Regular Expressions

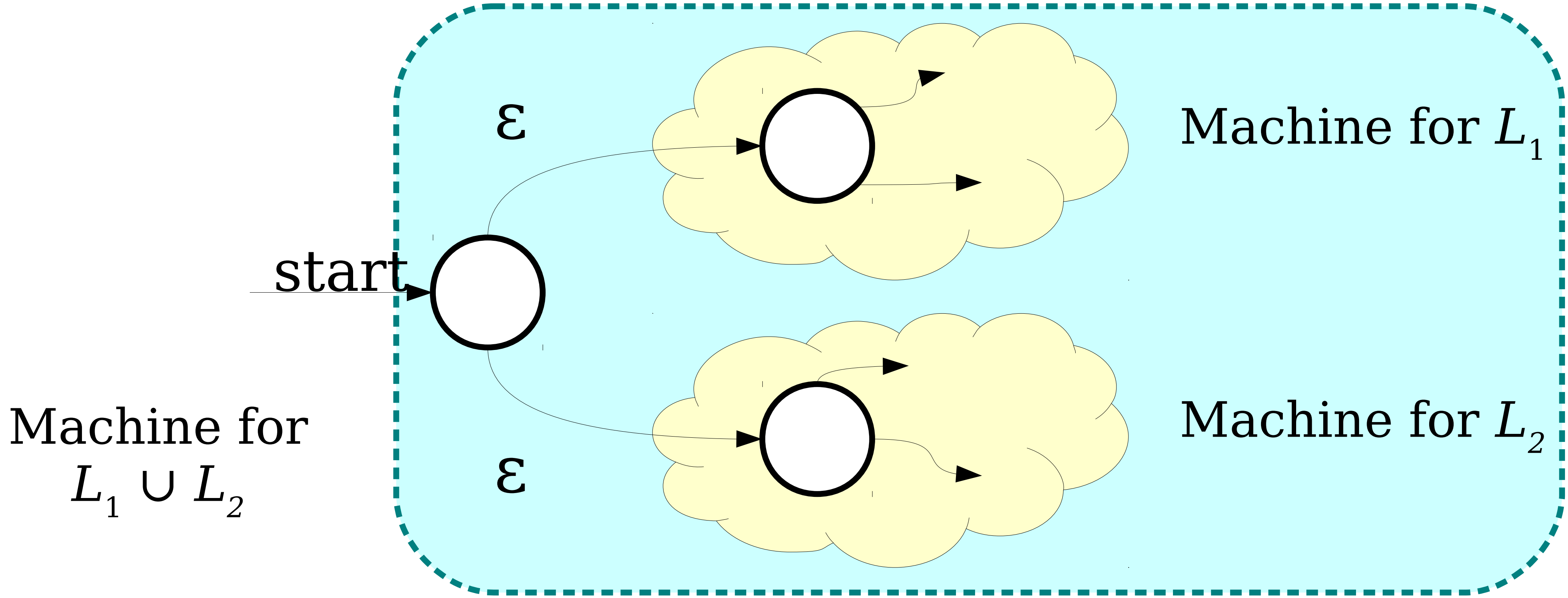
Recap from Last Time

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.

The Union of Two Languages

- If L_1 and L_2 are languages over the alphabet Σ, the language $L_1 \cup L_2$ is the language of all strings in at least one of the two languages.
- If L_1 and L_2 are regular languages, is $L_1 \cup L_2$?



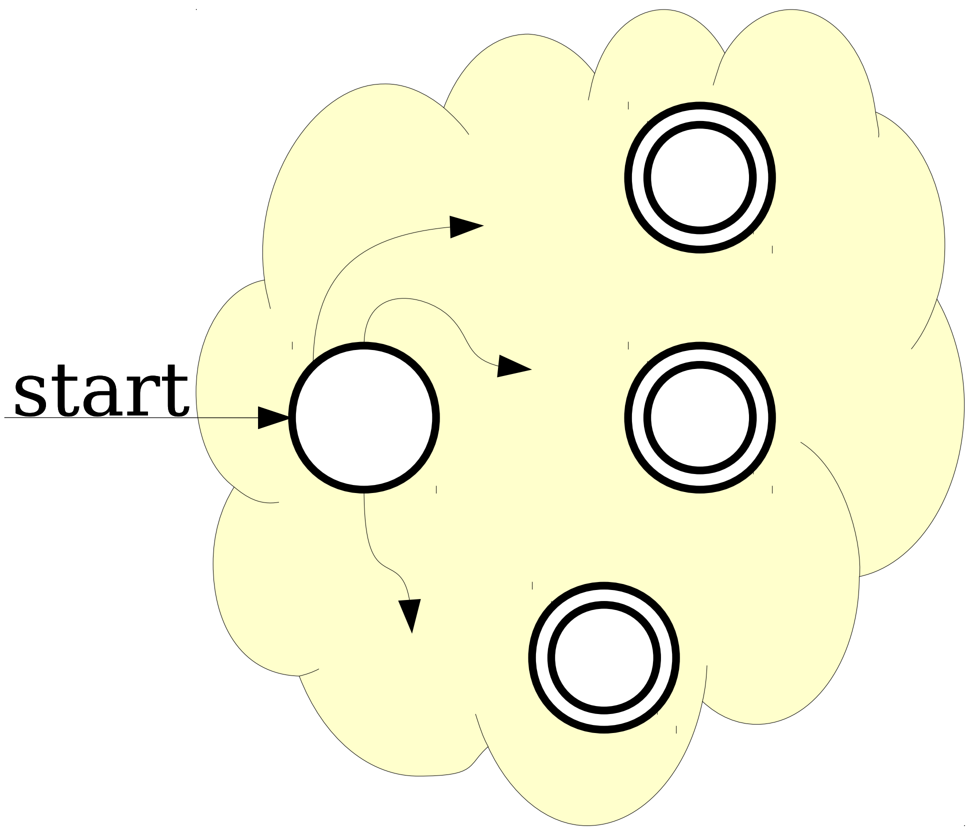
Concatenation Example

- Let Σ = { **a**, **b**, …, **z**, **A**, **B**, …, **Z** } and consider these languages over Σ:
  - ***Noun*** = { **Puppy**, **Rainbow**, **Whale**, … }
  - ***Verb*** = { **Hugs**, **Juggles**, **Loves**, … }
  - ***The*** = { **The** }
- The language ***TheNounVerbTheNoun*** is

  { **ThePuppyHugsTheWhale**,
  TheWhaleLovesTheRainbow,
  TheRainbowJugglesTheRainbow, … }

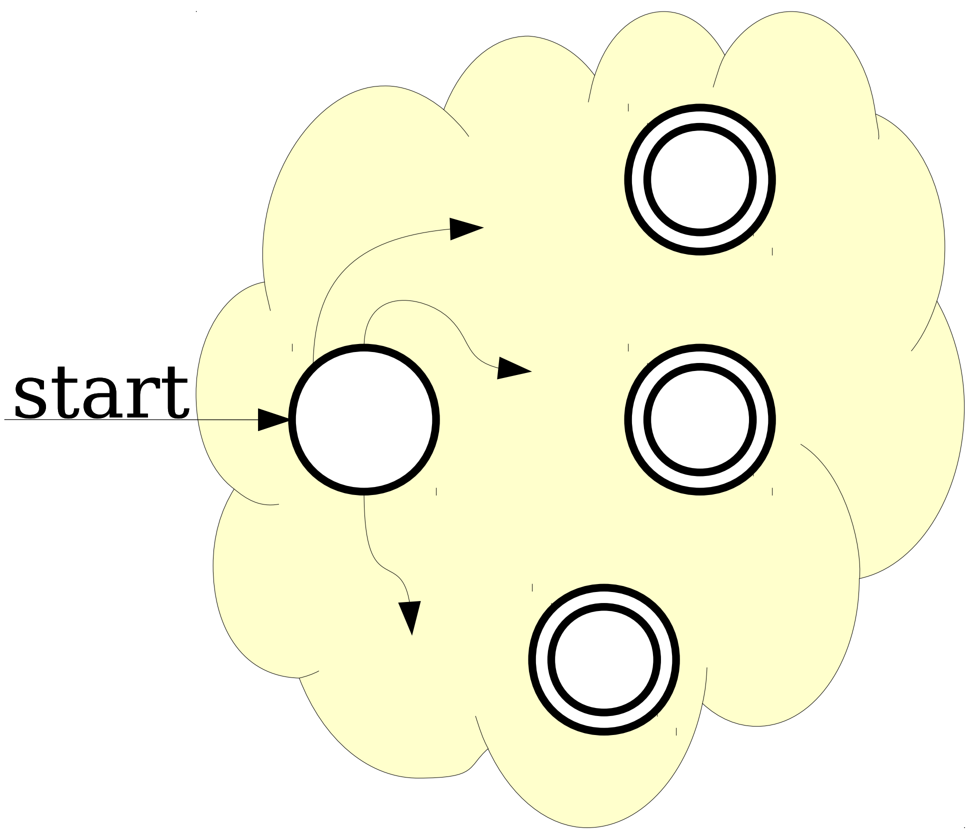
Concatenating Regular Languages

Concatenating Regular Languages

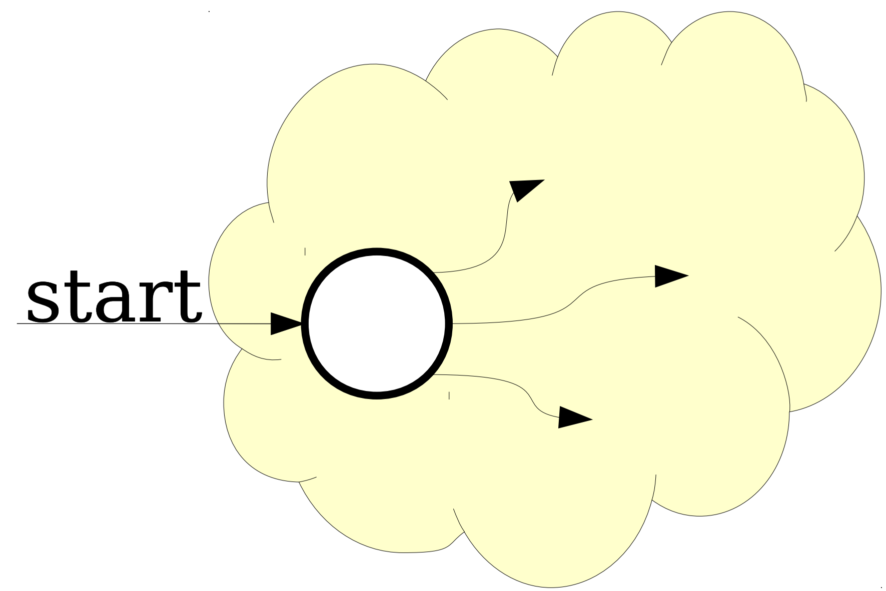


Machine for L_1

Concatenating Regular Languages

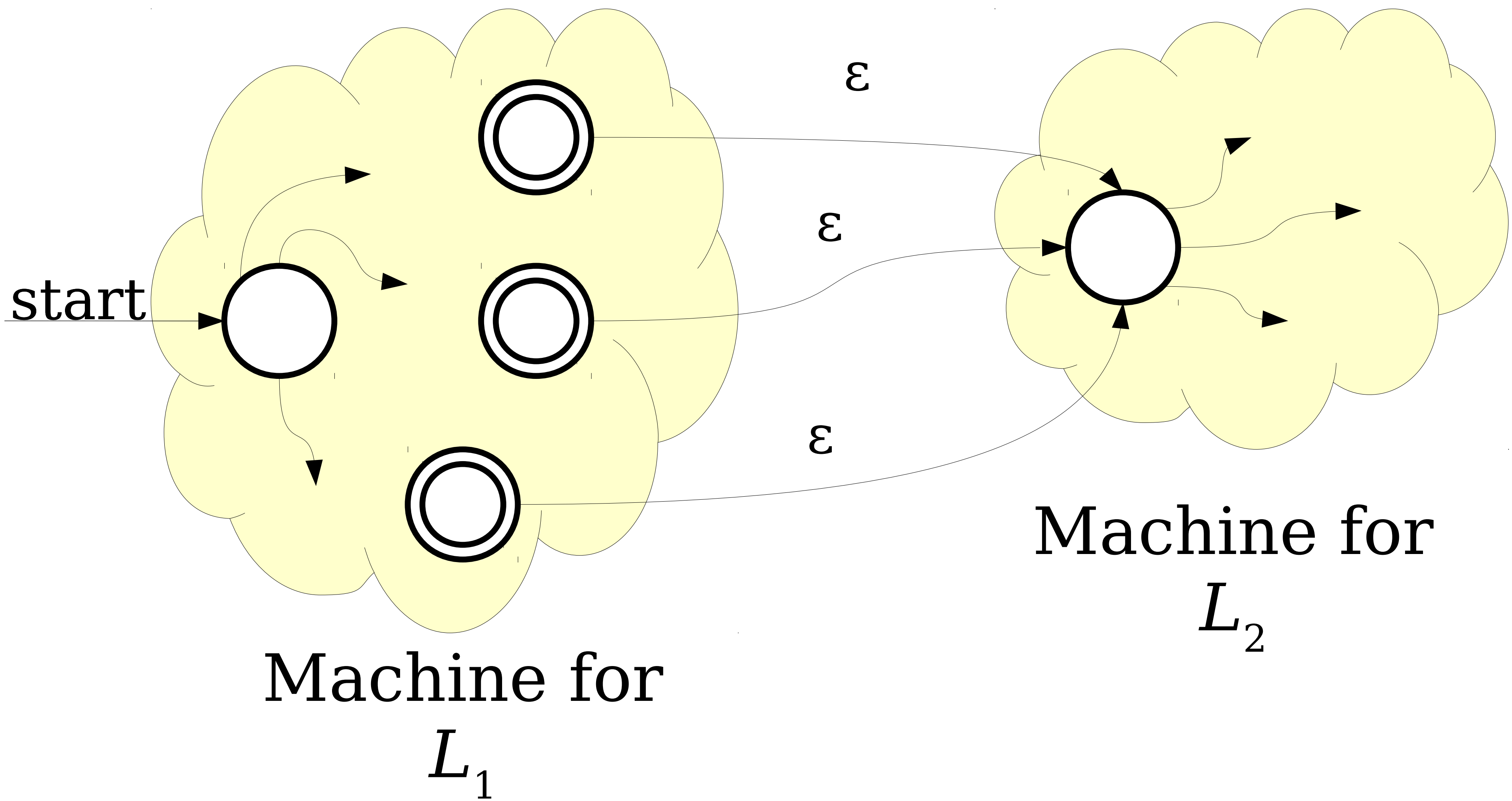


Machine for L_1

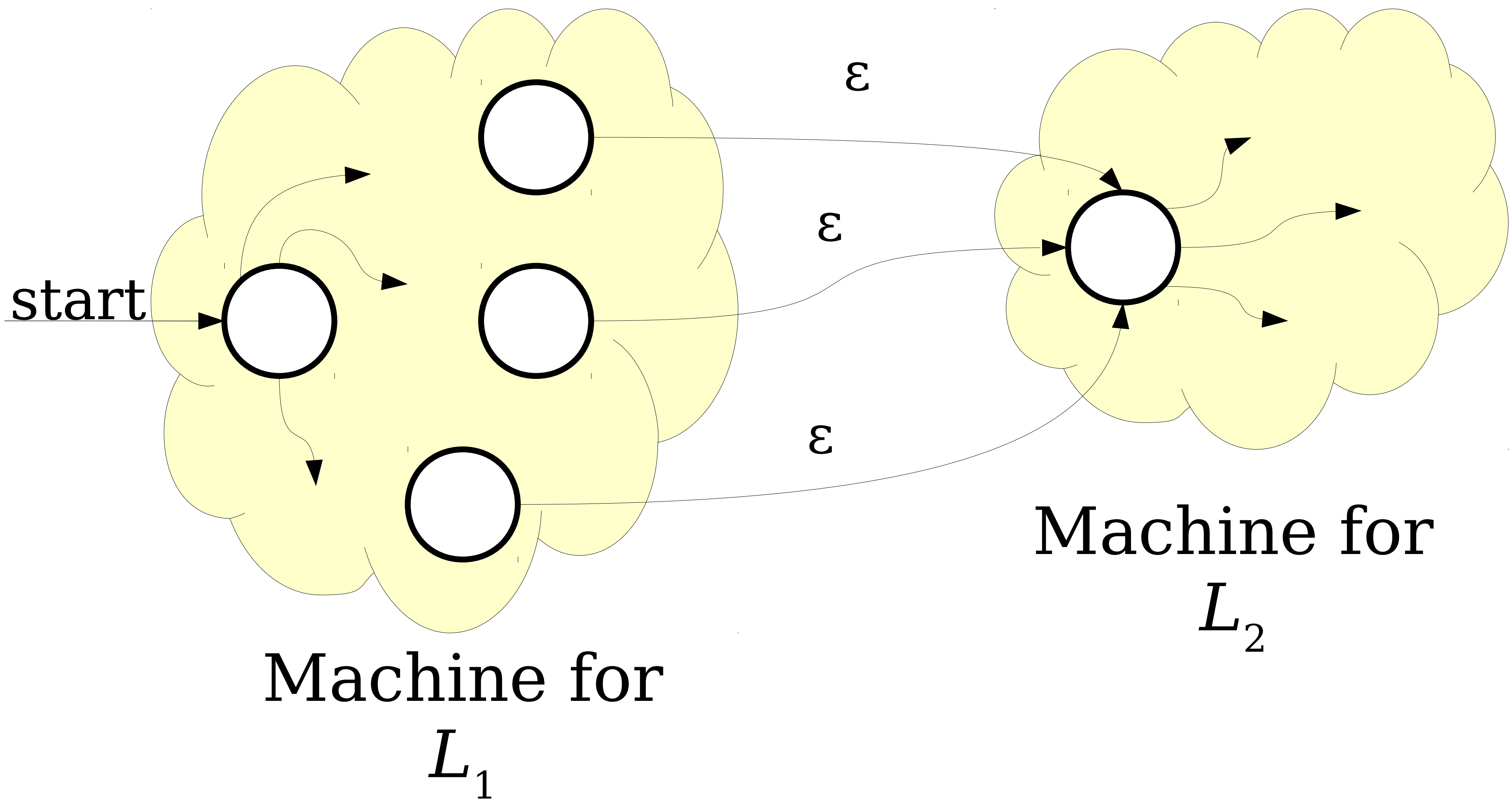


Machine for L_2

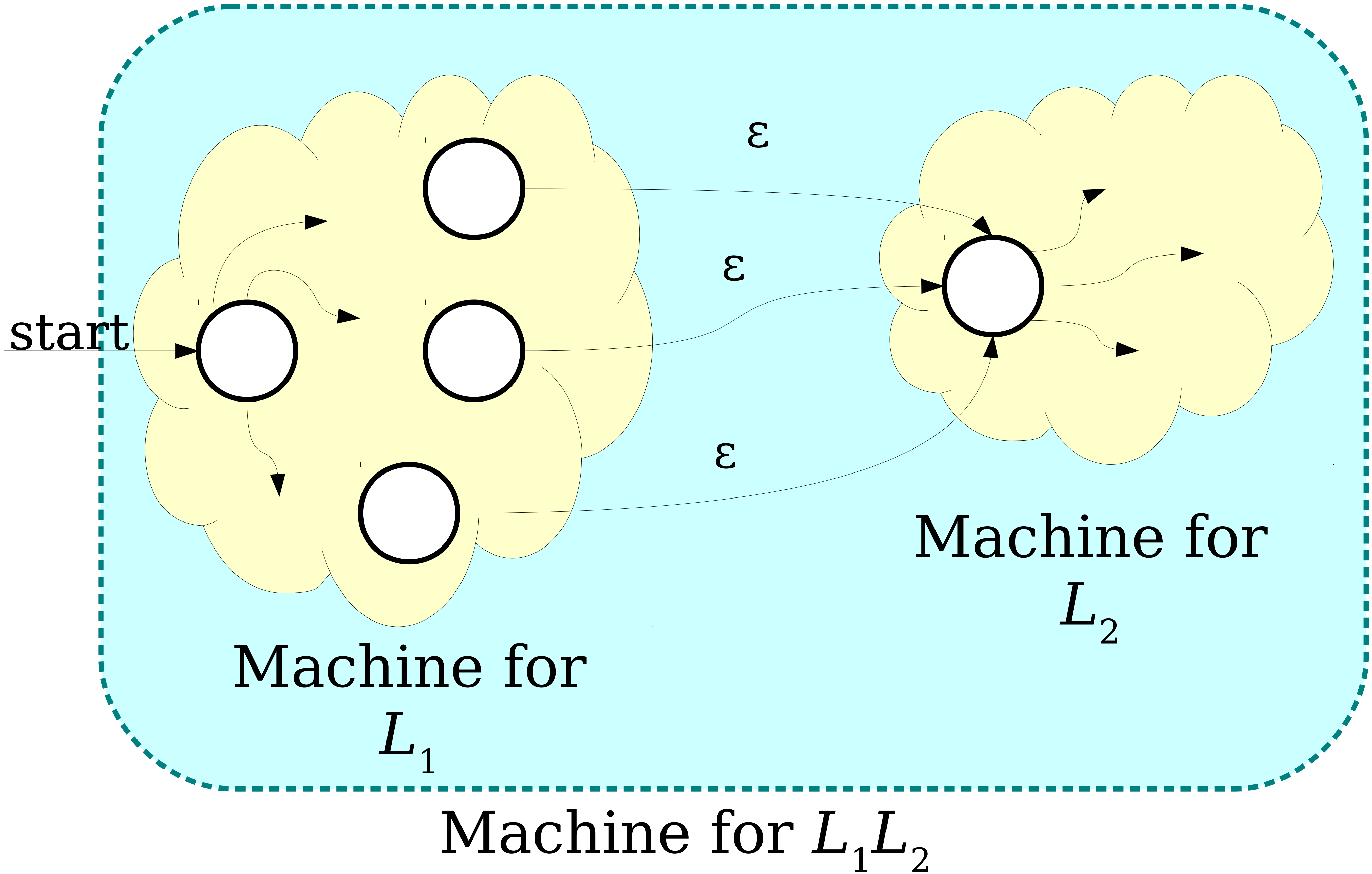
Concatenating Regular Languages



Concatenating Regular Languages



Concatenating Regular Languages



The Kleene Closure

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

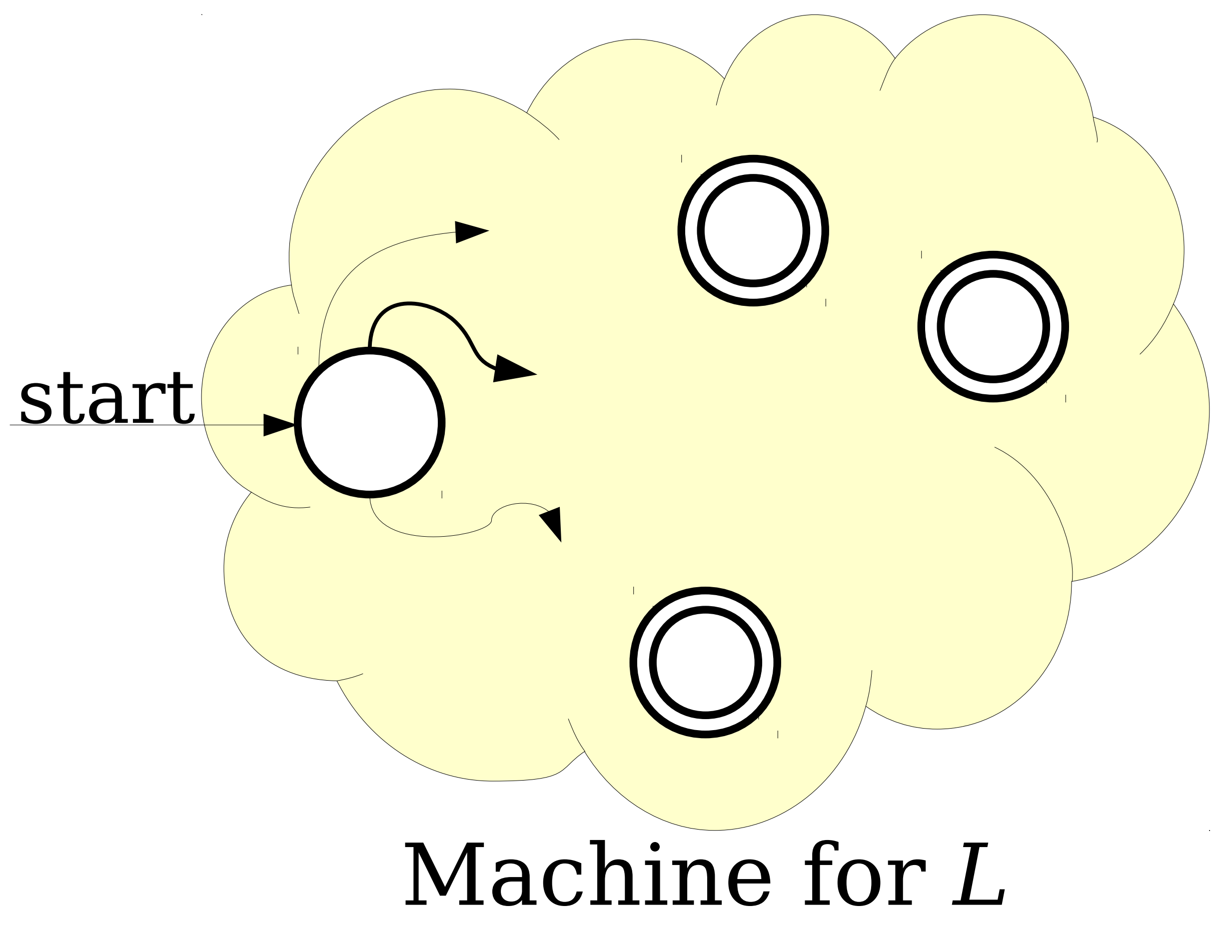
$$L^* = \bigcup_{i=0}^{\infty} L^i$$

- Mathematically:

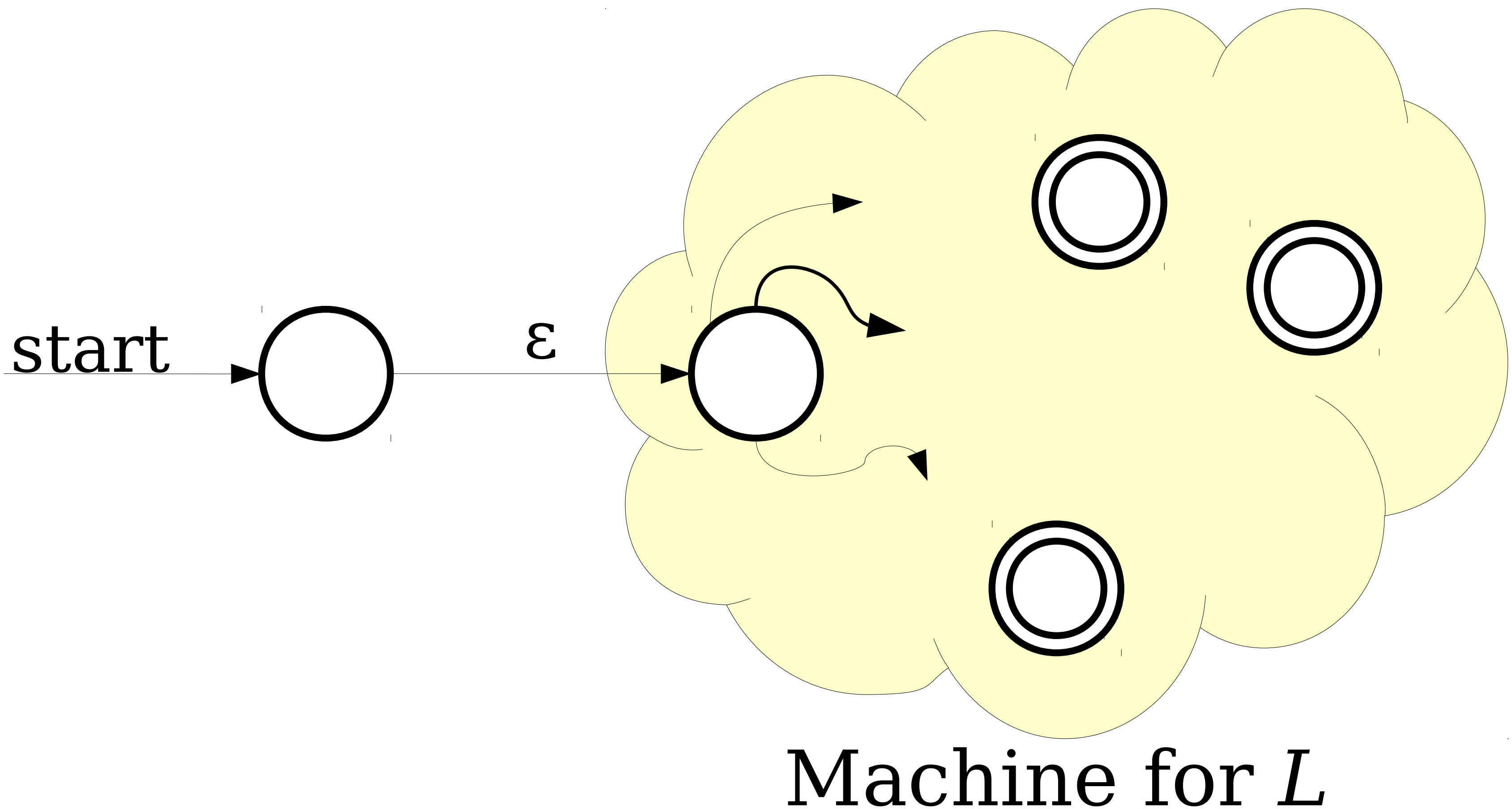
$$\boldsymbol{w \in L^*} \quad \textbf{iff} \quad \boldsymbol{\exists n \in \mathbb{N}.\ w \in L^n}$$

- Intuitively, all possible ways of concatenating any number of copies of strings in L together.

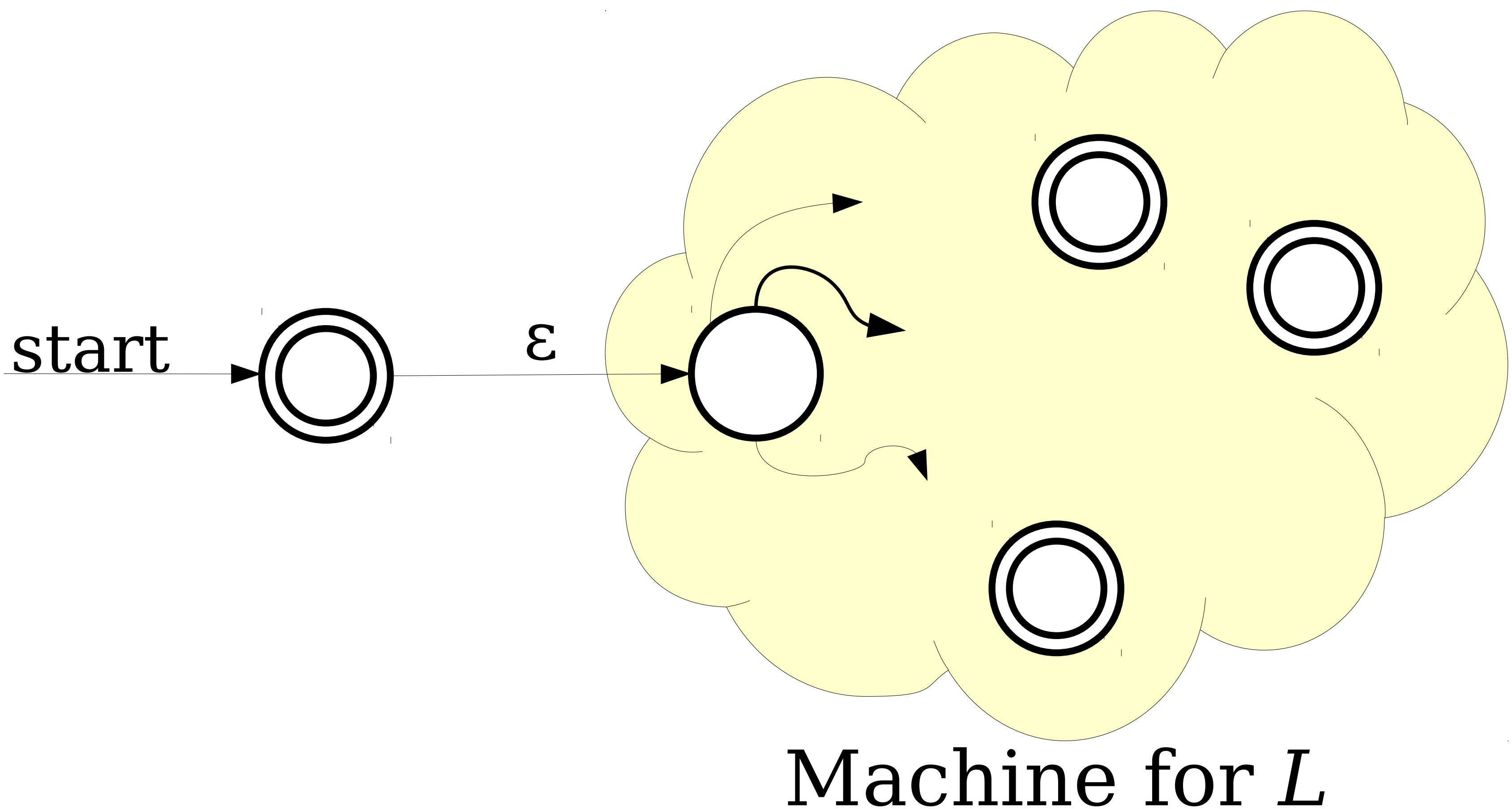
The Kleene Star



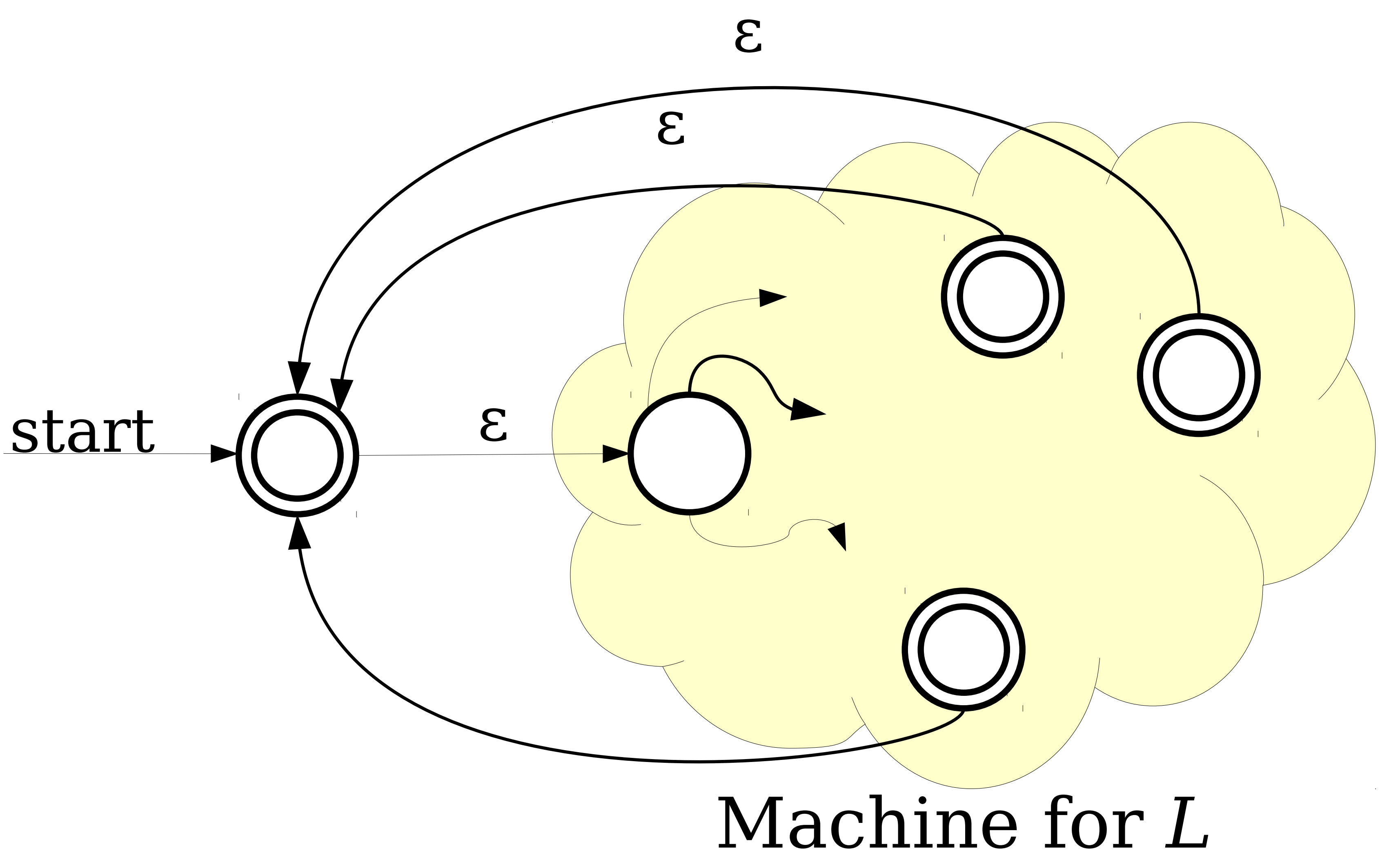
The Kleene Star



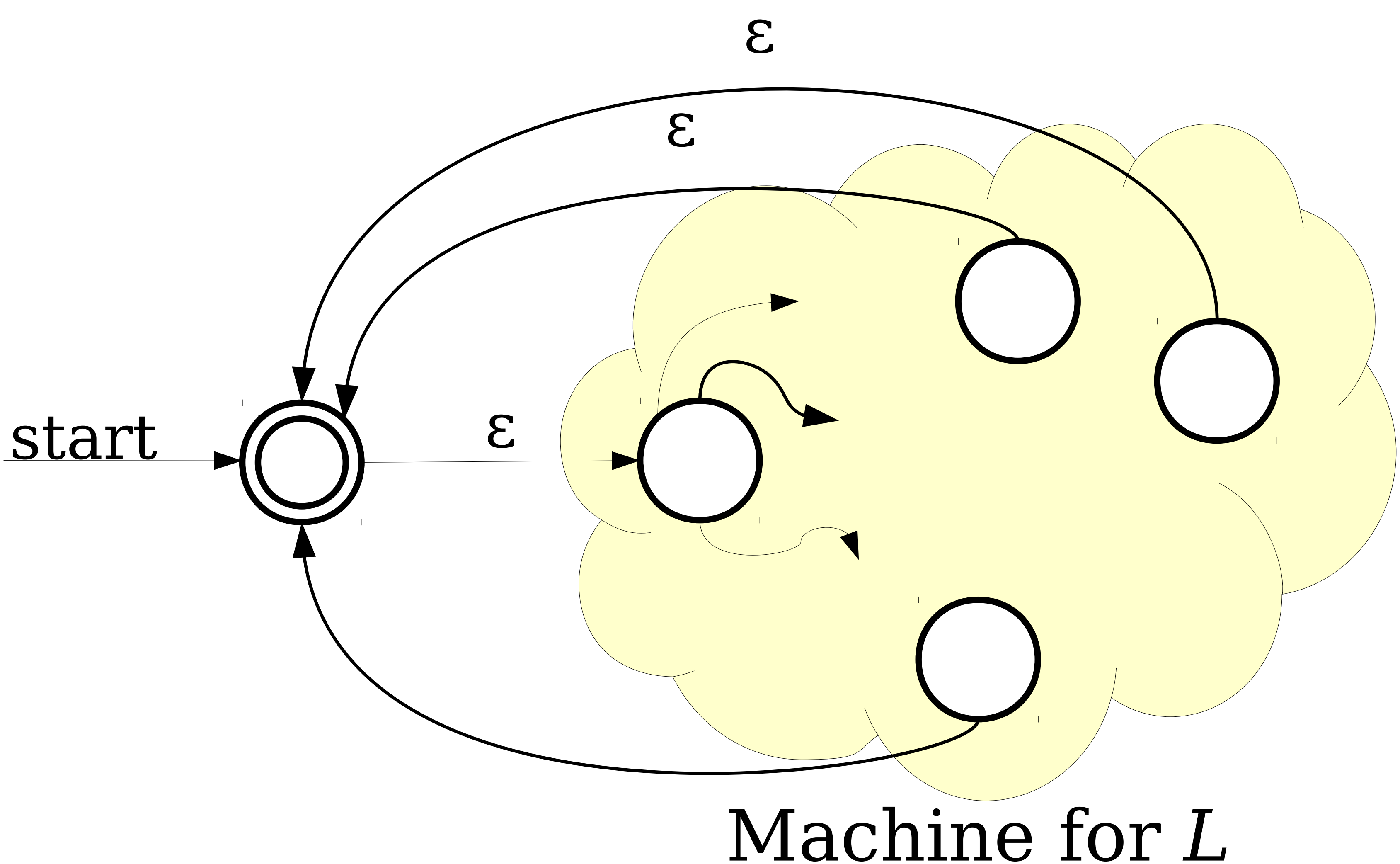
The Kleene Star



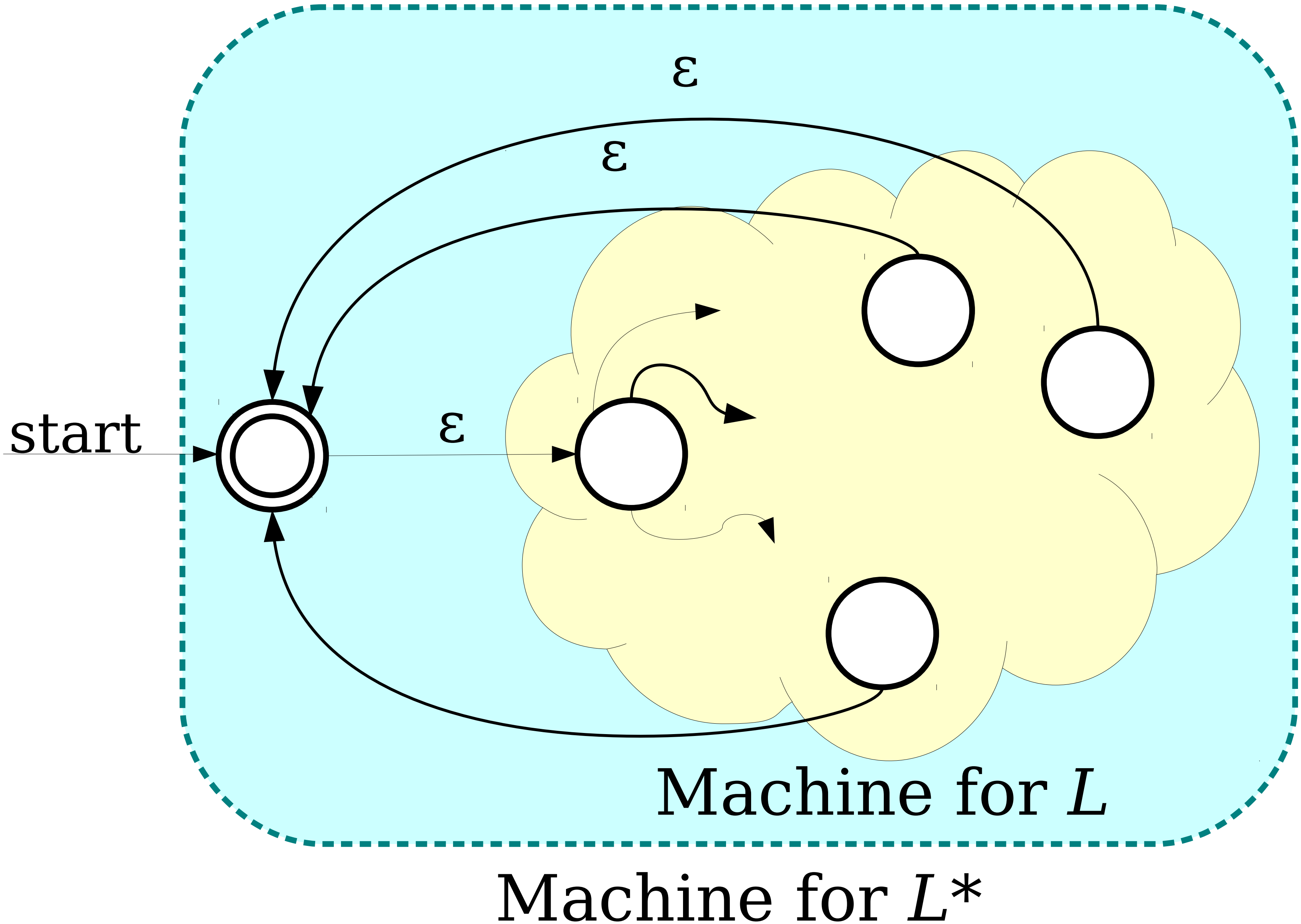
The Kleene Star



The Kleene Star



The Kleene Star



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 descriptive format used compactly describe a language.
- 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!***
- For any `a` ∈ Σ, the symbol `a` is a regular expression for the language { `a` }

Compound Regular Expressions

- We can combine together existing regular expressions in four ways.
- 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:

$$(R)$$

$$R^*$$

$$R_1 R_2$$

$$R_1 \mid R_2$$

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

Worthwhile activity: Apply this recursive definition to

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

and see what you get.

Regular Expressions are Awesome

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

Regular Expressions are Awesome

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

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

Regular Expressions are Awesome

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

$$\mathbf{(0 \mid 1)^*00(0 \mid 1)^*}$$

Regular Expressions are Awesome

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

$$\mathbf{(0 \mid 1)^*00(0 \mid 1)^*}$$

11011100101
0000
11111011110011111

Regular Expressions are Awesome

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

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

11011100101
0000
1111011110011111

Regular Expressions are Awesome

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

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

11011100101
0000
11111011110011111

Regular Expressions are Awesome

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

Regular Expressions are Awesome

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

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

Regular Expressions are Awesome

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

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

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

Regular Expressions are Awesome

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

Regular Expressions are Awesome

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

$$\mathbf{\Sigma\Sigma\Sigma\Sigma}$$

Regular Expressions are Awesome

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

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

Regular Expressions are Awesome

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

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

0000
1010
1111
1000

Regular Expressions are Awesome

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

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

0000
1010
1111
1000

Regular Expressions are Awesome

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

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

0000
1010
1111
1000

Regular Expressions are Awesome

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

Regular Expressions are Awesome

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

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

Regular Expressions are Awesome

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

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

Regular Expressions are Awesome

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

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

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:

Regular Expressions are Awesome

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

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

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:

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:

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:

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:

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

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

Regular Expressions are Awesome

- Let $\Sigma = \{$ `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:

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

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:

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

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

Regular Expressions are Awesome

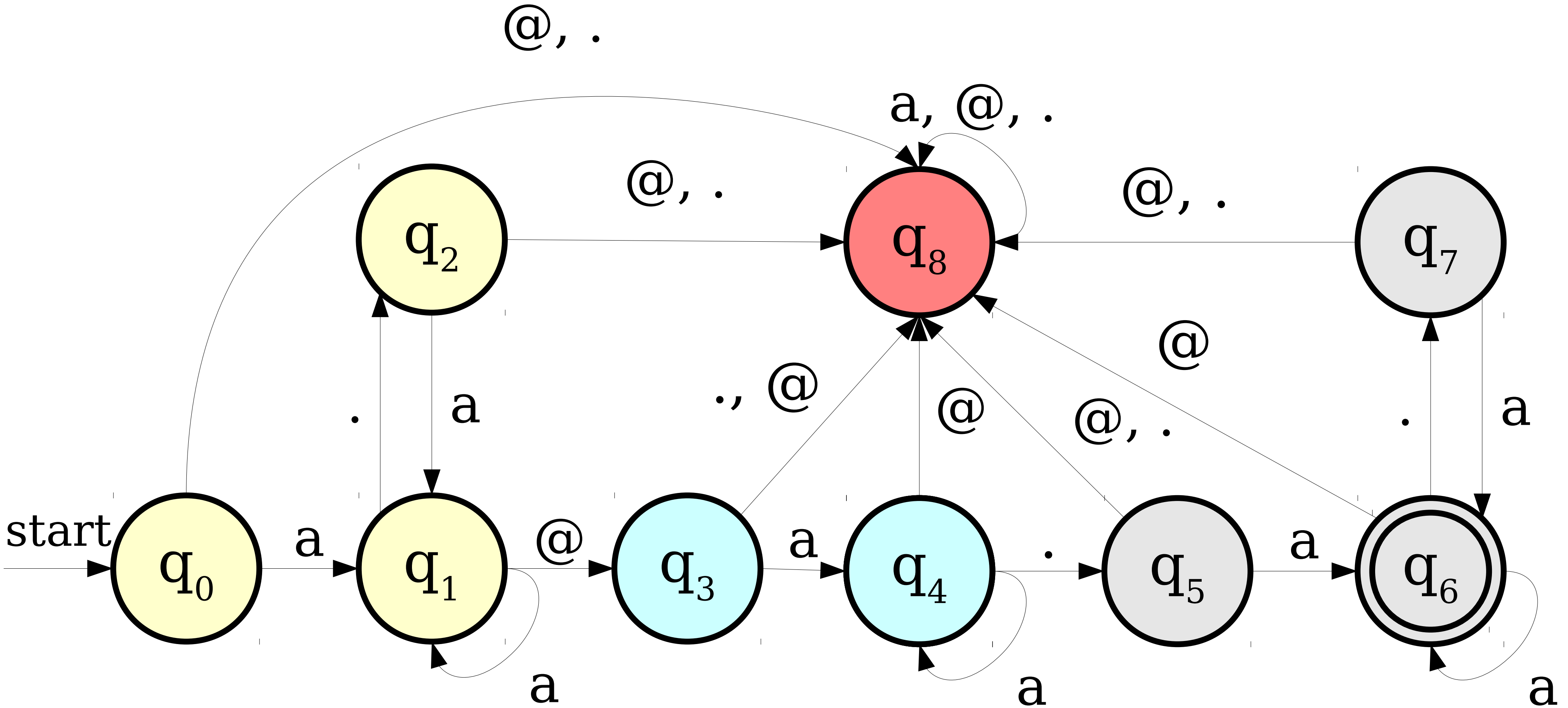
- Let Σ = { **a**, **.**, **@** }, where **a** represents "some letter."
- Regular expression for email addresses:

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

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 $RR \ldots R$ (n times).
- 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!

Upcoming Talk

- WiCS is hosting a talk by Yoky Matsuoka, VP of Technology at Nest Labs and cofounder of Google[*x*].
- Before that, she was a professor of CS, neuroscience, and ME.
- Coming up next Thursday, November 6 at 4PM at Braun Auditorium.
- RSVP requested: http://goo.gl/forms/KwDpCiUfHn

STEM Fellows Program

- “[T]he Stanford Undergraduate STEM Fellows Program provides support to students who will promote the diversity (broadly defined) of the future professoriate. The program [… seeks] to increase the number of PhDs earned by under-represented groups in the areas of science, technology, engineering, and math.”
- Deadline is December 8, but I'd encourage applying early.
- Details online at

  https://undergrad.stanford.edu/opportunities-research/fellowships/fellowships-listing/stanford-undergraduate-stem-fellows-program

Problem Set Four Solutions

- PS4 solutions will be available outside of class today and in the filing cabinet after that.
  - SCPD students: you should receive them soon.
- Want older solution sets? Pick them up soon before they get recycled!

Order in a World of Chaos

- *Please keep the box of exams alphabetized*. Everyone pays if even a small number of people scramble the exams.

Your Questions

“I've noticed that you're very passionate about diversity in CS. What advice would you have for someone who is part of a minority and is about to have a job/internship in an environment that is not likely to be diverse?”

“In your opinion, what's the biggest number?”

“What do you do in your free time?”

“Why don't we have a class called Data Structures? Is 106B/X our equivalent of this? Recruiters and interviewers ask me why I haven't taken data structures all the time and I never know how to respond.”

Back to CS103!

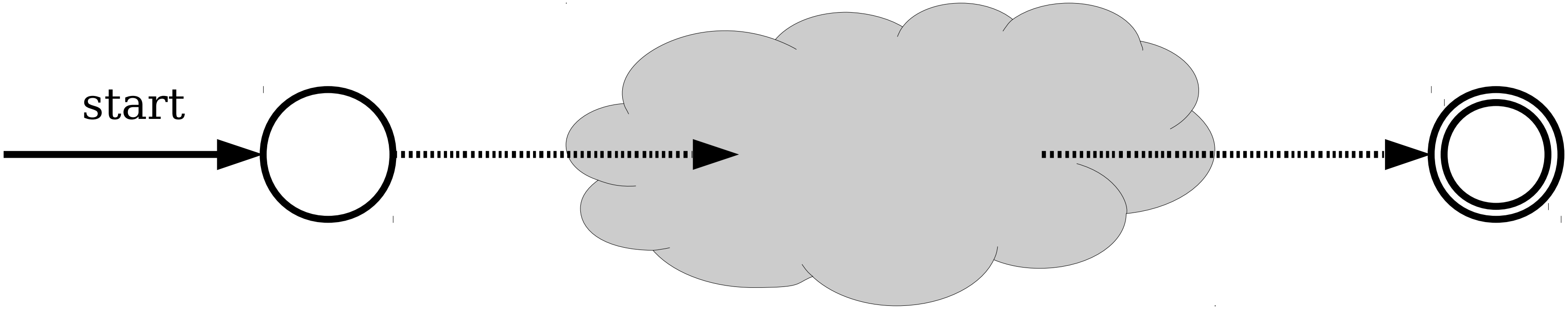
The Power of Regular Expressions

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.



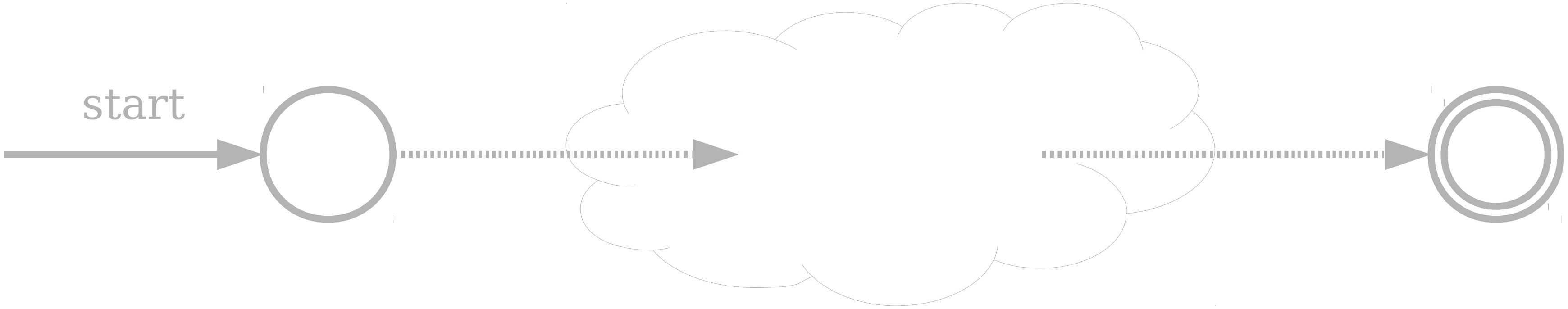
A Marvelous Construction

The following theorem 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

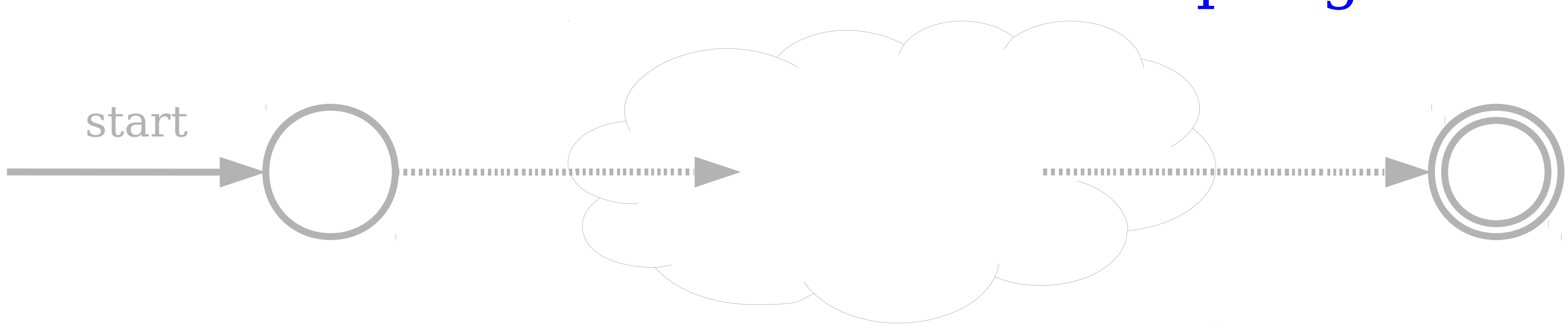
The following theorem p
any regular expression i

Theorem: For any regu
is an NFA N such that

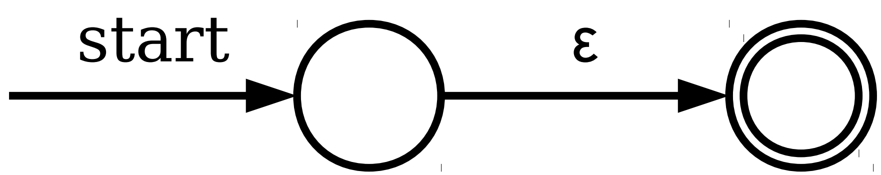
$$\mathscr{L}(R) = \mathscr{L}(N)$$

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

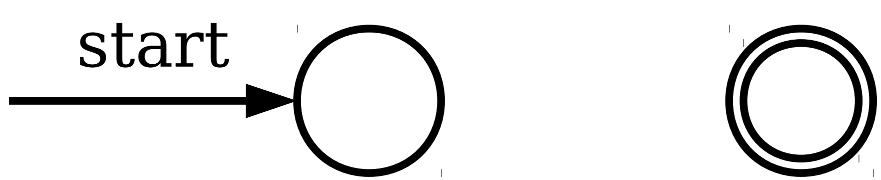
- N has exactly one accepting state.
- N has no transitions into its start state.
- N has no transitions out of its accepting state.



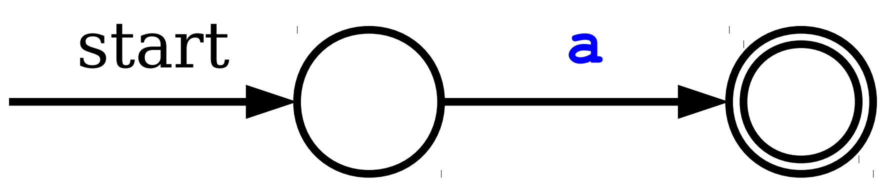
Base Cases



Automaton for ε



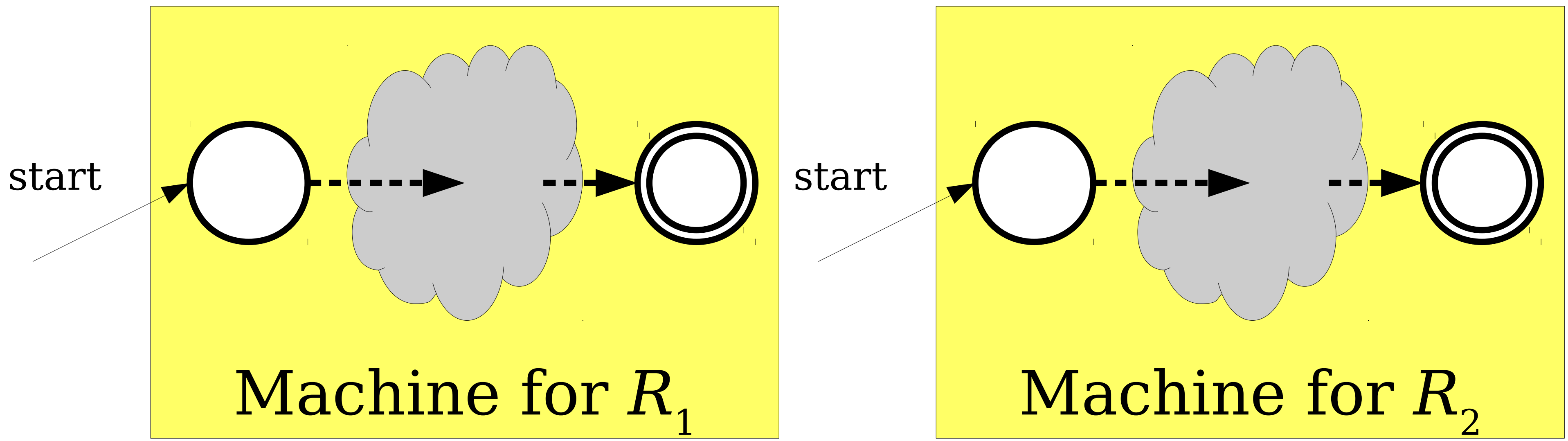
Automaton for Ø



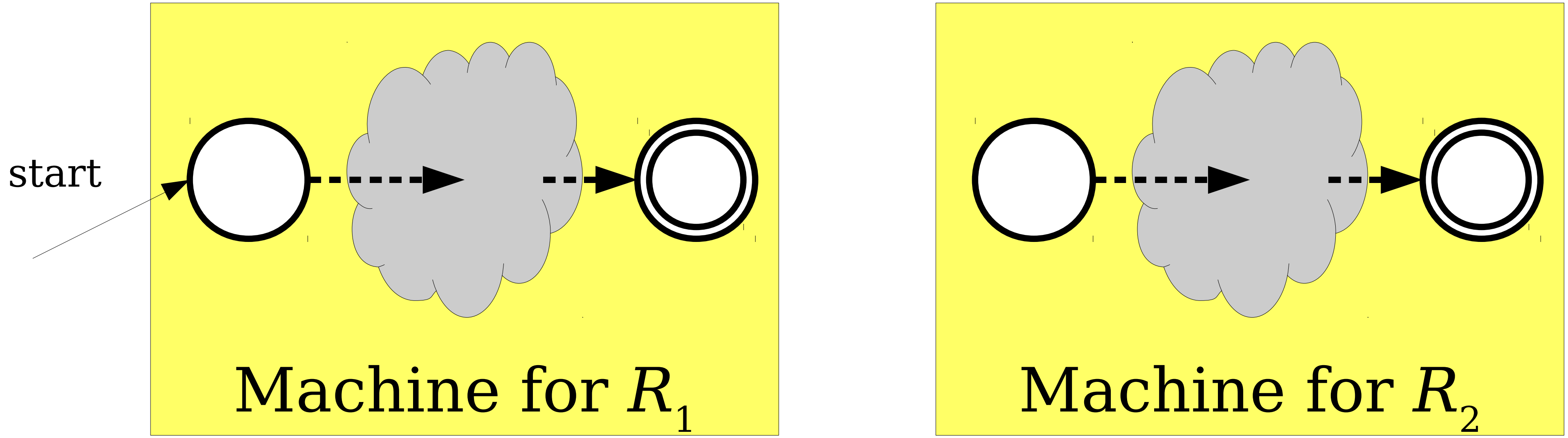
Automaton for single character **`a`**

Construction for R_1R_2

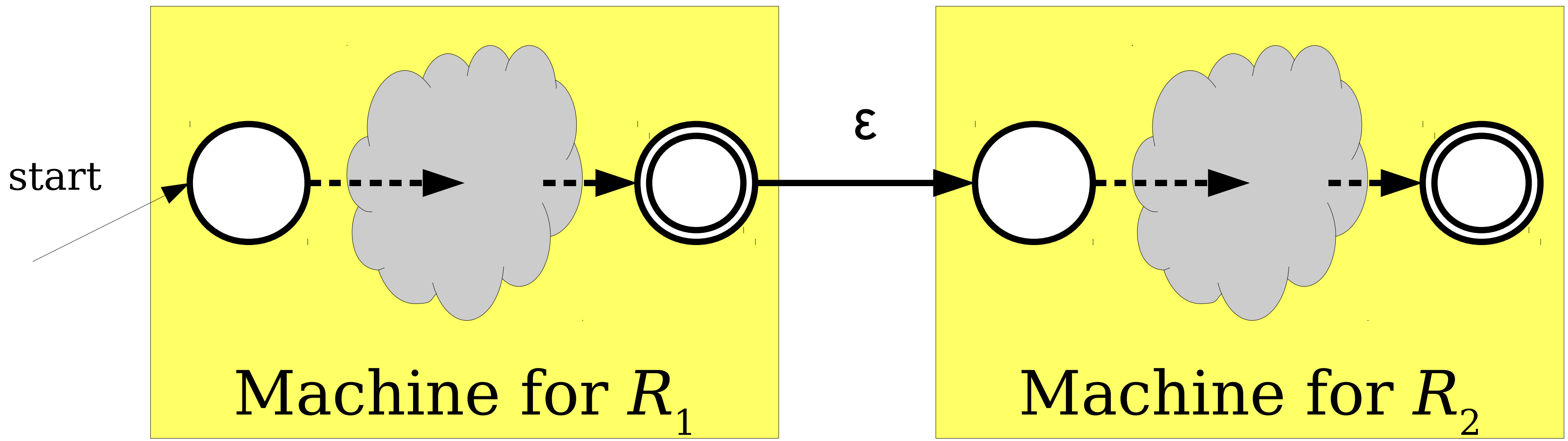
Construction for R_1R_2



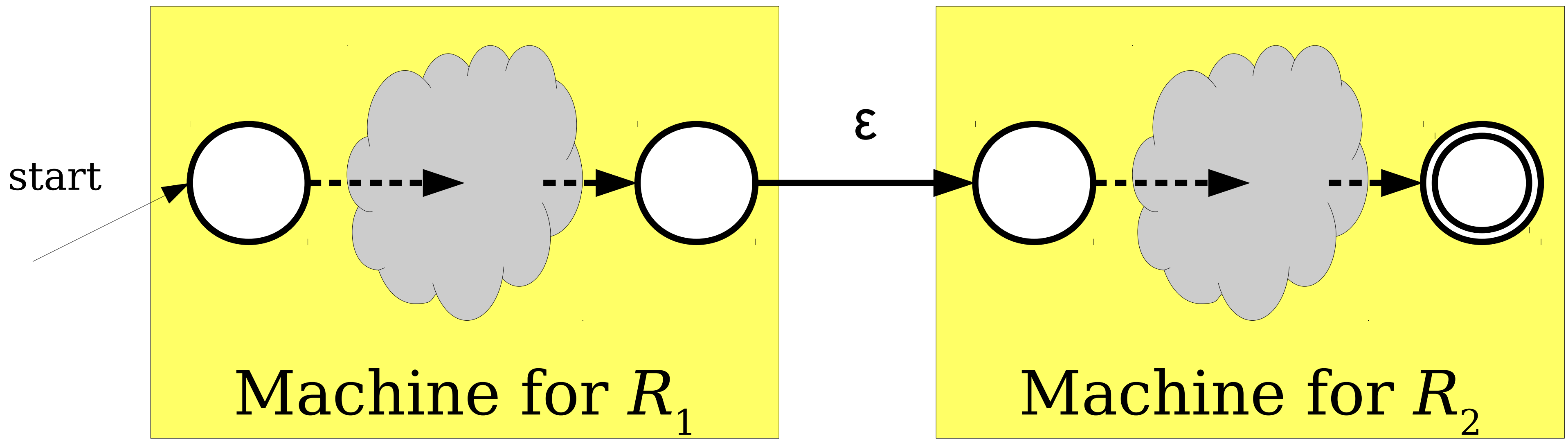
Construction for R_1R_2



Construction for R_1R_2

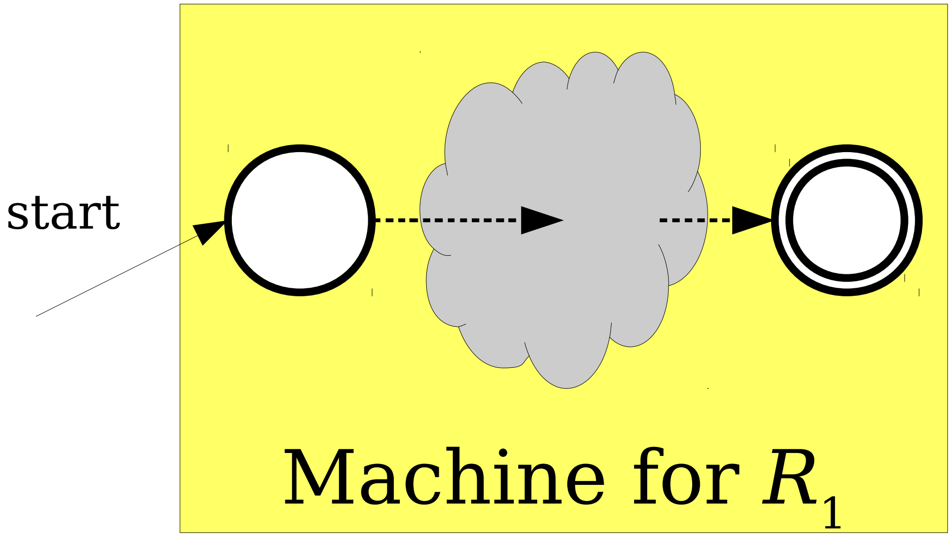


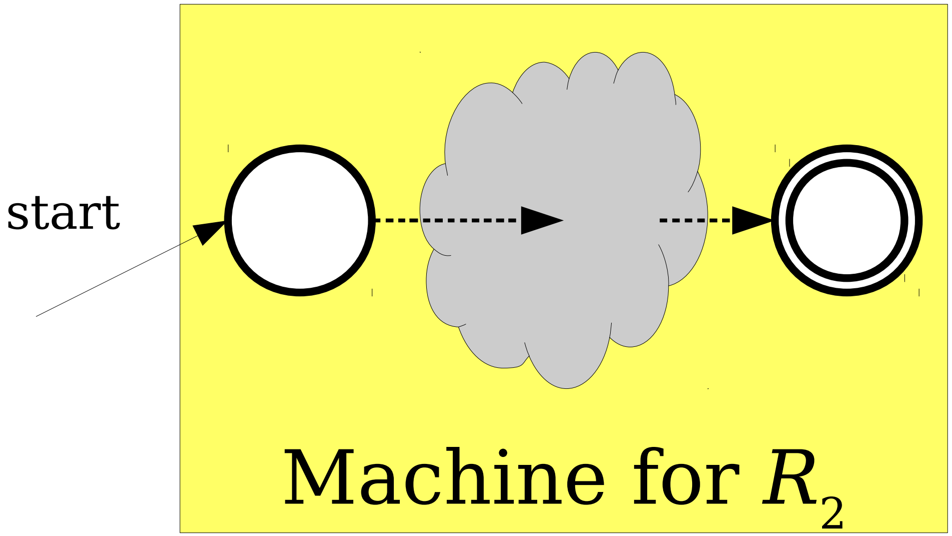
Construction for R_1R_2



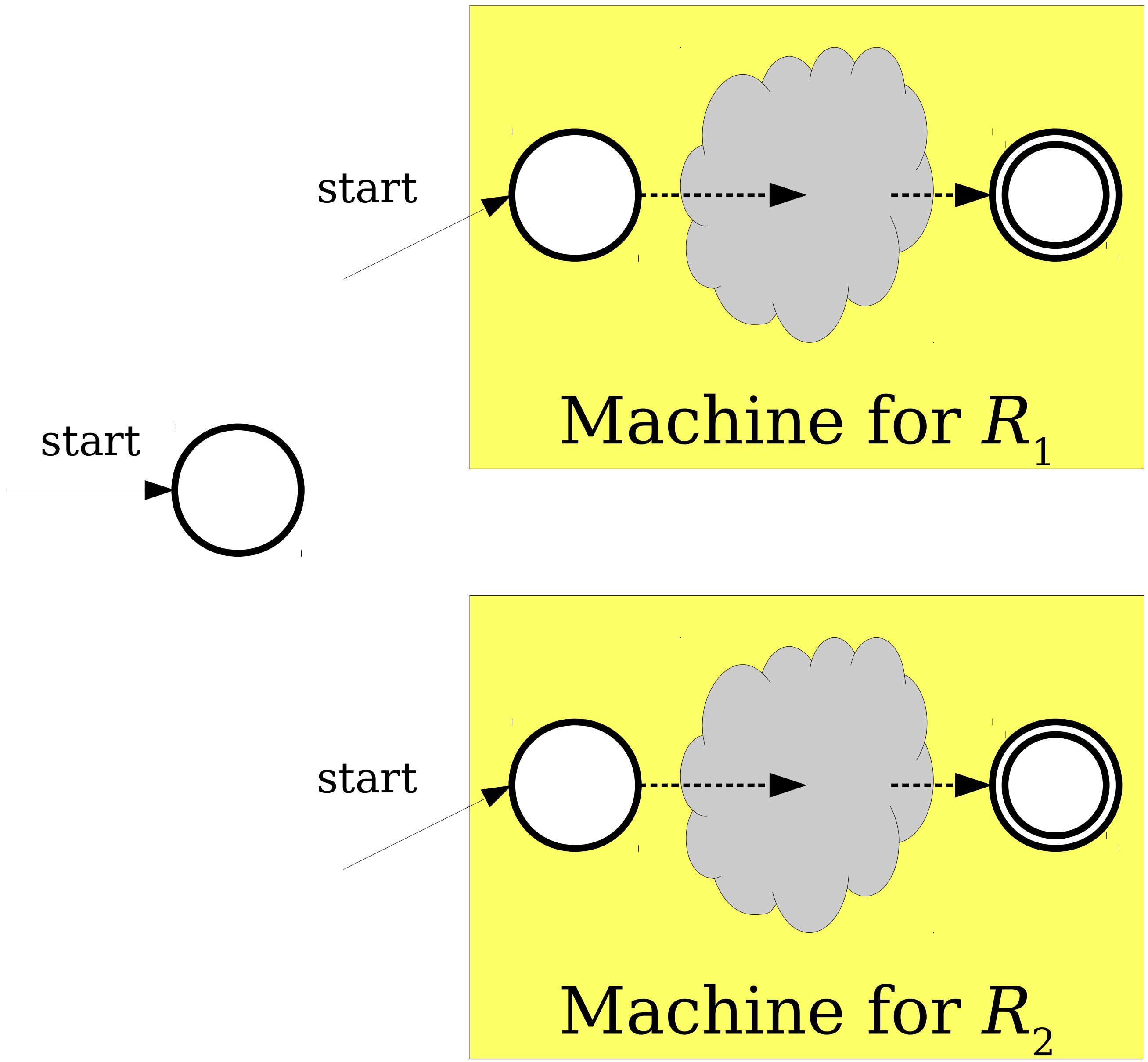
Construction for $R_1 \mid R_2$

Construction for R_1 | R_2

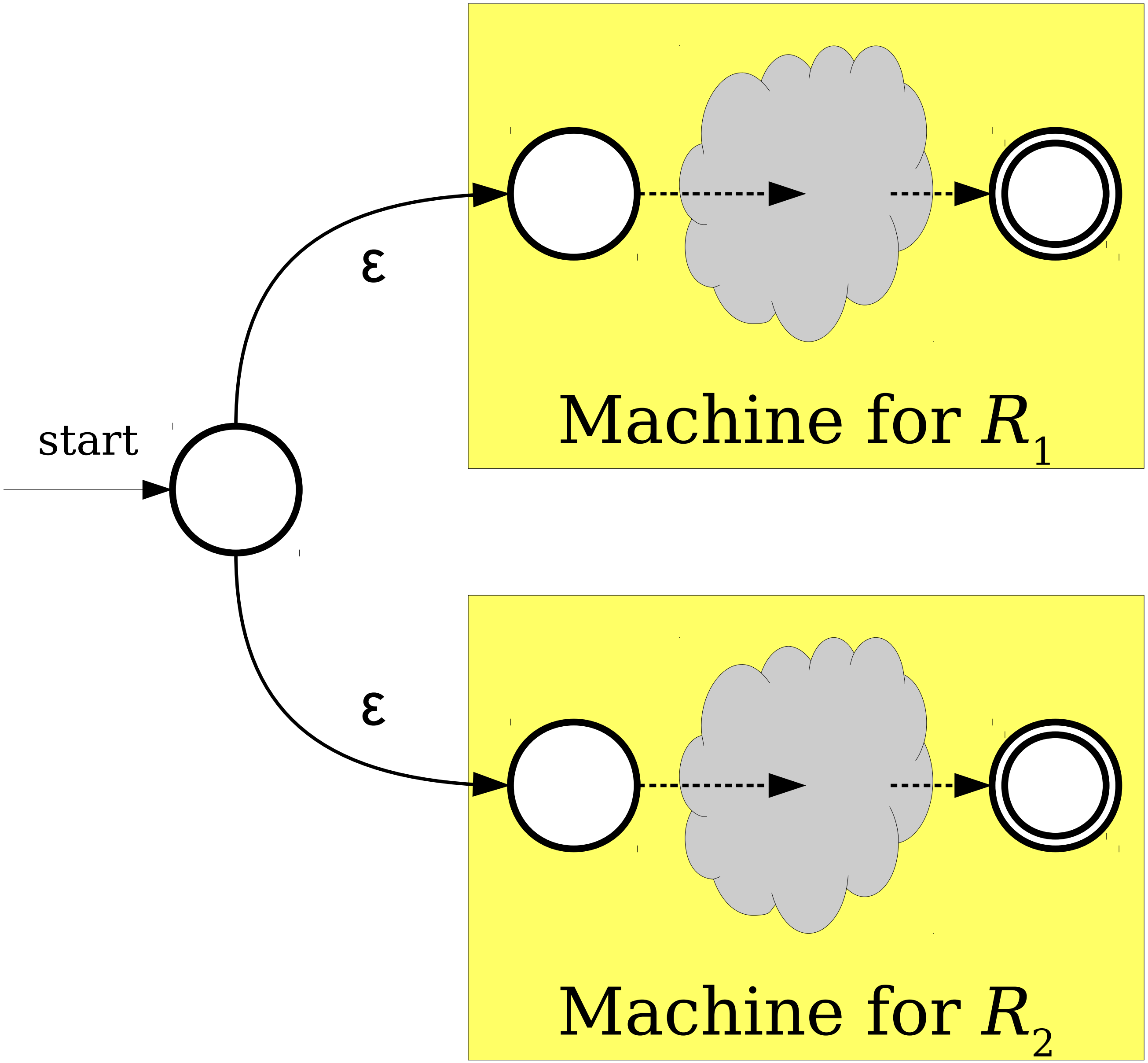




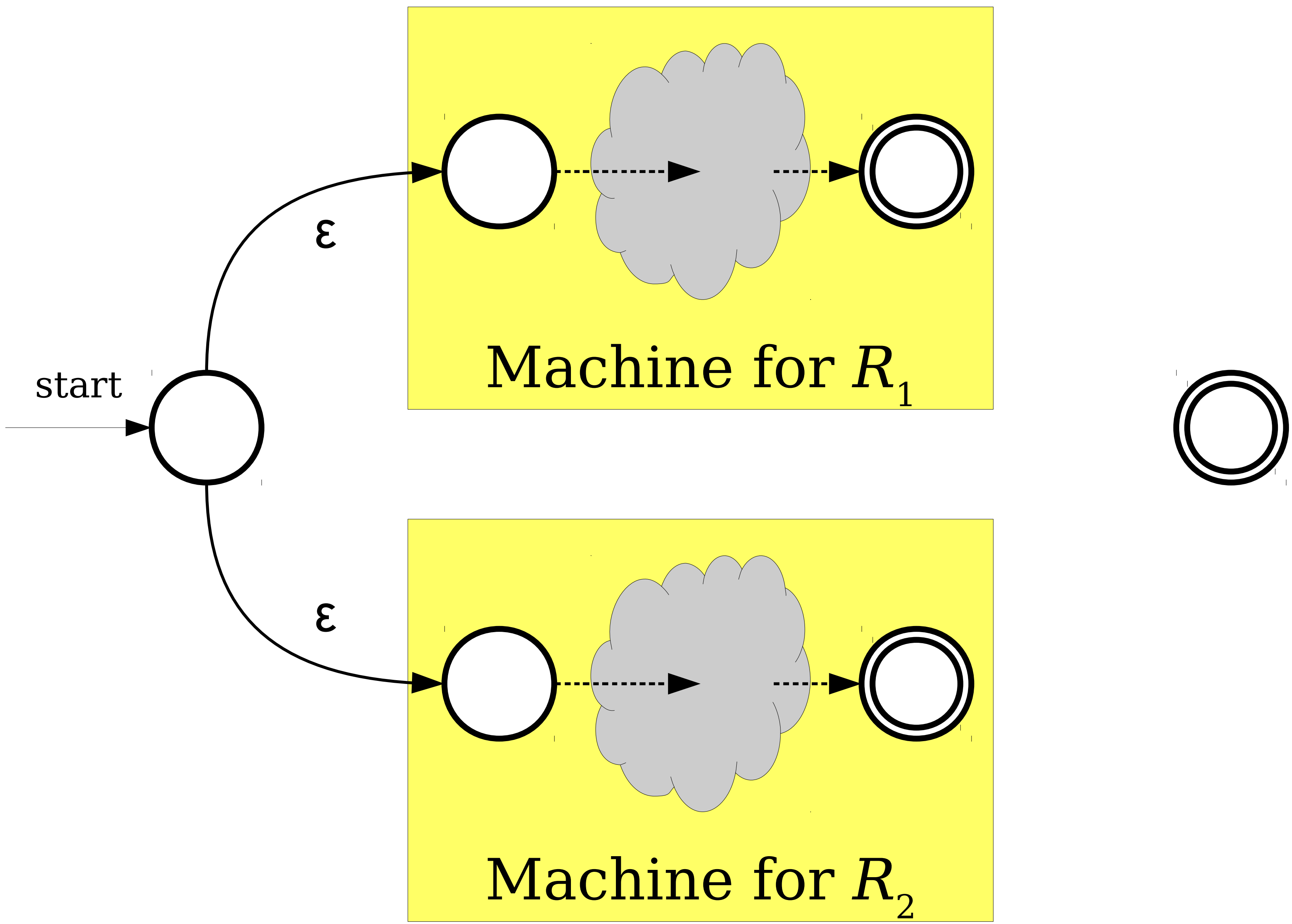
Construction for $R_1 \mid R_2$



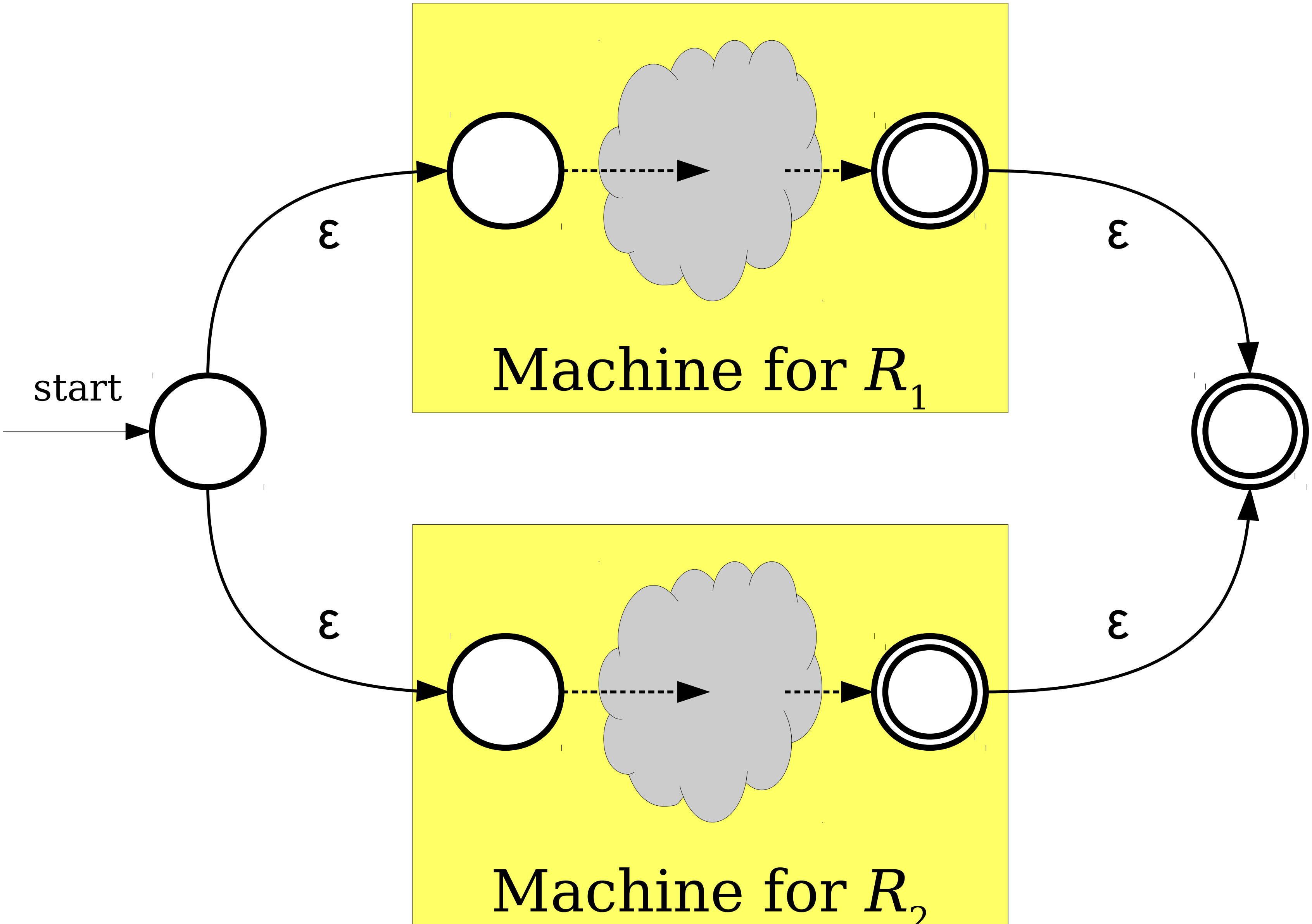
Construction for $R_1 \mid R_2$



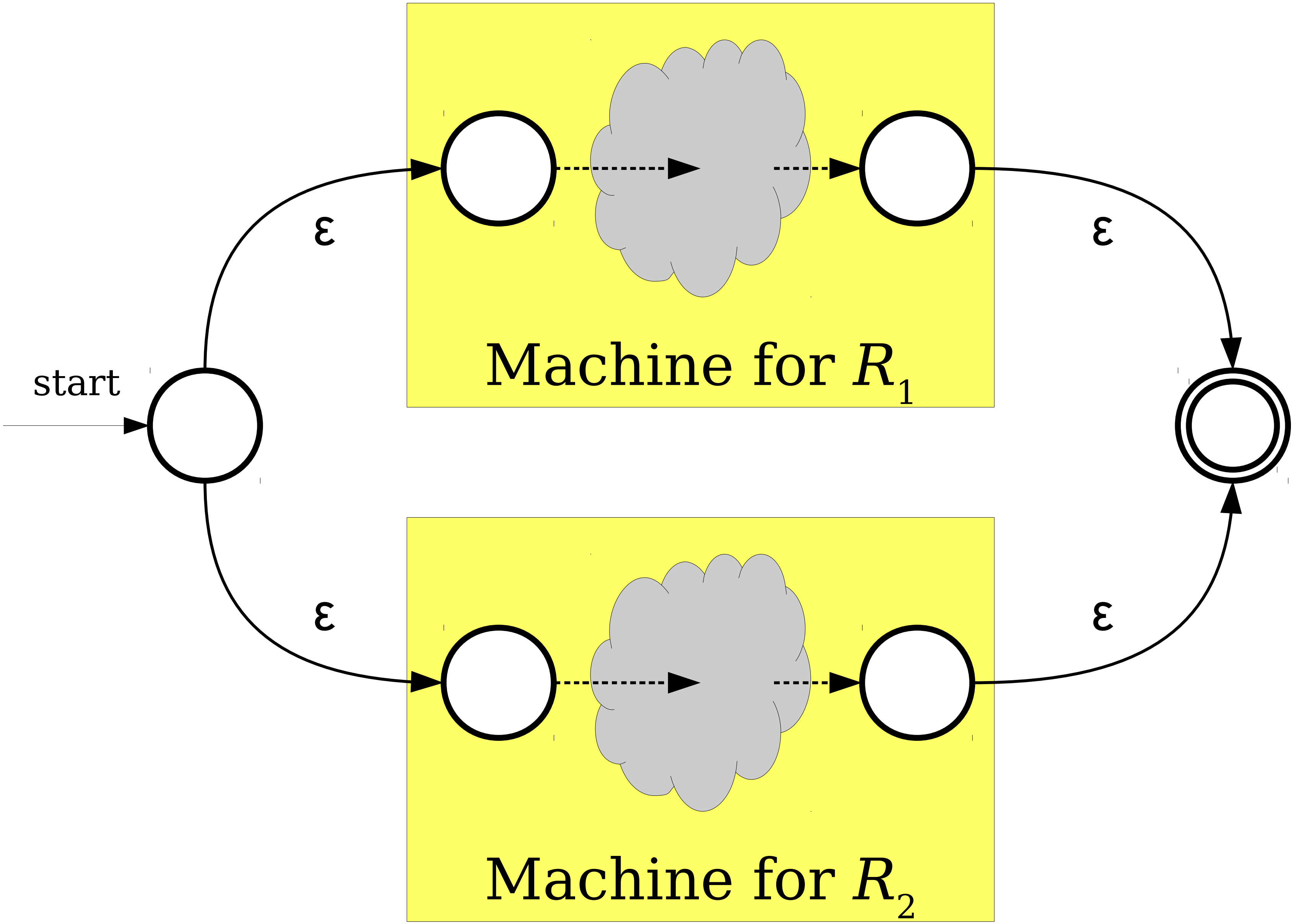
Construction for $R_1 \mid R_2$



Construction for $R_1 \mid R_2$

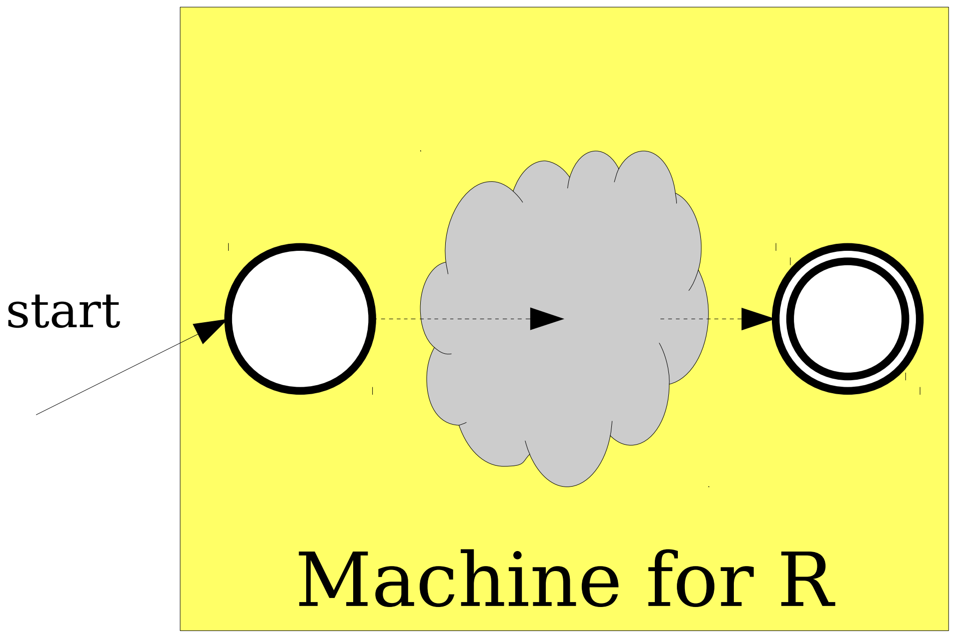


Construction for $R_1 \mid R_2$

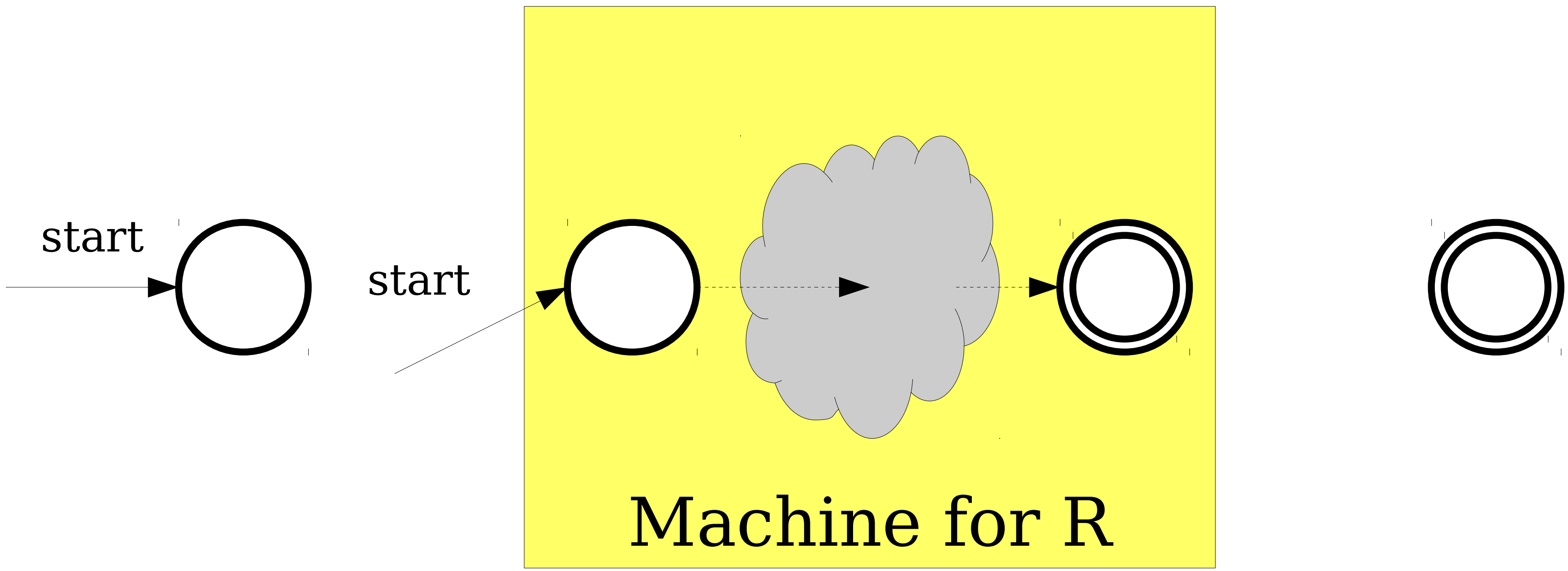


Construction for $R*$

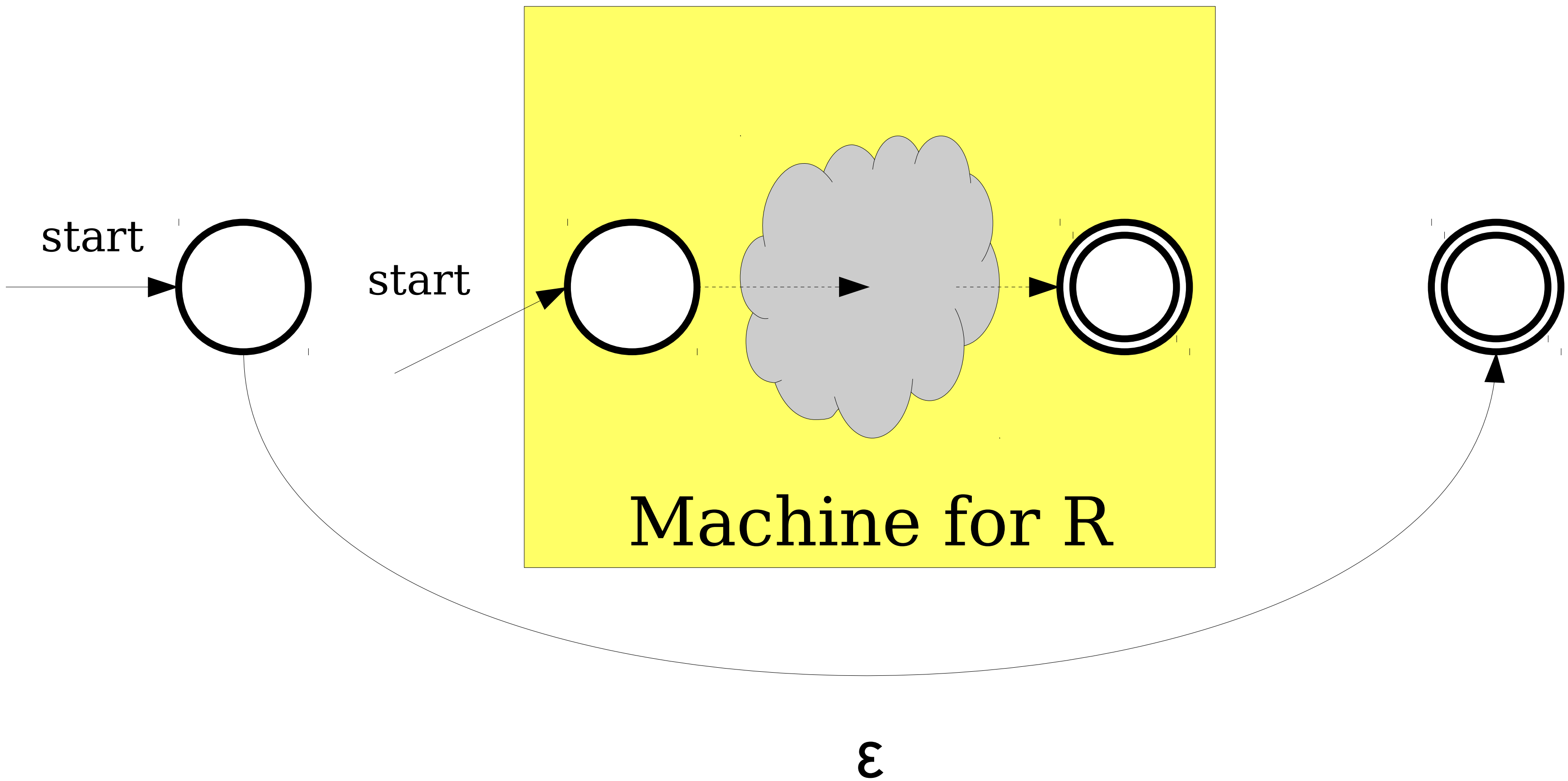
Construction for R^*



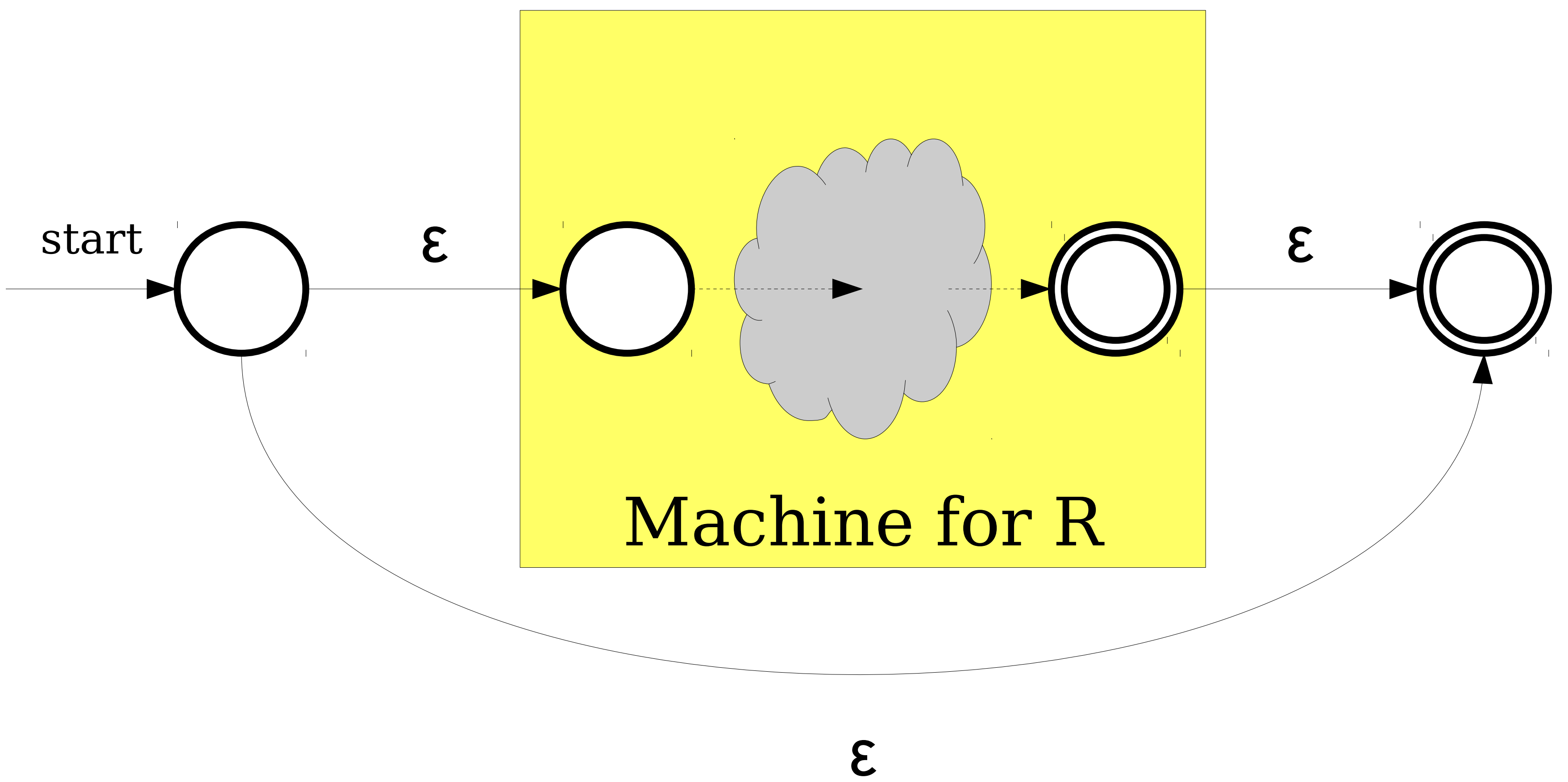
Construction for R^*



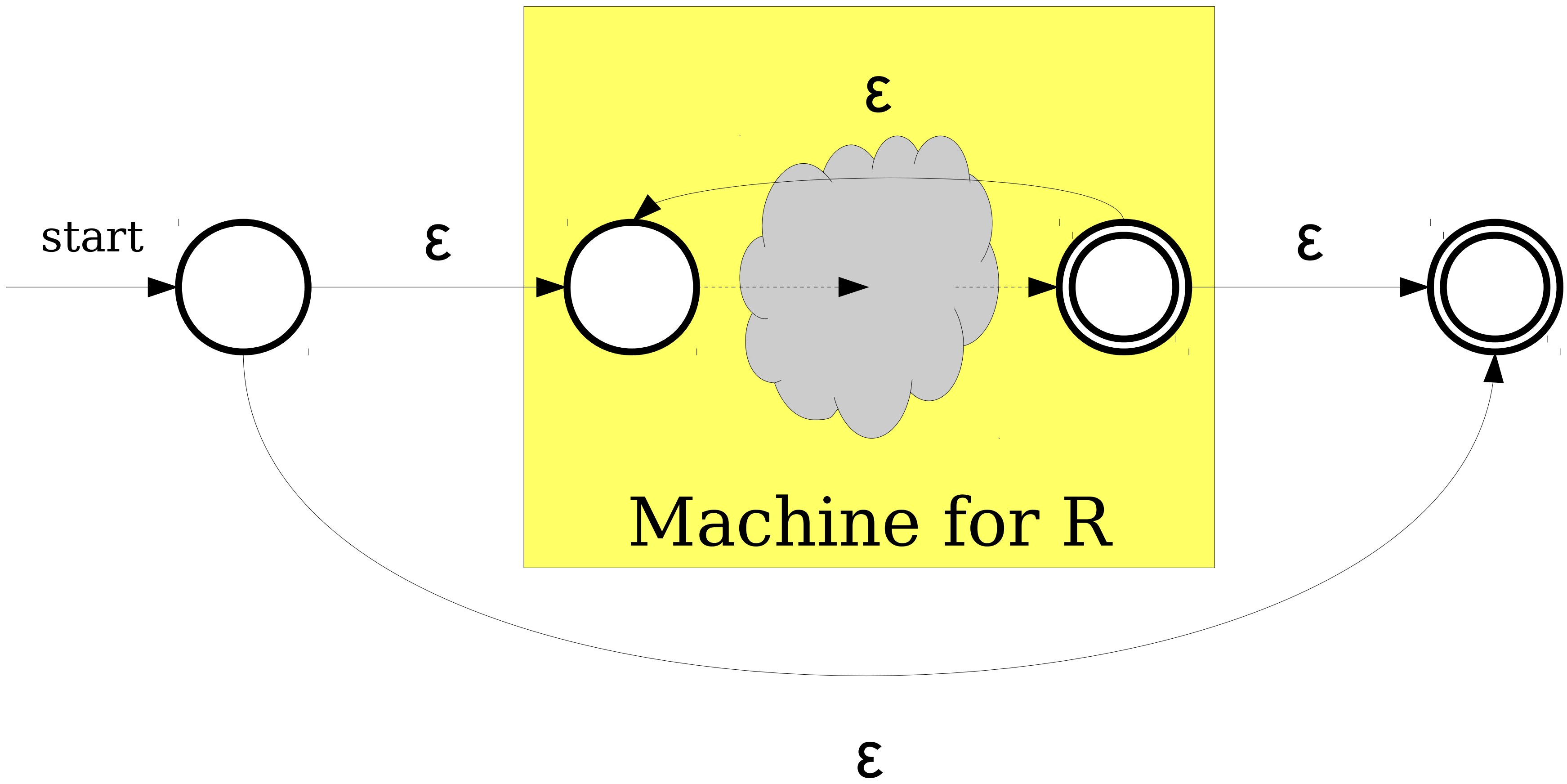
Construction for R*



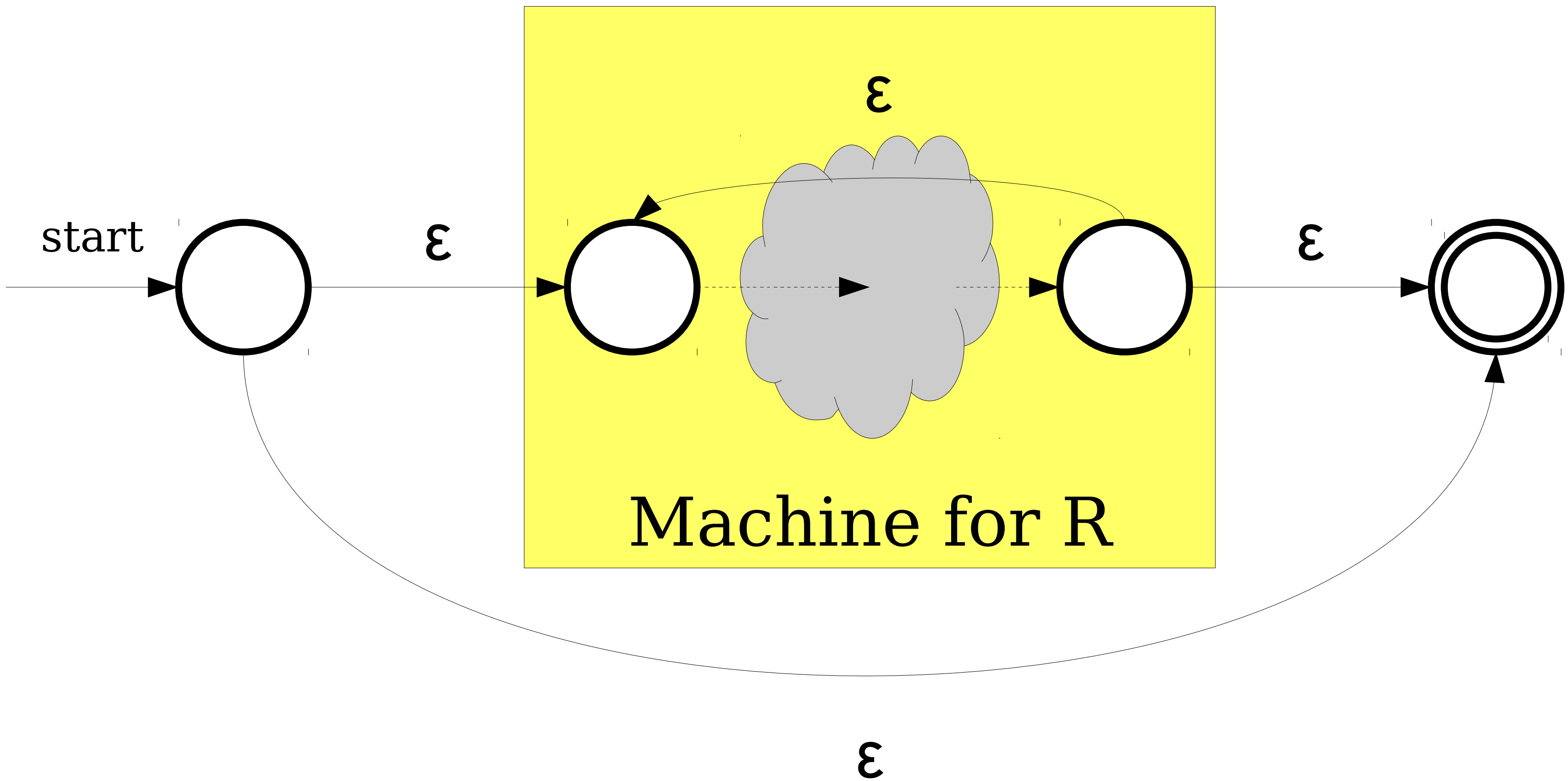
Construction for R*



Construction for *R**



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.
- Runs extremely quickly!

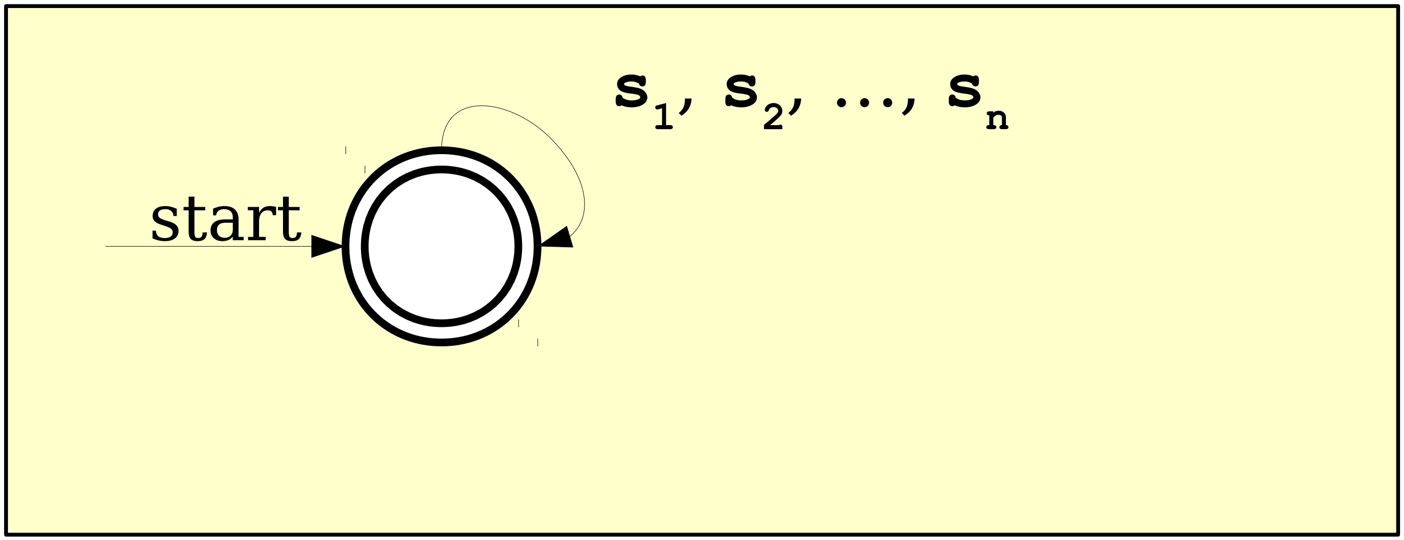
The Power of Regular Expressions

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

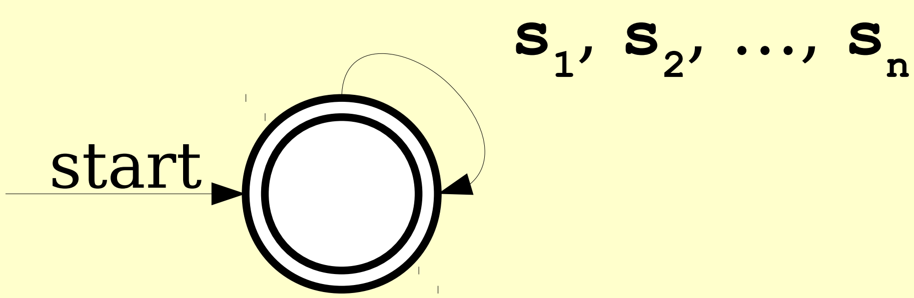
This is not obvious!

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

From NFAs to Regular Expressions

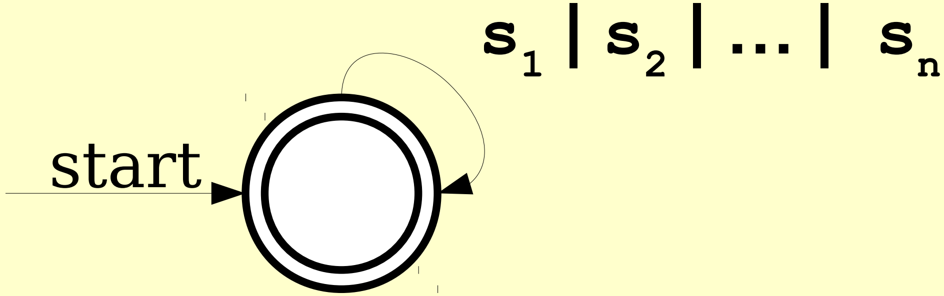


From NFAs to Regular Expressions



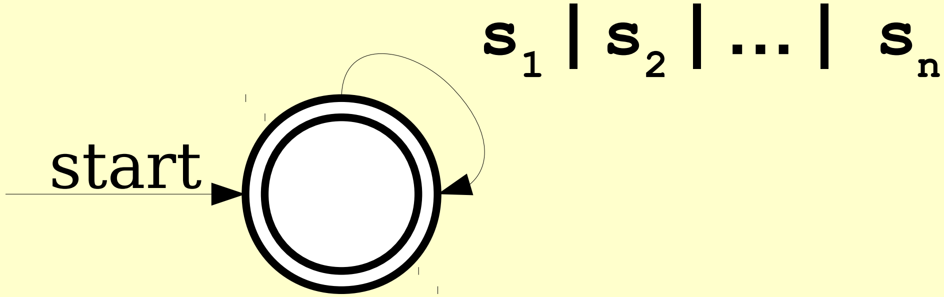
Regular expression: $(s_1 \mid s_2 \mid \ldots \mid s_n)^*$

From NFAs to Regular Expressions



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

From NFAs to Regular Expressions

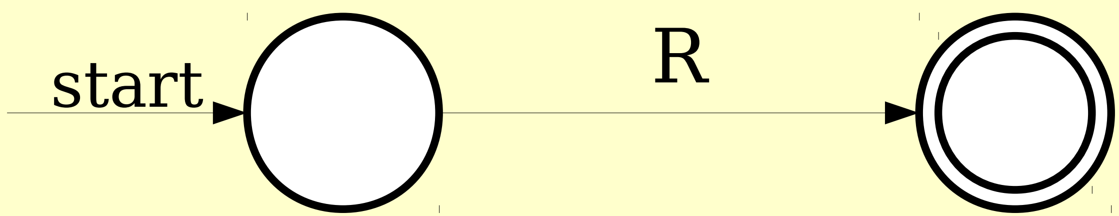


Regular expression: $(s_1 \mid s_2 \mid \ldots \mid s_n)^*$

Key idea: Label transitions with arbitrary regular expressions.

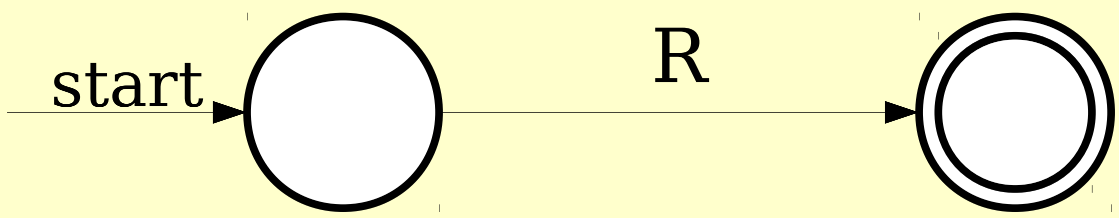
From NFAs to Regular Expressions

From NFAs to Regular Expressions



Regular expression: **R**

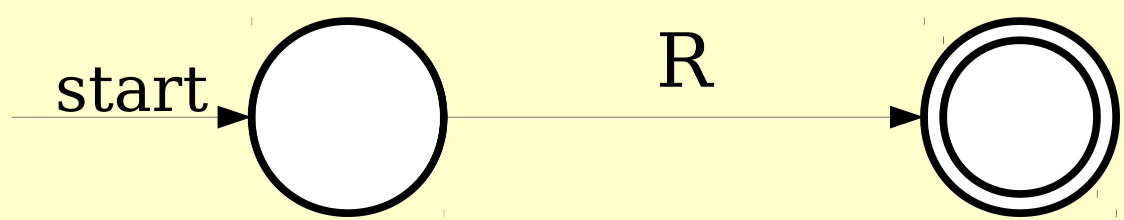
From NFAs to Regular Expressions



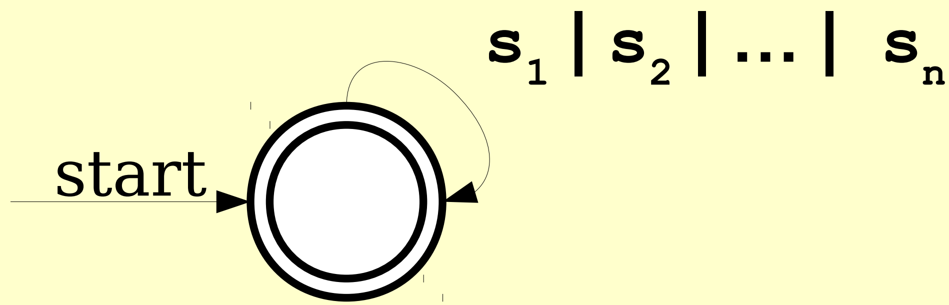
Regular expression: **R**

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

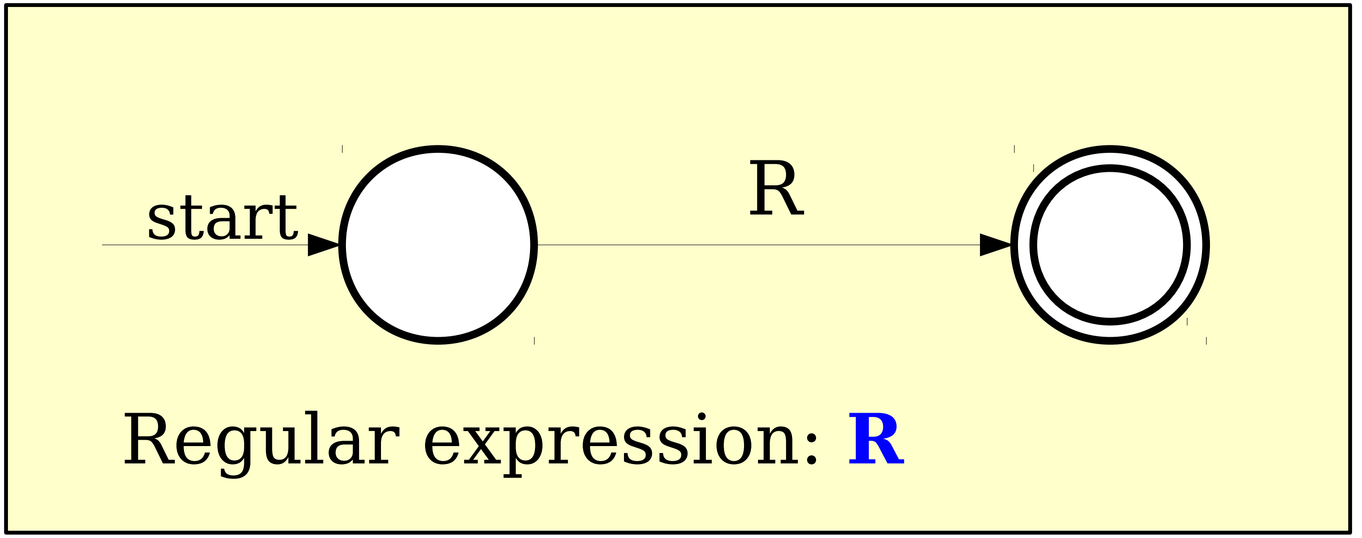
From NFAs to Regular Expressions

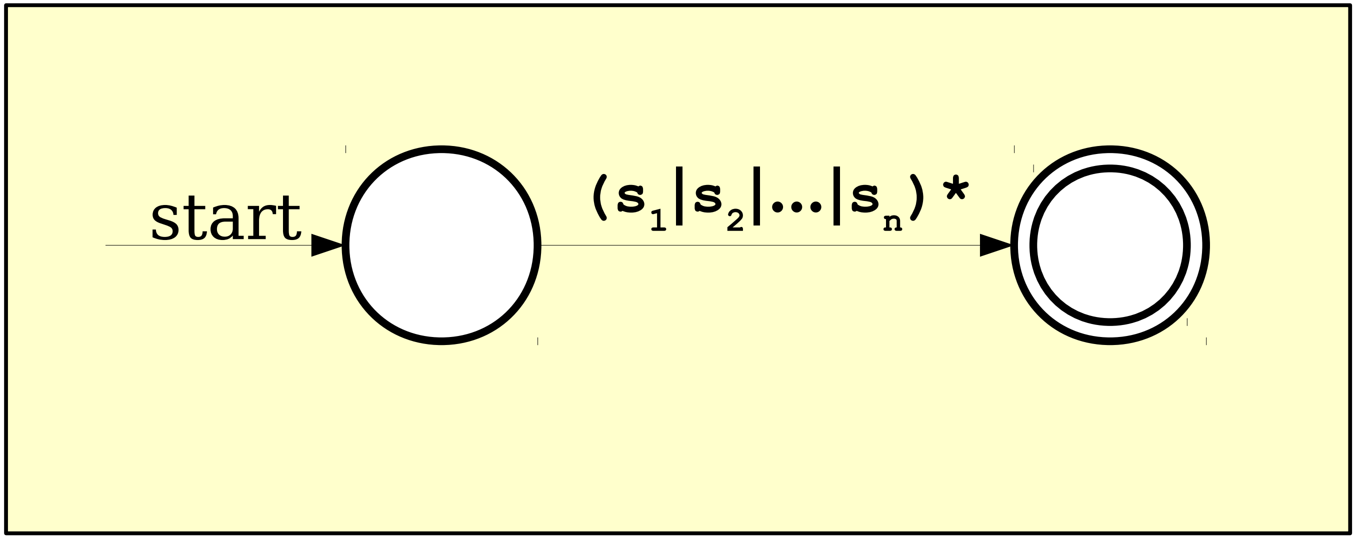


Regular expression: **R**

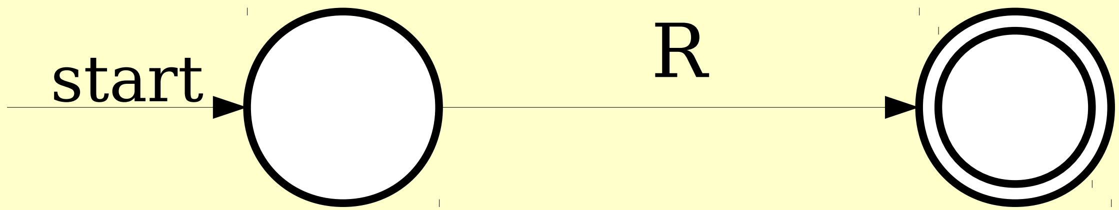


From NFAs to Regular Expressions

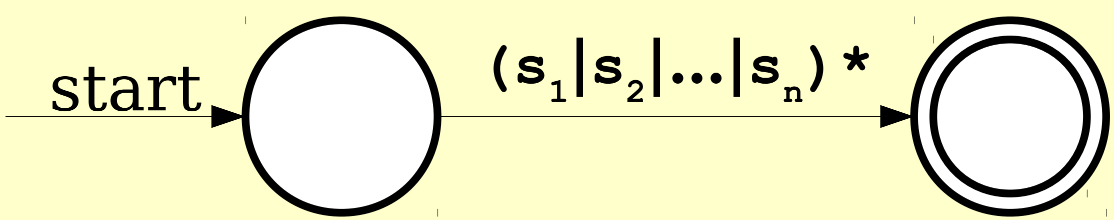




From NFAs to Regular Expressions

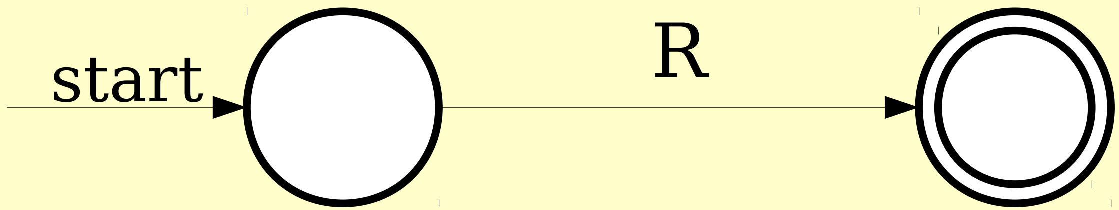


Regular expression: **R**



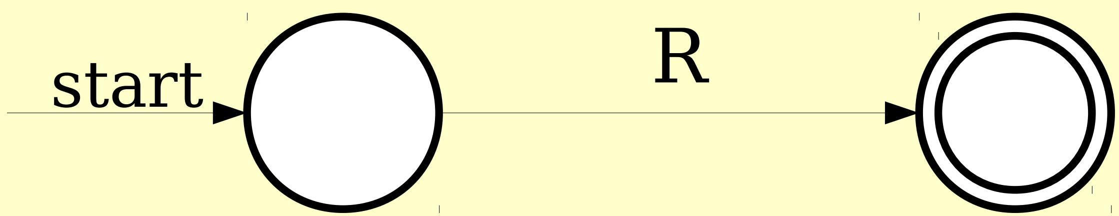
Regular expression: $(s_1|s_2| \ldots |s_n)*$

From NFAs to Regular Expressions

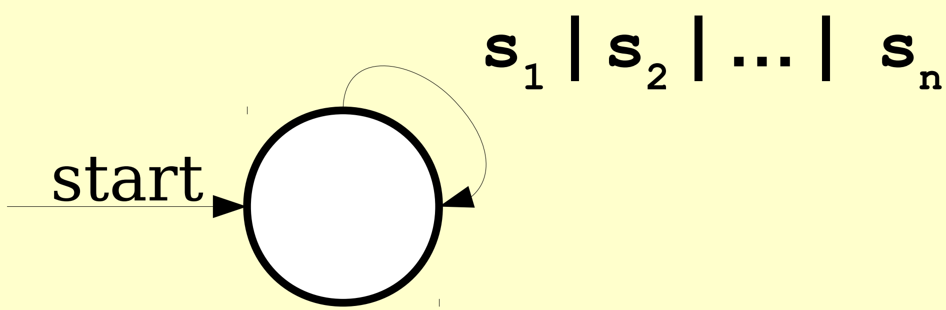


Regular expression: **R**

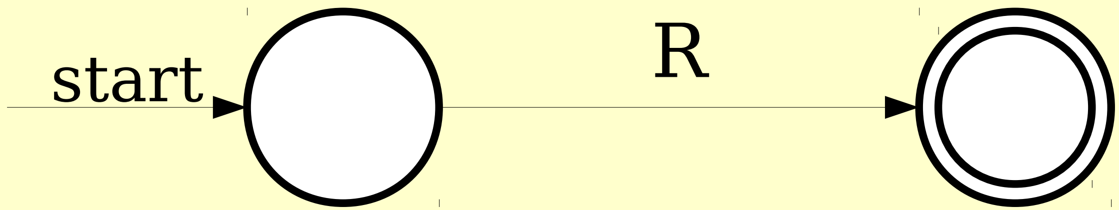
From NFAs to Regular Expressions



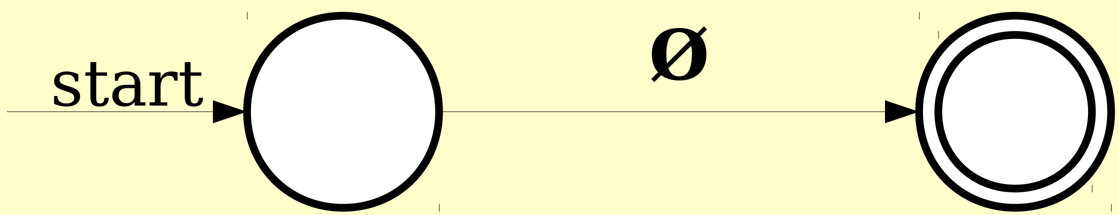
Regular expression: **R**



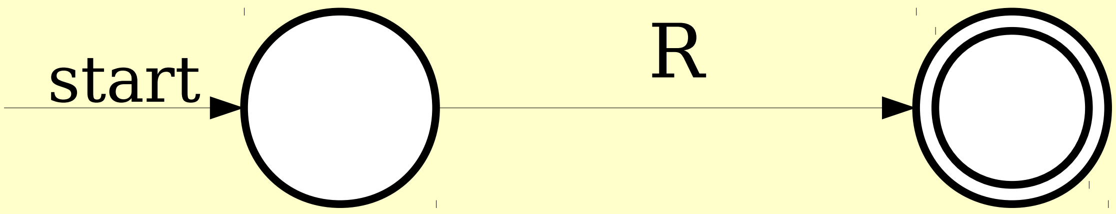
From NFAs to Regular Expressions



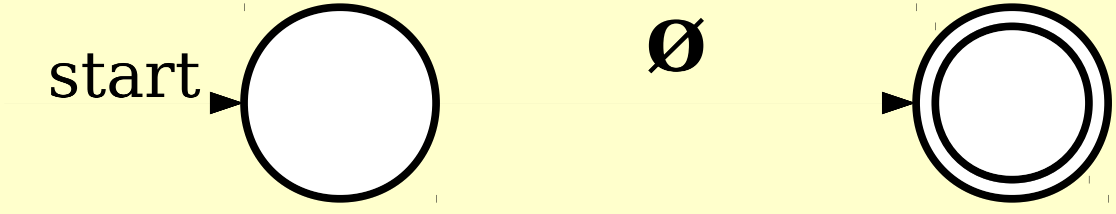
Regular expression: **R**



From NFAs to Regular Expressions

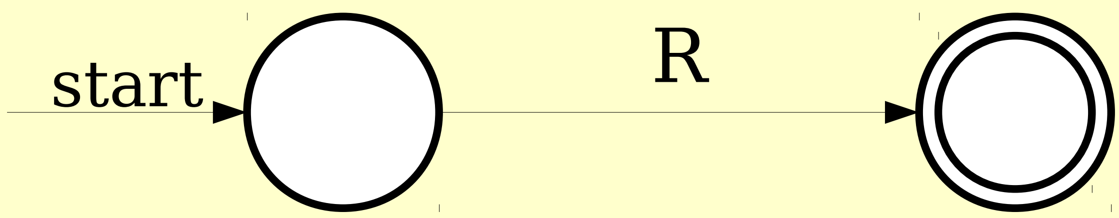


Regular expression: **R**



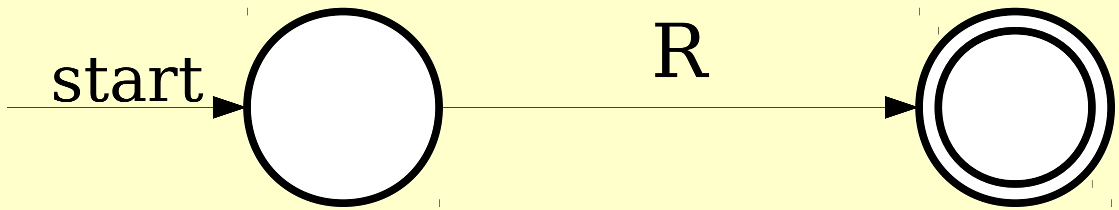
Regular expression: **Ø**

From NFAs to Regular Expressions

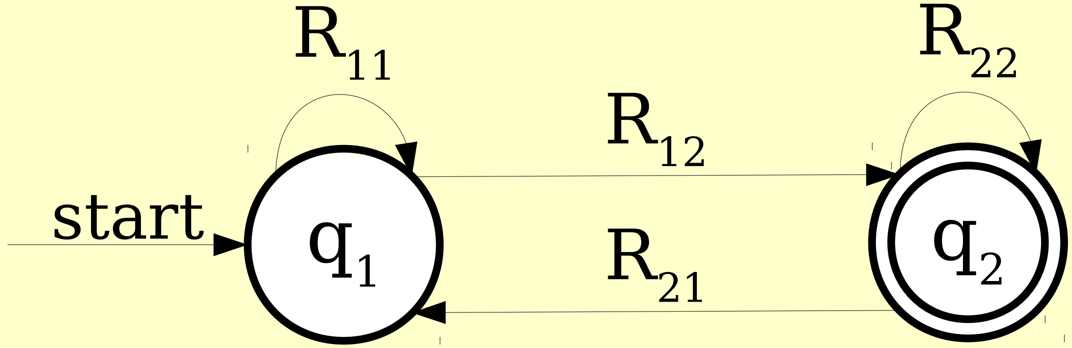


Regular expression: **R**

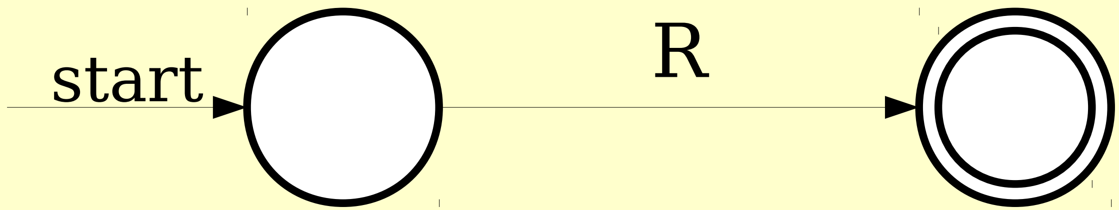
From NFAs to Regular Expressions



Regular expression: **R**

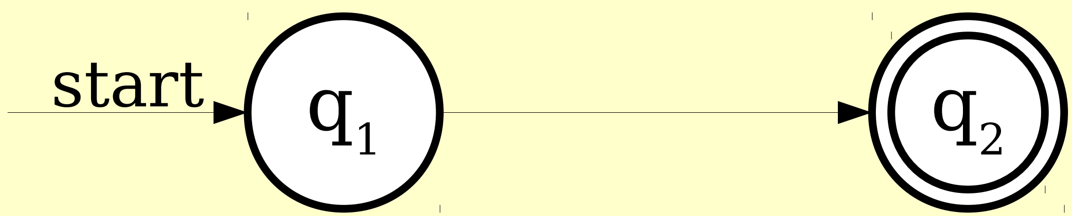


From NFAs to Regular Expressions

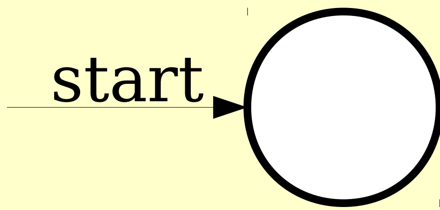


Regular expression: **R**

R_{11}* R_{12} (R_{22} | $R_{21}R_{11}$*R_{12})*

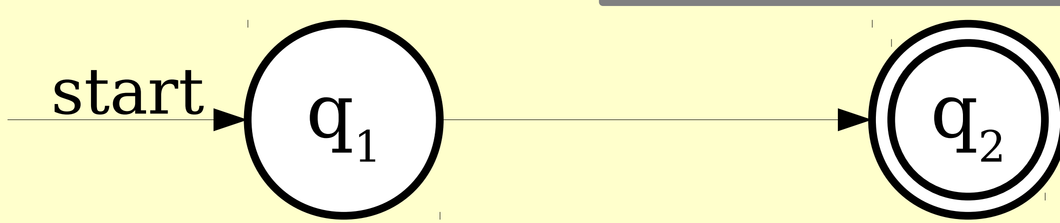


From NFAs to Regular Expressions

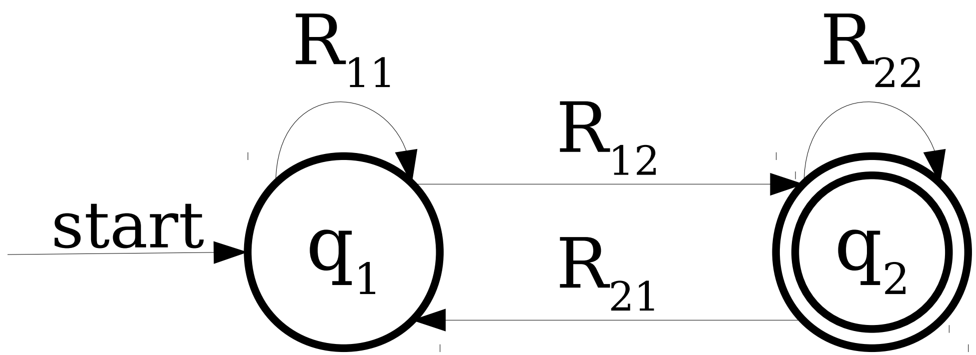


Regular expressi

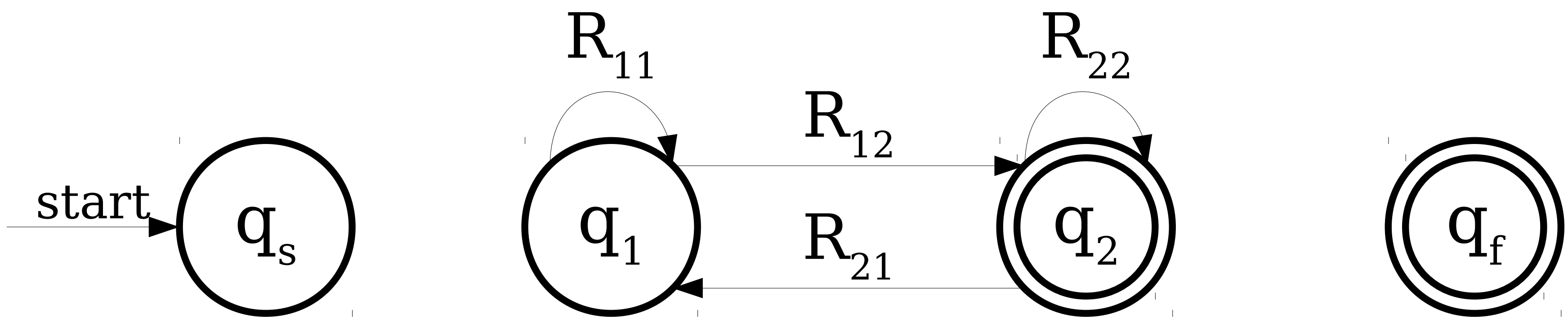
R_{11}* R_{12} (R



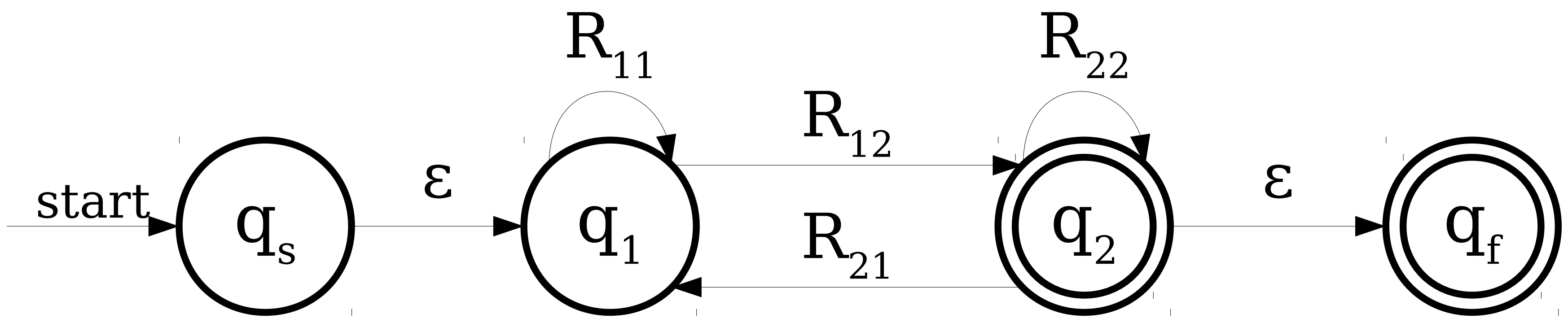
From NFAs to Regular Expressions



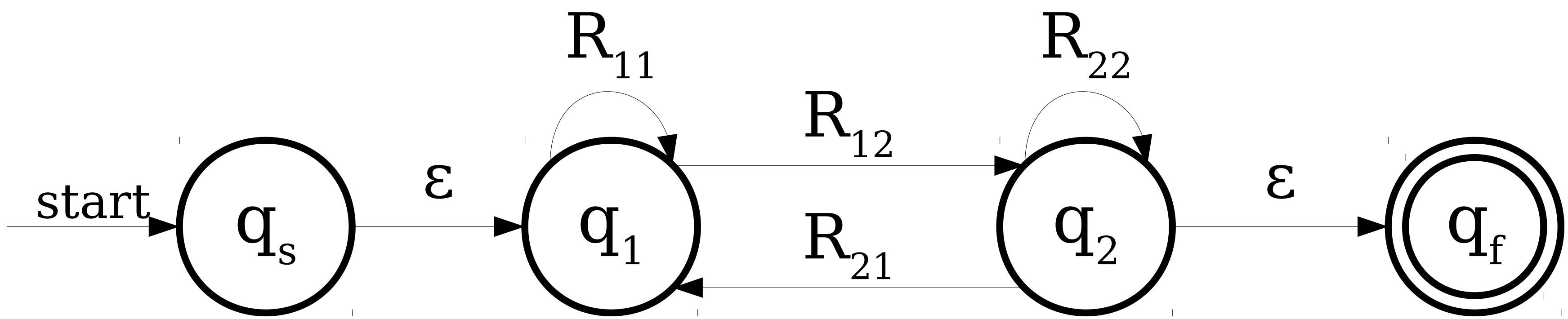
From NFAs to Regular Expressions



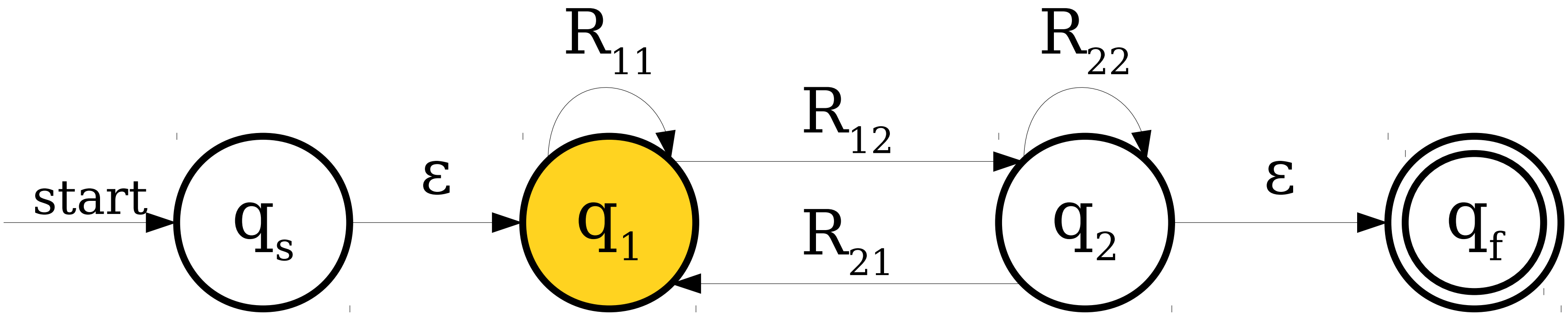
From NFAs to Regular Expressions



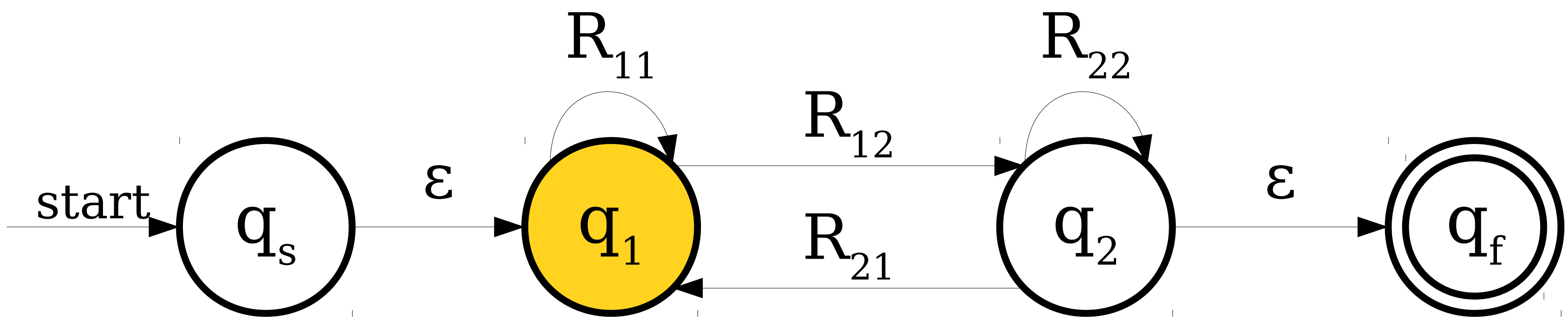
From NFAs to Regular Expressions



From NFAs to Regular Expressions

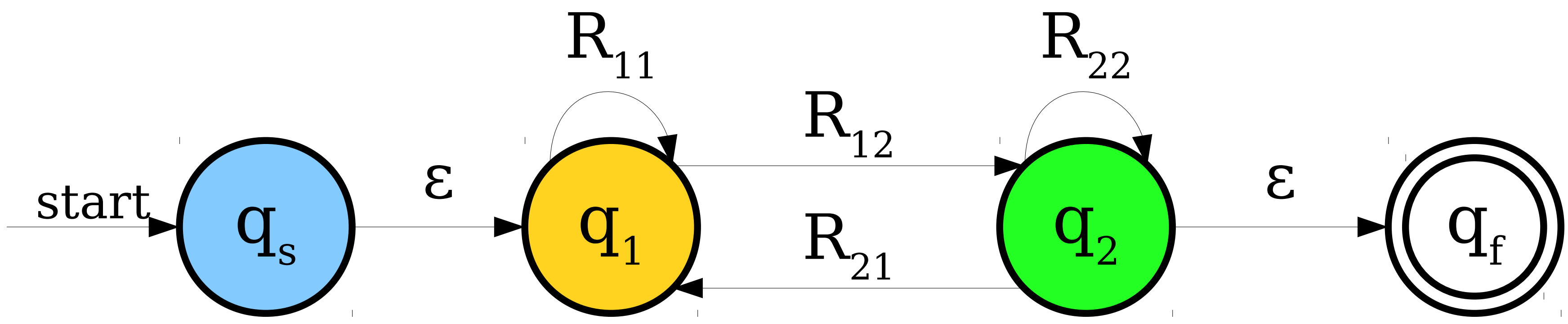


From NFAs to Regular Expressions

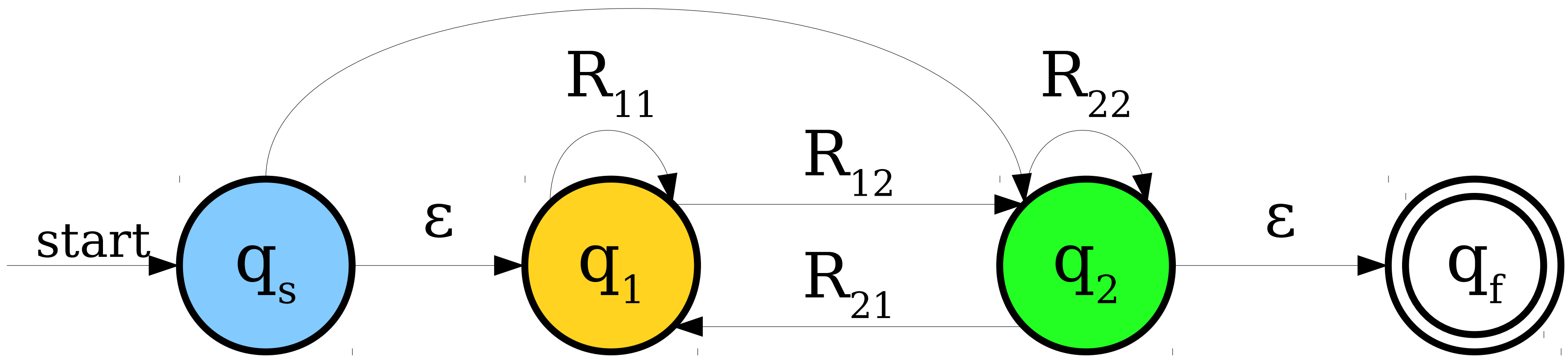


Could we eliminate this state from the NFA?

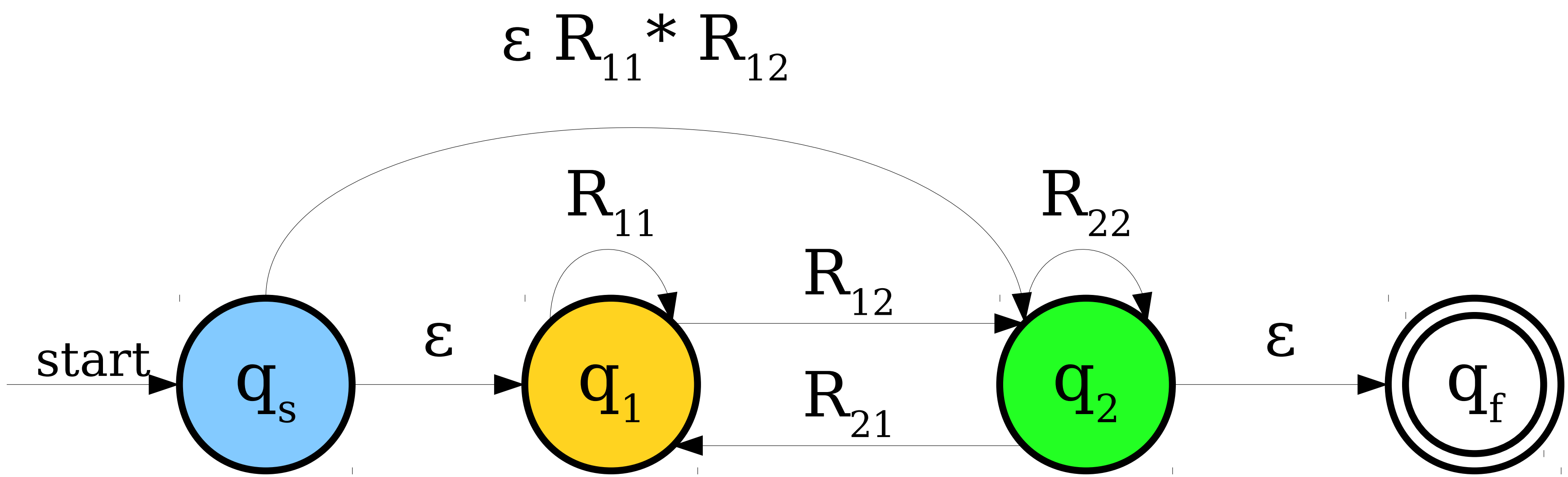
From NFAs to Regular Expressions



From NFAs to Regular Expressions

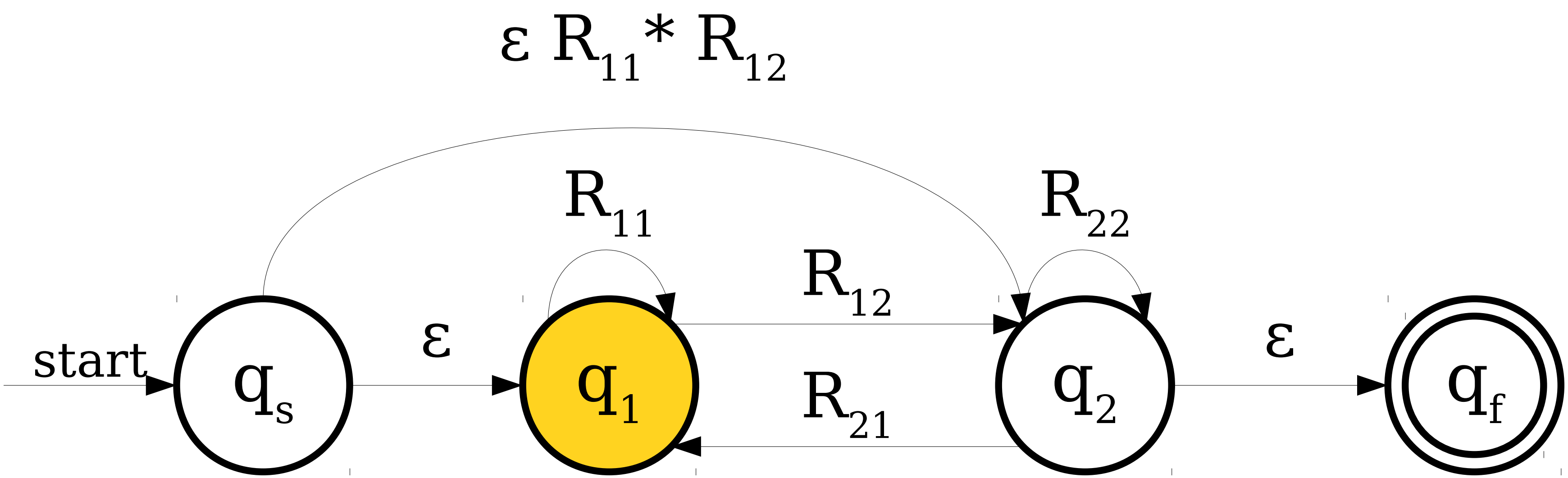


From NFAs to Regular Expressions

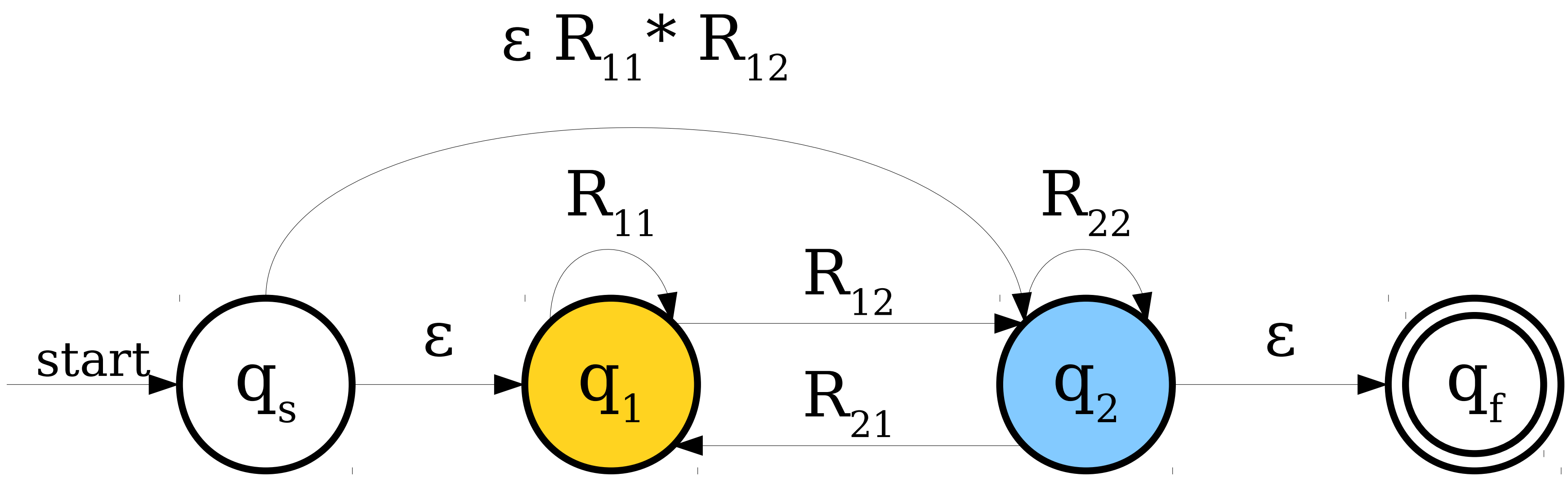


Note: We're using **concatenation** and **Kleene closure** in order to skip this state.

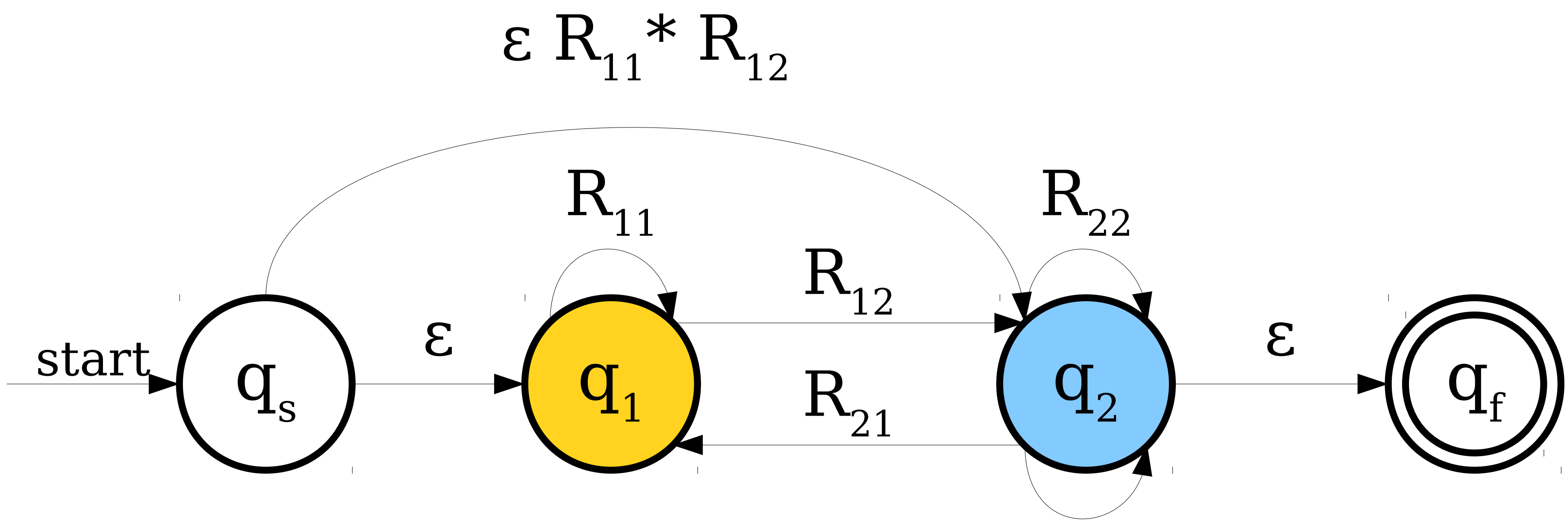
From NFAs to Regular Expressions



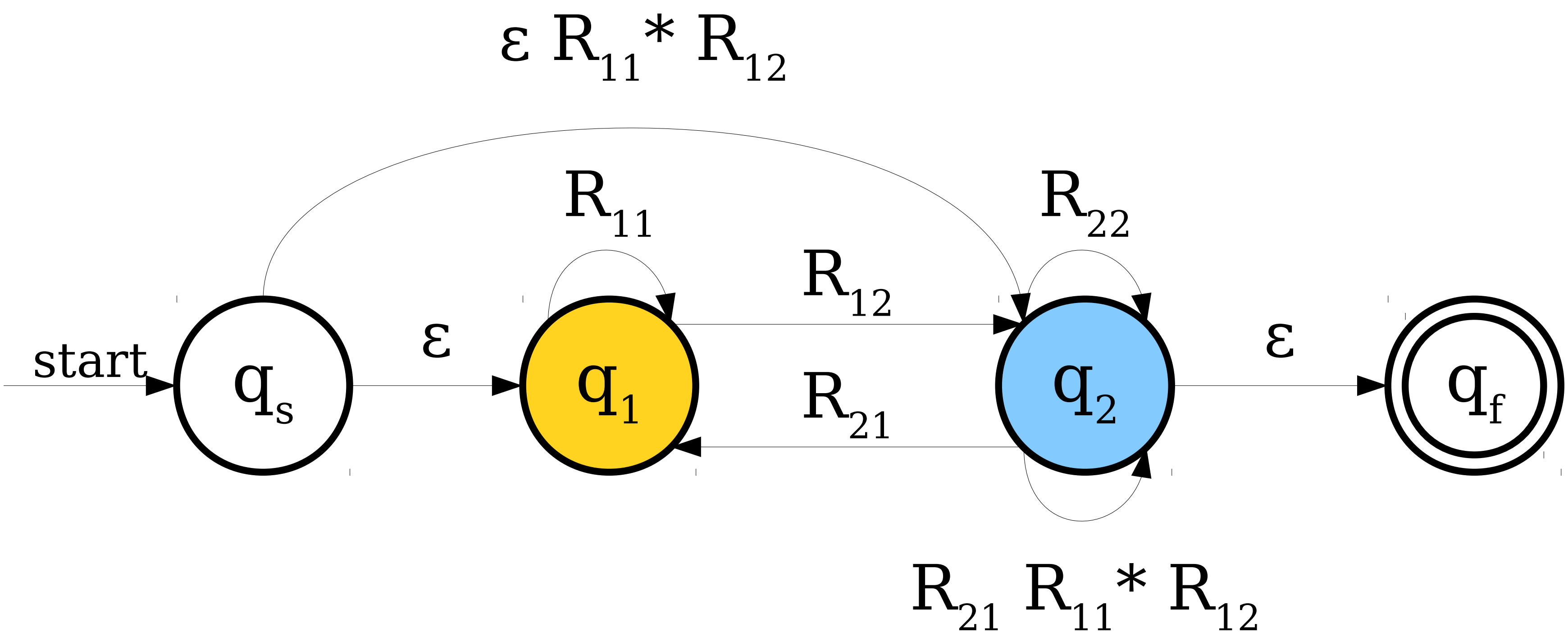
From NFAs to Regular Expressions



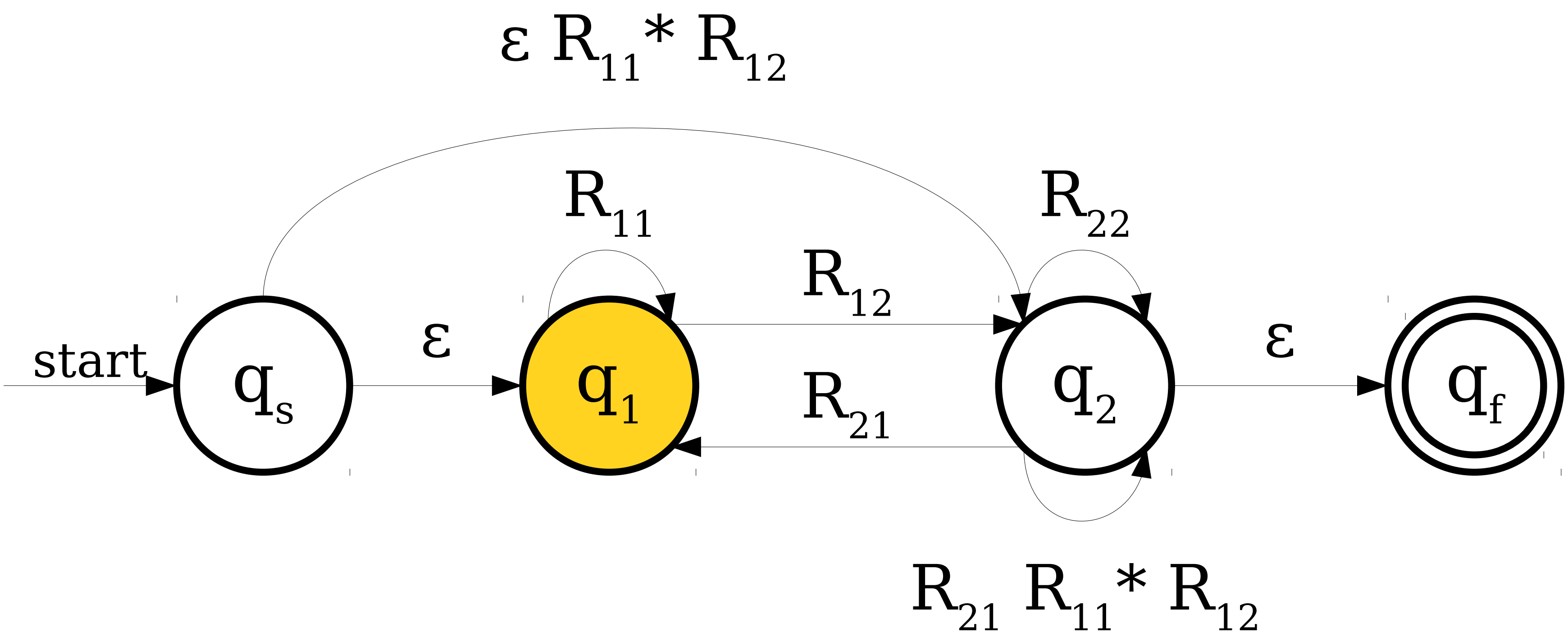
From NFAs to Regular Expressions



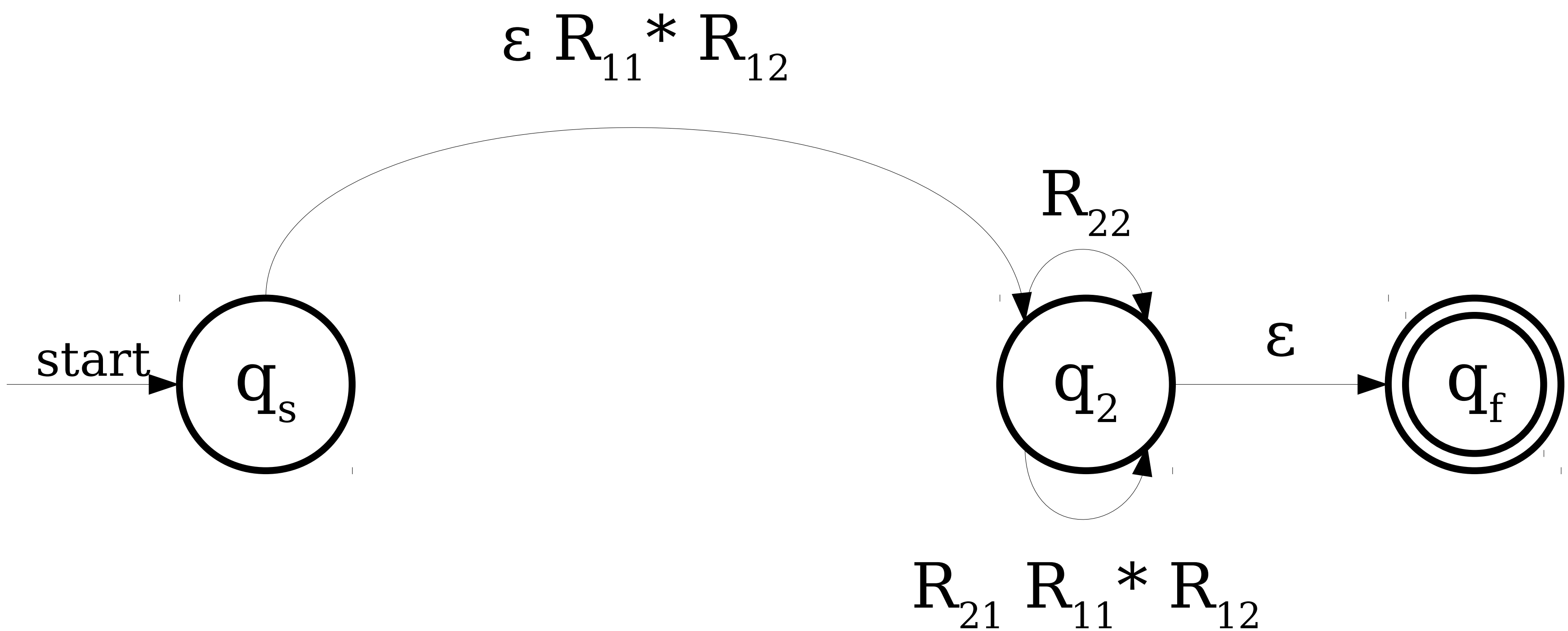
From NFAs to Regular Expressions



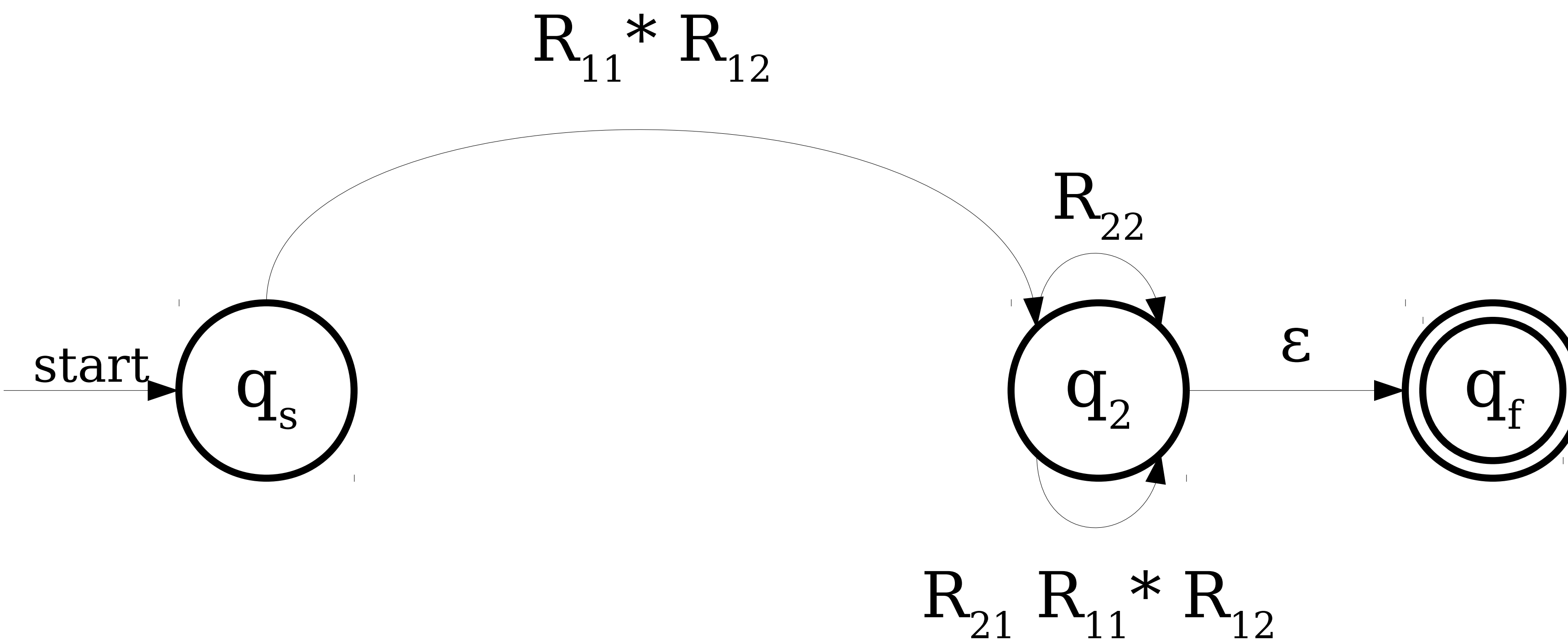
From NFAs to Regular Expressions



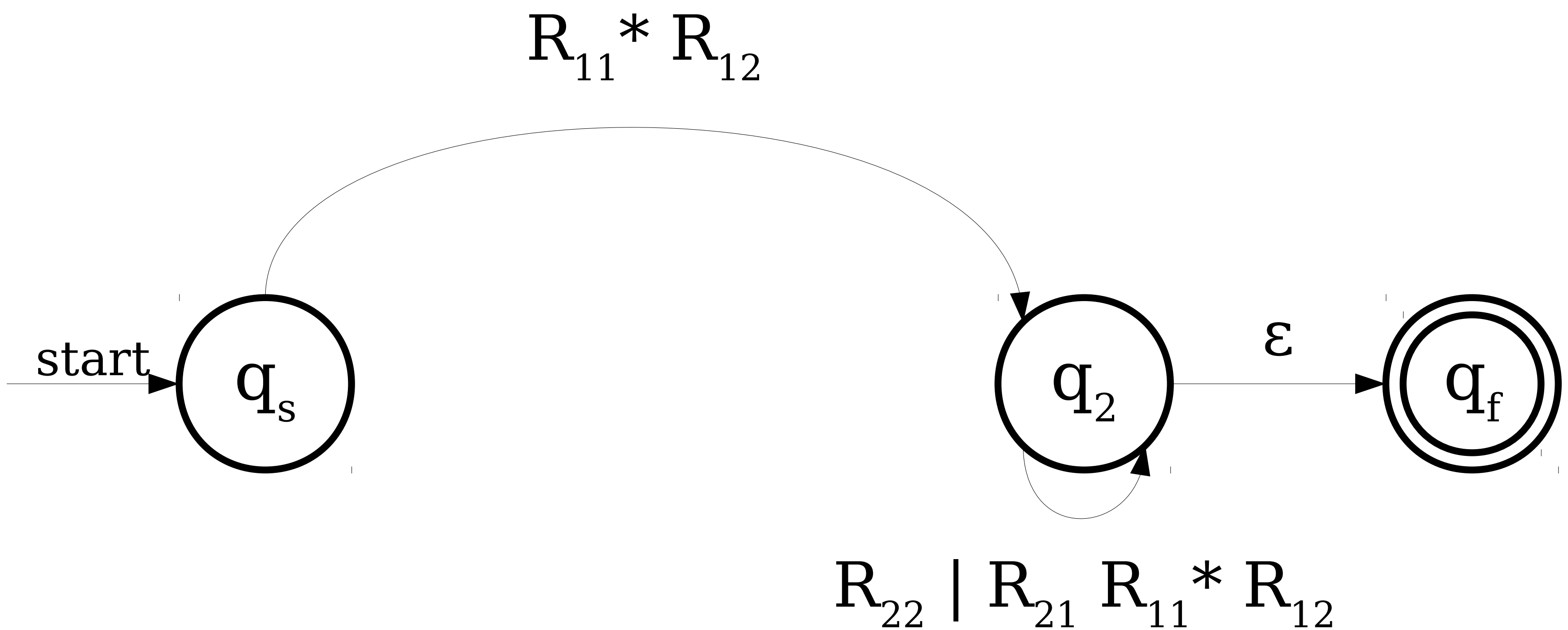
From NFAs to Regular Expressions



From NFAs to Regular Expressions

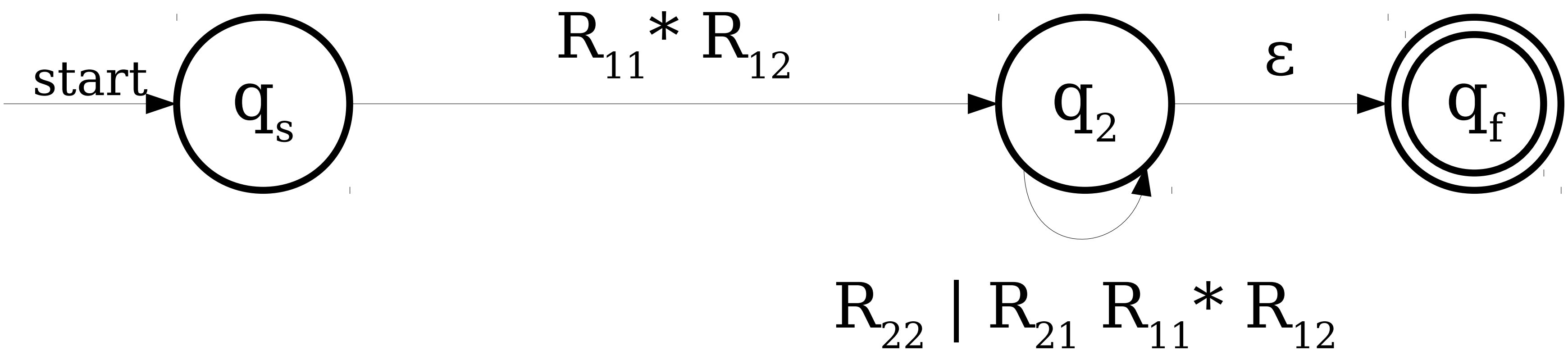


From NFAs to Regular Expressions

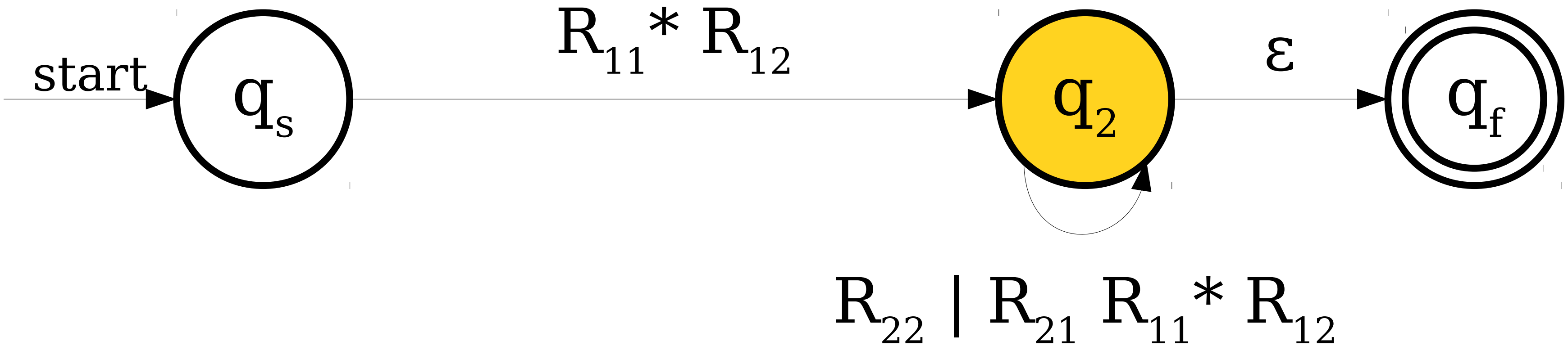


Note: We're using union to combine these transitions together.

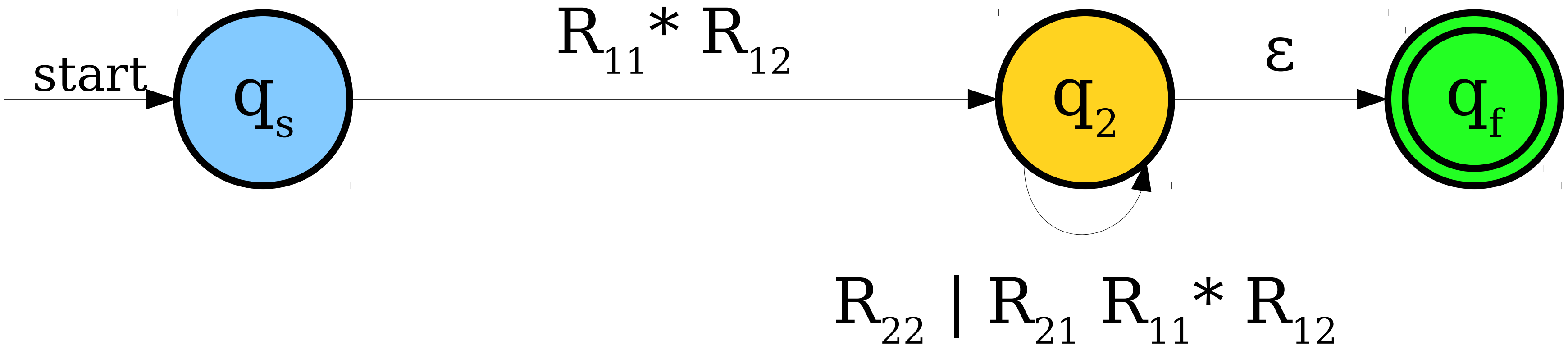
From NFAs to Regular Expressions



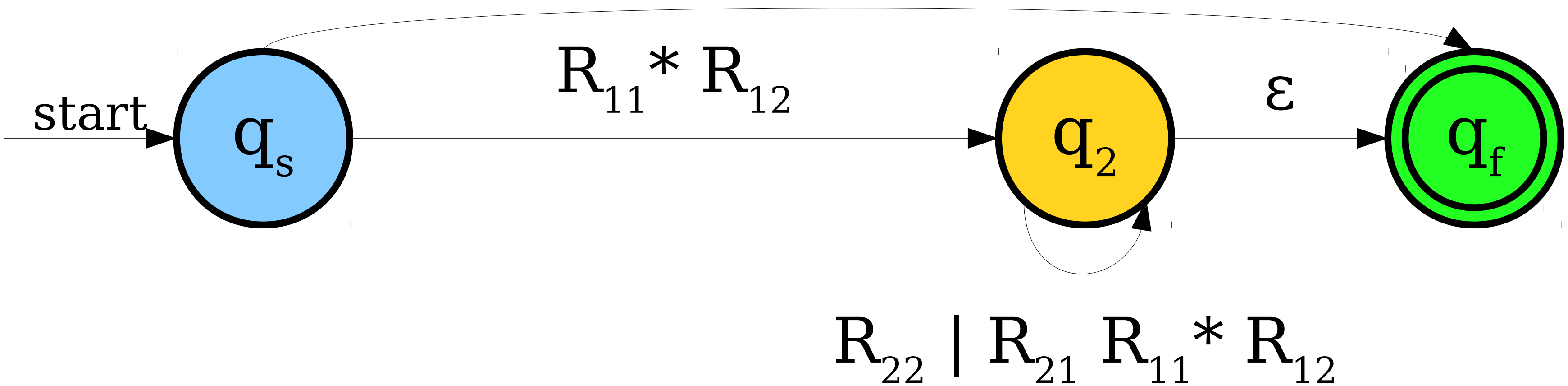
From NFAs to Regular Expressions



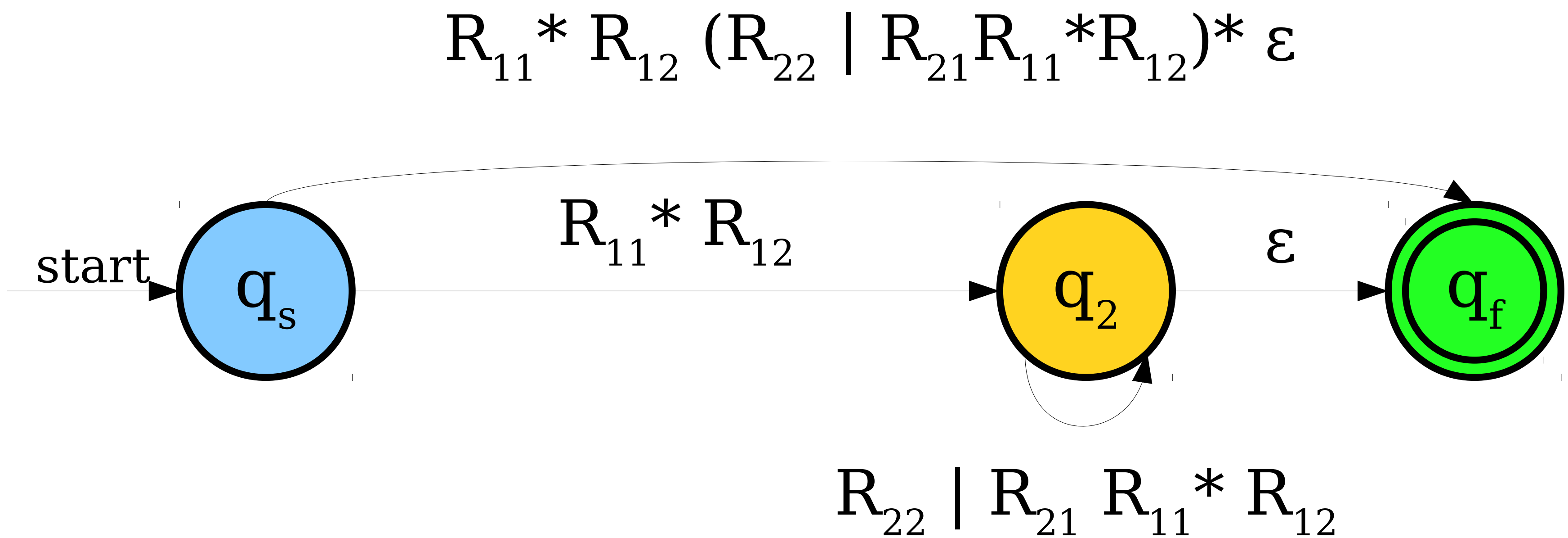
From NFAs to Regular Expressions



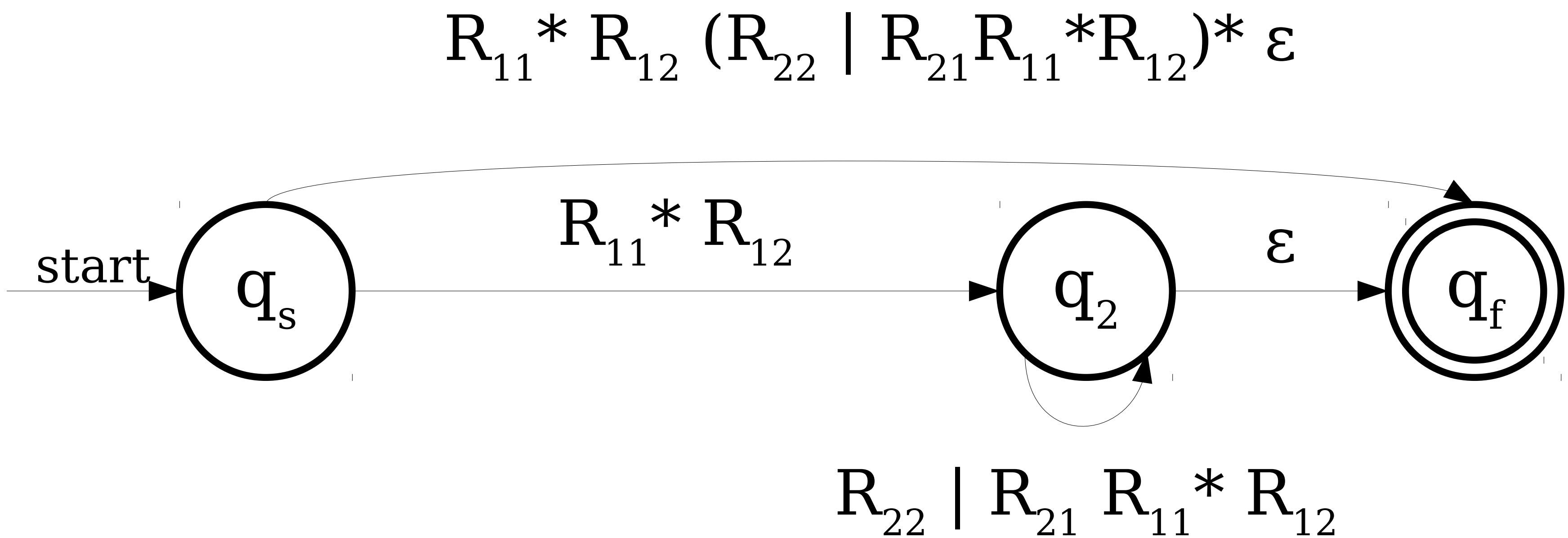
From NFAs to Regular Expressions



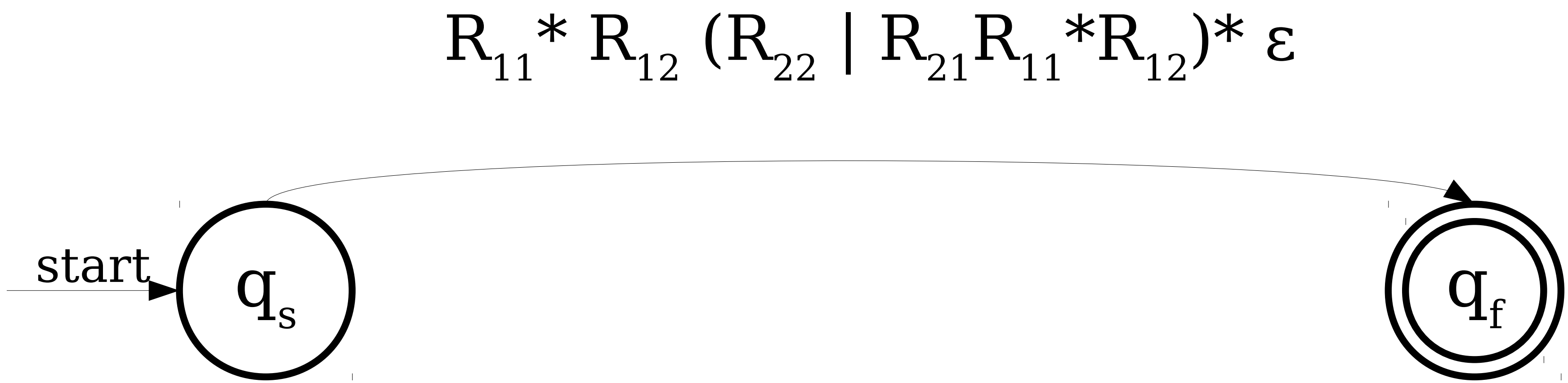
From NFAs to Regular Expressions



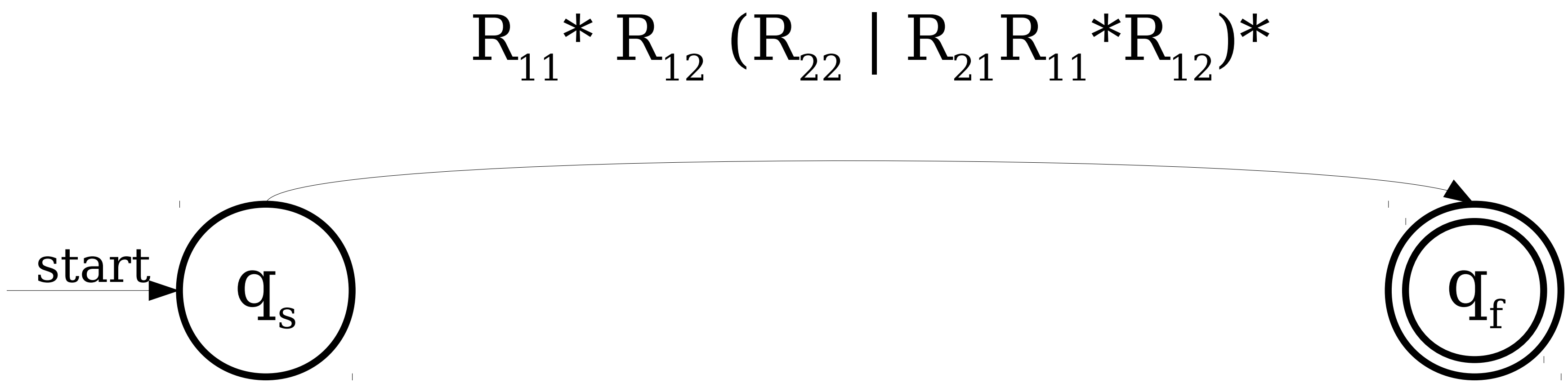
From NFAs to Regular Expressions



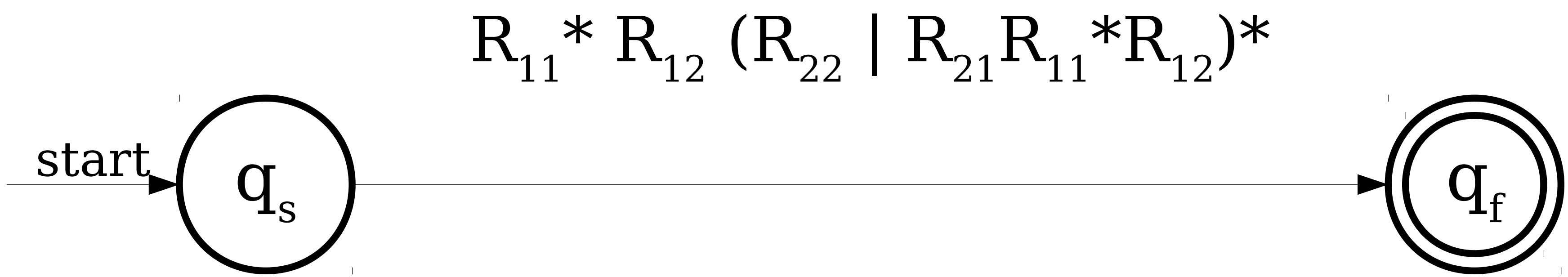
From NFAs to Regular Expressions



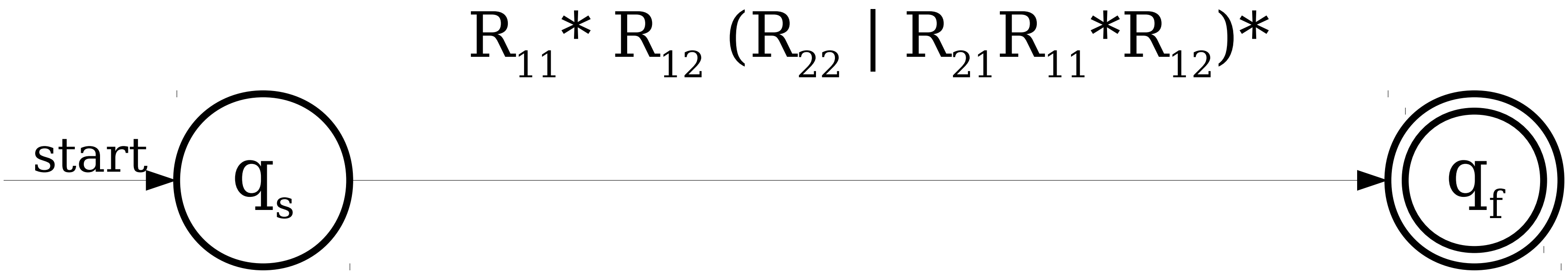
From NFAs to Regular Expressions

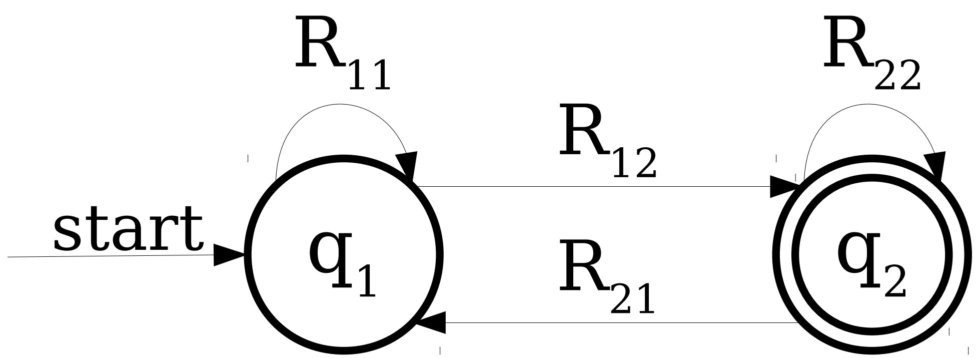


From NFAs to Regular Expressions



From NFAs to Regular Expressions

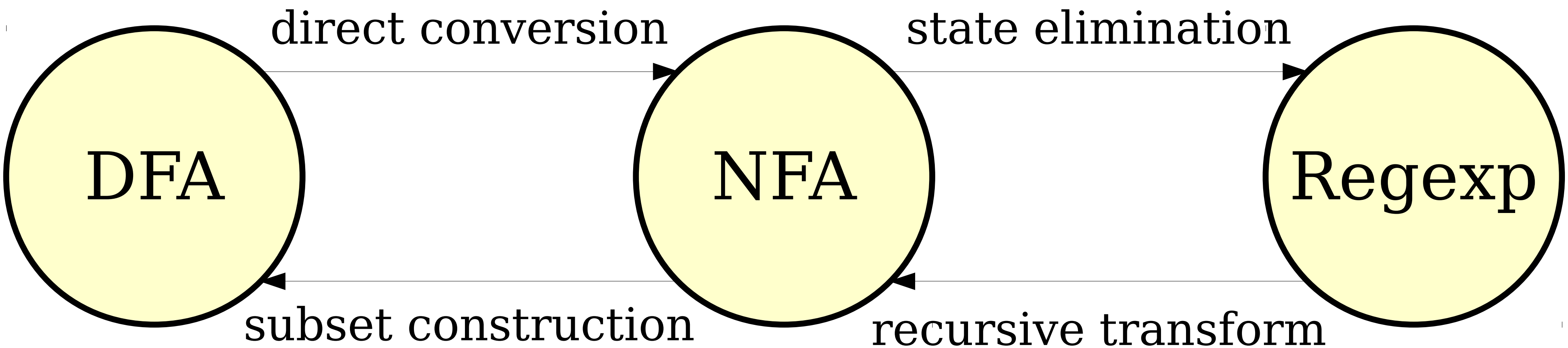




The Construction at a Glance

- 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 $\mathscr{L}(R) = L$.

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.