The Limits of Regular Languages

Announcements

- Midterm **tomorrow night** in Hewlett 200/201, 7PM – 10PM.
  - Open-book, open-note, open-computer, closed-network.
  - Covers material up to and including DFAs.

Regular Expressions

The Regular Expressions

- Goal: Assemble all regular languages from smaller building blocks!
- Atomic regular expressions:

$$\emptyset \qquad \varepsilon \qquad \mathtt{a}$$

- Compound regular expressions:

$$R_1R_2 \qquad R_1 \mid R_2 \qquad R^* \qquad (R)$$

Operator Precedence

- Regular expression operator precedence:

$$(R)$$

$$R^*$$

$$R_1R_2$$

$$R_1 \mid R_2$$

- `ab*c|d` is parsed as `((a(b*))c)|d`

Regular Expressions are Awesome

- Let Σ = { `a`, `.`, `@` }, where `a` represents "some letter."
- Regular expression for email addresses:

aa* (.aa*)* @ aa*.aa* (.aa*)*

cs103@cs.stanford.edu
first.middle.last@mail.site.org
barack.obama@whitehouse.gov

Regular Expressions are Awesome

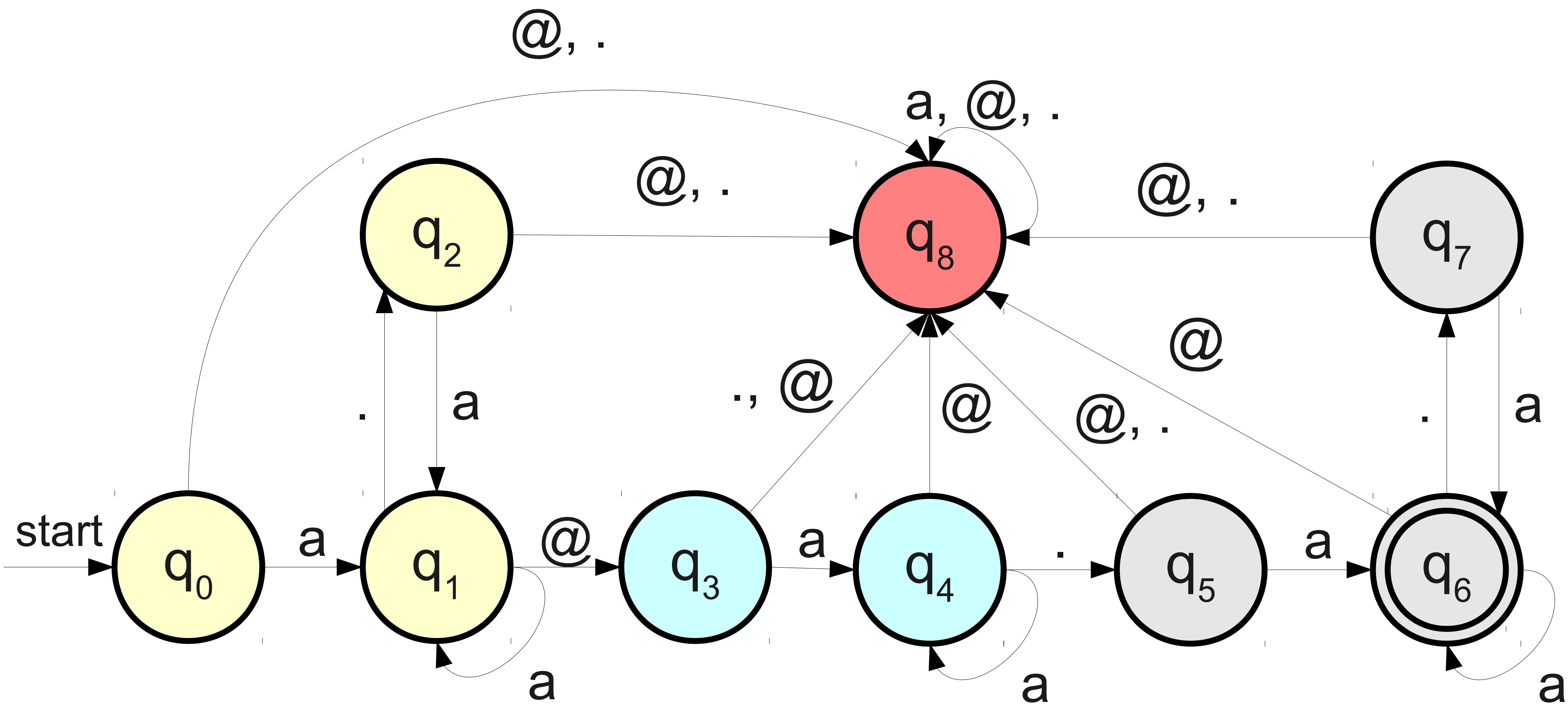
- Let Σ = { `a`, `.`, `@` }, where `a` represents "some letter."
- Regular expression for email addresses:

$$a^+(.a^+)^*@a^+(.a^+)^+$$

cs103@cs.stanford.edu
first.middle.last@mail.site.org
barack.obama@whitehouse.gov

Regular Expressions are Awesome

$$a^+(.a^+)^*@a^+(.a^+)^+$$



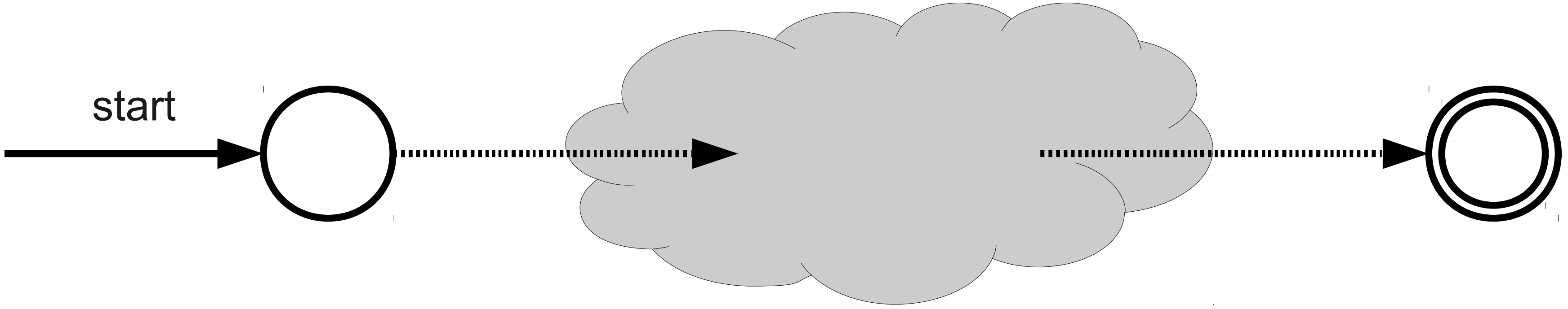
The Power of Regular Expressions

Theorem: If R is a regular expression, then $\mathscr{L}(R)$ is regular.

Proof idea: Induction over the structure of regular expressions. Atomic regular expressions are the base cases, and the inductive step handles each way of combining regular expressions.

A Marvelous Construction

- To show that any language described by a regular expression is regular, we show how to convert a regular expression into an NFA.
- *Theorem:* For any regular expression R, there is an NFA N such that
  - $\mathscr{L}(R) = \mathscr{L}(N)$
  - N has exactly one accepting state.
  - N has no transitions into its start state.
  - N has no transitions out of its accepting state.



A Marvelous Construction

To show that any language c
expression is regular, we sh
regular expression into an N

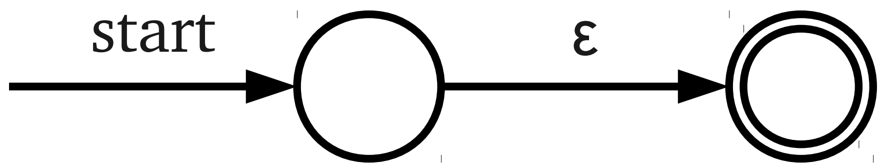
Theorem: For any regular e
NFA N such that

$$\mathscr{L}(R) = \mathscr{L}(N)$$

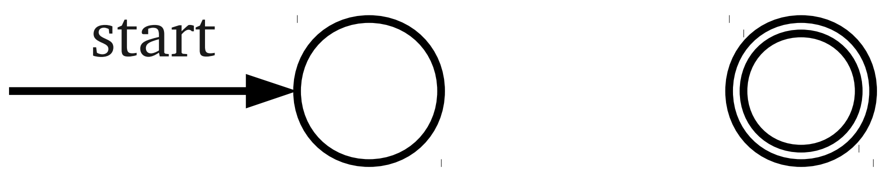
These are stronger requirements than are necessary for a normal NFA. We enforce these rules to simplify the construction.

- N has exactly one accepting state.
- N has no transitions into its start state.
- N has no transitions out of its accepting state.

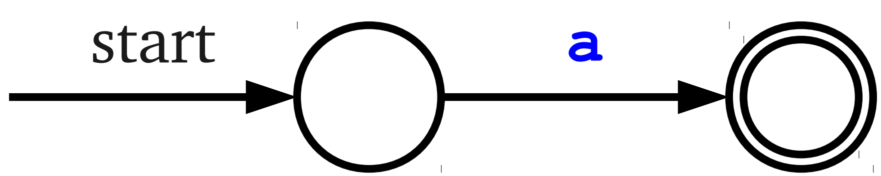
Base Cases



Automaton for ε

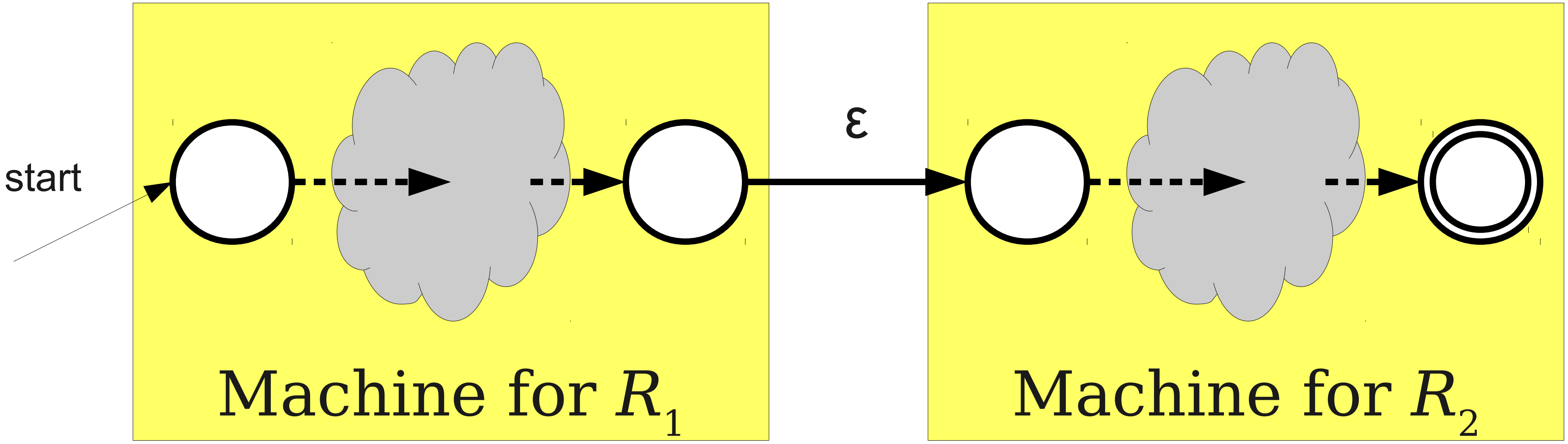


Automaton for Ø

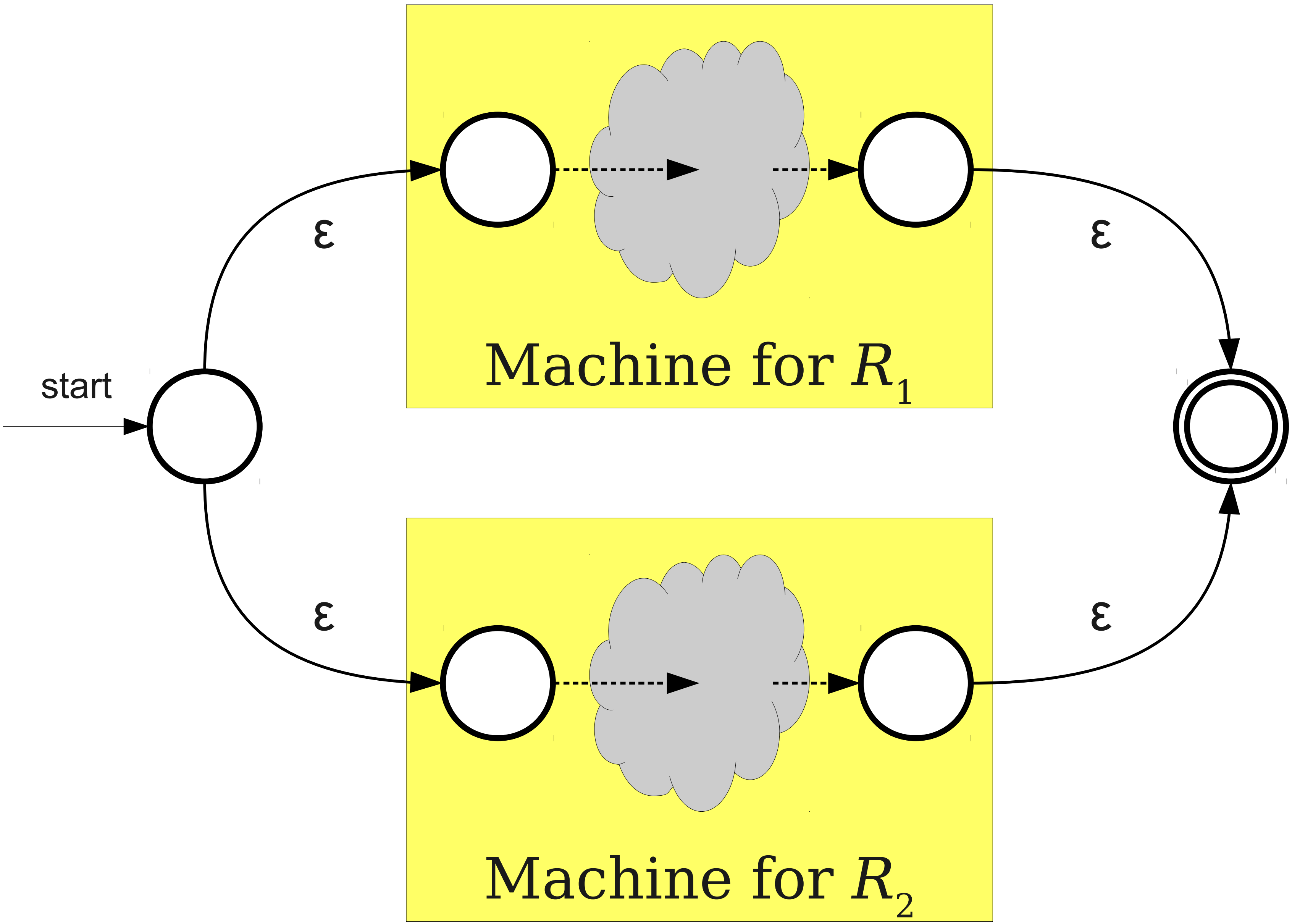


Automaton for single character **a**

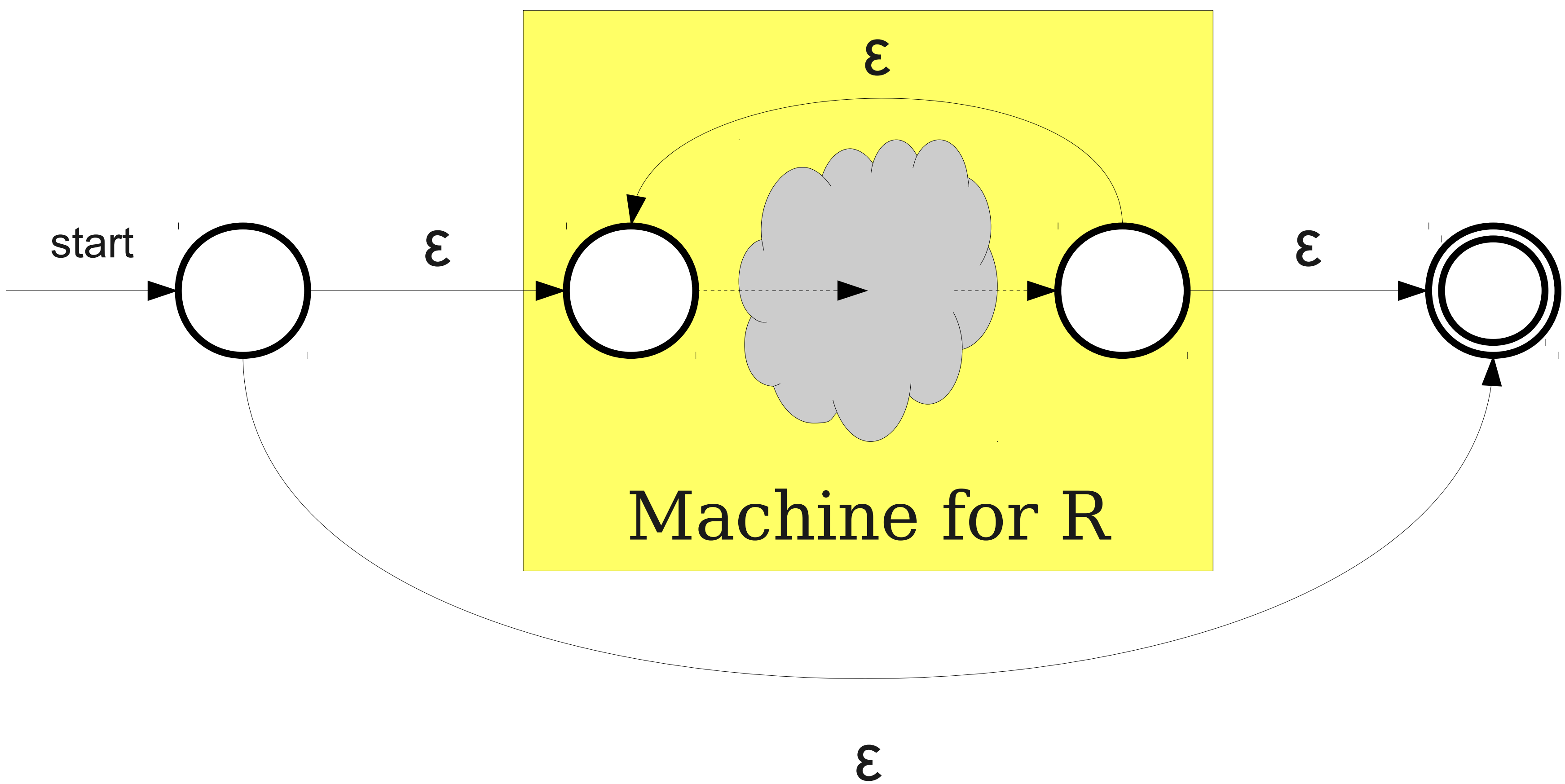
Construction for R_1R_2



Construction for $R_1 \mid R_2$



Construction for R*



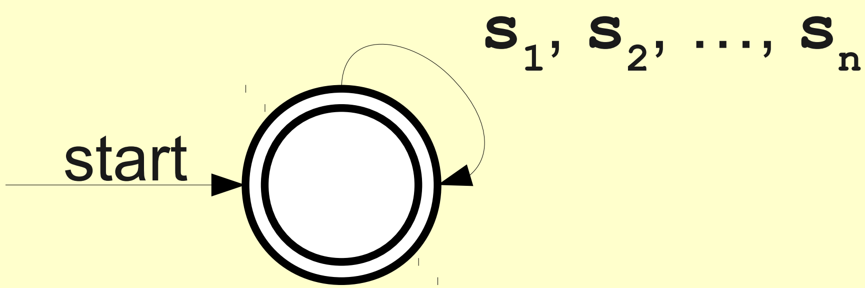
The Power of Regular Expressions

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

This is not obvious!

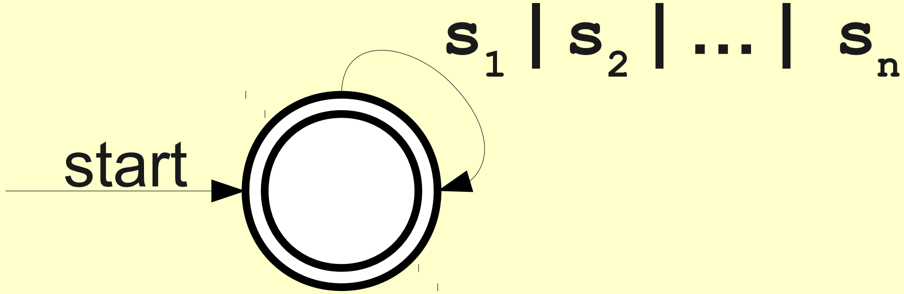
Proof idea: Show how to convert an arbitrary NFA into a regular expression.

From NFAs to Regular Expressions



Regular expression: $(s_1 \mid s_2 \mid \ldots \mid s_n)^*$

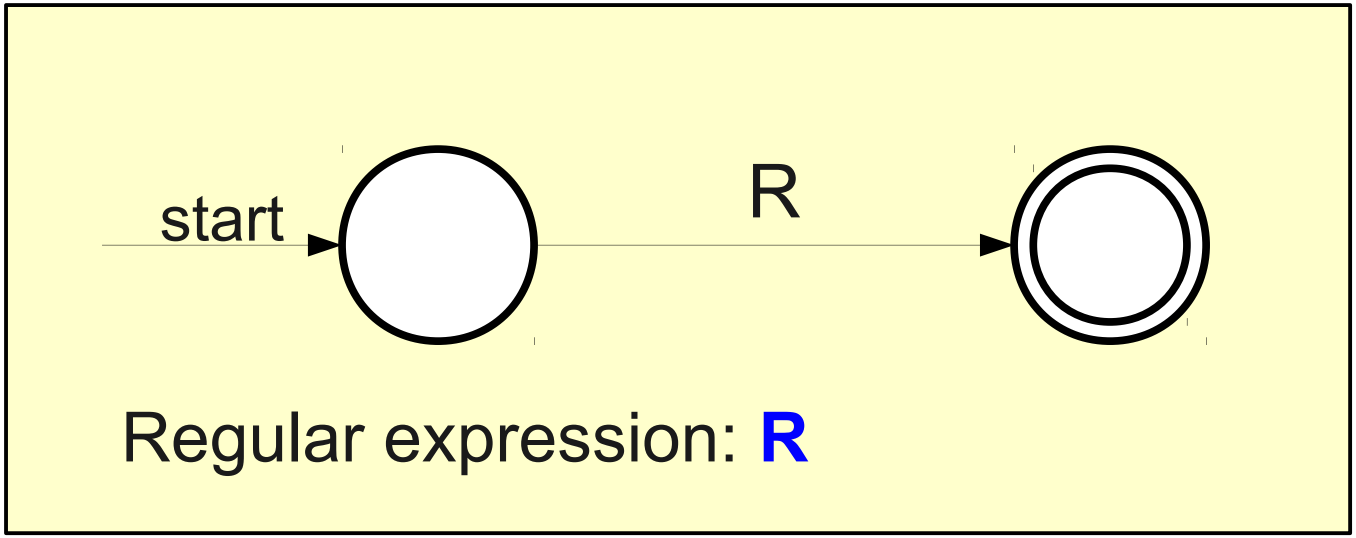
From NFAs to Regular Expressions



Regular expression: $(s_1 \mid s_2 \mid \ldots \mid s_n)^*$

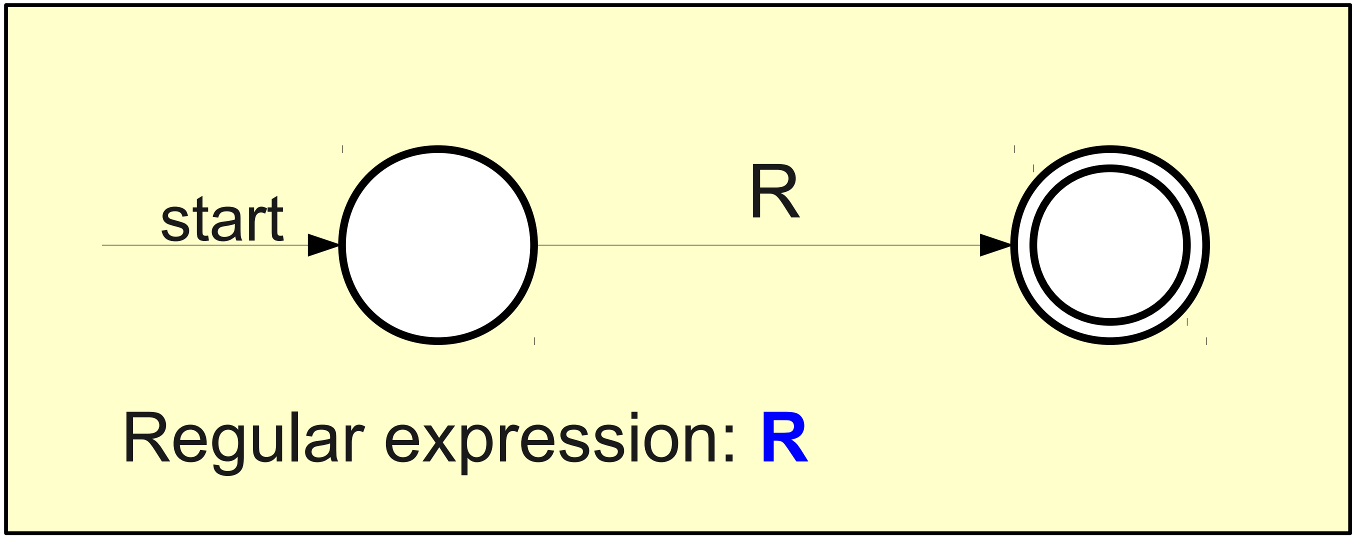
Key idea: Label transitions with arbitrary regular expressions.

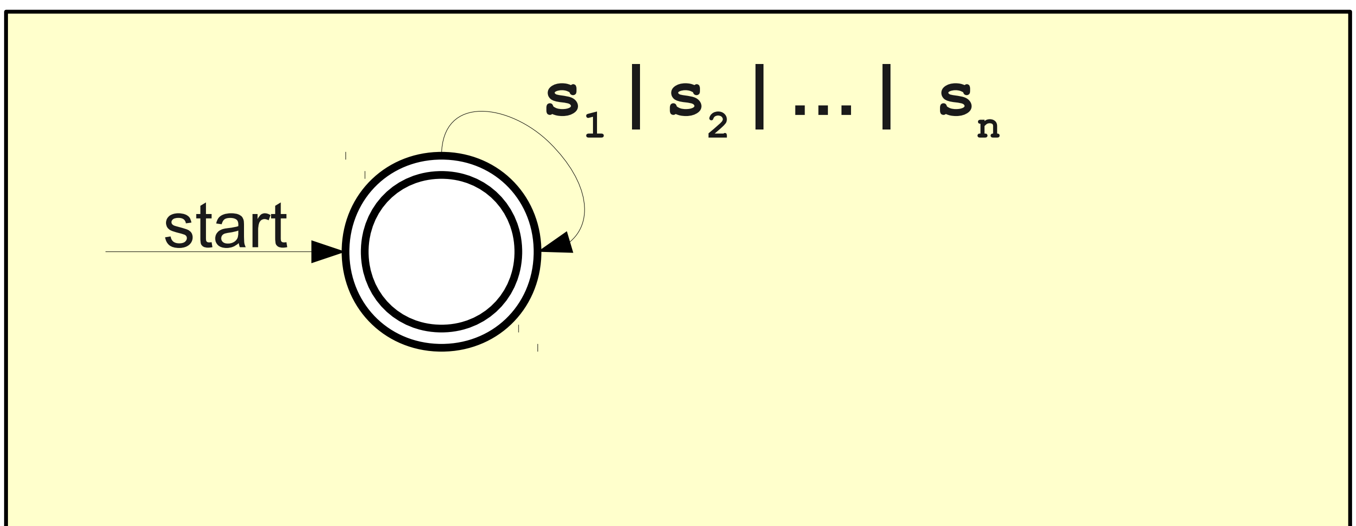
From NFAs to Regular Expressions



Key idea: If we can convert any NFA into something that looks like this, we can easily read off the regular expression.

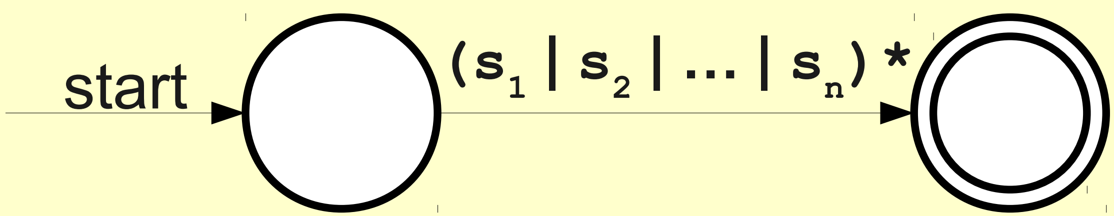
From NFAs to Regular Expressions





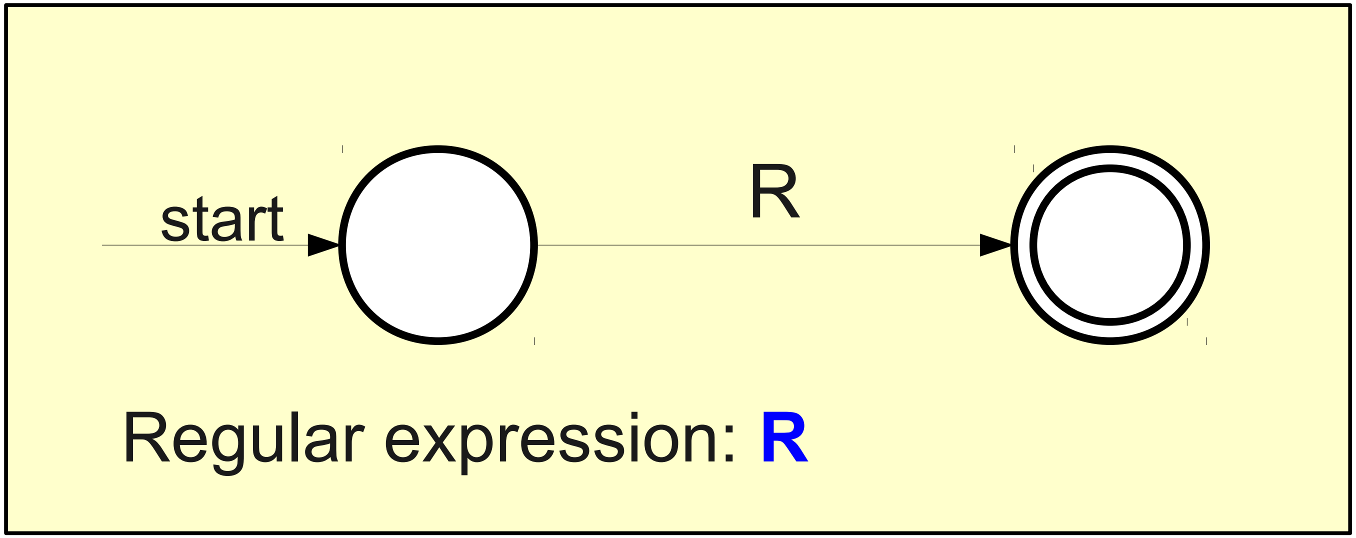
From NFAs to Regular Expressions

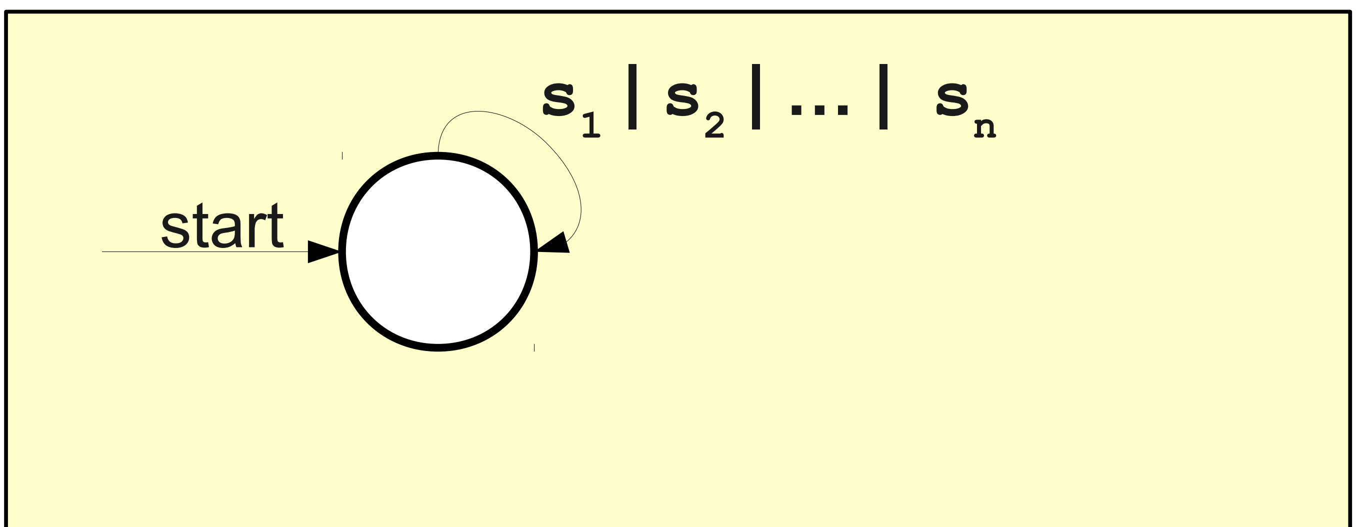
Regular expression: **R**



Regular expression: $(s_1 \mid s_2 \mid \dots \mid s_n)^*$

From NFAs to Regular Expressions



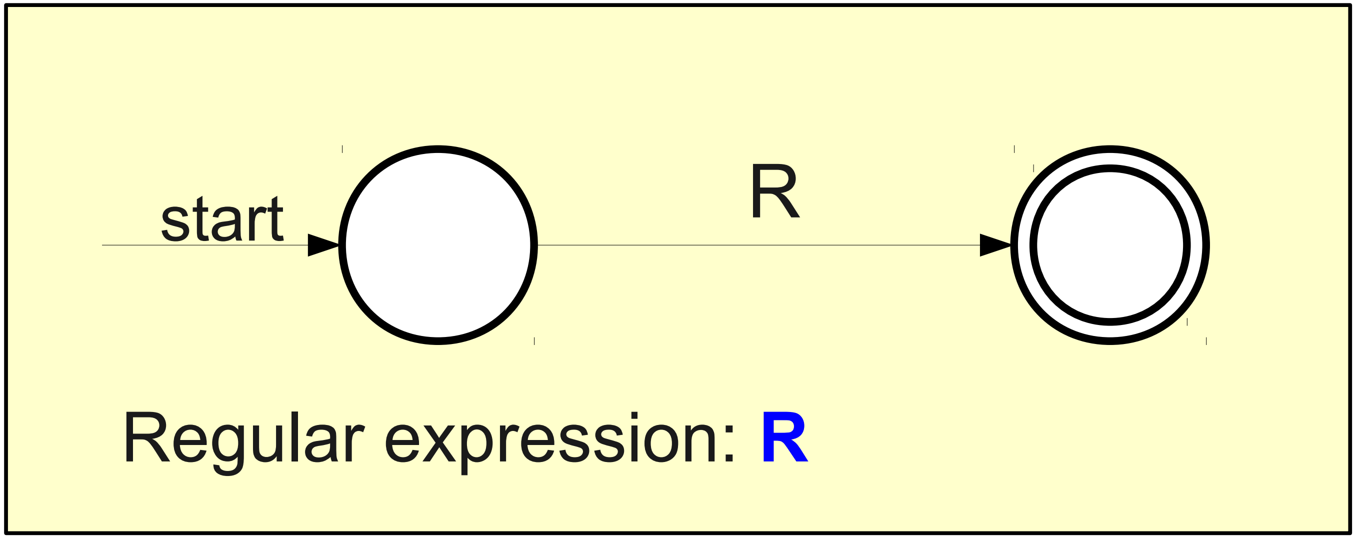


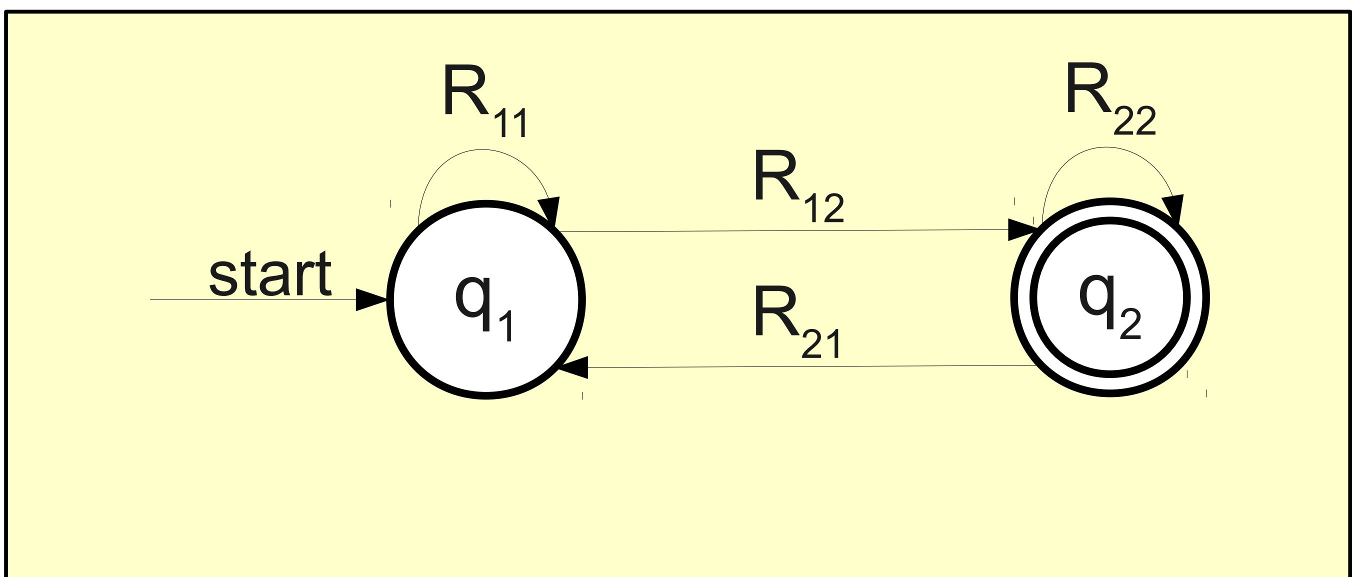
From NFAs to Regular Expressions

Regular expression: **R**

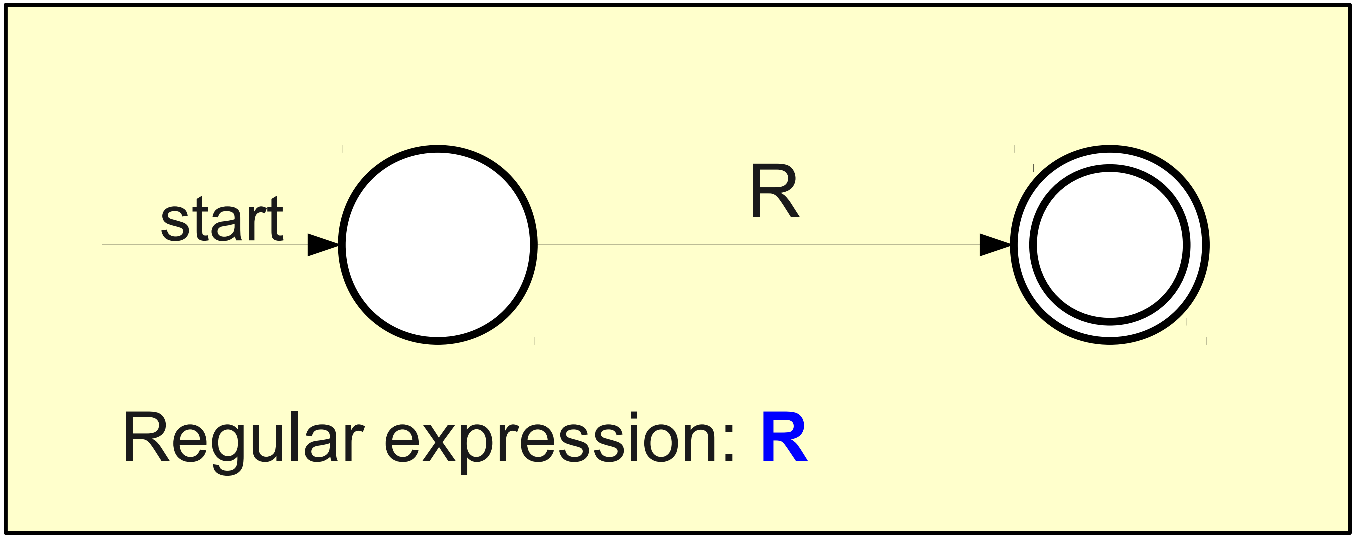
Regular expression: **Ø**

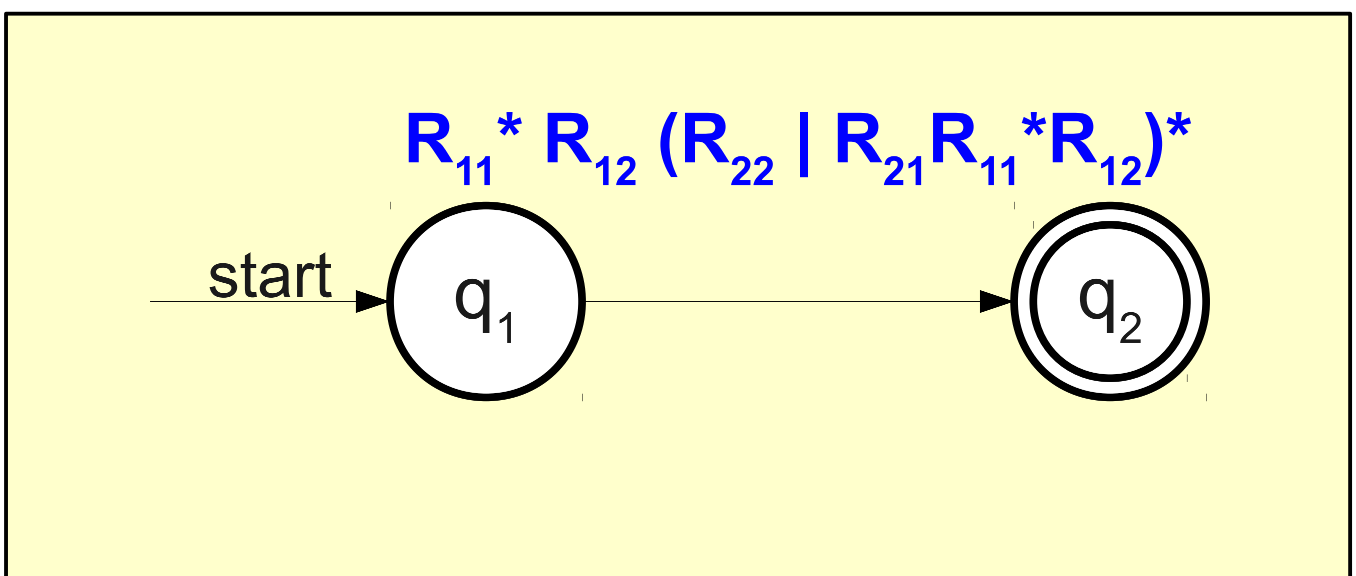
From NFAs to Regular Expressions



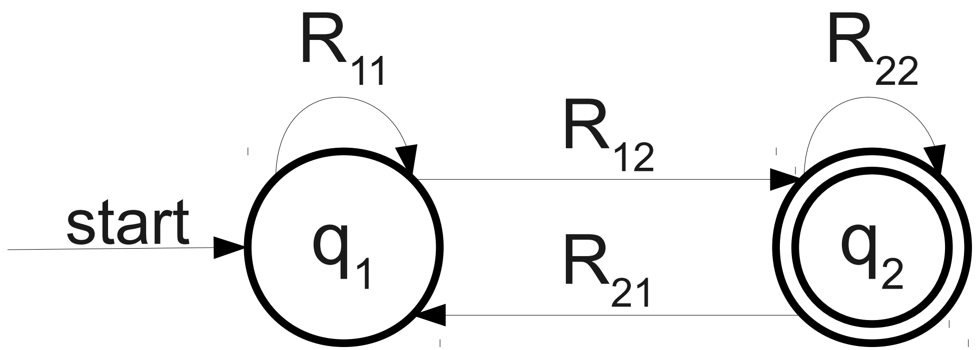


From NFAs to Regular Expressions

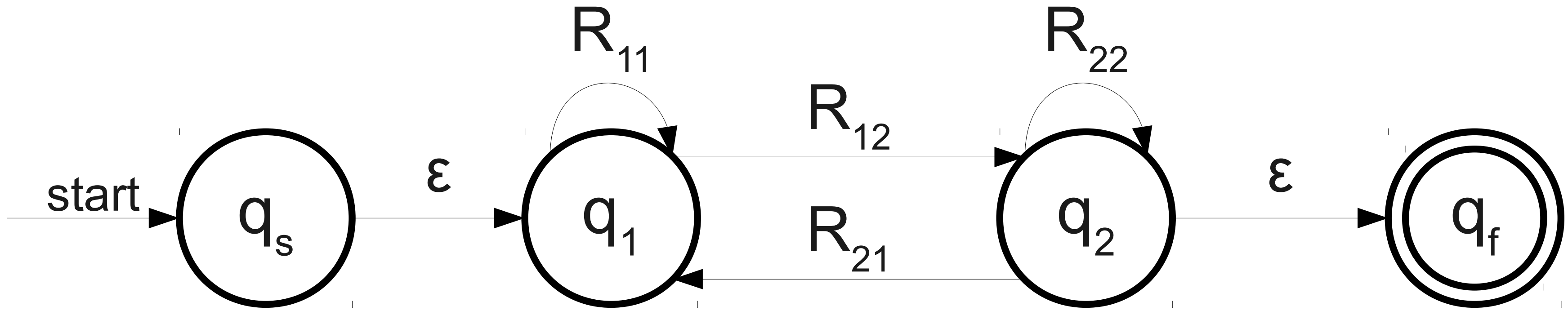




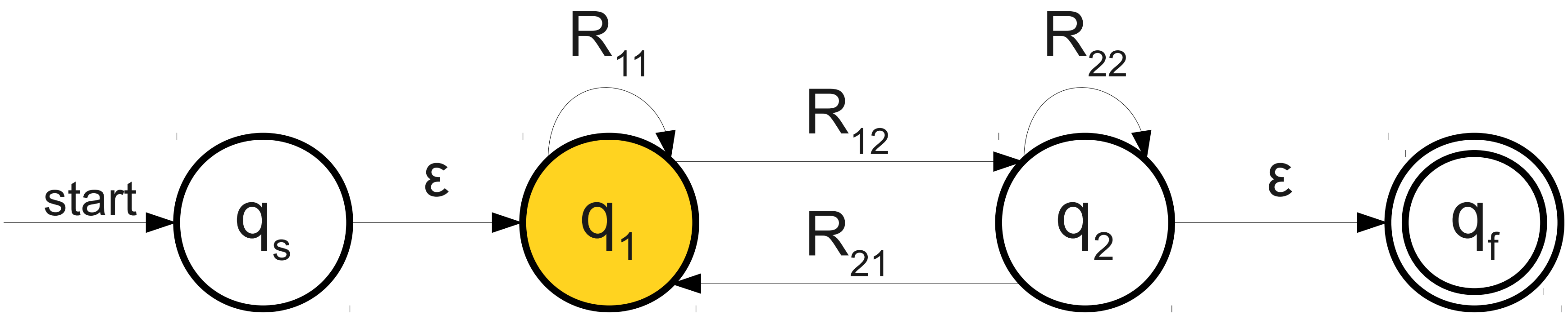
From NFAs to Regular Expressions



From NFAs to Regular Expressions

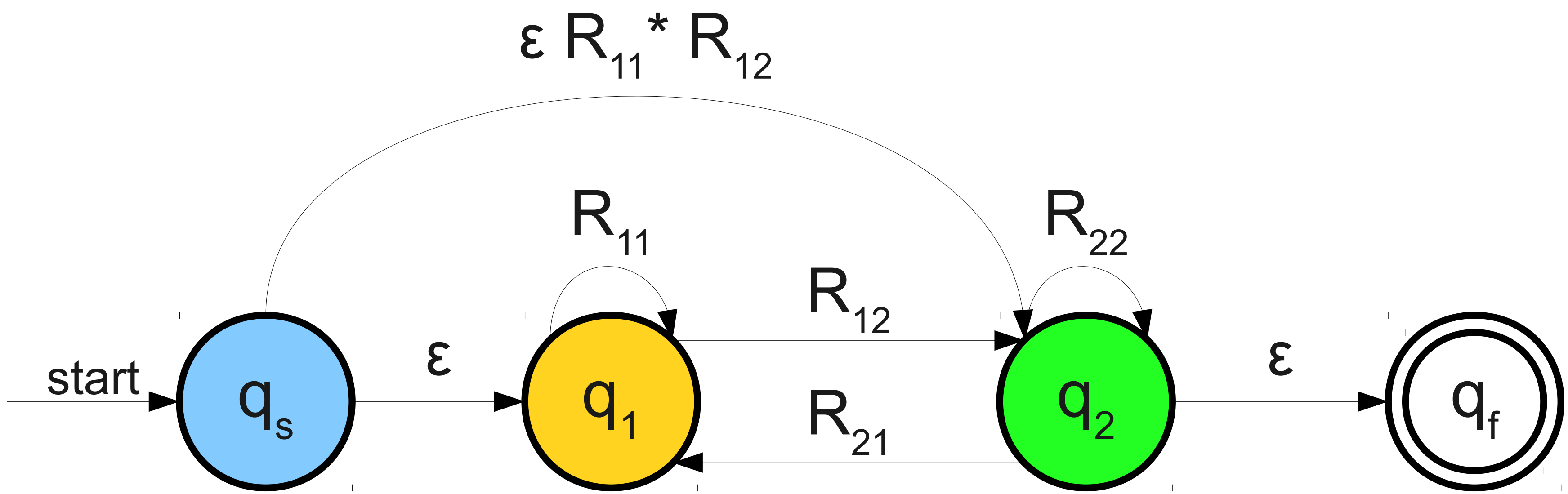


From NFAs to Regular Expressions



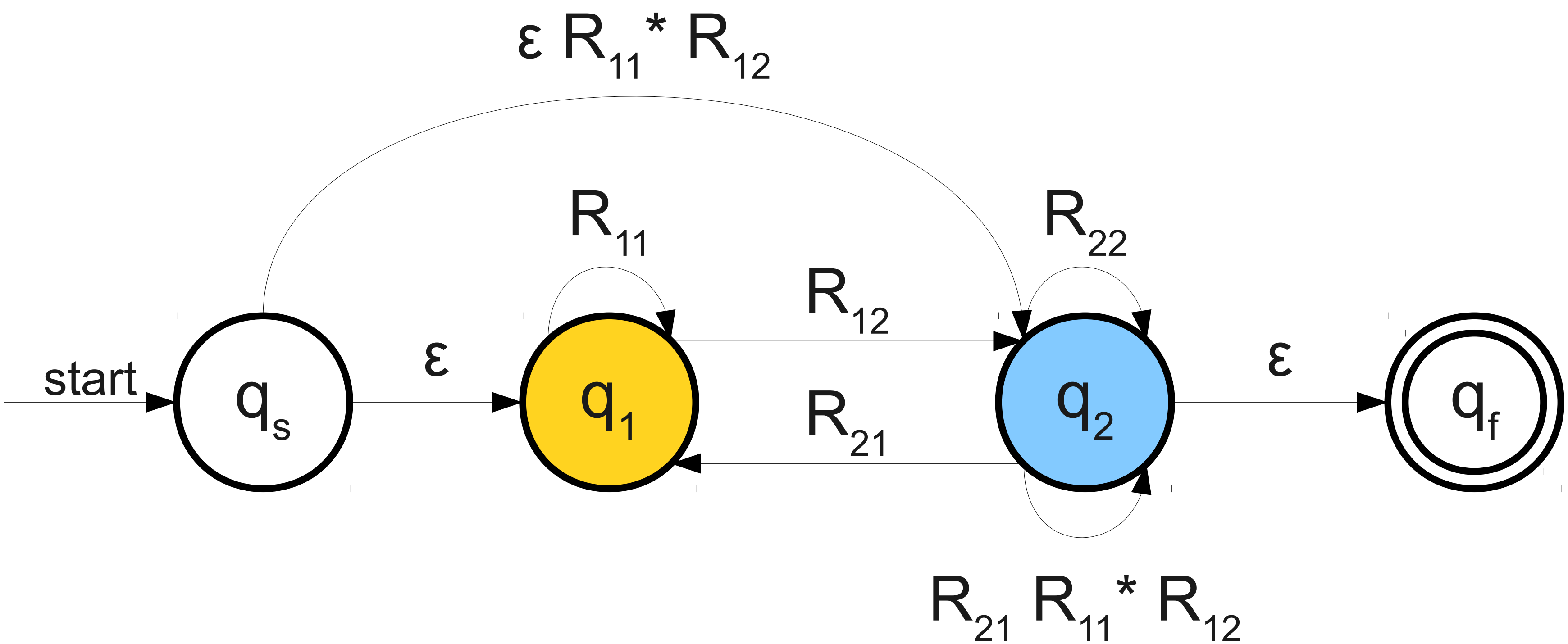
Could we eliminate this state from the NFA?

From NFAs to Regular Expressions

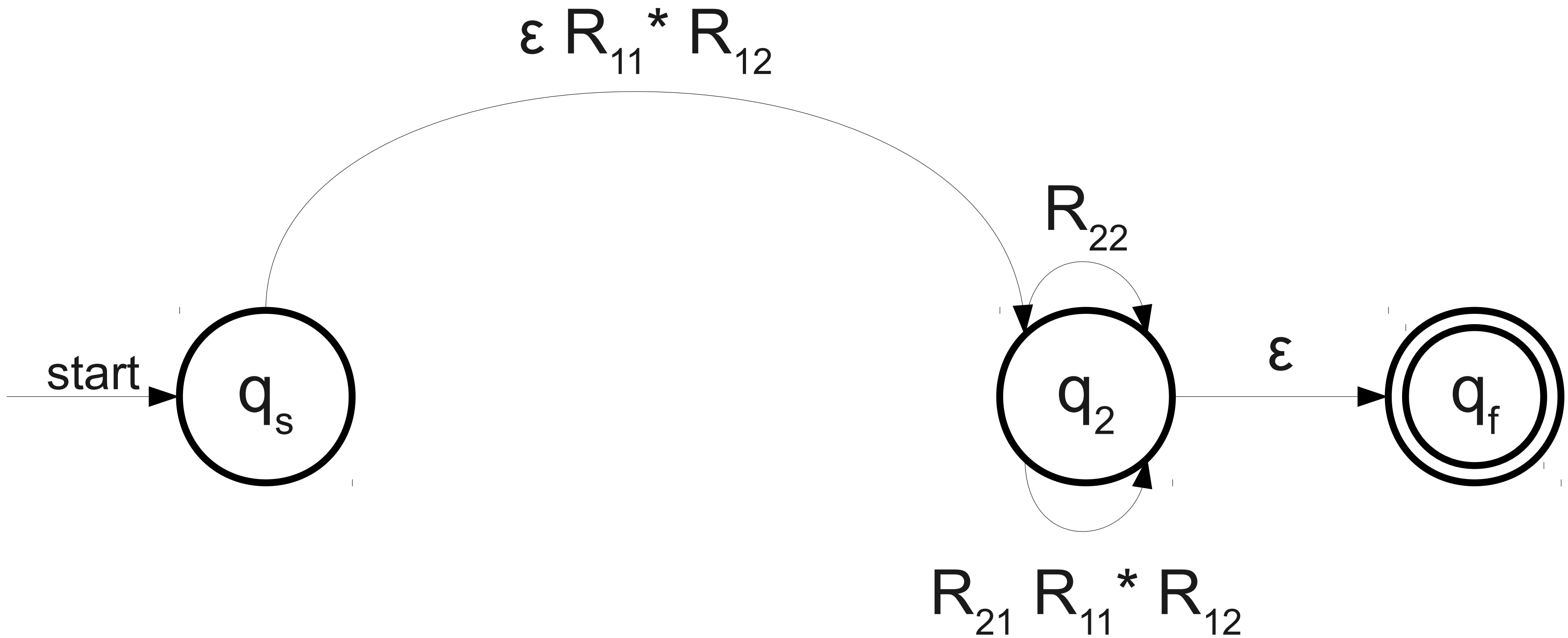


Note: We're using **concatenation** and **Kleene closure** in order to skip this state.

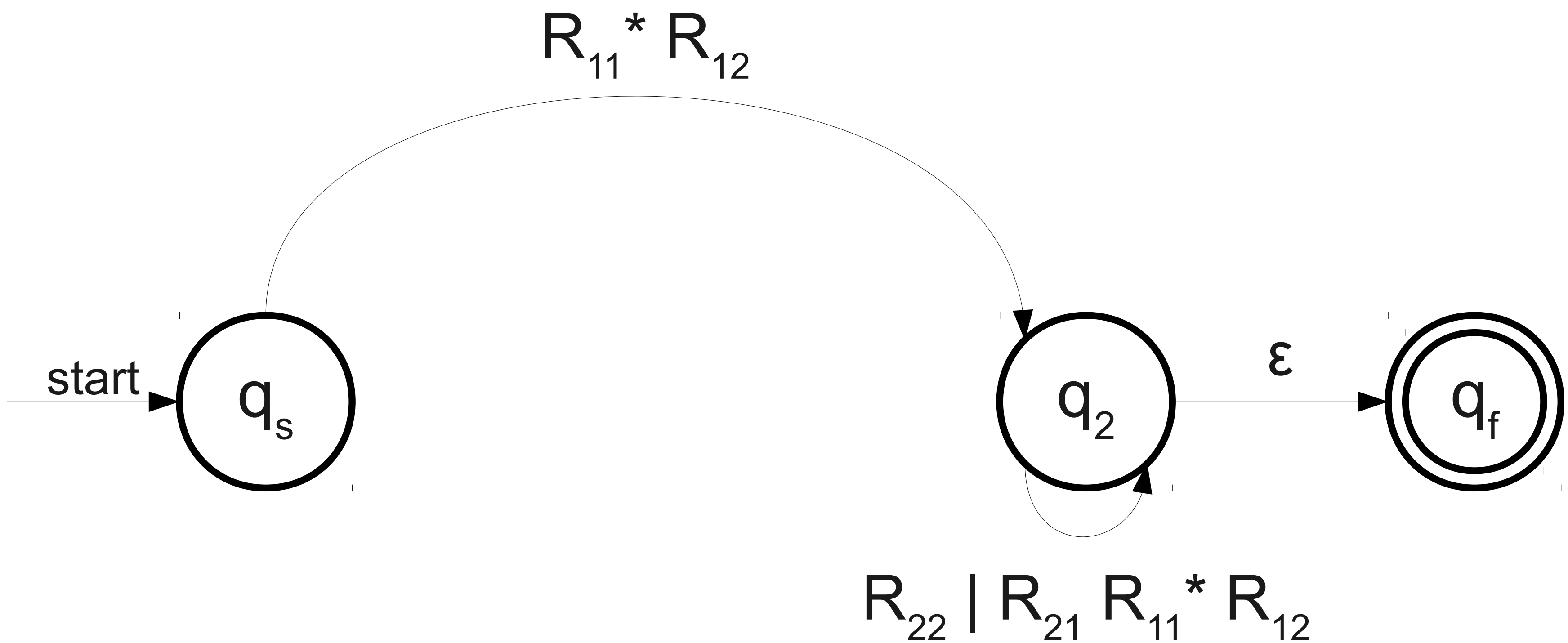
From NFAs to Regular Expressions



From NFAs to Regular Expressions

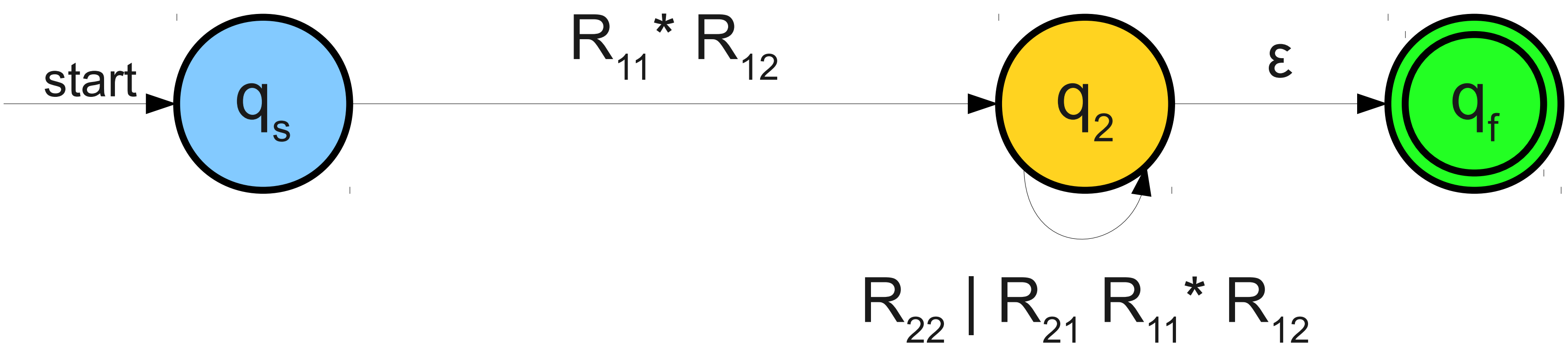


From NFAs to Regular Expressions

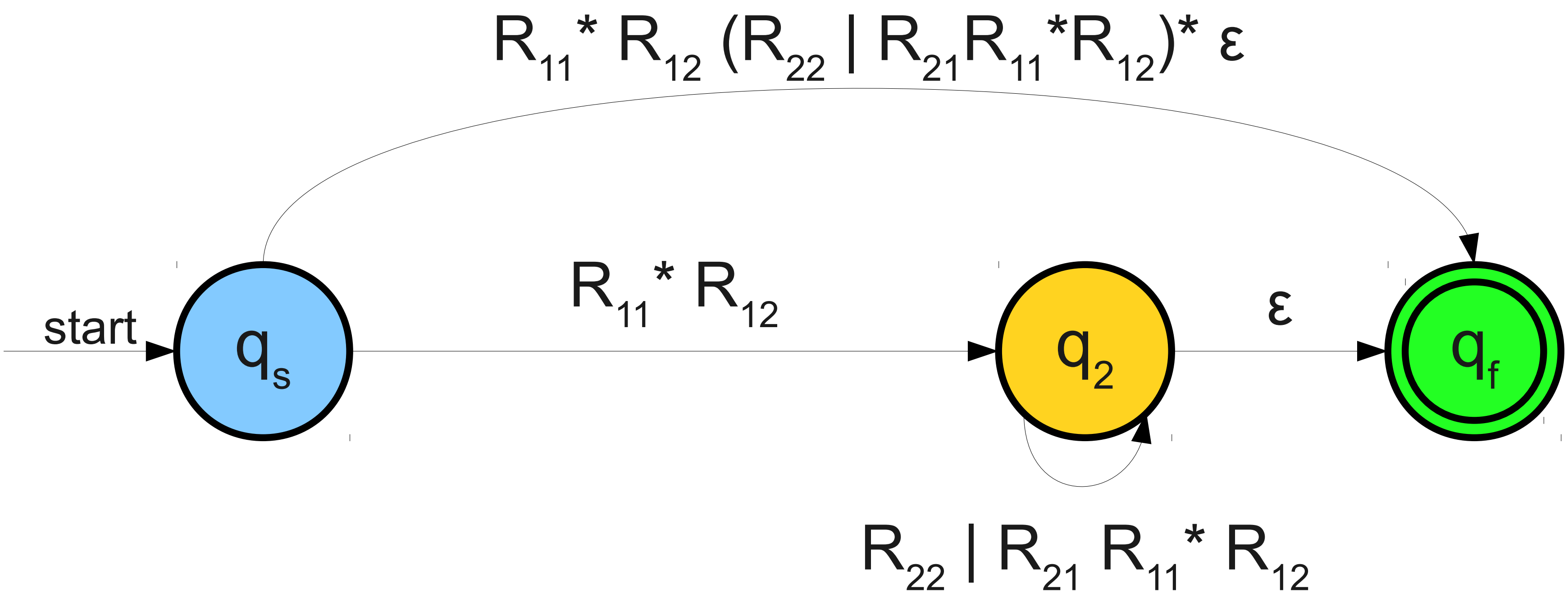


Note: We're using union to combine these transitions together.

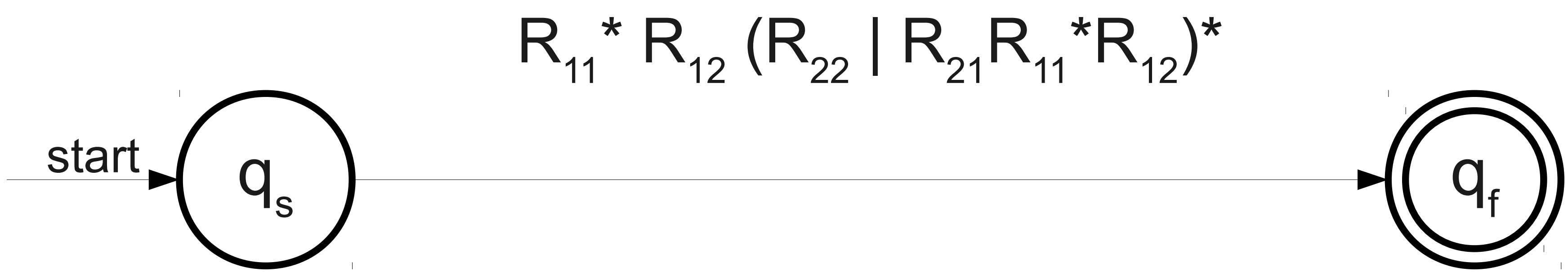
From NFAs to Regular Expressions

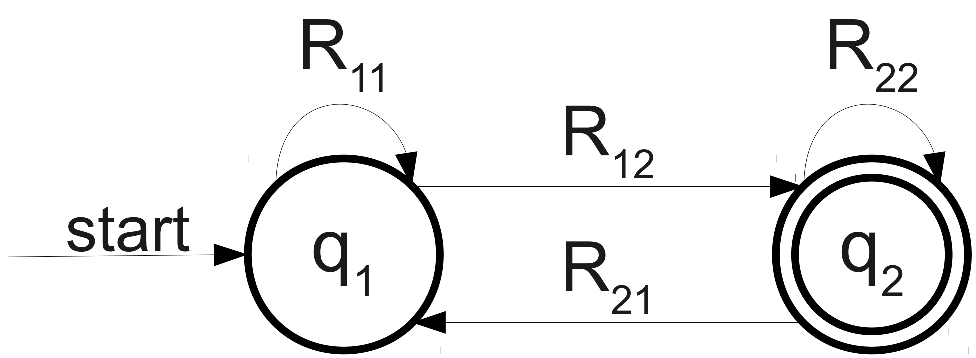


From NFAs to Regular Expressions



From NFAs to Regular Expressions

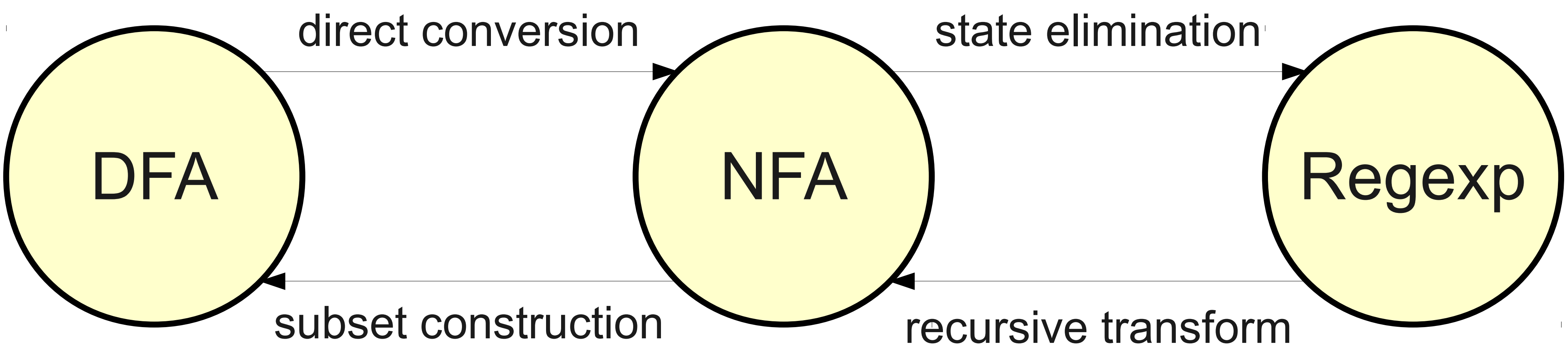




The Construction at a Glance

- Start with an NFA for the language L.
- Add a new start state q_s and accept state q_f to the NFA.
  - Add ε-transitions from each original accepting state 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.

Our Transformations



Theorem: The following are all equivalent:

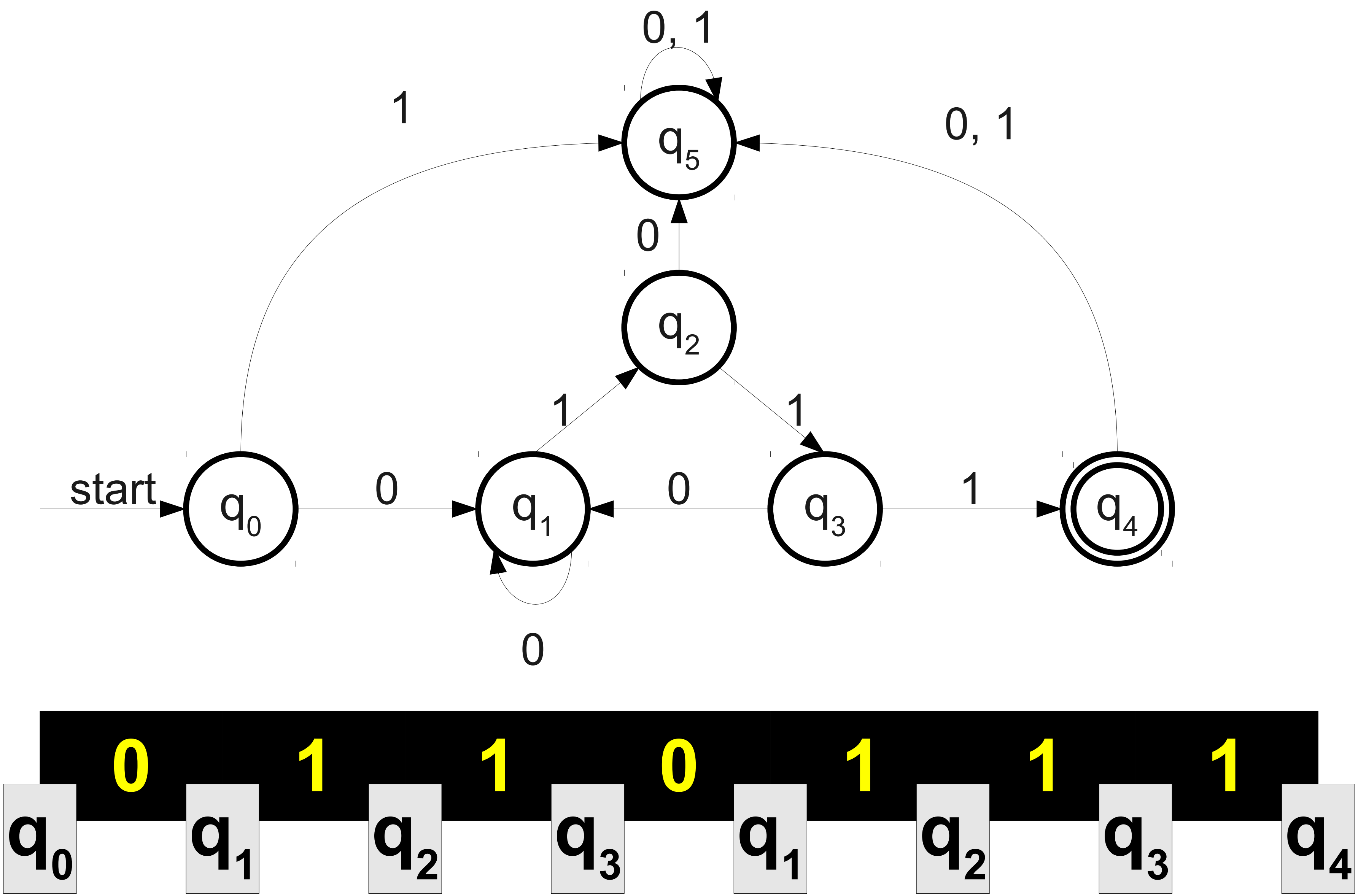
- L is a regular language.
- There is a DFA D such that $\mathscr{L}(D) = L$.
- There is an NFA N such that $\mathscr{L}(N) = L$.
- There is a regular expression R such that $\mathscr{L}(R) = L$.

Why This All Matters

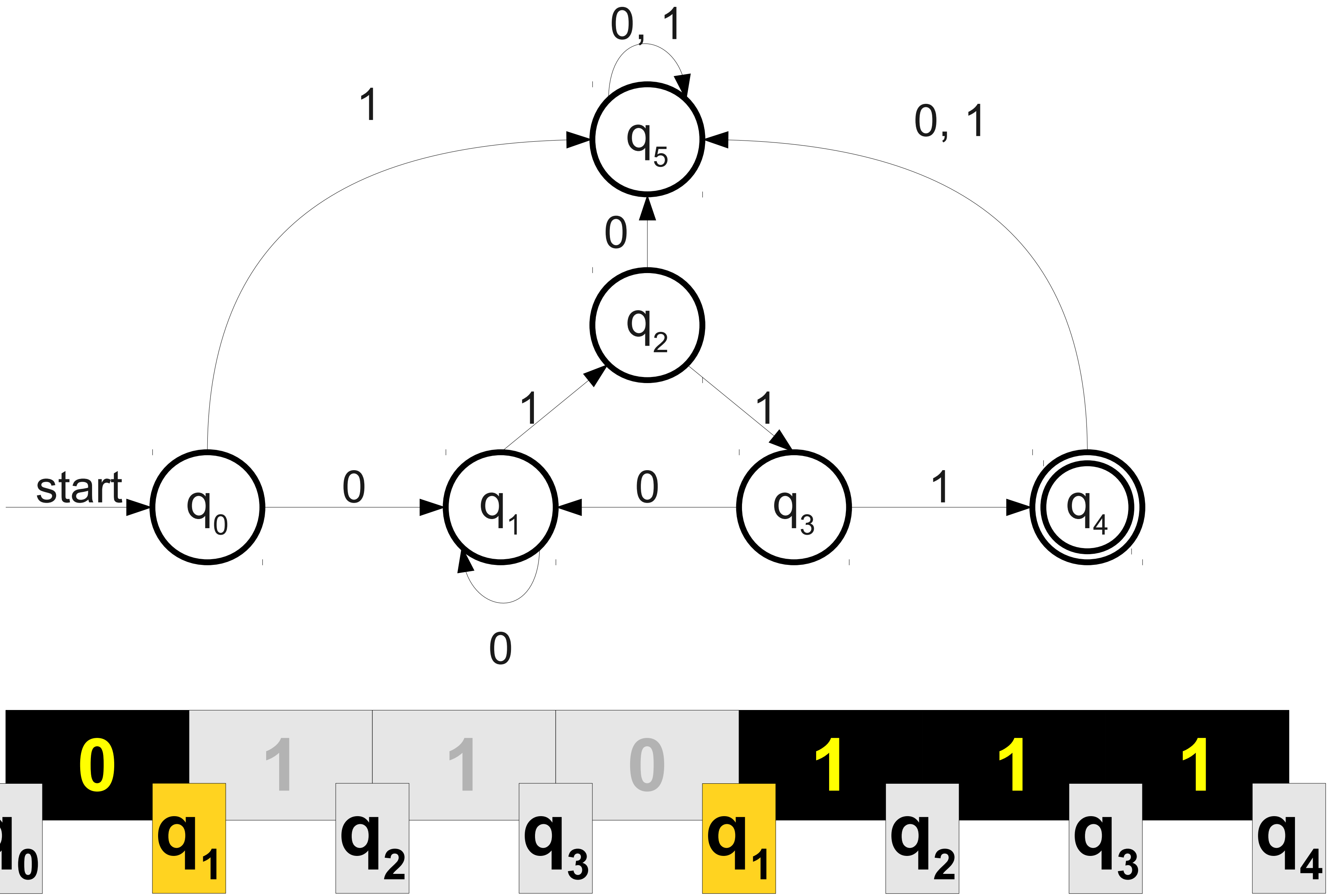
- DFAs correspond to computers with **finite memory**.
- The equivalence of DFAs and NFAs tells us that given finite memory, nondeterminism does not increase computational power.
  - Though it might save on memory.
- The equivalence of DFAs and regular expressions tells us that all problems solvable by finite computers can be assembled out of smaller building blocks.

Is every language regular?

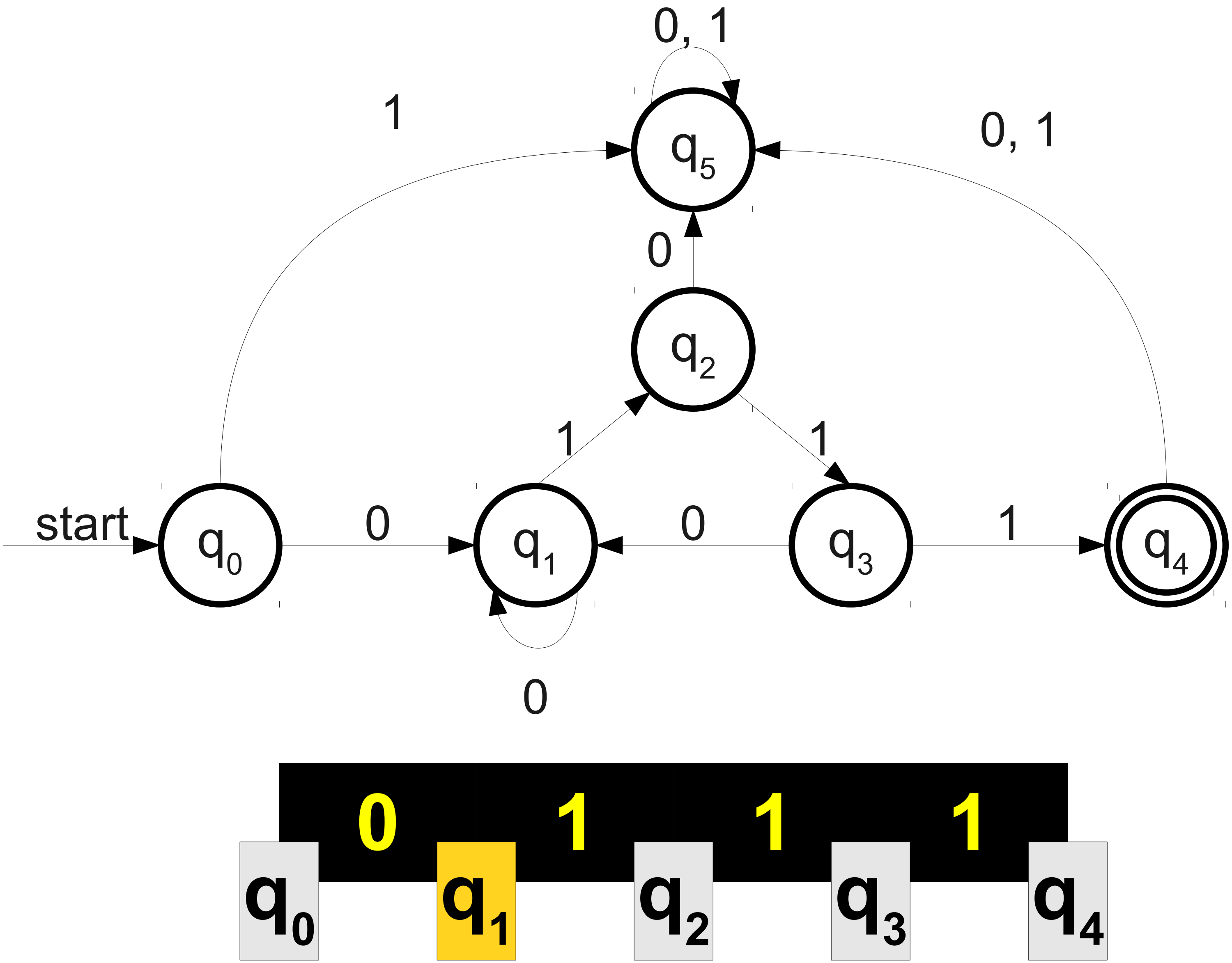
An Important Observation



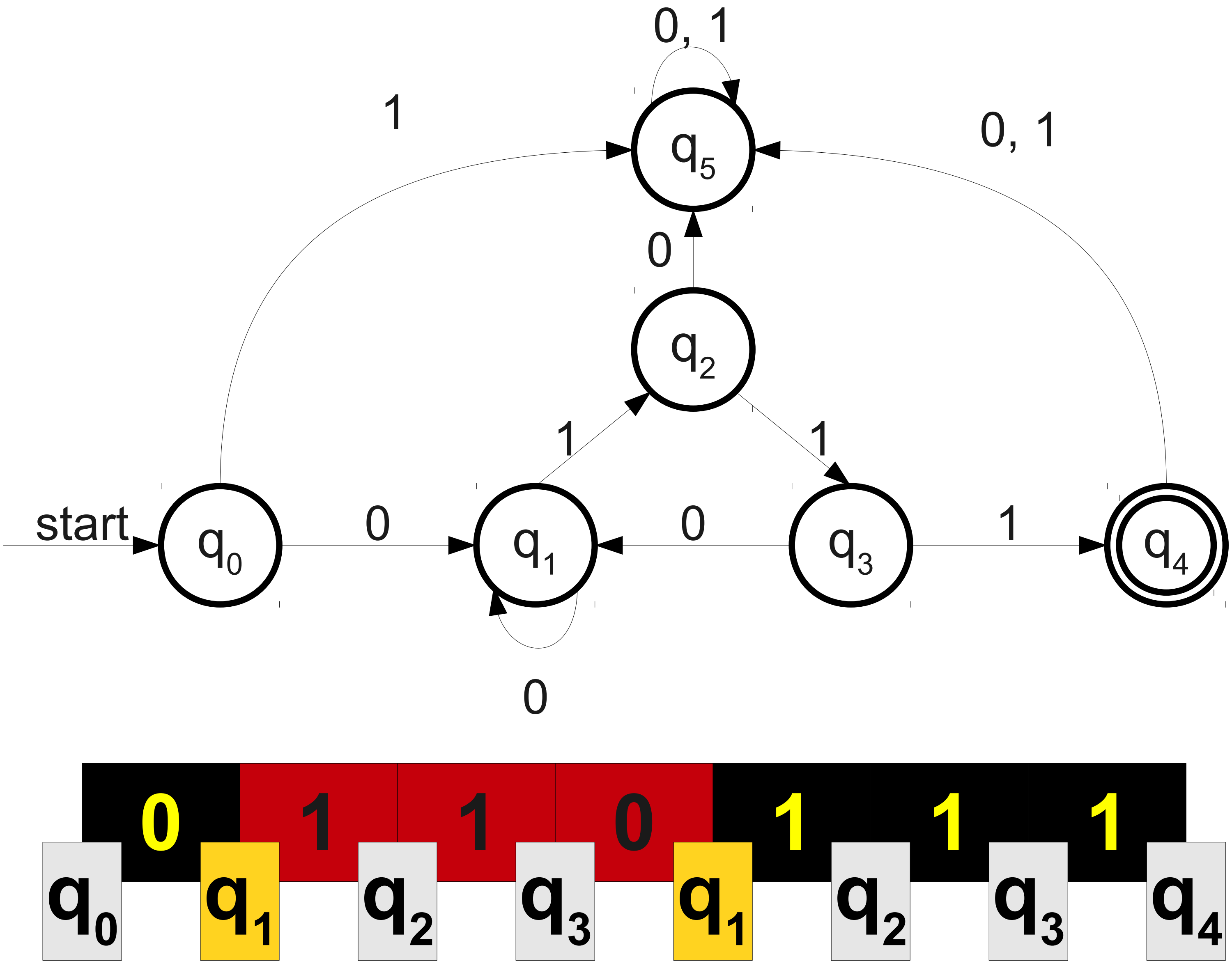
An Important Observation



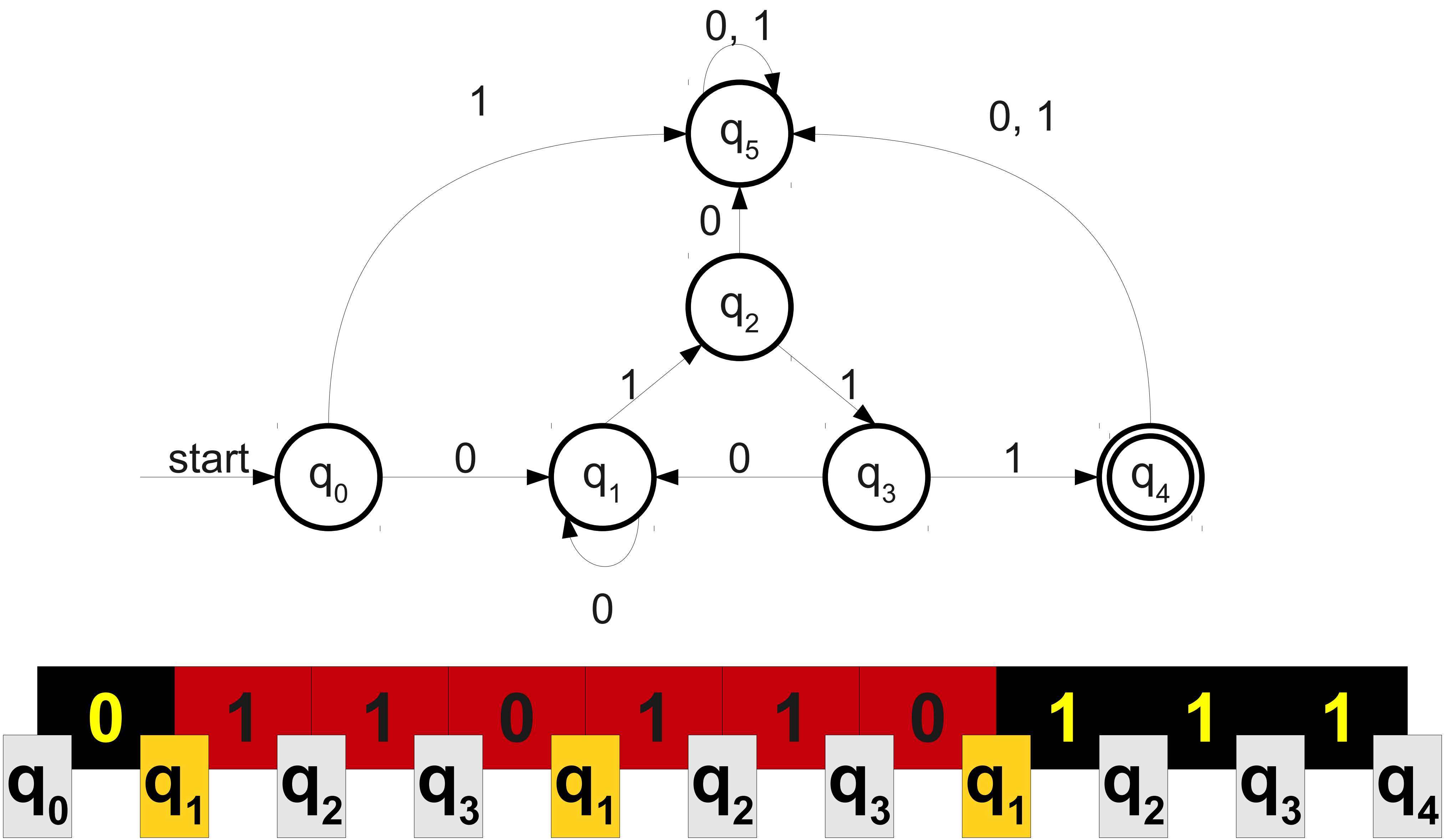
An Important Observation



An Important Observation



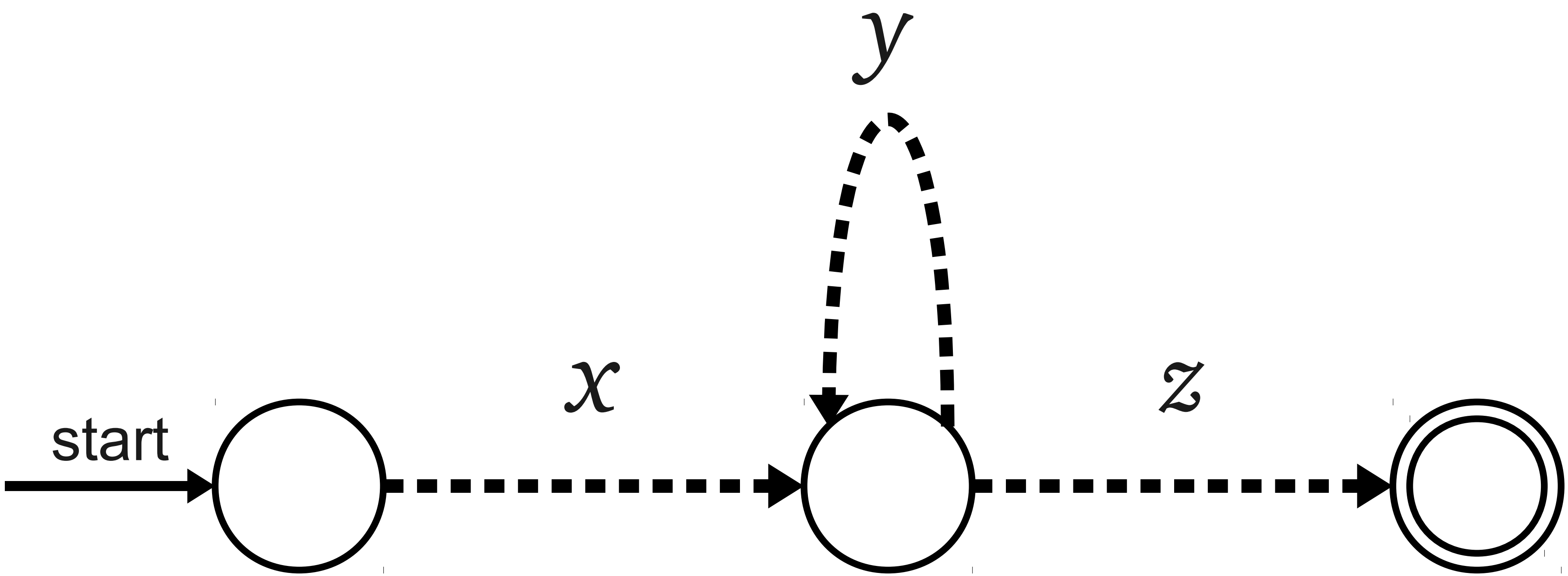
An Important Observation



Visiting Multiple States

- Let D be a DFA with n states.
- Any string w accepted by D that has length at least n must visit some state twice.
  - Number of states visited is equal to the length of the string plus one.
  - By the pigeonhole principle, some state is duplicated.
- The substring of w between those revisited states can be removed, duplicated, tripled, etc. without changing the fact that D accepts w.

Intuitively



Informally

- Let L be a regular language.
- If we have a string $w \in L$ that is "sufficiently long," then we can split the string into three pieces and "pump" the middle.
- We can write $w = xyz$ such that xy^0z, xy^1z, xy^2z, …, xy^nz, … are all in L.
  - **Notation**: y^n means "n copies of y."

The Weak Pumping Lemma

- The **Weak Pumping Lemma for Regular Languages** states that

For any regular language L,

There exists a positive natural number n such that

For any $w \in L$ with $|w| \geq n$,

There exists strings x, y, z such that

For any natural number i,

$w = xyz$,

$y \neq \varepsilon$

$xy^iz \in L$

The Weak Pumping Lemma

- The **Weak Pumping Lemma for Regular Languages** states that

For any regular language L,

There exists a positive natural number n such that

For any $w \in L$ with $|w| \geq n$,

There exists strings x, y, z such that

For any natural number i,

$w = xyz,$

$y \neq \varepsilon$

$xy^iz \in L$

This number n is sometimes called the pumping length.

The Weak Pumping Lemma

- The **Weak Pumping Lemma for Regular Languages** states that

For any regular language L,

There exists a positive natural number n such that

For any $w \in L$ with $|w| \geq n$,

There exists strings x, y, z such that

For any natural number i,

$w = xyz$,

$y \neq \varepsilon$

$xy^iz \in L$

Strings longer than the pumping length must have a special property.

The Weak Pumping Lemma

- The **Weak Pumping Lemma for Regular Languages** states that

For any regular language L,

There exists a positive natural number n such that

For any $w \in L$ with $|w| \geq n$,

There exists strings x, y, z such that

For any natural number i,

$w = xyz$, w can be broken into three pieces,

$y \neq \varepsilon$ where the middle piece isn't empty,

$xy^iz \in L$ where the middle piece can be replicated zero or more times.

The Weak Pumping Lemma

- Let $\Sigma = \{\mathtt{0}, \mathtt{1}\}$ and $L = \{ w \in \Sigma^* \mid w$ contains $\mathtt{00}$ as a substring. $\}$
- Any string of length 3 or greater can be split into three pieces, the second of which can be "pumped."

The Weak Pumping Lemma

- Let $\Sigma = \{0, 1\}$ and $L = \{ \varepsilon, 0, 1, 00, 01, 10, 11 \}$
- Any string of length 3 or greater can be split into three pieces, the second of which can be "pumped."

The weak pumping lemma holds for finite languages because the pumping length can be longer than the longest string!

Testing Equality

- The **equality problem** is defined as follows:

  Given two strings x and y, decide if $x = y$.

- Let $\Sigma = \{$`0`, `1`, `?`$\}$. We can encode the equality problem as a string of the form x`?`y.
  - "Is `001` equal to `110` ?" would be `001?110`
  - "Is `11` equal to `11` ?" would be `11?11`
  - "Is `110` equal to `110` ?" would be `110?110`
- Let $EQUAL = \{ w\texttt{?}w \mid w \in \{\texttt{0}, \texttt{1}\}^* \}$
- **Question**: Is $EQUAL$ a regular language?

The Weak Pumping Lemma

- The **Weak Pumping Lemma for Regular Languages** states that

For any regular language L,

There exists a positive natural number n such that

For any $w \in L$ with $|w| \geq n$,

There exists strings x, y, z such that

For any natural number i,

$w = xyz$, w can be broken into three pieces,

$y \neq \varepsilon$ where the middle piece isn't empty,

$xy^iz \in L$ where the middle piece can be replicated zero or more times.

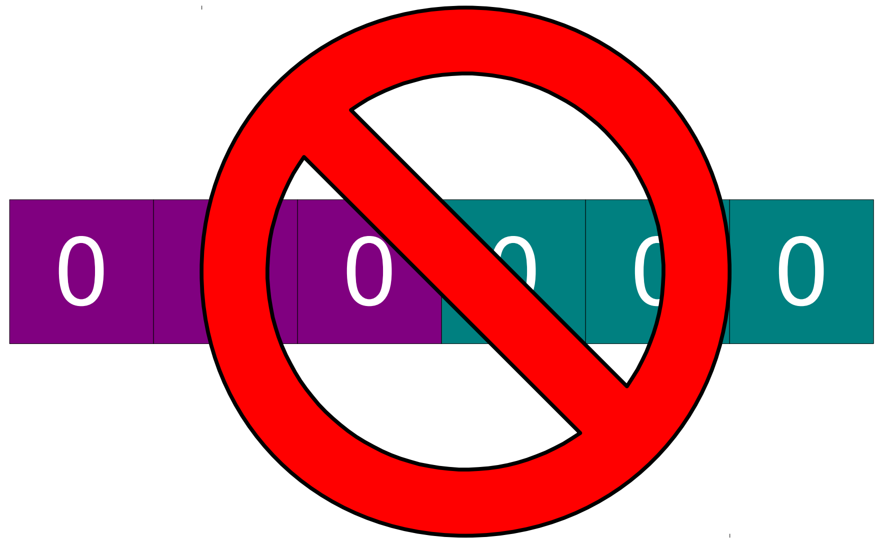
Using the Weak Pumping Lemma

$$EQUAL = \{\ w\texttt{?}w \mid w \in \{\texttt{0},\ \texttt{1}\}^*\ \}$$

0	0	0	?	0	0	0

Using the Weak Pumping Lemma

$$EQUAL = \{\ w\texttt{?}w \mid w \in \{\texttt{0}, \texttt{1}\}^*\ \}$$



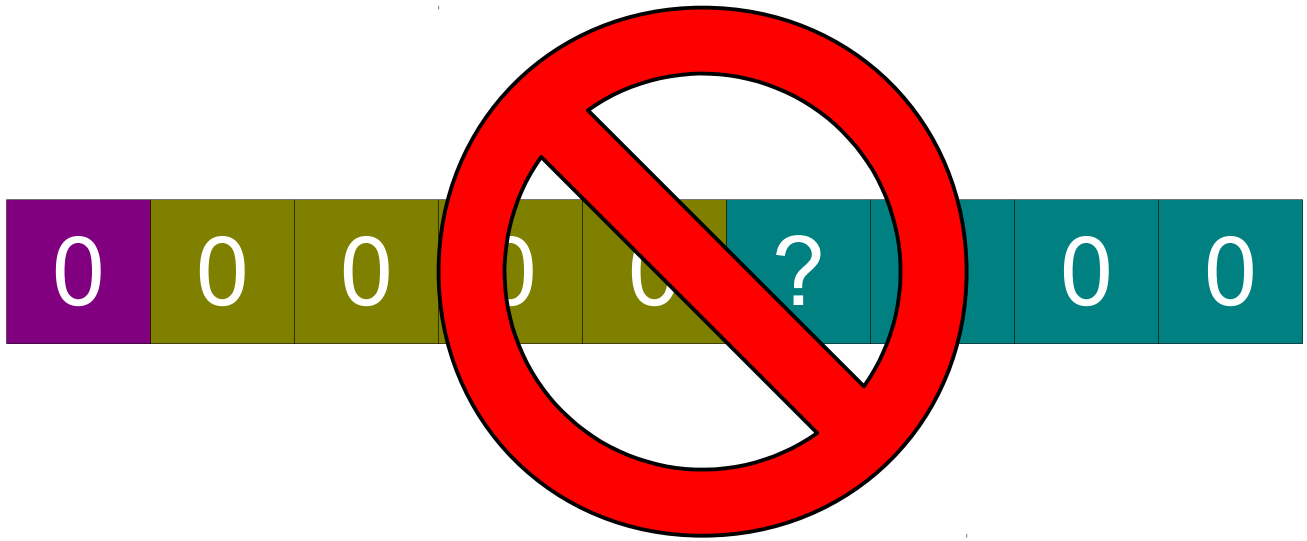
Using the Weak Pumping Lemma

$$EQUAL = \{\ w\texttt{?}w \mid w \in \{\texttt{0},\ \texttt{1}\}^*\ \}$$

0	0	0	?	0	0	0

Using the Weak Pumping Lemma

$$EQUAL = \{ \, w\texttt{?}w \mid w \in \{\texttt{0}, \texttt{1}\}^* \, \}$$



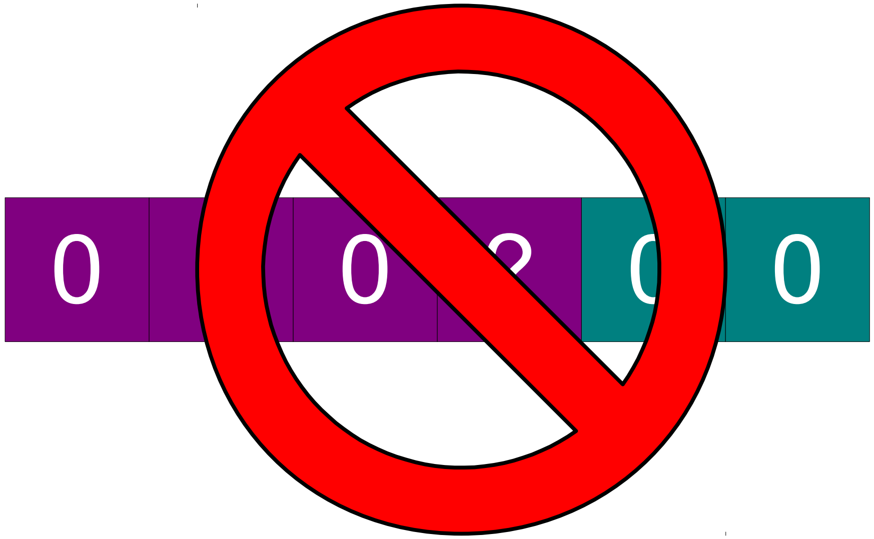
Using the Weak Pumping Lemma

$$EQUAL = \{\ w\texttt{?}w \mid w \in \{\texttt{0}, \texttt{1}\}^*\ \}$$

0	0	0	?	0	0	0

Using the Weak Pumping Lemma

$$EQUAL = \{\ w\mathtt{?}w \mid w \in \{\mathtt{0}, \mathtt{1}\}^*\ \}$$



What's Going On?

- The weak pumping lemma says that for "sufficiently long" strings, we should be able to pump some part of the string.
- We can't pump any part containing the `?`, because we can't duplicate or remove it.
- We can't pump just one part of the string, because then the strings on opposite sides of the `?` wouldn't match.
- **Can we formally show that *EQUAL* is not regular?**

For any regular language L,
There exists a positive natural number n such that
For any $w \in L$ with $|w| \geq n$,
There exists strings x, y, z such that
For any natural number i,
$w = xyz$,
$y \neq \varepsilon$
$xy^iz \in L$

Theorem: EQUAL is not regular.

Proof: By contradiction; assume that *EQUAL* is regular. Let n be the pumping length guaranteed by the weak pumping lemma. Let $w = \texttt{0}^n\texttt{?}\texttt{0}^n$. Then $w \in EQUAL$ and $|w| = 2n + 1 \geq n$. Thus by the weak pumping lemma, we can write $w = xyz$ such that $y \neq \varepsilon$ and for any $i \in \mathbb{N}$, $xy^iz \in EQUAL$. Then y cannot contain `?`, since otherwise if we let $i = 0$, then $xy^iz = xz$ does not contain `?` and would not be in *EQUAL*. So y is either completely to the left of the `?` or completely to the right of the `?`. Let $|y| = k$, so $k > 0$. Since y is completely to the left or right of the `?`, then $y = \texttt{0}^k$. Now, we consider two cases:

Case 1: y is to the left of the `?`. Then $xy^2z = \texttt{0}^{n+k}\texttt{?}\texttt{0}^n \notin EQUAL$, contradicting the weak pumping lemma.

Case 2: y is to the right of the `?`. Then $xy^2z = \texttt{0}^n\texttt{?}\texttt{0}^{n+k} \notin EQUAL$, contradicting the weak pumping lemma.

In either case we reach a contradiction, so our assumption was wrong. Thus *EQUAL* is not regular. ■

Nonregular Languages

- The weak pumping lemma describes a property common to all regular languages.
- Any language L which does not have this property *cannot be regular*.
- What other languages can we find that are not regular?

A Canonical Nonregular Language

- Consider the language $L = \{ \mathtt{0}^n\mathtt{1}^n \mid n \in \mathbb{N} \}$.

  $$L = \{ \varepsilon, \mathtt{01}, \mathtt{0011}, \mathtt{000111}, \mathtt{00001111}, \dots \}$$

- L is a classic example of a nonregular language.
- Intuitively: If you have only finitely many states in a DFA, you can't "remember" an arbitrary number of `0`s.
- How would we prove that L is nonregular?

The Pumping Lemma as a Game

- The weak pumping lemma can be thought of as a game between **you** and an **adversary**.
- **You win** if you can prove that the pumping lemma fails.
- **The adversary wins** if the adversary can make a choice for which the pumping lemma succeeds.
- The game goes as follows:
  - **The adversary** chooses a pumping length n.
  - **You** choose a string w with $|w| \geq n$ and $w \in L$.
  - **The adversary** breaks it into x, y, and z.
  - **You** choose an i such that $xy^iz \notin L$ (if you can't, you lose!)

The Pumping Lemma Game

$L = \{ 0^n1^n \mid n \in \mathbb{N} \}$

ADVERSARY	YOU
Maliciously choose pumping length n.	
	Cleverly choose a string $w \in L, \lvert w \rvert \geq n$
Maliciously split $w = xyz, y \neq \varepsilon$	
	Cleverly choose i such that $xy^iz \notin L$
Grrr! Aaaargh!	

0^n1^n

Theorem: $L = \{ \mathtt{0}^n \mathtt{1}^n \mid n \in \mathbb{N} \}$ is not regular.

Proof: By contradiction; assume L is regular. Let n be the pumping length guaranteed by the weak pumping lemma. Consider the string $w = \mathtt{0}^n \mathtt{1}^n$. Then $|w| = 2n \geq n$ and $w \in L$, so we can write $w = xyz$ such that $y \neq \varepsilon$ and for any $i \in \mathbb{N}$, we have $xy^iz \in L$. We consider three cases:

Case 1: y consists solely of $\mathtt{0}$s. Then $xy^0z = xz = \mathtt{0}^{n - |y|}\mathtt{1}^n$, and since $|y| > 0$, $xz \notin L$.

Case 2: y consists solely of $\mathtt{1}$s. Then $xy^0z = xz = \mathtt{0}^n\mathtt{1}^{n - |y|}$, and since $|y| > 0$, $xz \notin L$.

Case 3: y consists of $k > 0$ $\mathtt{0}$s followed by $m > 0$ $\mathtt{1}$s. Then xy^2z has the form $\mathtt{0}^n\mathtt{1}^m\mathtt{0}^k\mathtt{1}^n$, so $xy^2z \notin L$.

In all three cases we reach a contradiction, so our assumption was wrong and L is not regular. ■