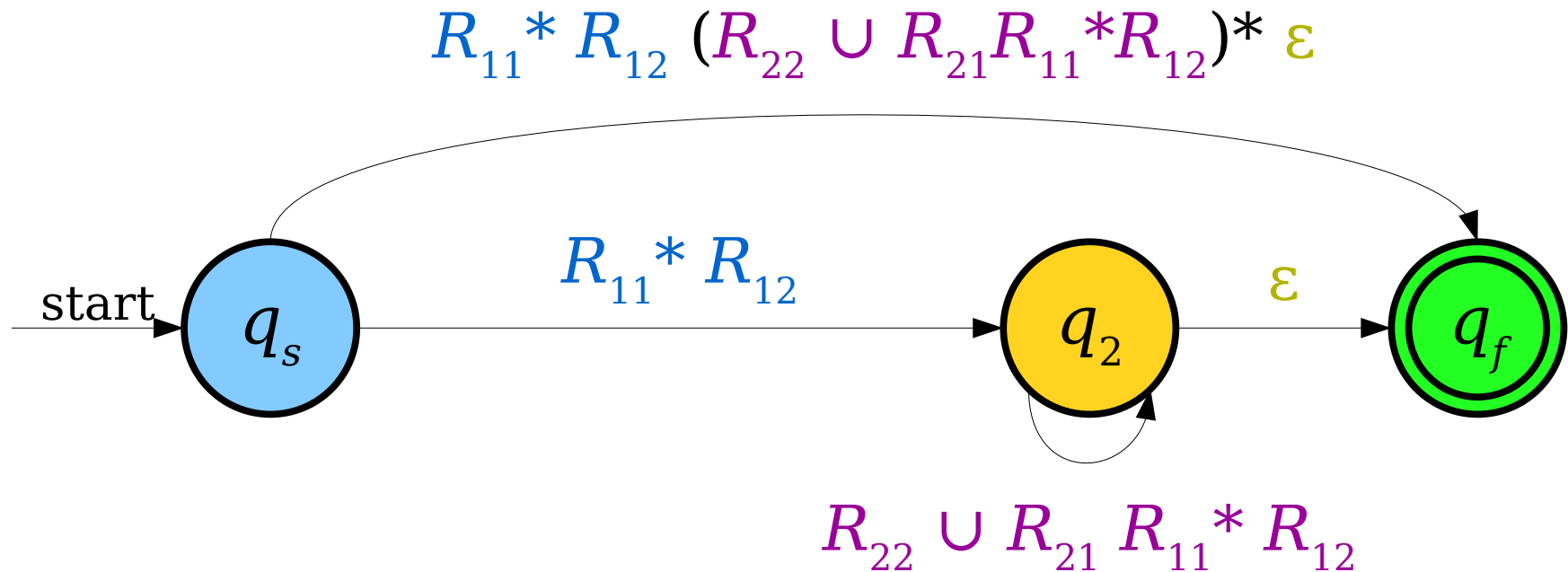
What should we put on this transition?



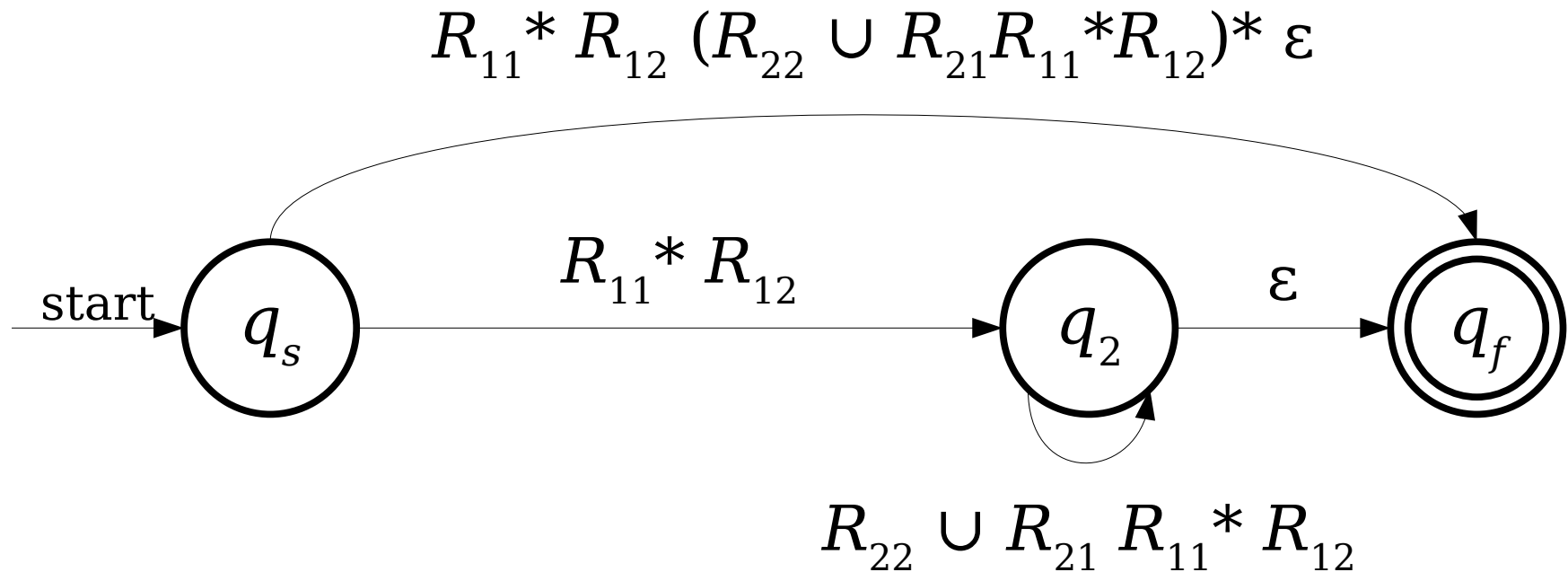
From NFAs to Regular Expressions



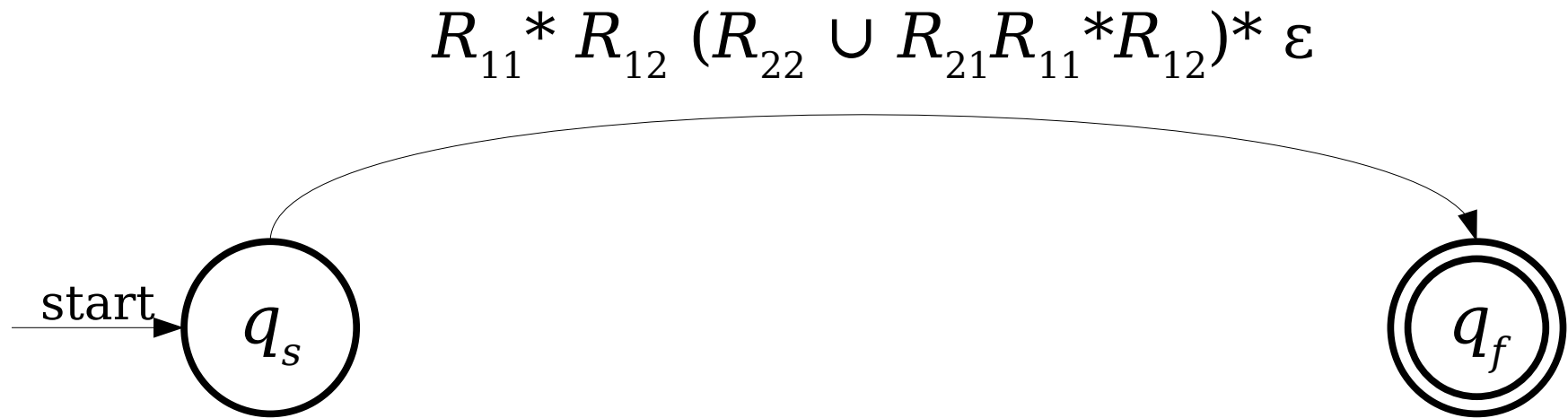
From NFAs to Regular Expressions



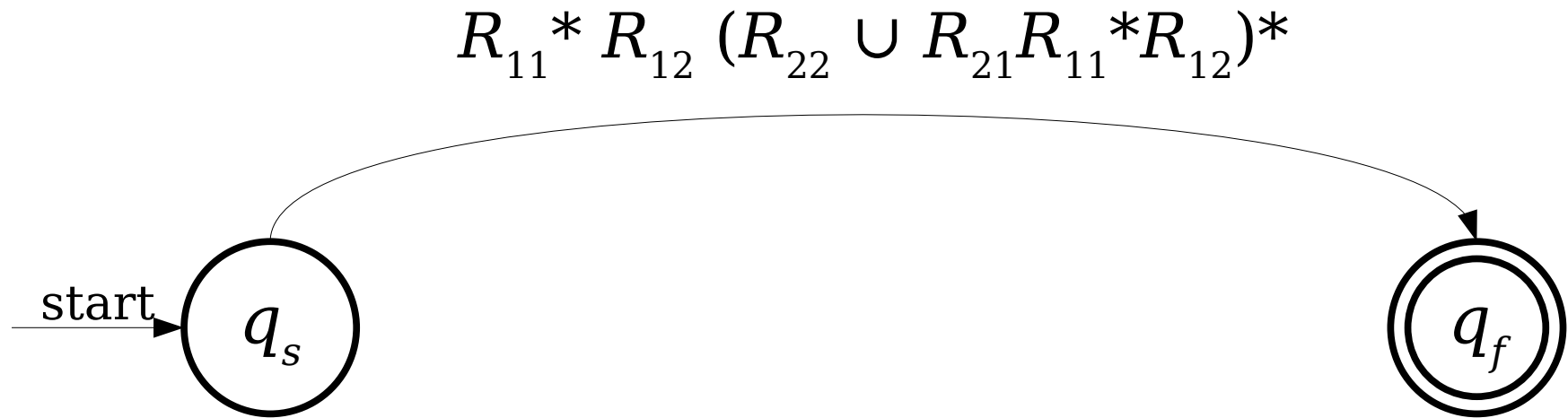
From NFAs to Regular Expressions



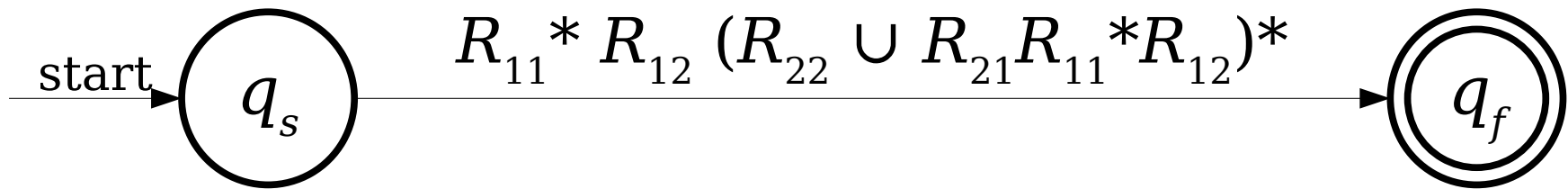
From NFAs to Regular Expressions



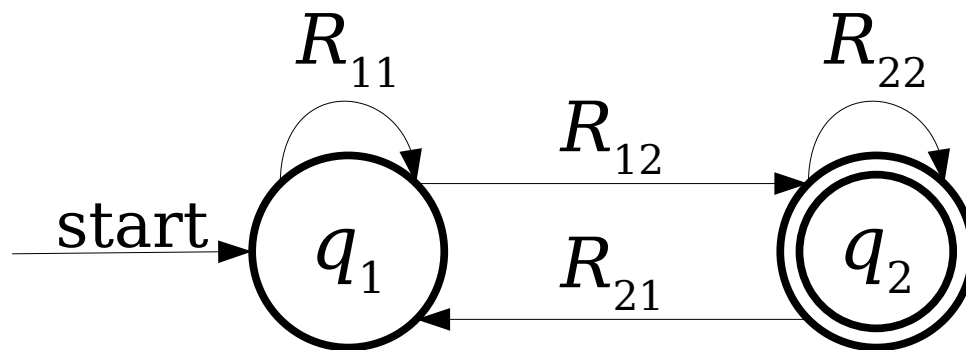
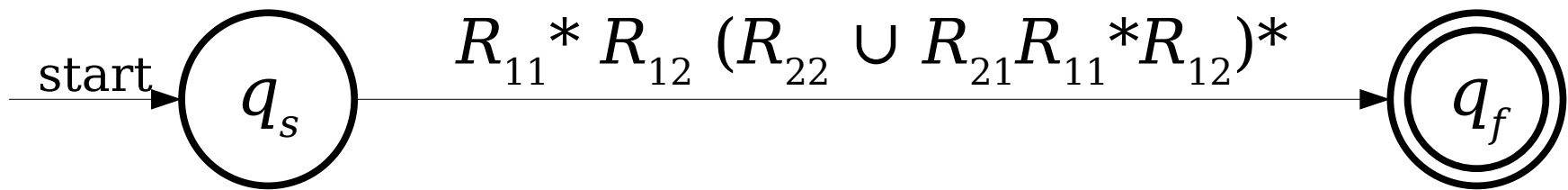
From NFAs to Regular Expressions



From NFAs to Regular Expressions



From NFAs to Regular Expressions



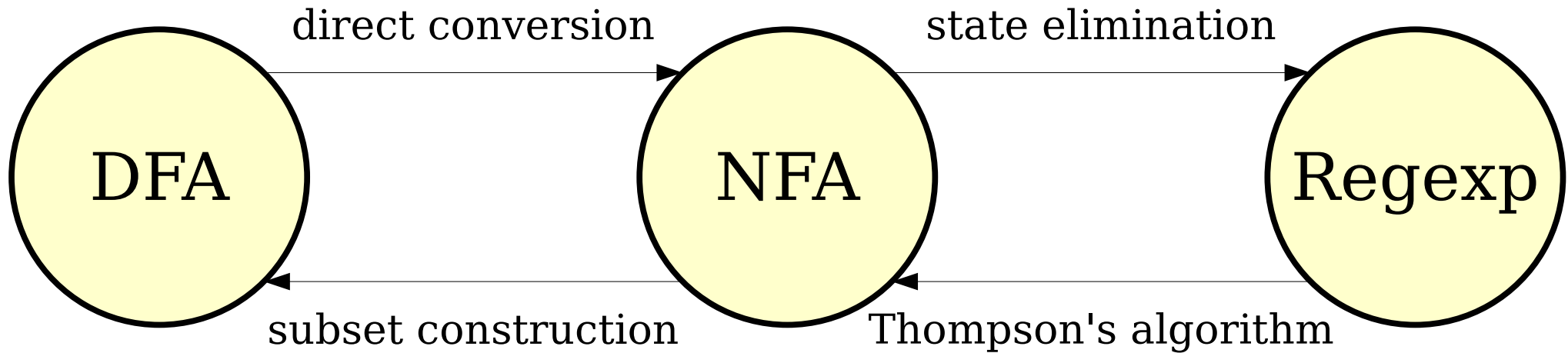
The State-Elimination Algorithm

- Start with an NFA N for the language L .
- Add a new start state q_s and accept state q_f to the NFA.
 - Add an ε -transition from q_s to the old start state of N .
 - Add ε -transitions from each accepting state of N to q_f , then mark them as not accepting.
- Repeatedly remove states other than q_s and q_f from the NFA by “shortcutting” them until only two states remain: q_s and q_f .
- The transition from q_s to q_f is then a regular expression for the NFA.

The State-Elimination Algorithm

- To eliminate a state q from the automaton, do the following for each pair of states q_0 and q_1 , where there's a transition from q_0 into q and a transition from q into q_1 :
 - Let R_{in} be the regex on the transition from q_0 to q .
 - Let R_{out} be the regex on the transition from q to q_1 .
 - If there is a regular expression R_{stay} on a transition from q to itself, add a new transition from q_0 to q_1 labeled $((R_{in})(R_{stay})^*(R_{out}))$.
 - If there isn't, add a new transition from q_0 to q_1 labeled $((R_{in})(R_{out}))$
- If a pair of states has multiple transitions between them labeled $R_1, R_2, \dots, R_k$, replace them with a single transition labeled $R_1 \cup R_2 \cup \dots \cup R_k$.

Our Transformations



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 $\mathcal{L}(R) = L$.

Why This Matters

- The equivalence of regular expressions and finite automata has practical relevance.
 - Regular expression matchers have all the power available to them of DFAs and NFAs.
- This also is hugely theoretically significant: the regular languages can be assembled “from scratch” using a small number of operations!

Automata Design Workshop

Part 1: *Designing DFAs*



OREO



O&REO



O&O



OREOREO



RERERERERE



OOOOO



OREOO



OREOREREREORE



OREOREORE

REREO



REORE



OREREREREREREREREORE



OOOREREREREREREOREOO



OREREREREOREOOOOOOOOO

Oreo Sandwiches

- Let $\Sigma = \{ \text{O}, \text{R} \}$

For simplicity, let's just use a single character for the
“cream” part of the Oreo :)

Oreo Sandwiches

- Let $\Sigma = \{ \mathbf{O}, \mathbf{R} \}$

Design a DFA for the language

$$L = \{ w \in \Sigma^* \mid w \neq \varepsilon \text{ and the first and last character of } w \text{ are the same} \}$$

Oreo Sandwiches

- Let $\Sigma = \{ \mathbf{O}, \mathbf{R} \}$

Design a DFA for the language

$$L = \{ w \in \Sigma^* \mid w \neq \varepsilon \text{ and the first and last character of } w \text{ are the same} \}$$

$$\mathbf{ORO} \in L$$

$$\mathbf{OR} \notin L$$

$$\mathbf{R00OR} \in L$$

$$\mathbf{00000R} \notin L$$

$$\mathbf{OR0ORORRO} \in L$$

$$\mathbf{RORORORO} \notin L$$

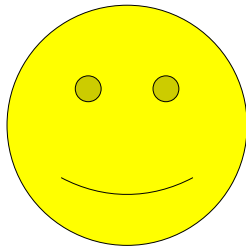
Designing DFAs

- ***States*** – pieces of information
 - What do I have to keep track of in the course of figuring out whether a string is in this language?
- ***Transitions*** – updating state
 - From the state I'm currently in, what do I know about my string? How would reading this character change what I know?

An Analogy

Imagine a scenario where Bob is thinking of a string and Alice has to figure out whether that string is in a particular language

$L = \{ w \text{ is divisible by } 5 \}$



Alice

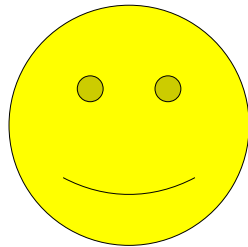


Bob

An Analogy

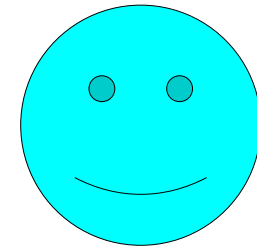
The catch: Bob can only send Alice one character at a time, and Alice doesn't know how long the string is until Bob tells her that he's done sending input

$L = \{ w \text{ is divisible by } 5 \}$



Alice

9

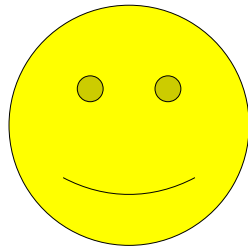


Bob

An Analogy

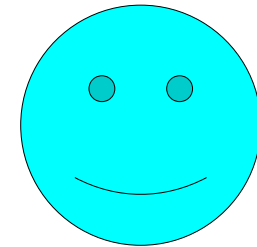
What does Alice need to remember about the characters she's receiving from Bob?

$L = \{ w \text{ is divisible by } 5 \}$



Alice

9

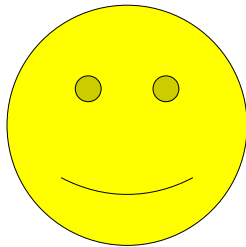


Bob

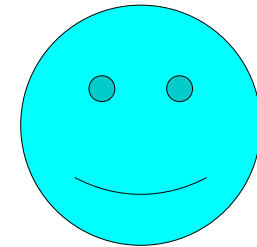
An Analogy

Key insight: Alice only needs to remember the last character she received from Bob

$L = \{ w \text{ is divisible by } 5 \}$



Alice

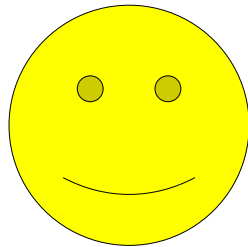


Bob

An Analogy

Key insight: Alice only needs to remember the last character she received from Bob

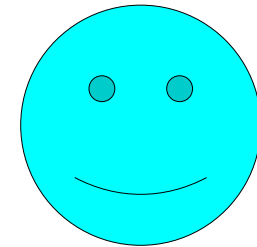
$L = \{ w \text{ is divisible by } 5 \}$



Alice



6

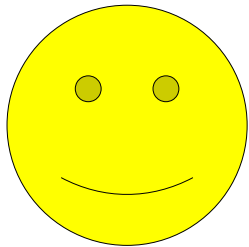


Bob

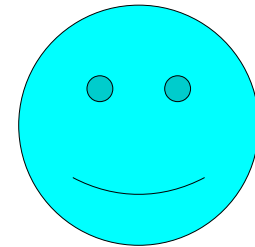
An Analogy

Key insight: Alice only needs to remember the last character she received from Bob

$L = \{ w \text{ is divisible by } 5 \}$



Alice

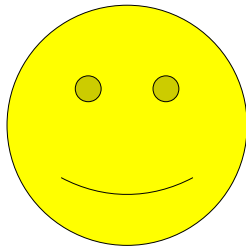


Bob

An Analogy

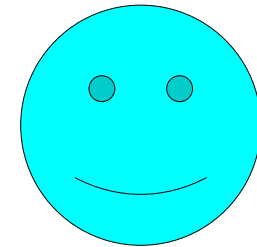
Key insight: Alice only needs to remember the last character she received from Bob

$L = \{ w \text{ is divisible by } 5 \}$



Alice

...

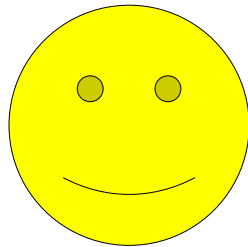


Bob

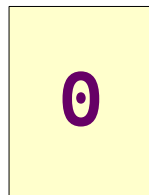
An Analogy

Eventually Bob gets to the end of his string and sends Alice a signal that he's done sending input

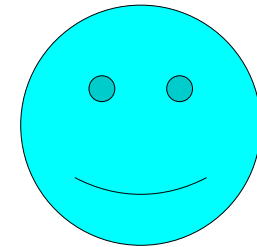
$L = \{ w \text{ is divisible by } 5 \}$



Alice



<end>

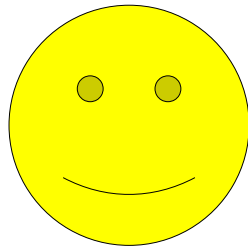


Bob

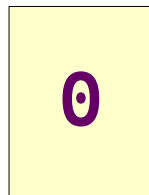
An Analogy

At this point, Alice just has to look at the last digit she wrote down and if it's a 5 or 0, Bob's string belongs in the language

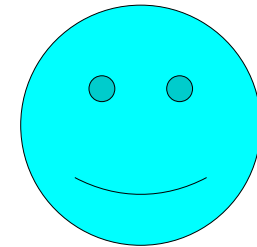
$L = \{ w \text{ is divisible by } 5 \}$



Alice



<end>



Bob

DFA Design Strategy

1. Answer the question “What do I have to keep track of in the course of figuring out whether a string is in this language?”
2. Create a state that represents each possible answer to that question.
3. From each state, go through all of the characters and answer the question “How would reading this character change what I know about my string?” and draw transitions to the appropriate states.

Oreo Sandwiches

- Let $\Sigma = \{ \text{O}, \text{R} \}$

Design a DFA for the language

$$L = \{ w \in \Sigma^* \mid w \neq \varepsilon \text{ and the first and last character of } w \text{ are the same} \}$$

What do I have to keep track of in the course of figuring out whether a string is in this language?

Oreo Sandwiches

$$L = \{ w \in \Sigma^* \mid w \neq \varepsilon \text{ and the first and last character of } w \text{ are the same} \}$$

- We need to keep track of the very first character
- And we need to keep track of the last character we've read so that when we reach the end, we can check whether the first and last characters were the same

Oreo Sandwiches

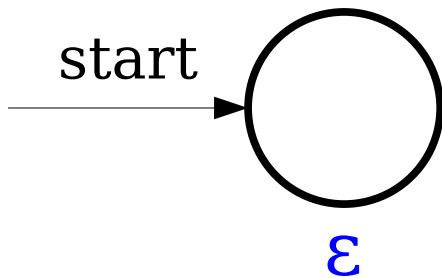
- Let $\Sigma = \{ \mathbf{O}, \mathbf{R} \}$

Design a DFA for the language

$$L = \{ w \in \Sigma^* \mid w \neq \varepsilon \text{ and the first and last character of } w \text{ are the same} \}$$

Oreo Sandwiches

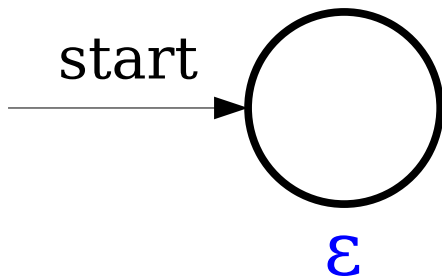
$$L = \{ w \in \Sigma^* \mid w \neq \varepsilon \text{ and the first and last character of } w \text{ are the same} \}$$



Remember that each state should represent a piece of information. We'll annotate what each state represents in blue.

Oreo Sandwiches

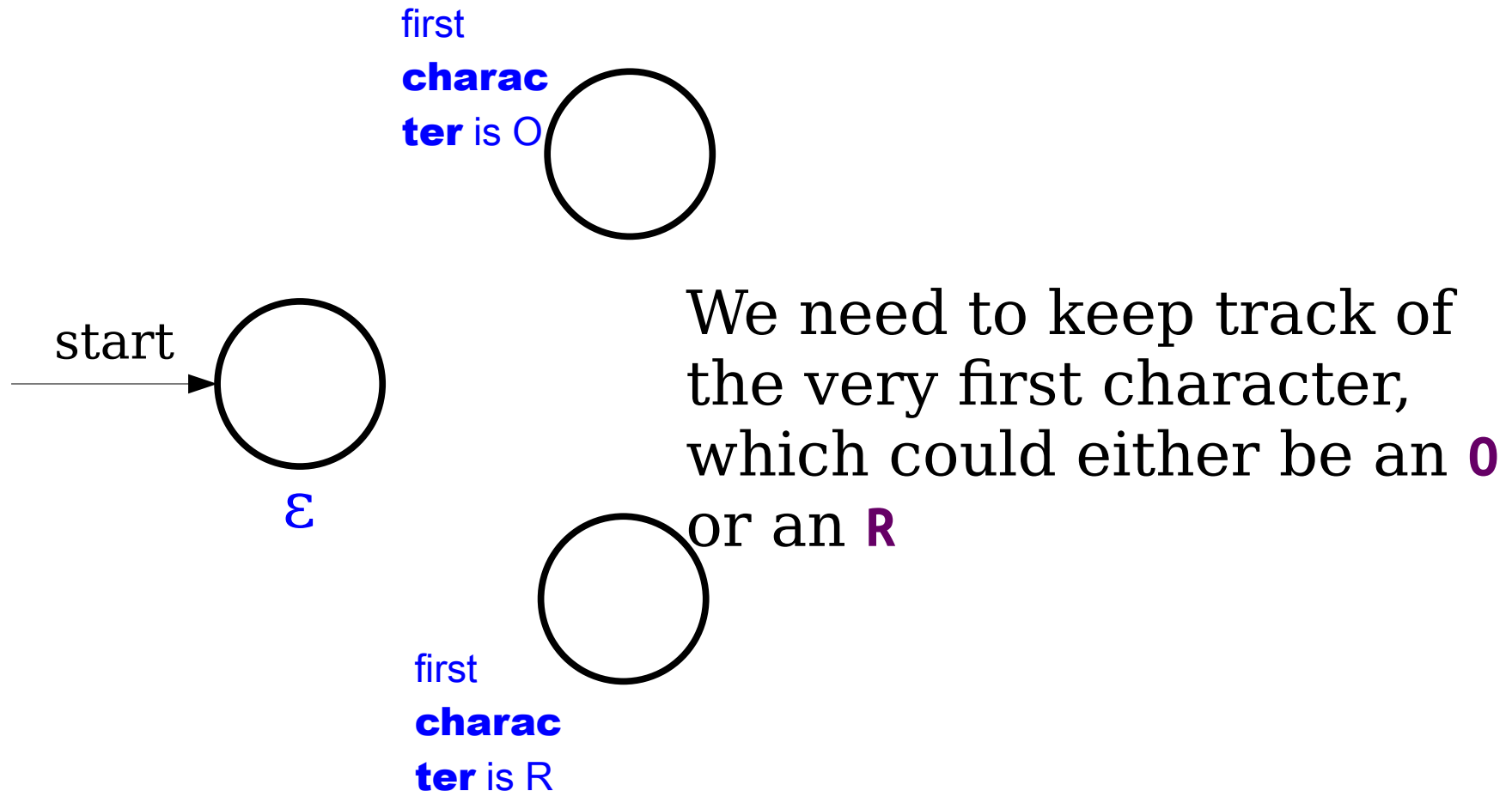
$$L = \{ w \in \Sigma^* \mid w \neq \varepsilon \text{ and the first and last character of } w \text{ are the same} \}$$



We need to keep track of the very first character, which could either be an **0** or an **R**

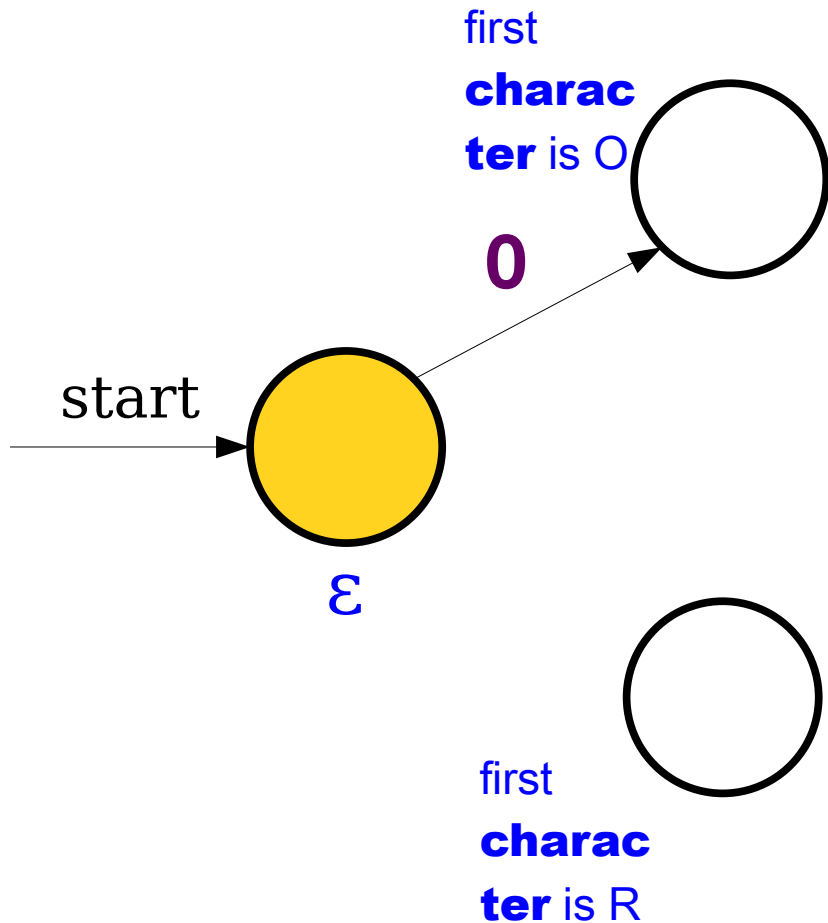
Oreo Sandwiches

$$L = \{ w \in \Sigma^* \mid w \neq \varepsilon \text{ and the first and last character of } w \text{ are the same} \}$$



Oreo Sandwiches

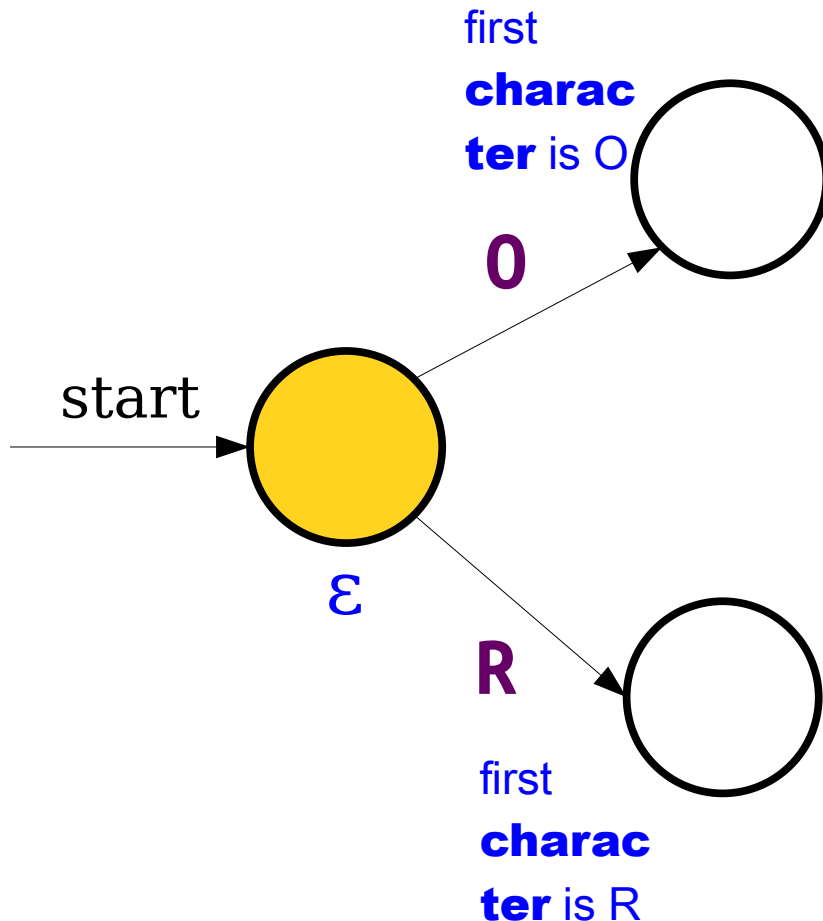
$$L = \{ w \in \Sigma^* \mid w \neq \varepsilon \text{ and the first and last character of } w \text{ are the same} \}$$



If I'm in the start state and I read an **0**, I should transition to this state

Oreo Sandwiches

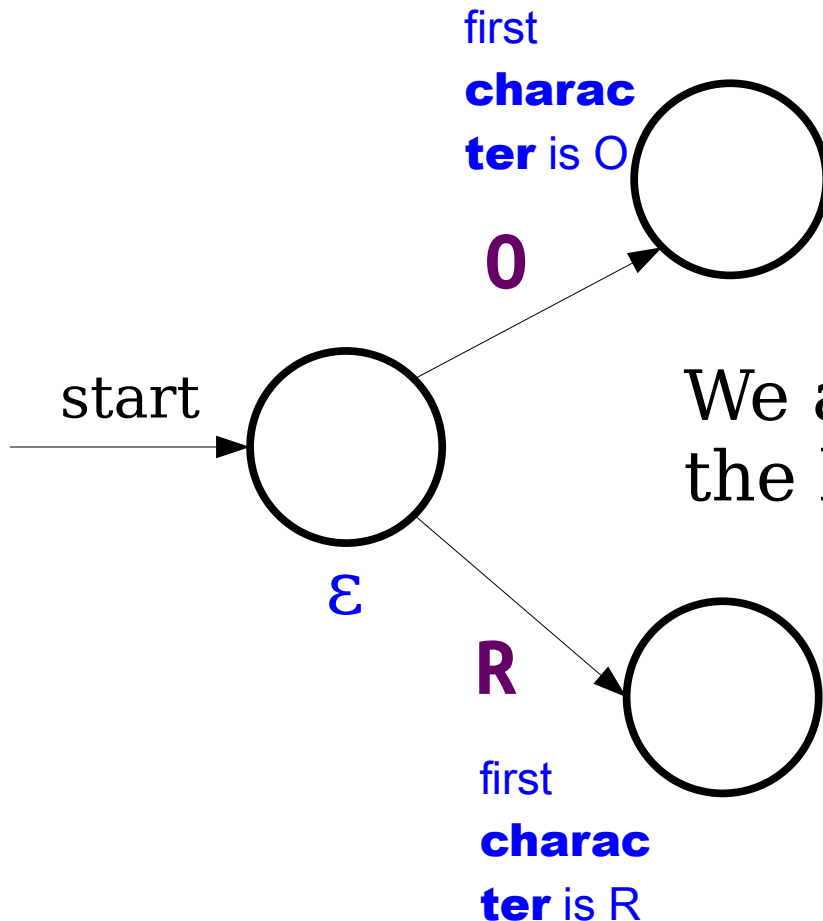
$L = \{ w \in \Sigma^* \mid w \neq \varepsilon \text{ and the first and last character of } w \text{ are the same} \}$



Likewise if I'm in the start state and I read an **R**, I should transition to this state

Oreo Sandwiches

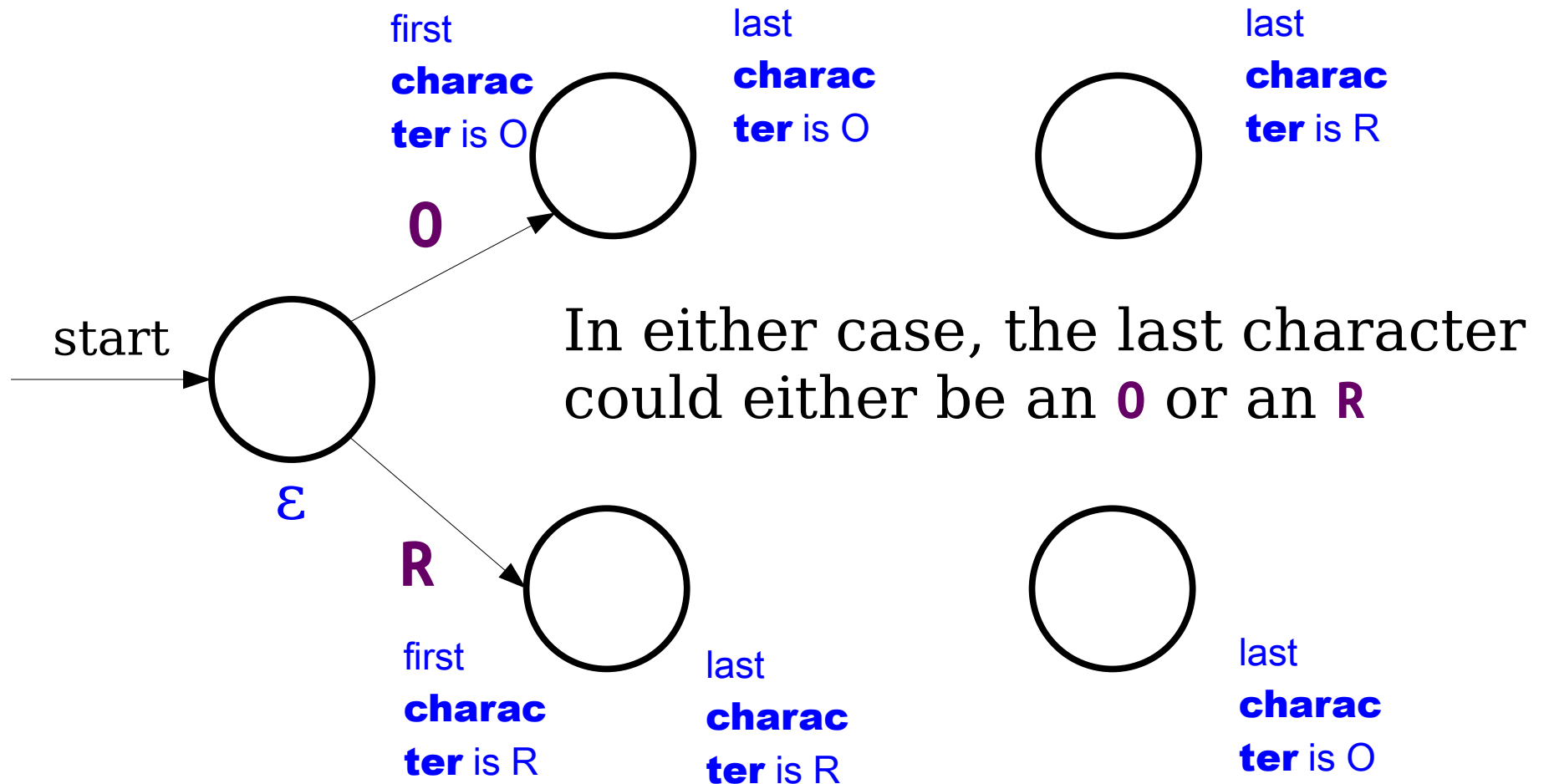
$$L = \{ w \in \Sigma^* \mid w \neq \varepsilon \text{ and the first and last character of } w \text{ are the same} \}$$



We also need to keep track of the last character we've read

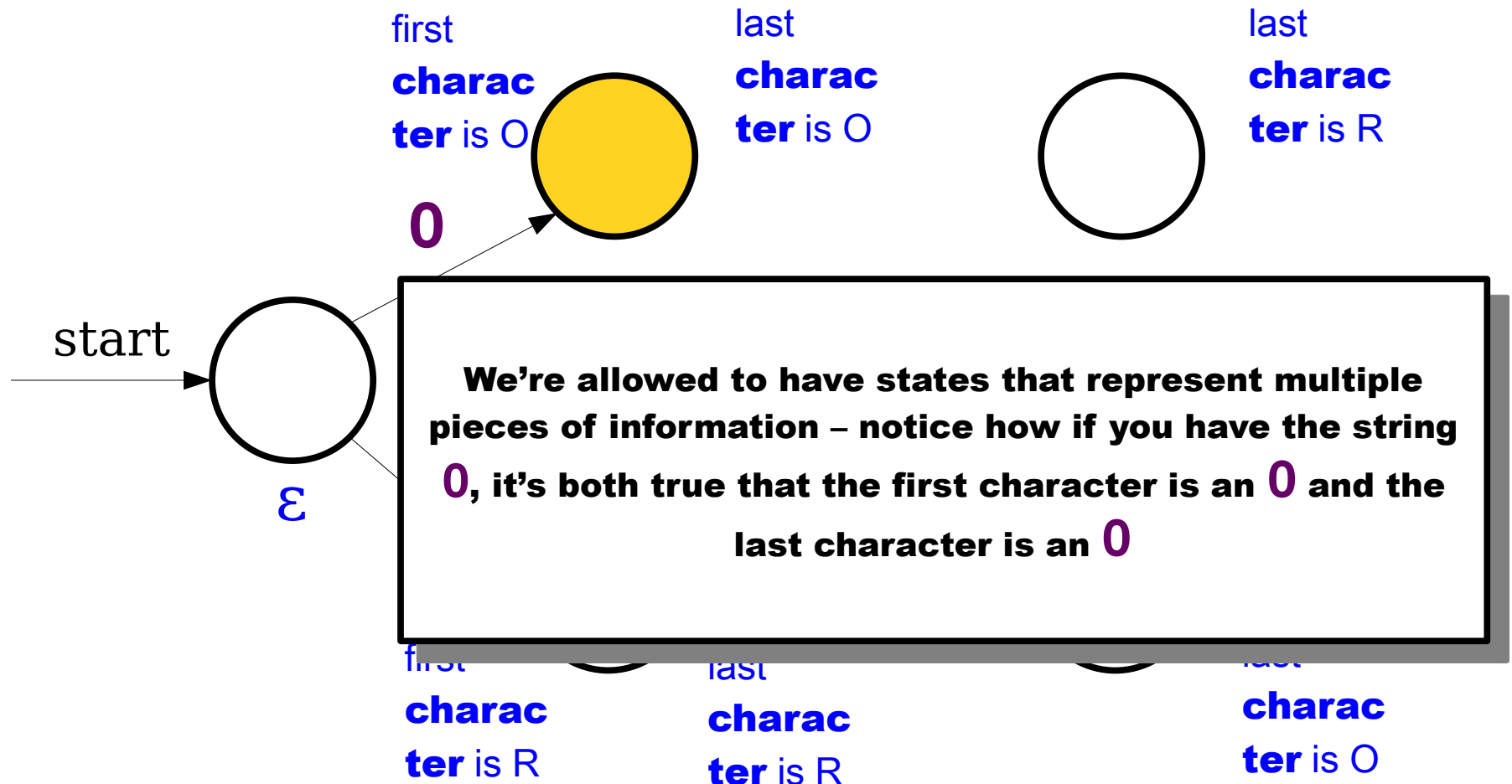
Oreo Sandwiches

$L = \{ w \in \Sigma^* \mid w \neq \varepsilon \text{ and the first and last character of } w \text{ are the same} \}$



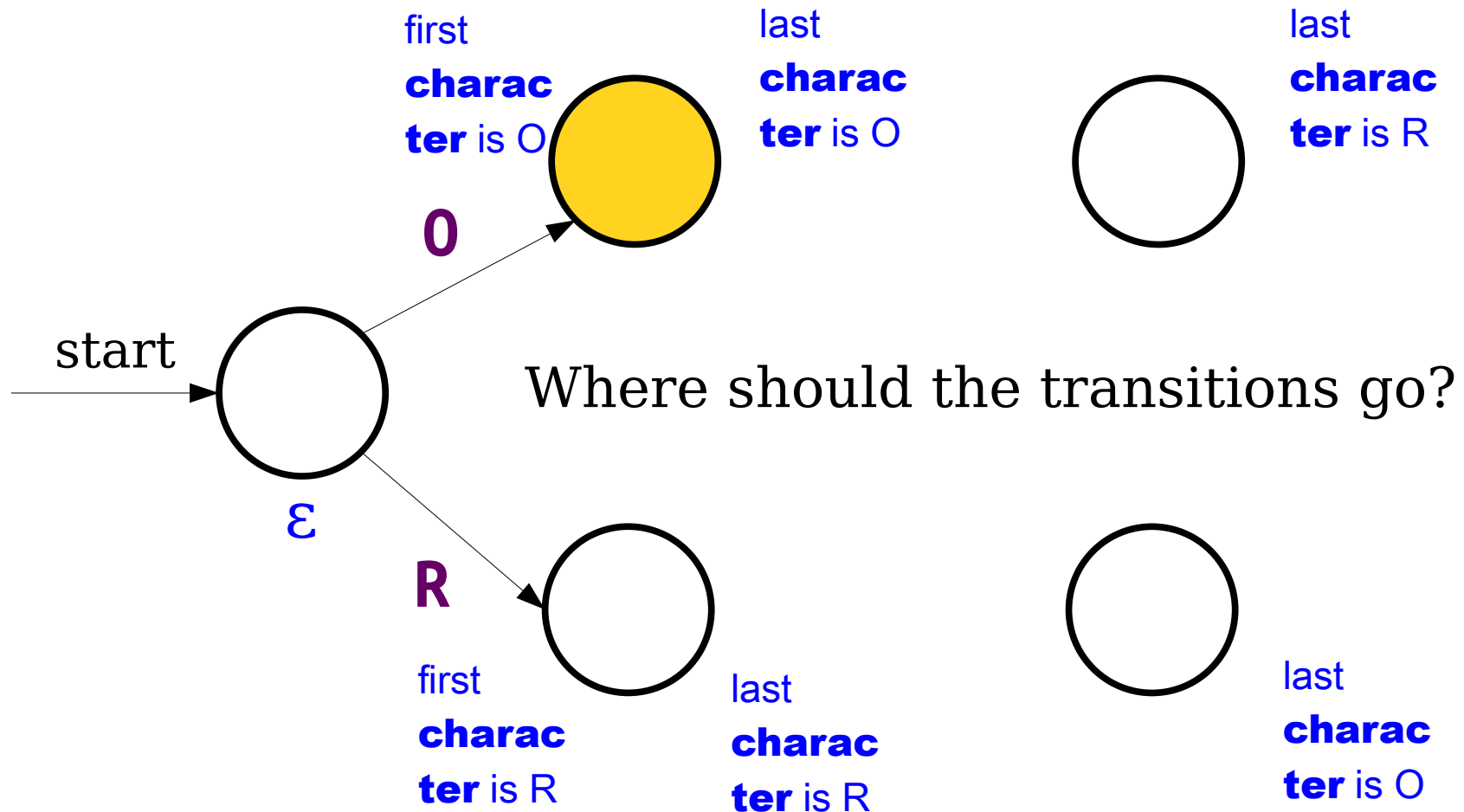
Oreo Sandwiches

$L = \{ w \in \Sigma^* \mid w \neq \varepsilon \text{ and the first and last character of } w \text{ are the same} \}$



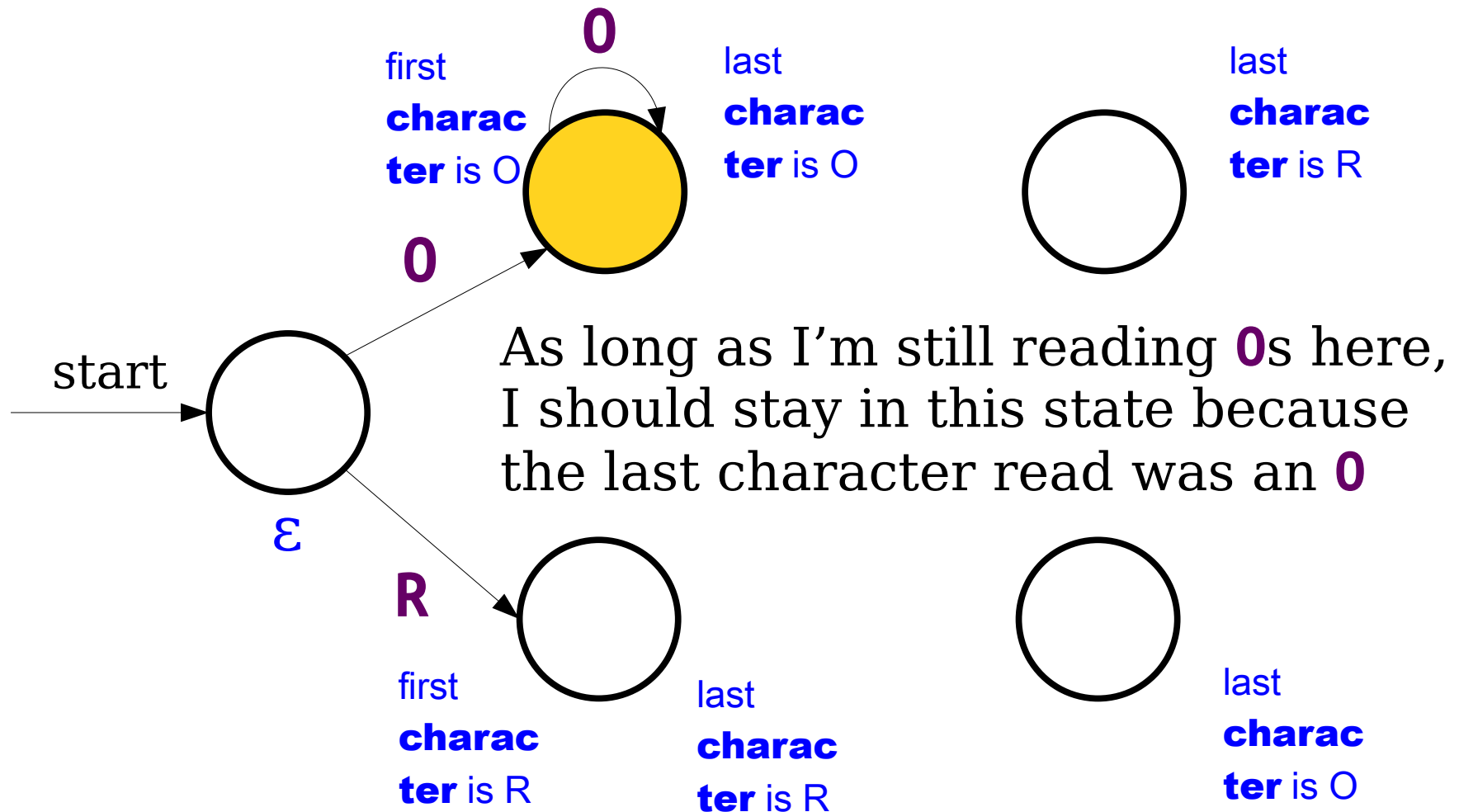
Oreo Sandwiches

$L = \{ w \in \Sigma^* \mid w \neq \varepsilon \text{ and the first and last character of } w \text{ are the same} \}$



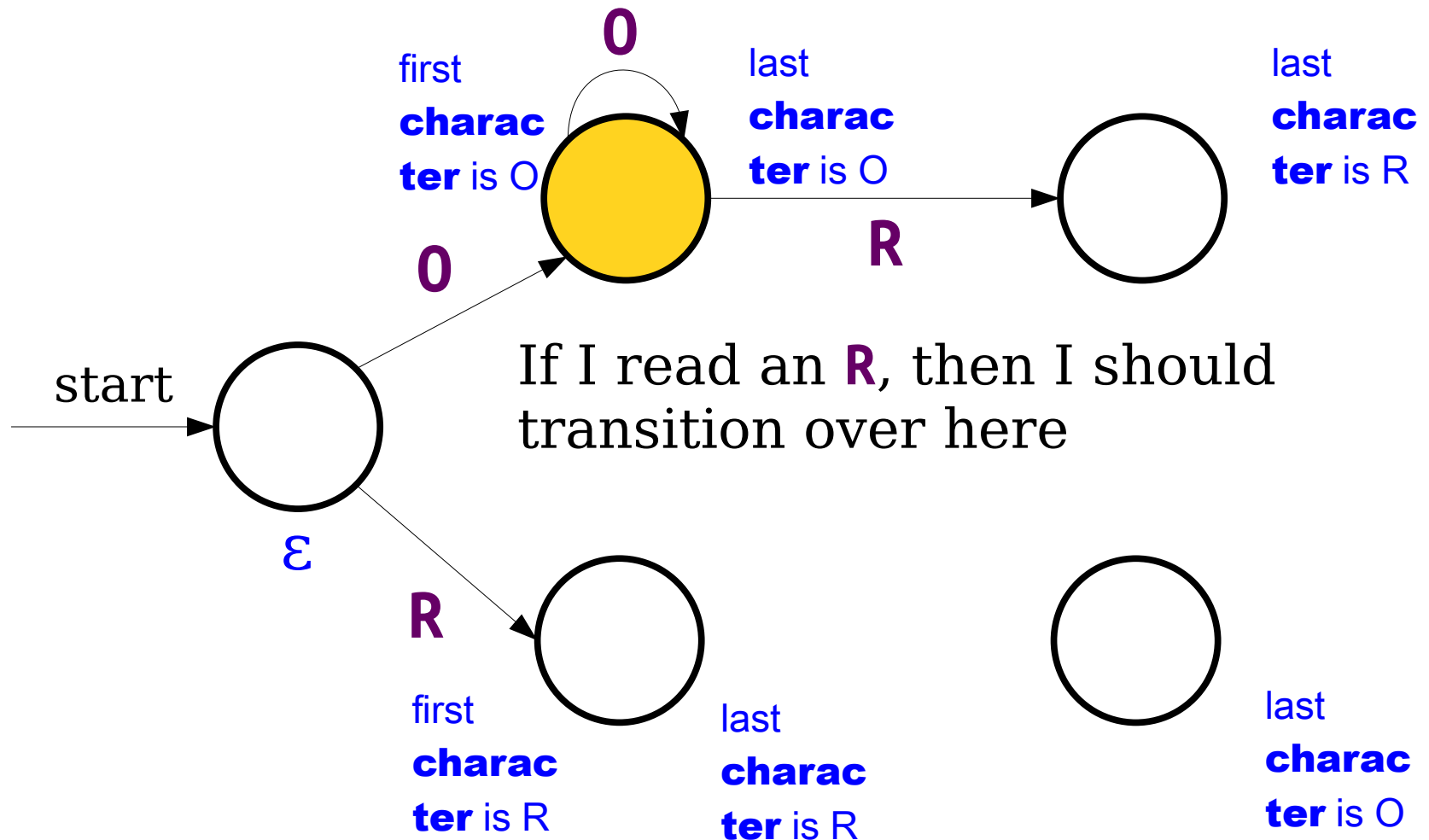
Oreo Sandwiches

$L = \{ w \in \Sigma^* \mid w \neq \varepsilon \text{ and the first and last character of } w \text{ are the same} \}$



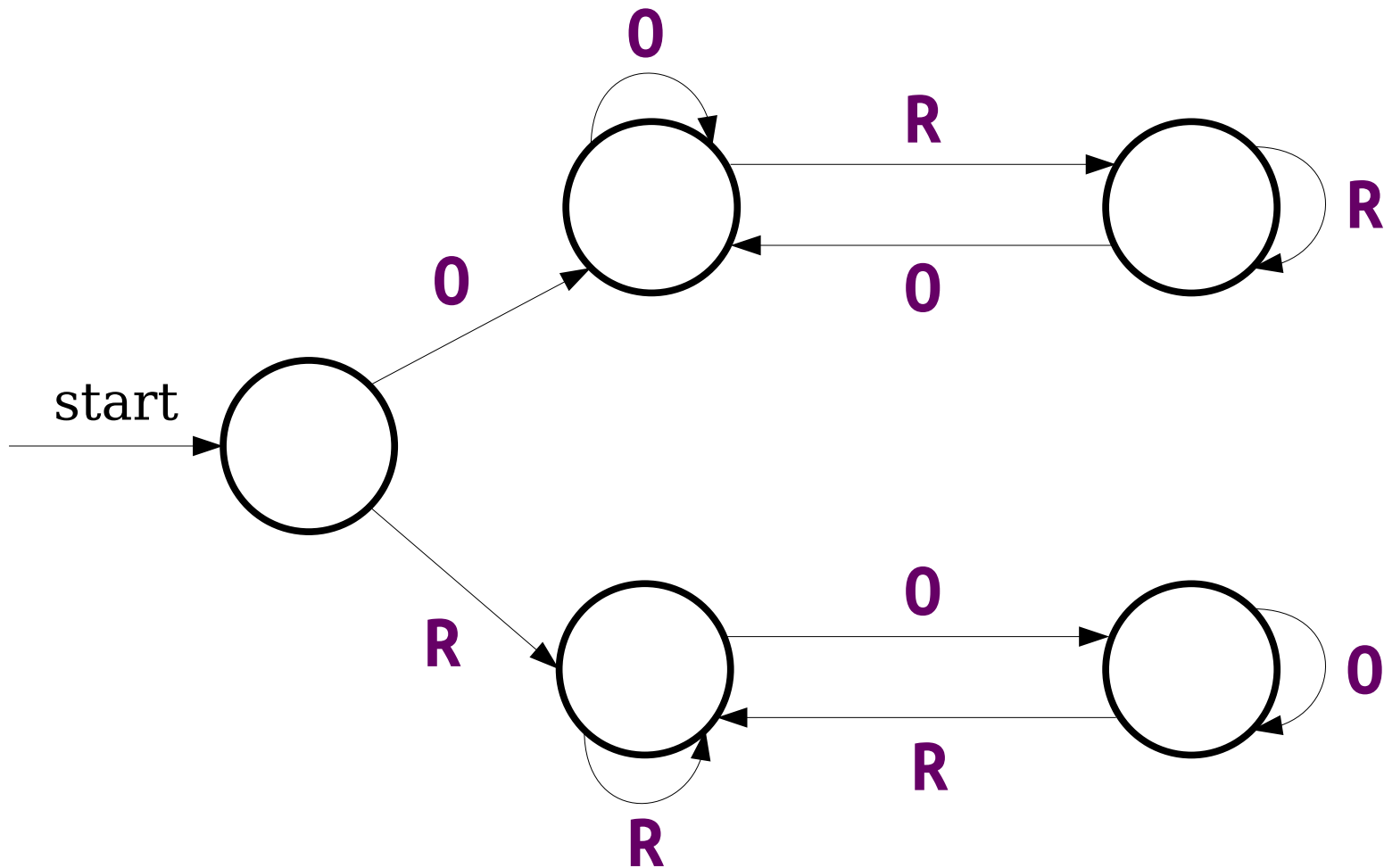
Oreo Sandwiches

$L = \{ w \in \Sigma^* \mid w \neq \varepsilon \text{ and the first and last character of } w \text{ are the same} \}$



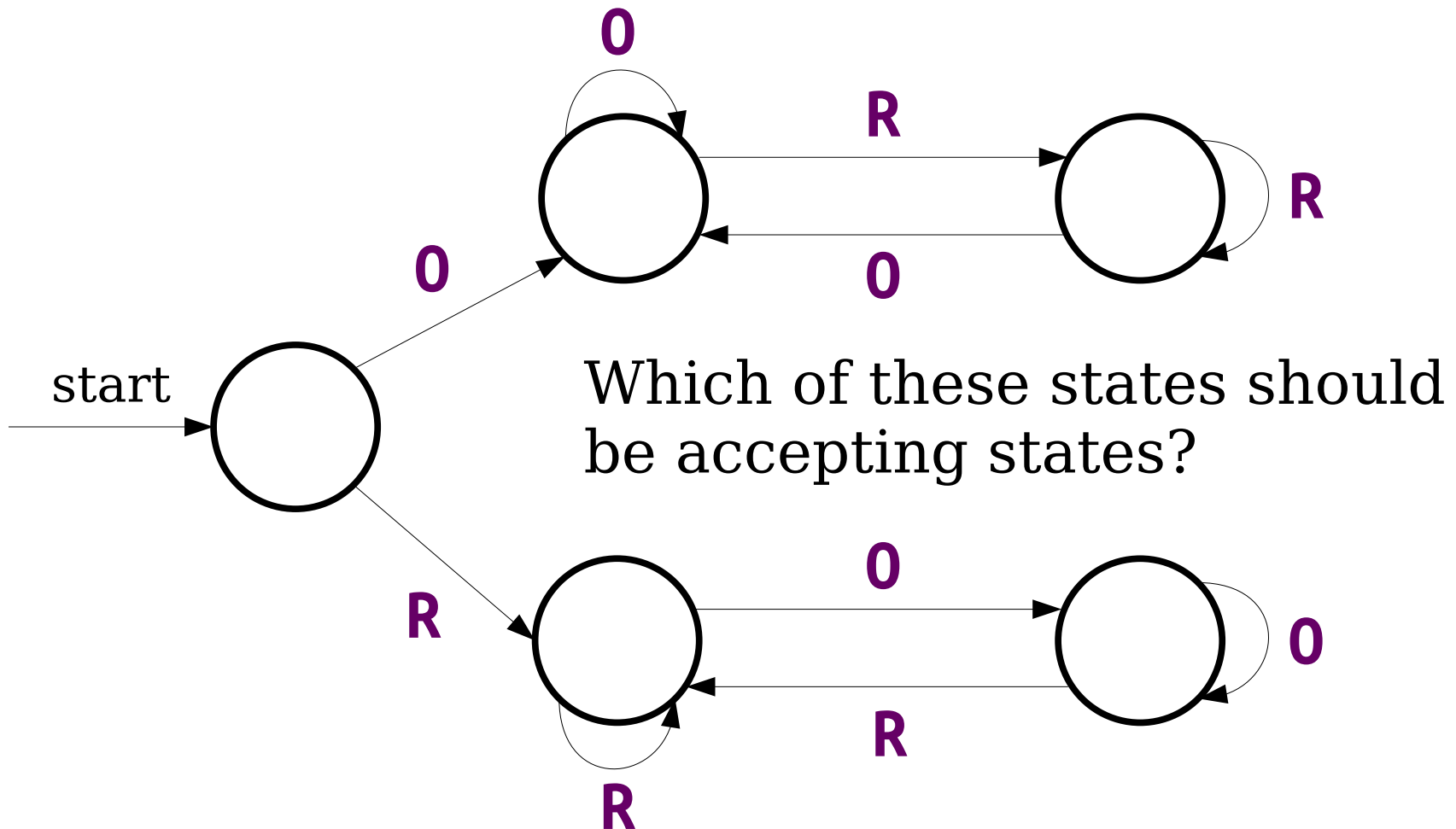
Oreo Sandwiches

$L = \{ w \in \Sigma^* \mid w \neq \varepsilon \text{ and the first and last character of } w \text{ are the same} \}$



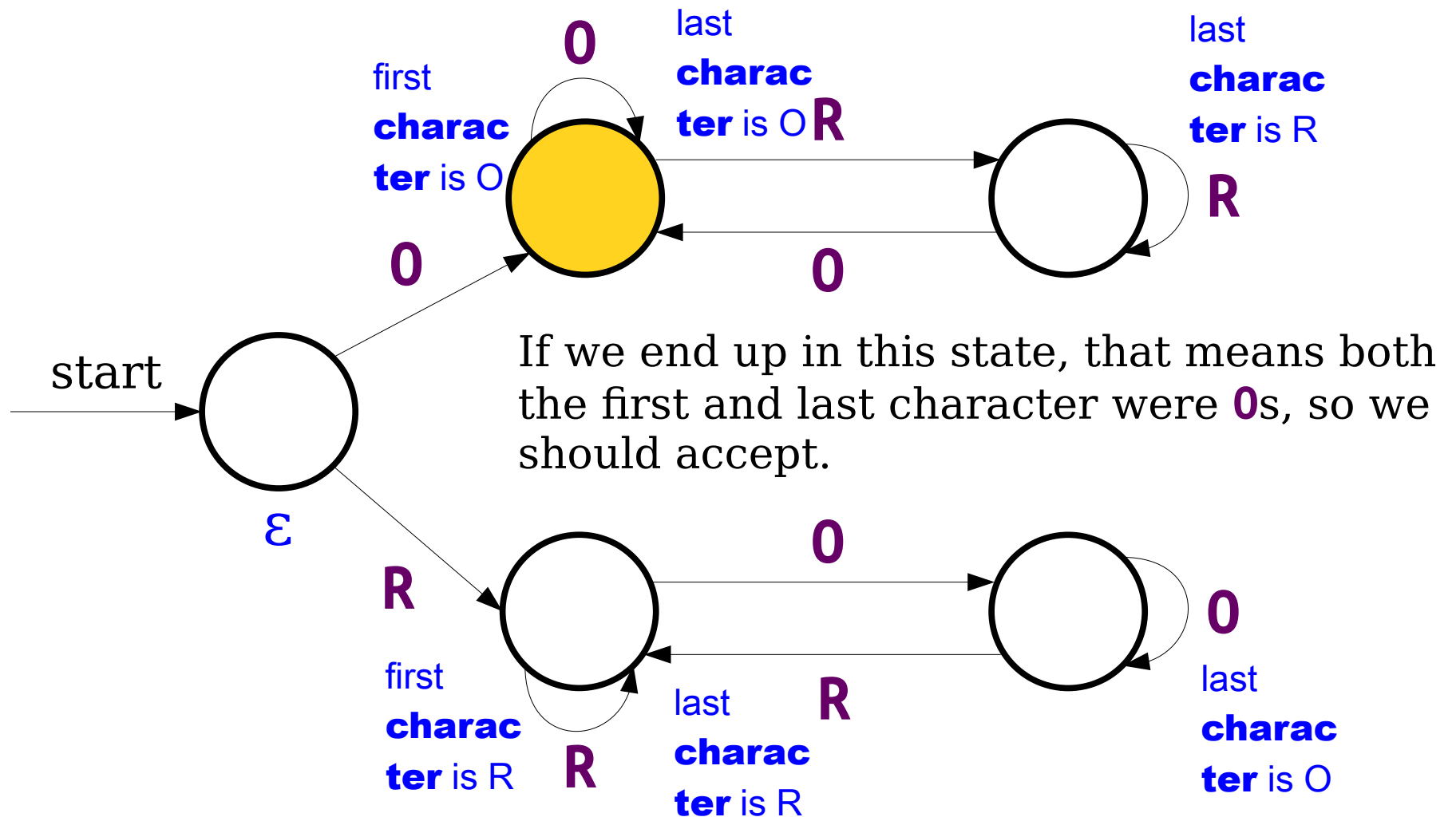
Oreo Sandwiches

$L = \{ w \in \Sigma^* \mid w \neq \varepsilon \text{ and the first and last character of } w \text{ are the same} \}$



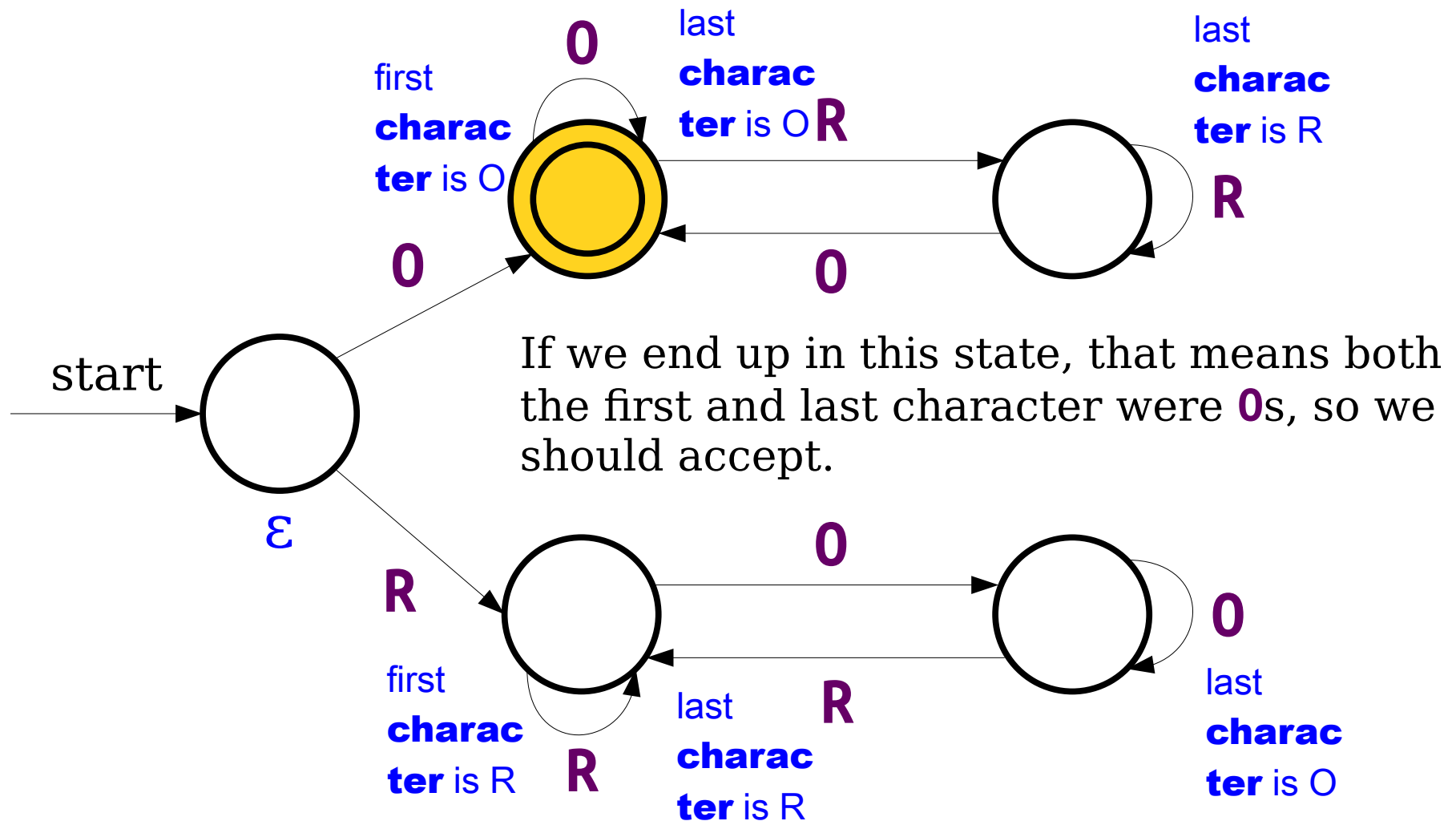
Oreo Sandwiches

$L = \{ w \in \Sigma^* \mid w \neq \varepsilon \text{ and the first and last character of } w \text{ are the same} \}$



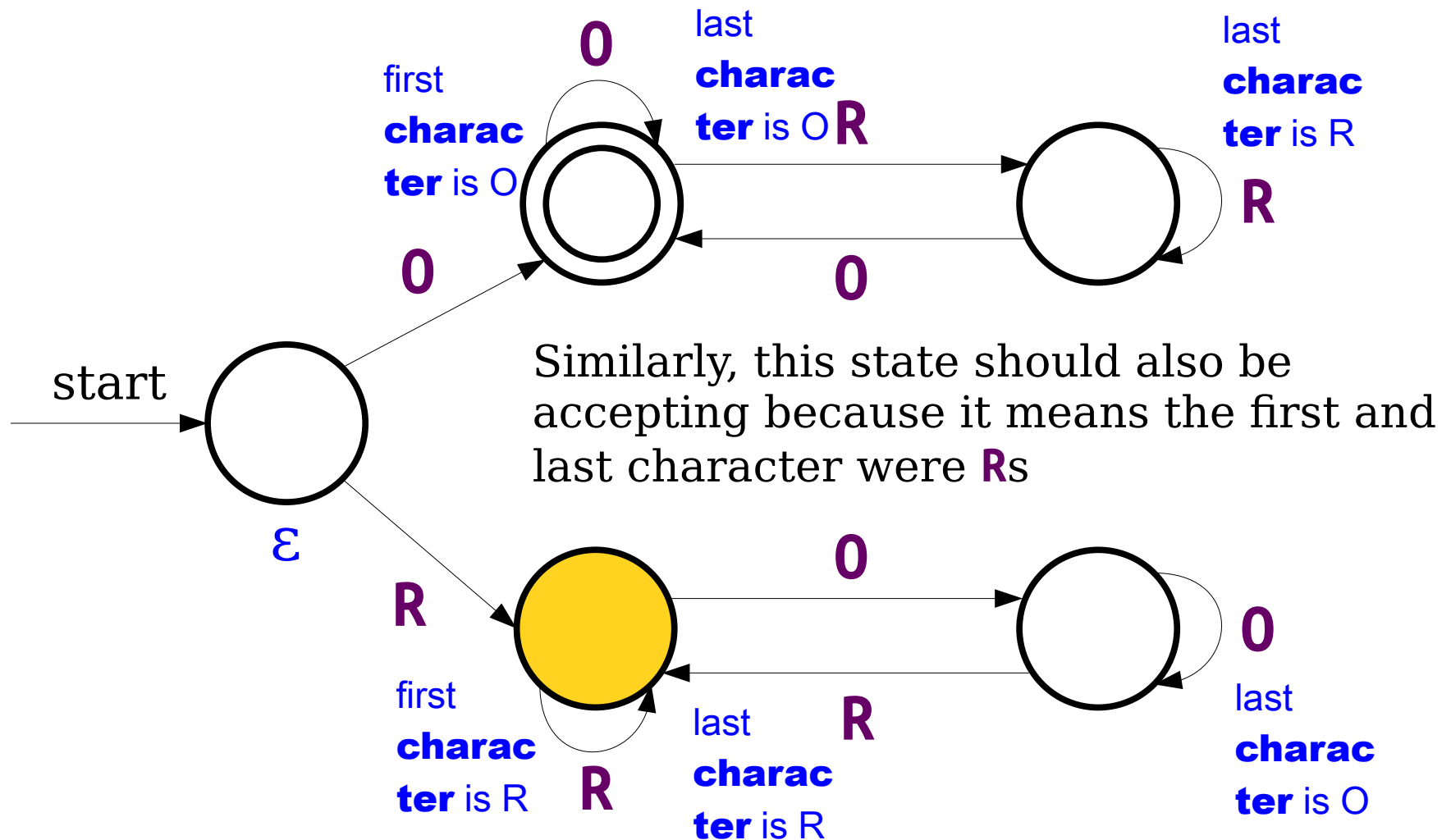
Oreo Sandwiches

$L = \{ w \in \Sigma^* \mid w \neq \varepsilon \text{ and the first and last character of } w \text{ are the same} \}$



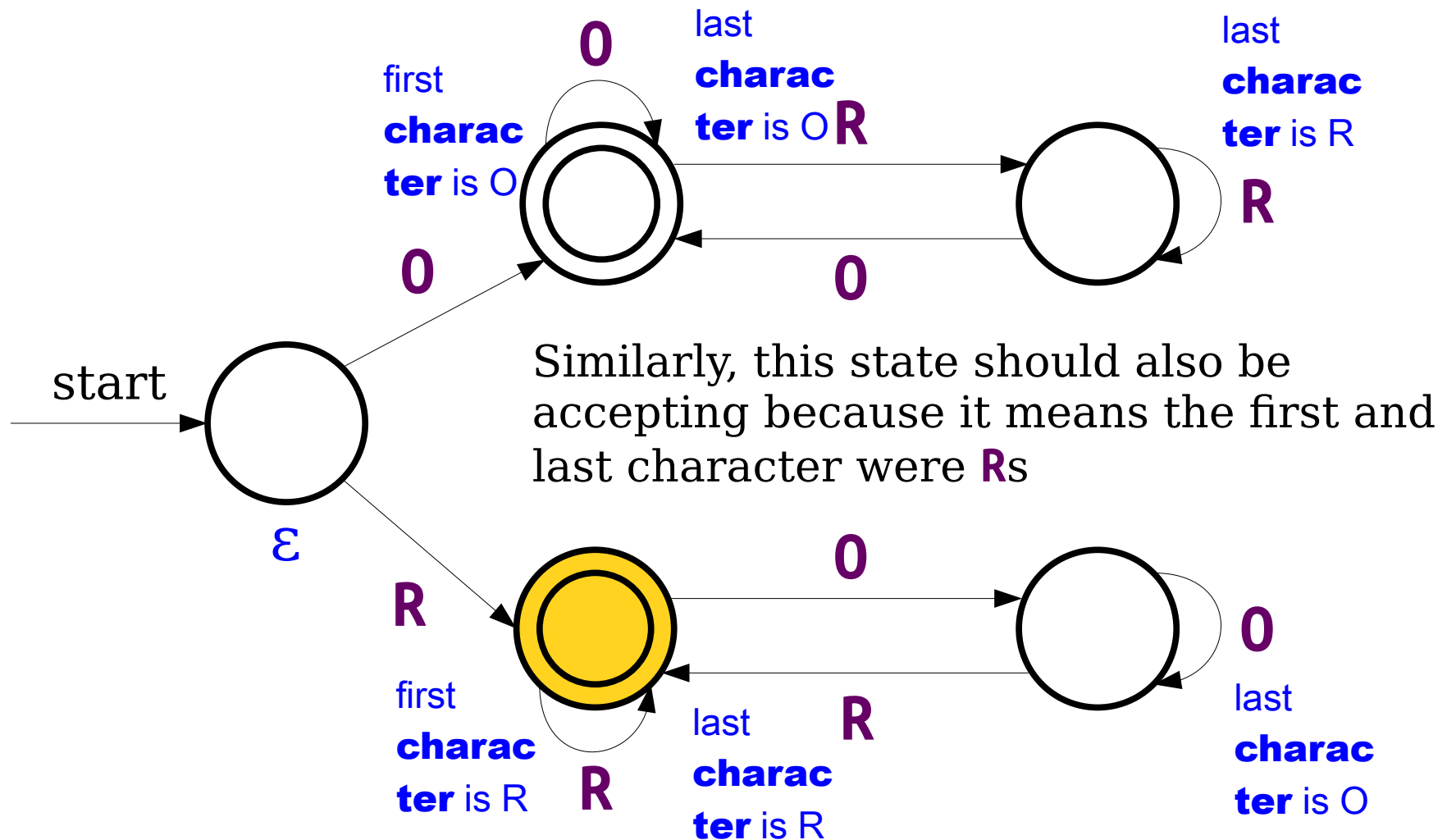
Oreo Sandwiches

$L = \{ w \in \Sigma^* \mid w \neq \varepsilon \text{ and the first and last character of } w \text{ are the same} \}$



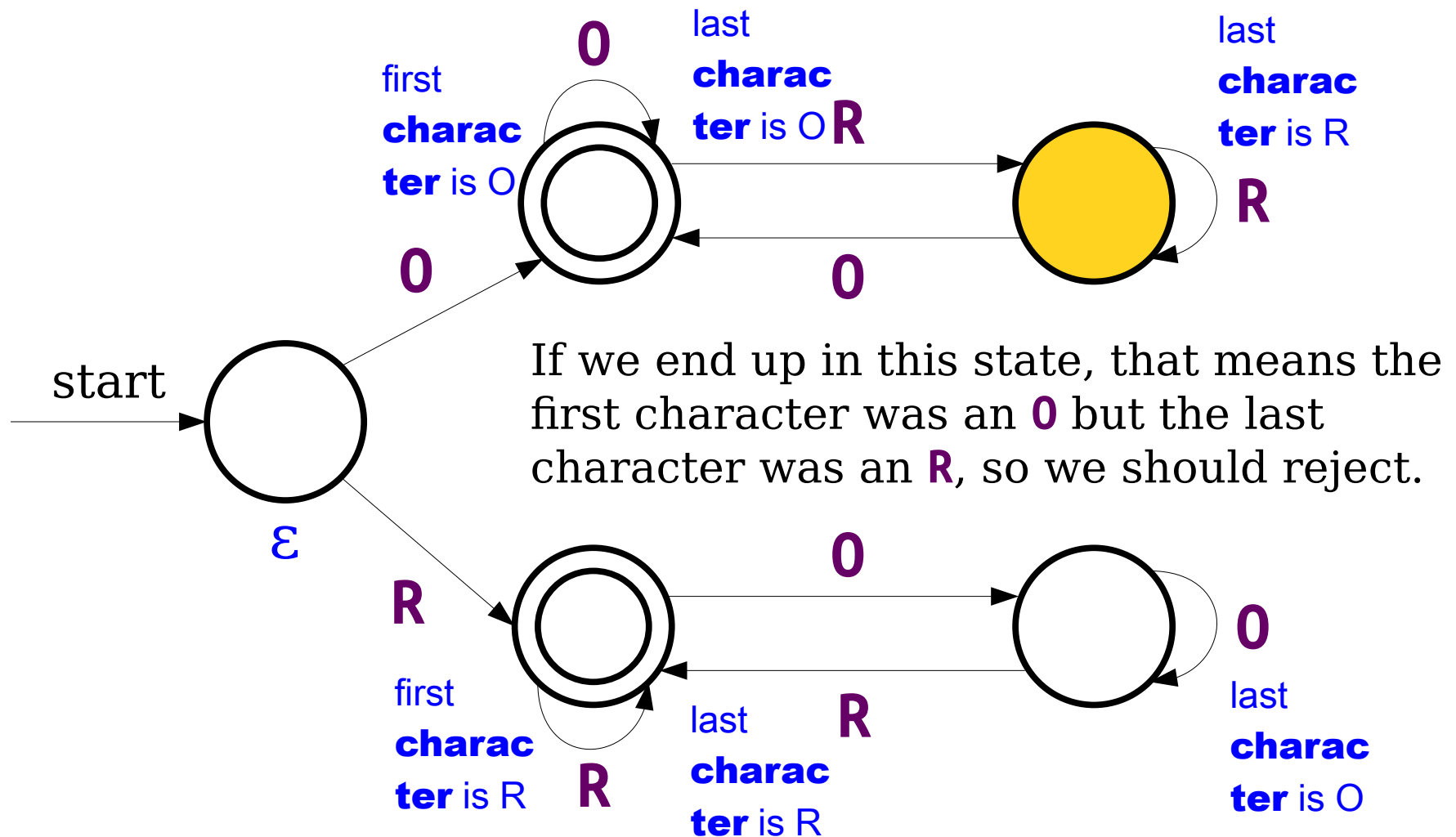
Oreo Sandwiches

$L = \{ w \in \Sigma^* \mid w \neq \varepsilon \text{ and the first and last character of } w \text{ are the same} \}$



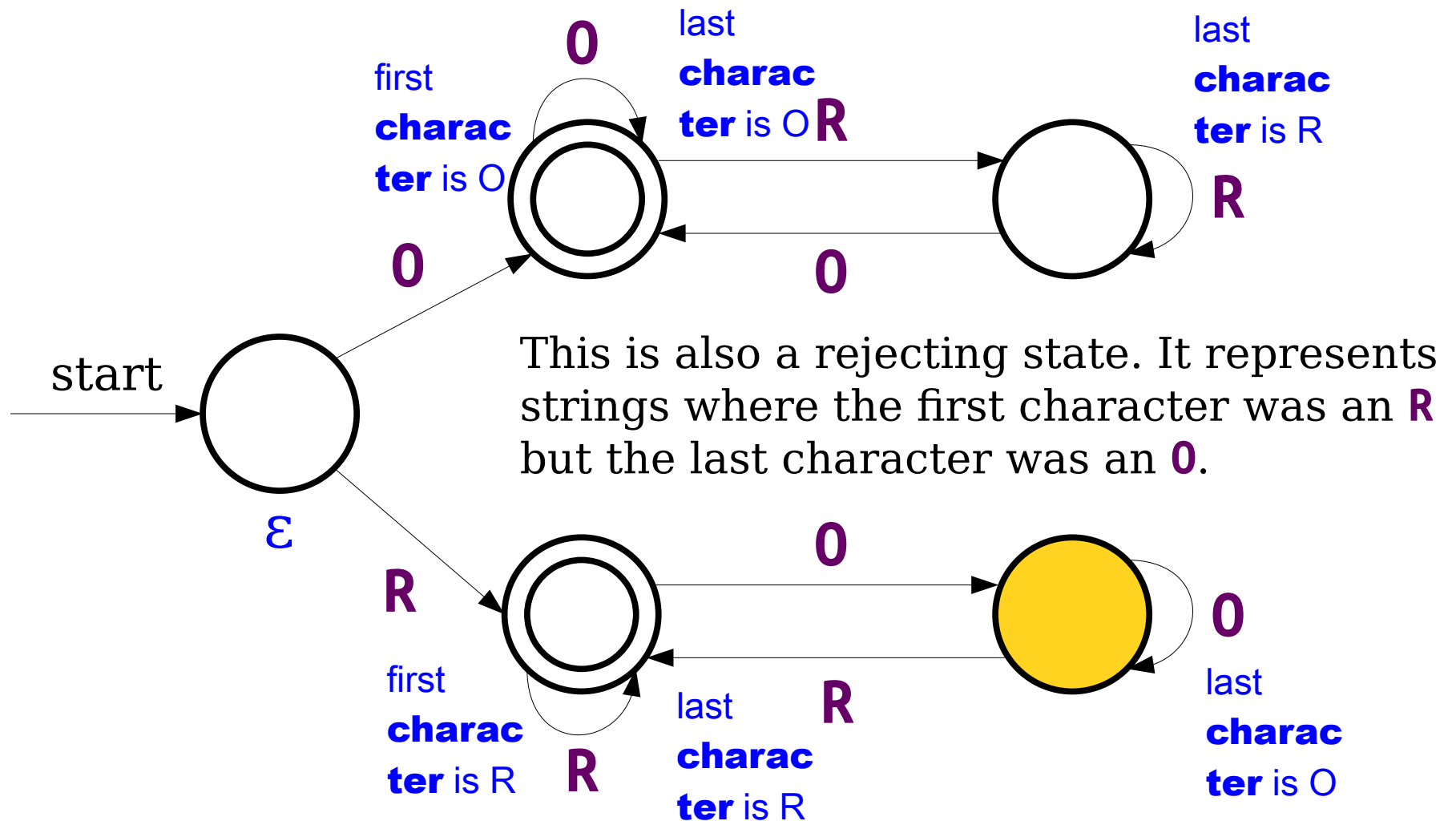
Oreo Sandwiches

$L = \{ w \in \Sigma^* \mid w \neq \varepsilon \text{ and the first and last character of } w \text{ are the same} \}$

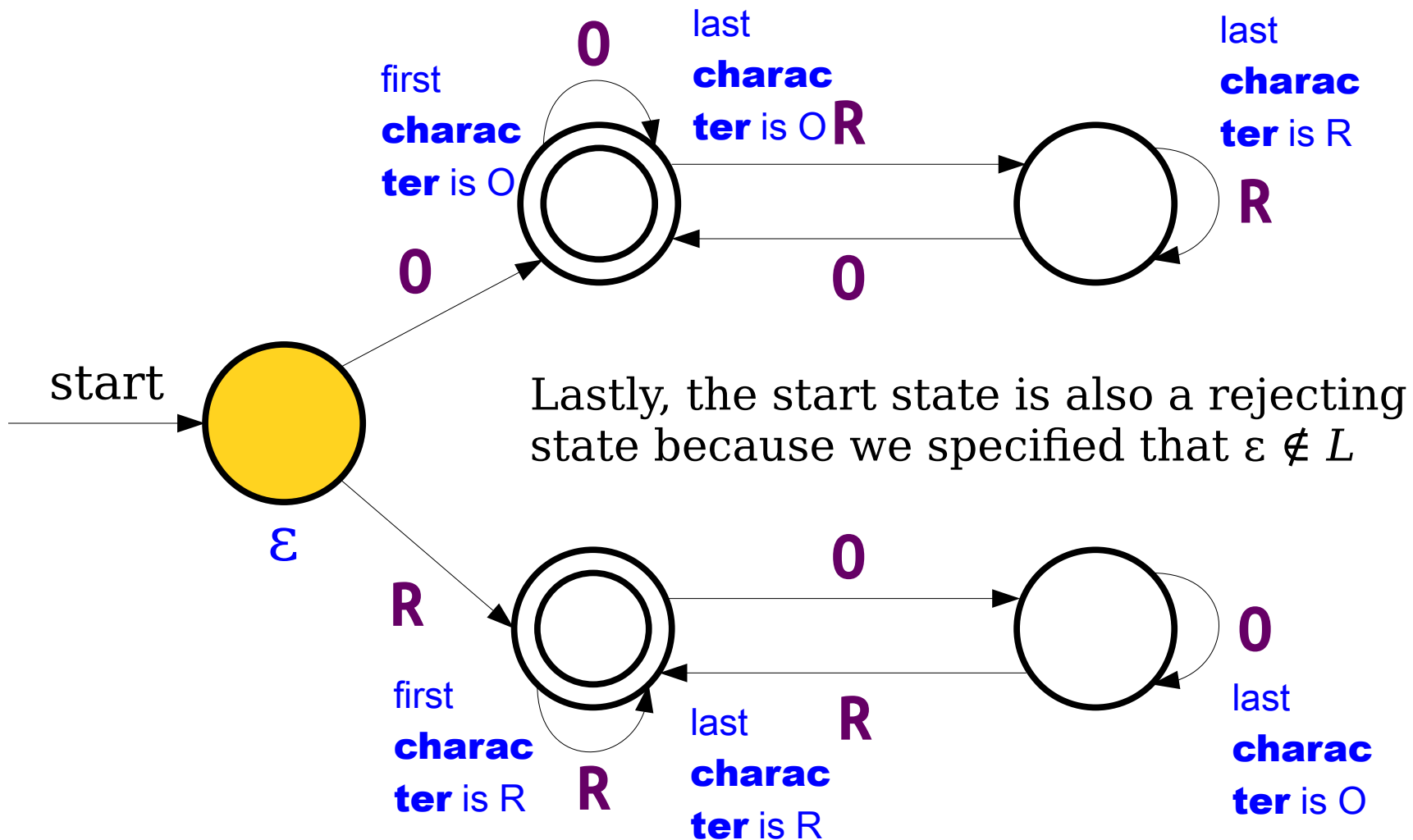


Oreo Sandwiches

$L = \{ w \in \Sigma^* \mid w \neq \varepsilon \text{ and the first and last character of } w \text{ are the same} \}$

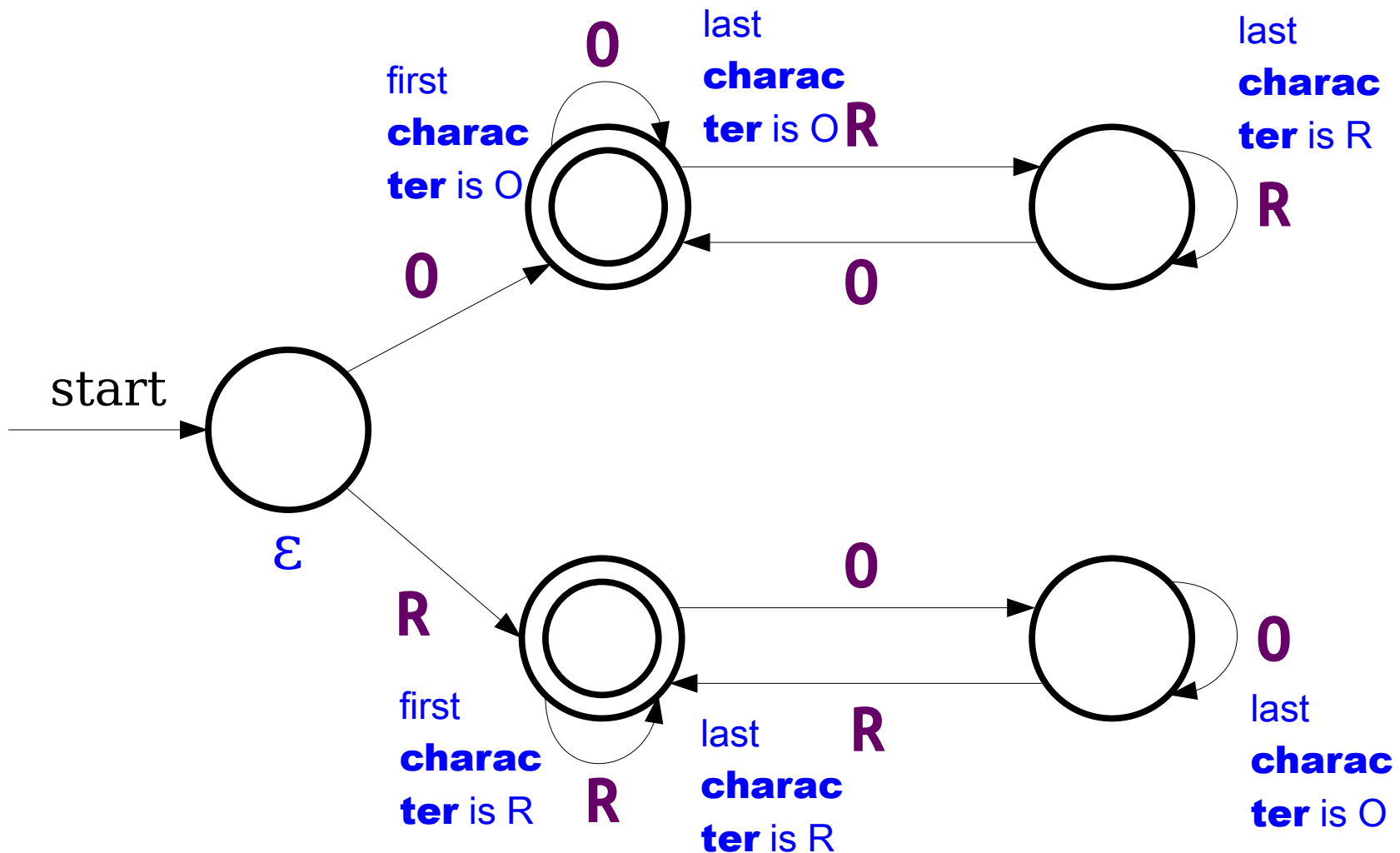


Oreo Sandwiches

$$L = \{ w \in \Sigma^* \mid w \neq \varepsilon \text{ and the first and last character of } w \text{ are the same} \}$$


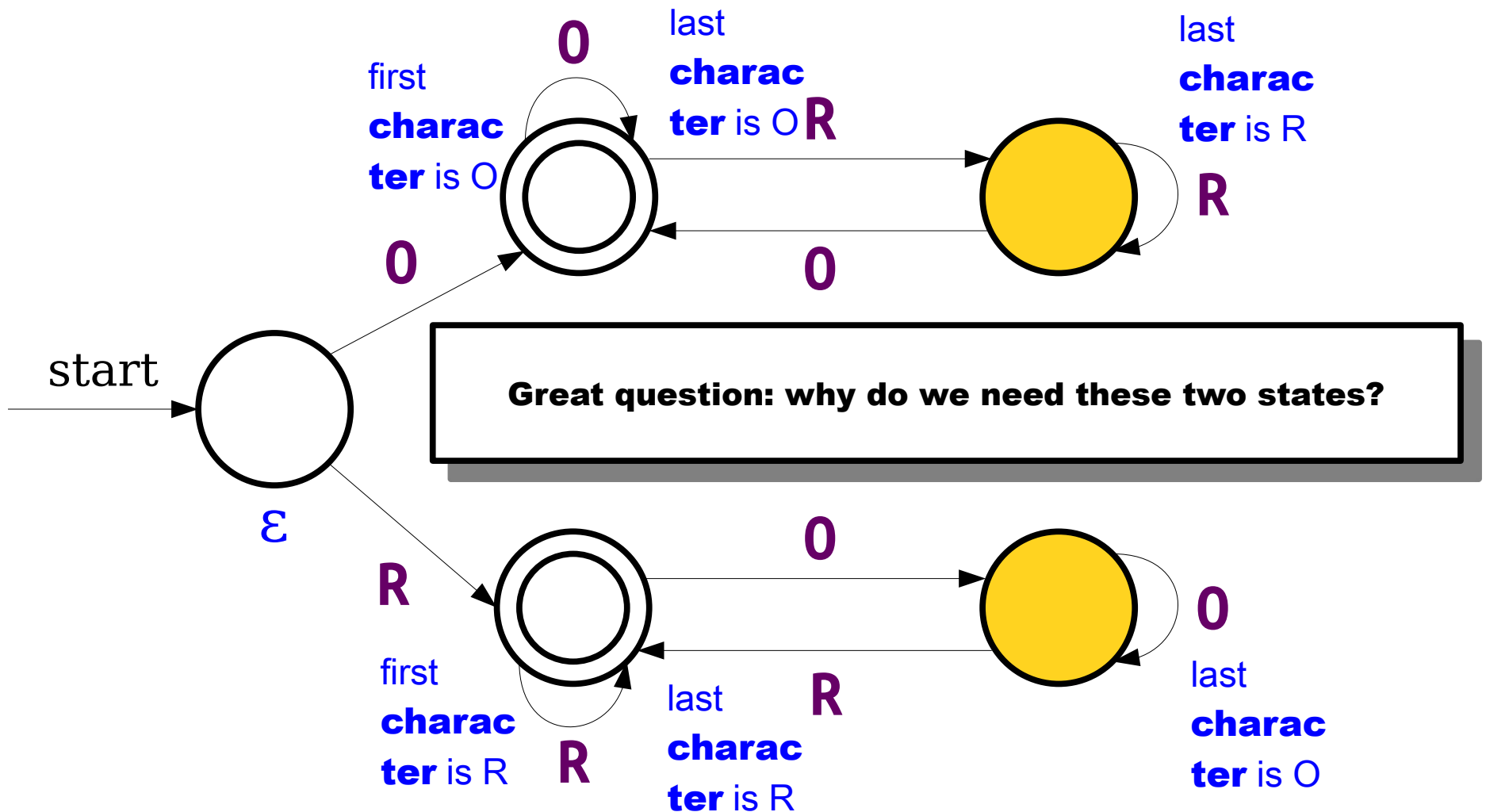
Oreo Sandwiches

$L = \{ w \in \Sigma^* \mid w \neq \varepsilon \text{ and the first and last character of } w \text{ are the same} \}$



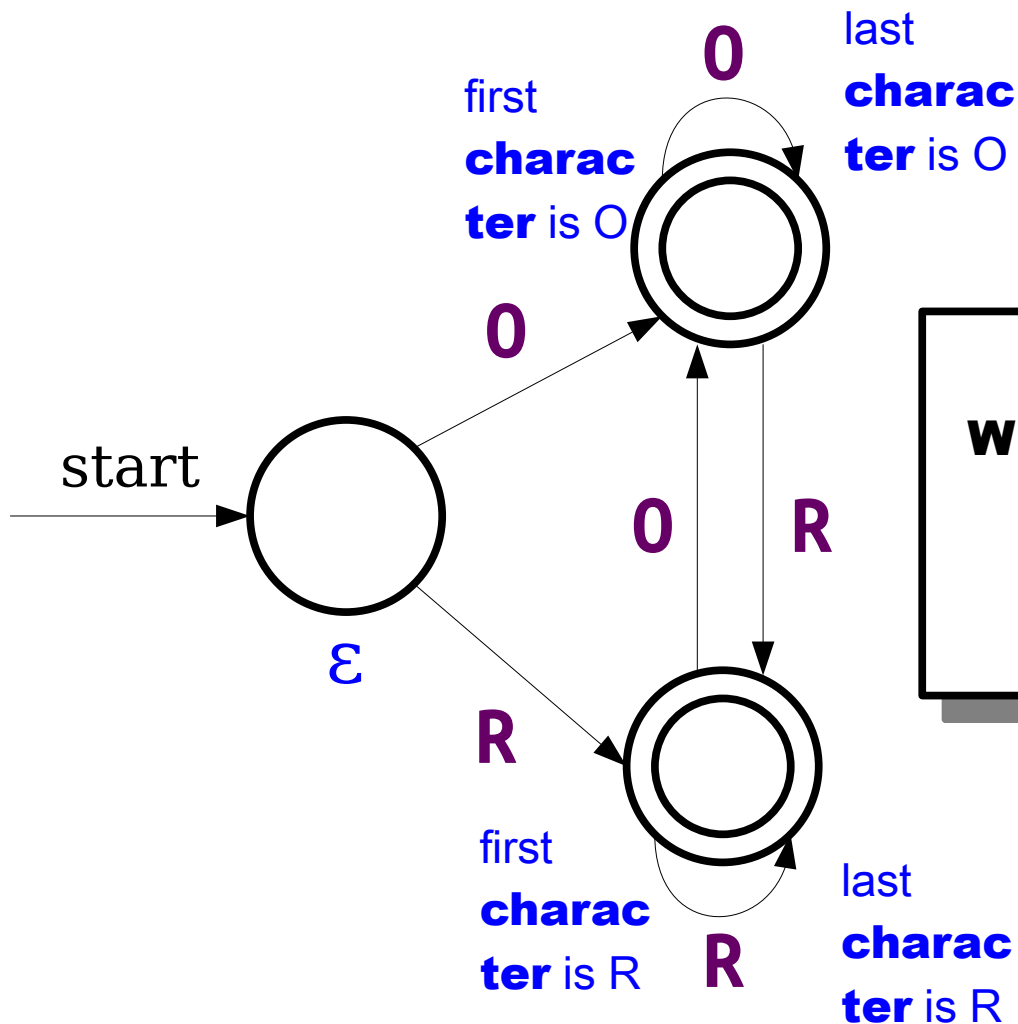
Oreo Sandwiches

$L = \{ w \in \Sigma^* \mid w \neq \varepsilon \text{ and the first and last character of } w \text{ are the same} \}$



Oreo Sandwiches

$L = \{ w \in \Sigma^* \mid w \neq \varepsilon \text{ and the first and last character of } w \text{ are the same} \}$



Why can't we have a DFA that looks like this for this language?

Part 2: ***Designing NFAs***

Designing NFAs

- Is there some information that you'd really like to have?
 - Have the machine *nondeterministically guess* that information.
 - Then, have the machine *deterministically check* that the choice was correct.

More Oreo Sandwiches

- Let $\Sigma = \{ \mathbf{O}, \mathbf{R} \}$

Design an NFA for the language

$$L = \{ w \in \Sigma^* \mid \text{Some character of } \Sigma \text{ appears at most twice in } w \}$$

More Oreo Sandwiches

- Let $\Sigma = \{ \mathbf{O}, \mathbf{R} \}$

Design an NFA for the language

$L = \{ w \in \Sigma^* \mid \text{Some character of } \Sigma \text{ appears at most twice in } w \}$

$\varepsilon \in L$

$\mathbf{R} \in L$

$\mathbf{ORO} \in L$

$\mathbf{RRORR} \in L$

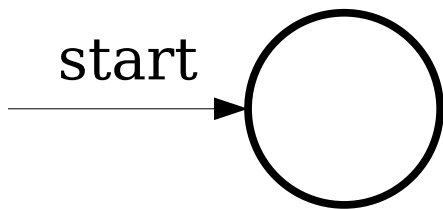
$\mathbf{RRR000} \notin L$

$\mathbf{OR0ORRO} \notin L$

$\mathbf{ROROR000} \notin L$

More Oreo Sandwiches

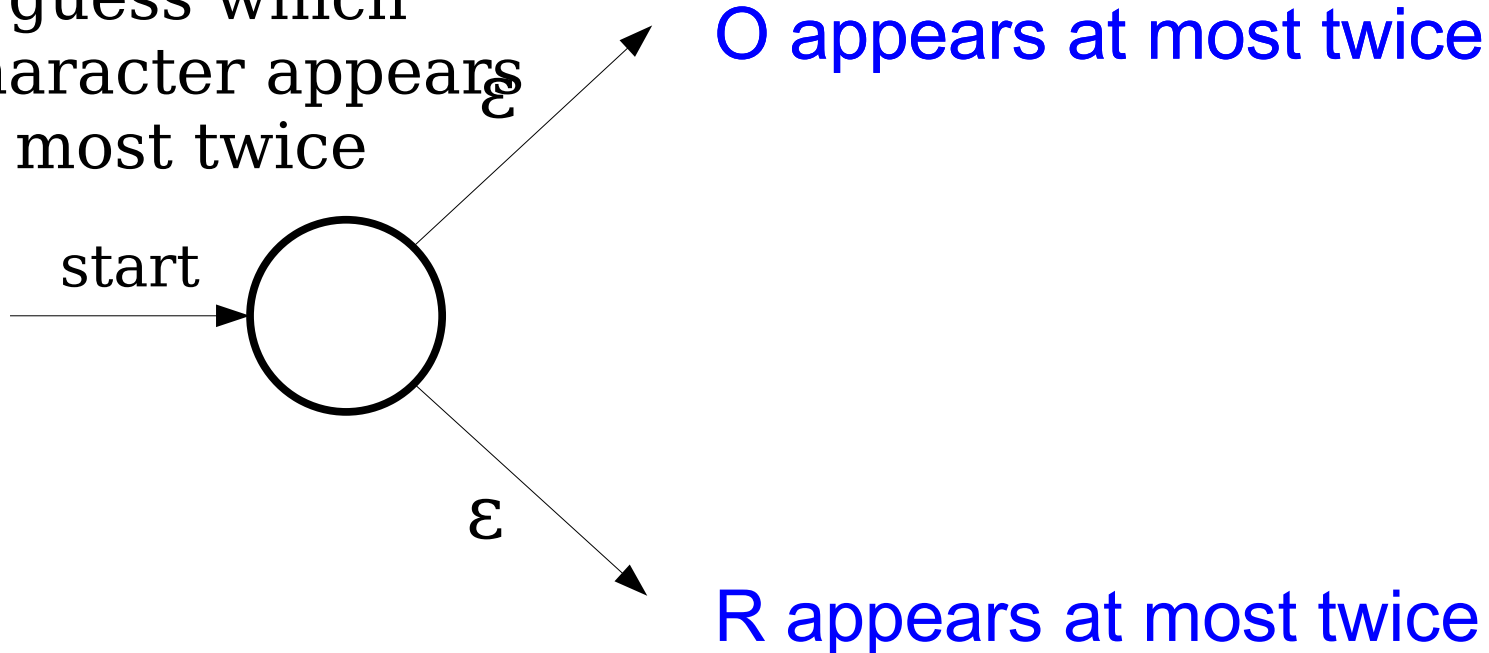
$L = \{ w \in \Sigma^* \mid \text{Some character of } \Sigma \text{ appears} \\ \text{at most twice in } w \}$



More Oreo Sandwiches

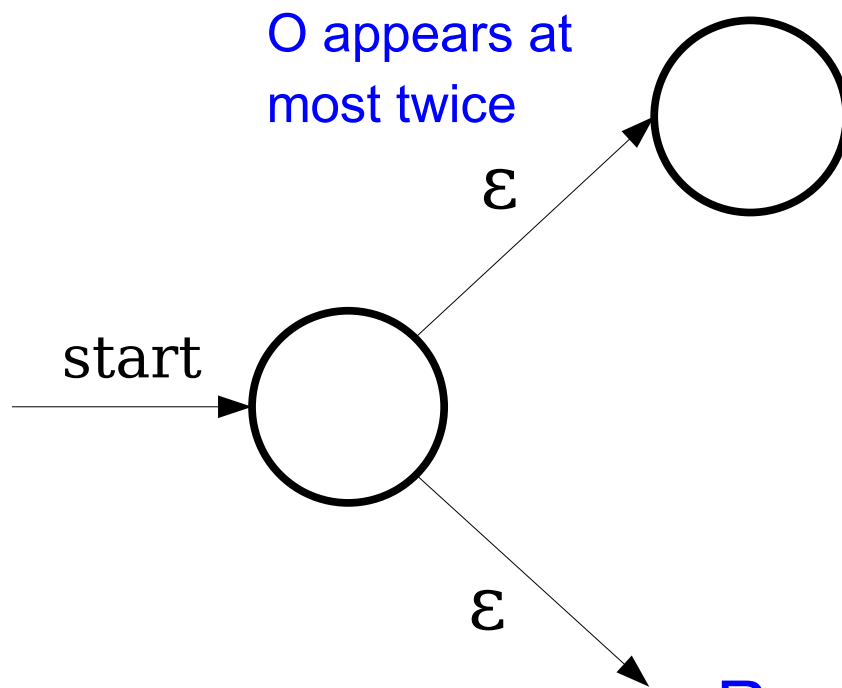
$$L = \{ w \in \Sigma^* \mid \text{Some character of } \Sigma \text{ appears at most twice in } w \}$$

Have the machine
nondeterministicaly
guess which
character appears
at most twice



More Oreo Sandwiches

$L = \{ w \in \Sigma^* \mid \text{Some character of } \Sigma \text{ appears at most twice in } w \}$

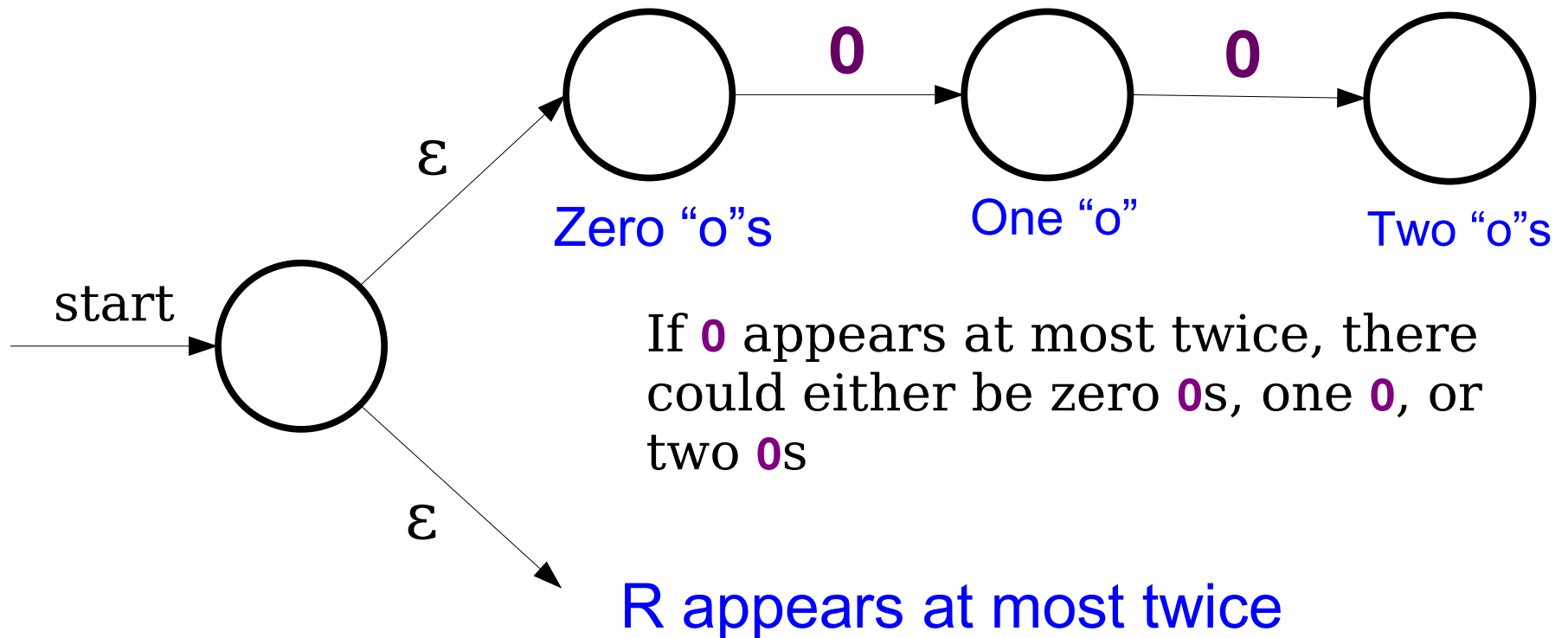


Now, have the machine deterministically check whether or not **o** actually does appear at most twice.

R appears at most twice

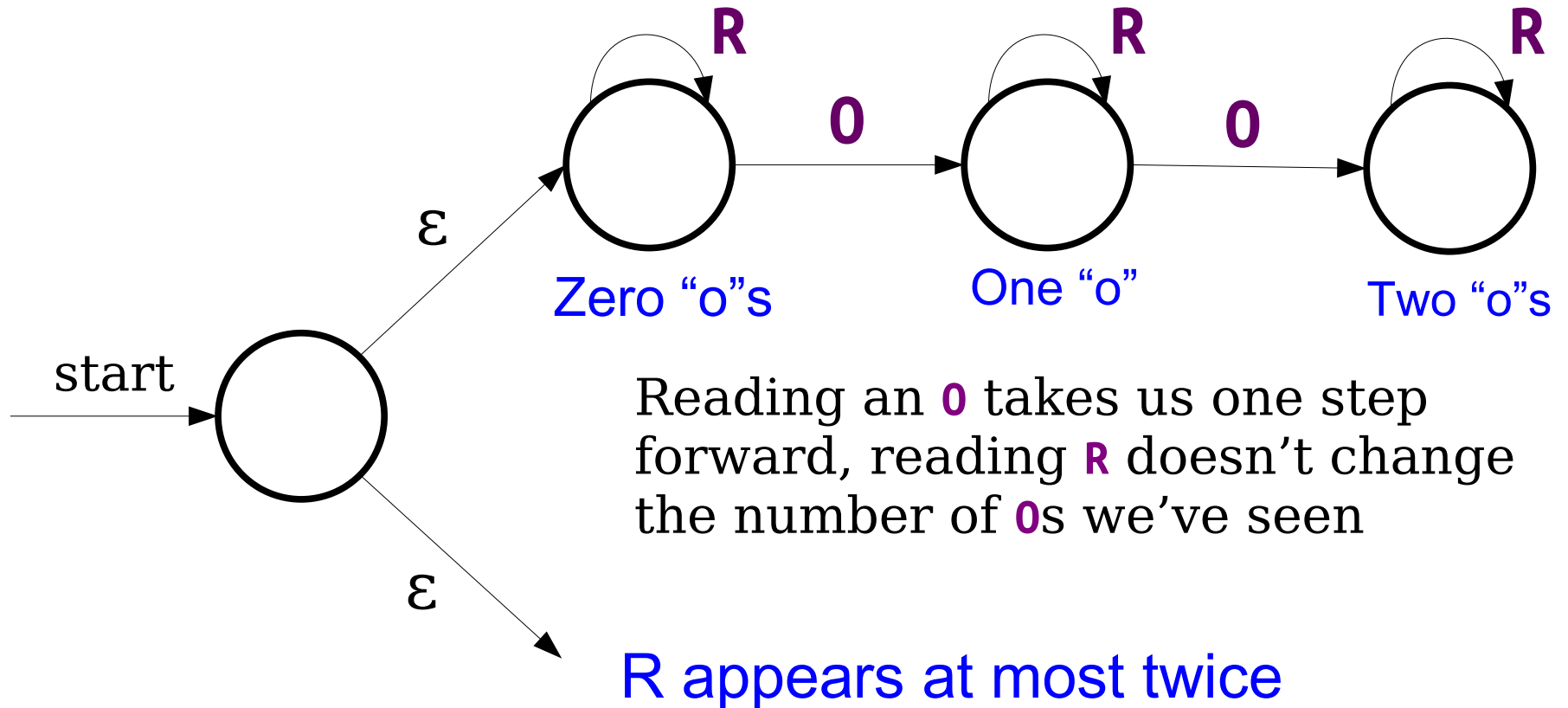
More Oreo Sandwiches

$L = \{ w \in \Sigma^* \mid \text{Some character of } \Sigma \text{ appears at most twice in } w \}$



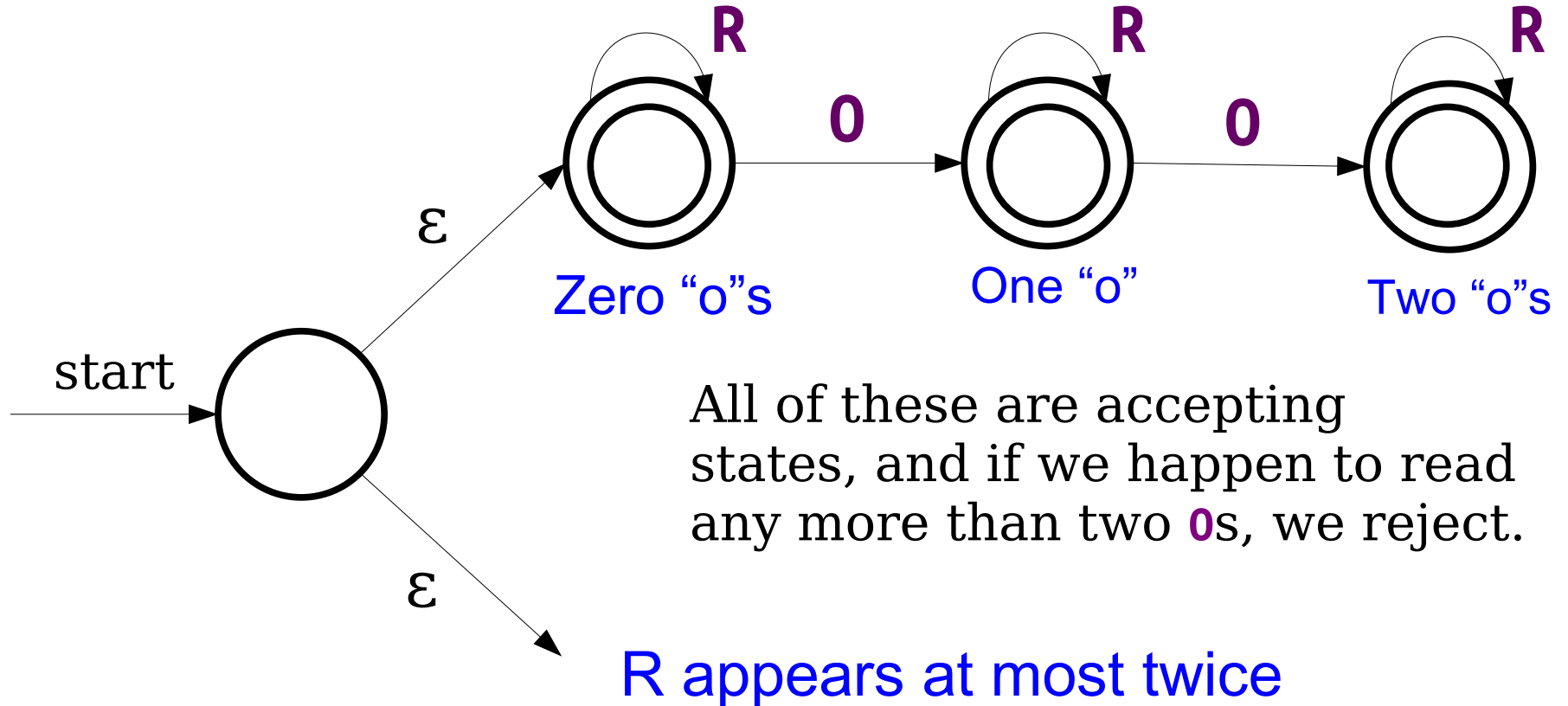
More Oreo Sandwiches

$L = \{ w \in \Sigma^* \mid \text{Some character of } \Sigma \text{ appears at most twice in } w \}$



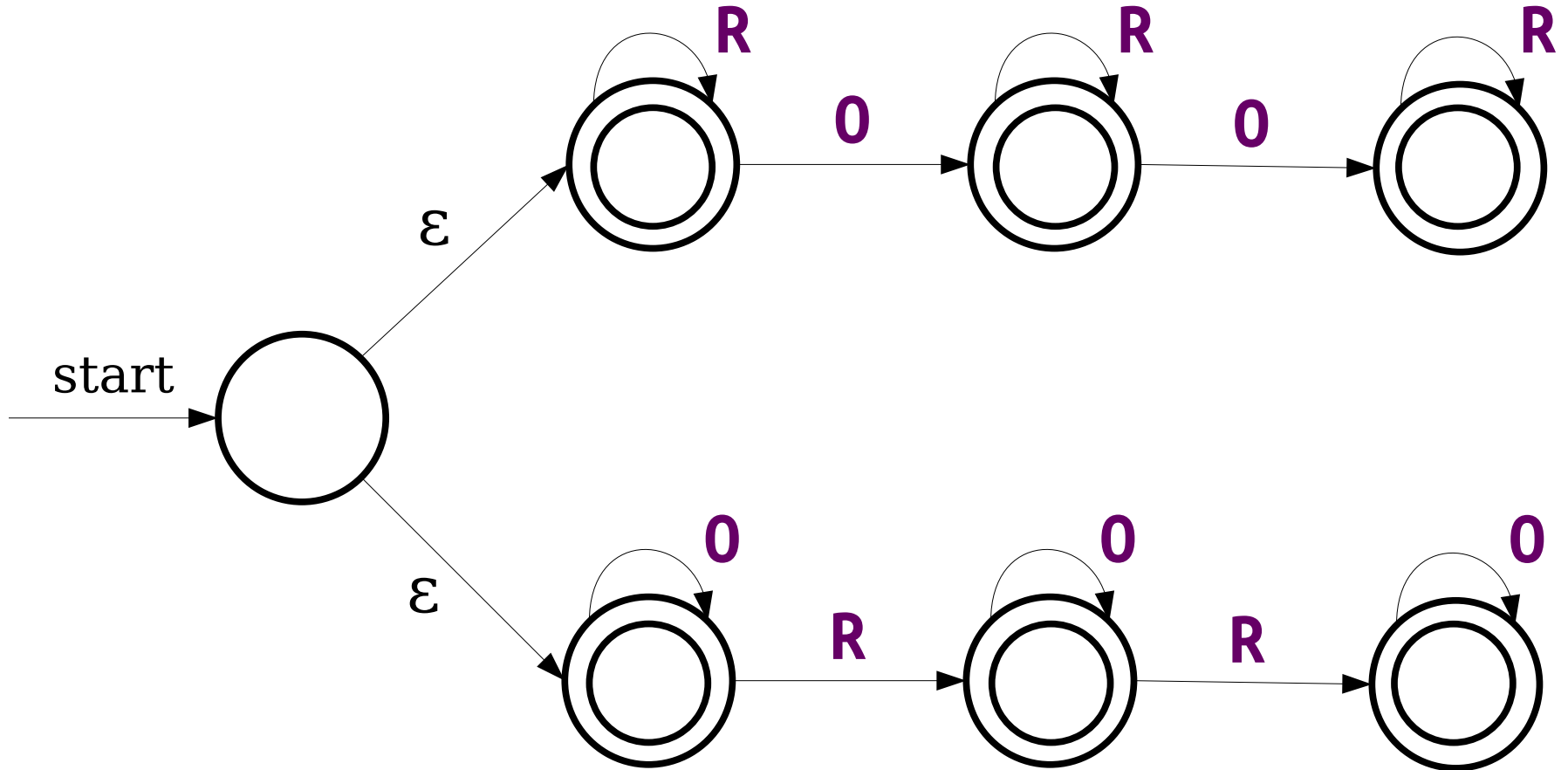
More Oreo Sandwiches

$L = \{ w \in \Sigma^* \mid \text{Some character of } \Sigma \text{ appears at most twice in } w \}$



More Oreo Sandwiches

$L = \{ w \in \Sigma^* \mid \text{Some character of } \Sigma \text{ appears at most twice in } w \}$



Next Time

- ***Applications of Regular Languages***
 - Answering “so what?”
- ***Intuiting Regular Languages***
 - What makes a language regular?
- ***The Myhill-Nerode Theorem***
 - The limits of regular languages.

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 $\mathcal{L}(R) = L$.

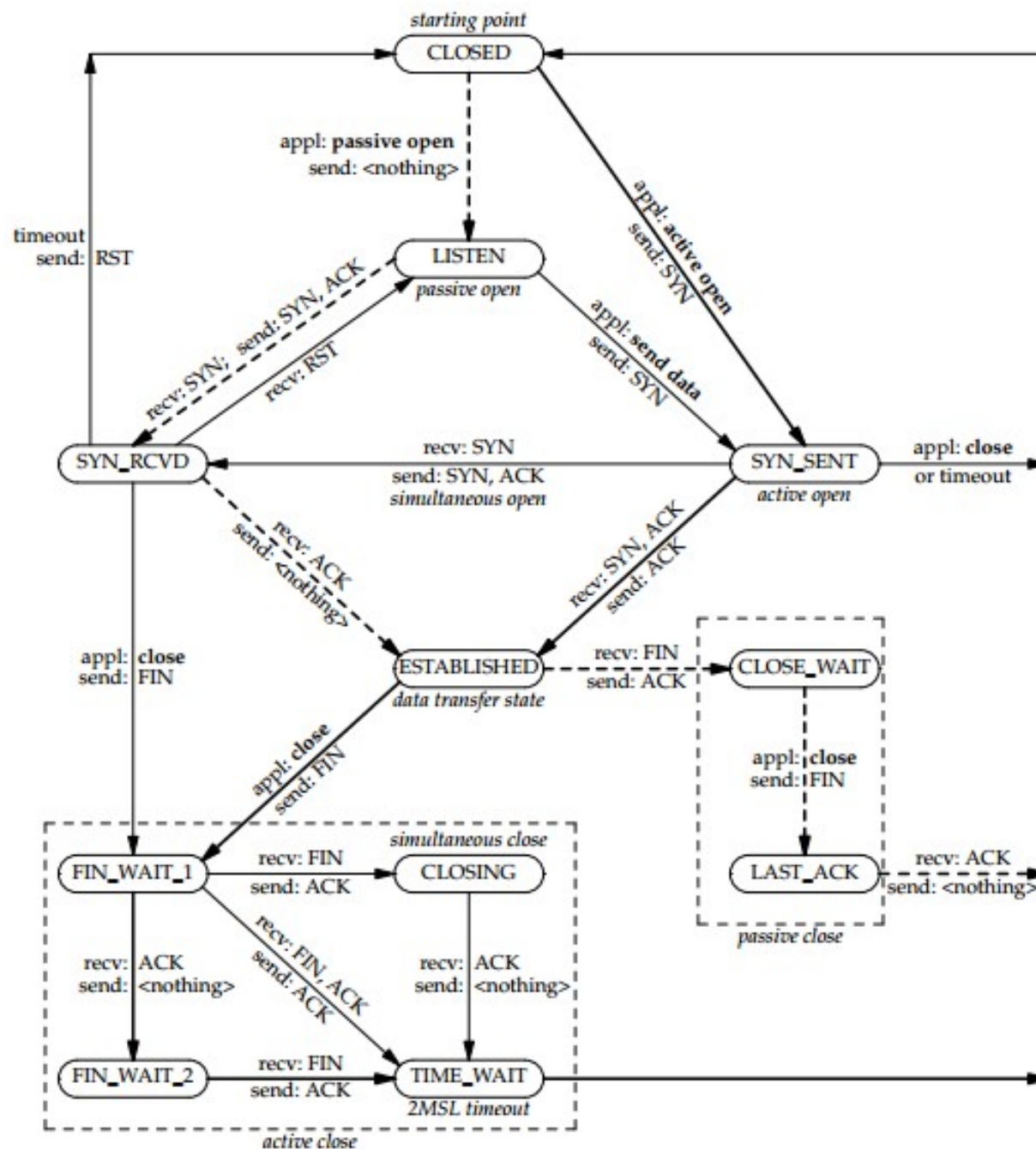
New Stuff!

Why does this matter?

Buttons as Finite-State Machines:

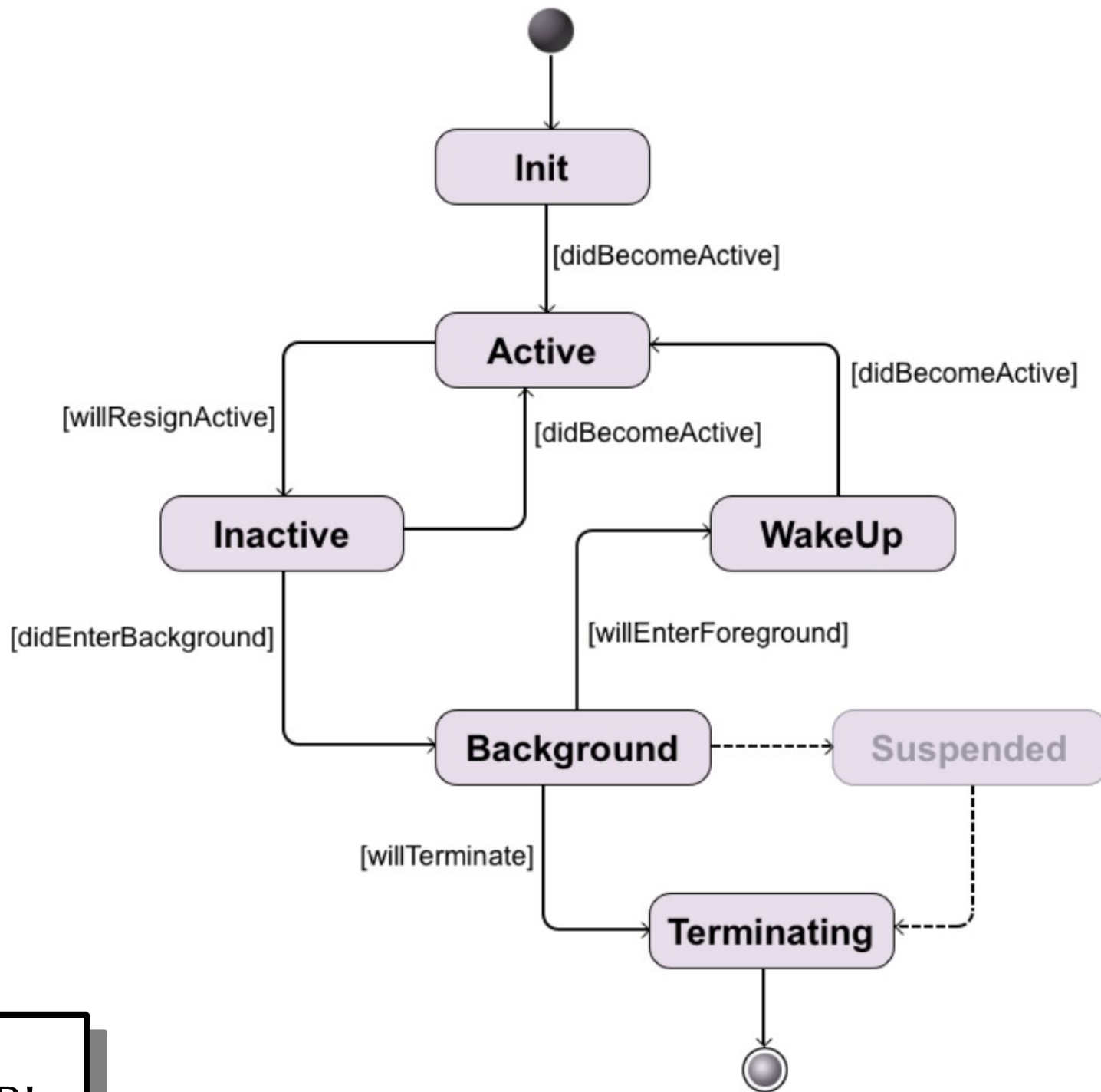
<http://cs103.stanford.edu/tools/button-fsm/>

Take
CS148!



—→ normal transitions for client
 - - -→ normal transitions for server
 appl: state transitions taken when application issues operation
 rcv: state transitions taken when segment received
 send: what is sent for this transition

Take
CS144!



Take
CS193P!

What exactly is a finite-state machine?

Ready!

Finite-Memory
Computing Device

a

b

c



Ready!

Finite-Memory
Computing Device

a

b

c



Working

Finite-Memory
Computing Device

a

b

c



Ready!

Finite-Memory
Computing Device

a

b

c



Ready!

Finite-Memory
Computing Device

a

b

c



Thinking

Finite-Memory
Computing Device

a

b

c



Ready!

Finite-Memory
Computing Device

a

b

c



Ready!

Finite-Memory
Computing Device

a

b

c



Thinking

Finite-Memory
Computing Device

a

b

c



Ready!

Finite-Memory
Computing Device

a

b

c



Ready!

Finite-Memory
Computing Device

a

b

c



Working

Finite-Memory
Computing Device

a

b

c



Ready!

Finite-Memory
Computing Device

a

b

c



Ready!

Finite-Memory
Computing Device

a

b

c



Working

Finite-Memory
Computing Device

a

b

c



YES

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

- An example 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 $\Sigma = \{\mathbf{a}, \mathbf{b}\}$ 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 $\mathbf{a}$'s followed by n $\mathbf{b}$'s:

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

A Simple Language

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

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

$\mathbf{a^*b^*}$

$\mathbf{(ab)^*}$

$\mathbf{\epsilon \cup ab \cup a^2b^2 \cup a^3b^3}$

Respond at pollev.com/zhenglian

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?

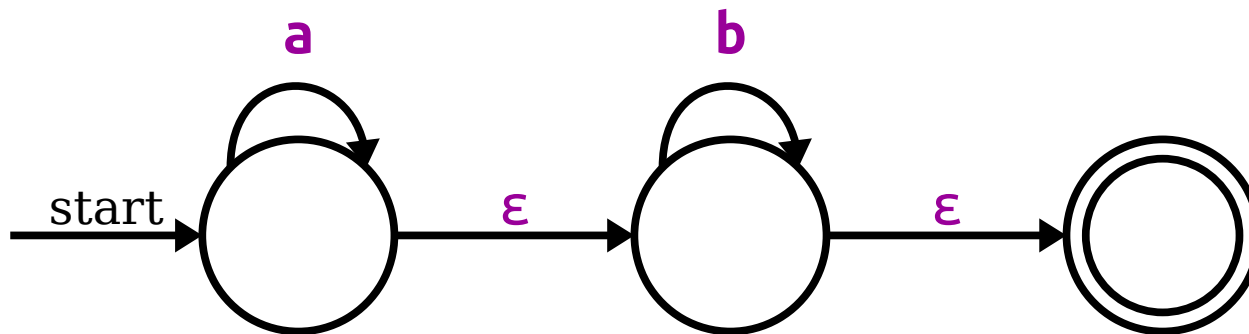


Another Attempt

- Let's try to design an NFA for

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

- How about this one?

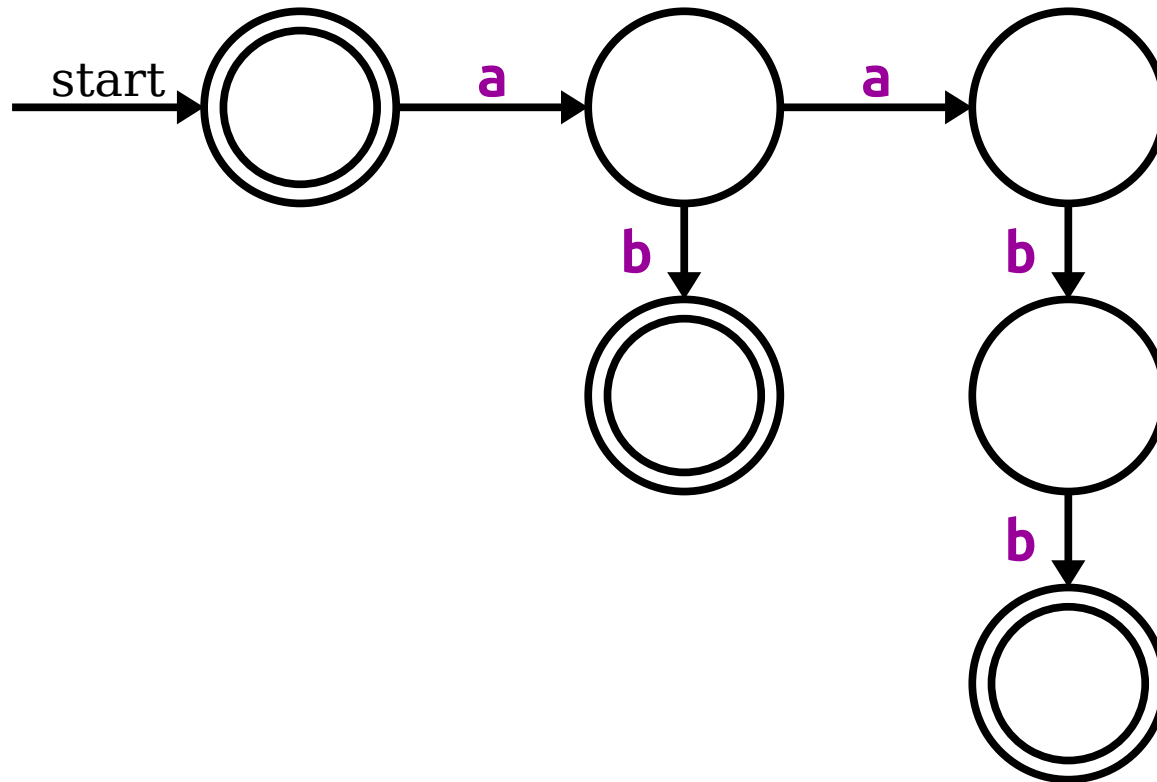


Another Attempt

- Let's try to design an NFA for

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

- What about this?

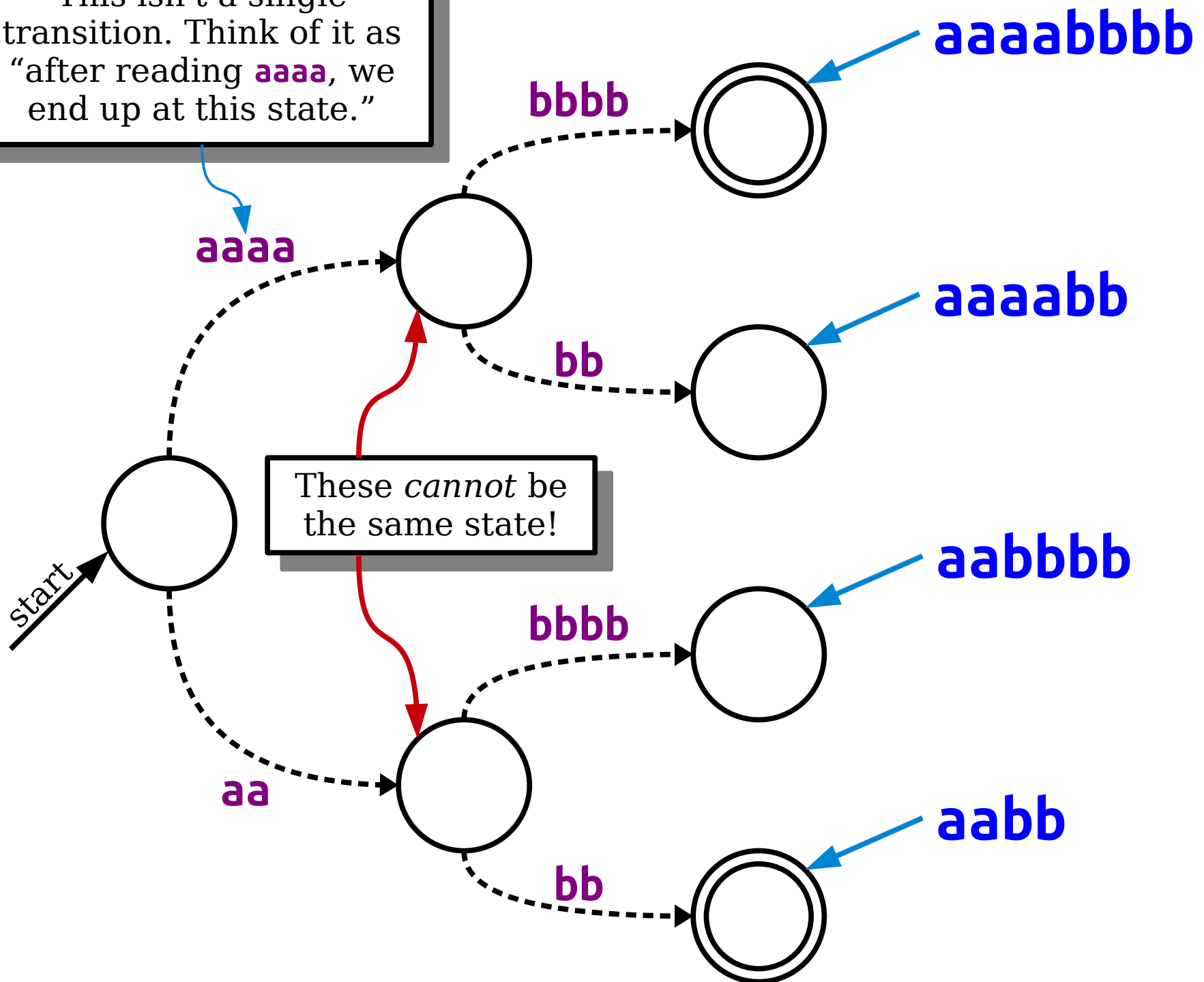


We seem to be running into some trouble.
Why is that?

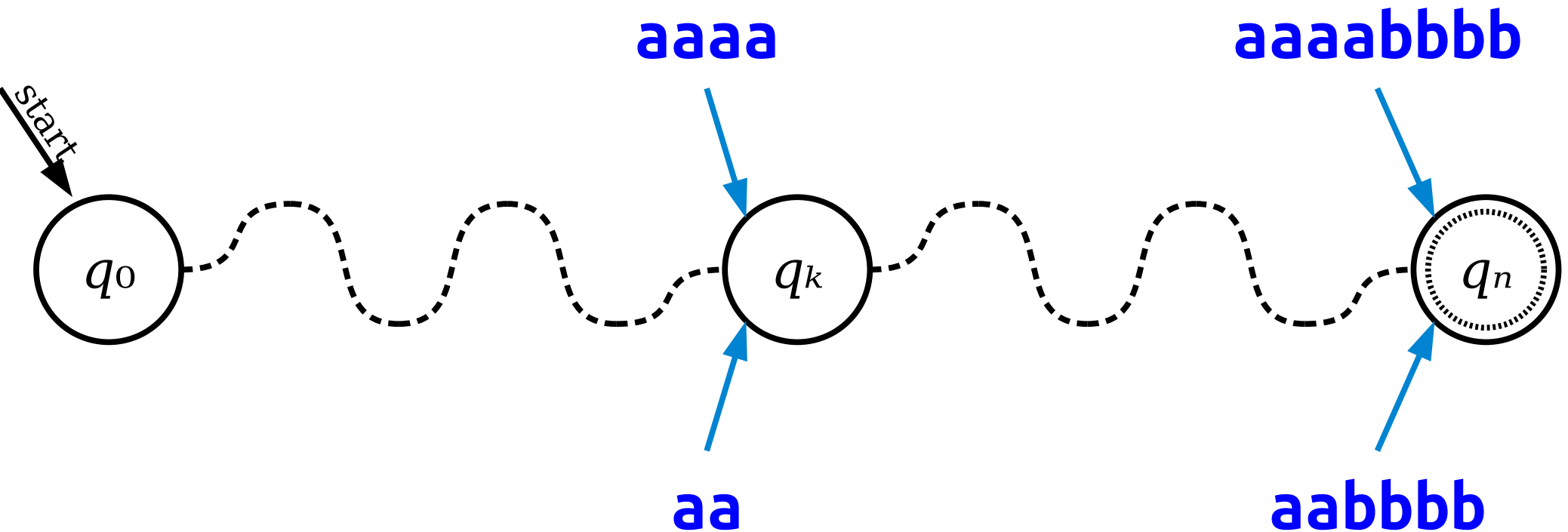
Let's imagine what a DFA for the language
 $\{ \mathbf{a}^n \mathbf{b}^n \mid n \in \mathbb{N} \}$ would have to look like.

Can we say anything about it?

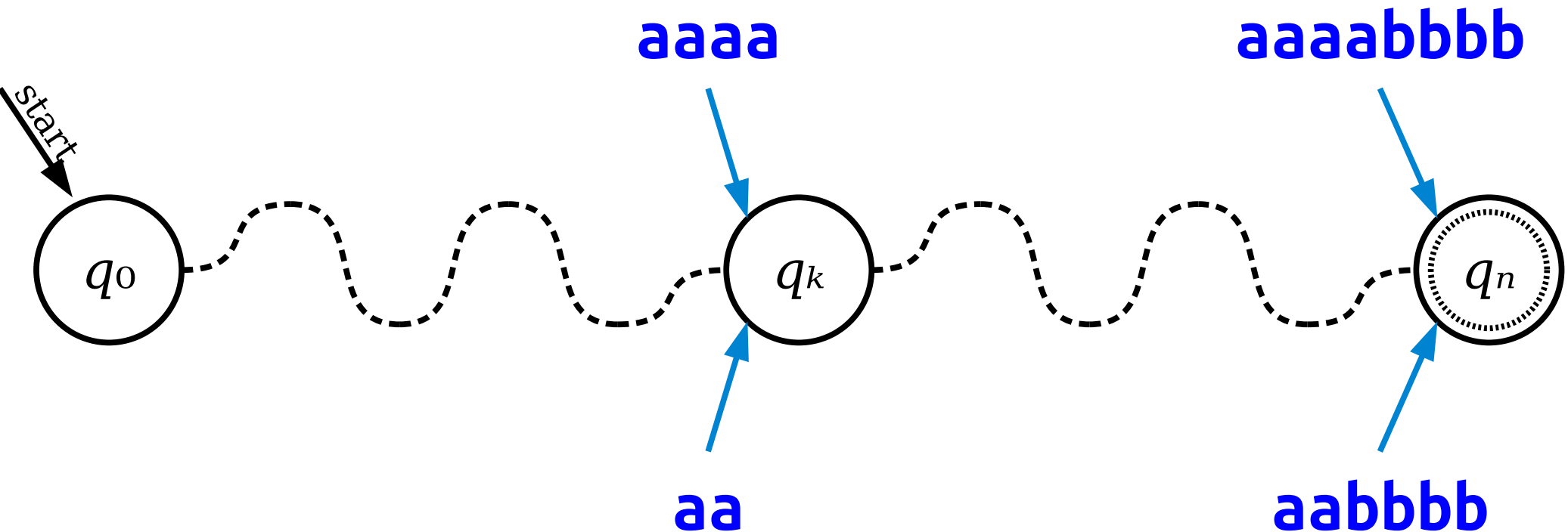
This isn't a single transition. Think of it as "after reading **aaaa**, we end up at this state."



A Different Perspective



A Different Perspective

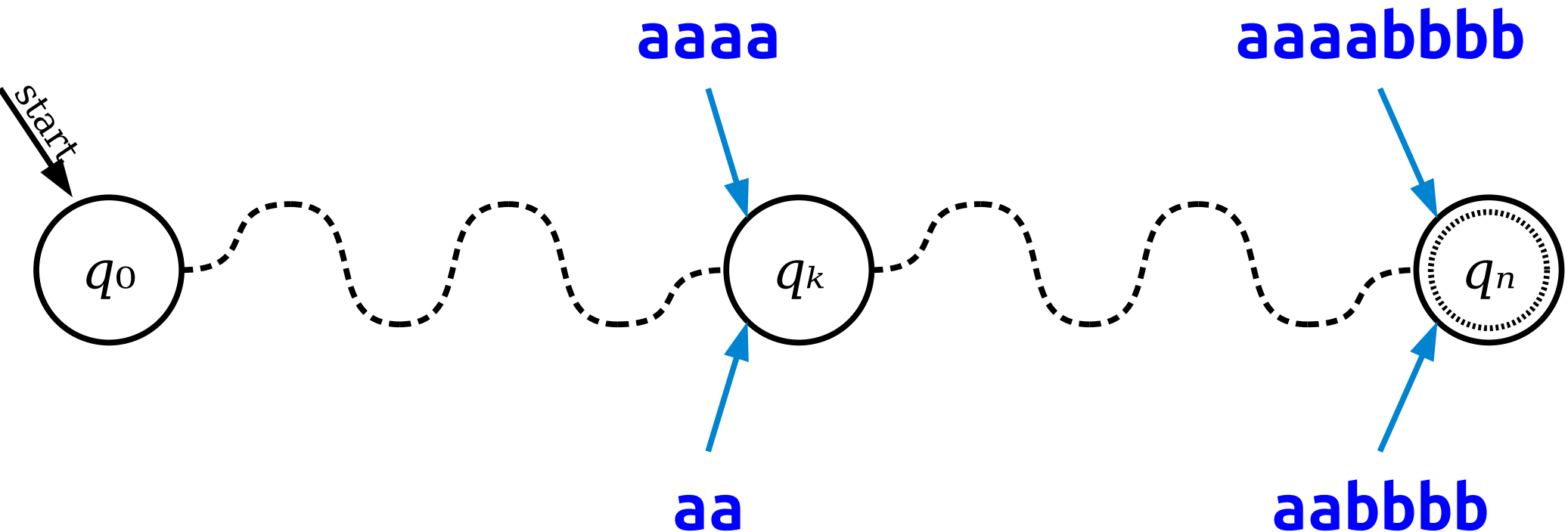


What happens if q_n is...

...an accepting state?

...a rejecting state?

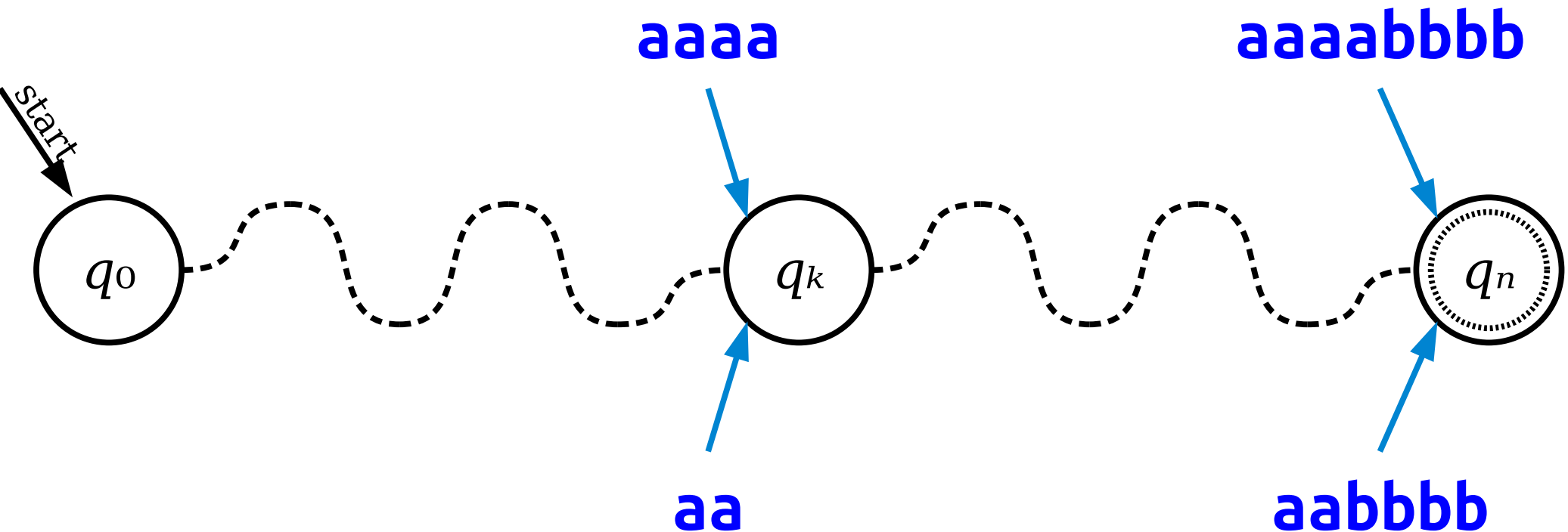
A Different Perspective



What happens if q_n is...

...an accepting state? We accept **aabbbb** $\notin E$!
...a rejecting 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 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 $x \not\equiv_L y$.
- 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

If L be an arbitrary language over Σ , and $x, y \in \Sigma^*$, we say that $x \not\equiv_L y$ if the following is true:

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

Let $L = \{ w \in \{a, b\}^* \mid |w| \text{ is a multiple of three} \}$.
How many of the following statements are true?

$$a \not\equiv_L aa$$

$$aaa \not\equiv_L aa$$

$$a \not\equiv_L aaaa$$

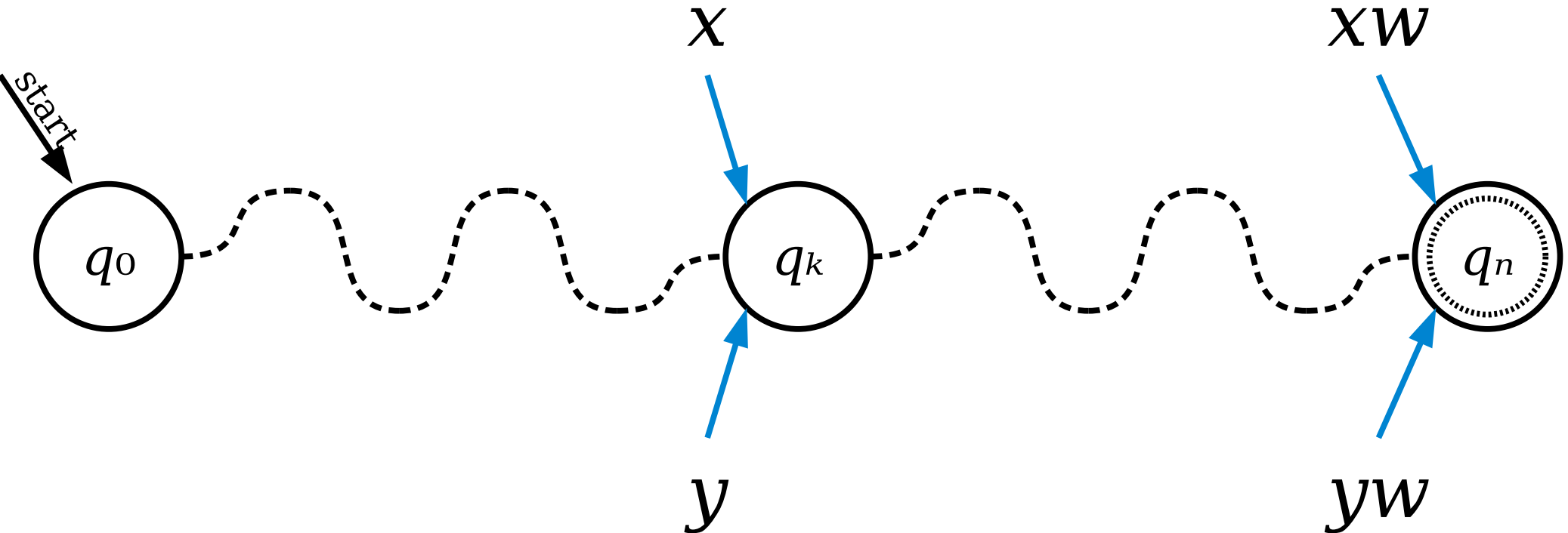
$$aa \not\equiv_L bb$$

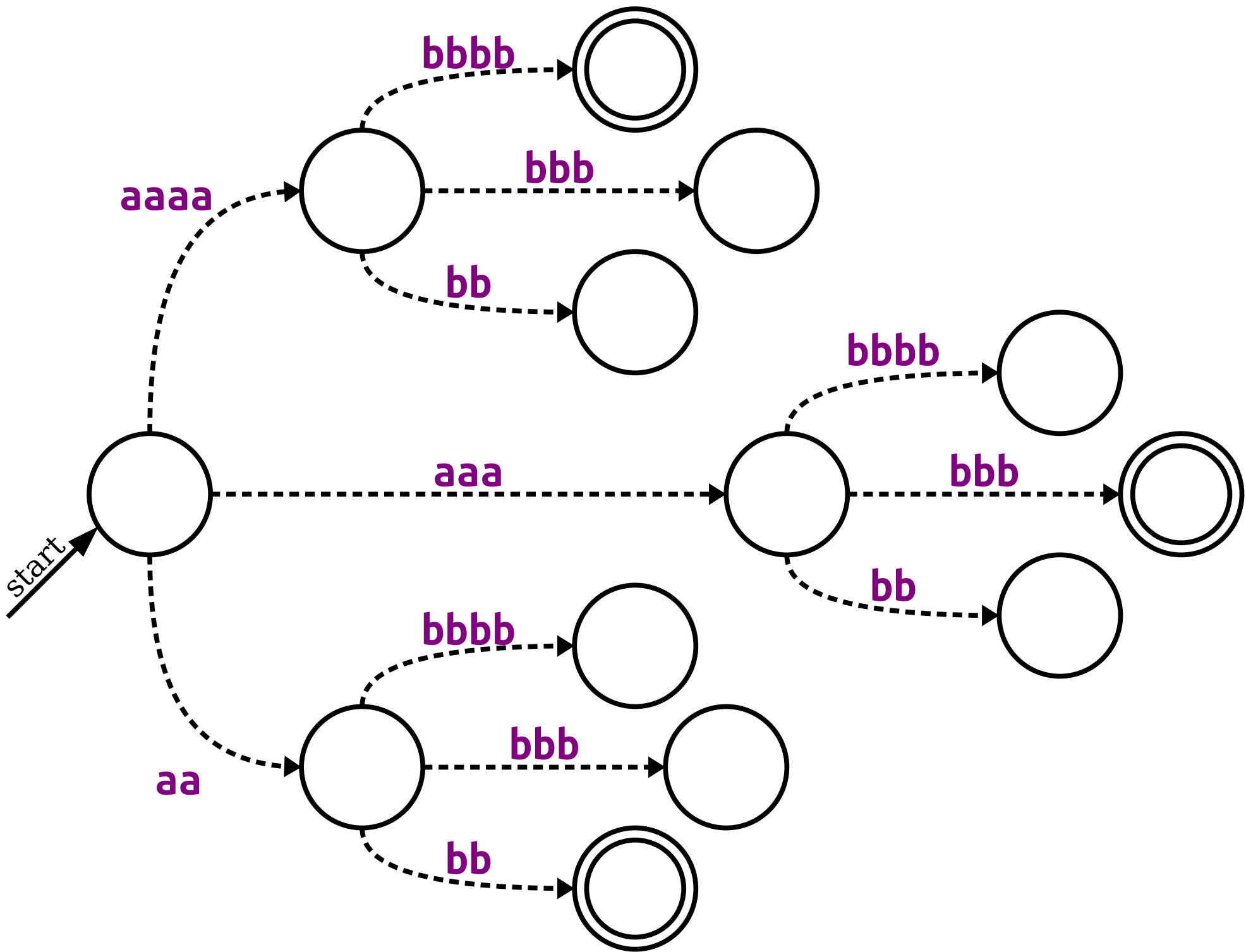
$$\varepsilon \not\equiv_L baba$$

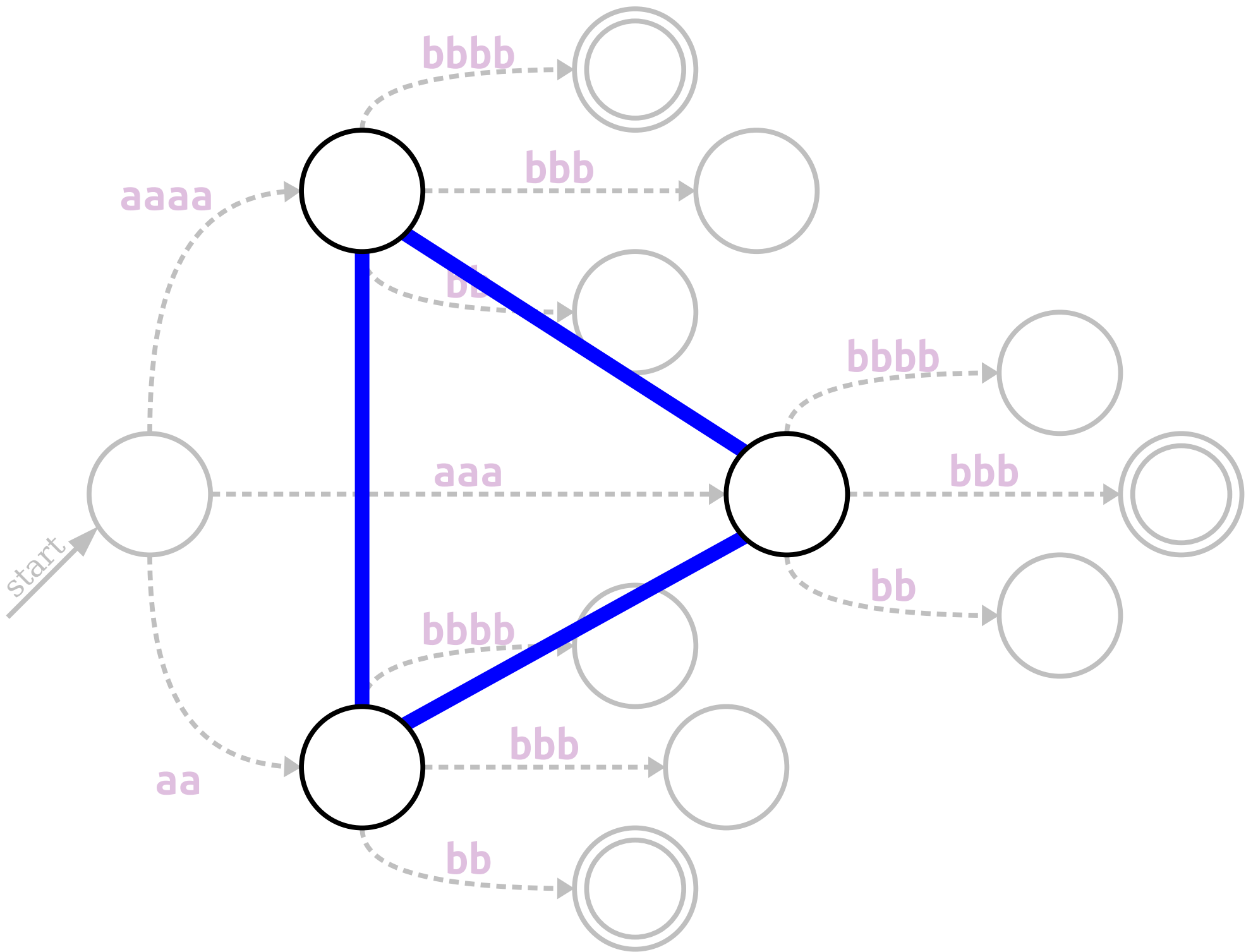
Respond at pollev.com/zhenglian

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:**







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 strings of the form $\mathbf{a}^m$.
 - 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 $E = \{ \mathbf{a^n b^n} \mid n \in \mathbb{N} \}$ is not regular.

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

Proof: Suppose for the sake of contradiction that E is regular.

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 .

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 $\mathbf{a}^0, \mathbf{a}^1, \mathbf{a}^2, \dots, \mathbf{a}^k$.

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 .

Theorem: The language $E = \{ a^n b^n \mid n \in \mathbb{N} \}$ 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 $a^0, a^1, a^2, \dots, a^k$. 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 a^m and a^n that end in the same state when run through D .

Theorem: The language $E = \{ a^n b^n \mid n \in \mathbb{N} \}$ 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 $a^0, a^1, a^2, \dots, a^k$. 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 a^m and a^n that end in the same state when run through D .

Our lemma tells us that $a^m \not\equiv_E a^n$, so by our earlier theorem we know that a^m and a^n cannot end in the same state when run through D .

Theorem: The language $E = \{ a^n b^n \mid n \in \mathbb{N} \}$ 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 $a^0, a^1, a^2, \dots, a^k$. 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 a^m and a^n that end in the same state when run through D .

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

We have reached a contradiction, so our assumption must have been wrong.

Theorem: The language $E = \{ a^n b^n \mid n \in \mathbb{N} \}$ 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 $a^0, a^1, a^2, \dots, a^k$. 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 a^m and a^n that end in the same state when run through D .

Our lemma tells us that $a^m \not\equiv_E a^n$, so by our earlier theorem we know that a^m and a^n cannot end in the same state when run through D . But this is impossible, since we know that a^m and 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 $E = \{ a^n b^n \mid n \in \mathbb{N} \}$ 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 $a^0, a^1, a^2, \dots, a^k$. 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 a^m and a^n that end in the same state when run through D .

Our lemma tells us that $a^m \not\equiv_E a^n$, so by our earlier theorem we know that a^m and a^n cannot end in the same state when run through D . But this is impossible, since we know that a^m and 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 $E = \{ a^n b^n \mid n \in \mathbb{N} \}$ 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 $a^0, a^1, a^2, \dots, a^k$. 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 a^m and a^n that end in the same state when run through D .

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

- We can think of things this way because

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

- 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

If L be an arbitrary language over Σ , and $x, y \in \Sigma^*$, we say that $x \not\equiv_L y$ if the following is true:

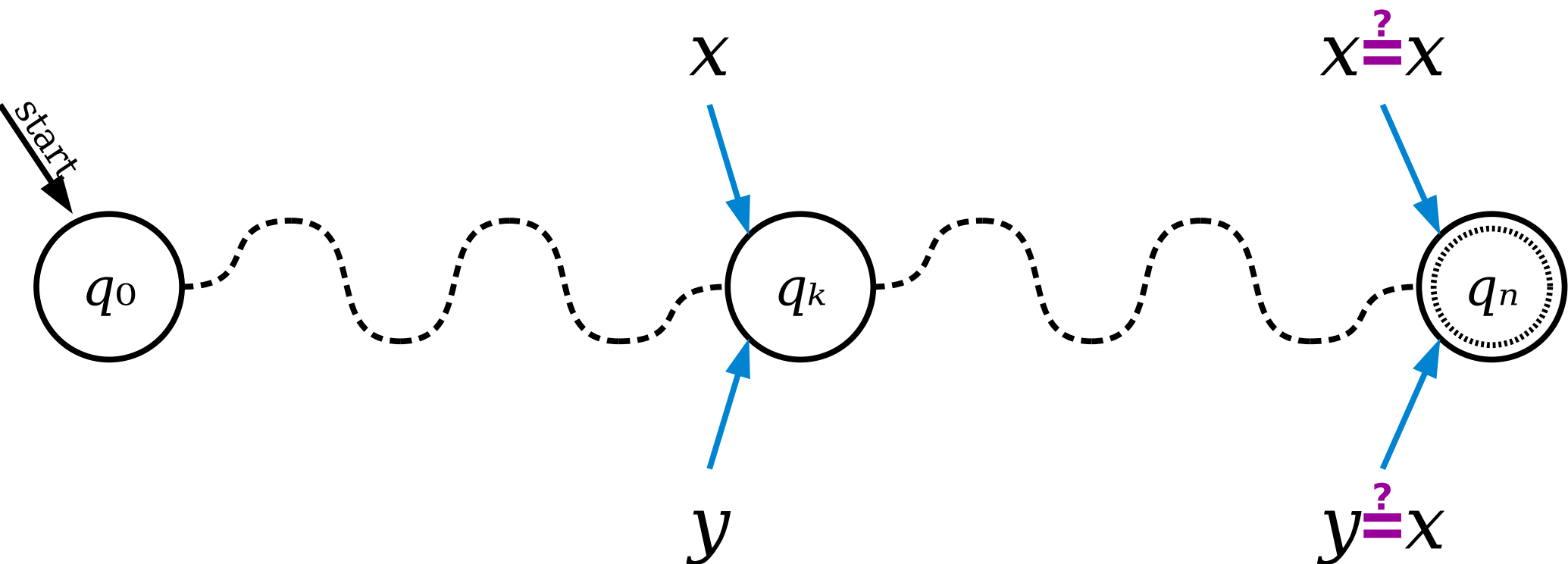
$$\exists w \in \Sigma^*. (xw \in L \leftrightarrow yw \notin L)$$

For the language $EQ = \{ w^?w \mid w \in \{a, b\}^* \}$, how many of the following statements are true?

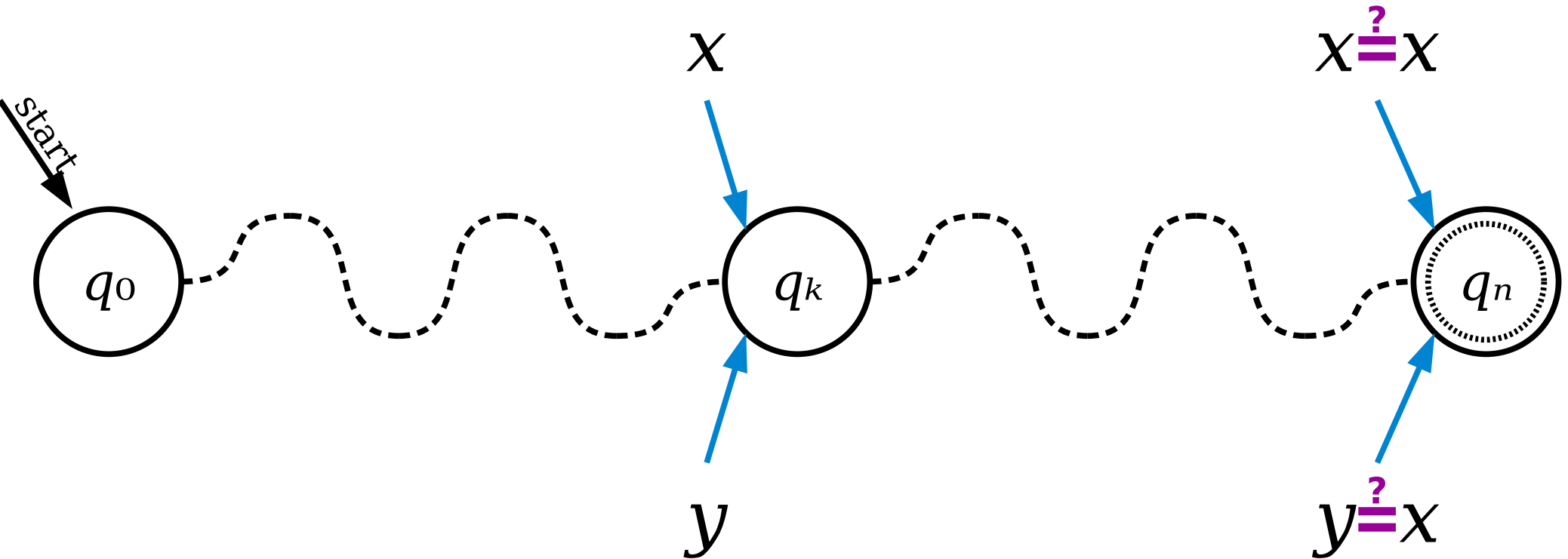
$$\begin{aligned} a &\not\equiv_{EQ} b \\ abb &\not\equiv_{EQ} abb \\ \varepsilon &\not\equiv_{EQ} ab \\ ?? &\not\equiv_{EQ} ? \\ ??? &\not\equiv_{EQ} ?? \end{aligned}$$

Respond at pollev.com/zhenglian

The Intuition



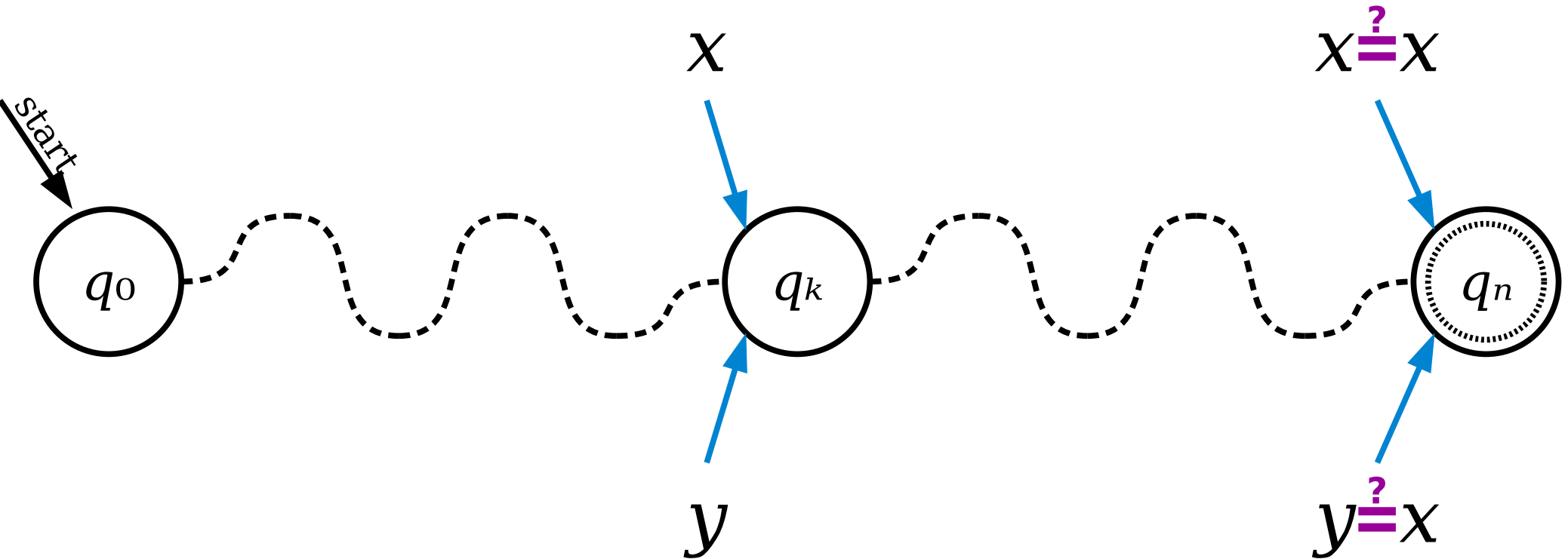
The Intuition



What happens if q_n is...

- ...an accepting state?
- ...a rejecting state?

The Intuition

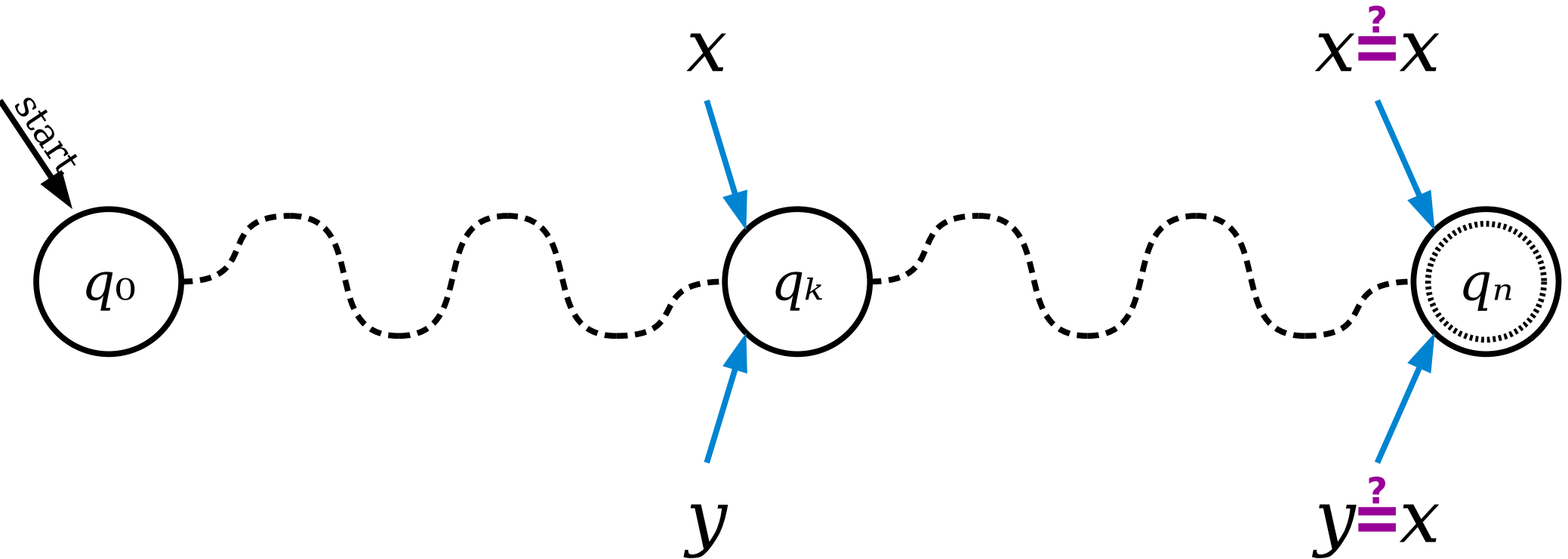


What happens if q_n is...

...an accepting state?
...a rejecting state?

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

The Intuition



What happens if q_n is...

...an accepting state?

...a rejecting state?

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

We reject $x \stackrel{?}{=} x \in EQ!$

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 $x \not\equiv_{EQ} y$.

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 $x \not\equiv_{EQ} y$, as required. ■

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

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

Proof:

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

Proof: Suppose for the sake of contradiction that EQ is regular.

Theorem: The language $EQ = \{ w \stackrel{?}{=} w \mid w \in \{a, b\}^* \}$ 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 .

Theorem: The language $EQ = \{ w \stackrel{?}{=} w \mid w \in \{\mathbf{a}, \mathbf{b}\}^* \}$ 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 $\{\mathbf{a}, \mathbf{b}\}^*$.

Theorem: The language $EQ = \{ w \stackrel{?}{=} w \mid w \in \{a, b\}^* \}$ 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.

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

Theorem: The language $EQ = \{ w \stackrel{?}{=} w \mid w \in \{a, b\}^* \}$ 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 .

Theorem: The language $EQ = \{ w \stackrel{?}{=} w \mid w \in \{a, b\}^* \}$ 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 we specifically chose x and y to end in the same state when run through D .

Theorem: The language $EQ = \{ w \stackrel{?}{=} w \mid w \in \{a, b\}^* \}$ 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 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.

Theorem: The language $EQ = \{ w \stackrel{?}{=} w \mid w \in \{a, b\}^* \}$ 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 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.

Theorem: The language $EQ = \{ w \stackrel{?}{=} w \mid w \in \{a, b\}^* \}$ 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 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. ■

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 we specifically chose x and y to end in the same state when run through D .

We have reached a contradiction. Our assumption that EQ is regular has been wrong. Therefore, EQ 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.

Time-Out for Announcements!

Problem Sets

- Problem Set 4 solutions are up on the course website – please take a look at them as soon as possible.
- Problem Set 5 will be due this Friday at 5:30 PM.
- TAs are hard at work grading your submissions for Problem Set 4. We're aiming to have those back to you by tomorrow.

How to Improve

- We posted it guide titled “How to Improve” on the course website under the “COURSE” column.
- Consider checking it out if you are looking for tips on how to approach the latter half of this quarter!

Back to CS103!

Comparing Proofs

Theorem: The language $E = \{ \mathbf{a}^n \mathbf{b}^n \mid n \in \mathbb{N} \}$ 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 $\mathbf{a}^0, \mathbf{a}^1, \mathbf{a}^2, \dots, \mathbf{a}^k$. 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 $\mathbf{a}^m \not\equiv_E \mathbf{a}^n$. 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. ■

Theorem: The language $EQ = \{ w^?w \mid w \in \{a, b\}^* \}$ 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 $\{a, b\}^*$. 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 $L = [\text{fill in the blank}]$ 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. ■

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. ■

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

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...

Proof: Suppose for the sake of contradiction that L is a regular language. 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, so our assumption must have been wrong. Therefore, L is not regular. ■

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

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...

Proof: Suppose for the sake of contradiction that L is not regular.
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, so our assumption must have been wrong. Therefore, L is not regular.

... 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:

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

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

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

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 $E = \{ \mathbf{a}^n \mathbf{b}^n \mid n \in \mathbb{N} \}$ is not regular.

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

Proof:

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

Proof: Let $S = \{ \mathbf{a}^n \mid n \in \mathbb{N} \}$.

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

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

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

Proof: Let $S = \{ \mathbf{a}^n \mid n \in \mathbb{N} \}$. 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.

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

Proof: Let $S = \{ \mathbf{a}^n \mid n \in \mathbb{N} \}$. 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 $\mathbf{a}^m, \mathbf{a}^n \in S$ where $m \neq n$.

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

Proof: Let $S = \{ \mathbf{a}^n \mid n \in \mathbb{N} \}$. 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 $\mathbf{a}^m, \mathbf{a}^n \in S$ where $m \neq n$. Note that $\mathbf{a}^m \mathbf{b}^m \in E$ and that $\mathbf{a}^n \mathbf{b}^m \notin E$.

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

Proof: Let $S = \{ \mathbf{a}^n \mid n \in \mathbb{N} \}$. 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 $\mathbf{a}^m, \mathbf{a}^n \in S$ where $m \neq n$. Note that $\mathbf{a}^m \mathbf{b}^m \in E$ and that $\mathbf{a}^n \mathbf{b}^m \notin E$. Therefore, we see that $\mathbf{a}^m \not\equiv_E \mathbf{a}^n$, as required.

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

Proof: Let $S = \{ \mathbf{a}^n \mid n \in \mathbb{N} \}$. 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 $\mathbf{a}^m, \mathbf{a}^n \in S$ where $m \neq n$. Note that $\mathbf{a}^m \mathbf{b}^m \in E$ and that $\mathbf{a}^n \mathbf{b}^m \notin E$. Therefore, we see that $\mathbf{a}^m \not\equiv_E \mathbf{a}^n$, 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 $E = \{ \mathbf{a}^n \mathbf{b}^n \mid n \in \mathbb{N} \}$ is not regular.

Proof: Let $S = \{ \mathbf{a}^n \mid n \in \mathbb{N} \}$. 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 $\mathbf{a}^m, \mathbf{a}^n \in S$ where $m \neq n$. Note that $\mathbf{a}^m \mathbf{b}^m \in E$ and that $\mathbf{a}^n \mathbf{b}^m \notin E$. Therefore, we see that $\mathbf{a}^m \not\equiv_E \mathbf{a}^n$, 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 $EQ = \{ w \stackrel{?}{=} w \mid w \in \{a, b\}^* \}$ is not regular.

The Myhill Nerode theorem asks for a set $S \subseteq \{a, b, \stackrel{?}{=}\}^*$ where S is infinite and

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

Which of these sets meets these criteria?

A) $S = \{a, b, \stackrel{?}{=}\}^*$

B) $S = \{a, b\}^*$

C) $S = \{a \stackrel{?}{=}\}^*$

D) $S = \{a\}^*$

E) None of these, or two or more of these.

Respond at pollev.com/zhengliao

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

Proof:

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

Proof: Let $S = \{a, b\}^*$.

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

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

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

Proof: Let $S = \{a, b\}^*$. 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 $a^n \in S$.

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

Proof: Let $S = \{a, b\}^*$. 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 $a^n \in S$. Therefore, S contains at least one string for each natural number, so S is infinite.

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

Proof: Let $S = \{a, b\}^*$. 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 $a^n \in S$. 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 $x \neq y$.

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

Proof: Let $S = \{a, b\}^*$. 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 $a^n \in S$. 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 $y \stackrel{?}{=} x \notin EQ$.

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

Proof: Let $S = \{a, b\}^*$. 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 $a^n \in S$. 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.

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

Proof: Let $S = \{a, b\}^*$. 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 $a^n \in S$. 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.

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

Proof: Let $S = \{a, b\}^*$. 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 $a^n \in S$. 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 and 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

- ***Turing Machines***
 - What does a computer with unbounded memory look like?
 - How would you program it?