CS103
Winter 2026

Lecture 17:

Regular Expressions

Regular Expressions

1. **Recap from Last Time**
2. Devices for Articulating Regular Languages
3. Regular Expressions
4. Designing Regular Expressions
5. Shorthand Summary
6. Regular Expressions and Regular Languages
7. State Elimination
8. Closing Thoughts
9. Action Items and What's Next?

Regular Languages

- A language L is called a ***regular language*** if there is a DFA or an NFA for L.
- ***Theorem:*** The following are equivalent:
  - L is a regular language.
  - There is a DFA D where $\mathscr{L}(D) = L$.
  - There is an NFA N where $\mathscr{L}(N) = L$.
- In other words, knowing any one of the above three facts means you know the other two.

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

$$L_1L_2 = \{ x \mid \exists w_1 \in L_1.\ \exists w_2 \in L_2.\ x = w_1w_2 \}$$

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

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

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:

$$L^0 = \{\varepsilon\} \qquad L^{n+1} = LL^n$$

- So, for example, { **aa**, **b** }^{3} is the language

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

The Kleene Closure

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

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

- Mathematically:

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

- 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 rubber stamps – one for each string in L – and you form every possible string that can be made from those stamps.

Closure Properties

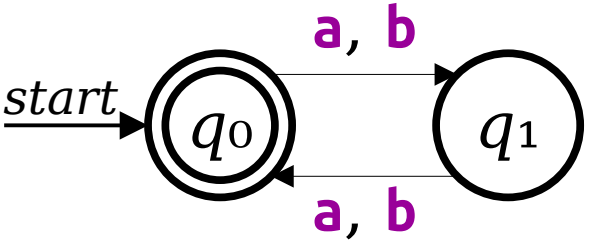
- ***Theorem:*** If L_1 and L_2 are regular languages over an alphabet Σ, then so are the following languages:
  - $L_1 \cup L_2$
  - $L_1 L_2$
  - L_1^*
- These (and other) properties are called ***closure properties of the regular languages***.

Regular Expressions

1. Recap from Last Time
2. **Devices for Articulating Regular Languages**
3. Regular Expressions
4. Designing Regular Expressions
5. Shorthand Summary
6. Regular Expressions and Regular Languages
7. State Elimination
8. Closing Thoughts
9. Action Items and What's Next?

Devices for Articulating Regular Languages

- State Transition Diagram



- Set (or other Mathematical) Notation

{ *w* ∈ Σ* | *w*'s length is even }

- State Transition Table

	a	b
q_0	q_1	q_1
q_1	q_0	q_0

- ***New!*** Regular Expressions

Devices for Articulating Regular Languages

- State Transition Diagram

- Set (or other Mathematical) Notation

{ *w* ∈ Σ* | *w*'s length is even }

- State Transition

- ***New!*** Regular

Note: This one is <u>not unique</u> to regular languages! We can express non-regular languages with set builder notation, as well. In contrast, having a DFA or NFA for a language means it's certainly regular.

Regular Expressions

1. Recap from Last Time
2. Devices for Articulating Regular Languages
3. **Regular Expressions**
4. Designing Regular Expressions
5. Shorthand Summary
6. Regular Expressions and Regular Languages
7. State Elimination
8. Closing Thoughts
9. Action Items and What's Next?

Regular Expressions

- ***Regular expressions*** are a way of describing a language via a string representation.
- They're used just about everywhere:
  - They're built into the JavaScript language and used for data validation.
  - They're used in the UNIX `grep` and `flex` tools to search files and build compilers.
  - They're employed to clean and scrape data for large-scale analysis projects.
- Conceptually, regular expressions are strings describing how to assemble a larger language out of smaller pieces.

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.
- *This is a bottom-up approach to the regular languages.*

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:

$$(R)$$

$$R*$$

$$R_1 R_2$$

$$R_1 \cup R_2$$

- So **ab*c∪d** is parsed as **((a(b*))c)∪d**

Regular Expression Examples

- The regular expression `thisUthat` represents the language

  { `this`, `that` }.

- The regular expression `brrr*` represents the regular language

  { `brr`, `brrr`, `brrrr`, … }.

- The regular expression `splish!(splish!)*` represents the regular language

  { `splish!`, `splish!splish!`, `splish!splish!splish!`

Regular Expressions, Formally

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

Worthwhile activity: Apply this recursive definition to

a(b∪c)((d))

and see what you get.

Regular Expressions

1. Recap from Last Time
2. Devices for Articulating Regular Languages
3. Regular Expressions
4. **Designing Regular Expressions**
5. Shorthand Summary
6. Regular Expressions and Regular Languages
7. State Elimination
8. Closing Thoughts
9. Action Items and What's Next?

Designing Regular Expressions

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

(a ∪ b)*aa(a ∪ b)*

bbabbbaabab
aaaa
bbbbbabbbbaabbbbb

Designing Regular Expressions

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

ΣΣΣΣ

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

Here are some candidate regular expressions for the language L. Which of these are correct?

`Σ*aΣ*`

`b*ab* U b*`

`b*(a U ε)b*`

`b*a*b* U b*`

`b*(a* U ε)b*`

Answer at ***https://cs103.stanford.edu/pollev***

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 Σ = { **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.

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

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

A More Elaborate Design

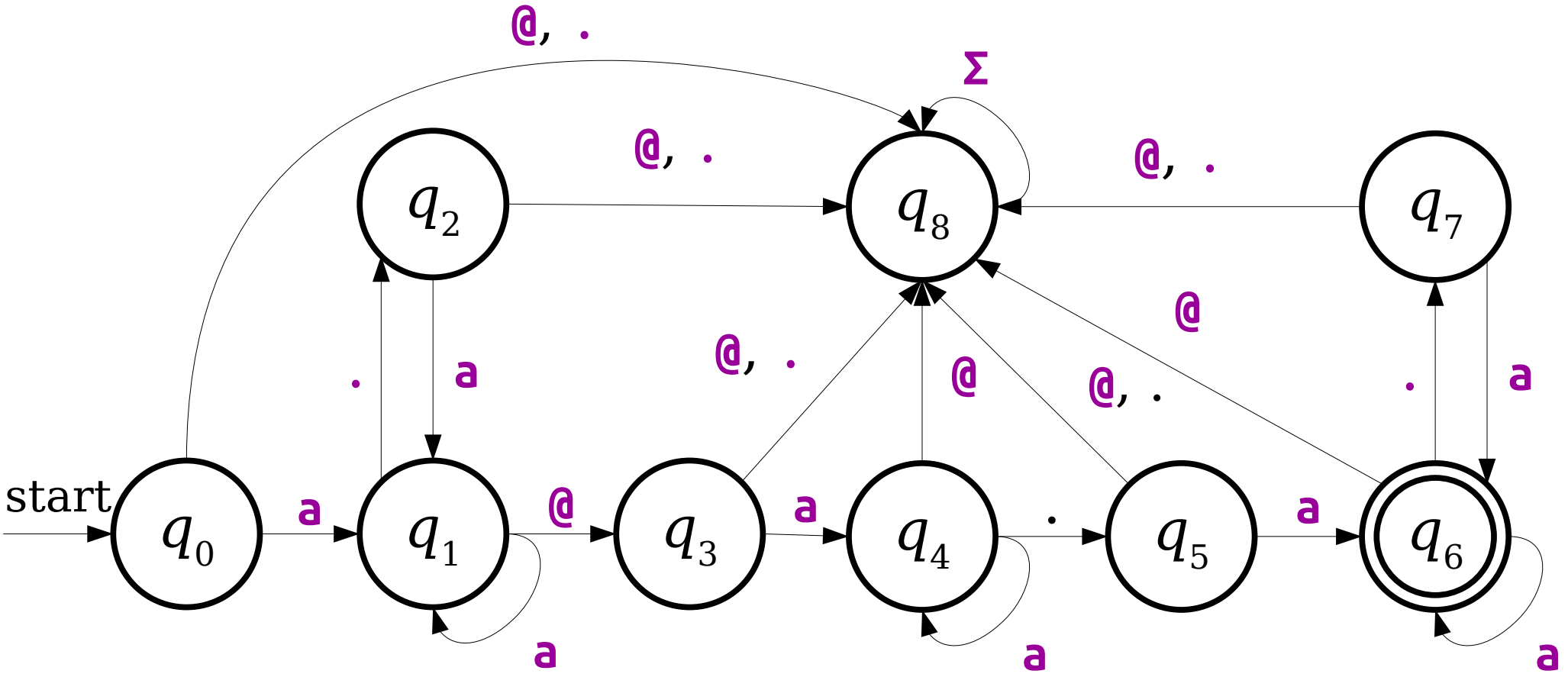
- Let Σ = { **a**, **.**, **@** }, where **a** represents "some letter."
- Let's make a regex for email addresses.

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

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

For Comparison

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



Regular Expressions

1. Recap from Last Time
2. Devices for Articulating Regular Languages
3. Regular Expressions
4. Designing Regular Expressions
5. **Shorthand Summary**
6. Regular Expressions and Regular Languages
7. State Elimination
8. Closing Thoughts
9. Action Items and What's Next?

Shorthand Summary

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

Regular Expressions

1. Recap from Last Time
2. Devices for Articulating Regular Languages
3. Regular Expressions
4. Designing Regular Expressions
5. Shorthand Summary
6. **Regular Expressions and Regular Languages**
7. State Elimination
8. Closing Thoughts
9. Action Items and What's Next?

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.

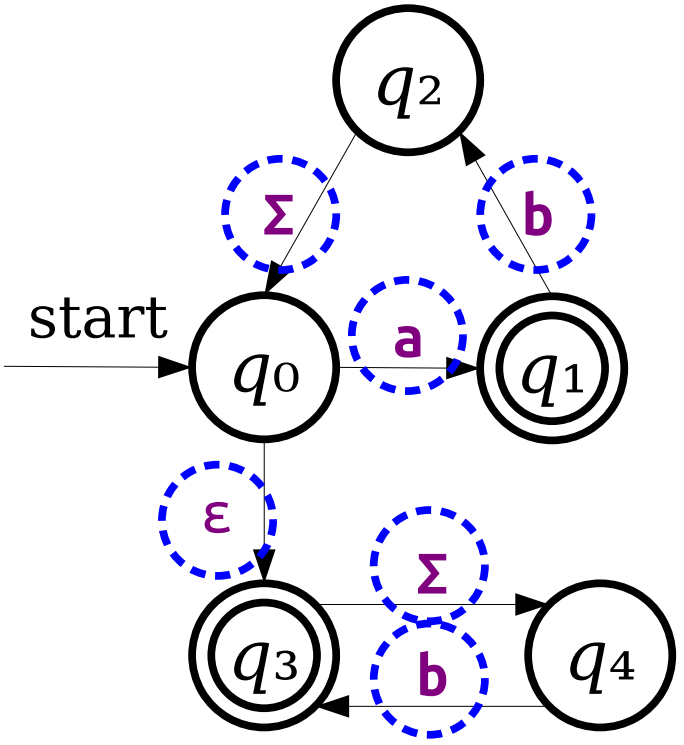
This is not obvious!

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

Regular Expressions

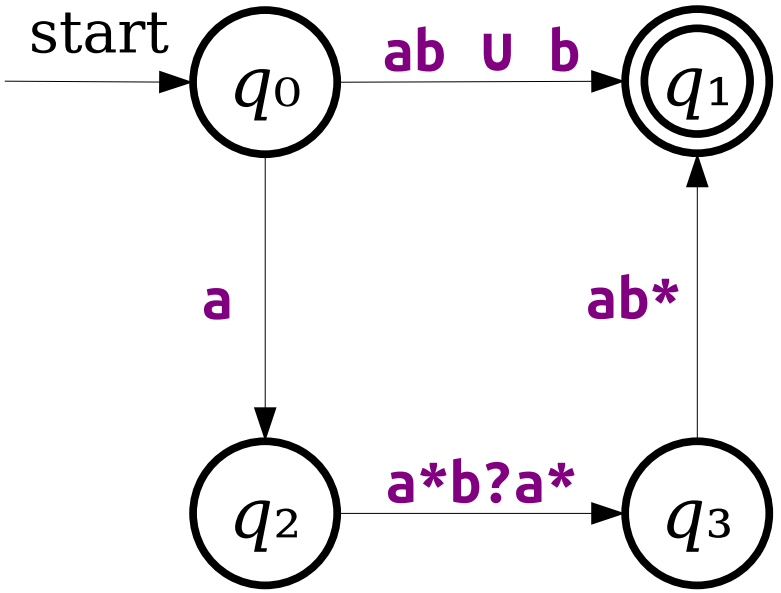
1. Recap from Last Time
2. Devices for Articulating Regular Languages
3. Regular Expressions
4. Designing Regular Expressions
5. Shorthand Summary
6. Regular Expressions and Regular Languages
7. **State Elimination**
8. Closing Thoughts
9. Action Items and What's Next?

Generalizing NFAs



These are all regular expressions!

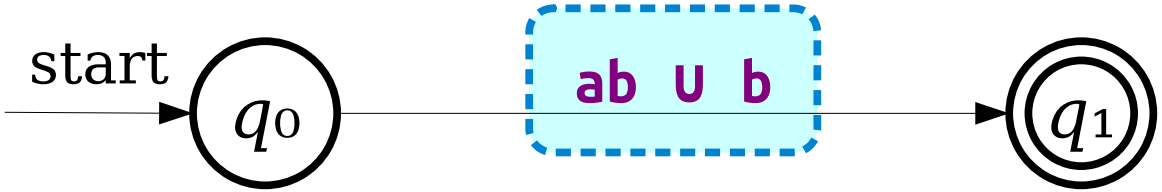
Generalizing NFAs



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

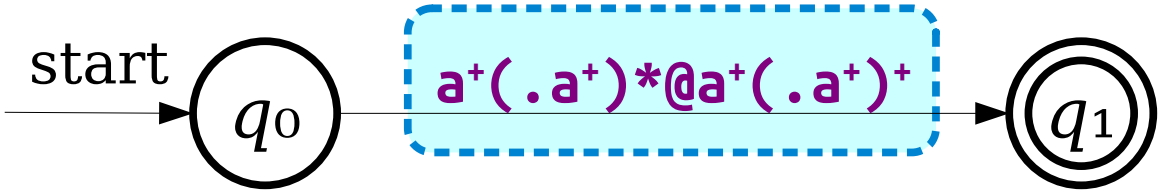
Key Idea 1: Imagine that we can label transitions in an NFA with arbitrary regular expressions.

Generalizing NFAs



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

Generalizing NFAs

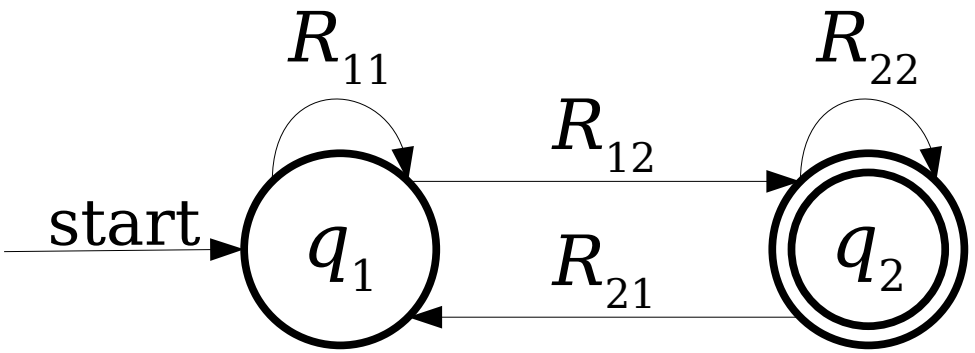


Is there a simple regular expression for the language of this generalized NFA?

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

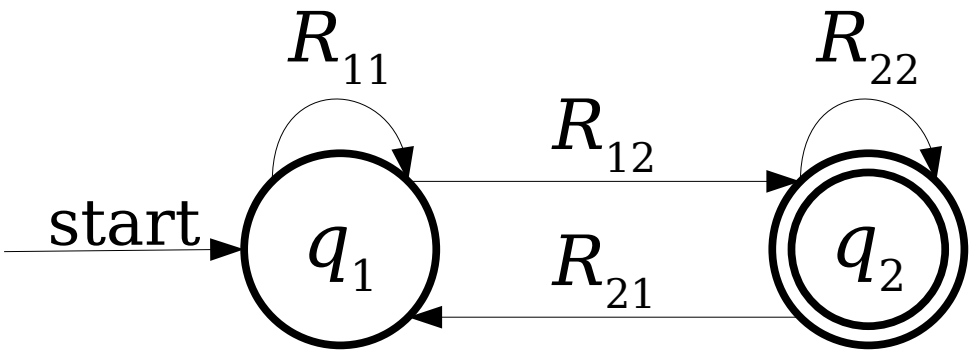
...then we can easily read off a regular expression for the original NFA.

From NFAs to Regular Expressions



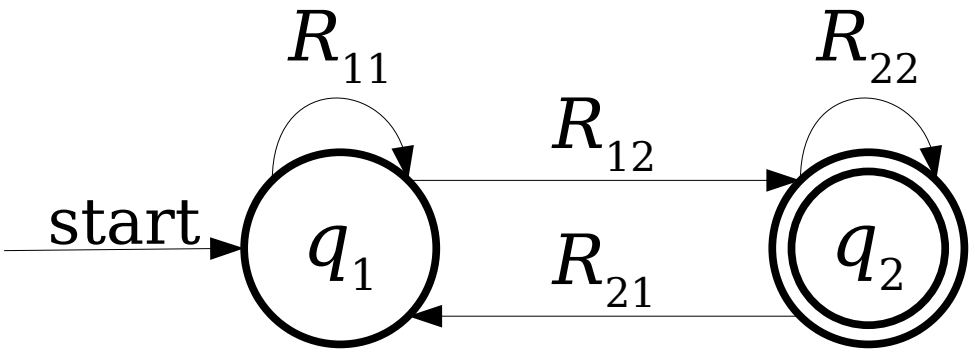
Here, R_{11}, R_{12}, R_{21}, and R_{22} are arbitrary regular expressions.

From NFAs to Regular Expressions



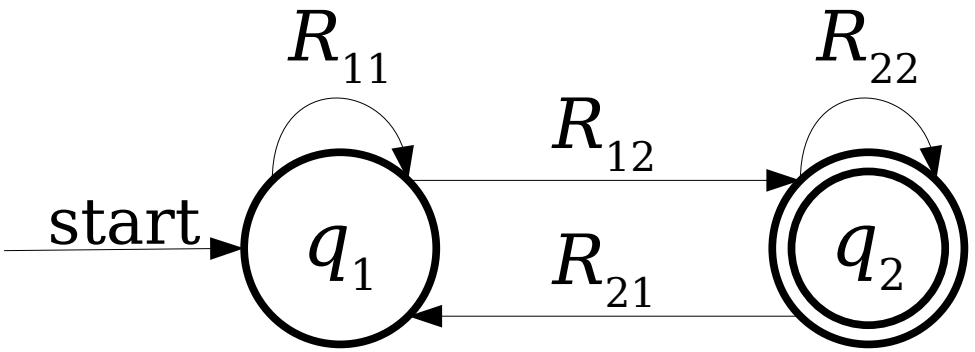
Question: Can we get a clean regular expression from this NFA?

From NFAs to Regular Expressions



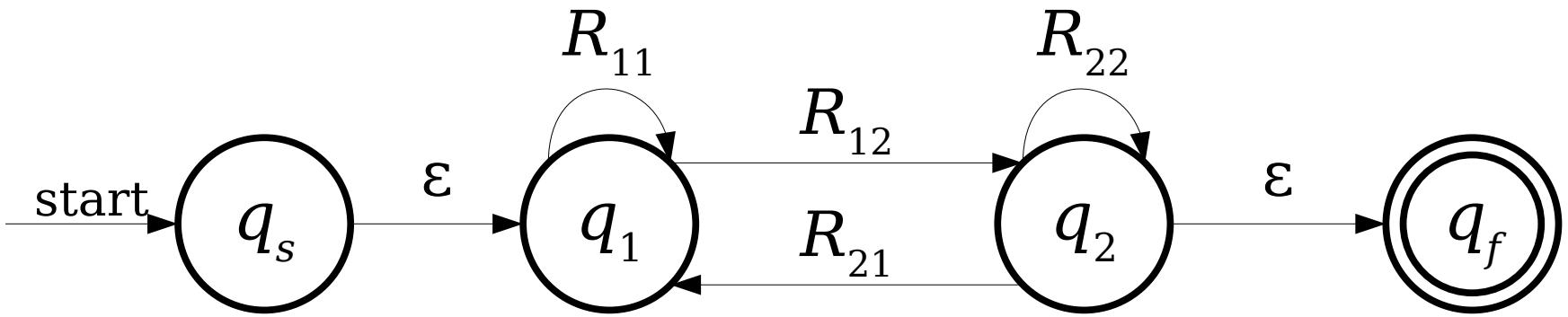
Key Idea 3: Somehow transform this NFA so that it looks like this:

From NFAs to Regular Expressions

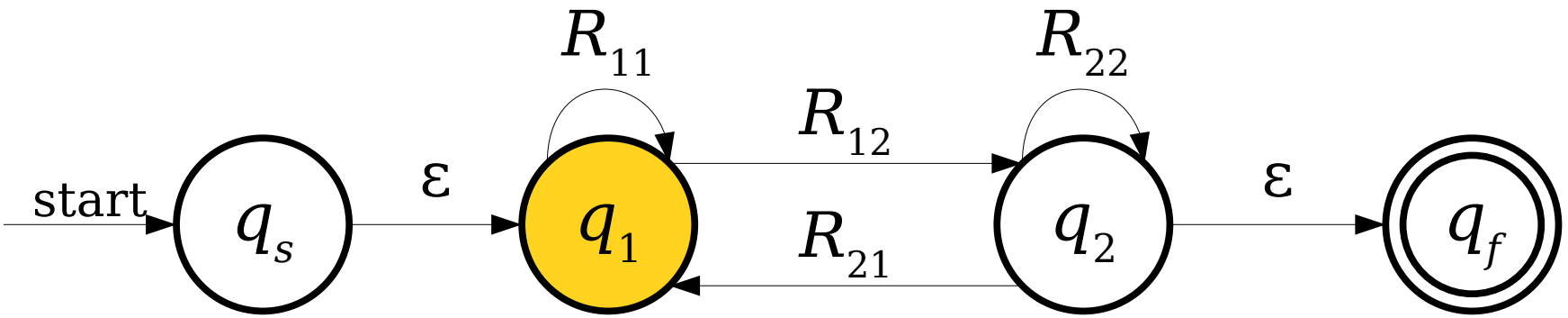


Here's the first step...

From NFAs to Regular Expressions

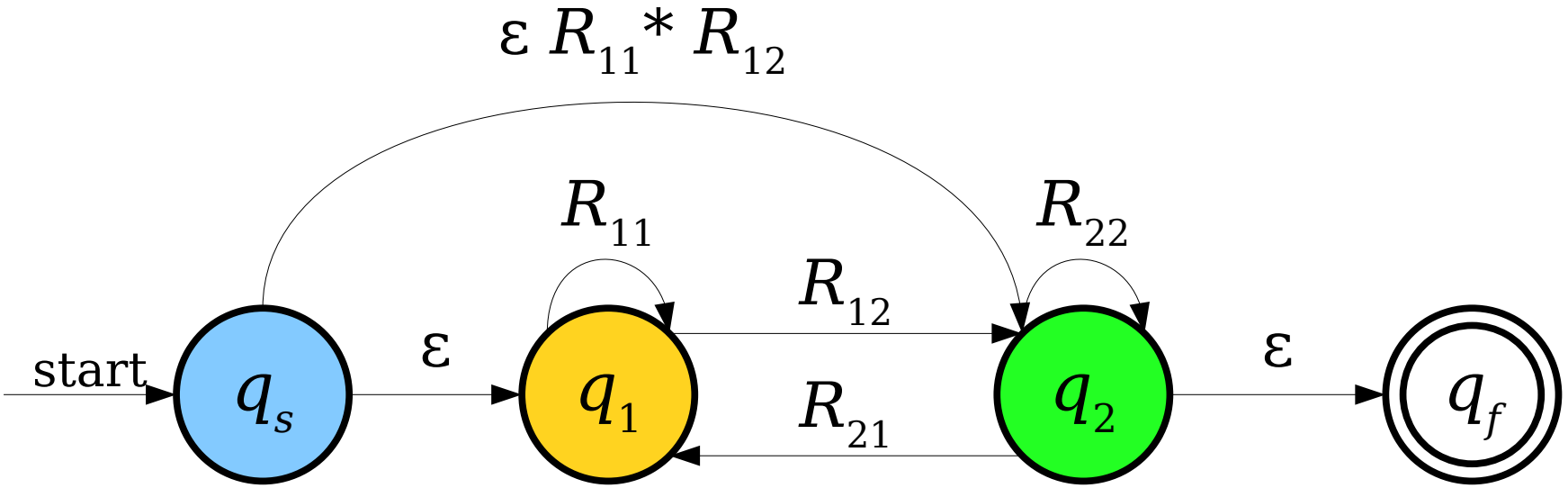


From NFAs to Regular Expressions



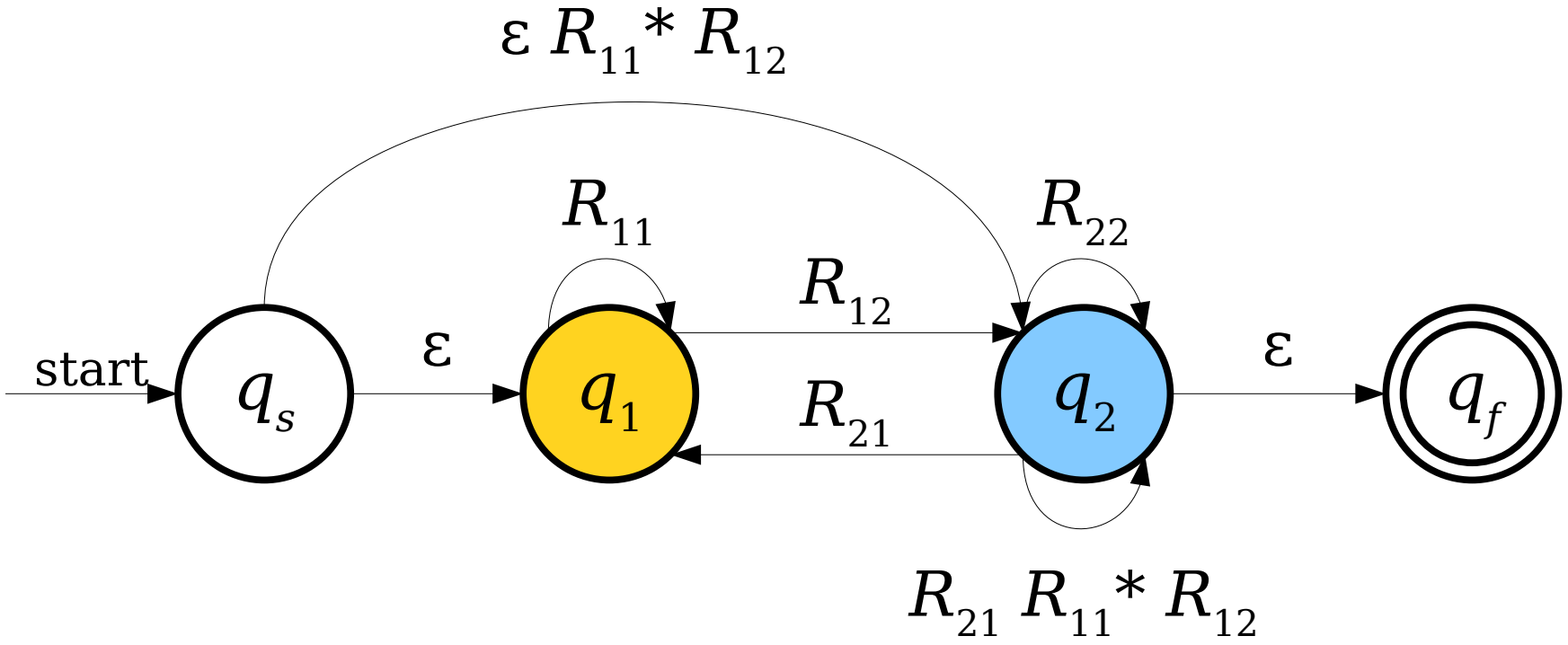
Could we eliminate this state from the NFA?

From NFAs to Regular Expressions

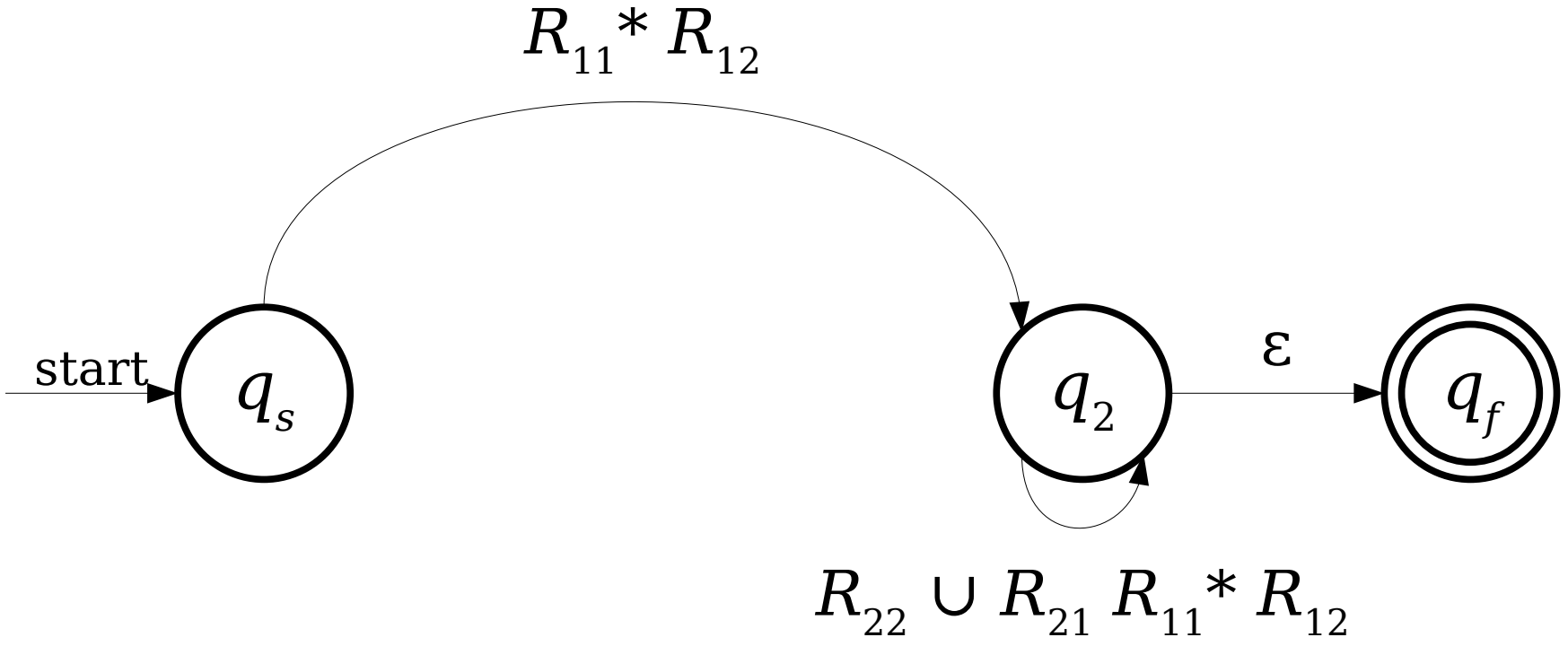


Note: We're using concatenation and Kleene closure in order to skip this state.

From NFAs to Regular Expressions

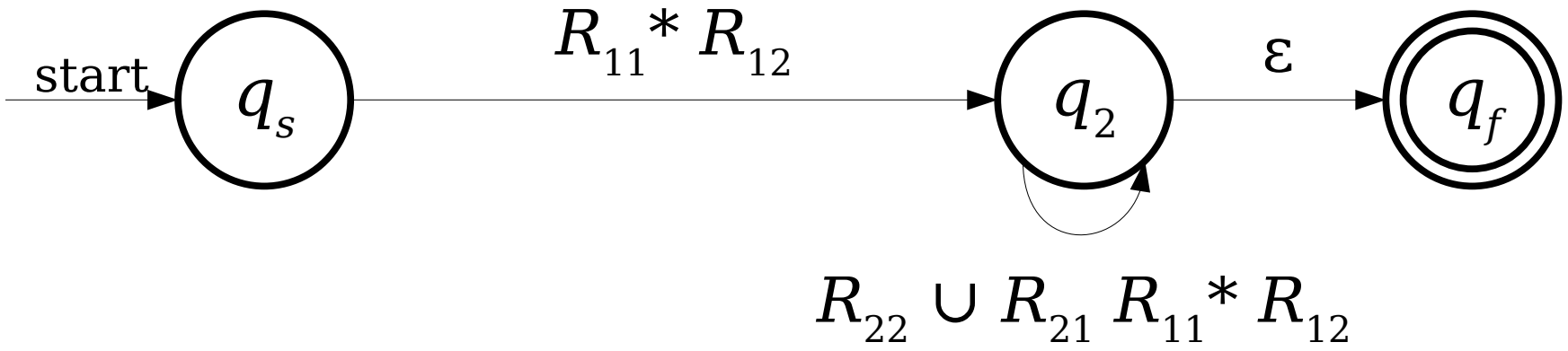


From NFAs to Regular Expressions

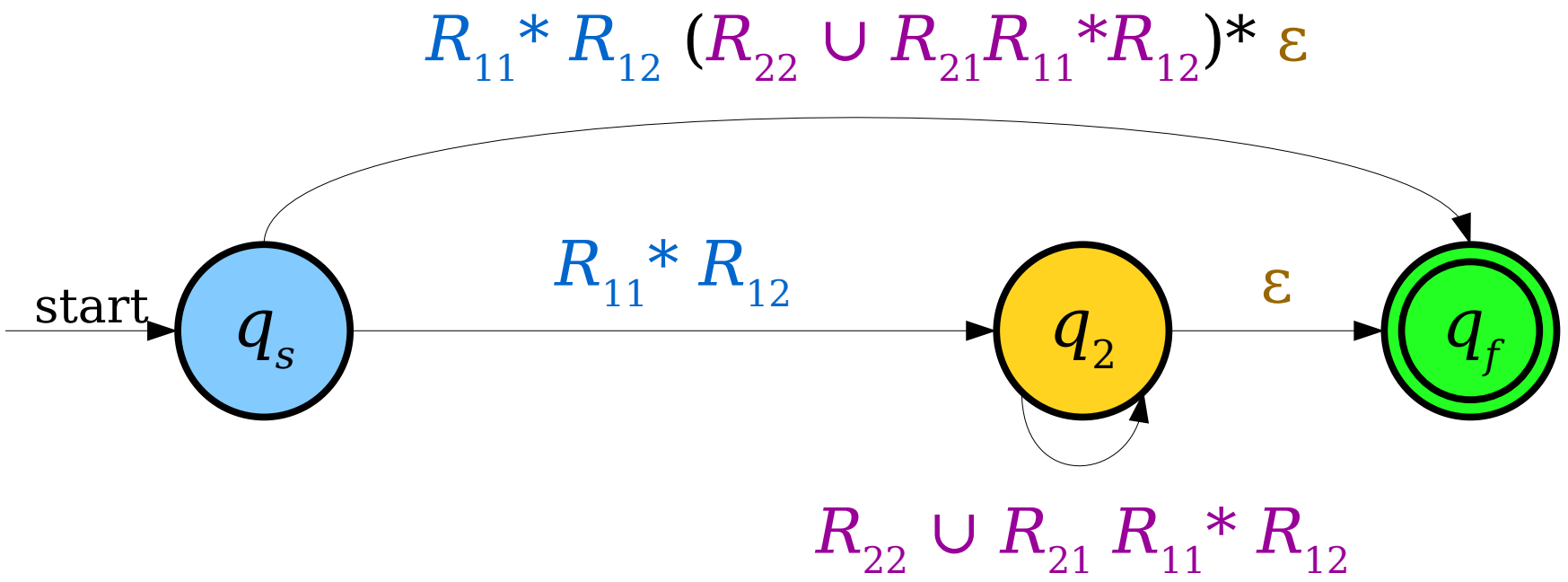


Note: We're using union to combine these transitions together.

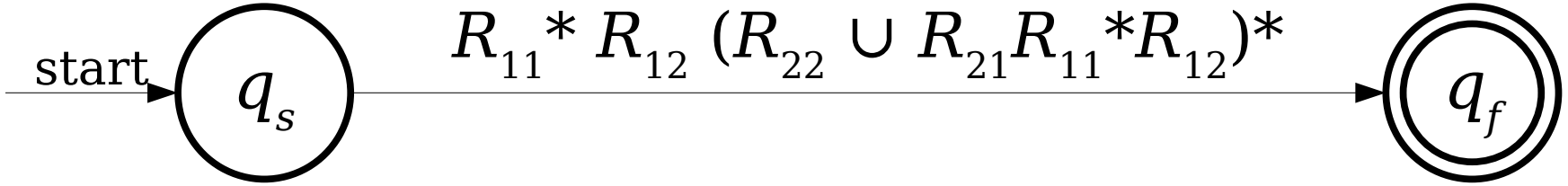
From NFAs to Regular Expressions

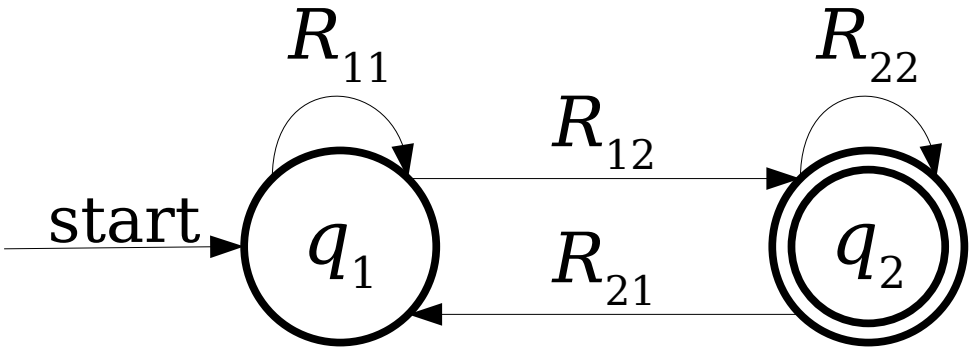


From NFAs to Regular Expressions



From NFAs to Regular Expressions





The 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.

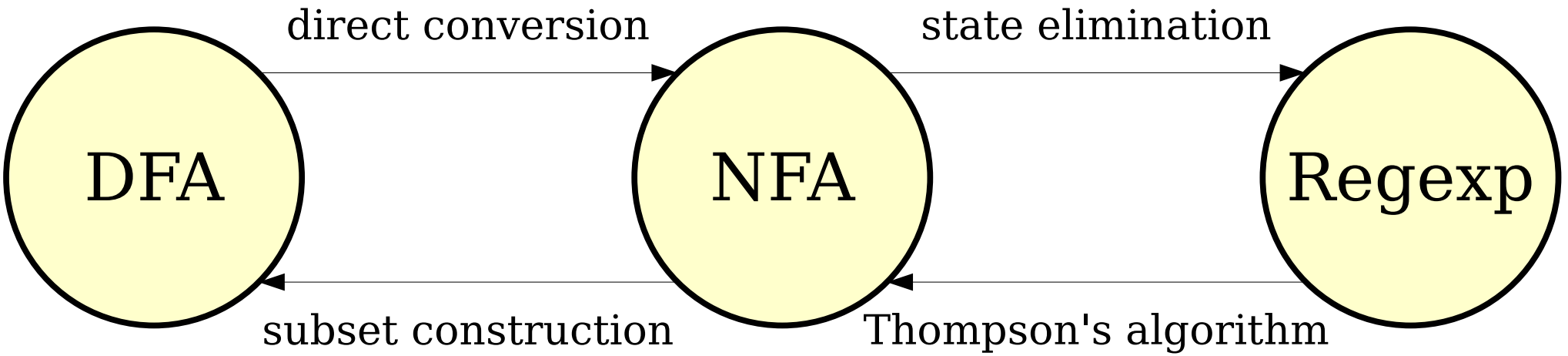
The State-Elimination Algorithm

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

Regular Expressions

1. Recap from Last Time
2. Devices for Articulating Regular Languages
3. Regular Expressions
4. Designing Regular Expressions
5. Shorthand Summary
6. Regular Expressions and Regular Languages
7. State Elimination
8. **Closing Thoughts**
9. Action Items and What's Next?

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.
  - 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!

Regular Expressions

1. Recap from Last Time
2. Devices for Articulating Regular Languages
3. Regular Expressions
4. Designing Regular Expressions
5. Shorthand Summary
6. Regular Expressions and Regular Languages
7. State Elimination
8. Closing Thoughts
9. **Action Items and What's Next?**

Your Action Items

- ***Read "Guide to Regexes"***
  - There's a lot of information and advice there about how to write regular expressions, plus a bunch of worked exercises.
- ***Read "Guide to State Elimination"***
  - It's a beautiful algorithm. The Guide goes into a lot more detail than what we did here.

Next Time

- ***Intuiting Regular Languages***
  - What makes a language regular?
- ***The Myhill-Nerode Theorem***
  - The limits of regular languages.