Regular Expressions

Announcements

- Problem Set 5 due this Friday at 2:15.
  - Stop by OH with questions!
  - Email **cs103@cs.stanford.edu** with questions!
- Midterm graded; will be returned after lecture.
  - Solutions distributed in hardcopy for the time being; will be uploaded to the website soon.
  - Unclaimed midterms will be available outside of Gates 178.

Describing Regular Languages

- So far, we have the following tools for describing regular languages:
  - DFAs, as transition graphs, transition tables, or five-tuples.
  - NFAs, in the same way.
  - Set-builder notation: { w | w has some property }, though this may not be a regular language.
- Is there an easier way to encode regular languages?

Regular Expressions

- **Regular expressions** are a family of descriptions that can be used to capture the regular languages.
- Often provide a compact and human-readable description of the language.
- Used as the basis for numerous software systems (Perl, **`flex`**, **`grep`**, etc.)

Regular Expressions, Formally

- The **regular expressions over Σ** are
  - **ε**, which represents the language {ε},
  - **Ø**, which represents the empty language Ø, and
  - **a** for any a ∈ Σ, which represents the language {a}
  - If R_1 and R_2 are regular expressions, $\mathbf{R_1R_2}$ is a regular expression represents the **concatenation** of the languages of R_1 and R_2.
  - If R_1 and R_2 are regular expressions, $\mathbf{R_1 \mid R_2}$ is a regular expression representing the **union** of R_1 and R_2.
  - If R is a regular expression, **R*** is a regular expression for the **Kleene closure** of R.
  - If R is a regular expression, **(R)** is a regular expression with the same meaning as R.

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

Operator Precedence

- Regular expression operator precedence is
  - Parentheses, then
  - Kleene star, then
  - Concatenation, then
  - Union
- For example, 1|01* is parsed as 1 | (0(1*))

Regular Expressions are Awesome

- Let $\Sigma = \{0, 1\}$
- Let L = { w | w contains 00 as a substring }

Regular Expressions are Awesome

- Let Σ = {0, 1}
- Let L = { w | w contains 00 as a substring }

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

Regular Expressions are Awesome

- Let Σ = {0, 1}
- Let L = { w | w contains 00 as a substring }

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

Regular Expressions are Awesome

- Let Σ = {0, 1}
- Let L = { w | w contains 00 as a substring }

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

11011100101
0000
11111011110011111

Regular Expressions are Awesome

- Let Σ = {0, 1}
- Let L = { w | w contains 00 as a substring }

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

11011100101
0000
1111101111100011111

Regular Expressions are Awesome

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

Regular Expressions are Awesome

- Let Σ = {0, 1}
- Let L = { w | |w| = 4 }

(0|1)(0|1)(0|1)(0|1)

Regular Expressions are Awesome

- Let Σ = {0, 1}
- Let L = { w | |w| = 4 }

(0|1)(0|1)(0|1)(0|1)

Regular Expressions are Awesome

- Let Σ = {0, 1}
- Let L = { w | |w| = 4 }

(0|1)(0|1)(0|1)(0|1)

0000
1010
1111
1000

Regular Expressions are Awesome

- Let Σ = {0, 1}
- Let L = { w | |w| = 4 }

(0|1)(0|1)(0|1)(0|1)

0000
1010
1111
1000

Regular Expressions are Awesome

- Let Σ = {0, 1}
- Let L = { w | |w| = 4 }

$$(0|1)^4$$

0000
1010
1111
1000

Regular Expressions are Awesome

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

$$(0|1)^4$$

0000
1010
1111
1000

Regular Expressions are Awesome

- Let $\Sigma = \{0, 1\}$
- Let L = { w | w contains at most one 0 }

Regular Expressions are Awesome

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

1*(0 | ε)1*

Regular Expressions are Awesome

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

1*(0 | ε)1*

Regular Expressions are Awesome

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

1*(0 | ε)1*

11110111
111111
0111
0

Regular Expressions are Awesome

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

1*(0 | ε)1*

11110111
111111
0111
0

Regular Expressions are Awesome

- Let Σ = {0, 1}
- Let L = { w | 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.”
- A regular expression for email addresses is

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

Regular Expressions are Awesome

- Let Σ = { a, ., @ }, where a represents “some letter.”
- A regular expression for email addresses is

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.”
- A regular expression for email addresses is

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.”
- A regular expression for email addresses is

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.”
- A regular expression for email addresses is

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.”
- A regular expression for email addresses is

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

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.”
- A regular expression for email addresses is

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

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.”
- A regular expression for email addresses is

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

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.”
- A regular expression for email addresses is

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

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

Regular Expressions are Awesome

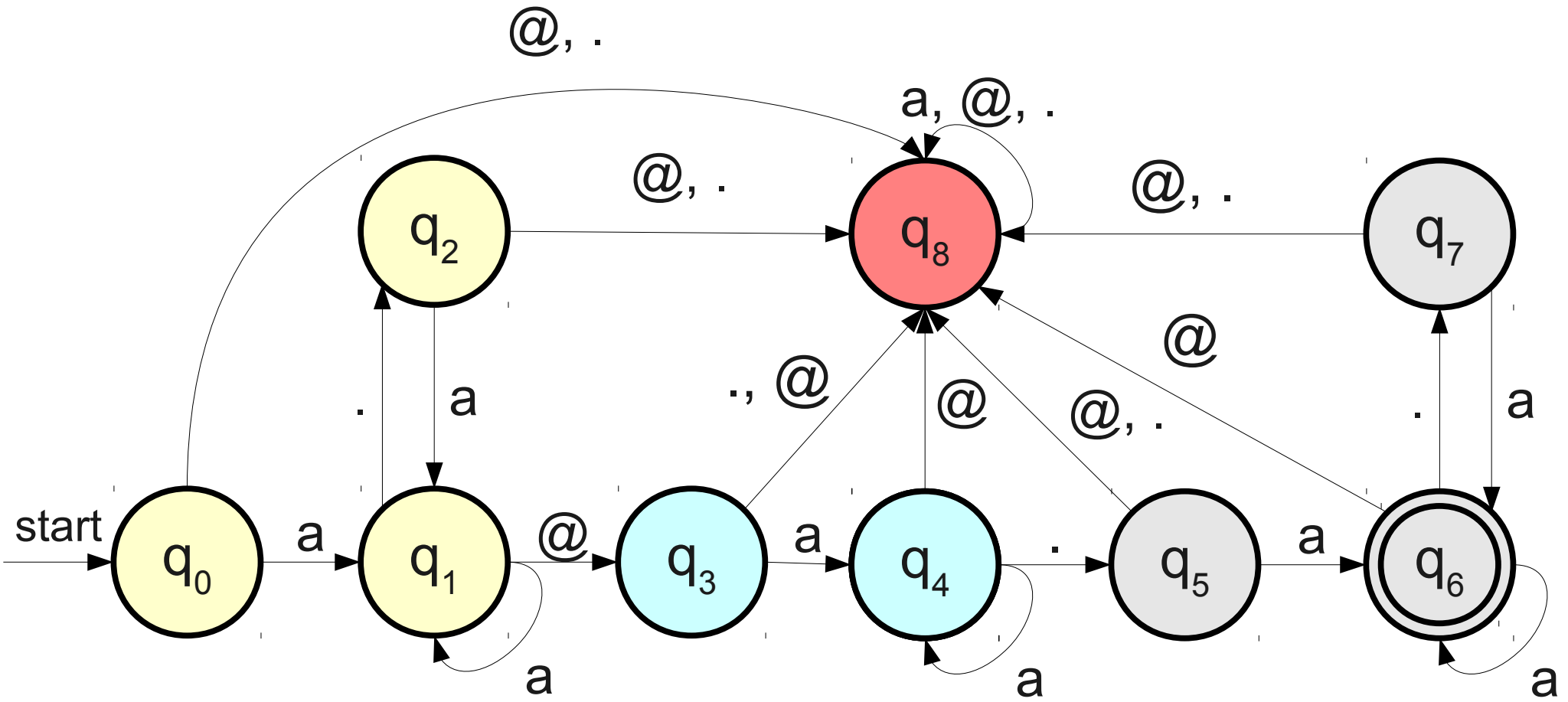
- Let Σ = { a, ., @ }, where a represents “some letter.”
- A regular expression for email addresses is

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

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

Regular Expressions are Awesome

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



Extensions to Regular Expressions

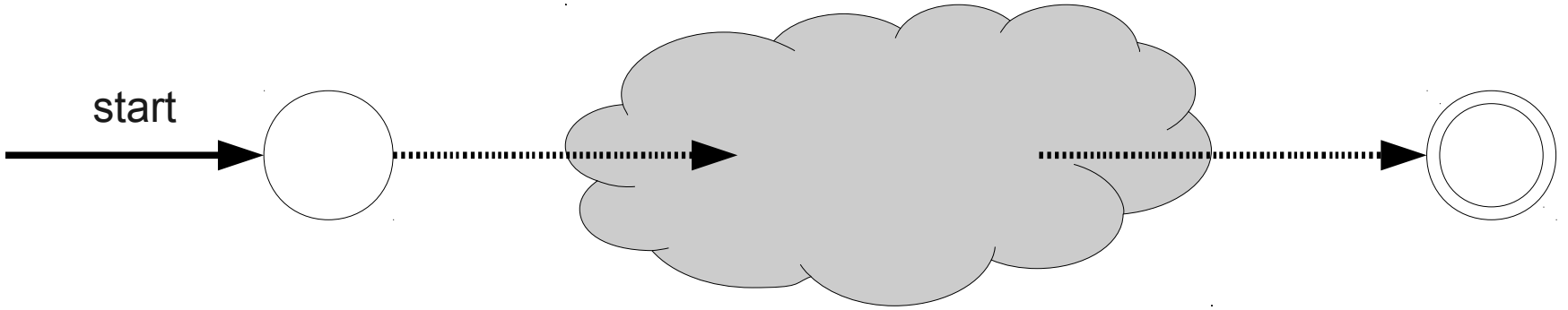
- To simplify regular expressions, we introduce the following shorthand (which is just **syntax sugar** for some other regular expression).
- R^n represents the language RR...R (n times).
- R? represents R | ε.
  - Either zero or one copies of R.
- R^+ represents RR*.
  - n ≥ 1 copies of R.
  - Sometimes called **Kleene Plus**.

Regular Expressions and Languages

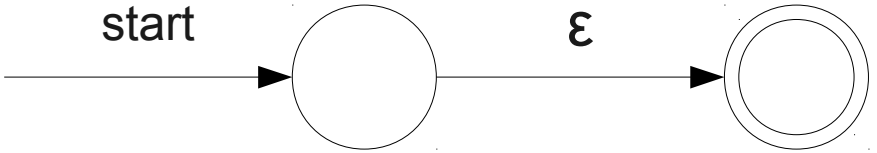
- So far, we have two characterizations of regular languages:
  - Any language accepted by some DFA.
  - Any language accepted by some NFA.
- **Theorem:** A language is regular iff it is described by some regular expression.
  - Need to show if a regular expression exists for L, L is regular.
  - Need to show that if L is regular, there is a regular expression for L.

A Marvelous Construction

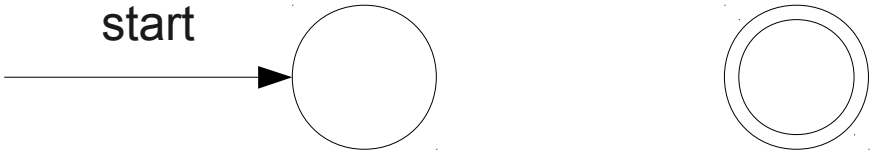
- To show that any language described by a regular expression is regular, we show how to convert a regular expression into an NFA.
- *Theorem:* For any regular expression R, there is an NFA N such that
  - L(R) = L(N)
  - N has exactly one accepting state.
  - N has no transitions into its start state.
  - N has no transitions out of its accepting state.



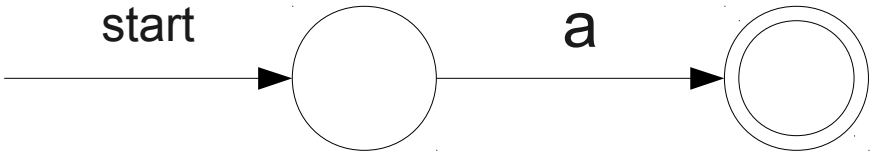
Base Cases



Automaton for ε



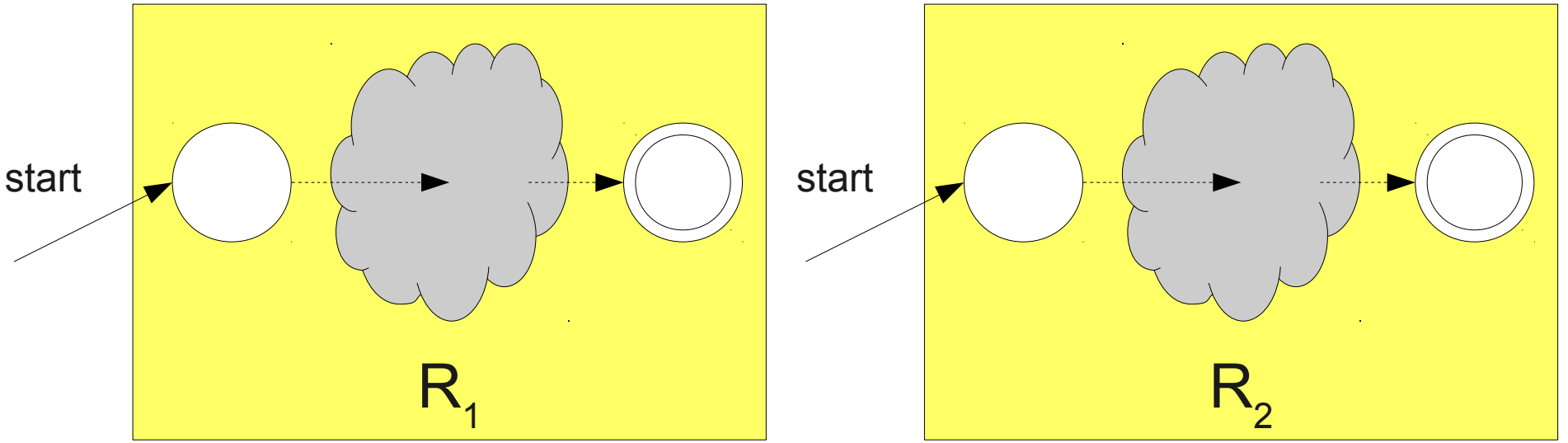
Automaton for Ø



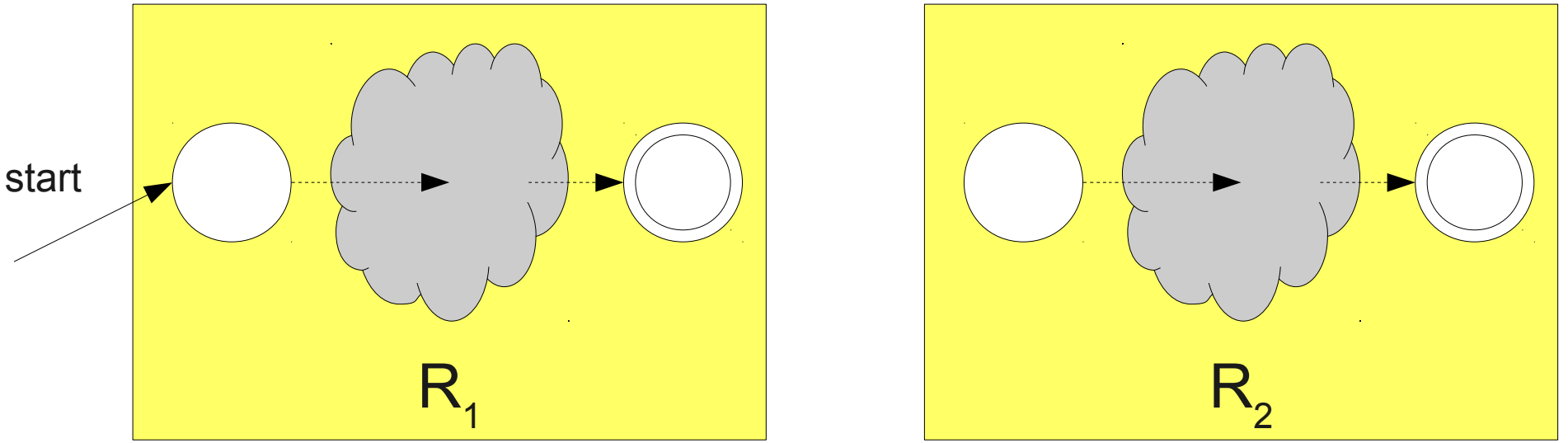
Automaton for single character **a**

Construction for R_1R_2

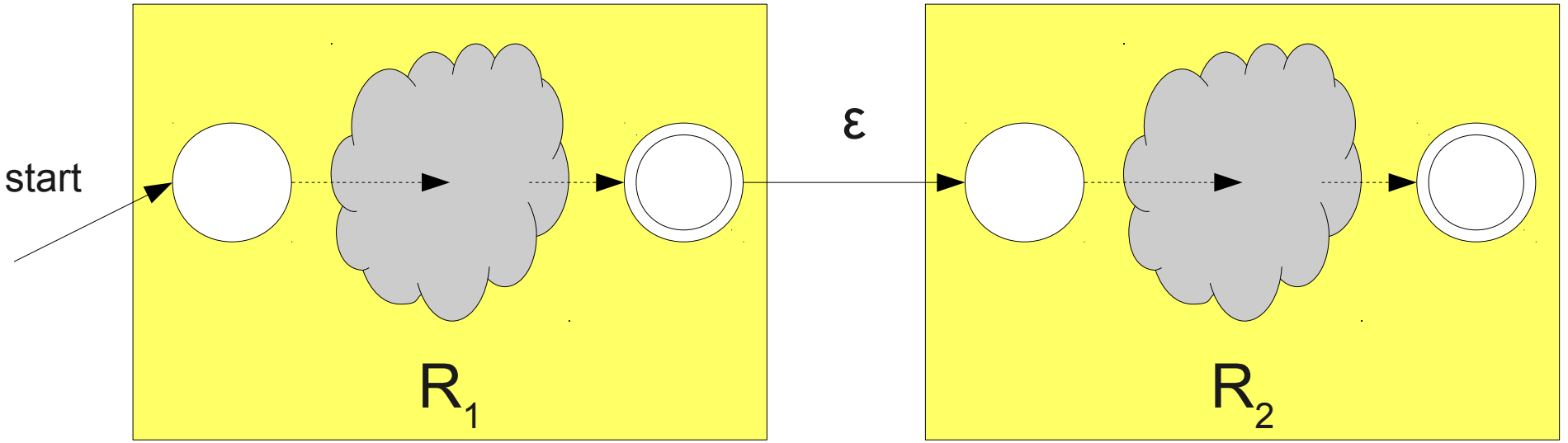
Construction for R_1R_2



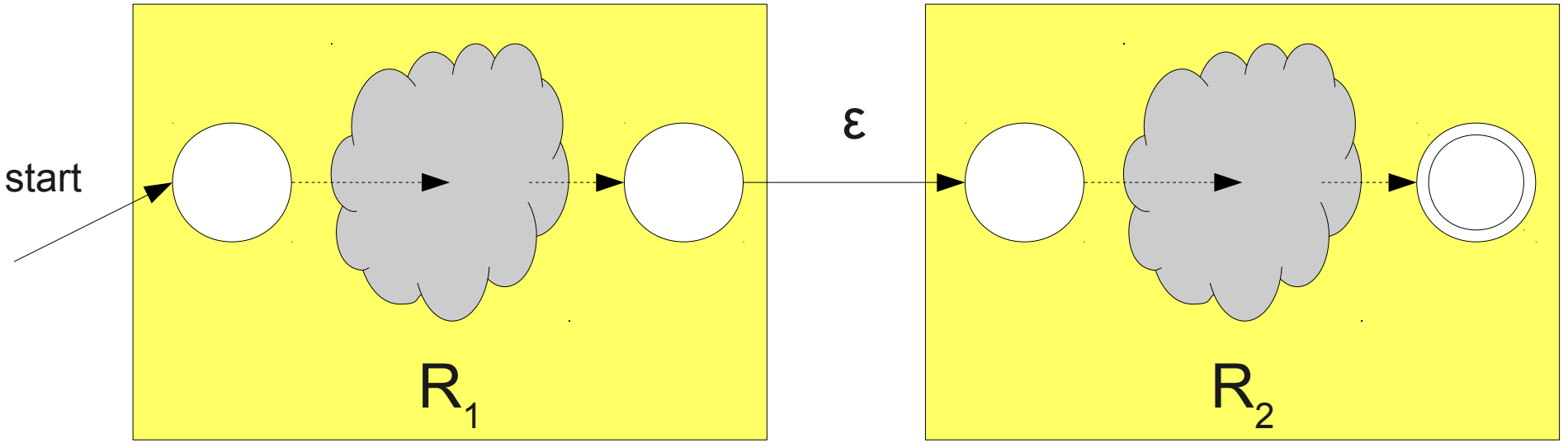
Construction for R_1R_2



Construction for R_1R_2

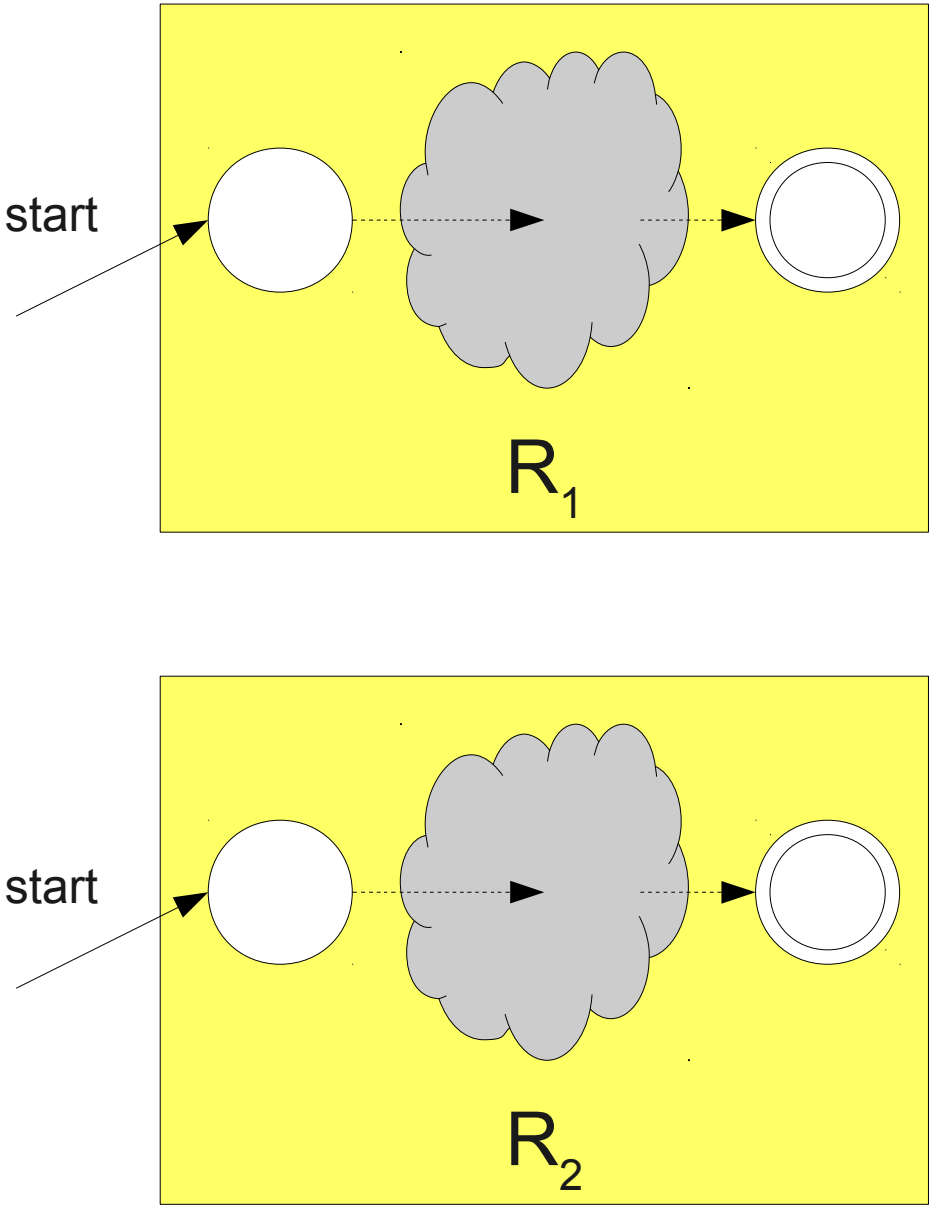


Construction for R_1R_2

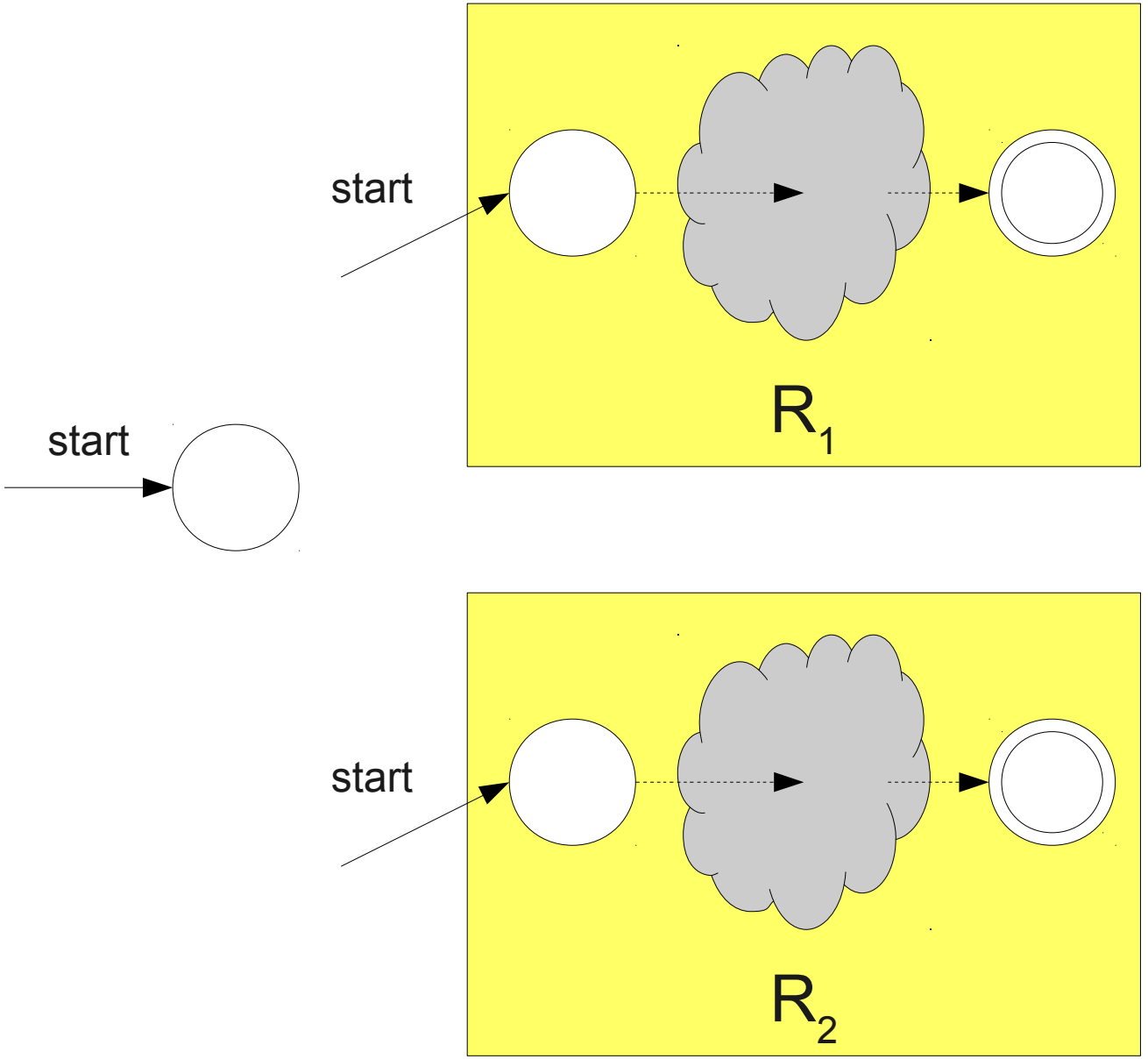


Construction for $R_1 \mid R_2$

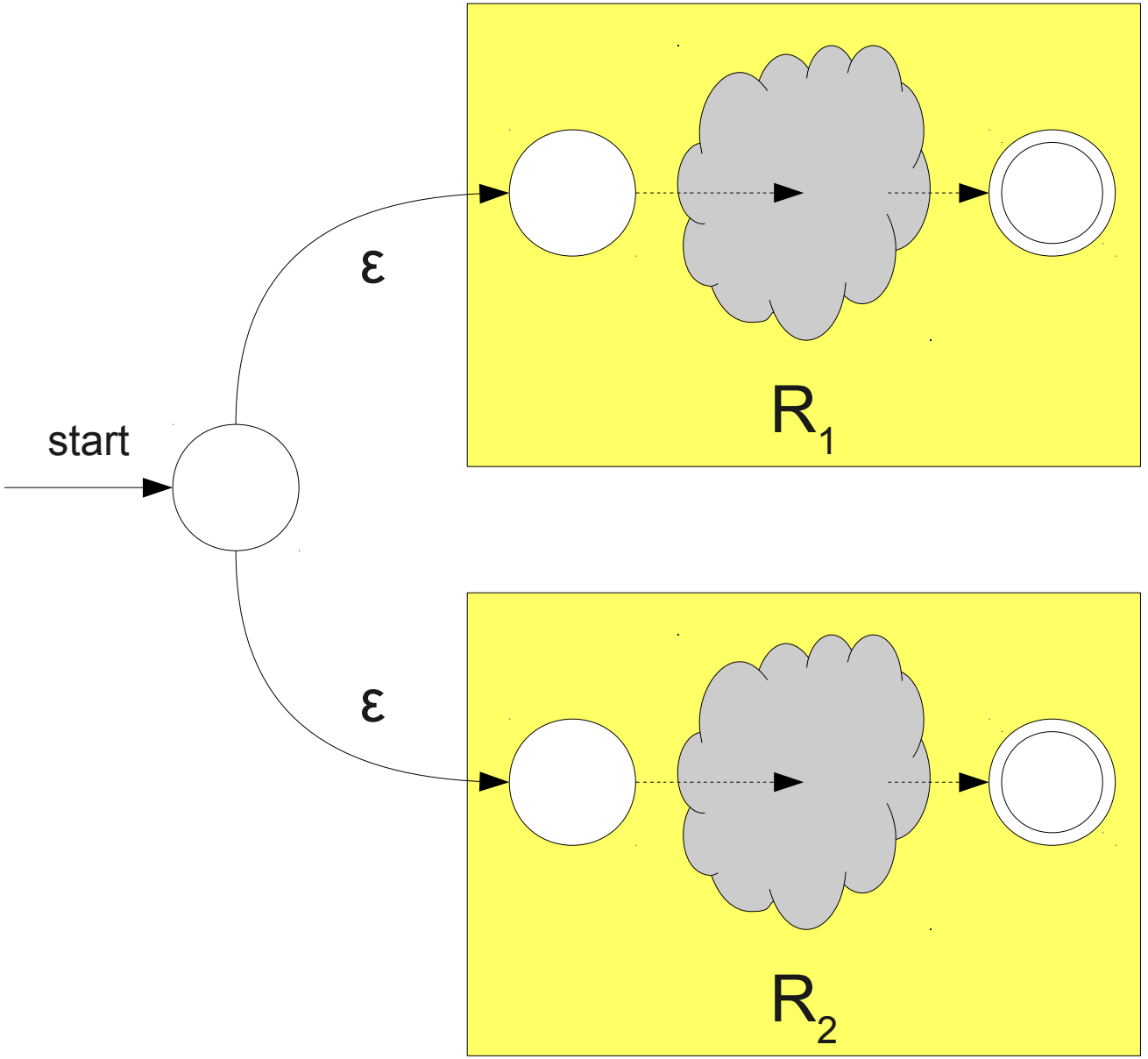
Construction for $R_1 \mid R_2$



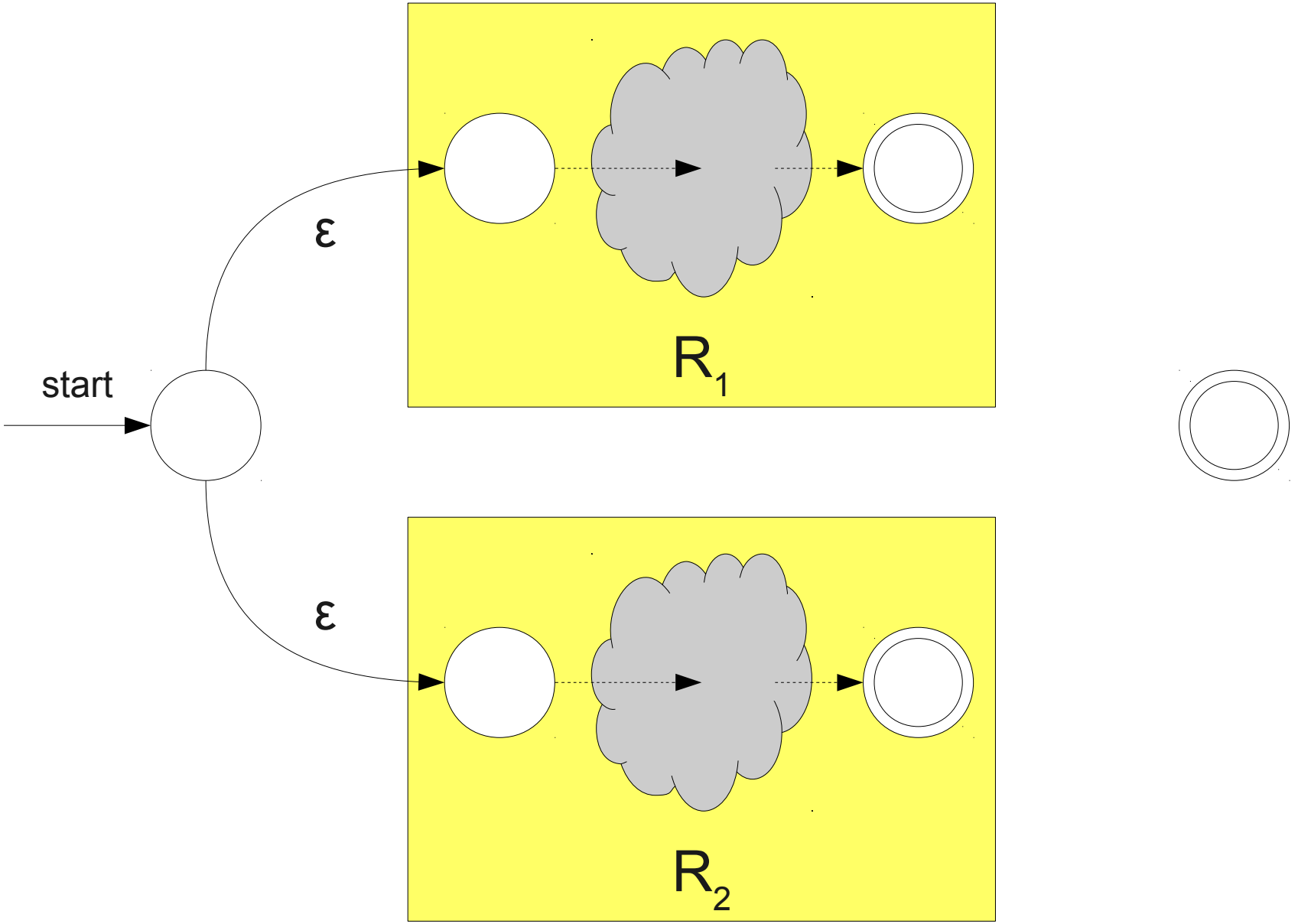
Construction for $R_1 \mid R_2$



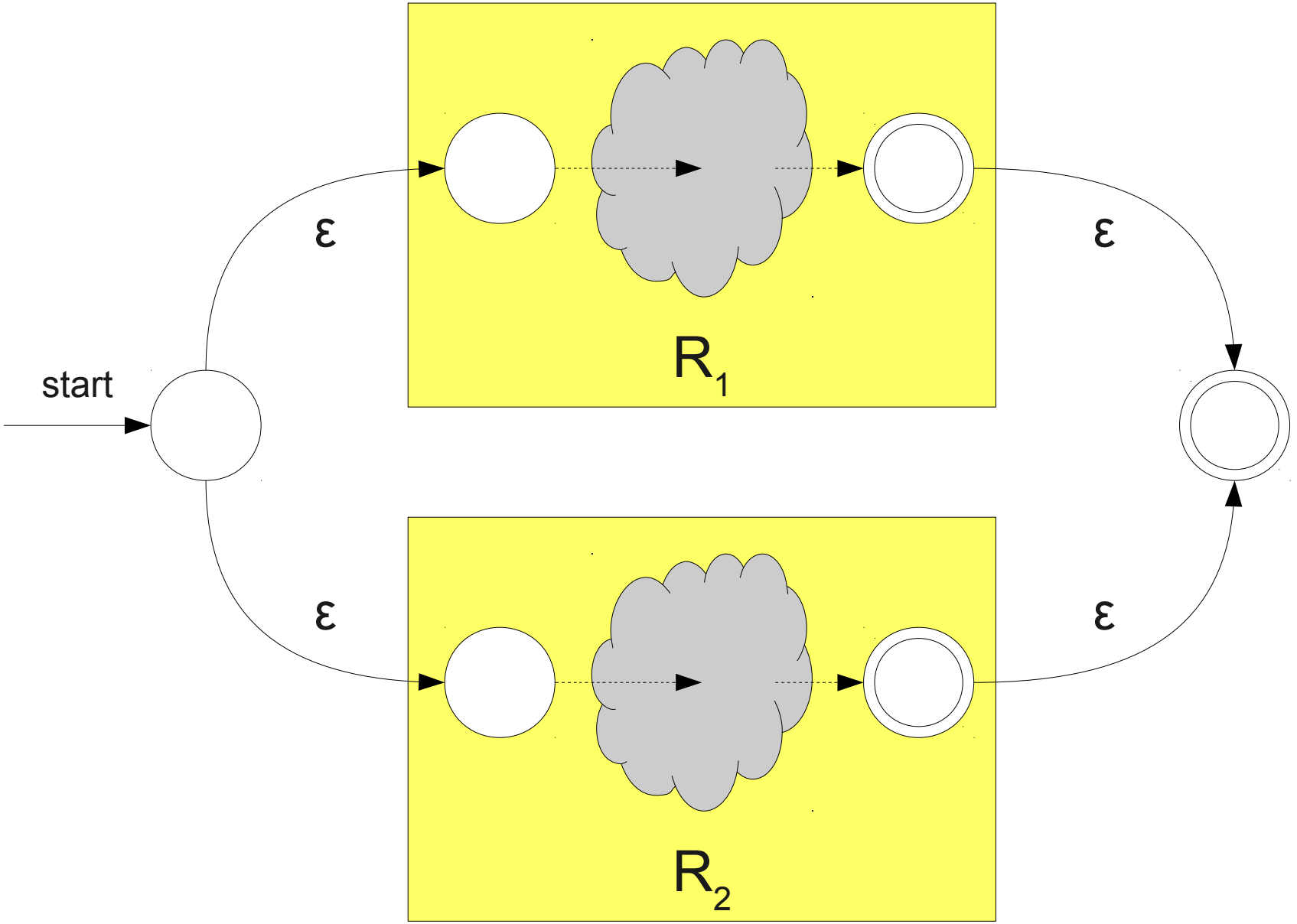
Construction for $R_1 \mid R_2$



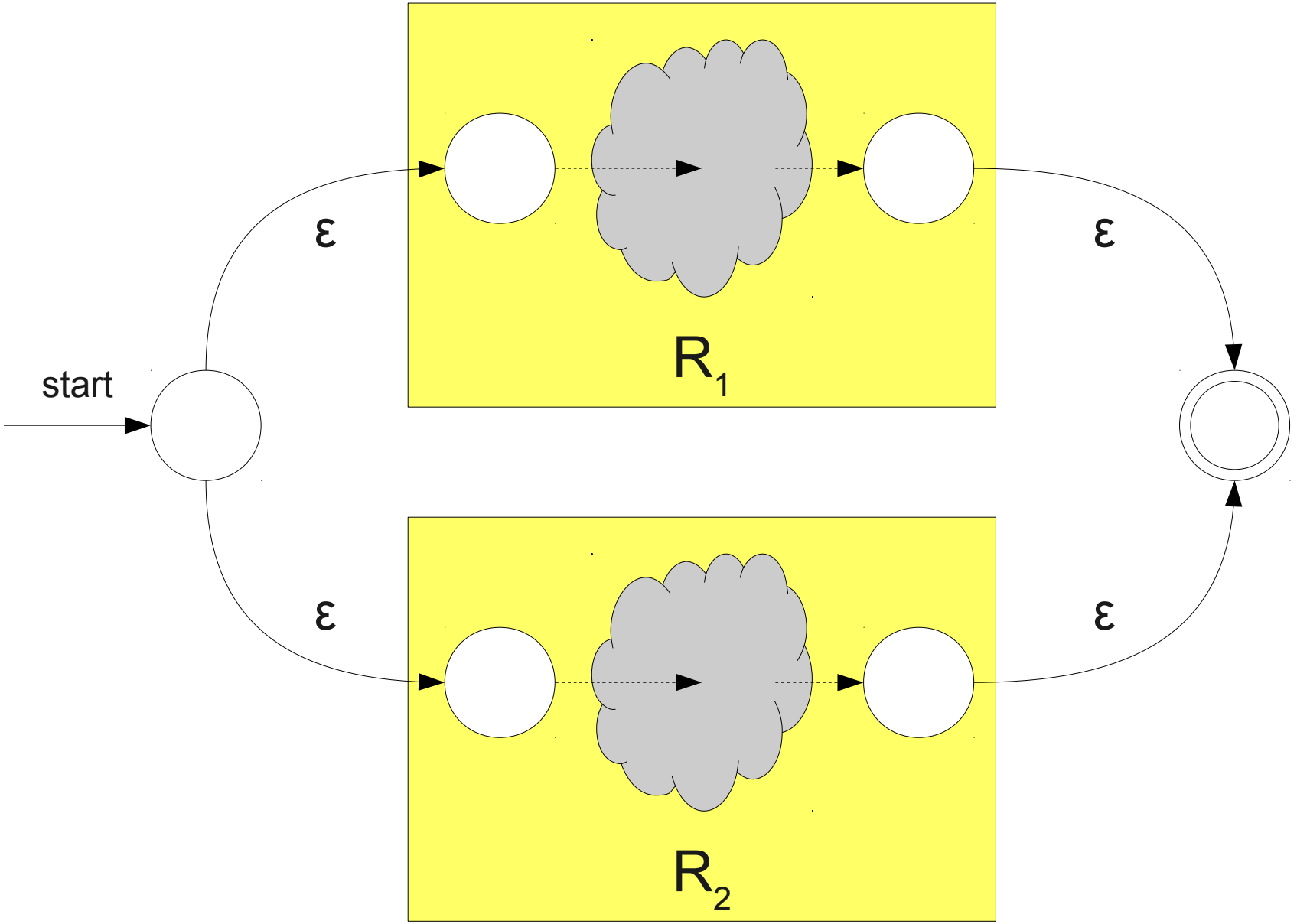
Construction for $R_1 \mid R_2$



Construction for $R_1 \mid R_2$

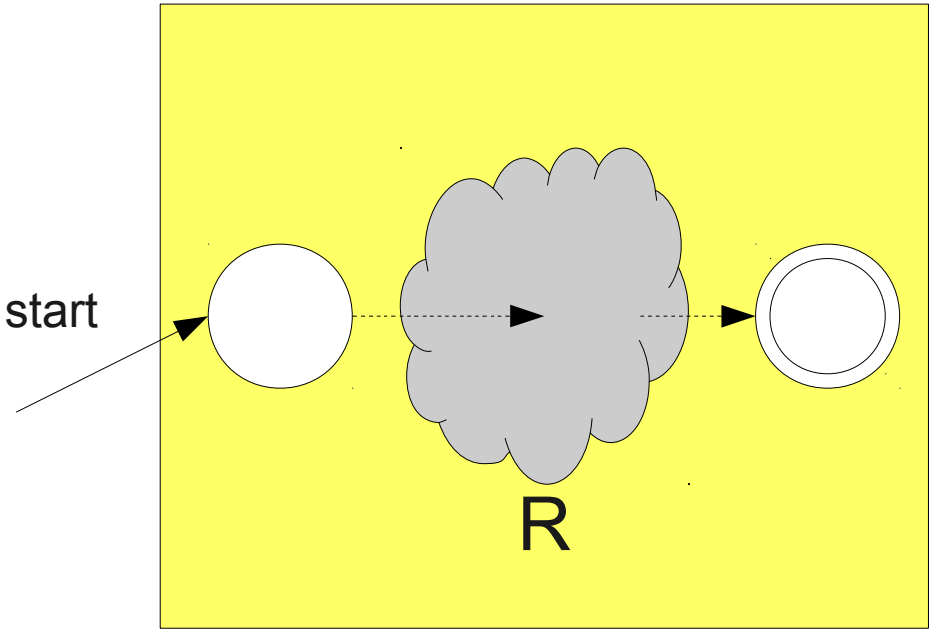


Construction for $R_1 \mid R_2$

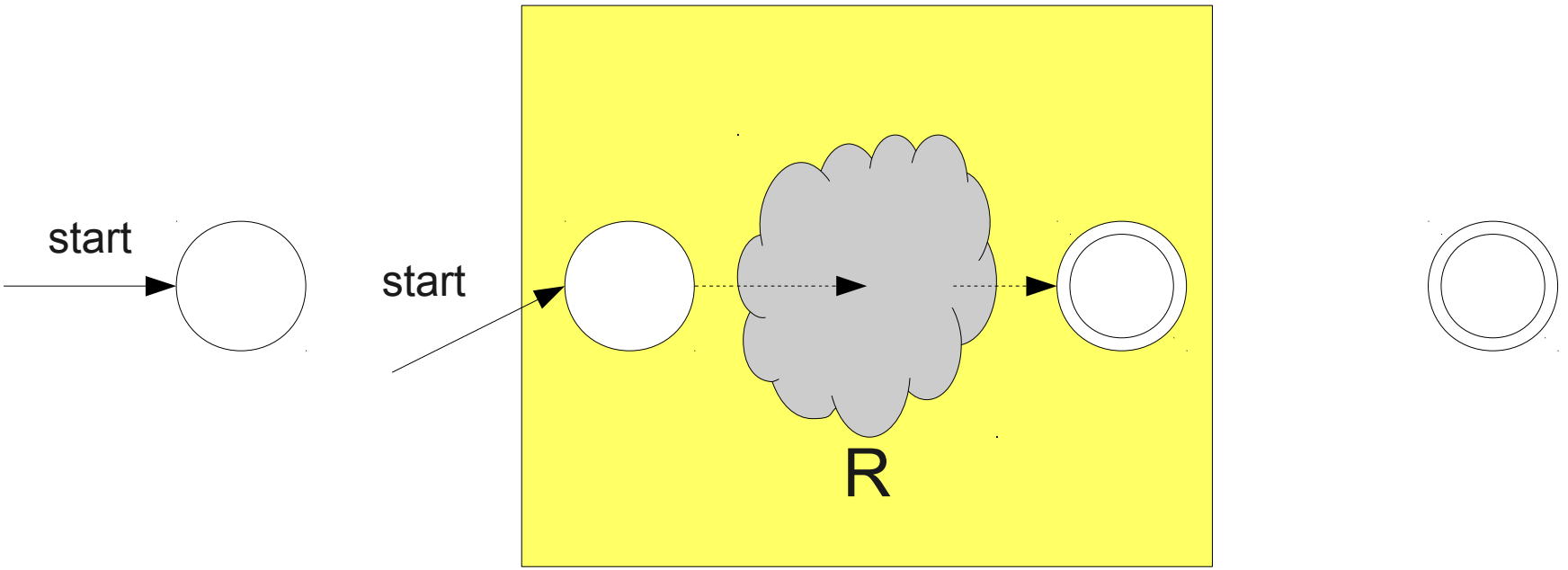


Construction for R*

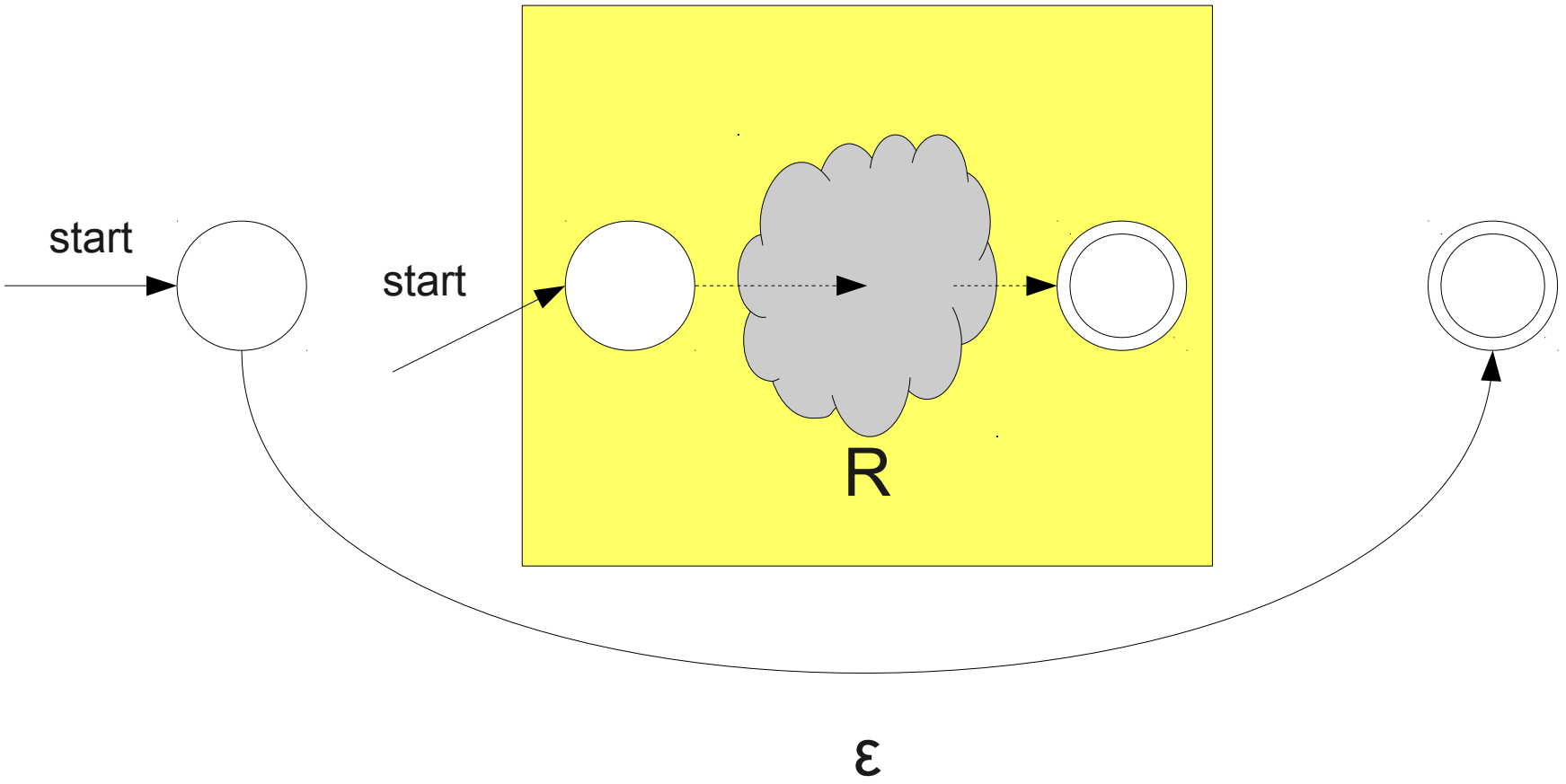
Construction for R*



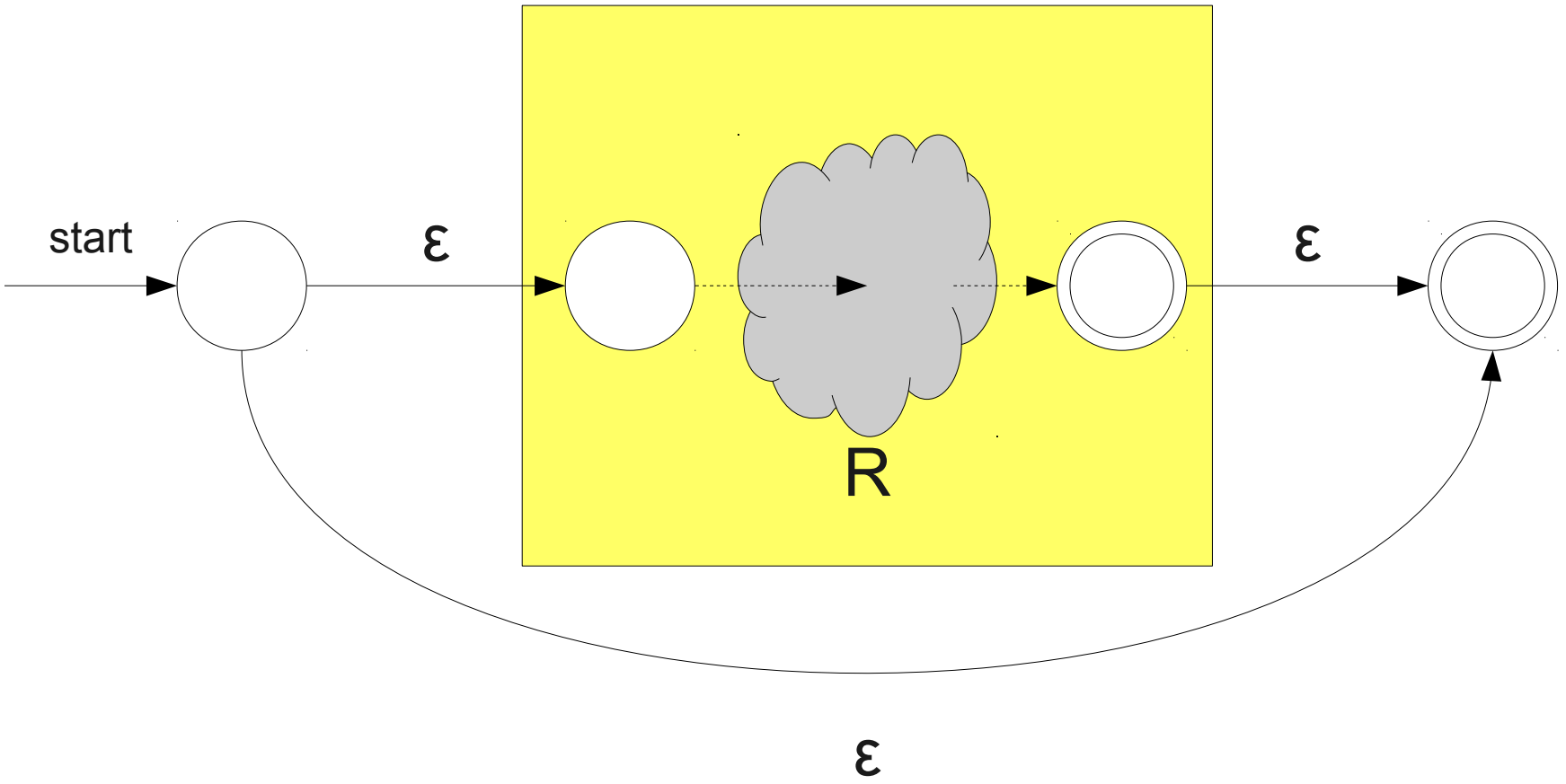
Construction for R*



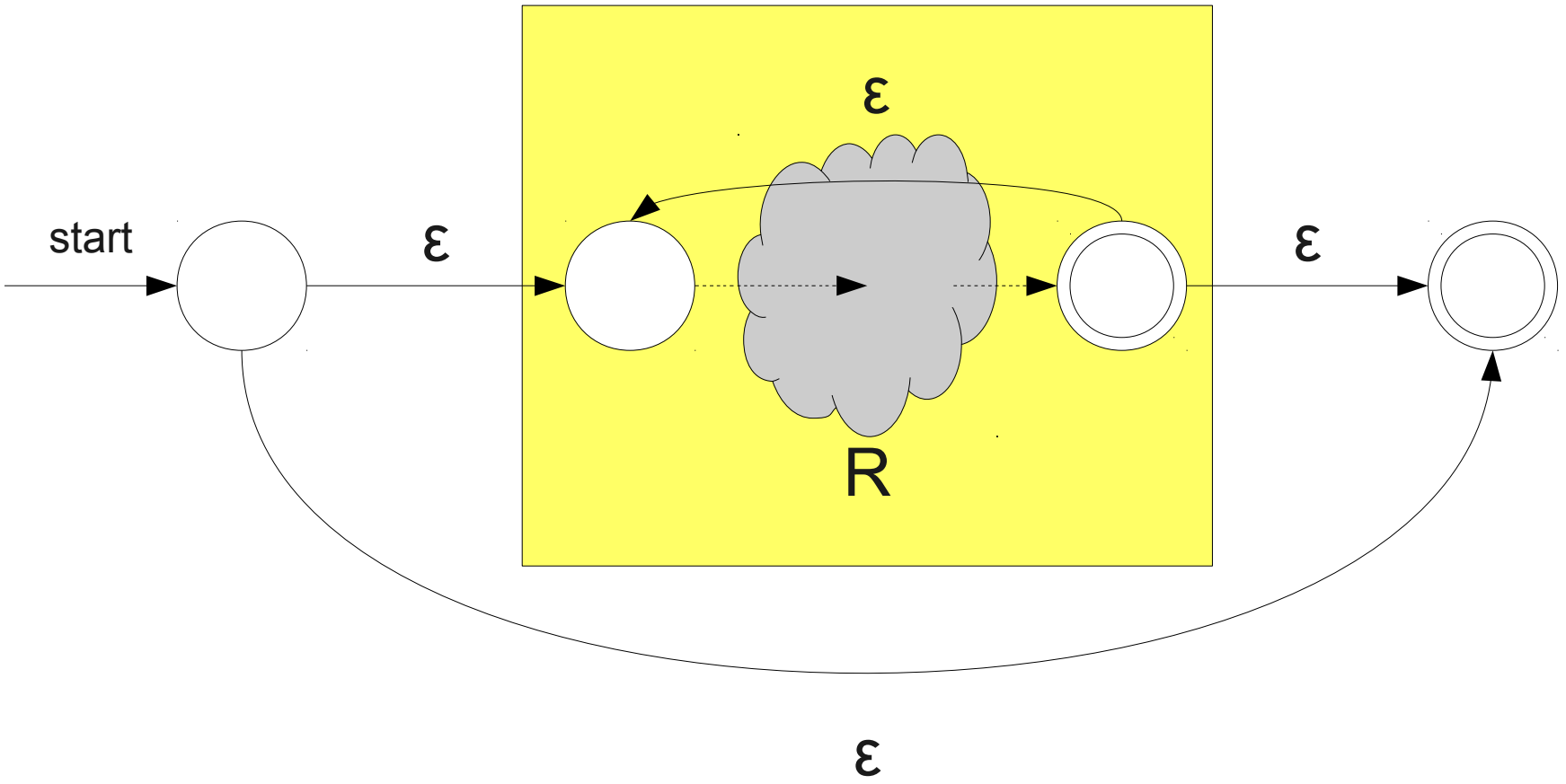
Construction for R*



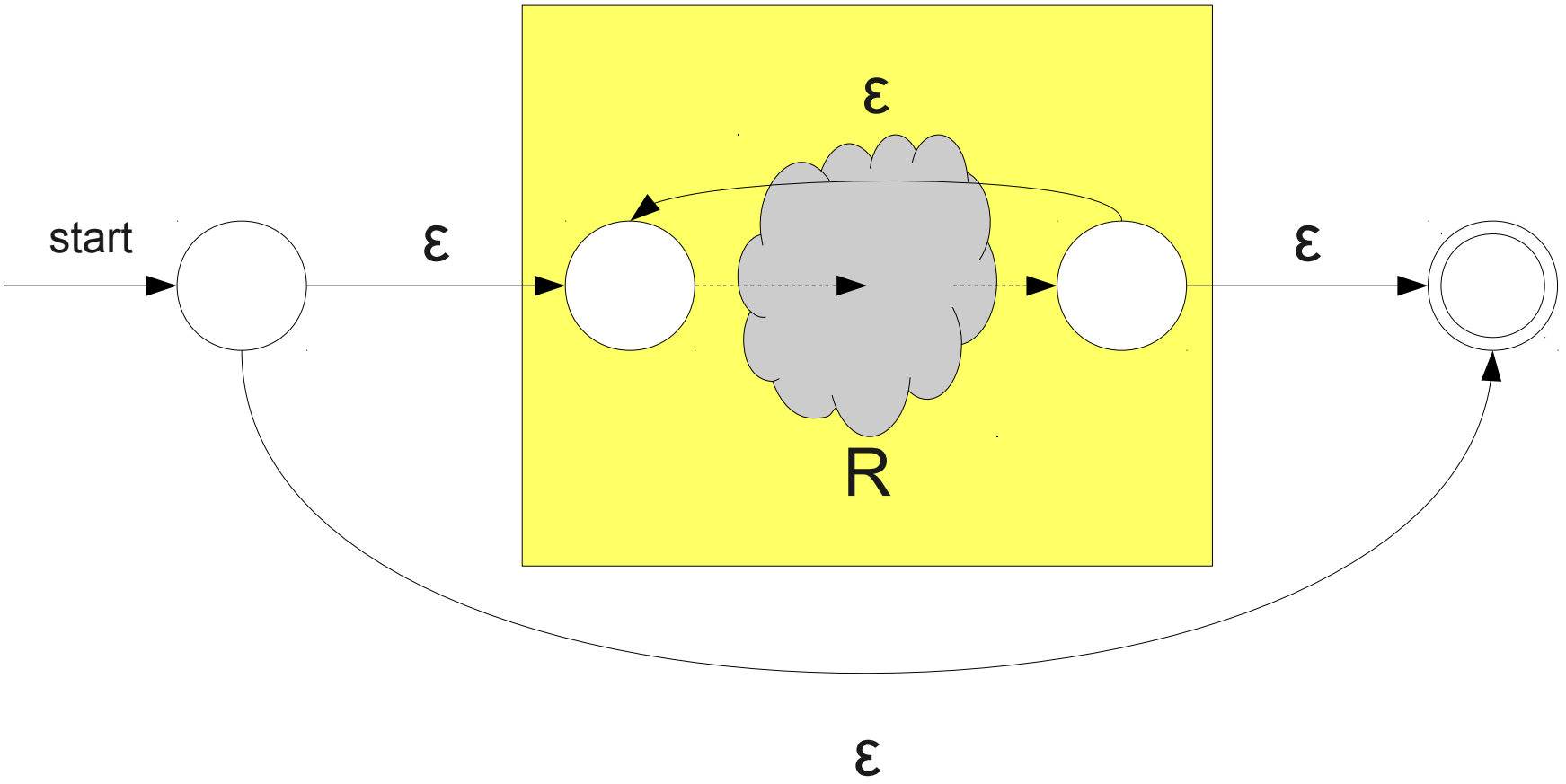
Construction for R*



Construction for R*



Construction for R*



The Other Direction

- Proving that if L is regular, there is a regular expression R for L is much trickier.
- There are two general approaches:
  - **Option One:** Convert a DFA for L into a regular expression by using **generalized DFAs**.
  - **Option Two:** Convert a DFA for L into a regular expression by using a **transitive closure algorithm**.
- Both approaches are theoretically elegant but unintuitive; read Sipser's discussion of the first option if you're interested.

Regular Languages

- The following are all equivalent:
  - L is a regular language.
  - L is accepted by some DFA.
  - L is accepted by some NFA.
  - L is described by some regular expression.
- What constructions on regular languages can we do with regular expressions?

String Homomorphism

- Given two alphabets Σ_1 and Σ_2, a **homomorphism from Σ_1 to Σ_2** is a function $h : \Sigma_1 \rightarrow \Sigma_2^*$.
- Intuitively, associates each symbol in Σ_1 with some (potentially empty) string in Σ_2^*.
- Example: Let $\Sigma_1 = \{ 0, 1 \}$, $\Sigma_2 = \{ a, b, c\}$
  - $h(0) = ab$
  - $h(1) = cc$

String Homomorphism

- Given a homomorphism $h : \Sigma_1 \rightarrow \Sigma_2^*$, the **image of a string w** is the string formed by replacing each symbol of $w \in \Sigma_1^*$ with the string specified by *h*.
- Example: Let $\Sigma_1 = \{ 0, 1 \}$, $\Sigma_2 = \{ a, b, c\}$
  - $h(0) = ab$
  - $h(1) = cc$
- Then $h(1101) = ccccabcc$.

String Homomorphism

- Given a homomorphism $h : \Sigma_1 \rightarrow \Sigma_2^*$, the **image of a string w** is the string formed by replacing each symbol of $w \in \Sigma_1^*$ with the string specified by *h*.

String Homomorphism

- Given a homomorphism $h : \Sigma_1 \rightarrow \Sigma_2^*$, the **image of a string w** is the string formed by replacing each symbol of $w \in \Sigma_1^*$ with the string specified by *h*.
- Example: Let $\Sigma_1 = \{ 0, 1 \}$, $\Sigma_2 = \{ a, b, c\}$
  - $h(0) = cbb$
  - $h(1) = \varepsilon$
- Then $h(1101) = cbb$.

String Homomorphism, Intuitively

- String homomorphism represents building a new string that has the **same structure** as an older string.
- Example: Let Σ_1 = { 0, 1, 2 } and consider the string 0121
- If Σ_2 = {A, B, C, …, Z, a, b, …, z, ', [,], . }, define the homomorphism $h : \Sigma_1 \rightarrow \Sigma_2^*$ as
  - h(0) = That's the way
  - h(1) = [Uh huh uh huh]
  - h(2) = I like it
- Then *h*(0121) = That's the way [Uh huh uh huh] I like it [Uh huh uh huh]
- Note that *h*(0121) has the same structure as 0121, just expressed differently.

String Homomorphisms, Formally

- Formally, if $h : \Sigma_1 \rightarrow \Sigma_2^*$ and $w \in \Sigma_1^*$, then $h(w)$ is defined as follows:
  - $h(\varepsilon) = \varepsilon$
  - $h(wa) = h(w)h(a)$
- Example: $h(0) = a$, $h(1) = bc$

$$
\begin{aligned}
& h(0110) \\
= \; & h(011)h(0) \\
= \; & h(01)h(1)h(0) \\
= \; & h(0)h(1)h(1)h(0) \\
= \; & h(\varepsilon)h(0)h(1)h(1)h(0) \\
= \; & \varepsilon abcbca \\
= \; & abcbca
\end{aligned}
$$

Homomorphisms of Languages

- If $L \subseteq \Sigma_1^*$ is a language and $h : \Sigma_1 \rightarrow \Sigma_2^*$ is a homomorphism, the language $h(L)$ is defined as

$$h(L) = \{ h(w) \mid w \in L \}$$

- The language formed by applying the homomorphism to every string in L.

Homomorphisms of Regular Languages

- If $L \subseteq \Sigma_1^*$ is a regular language and $h : \Sigma_1 \to \Sigma_2^*$ is a homomorphism, is $h(L)$ a regular language?
- **Idea**: Transform a regular expression for L into a regular expression for $h(L)$.
- Define $HOM(R)$ as
  - $HOM(\varepsilon) = \varepsilon$
  - $HOM(\emptyset) = \emptyset$
  - $HOM(a) = (h(a))$
  - $HOM(R_1\ R_2) = HOM(R_1)\ HOM(R_2)$
  - $HOM(R_1 \mid R_2) = HOM(R_1) \mid HOM(R_2)$
  - $HOM(R^*) = HOM(R)^*$
  - $HOM((R)) = (HOM(R))$

Homomorphisms of Regular Languages

- If $L \subseteq \Sigma_1^*$ is a regular language and $h : \Sigma_1 \to \Sigma_2^*$ is a homomorphism, is $h(L)$ a regular language?
- **Idea**: Transform a regular expression for L into a regular expression for $h(L)$.
- Define $HOM(R)$ as
  - $HOM(\varepsilon) = \varepsilon$
  - $HOM(\emptyset) = \emptyset$
  - $HOM(a) = (h(a))$
  - $HOM(R_1 R_2) = HOM(R_1) HOM(R_2)$
  - $HOM(R_1 \mid R_2) = HOM(R_1) \mid HOM(R_2)$
  - $HOM(R^*) = HOM(R)^*$
  - $HOM((R)) = (HOM(R))$

Why do we parenthesize the result of $h(a)$?

Homomorphisms in Practice

- Consider the language (0120)* and the homomorphism
  - h(0) = n
  - h(1) = y
  - h(2) = a
- New language is

  ((n)(y)(a)(n))*

Homomorphisms in Practice

- Consider the language 011* and the homomorphism
  - h(0) = Here
  - h(1) = Kitty
- New language is

  (Here)(Kitty)(Kitty)*

Summary of Regular Expressions

- Regular expressions give a (usually) compact encoding for representing regular languages.
- Transformations on regular expressions can be used to show that regular languages are closed under **string homomorphism**.