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

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

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

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

Language Exponentiation

- If L is a language over Σ, the language L^n is the concatenation of n copies of L with itself.
  - Special case: $L^0 = \{\varepsilon\}$.
- The ***Kleene closure*** of a language L, denoted L^*, is defined as

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

- Intuitively, all strings that can be formed by concatenating any number of strings in L with one another.
- Example: if L = { a, bb }, then

L^* = { ε, a, bb, aa, abb, bba, bbbb, aaa, aabb, abba, abbbb, bbaa, bbabb, bbbba, bbbbbb, … }

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 is regular.
  - Construct a DFA for it.
  - Construct an NFA for it.
  - Apply closure properties to existing languages.
- 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.
- Used extensively in software systems for string processing and as the basis for tools like `grep` and `flex`.
- Conceptually, regular languages 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

- Regular expression operator precedence:

$$(R)$$

$$R^*$$

$$R_1R_2$$

$$R_1 \cup R_2$$

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

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:
  - $\mathscr{L}(\varepsilon) = \{\varepsilon\}$
  - $\mathscr{L}(\emptyset) = \emptyset$
  - $\mathscr{L}(\mathbf{a}) = \{\mathbf{a}\}$
  - $\mathscr{L}(R_1 R_2) = \mathscr{L}(R_1)\,\mathscr{L}(R_2)$
  - $\mathscr{L}(R_1 \cup R_2) = \mathscr{L}(R_1) \cup \mathscr{L}(R_2)$
  - $\mathscr{L}(R^*) = \mathscr{L}(R)^*$
  - $\mathscr{L}((R)) = \mathscr{L}(R)$

Worthwhile activity: Apply this recursive definition to

a(b∪c)((d))

and see what you get.

Designing Regular Expressions

- Let $\Sigma = \{$**0**, **1**$\}$
- Let $L = \{ w \in \Sigma^* \mid w$ contains **00** as a substring $\}$

$$(0 \cup 1)^*00(0 \cup 1)^*$$

11011100101
0000
11111011110011111

Designing Regular Expressions

- Let $\Sigma = \{$**0**, **1**$\}$
- Let $L = \{ w \in \Sigma^* \mid w$ contains **00** as a substring $\}$

Σ*00Σ*

11011100101
0000
11111011110011111

Designing Regular Expressions

Let $\Sigma = \{\mathtt{0}, \mathtt{1}\}$

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

The length of a string w is denoted |w|

Designing Regular Expressions

- Let $\Sigma = \{$**0**, **1**$\}$
- Let $L = \{ w \in \Sigma^* \mid |w| = 4 \}$

$$\Sigma\Sigma\Sigma\Sigma$$

0000
1010
1111
1000

Designing Regular Expressions

- Let $\Sigma = \{$**0**, **1**$\}$
- Let $L = \{ w \in \Sigma^* \mid |w| = 4 \}$

$$\boldsymbol{\Sigma^4}$$

0000
1010
1111
1000

Designing Regular Expressions

- Let $\Sigma = \{$**0**, **1**$\}$
- Let $L = \{ w \in \Sigma^* \mid w$ contains at most one **0** $\}$

$$\mathbf{1^*(0 \cup \varepsilon)1^*}$$

11110111
111111
0111
0

Designing Regular Expressions

- Let $\Sigma = \{$**`0`**, **`1`**$\}$
- Let $L = \{ w \in \Sigma^* \mid w$ contains at most one **`0`** $\}$

1*0?1*

11110111
111111
0111
0

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
barack.obama@whitehouse.gov

A More Elaborate Design

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

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

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

A More Elaborate Design

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

$$\mathbf{a^+ \quad (.a^+)^* \; @ \quad a^+.a^+ \quad (.a^+)^*}$$

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

A More Elaborate Design

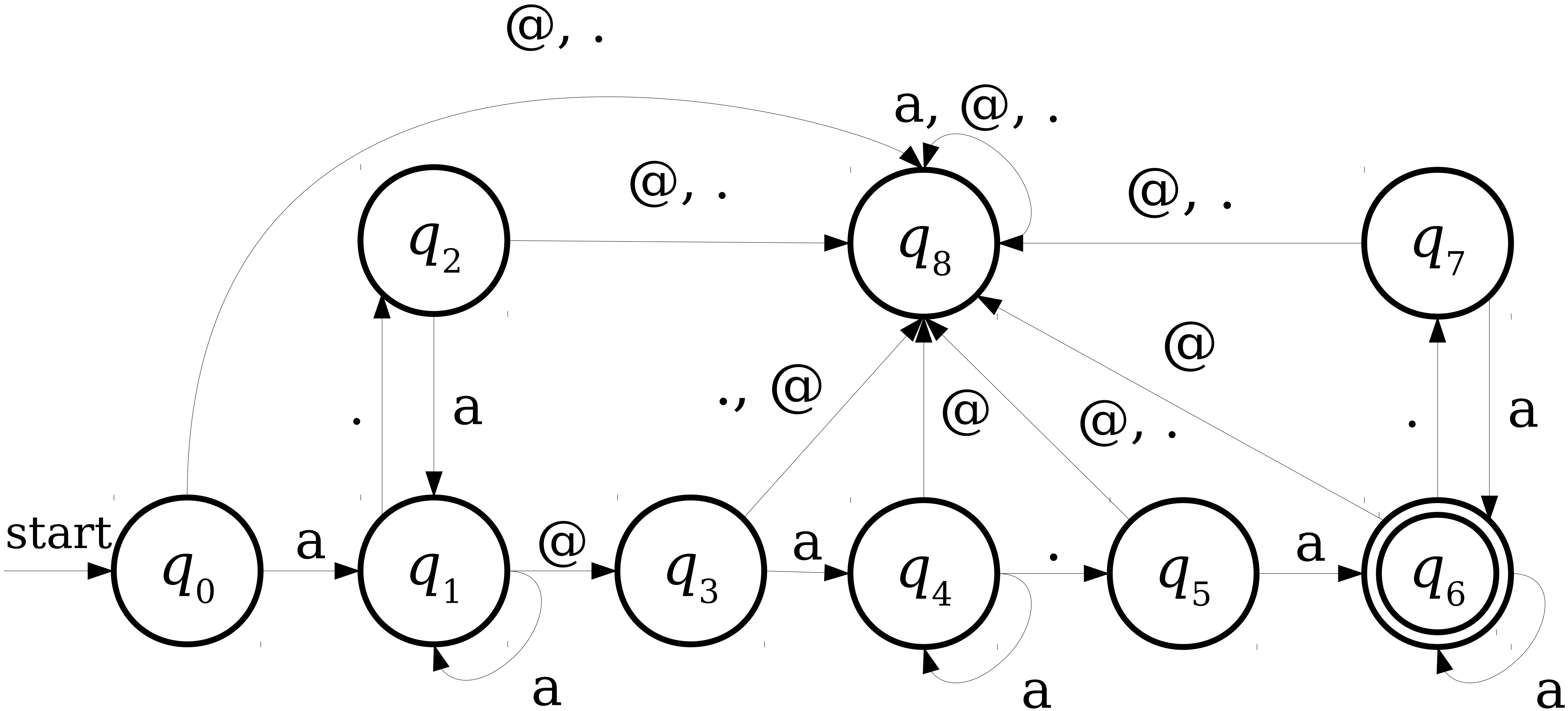
- Let Σ = { **a**, **.**, **@** }, where **a** represents "some letter."
- Let's make a regex for email addresses.

$$\mathbf{a^{+}(.a^{+})^{*}@\,a^{+}\,(.a^{+})^{+}}$$

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

Regular Expressions are Awesome

$$\mathbf{a^+(.a^+)^*@a^+(.a^+)^+}$$



Shorthand Summary

- R^n is shorthand for RR … 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!

Problem Sets

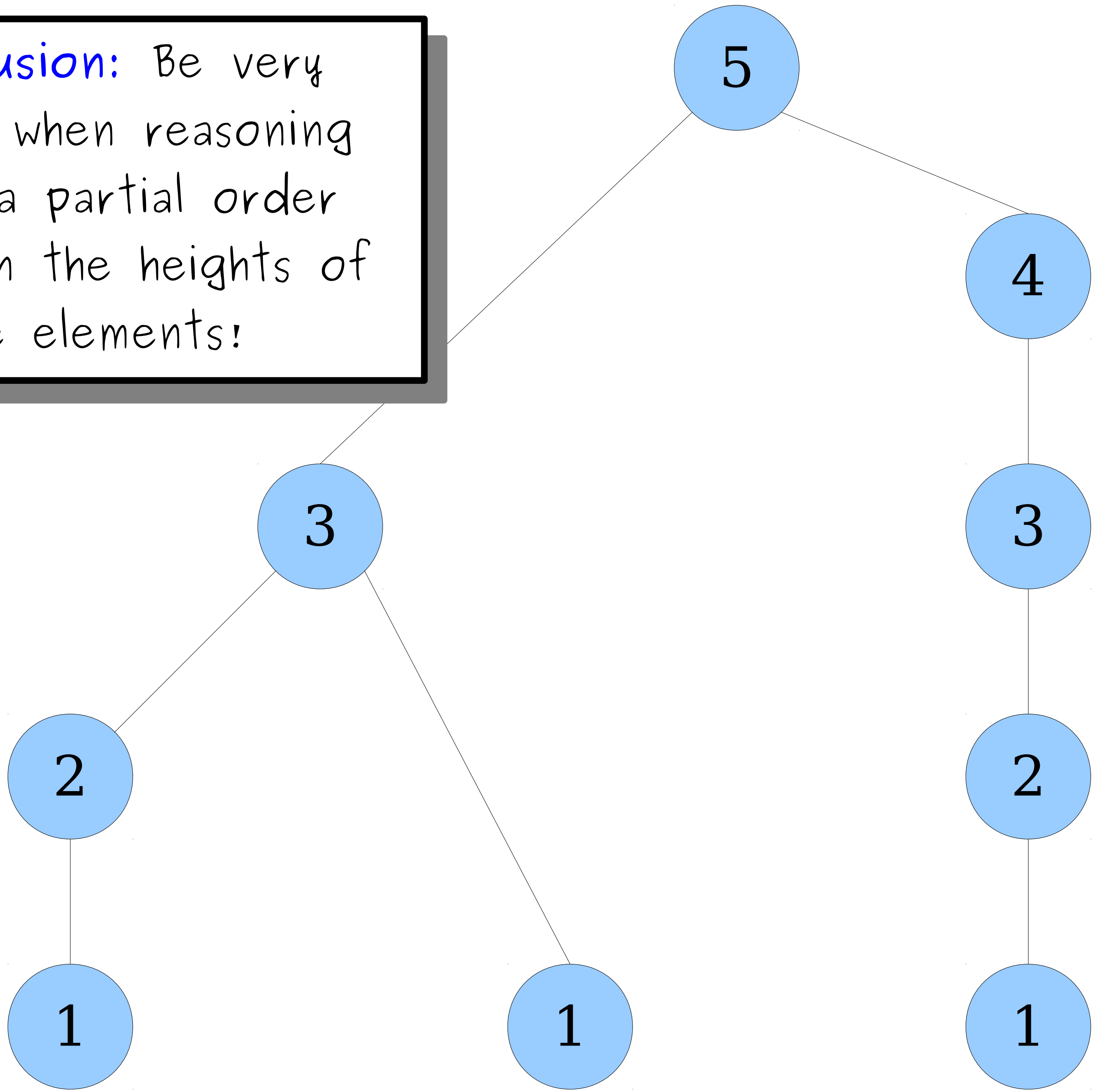
- Problem Set Five was due at 3:00PM today.
  - Want to use late days? Submit by Monday at 3:00PM.
- Problem Set Six goes out today. It's due next Friday at 3:00PM.
  - Play around with DFAs, NFAs, regular expressions, and properties of regular languages.
  - ***Please use our online tools to design and submit your automata and regexes***. They're really, really useful!

Mental Health Tea

- DiversityBase is holding a Mental Health Tea event next Wednesday, February 17, at 8:00PM in the Kimball Lounge.
- Want to destress a bit? Like tea and cookies? Feel free to show up!
- They recommend bringing a fun mug if you happen to have one.

PS4: Common Mistakes

Let's Talk Hasse Diagrams



Conclusion: Be very careful when reasoning about a partial order based on the heights of the elements!

Let's Talk Hasse Diagrams

What does the Hasse diagram for the $<$ relation over $\mathbb{R}$ look like?

There are no lines in this Hasse diagram!

Let's Talk Hasse Diagrams

What does the Hasse diagram for the > relation over ℝ look like?

It's exactly the same as the Hasse diagram for < over ℝ!

Conclusion: It's not safe to reason about a strict order purely by talking about its Hasse diagram.

There are no lines in this Hasse diagram!

Your Questions

“Ultimately, which do you think is more important: career or love? Professional life or personal life?”

In some sense I think this question is like this one: who should you love more, your spouse(s), your child(ren), or your parent(s)? The correct answer is “you should love all of them.”

I think that the real question is how best to strike a balance between your personal life and professional life. From experience, you do not want to get into a position where you're ignoring everyone around you to purely focus on your job. You also don't want to let your personal commitments disablingly interfere with your career. There's a lot of public conversation about employers creating environments that are amenable to new parents, and there's a lot of private conversations about how couples and families will find a way to manage competing priorities. I don't think anyone has a good answer for how to do this right.

Back to CS103!

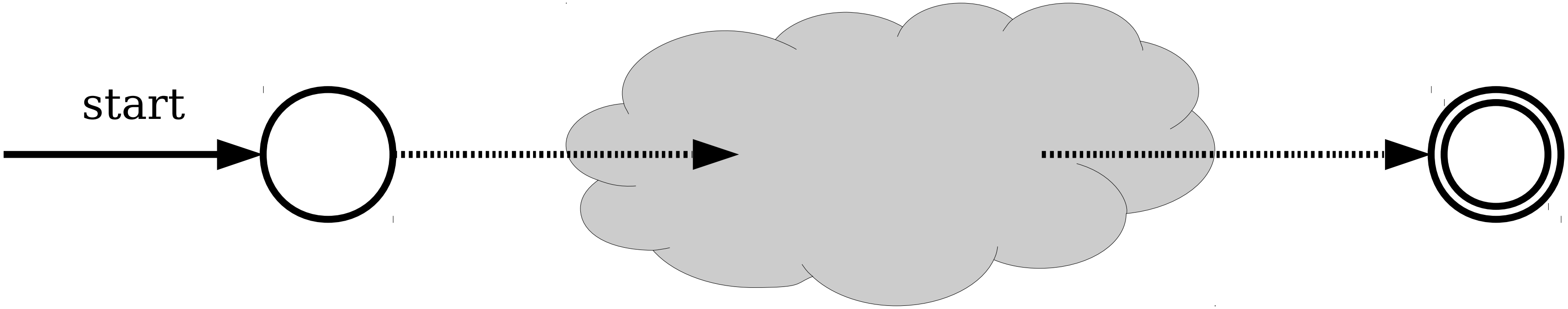
The Power of Regular Expressions

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

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

Thompson's Algorithm

- ***Thompson's algorithm*** is an algorithm for converting any 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.



Thompson's Algorithm

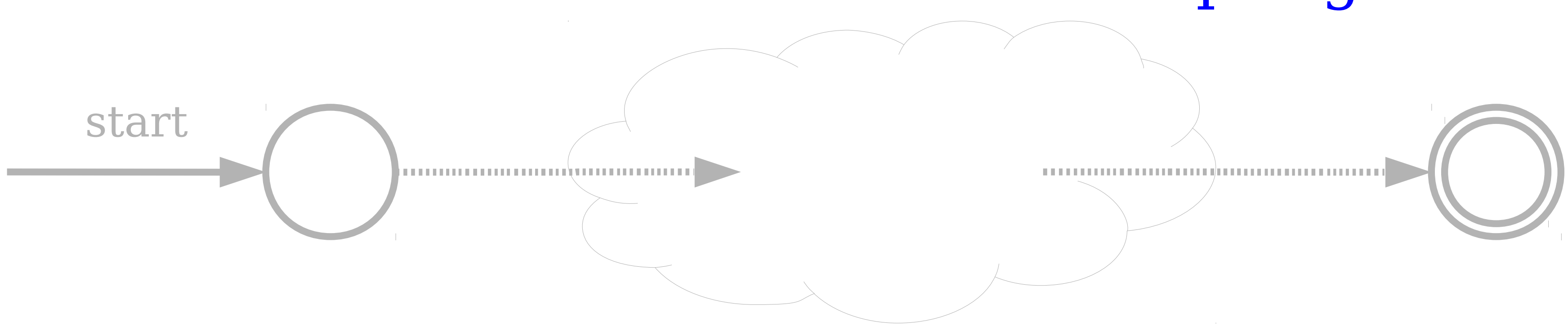
Thompson's algorithn
converting any regular e

Theorem: For any regu
is an NFA N such that

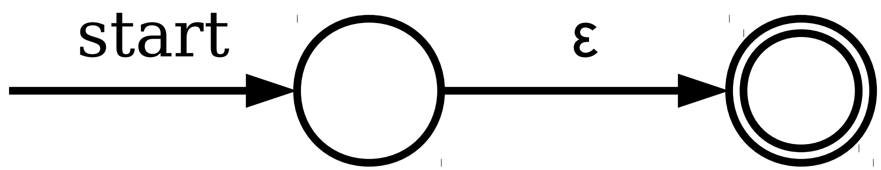
These are stronger requirements than are necessary for a normal NFA. We enforce these rules to simplify the construction.

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

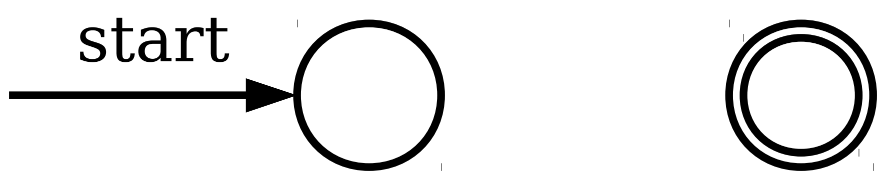
- N has exactly one accepting state.
- N has no transitions into its start state.
- N has no transitions out of its accepting state.



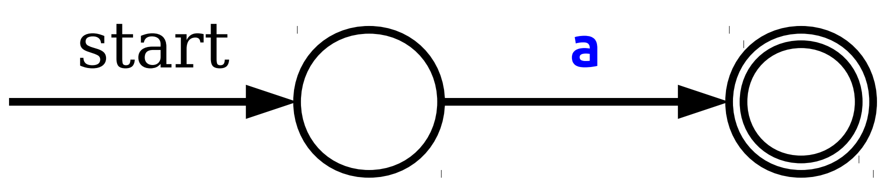
Base Cases



Automaton for ε

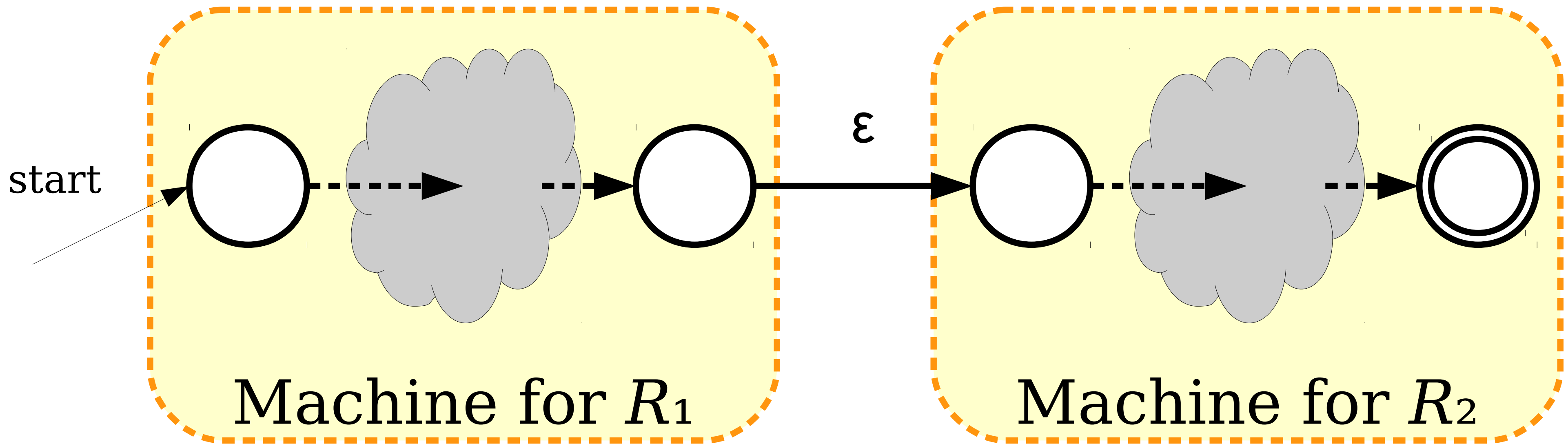


Automaton for Ø

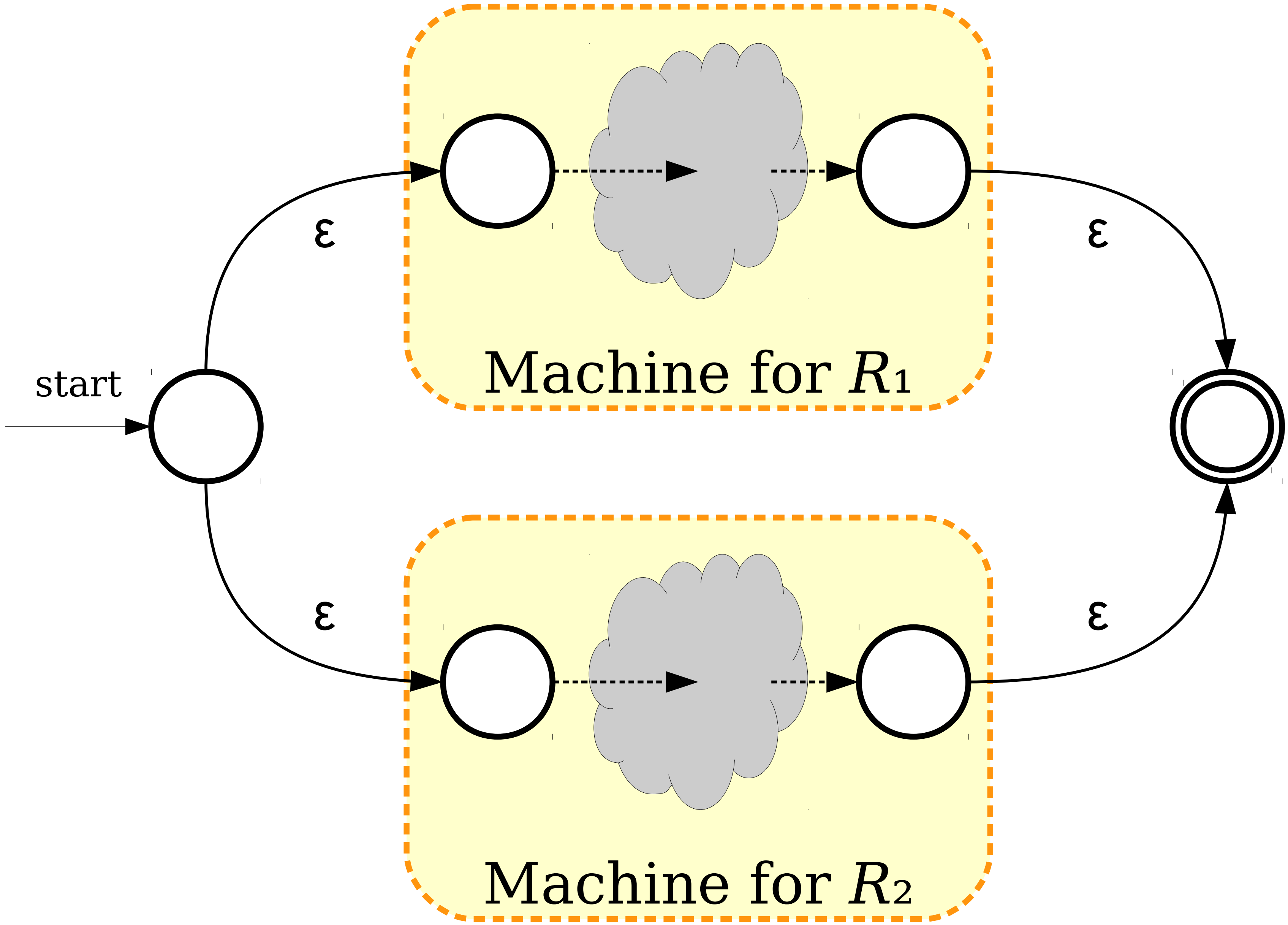


Automaton for single character a

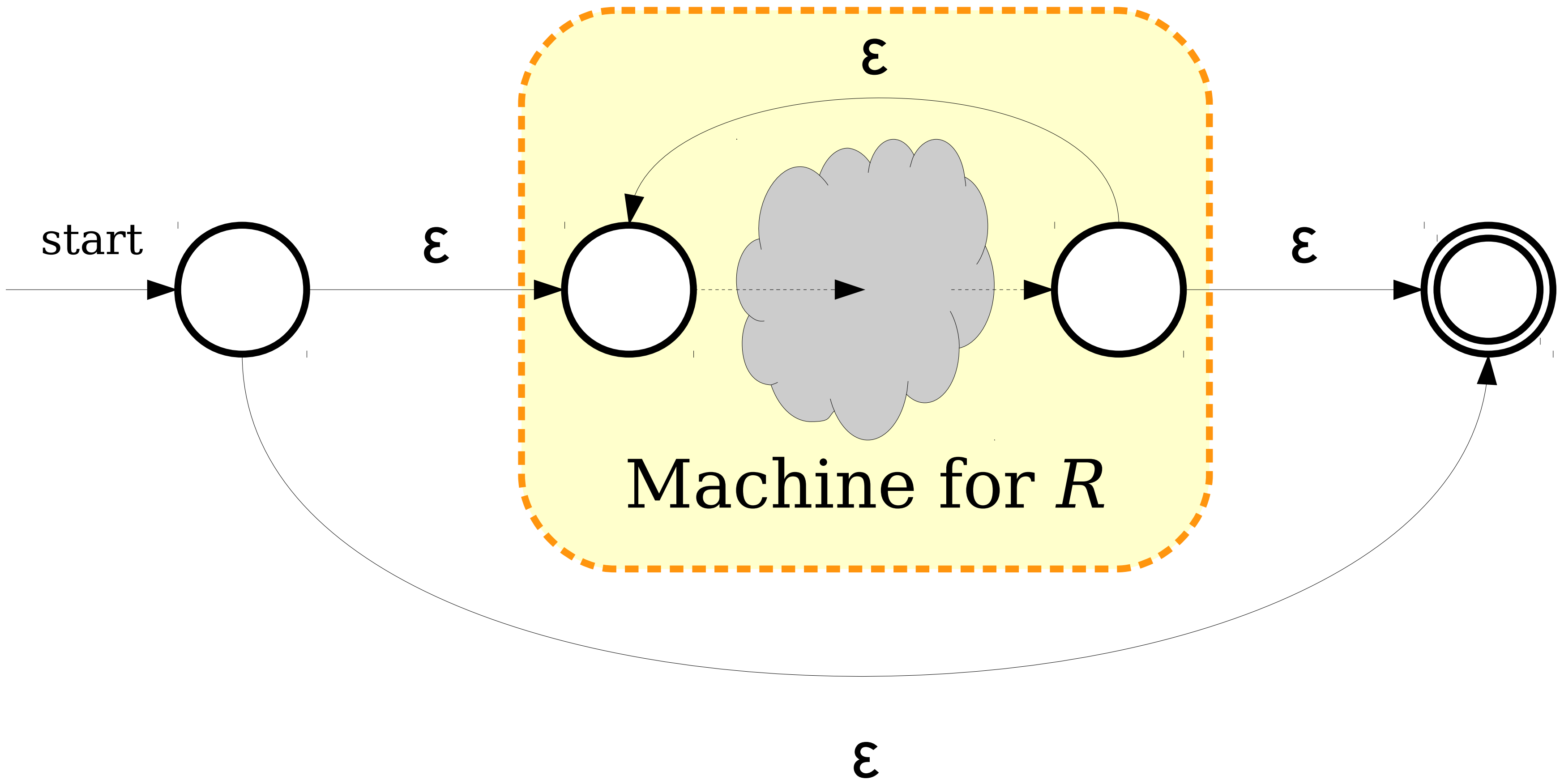
Construction for R_1R_2



Construction for $R_1 \cup R_2$



Construction for *R**



Why This Matters

- Many software tools work by matching regular expressions against text.
- One possible algorithm for doing so:
  - Convert the regular expression to an NFA.
  - (Optionally) Convert the NFA to a DFA using the subset construction.
  - Run the text through the finite automaton and look for matches.
- This is actually used in practice! The compiled matching automata run extremely quickly.

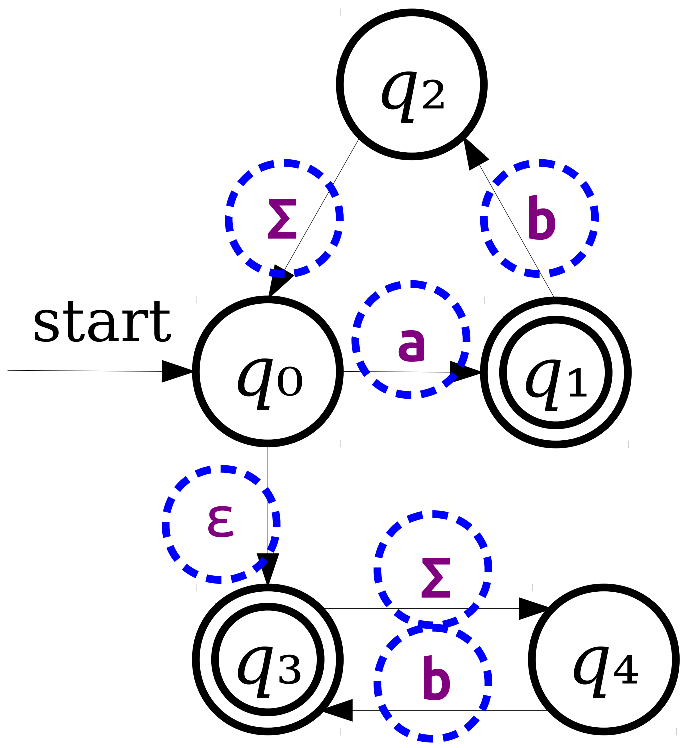
The Power of Regular Expressions

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

This is not obvious!

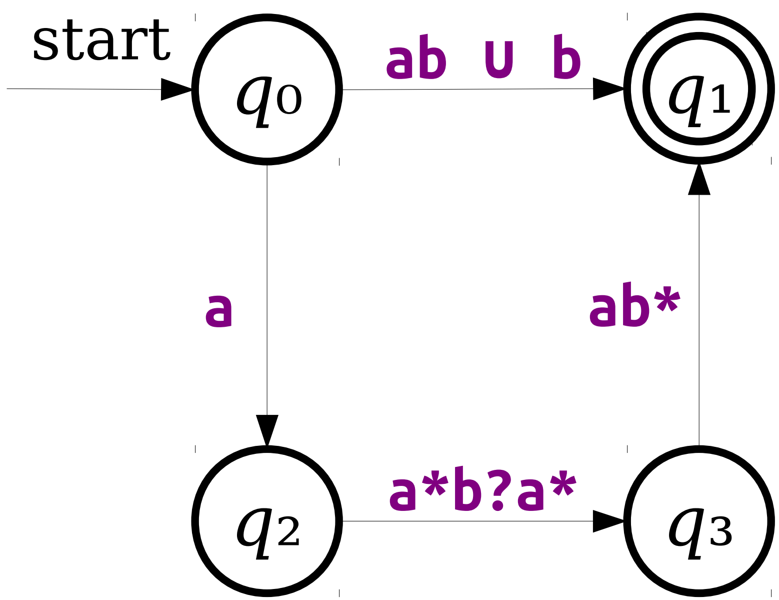
Proof idea: Show how to convert an arbitrary NFA into a regular expression.

Generalizing NFAs



These are all regular expressions!

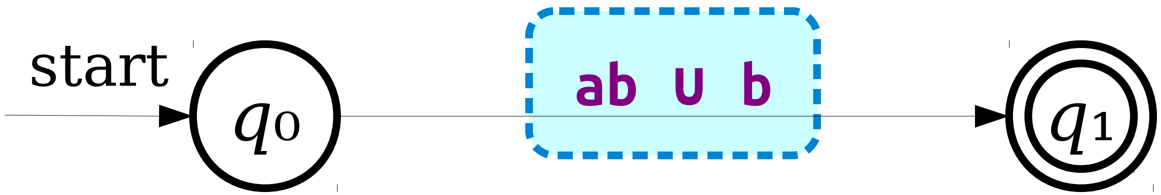
Generalizing NFAs



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

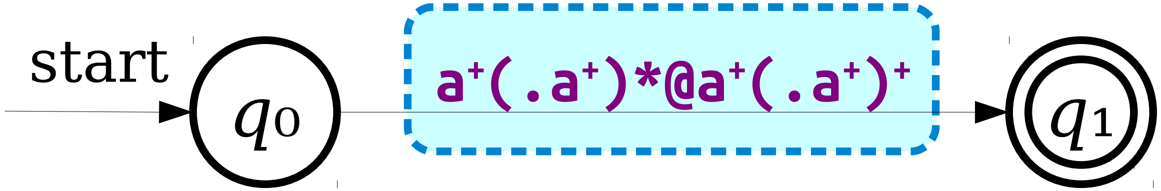
Key Idea 1: Imagine that we can label transitions in an NFA with arbitrary regular expressions.

Generalizing NFAs



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

Generalizing NFAs

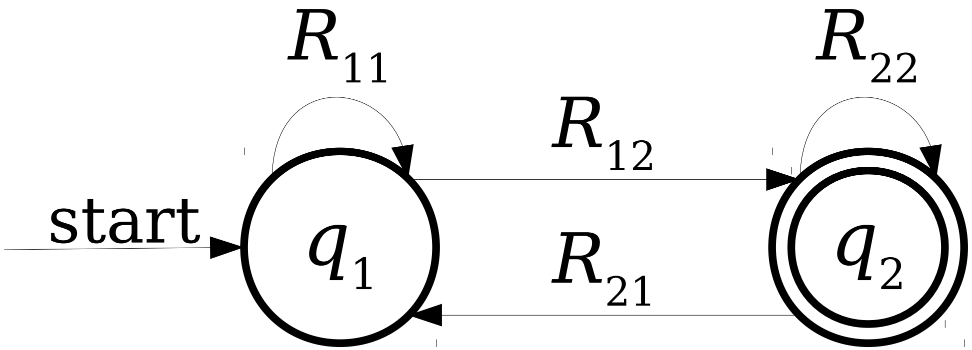


Is there a simple regular expression for the language of this generalized NFA?

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

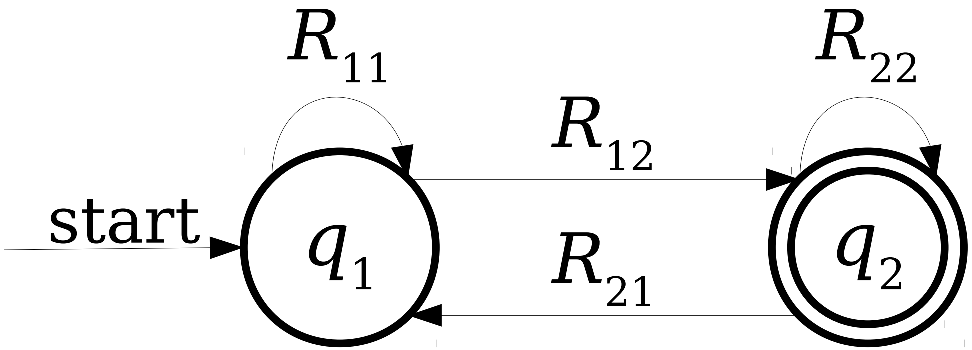
…then we can easily read off a regular expression for that NFA.

From NFAs to Regular Expressions



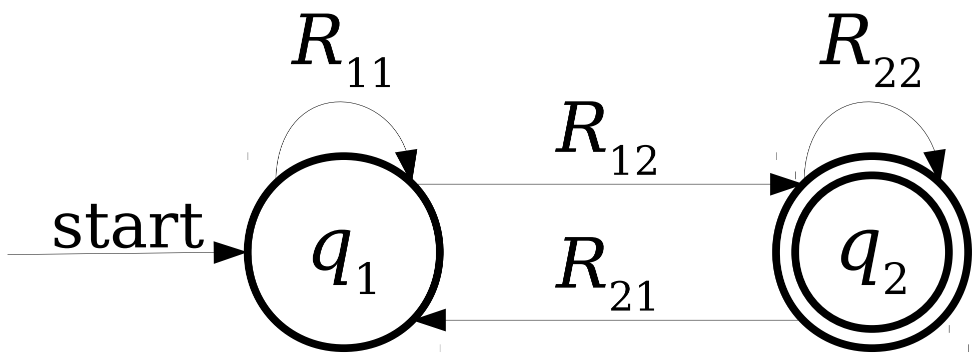
Here, R_{11}, R_{12}, R_{21}, and R_{22} are arbitrary regular expressions.

From NFAs to Regular Expressions



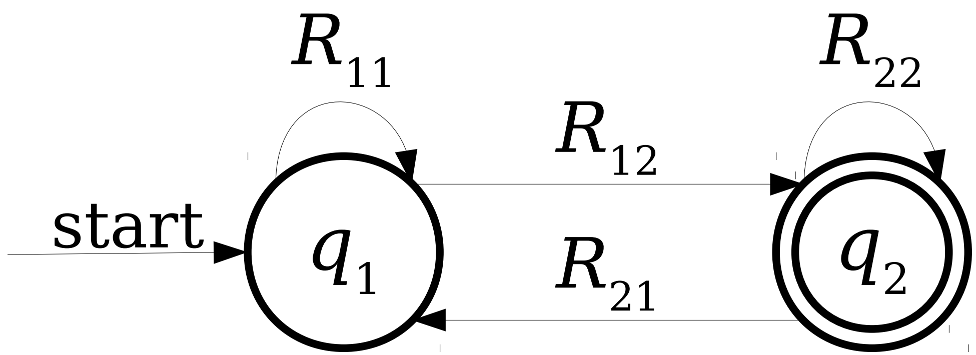
Question: Can we get a clean regular expression from this NFA?

From NFAs to Regular Expressions



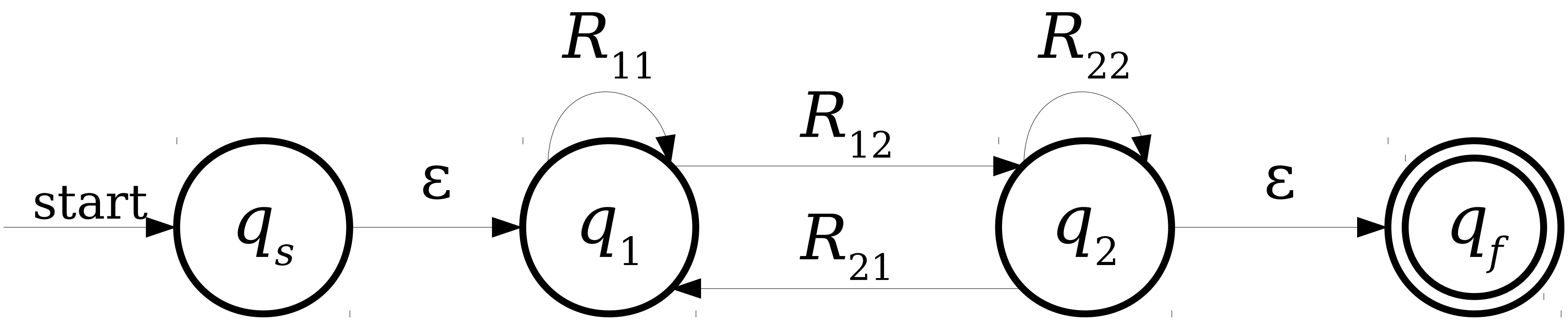
Key Idea 3: Somehow transform this NFA so that it looks like this:

From NFAs to Regular Expressions

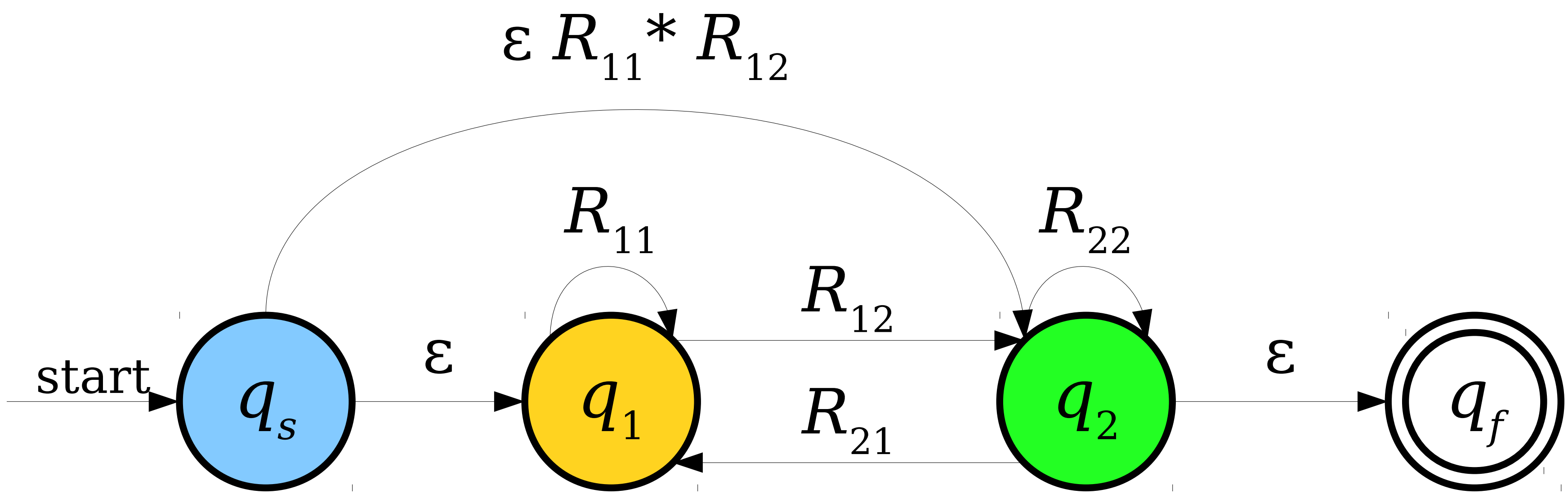


The first step is going to be a bit weird...

From NFAs to Regular Expressions

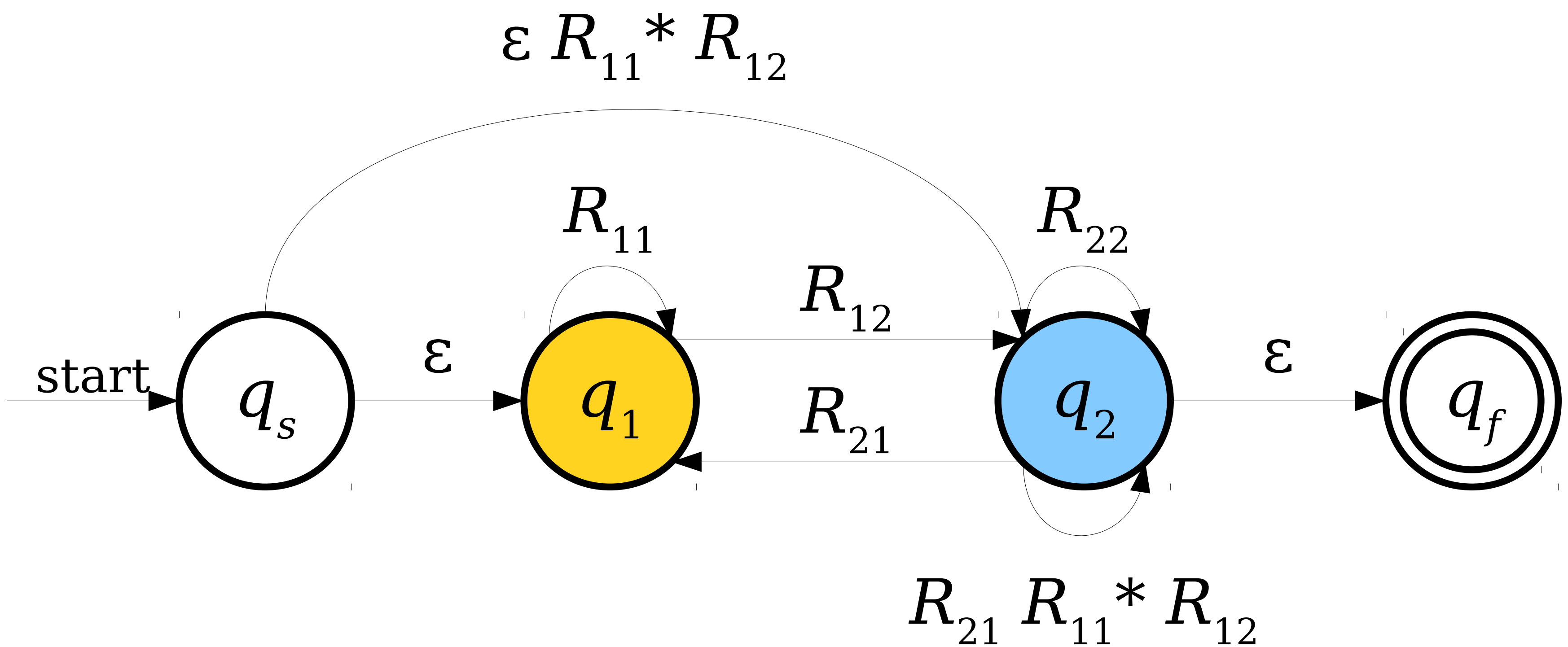


From NFAs to Regular Expressions

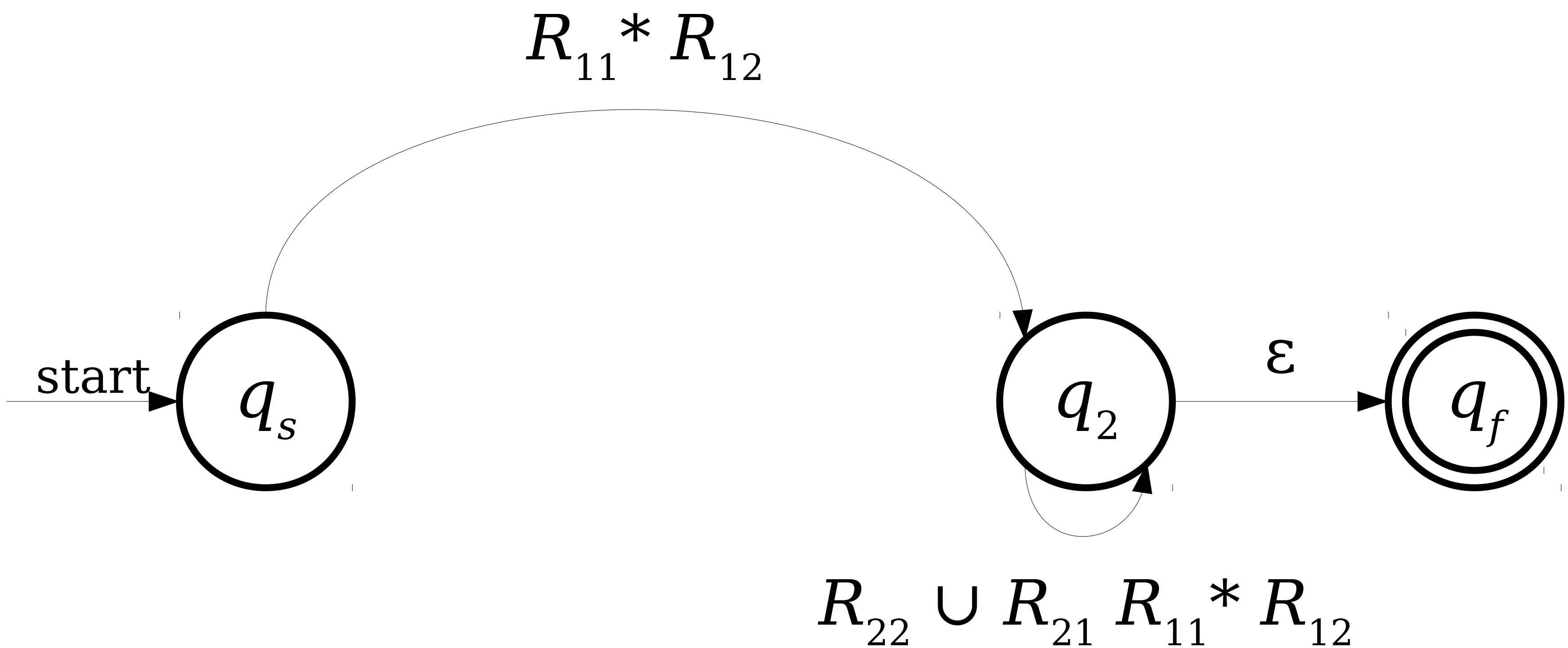


Note: We're using **concatenation** and **Kleene closure** in order to skip this state.

From NFAs to Regular Expressions

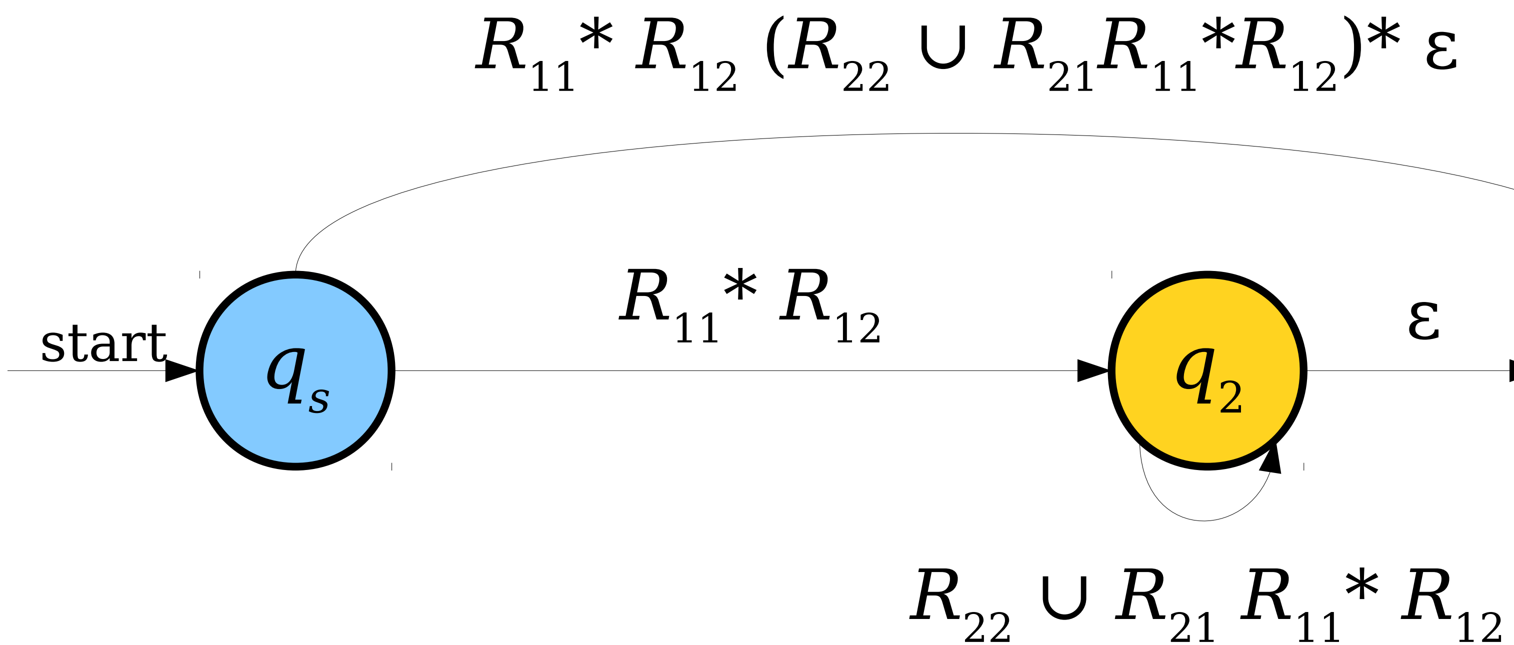


From NFAs to Regular Expressions

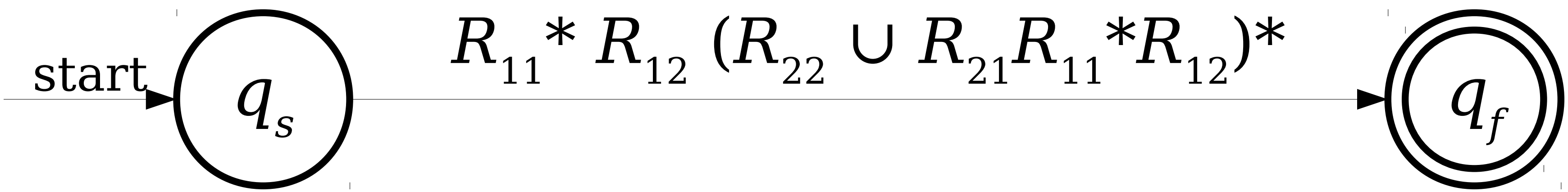


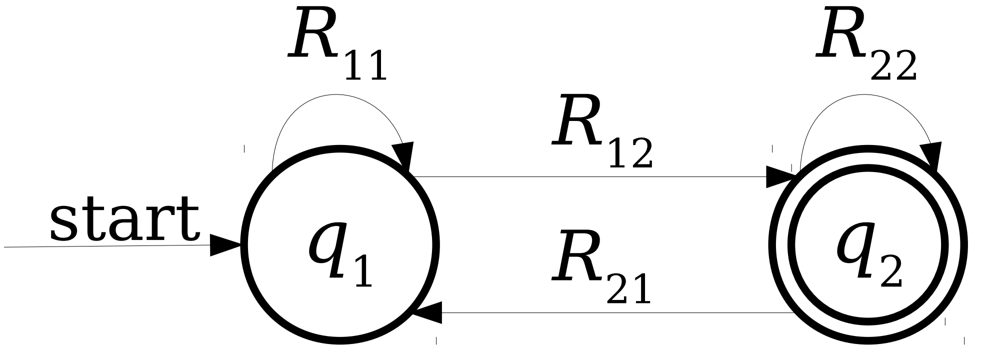
Note: We're using union to combine these transitions together.

From NFAs to Regular Expressions



From NFAs to Regular Expressions





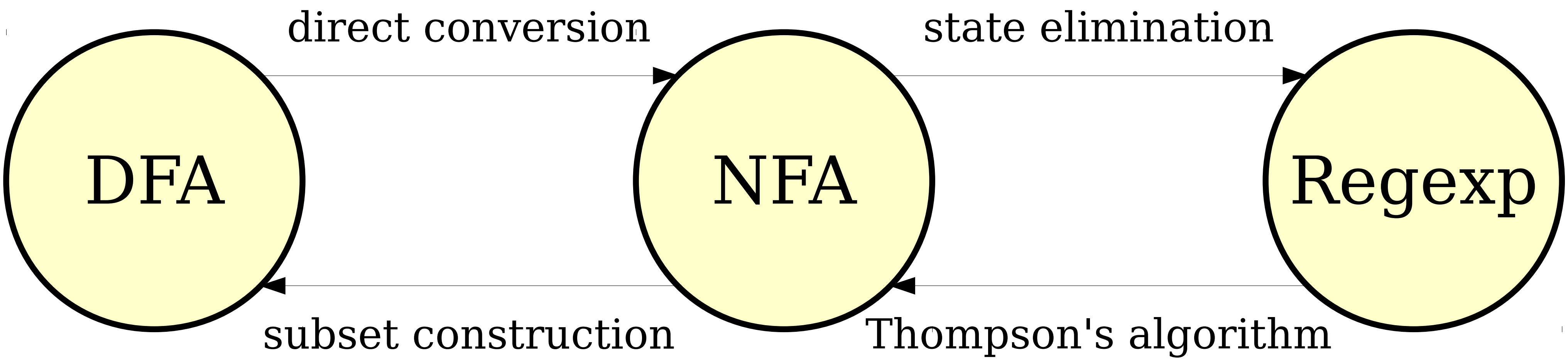
The Construction at a Glance

- 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.