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

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

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.

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***.

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.
- The symbol **ε** is a regular expression that represents the language {ε}
  - ***This is not the same as Ø!***
  - ***This is not the same as ε!***
- For any **`a`** ∈ Σ, the symbol **`a`** is a regular expression for the language {**`a`**}

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, $\boldsymbol{R_1 \mid R_2}$ 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:

$$(R)$$

$$R^*$$

$$R_1R_2$$

$$R_1 \mid R_2$$

- 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:
  - $\mathscr{L}(\varepsilon) = \{\varepsilon\}$
  - $\mathscr{L}(\emptyset) = \emptyset$
  - $\mathscr{L}(\mathtt{a}) = \{\mathtt{a}\}$
  - $\mathscr{L}(R_1R_2) = \mathscr{L}(R_1)\,\mathscr{L}(R_2)$
  - $\mathscr{L}(R_1 \mid R_2) = \mathscr{L}(R_1) \cup \mathscr{L}(R_2)$
  - $\mathscr{L}(R^*) = \mathscr{L}(R)^*$
  - $\mathscr{L}((R)) = \mathscr{L}(R)$

Worthwhile activity: Apply this recursive definition to

`a(b|c)((d))`

and see what you get.

Regular Expressions are Awesome

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

(0 | 1)*00(0 | 1)*

11011100101
0000
11111011110011111

Regular Expressions are Awesome

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

$$\Sigma^* 00 \Sigma^*$$

11011100101
0000
11111011110011111

Regular Expressions are Awesome

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

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

The length of a string w is denoted $|w|$

Regular Expressions are Awesome

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

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

0000
1010
1111
1000

Regular Expressions are Awesome

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

Σ^4

0000
1010
1111
1000

Regular Expressions are Awesome

- Let Σ = {**0**, **1**}
- Let L = { w ∈ Σ* | w contains at most one **0** }

1*(0 | ε)1*

11110111
111111
0111
0

Regular Expressions are Awesome

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

1*0?1*

11110111
111111
0111
0

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:

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

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

Regular Expressions are Awesome

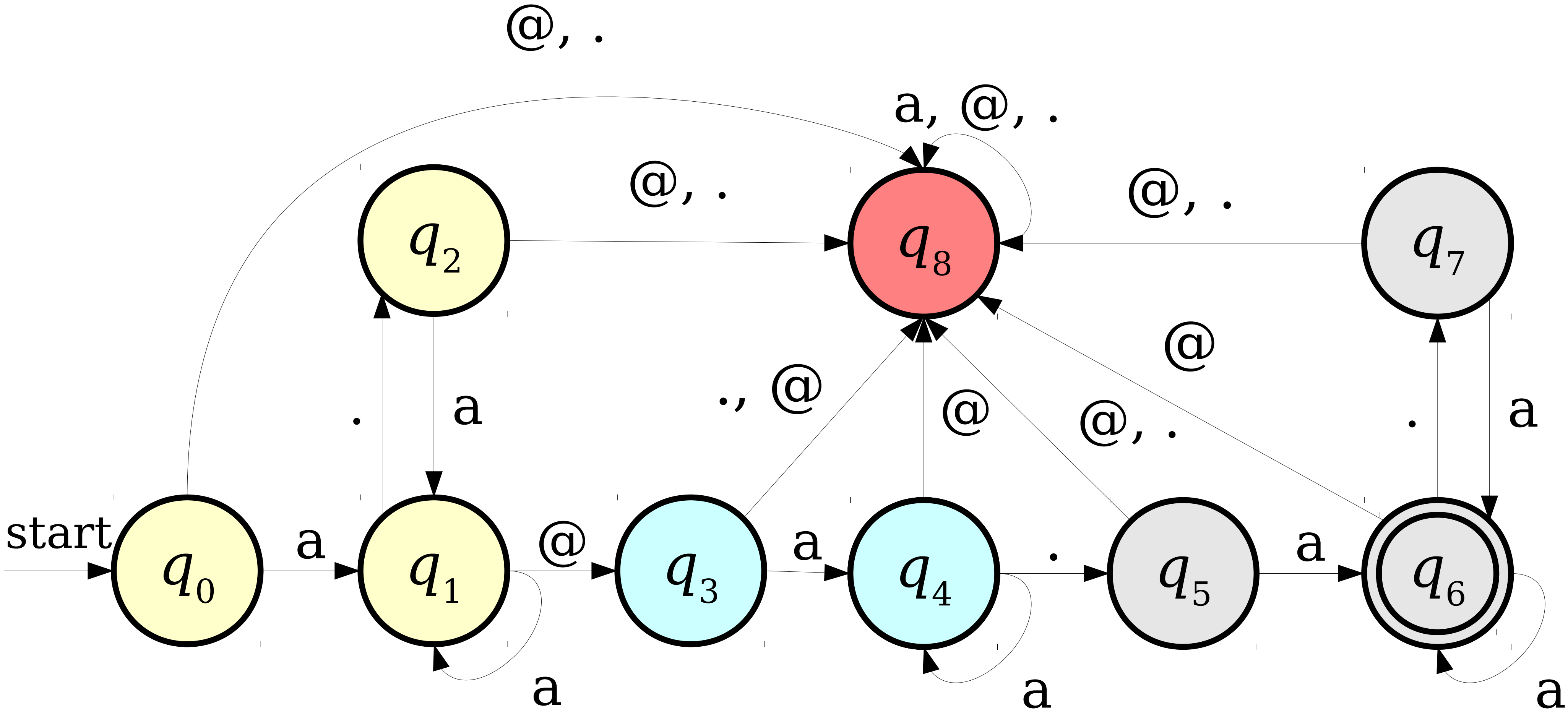
- Let Σ = { **a**, **.**, **@** }, where **a** represents "some letter."
- Regular expression for email addresses:

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

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

Regular Expressions are Awesome

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



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 | ε), meaning "zero or one copies of R."
- R^+ is shorthand for RR*, meaning "one or more copies of R."

Time-Out for Announcements!

Problem Set Five

- Problem Set Four was due at the start of class today.
  - Need more time? Turn it in by Tuesday at 12:50 with one late day or Wednesday at 12:50 with two.
- Problem Set Five goes out today. It's due next Monday at the start of class.
  - Play around with DFAs, NFAs, regular expressions, and properties of regular languages!
  - We have online tools for developing finite automata and regular expressions; we hope you find them useful!

Your Questions

“Why don't you use a Mac? Also, what are your thoughts on Disney making more Star Wars films?”

I only really use my laptop for email, word processing, slides, and development, so I didn't think it was worth shelling out the extra to get a Mac.

Also, I have absolutely no opinion on the new Star Wars movies.

"Why is there such a huge emphasis on women in CS or women in STEM in general? If the university is 50/50 at large, some departments would inevitably have more or fewer men or women. I have never understood this besides *en fait* assuming discrimination"

1. Random distributions alone would not account for the gender skew in CS.
2. The world as a whole benefits when everyone gets a fair shot at creating technology.
3. The assumption that technology is a meritocracy doesn't hold up to experimental evidence.
4. There are definite gender/racial/socioeconomic biases in the tech industry and in CS as a whole.

“sup”

Not much! I learned a really cool theorem and put it as the extra credit problem on the problem set! And I recently found an enchilada recipe that I (vegetarian) and my relatives (dairy allergy) can both enjoy! And I'm thinking deep thoughts about what I want to do with my life. You?

Back to CS103!

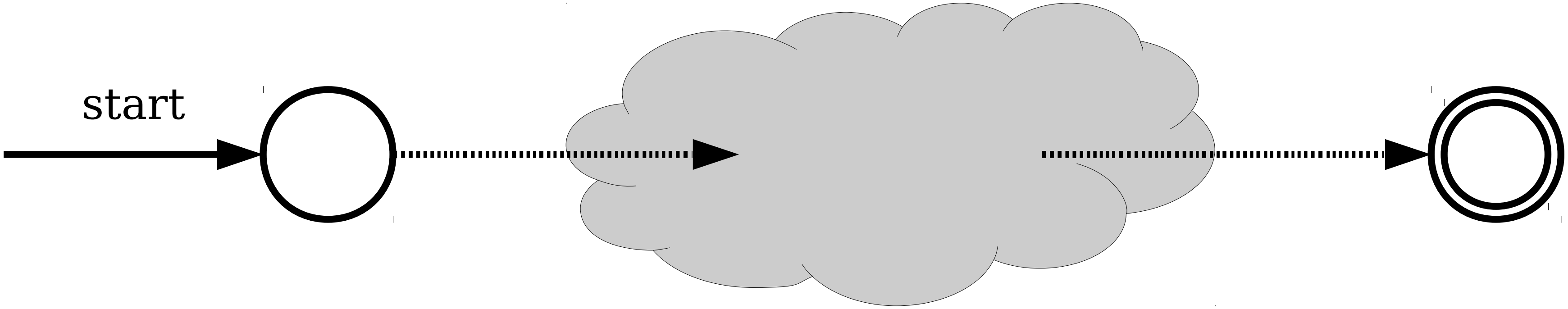
The Power of Regular Expressions

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.

A Marvelous Construction

- The following theorem proves the language of any regular expression is regular:
- ***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

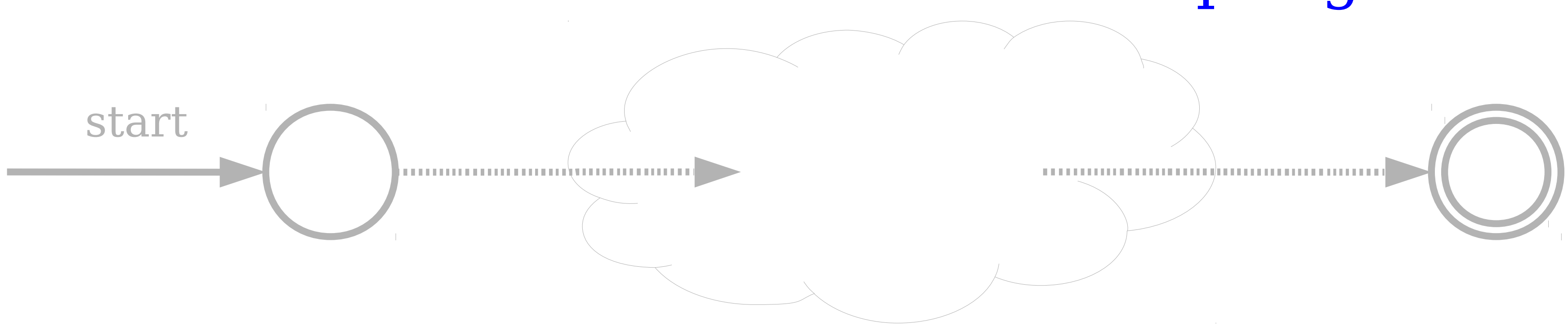
The following theorem p
any regular expression i

Theorem: For any regu
is an NFA N such that

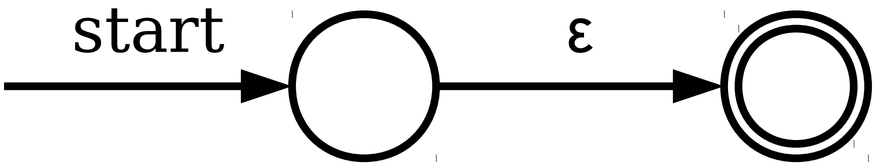
$$\mathscr{L}(R) = \mathscr{L}(N)$$

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

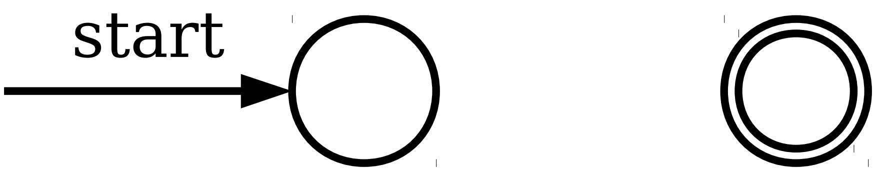
- N has exactly one accepting state.
- N has no transitions into its start state.
- N has no transitions out of its accepting state.



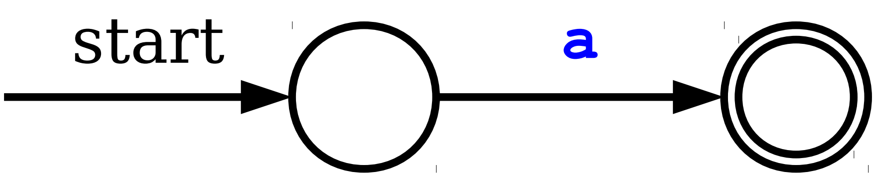
Base Cases



Automaton for ε

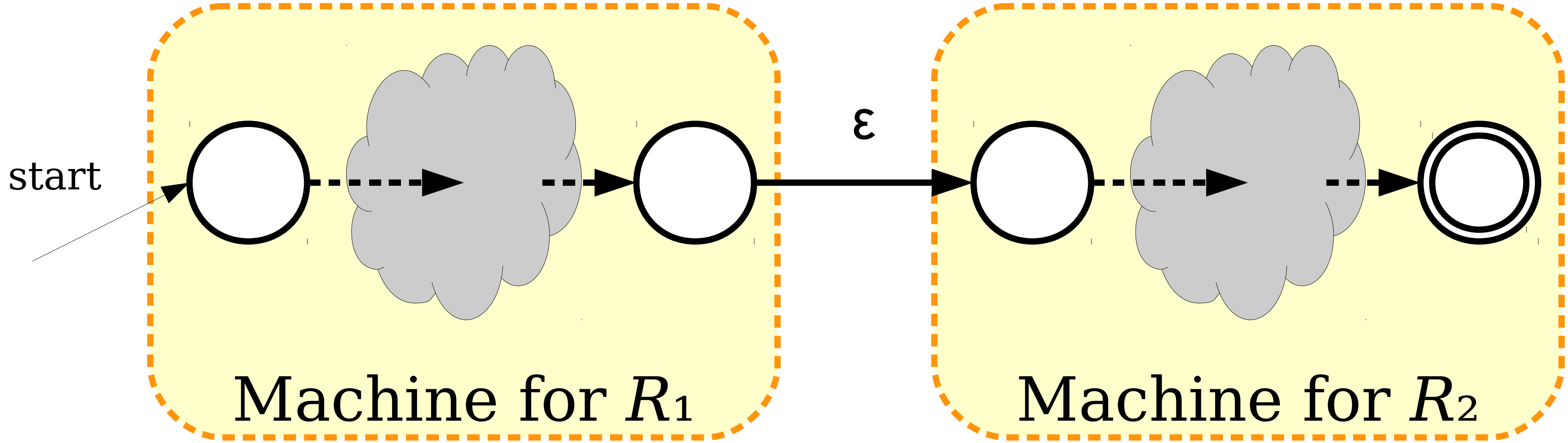


Automaton for Ø

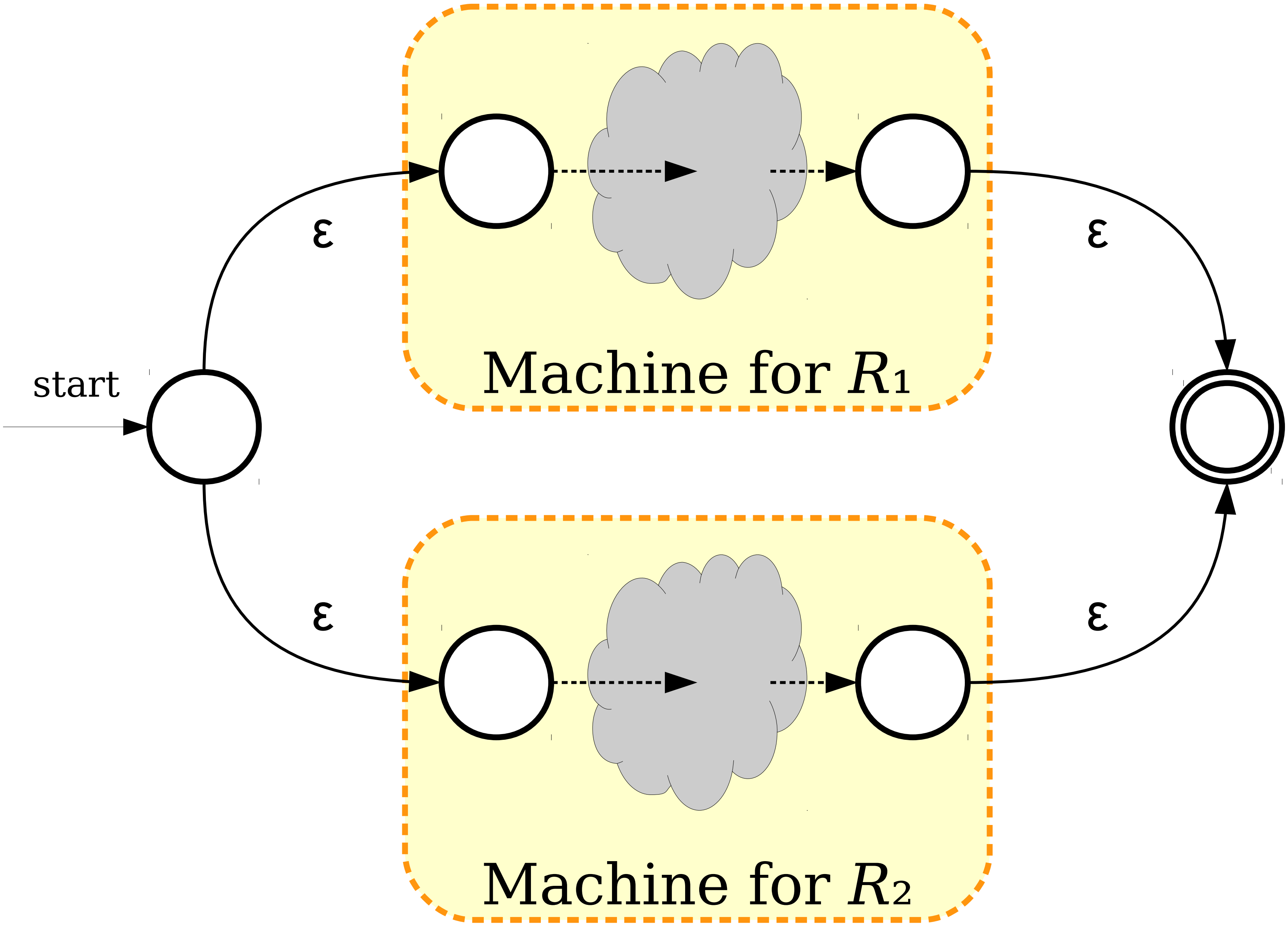


Automaton for single character **a**

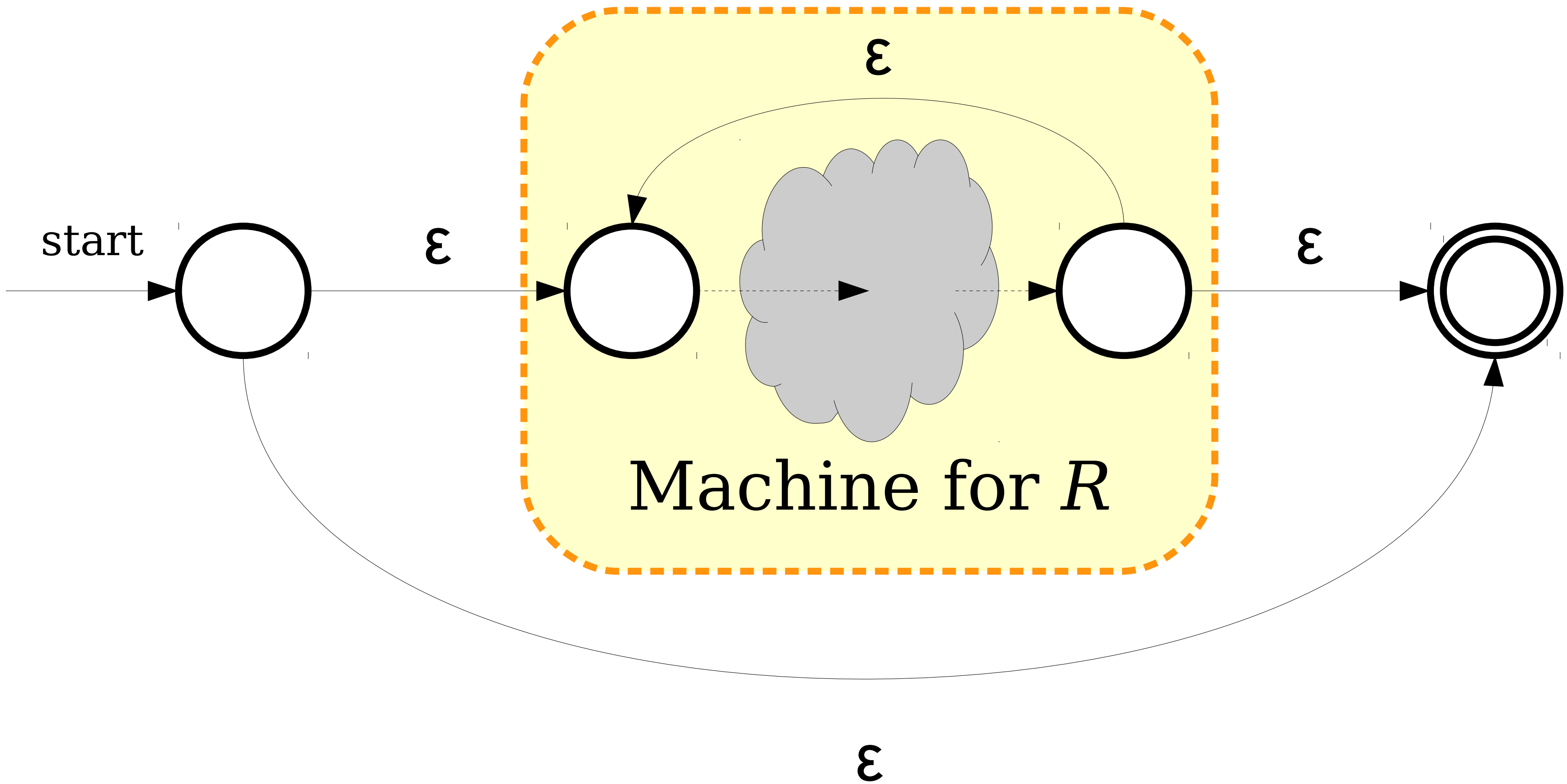
Construction for R_1R_2



Construction for $R_1 \mid R_2$



Construction for *R**



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.

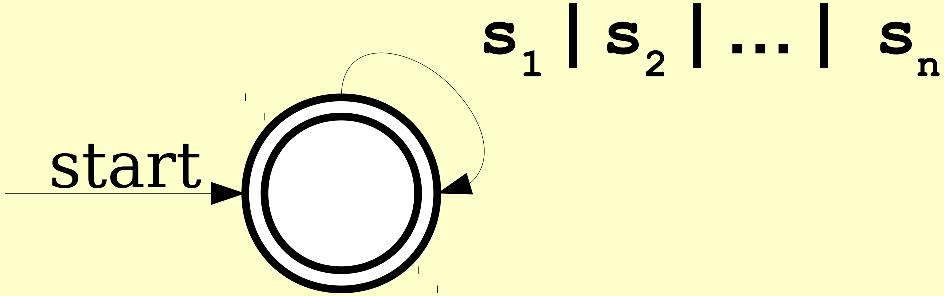
The Power of Regular Expressions

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.

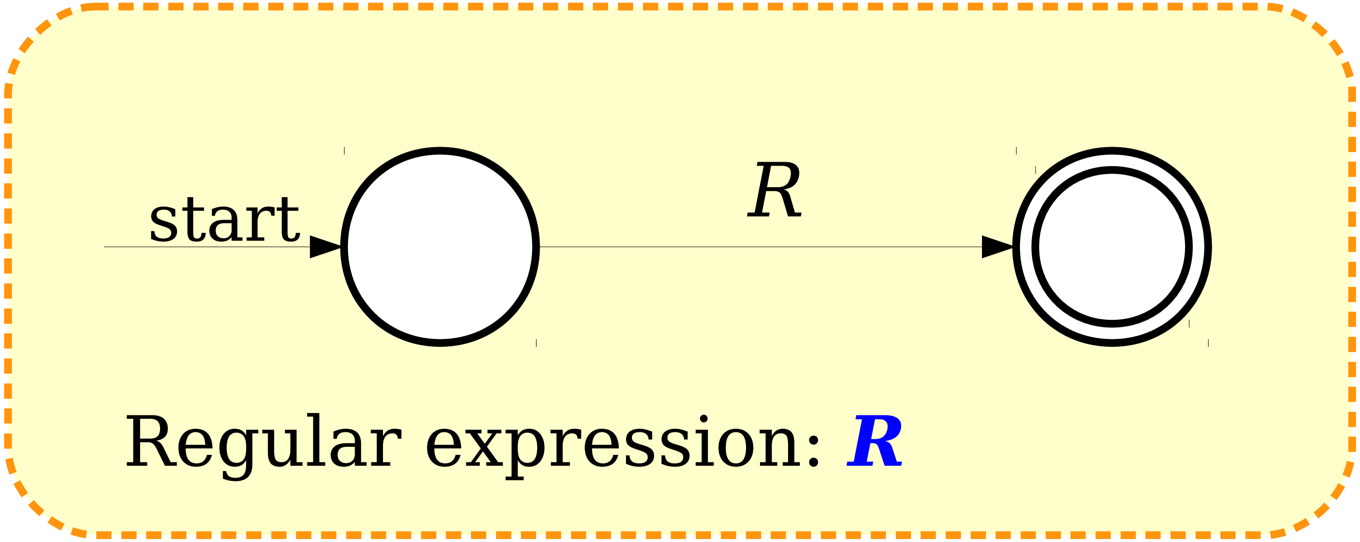
From NFAs to Regular Expressions



Regular expression: $(s_1 | s_2 | ... | s_n)*$

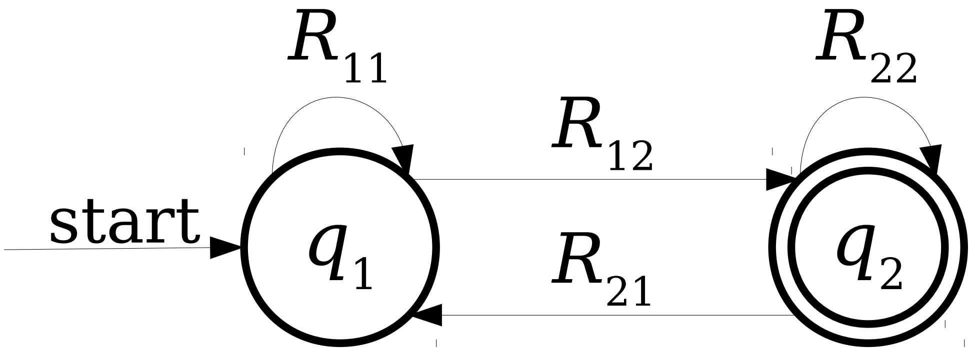
Key idea: Label transitions with arbitrary regular expressions.

From NFAs to Regular Expressions

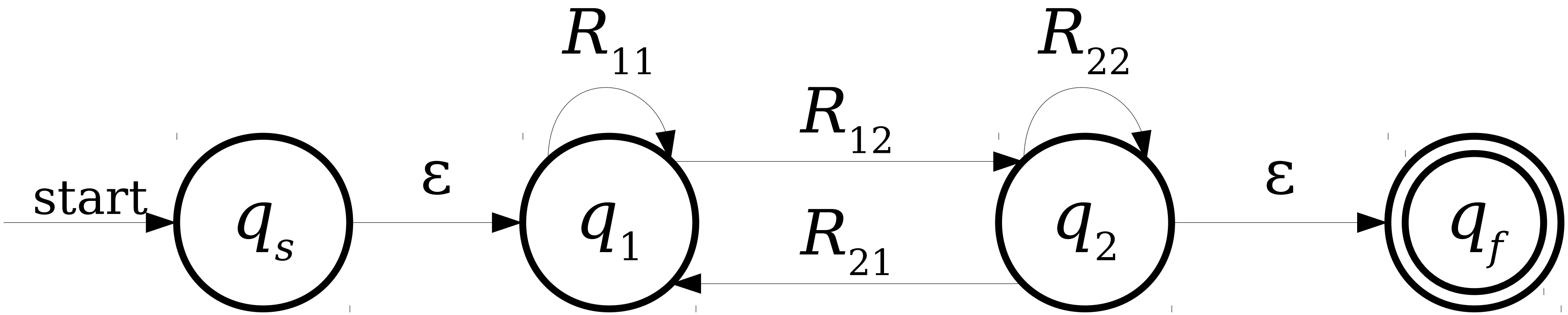


Key idea: If we can convert any NFA into something that looks like this, we can easily read off the regular expression.

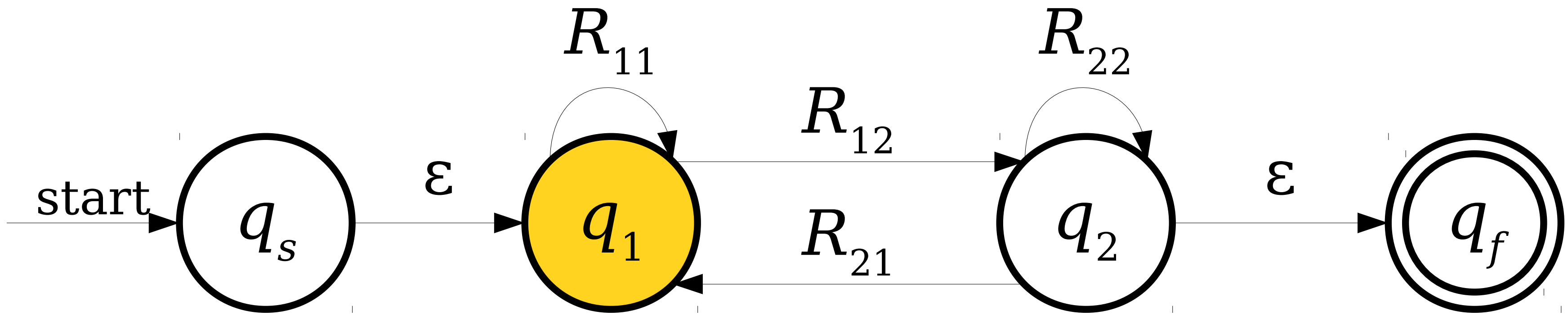
From NFAs to Regular Expressions



From NFAs to Regular Expressions

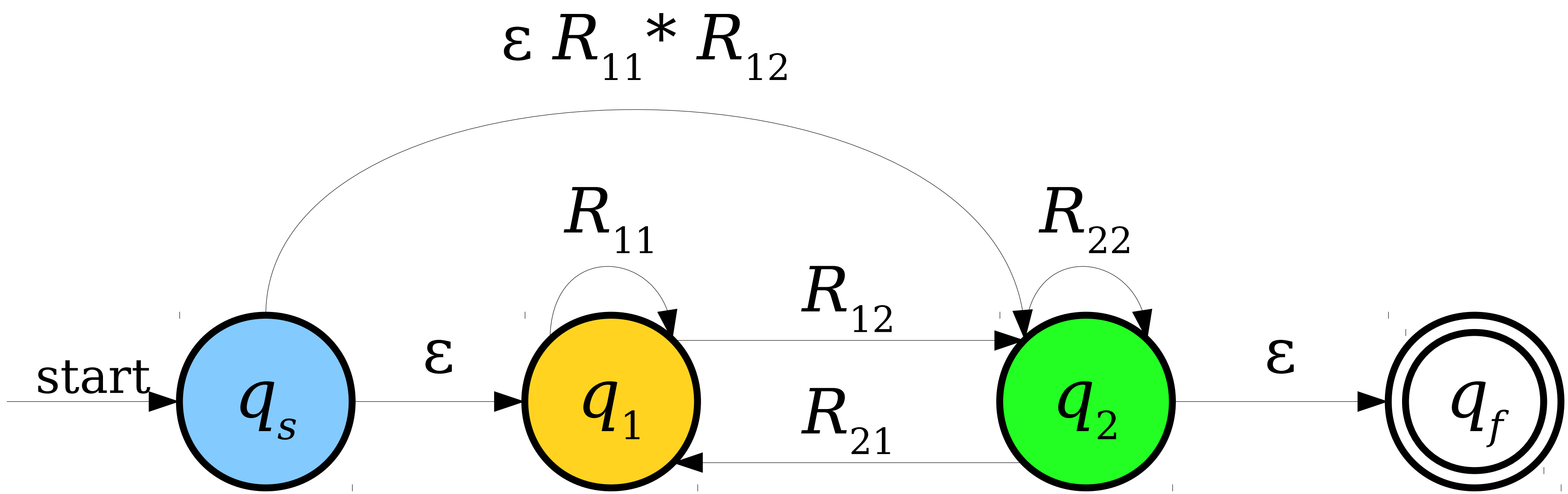


From NFAs to Regular Expressions



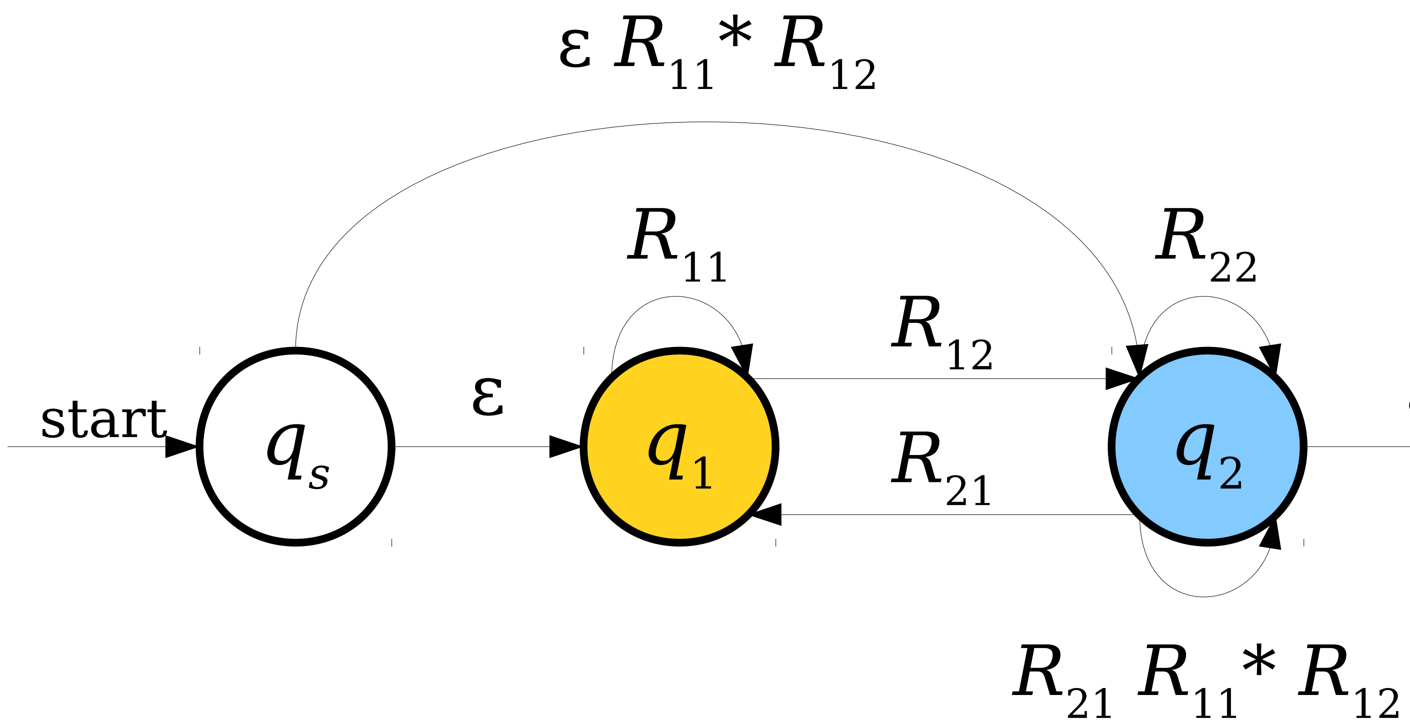
Could we eliminate this state from the NFA?

From NFAs to Regular Expressions

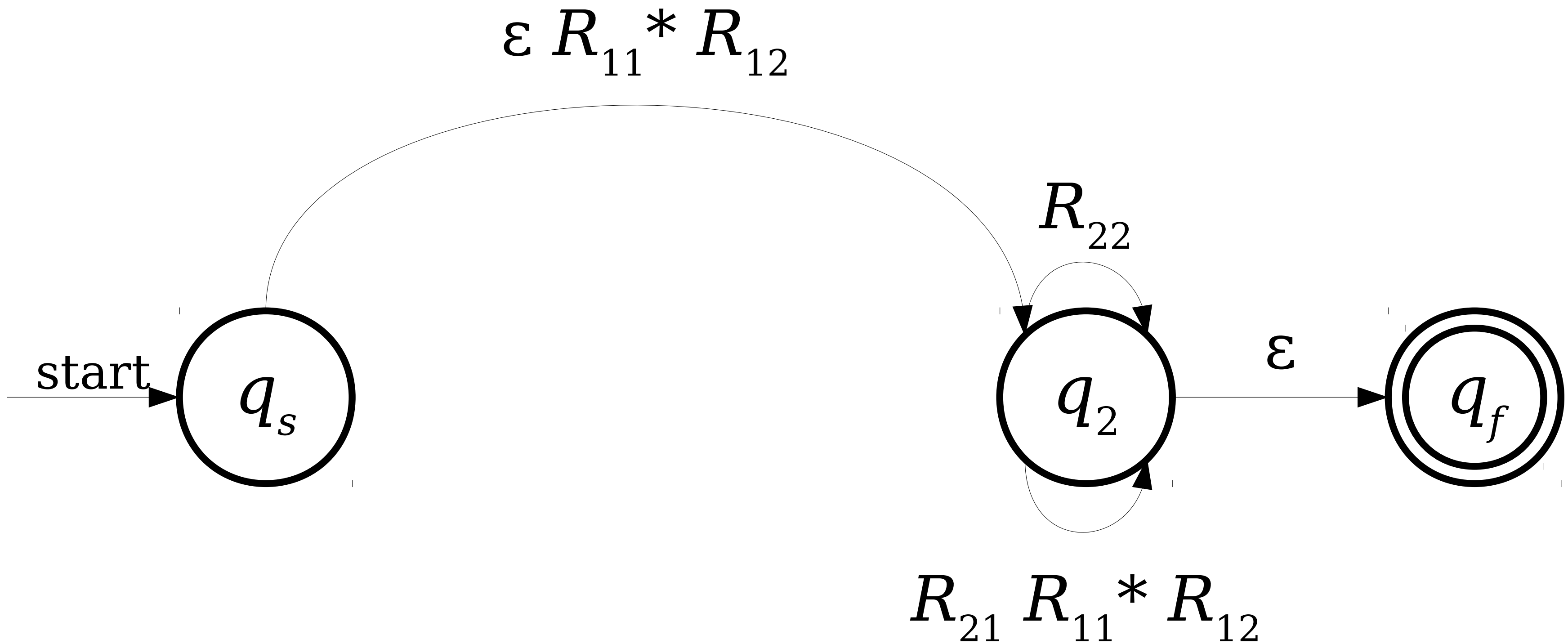


Note: We're using **concatenation** and **Kleene closure** in order to skip this state.

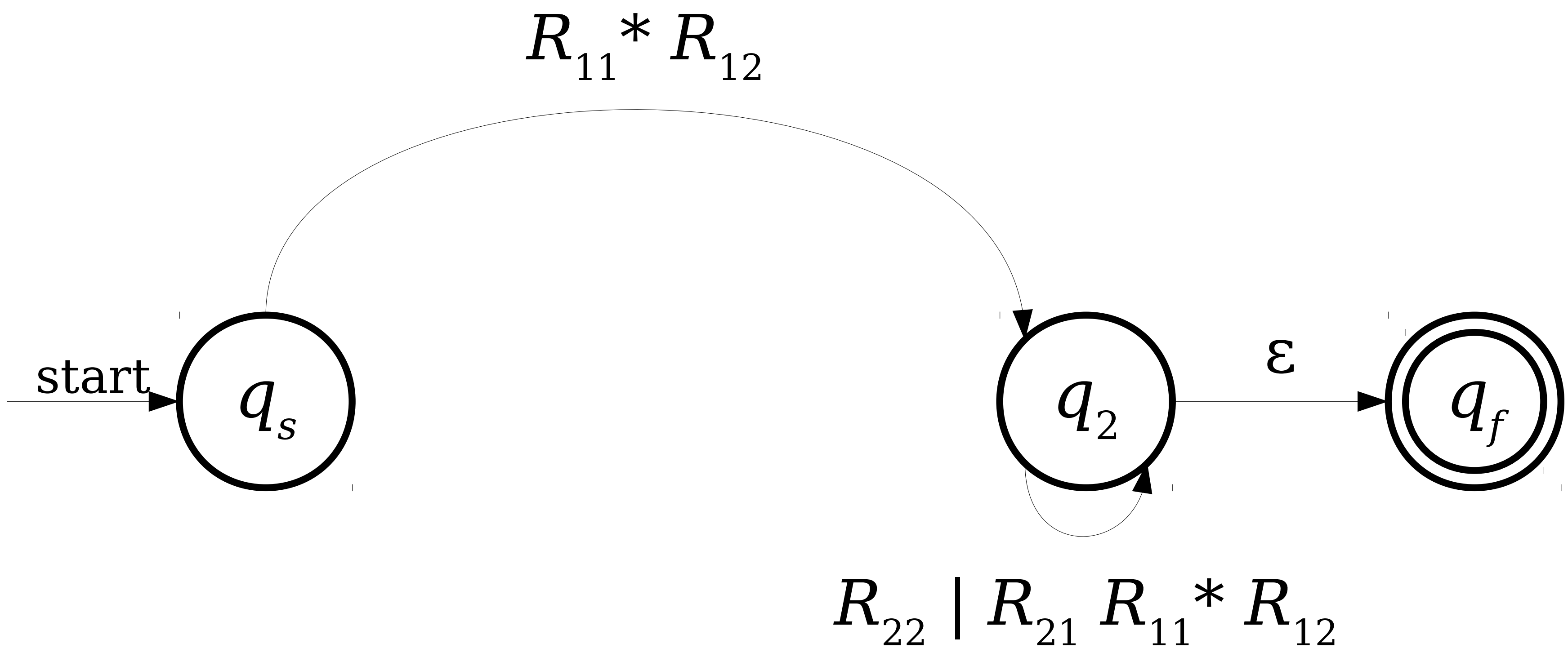
From NFAs to Regular Expressions



From NFAs to Regular Expressions

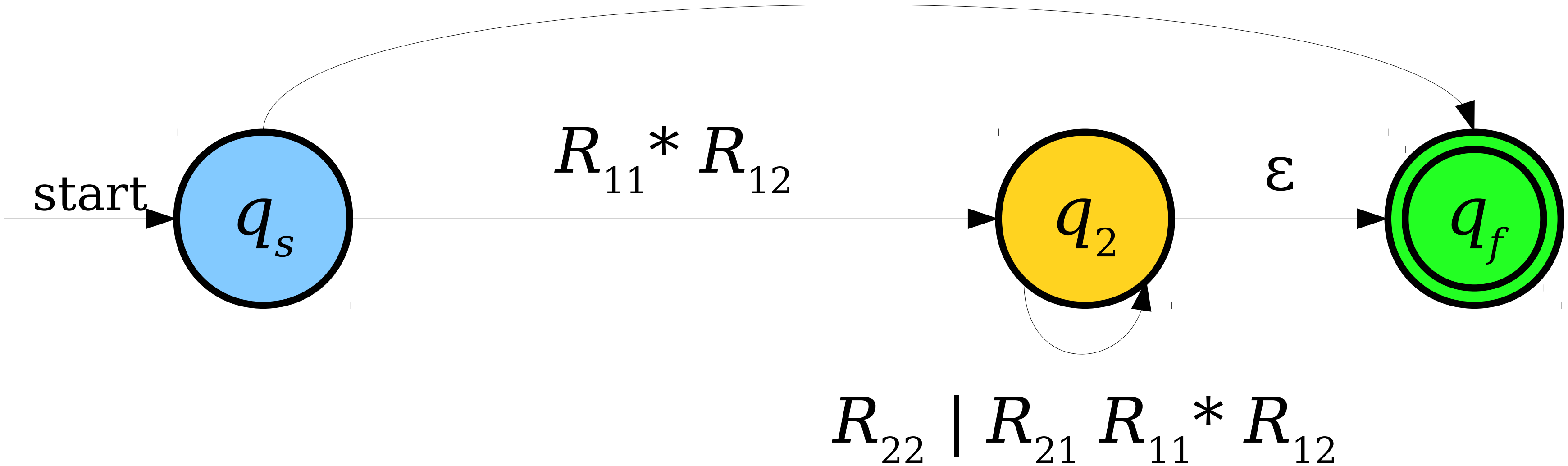


From NFAs to Regular Expressions

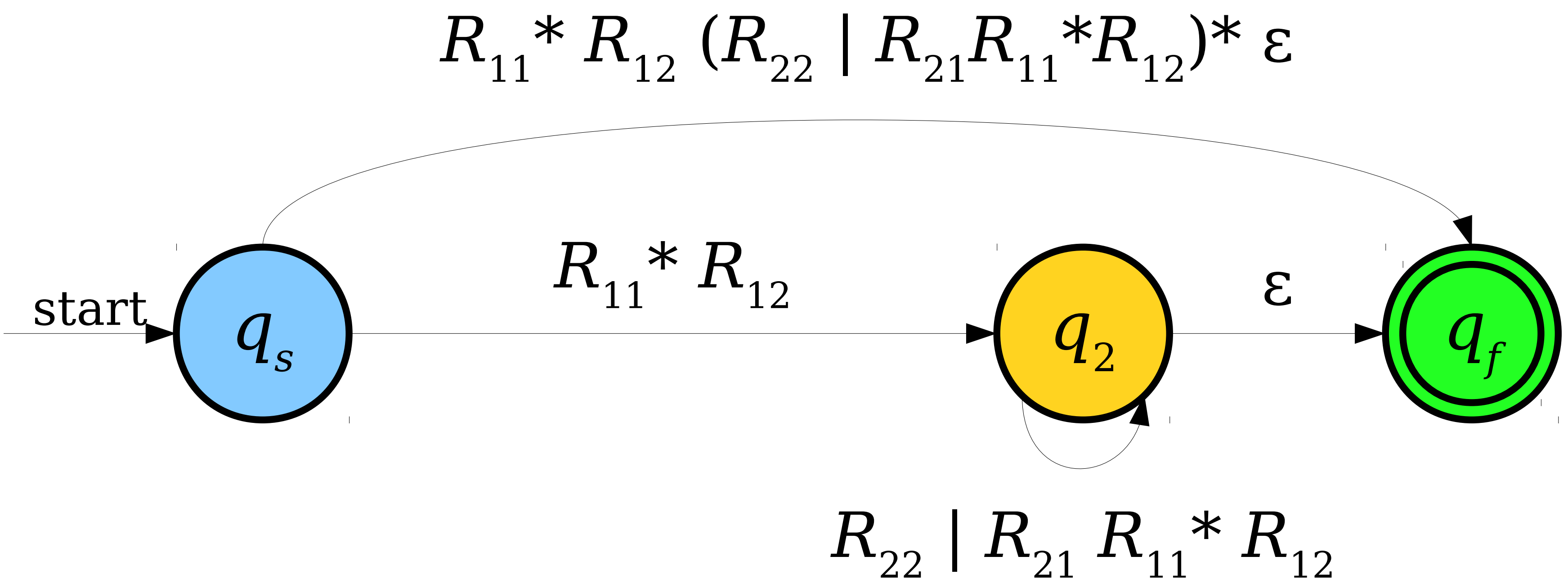


Note: We're using union to combine these transitions together.

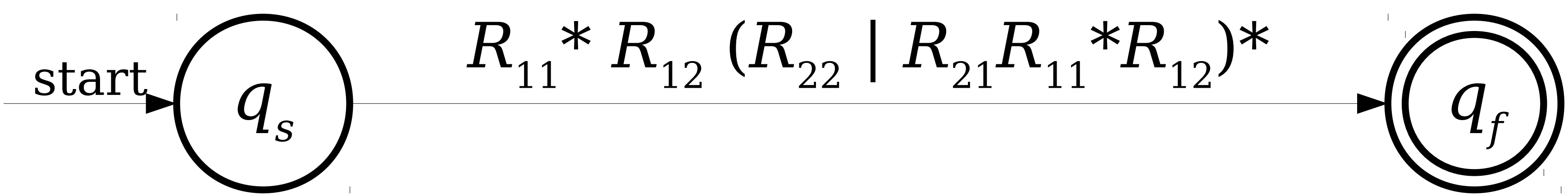
From NFAs to Regular Expressions



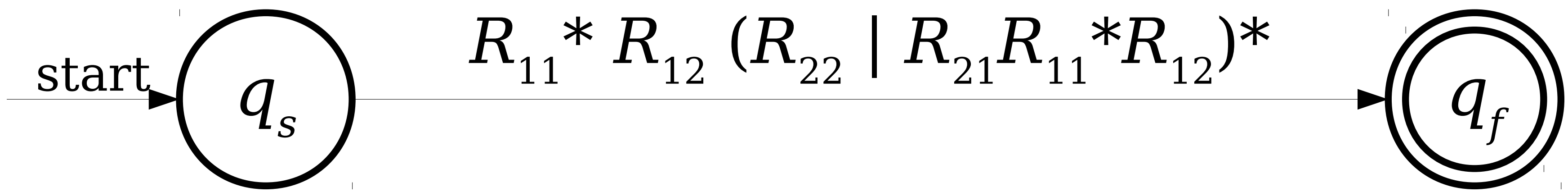
From NFAs to Regular Expressions

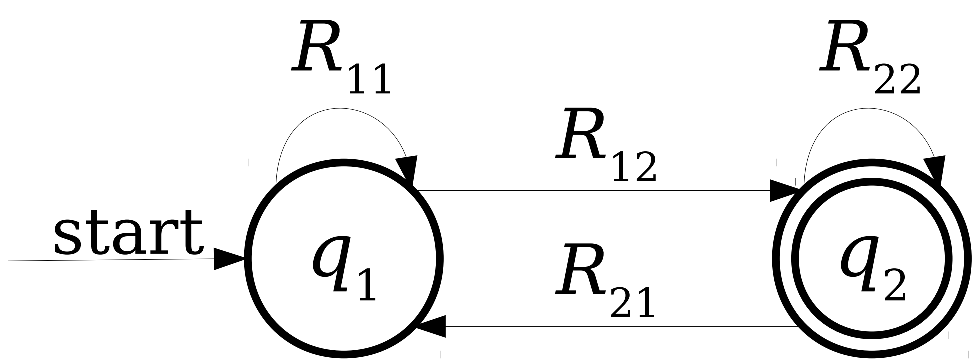


From NFAs to Regular Expressions



From NFAs to Regular Expressions





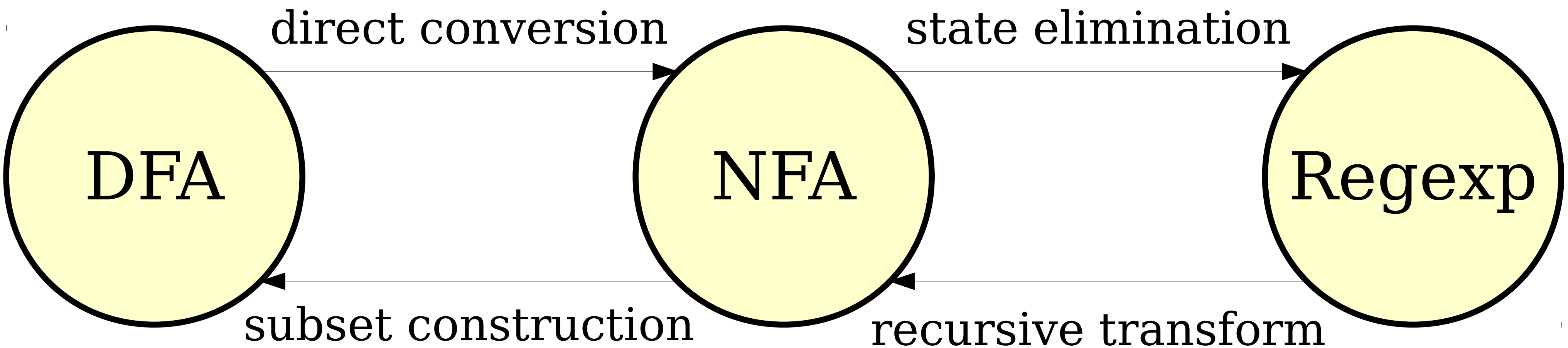
The Construction at a Glance

- 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.

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 $R_1 \mid R_2 \mid \ldots \mid R_k$.

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.