Regular Expressions

Recap from Last Time

Regular Languages

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

Language Concatenation

- If $w \in \Sigma^*$ and $x \in \Sigma^*$, then wx is the ***concatenation*** of w and x.
- If L_1 and L_2 are languages over Σ, the ***concatenation*** of L_1 and L_2 is the language L_1L_2 defined as

$$L_1L_2 = \{ wx \mid w \in L_1 \text{ and } x \in L_2 \}$$

- Example: if L_1 = { **a**, **ba**, **bb** } and L_2 = { **aa**, **bb** }, then

$$L_1L_2 = \{ \mathbf{aaa}, \mathbf{abb}, \mathbf{baaa}, \mathbf{babb}, \mathbf{bbaa}, \mathbf{bbbb} \}$$

Lots and Lots of Concatenation

- Consider the language L = { aa, b }
- LL is the set of strings formed by concatenating pairs of strings in L.

{ aaaa, aab, baa, bb }

- LLL is the set of strings formed by concatenating triples of strings in L.

{ aaaaaa, aaaab, aabaa, aabb, baaaa, baab, bbaa, bbb}

- $LLLL$ is the set of strings formed by concatenating quadruples of strings in L.

{ aaaaaaaa, aaaaaab, aaaabaa, aaaabb, aabaaaa, aabaab, aabbaa, aabbb, baaaaaa, baaaab, baabaa, baabb, bbaaaa, bbaab, bbbaa, bbbb}

Language Exponentiation

- We can define what it means to "exponentiate" a language as follows:
- $L^0 = \{\varepsilon\}$
  - The set containing just the empty string.
  - Idea: Any string formed by concatenating zero strings together is the empty string.
- $L^{n+1} = LL^n$
  - Idea: Concatenating (n+1) strings together works by concatenating n strings, then concatenating one more.
- ***Question:*** Why define $L^0 = \{\varepsilon\}$?

The Kleene Closure

- An important operation on languages is the ***Kleene Closure***, which is defined as

$$L^* = \{ w \in \Sigma^* \mid \exists n \in \mathbb{N}.\ w \in L^n \}$$

- Mathematically:

$$w \in L^* \quad \text{iff} \quad \exists n \in \mathbb{N}.\ w \in L^n$$

- Intuitively, all possible ways of concatenating zero or more strings in L together, possibly with repetition.

The Kleene Closure

If L = { **a**, **bb** }, then L^* = {

ε,

a, **bb**,

aa, **abb**, **bba**, **bbbb**,

aaa, **aabb**, **abba**, **abbbb**, **bbaa**, **bbabb**, **bbbba**, **bbbbbb**,

…

}

Think of L* as the set of strings you can make if you have a collection of stamps – one for each string in L – and you form every possible string that can be made from those stamps.

Closure Properties

- ***Theorem:*** If L_1 and L_2 are regular languages over an alphabet Σ, then so are the following languages:
  - $\overline{L_1}$
  - $L_1 \cup L_2$
  - $L_1 \cap L_2$
  - L_1L_2
  - L_1*
- These properties are called ***closure properties of the regular languages***.

New Stuff!

Another View of Regular Languages

Rethinking Regular Languages

- We currently have several tools for showing a language L is regular:
  - Construct a DFA for L.
  - Construct an NFA for L.
  - Combine several simpler regular languages together via closure properties to form L.
- We have not spoken much of this last idea.

Constructing Regular Languages

- ***Idea:*** Build up all regular languages as follows:
  - Start with a small set of simple languages we already know to be regular.
  - Using closure properties, combine these simple languages together to form more elaborate languages.
- *A bottom-up approach to the regular languages.*

Regular Expressions

- ***Regular expressions*** are a way of describing a language via a string representation.
- They're used extensively in software systems for string processing and as the basis for tools like `grep` and `flex`.
- Conceptually, regular expressions are strings describing how to assemble a larger language out of smaller pieces.

Atomic Regular Expressions

- The regular expressions begin with three simple building blocks.
- The symbol **Ø** is a regular expression that represents the empty language Ø.
- For any **a** ∈ Σ, the symbol **a** is a regular expression for the language {**a**}.
- The symbol **ε** is a regular expression that represents the language {ε}.
  - ***Remember:* {ε} ≠ Ø!**
  - ***Remember:* {ε} ≠ ε!**

Compound Regular Expressions

- If R_1 and R_2 are regular expressions, $\boldsymbol{R_1R_2}$ is a regular expression for the *concatenation* of the languages of R_1 and R_2.
- If R_1 and R_2 are regular expressions, $\boldsymbol{R_1 \cup R_2}$ is a regular expression for the *union* of the languages of R_1 and R_2.
- If R is a regular expression, $\boldsymbol{R^*}$ is a regular expression for the *Kleene closure* of the language of R.
- If R is a regular expression, $\boldsymbol{(R)}$ is a regular expression with the same meaning as R.

Operator Precedence

- Here's the operator precedence for regular expressions, from highest to lowest:

$$(R)$$

$$R^*$$

$$R_1R_2$$

$$R_1 \cup R_2$$

Consider the regular expression

ab*c∪d

How many of the strings below are in the language described by this regular expression?

ababc
abd
ac
abcd

Answer at **PollEv.com/cs103** or
text **CS103** to **22333** once to join, then **a number**.

Regular Expression Examples

- The regular expression **trick∪treat** represents the regular language { **trick**, **treat** }.
- The regular expression **booo*** represents the regular language { **boo**, **booo**, **boooo**, … }.
- The regular expression **candy!(candy!)*** represents the regular language { **candy!**, **candy!candy!**, **candy!candy!candy!**, … }.

Regular Expressions, Formally

- The ***language of a regular expression*** is the language described by that regular expression.
- Formally:
  - $\mathscr{L}(\boldsymbol{\varepsilon}) = \{\varepsilon\}$
  - $\mathscr{L}(\boldsymbol{\emptyset}) = \emptyset$
  - $\mathscr{L}(\mathbf{a}) = \{\mathbf{a}\}$
  - $\mathscr{L}(R_1R_2) = \mathscr{L}(R_1)\,\mathscr{L}(R_2)$
  - $\mathscr{L}(R_1 \cup R_2) = \mathscr{L}(R_1) \cup \mathscr{L}(R_2)$
  - $\mathscr{L}(R\mathbf{*}) = \mathscr{L}(R)^*$
  - $\mathscr{L}(\mathbf{(}R\mathbf{)}) = \mathscr{L}(R)$

Worthwhile activity: Apply this recursive definition to

a(b∪c)((d))

and see what you get.

Designing Regular Expressions

- Let Σ = {**a**, **b**}.
- Let L = { $w \in \Sigma^*$ | w contains **aa** as a substring }.

(a ∪ b)*aa(a ∪ b)*

bbabbbaabab

aaaa

bbbbbabbbbaabbbbb

Designing Regular Expressions

- Let Σ = {**a**, **b**}.
- Let L = { $w \in \Sigma^*$ | w contains **aa** as a substring }.

Σ*aaΣ*

bbabbbaabab
aaaa
bbbbbabbbbaabbbbb

Designing Regular Expressions

Let $\Sigma = \{\mathtt{a}, \mathtt{b}\}$.

Let $L = \{ w \in \Sigma^* \mid |w| = 4 \}$.

The length of a string w is denoted |w|

Designing Regular Expressions

- Let $\Sigma = \{$**a**, **b**$\}$.
- Let $L = \{ w \in \Sigma^* \mid |w| = 4 \}$.

$\Sigma\Sigma\Sigma\Sigma$

aaaa
baba
bbbb
baaa

Designing Regular Expressions

- Let $\Sigma = \{$**a**, **b**$\}$.
- Let $L = \{ w \in \Sigma^* \mid |w| = 4 \}$.

$$\Sigma^4$$

aaaa
baba
bbbb
baaa

Designing Regular Expressions

- Let Σ = {**a**, **b**}.
- Let *L* = { *w* ∈ Σ* | *w* contains at most one **a** }.

Which of the following is a regular expression for *L?*

A. **Σ*aΣ***
B. **b*ab* ∪ b***
C. **b*(a ∪ ε)b***
D. **b*a*b* ∪ b***
E. **b*(a* ∪ ε)b***
F. None of the above, or two or more of the above.

Answer at **PollEv.com/cs103** or
text **CS103** to **22333** once to join, then ***A***, ***B***, ***C***, ***D***, ***E***, or ***F***.

Designing Regular Expressions

- Let $\Sigma = \{$**a**, **b**$\}$.
- Let $L = \{ w \in \Sigma^* \mid w$ contains at most one **a** $\}$.

b*(a ∪ ε)b*

bbbbabbb
bbbbbb
abbb
a

Designing Regular Expressions

- Let $\Sigma = \{$`a`, `b`$\}$.
- Let $L = \{ w \in \Sigma^* \mid w$ contains at most one `a` $\}$.

b*a?b*

bbbbabbb
bbbbbb
abbb
a

A More Elaborate Design

- Let $\Sigma = \{$ a, ., @ $\}$, where a represents "some letter."
- Let's make a regex for email addresses.

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

```
cs103@cs.stanford.edu
first.middle.last@mail.site.org
dot.at@dot.com
```

A More Elaborate Design

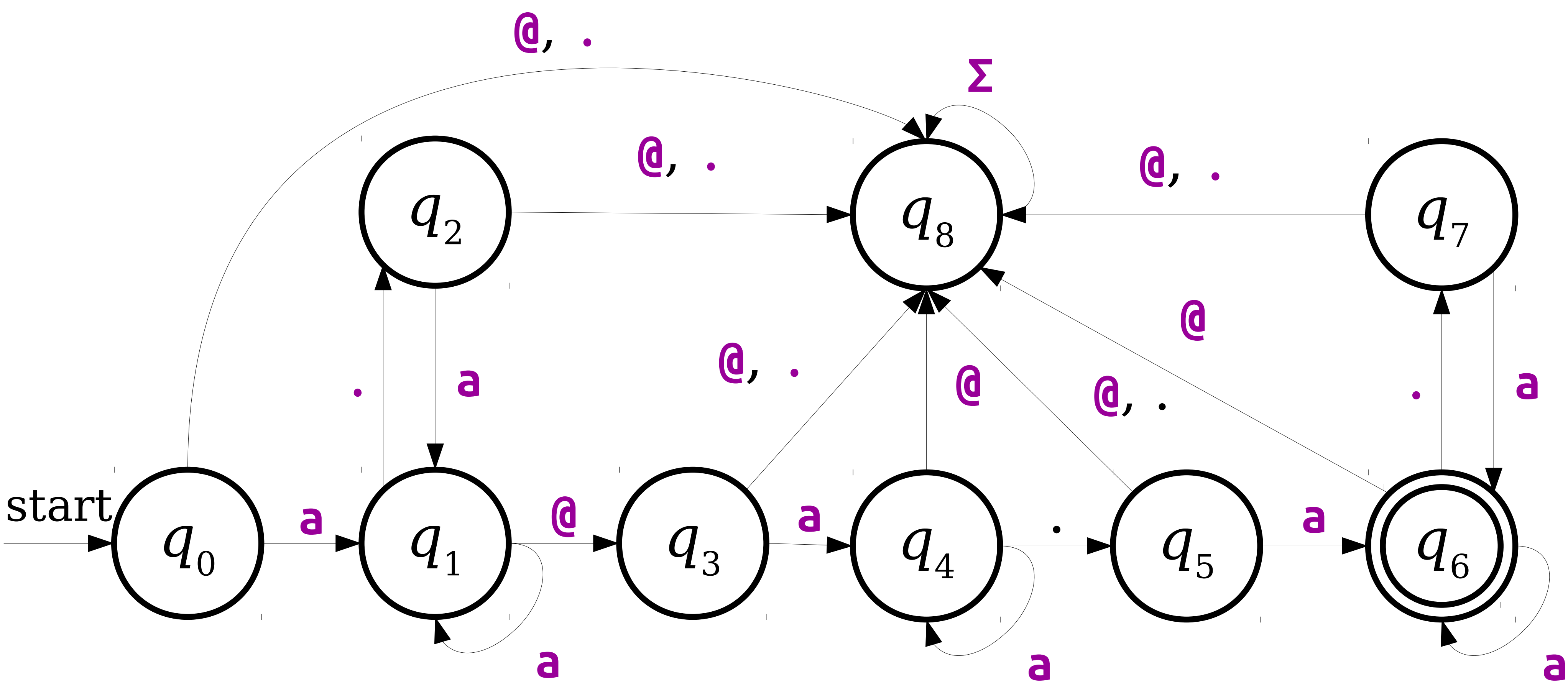
- Let Σ = { **a**, **.**, **@** }, where **a** represents "some letter."
- Let's make a regex for email addresses.

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

cs103@cs.stanford.edu
first.middle.last@mail.site.org
dot.at@dot.com

For Comparison

a⁺(.a⁺)*@a⁺(.a⁺)⁺



Shorthand Summary

- R^n is shorthand for $RR \dots R$ (n times).
  - Edge case: define $R^0 = \varepsilon$.
- Σ is shorthand for "any character in Σ."
- R? is shorthand for ($R \cup \varepsilon$), meaning "zero or one copies of R."
- R^+ is shorthand for RR^*, meaning "one or more copies of R."

Time-Out for Announcements!

Midterm Exam Logistics

- The next midterm is ***Monday, February 26th*** from ***7:00PM – 10:00PM***. Locations are divvied up by last (family) name:
  - `A-I`: Go to ***Cubberley Auditorium***.
  - `J-Z`: Go to ***Cemex Auditorium***.
- The exam focuses on Lecture 06 – 13 (binary relations through induction) and PS3 – PS5. Finite automata onward is *not* tested.
  - Topics from earlier in the quarter (proofwriting, first-order logic, set theory, etc.) are also fair game, but that's primarily because the later material builds on this earlier material.
- The exam is closed-book, closed-computer, and limited-note. You can bring a double-sided, 8.5" × 11" sheet of notes with you to the exam, decorated however you'd like.
- Students with OAE accommodations: please contact us ***immediately*** if you haven't yet done so. We'll ping you about setting up alternate exams.

Practice Midterm Exam

- We'll be holding a practice midterm exam ***tonight*** from ***7PM – 10PM*** in ***320-105***.
- The practice midterm exam is composed of what we think is a good representative sample of older midterm questions from across the years. It’s probably the best indicator of what you should expect to see.
- Course staff will be on hand to answer your questions.
- Can't make it? We'll release the practice exam and solutions online. Set up your own practice exam time with a small group and work through it under realistic conditions!

Other Practice Materials

- We’ve posted four practice midterms to the course website, with solutions.
  - We’ll post the practice exam from this evening a little bit later, bringing the total to five.
- There’s also Extra Practice Problems 2, plus all the CS103A materials.
- Need more practice? Let us know and we’ll see what we can do!

Problem Sets

- Problem Set Five solutions are now out.
  - Please read over them – there's a lot of good stuff in there!
  - We'll get PS5 graded and returned as soon as we can.
- Problem Set Six is out and is due this Friday at 2:30PM.
  - ***Be careful about using late days here***, since the exam is on Monday.

Back to CS103!

The Power of Regular Expressions

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

Proof idea: Use induction!

- The atomic regular expressions all represent regular languages.
- The combination steps represent closure properties.
- So anything you can make from them must be regular!

Thompson’s Algorithm

- In practice, many regex matchers use an algorithm called ***Thompson's algorithm*** to convert regular expressions into NFAs (and, from there, to DFAs).
  - Read Sipser if you’re curious!
- ***Fun fact:*** the “Thompson” here is Ken Thompson, one of the co-inventors of Unix!

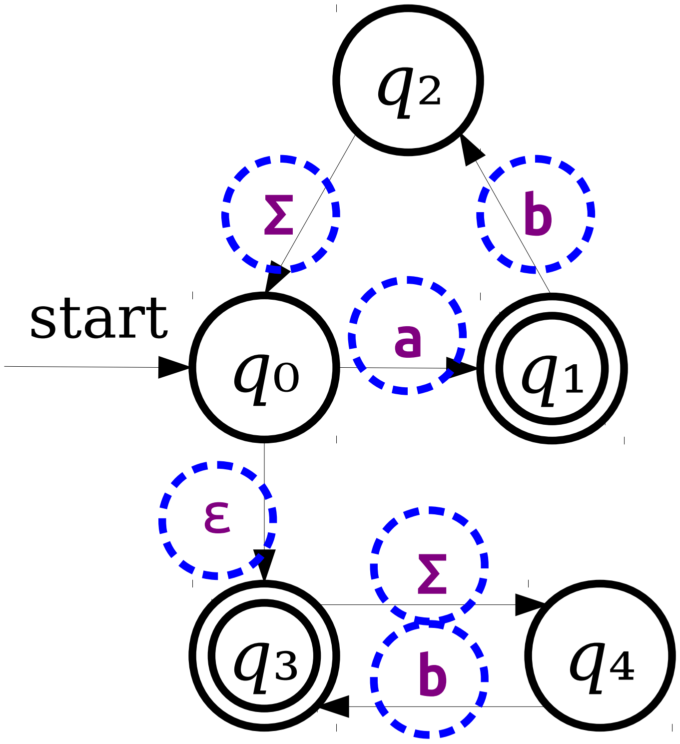
The Power of Regular Expressions

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

This is not obvious!

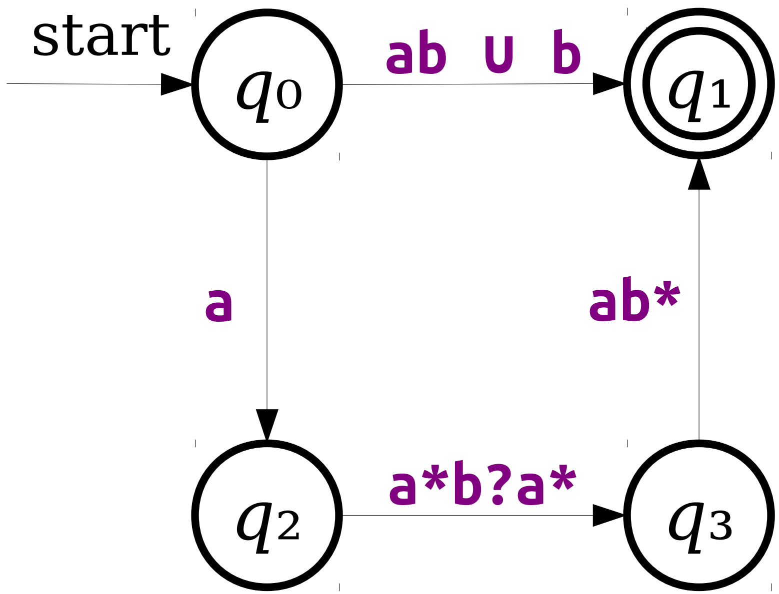
Proof idea: Show how to convert an arbitrary NFA into a regular expression.

Generalizing NFAs



These are all regular expressions!

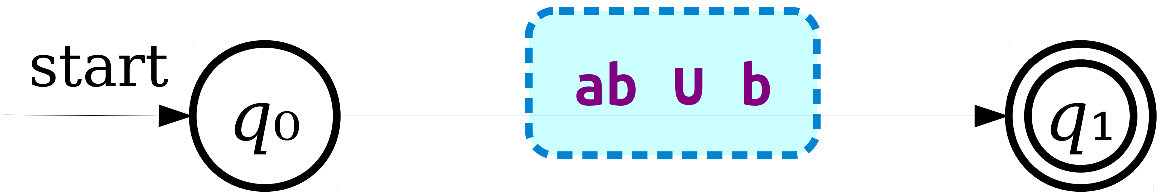
Generalizing NFAs



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

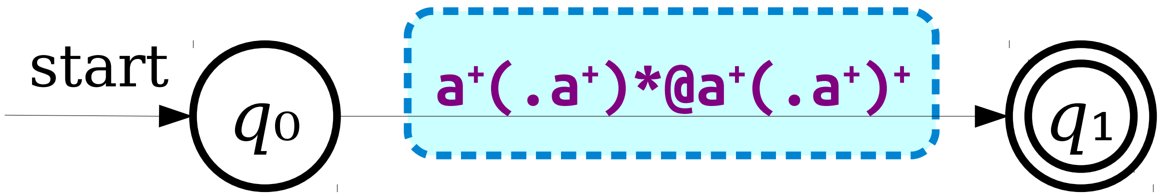
Key Idea 1: Imagine that we can label transitions in an NFA with arbitrary regular expressions.

Generalizing NFAs



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

Generalizing NFAs

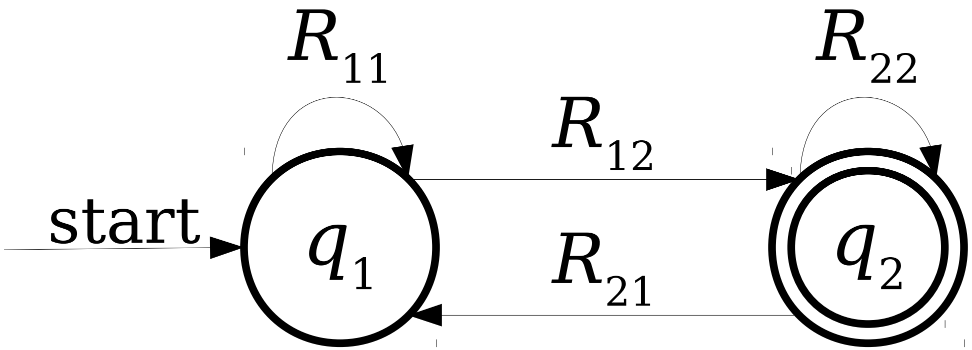


Is there a simple regular expression for the language of this generalized NFA?

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

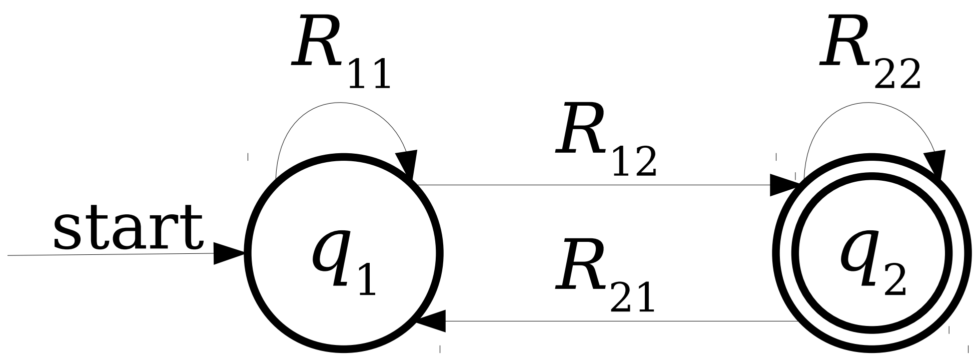
…then we can easily read off a regular expression for the original NFA.

From NFAs to Regular Expressions



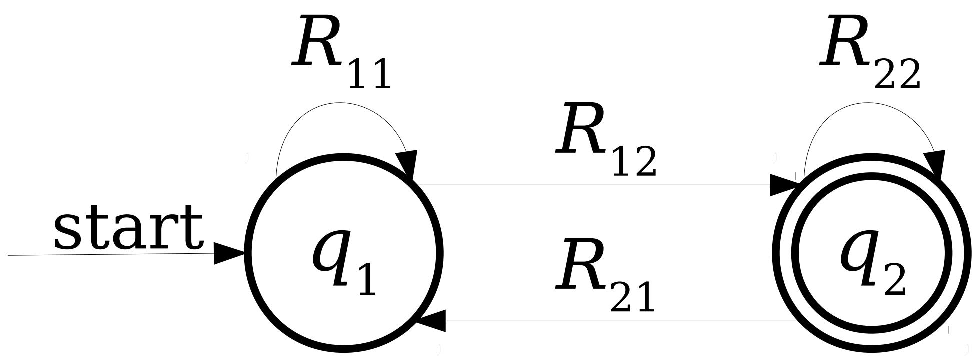
Here, R_{11}, R_{12}, R_{21}, and R_{22} are arbitrary regular expressions.

From NFAs to Regular Expressions



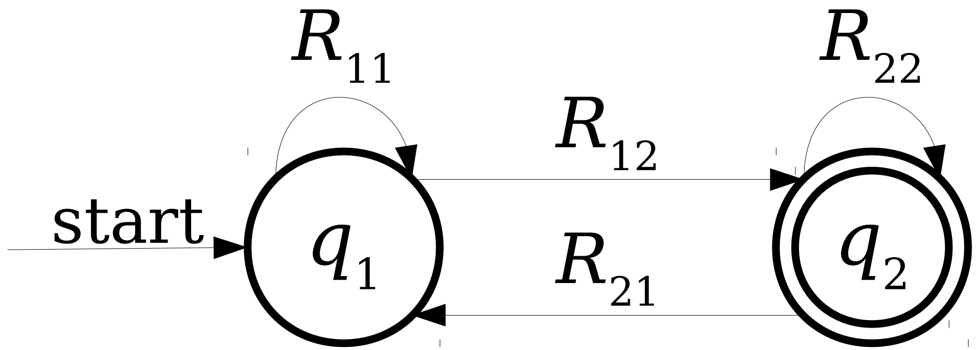
Question: Can we get a clean regular expression from this NFA?

From NFAs to Regular Expressions



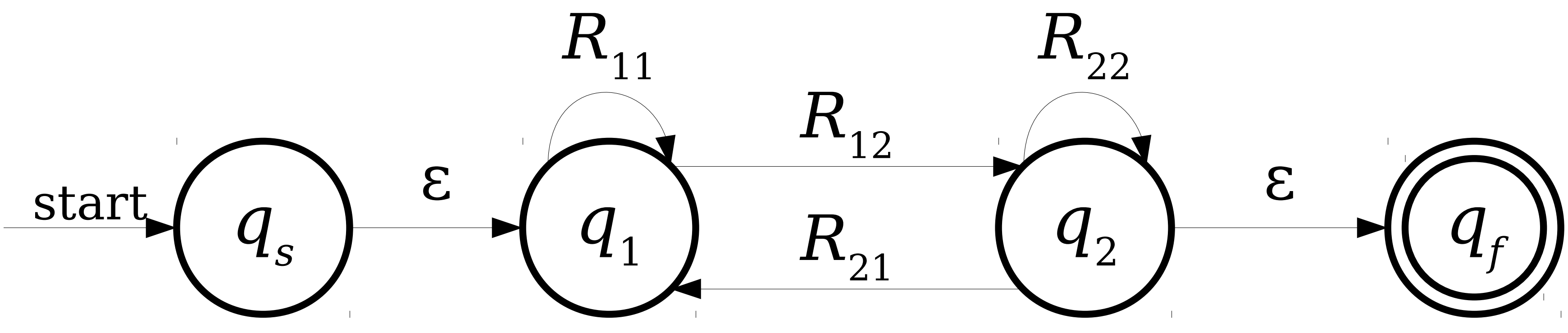
Key Idea 3: Somehow transform this NFA so that it looks like this:

From NFAs to Regular Expressions

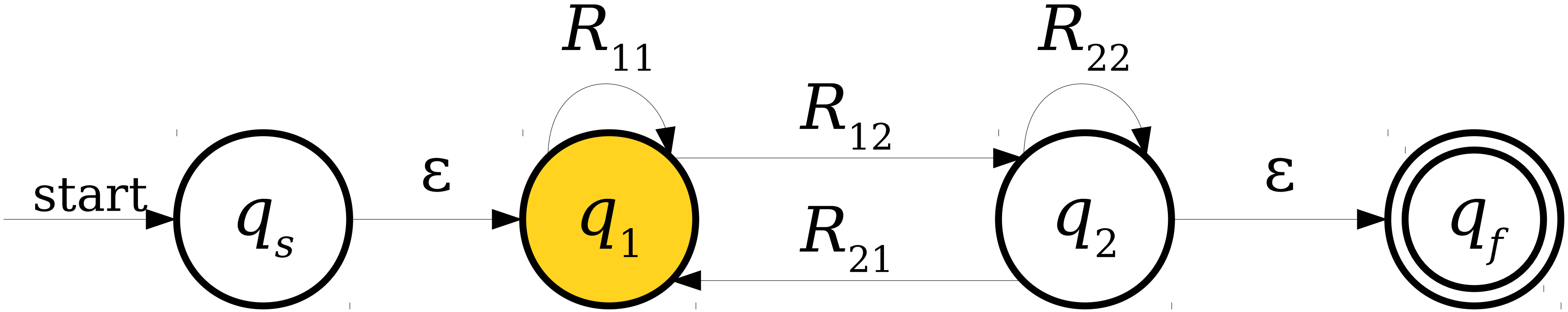


The first step is going to be a bit weird...

From NFAs to Regular Expressions

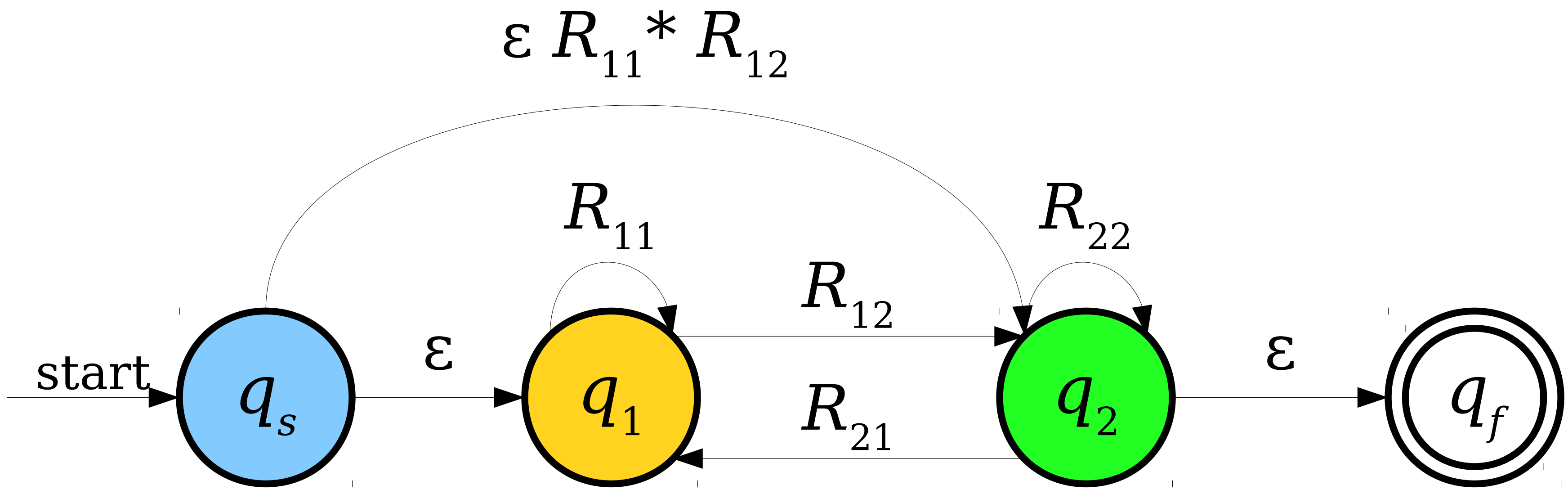


From NFAs to Regular Expressions



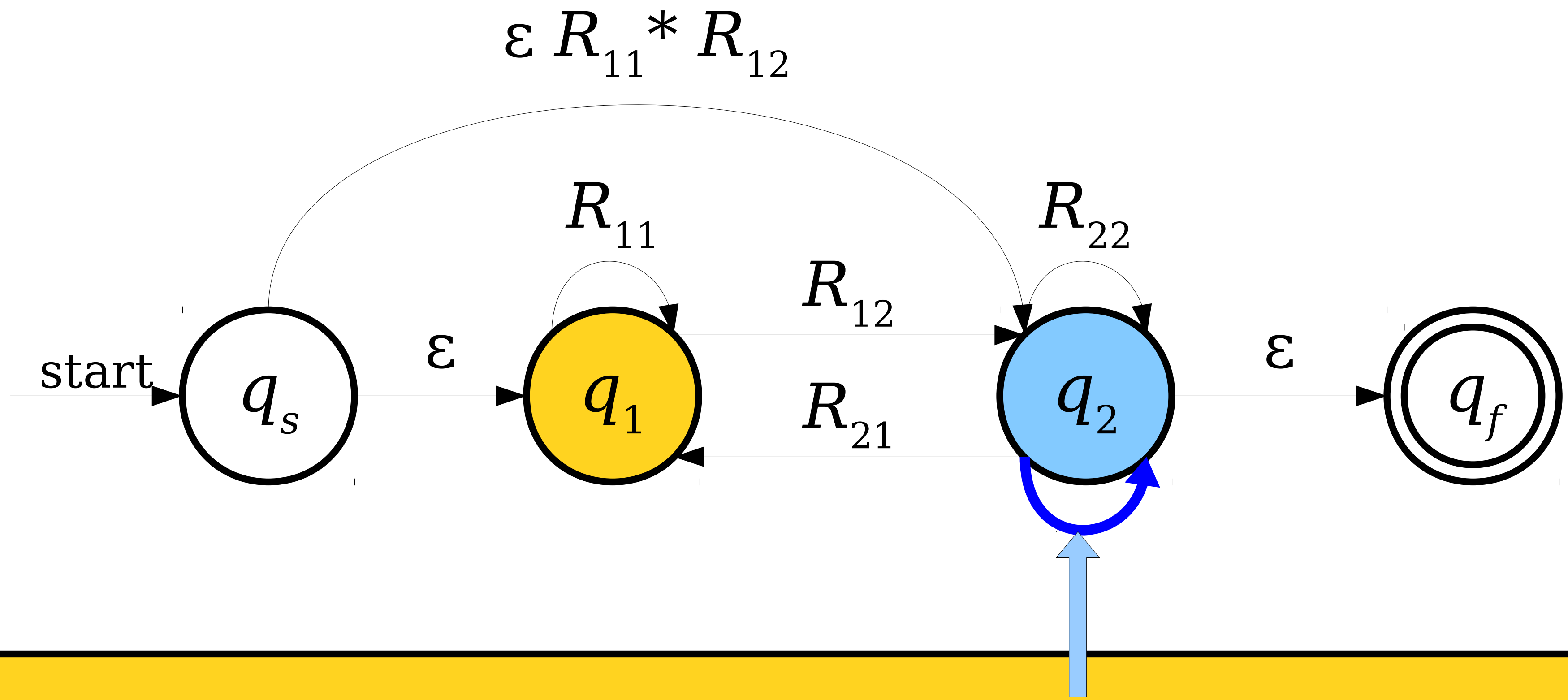
Could we eliminate this state from the NFA?

From NFAs to Regular Expressions



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

From NFAs to Regular Expressions

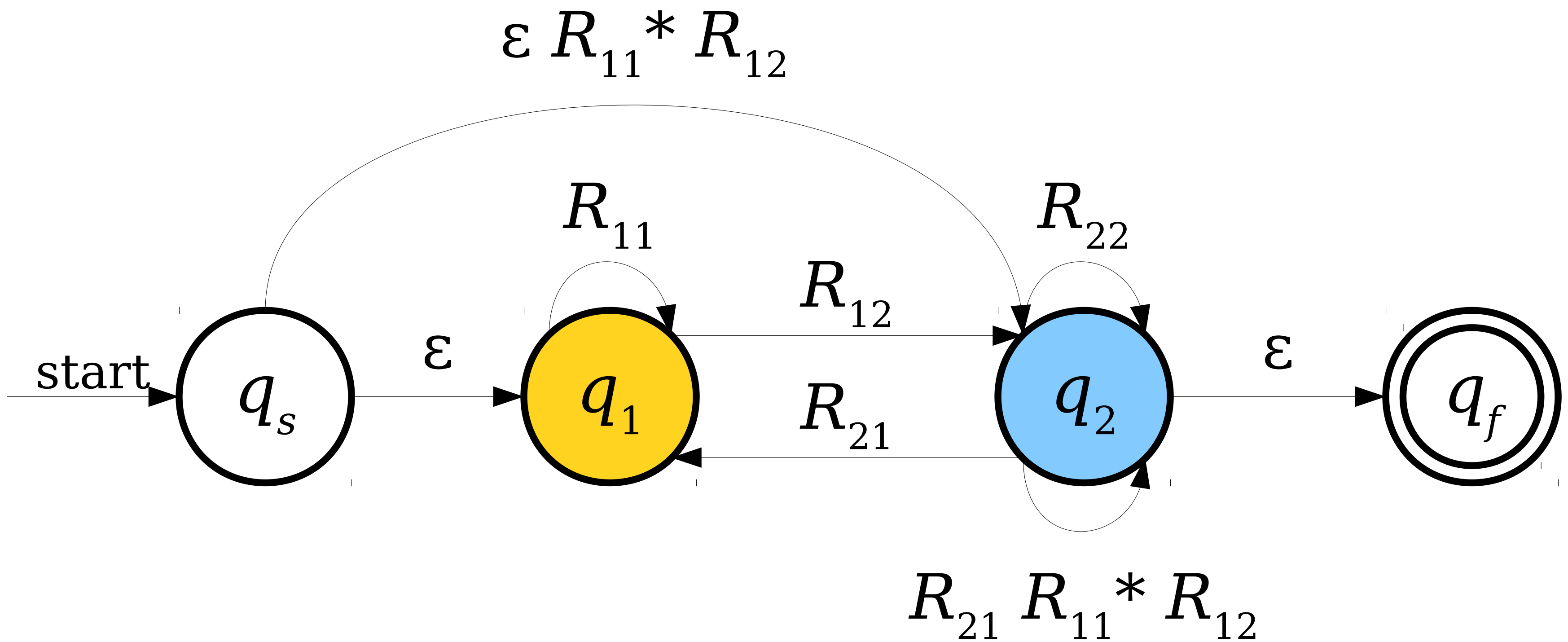


What regex should go on this edge?

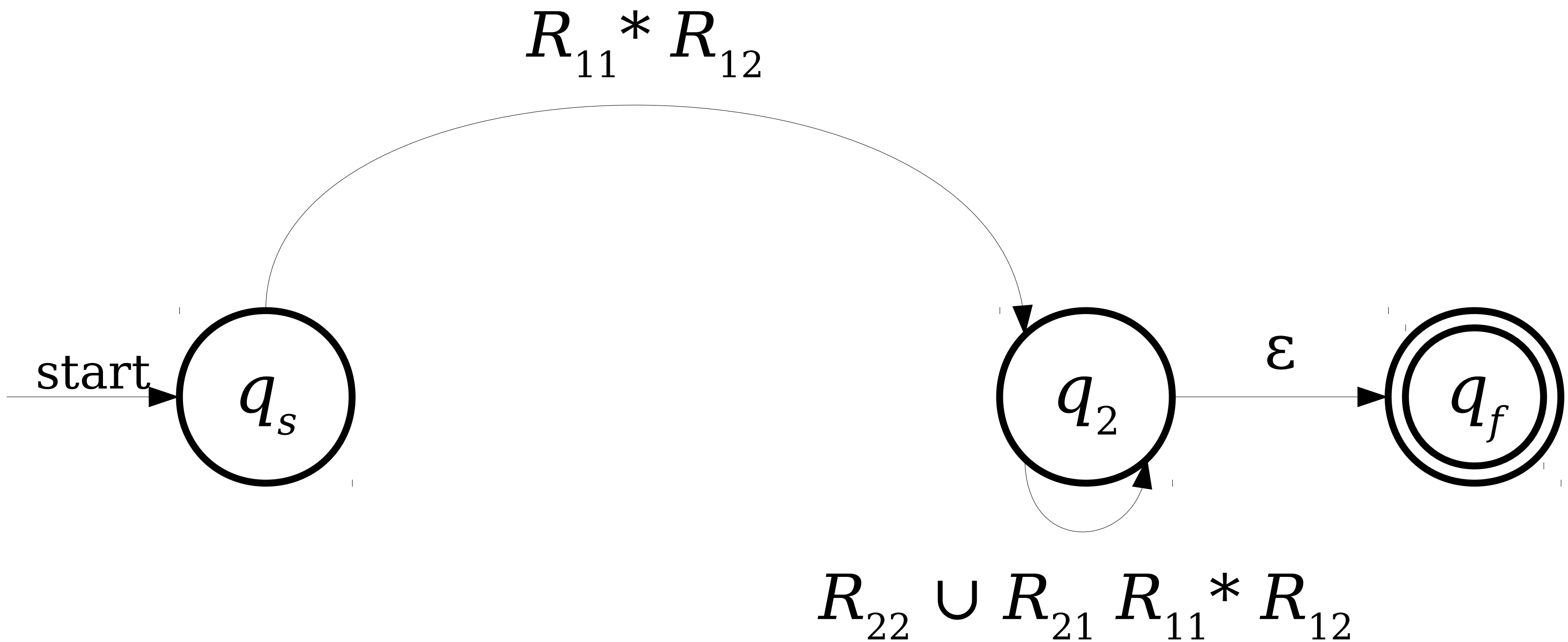
A. $R_{12}\ R_{21}$ ***B.*** $R_{12}\ R_{22}^*\ R_{21}$ ***C.*** $R_{21}\ R_{12}$ ***D.*** $R_{21}\ R_{11}^*\ R_{12}$

Answer at **PollEv.com/cs103** or
text **CS103** to **22333** once to join, then ***A***, ***B***, ***C***, or ***D***.

From NFAs to Regular Expressions

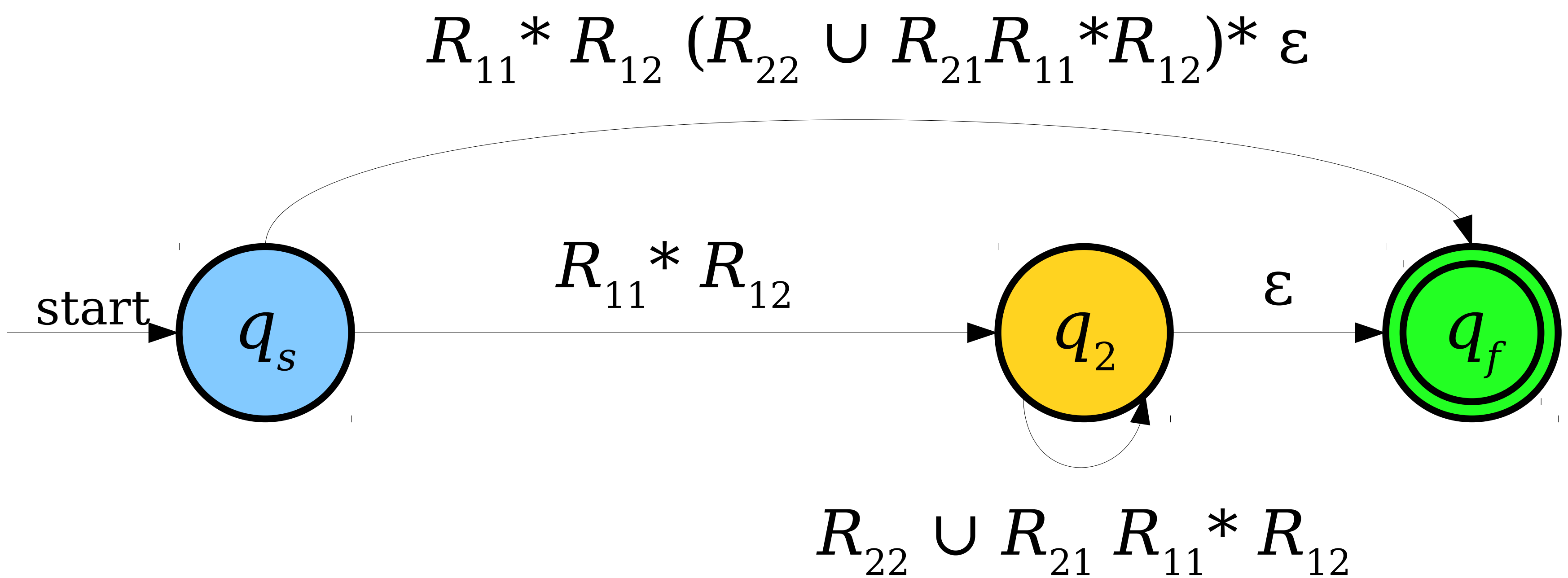


From NFAs to Regular Expressions

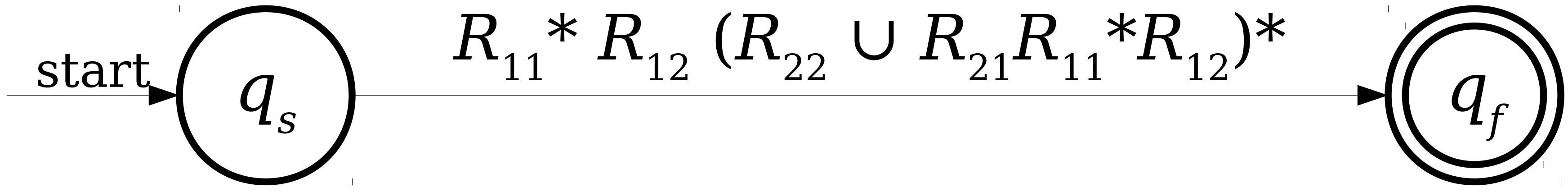


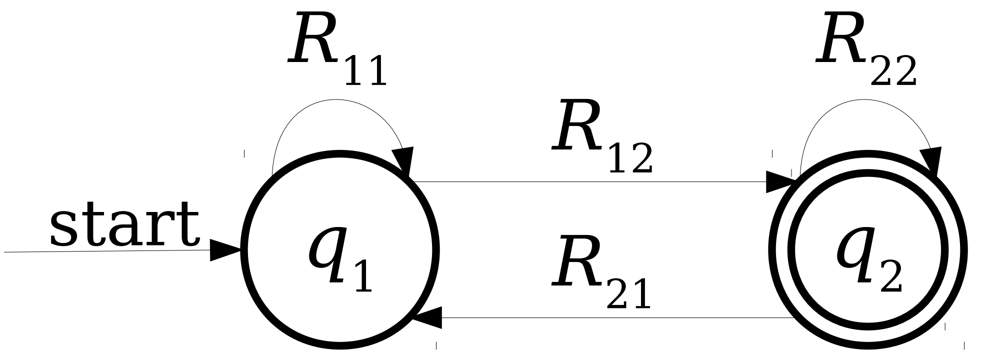
Note: We're using union to combine these transitions together.

From NFAs to Regular Expressions



From NFAs to Regular Expressions





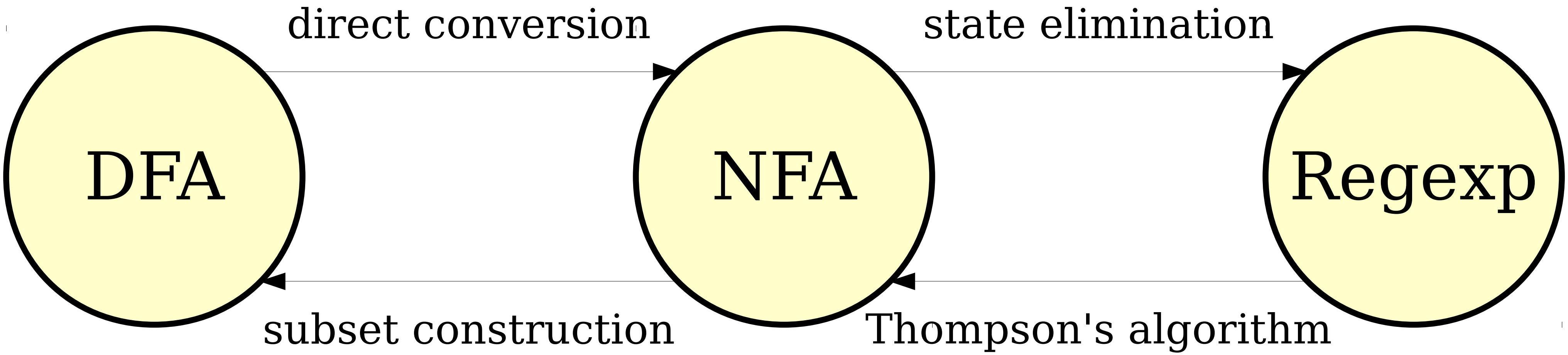
The Construction at a Glance

- 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.

Eliminating a State

- 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, …, R_k, replace them with a single transition labeled $R_1 \cup R_2 \cup \ldots \cup R_k$.

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 Matters

- The equivalence of regular expressions and finite automata has practical relevance.
  - Tools like `grep` and `flex` that use regular expressions capture all the power available via DFAs and NFAs.
- This also is hugely theoretically significant: the regular languages can be assembled “from scratch” using a small number of operations!

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.