Finite Automata

Part Three

Friday Four Square!

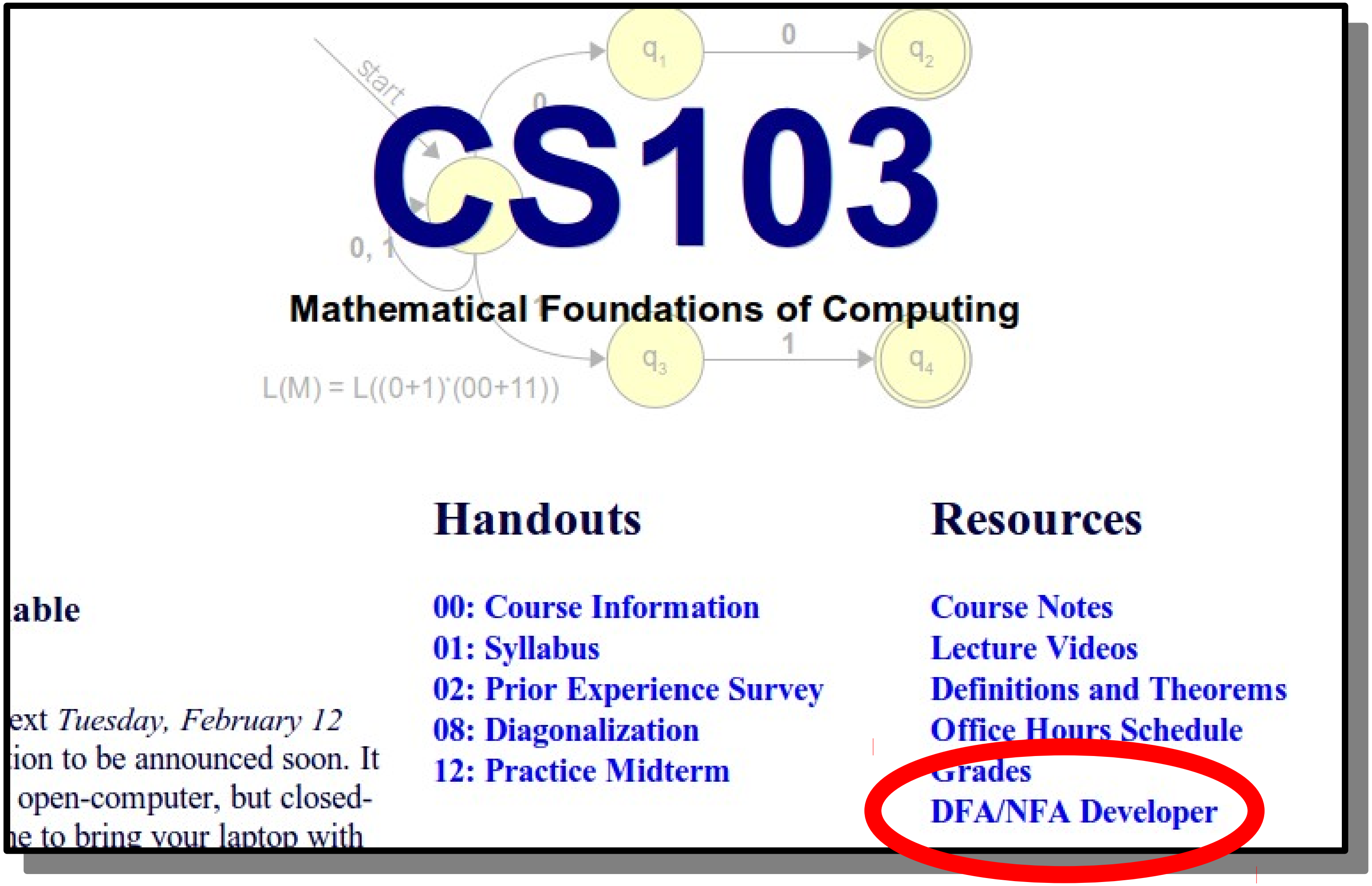
Today at 4:15PM, Outside Gates.

Announcements

- Problem Set 4 due at 2:15PM today.
  - We'll be around after lecture.
- Problem Set 5 out, due next Friday, February 15.
  - Play around with finite automata and regular languages.
  - **No checkpoint problems**.

Midterm

- Midterm is next **Tuesday, February 12** in **Hewlett 200** / **Hewlett 201**.
  - Can show up to either room.
- Covers material up through and including DFAs.
- Review session this **Sunday, February 10** in **Gates 104** from **5PM – 7PM**.
  - Show up with questions, leave with answers!


CS103
Mathematical Foundations of Computing
start
0
0, 1
1
q1
q2
q3
q4
L(M) = L((0+1)*(00+11))
Handouts
00: Course Information
01: Syllabus
02: Prior Experience Survey
08: Diagonalization
12: Practice Midterm
Resources
Course Notes
Lecture Videos
Definitions and Theorems
Office Hours Schedule
Grades
DFA/NFA Developer
able
ext Tuesday, February 12
ion to be announced soon. It
open-computer, but closed-
e to bring your laptop with

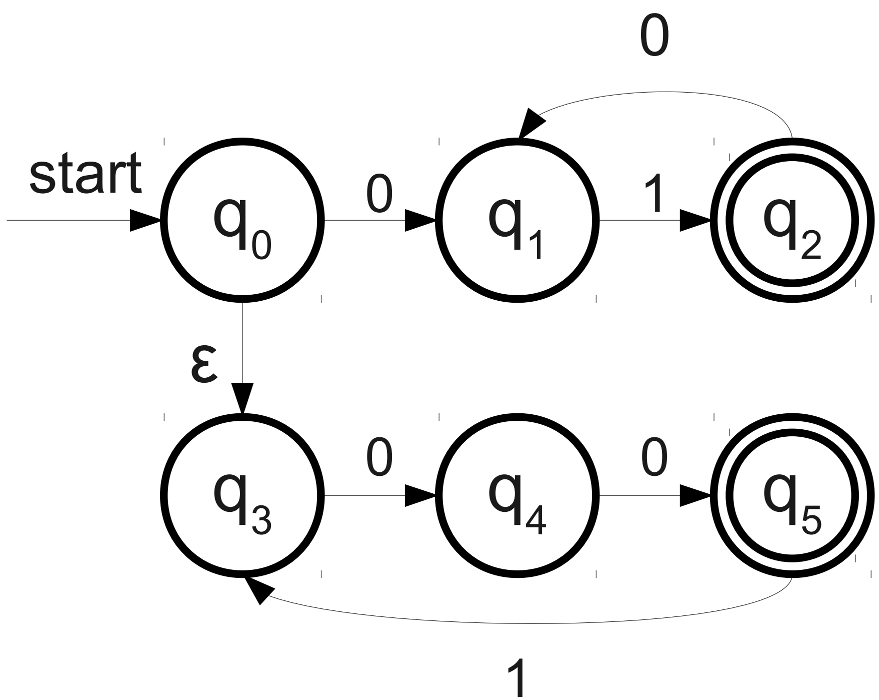
NFAs

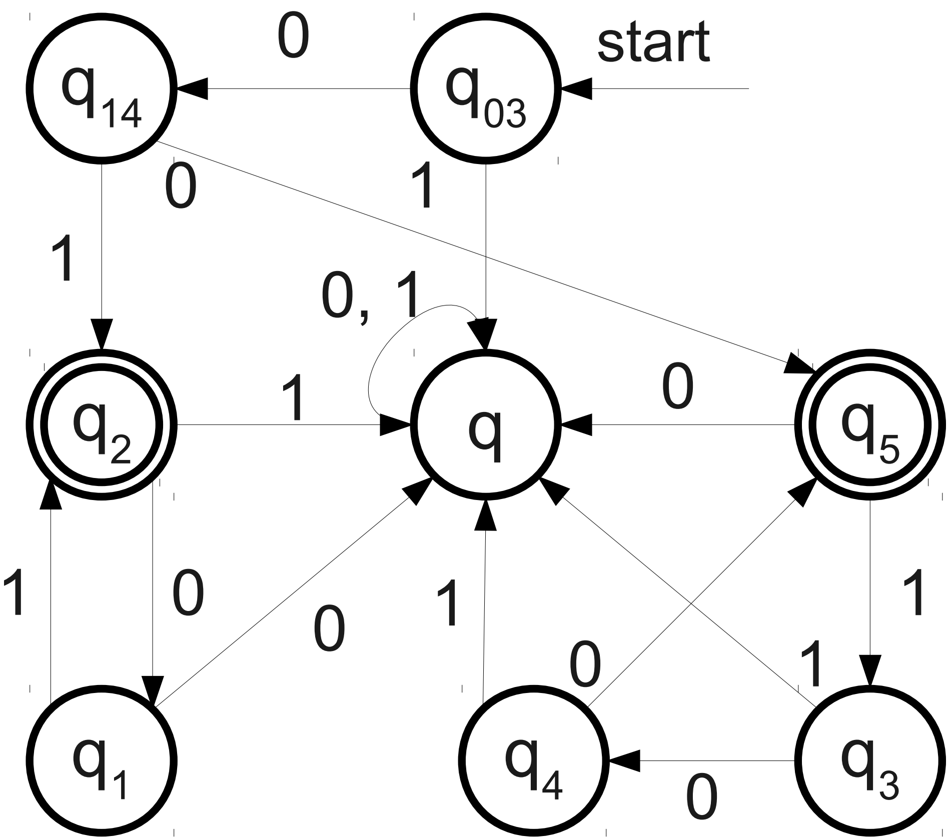
- An **NFA** is a
    - **N**ondeterministic
    - **F**inite
    - **A**utomaton
- Conceptually similar to a DFA, but equipped with the vast power of **nondeterminism**.
- There can be many or no transitions defined on certain inputs.
- An NFA accepts a string if *any* series of choices causes the string to enter an accepting state.

Three Intuitions for Nondeterminism

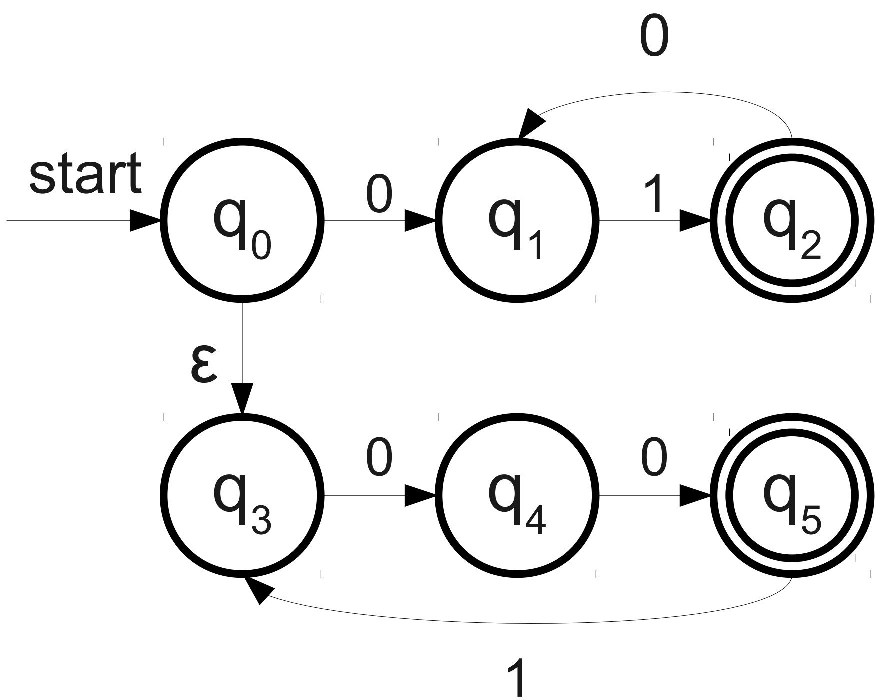
Tree Computation
Massive Parallelism
Perfect Guessing

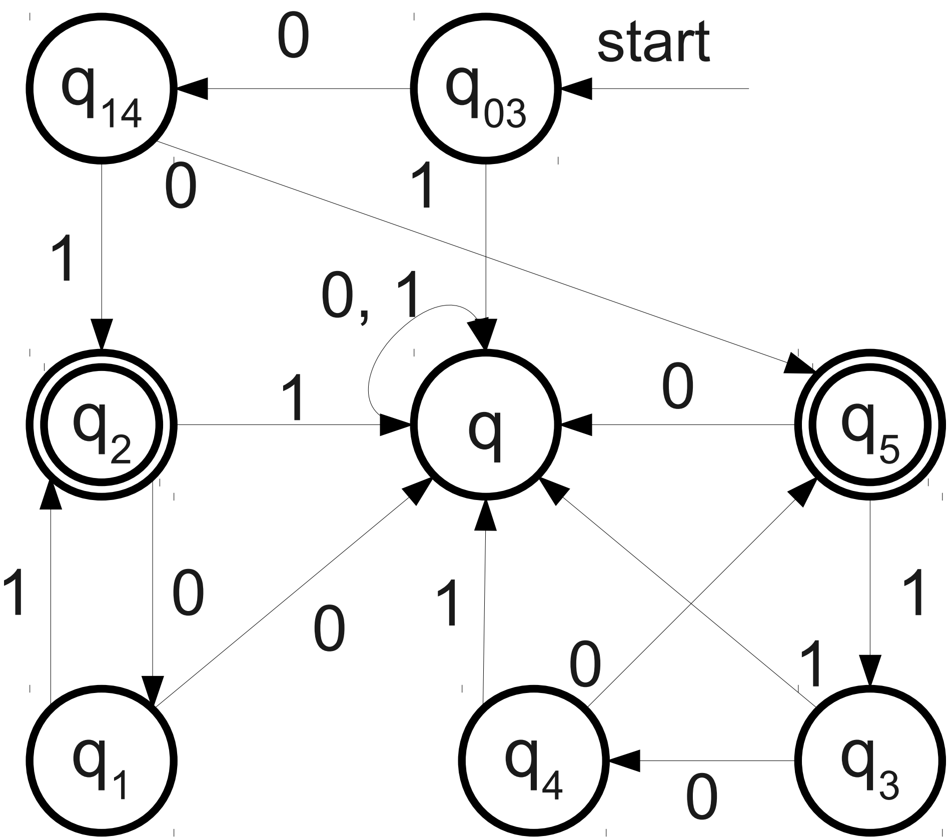
Simulating an NFA with a DFA



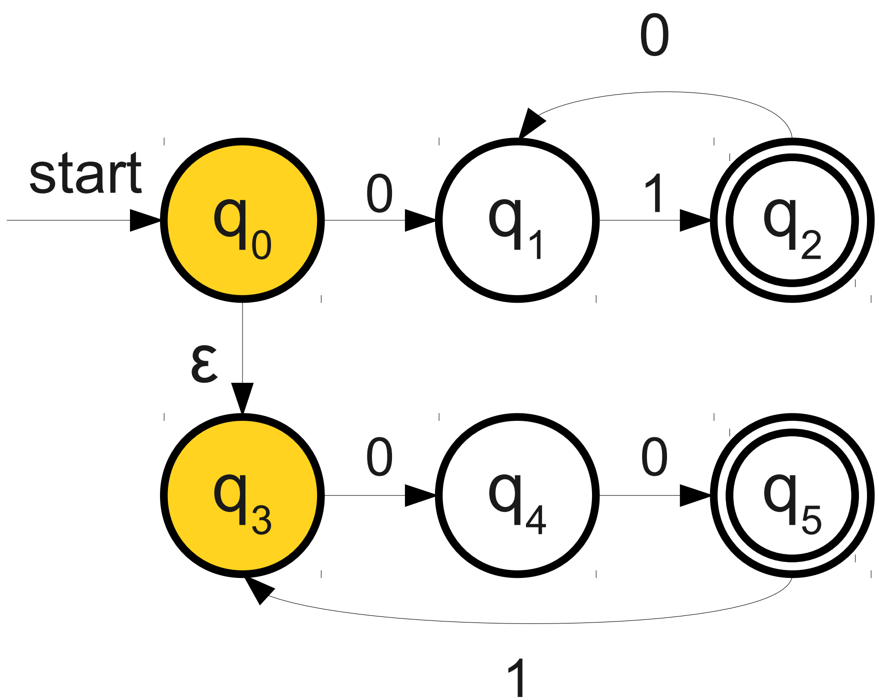


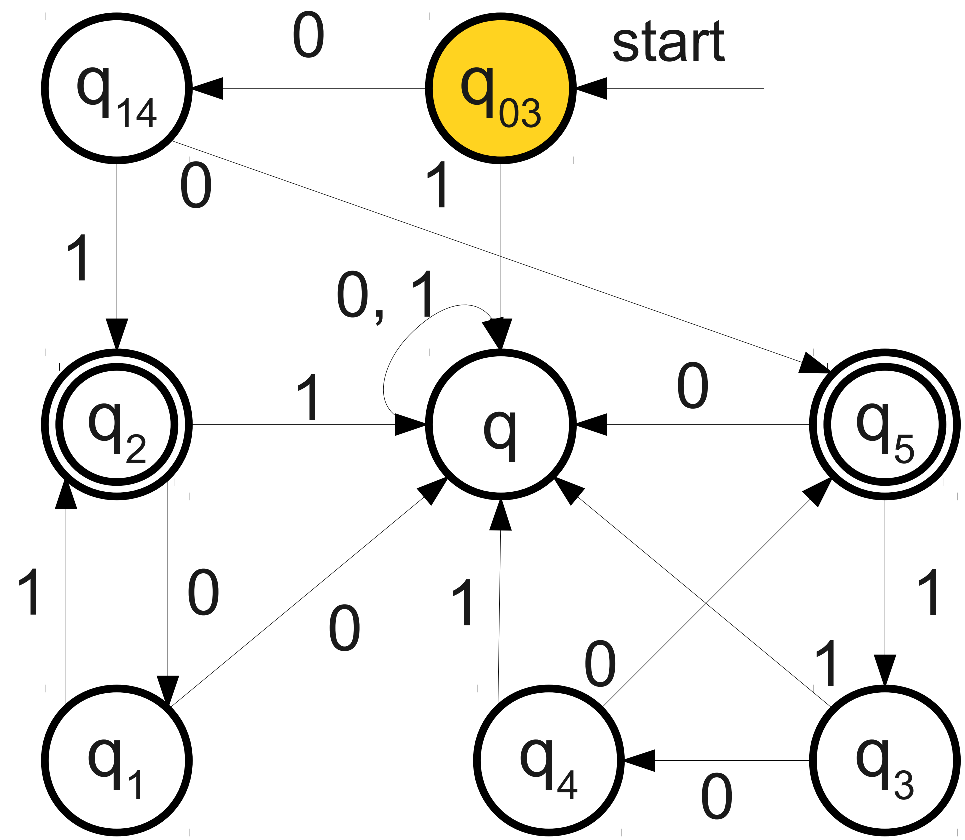
Simulating an NFA with a DFA



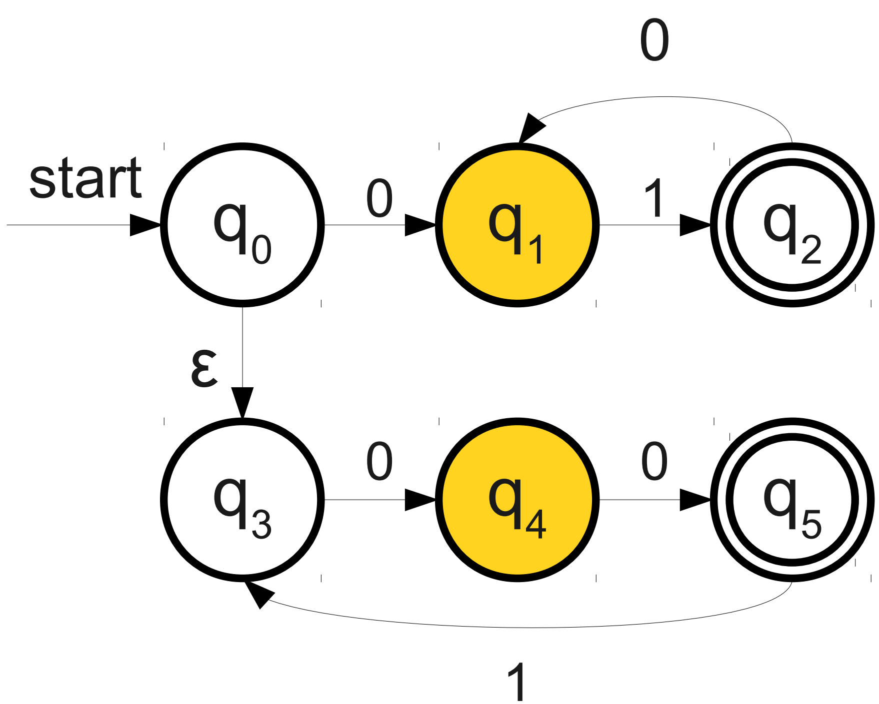


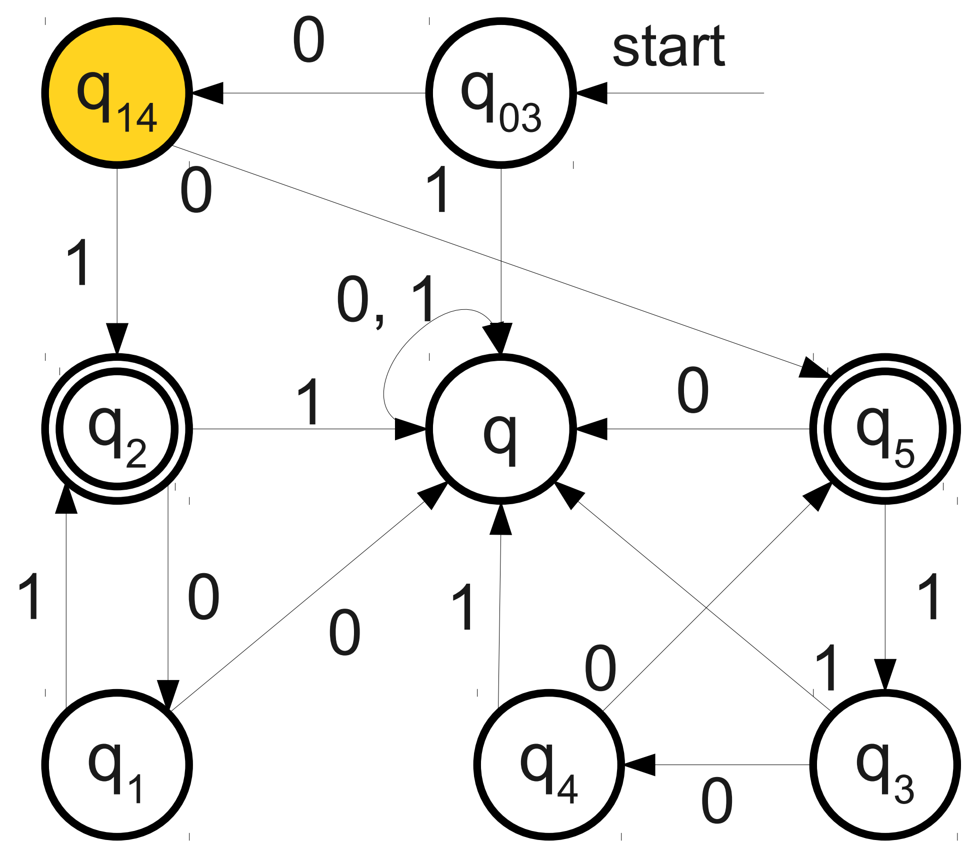
Simulating an NFA with a DFA



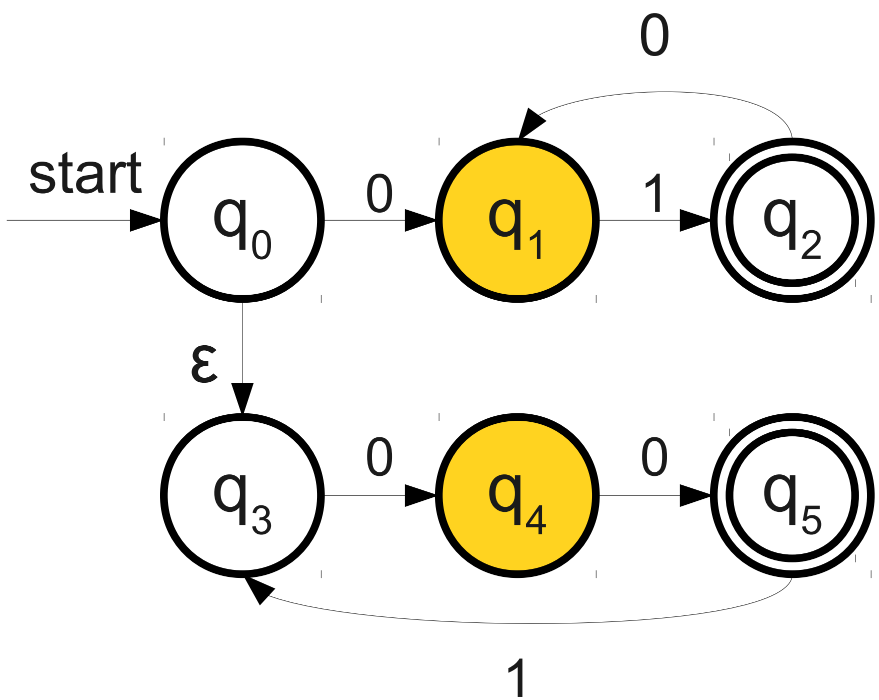


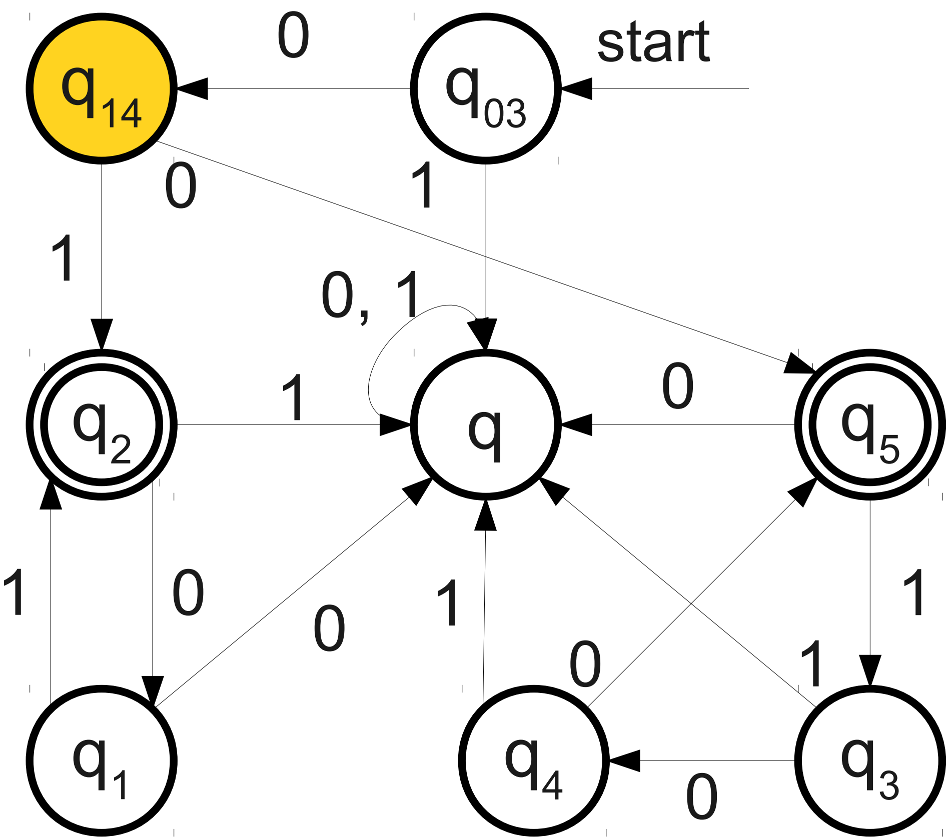
Simulating an NFA with a DFA





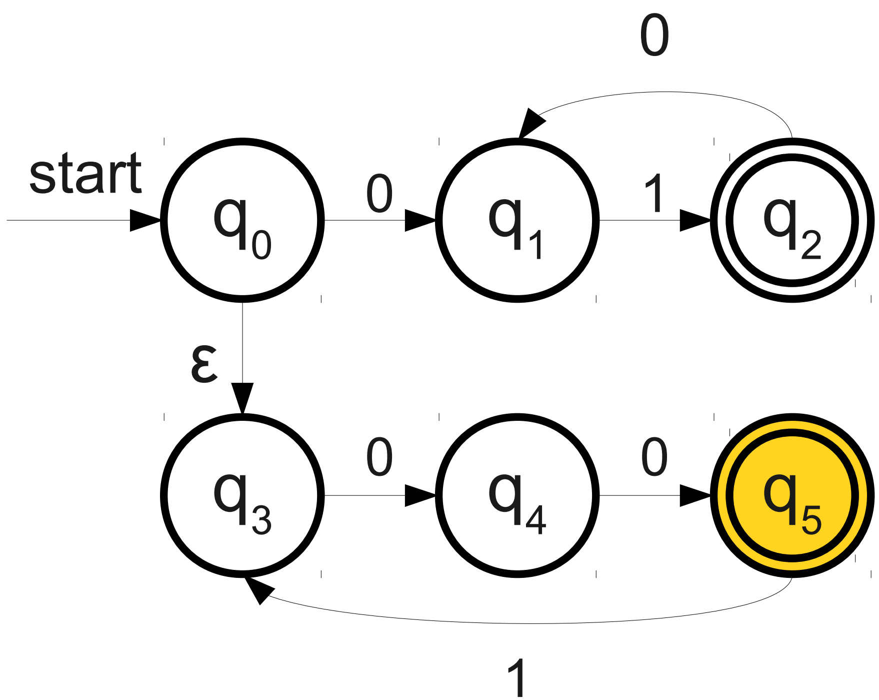
Simulating an NFA with a DFA

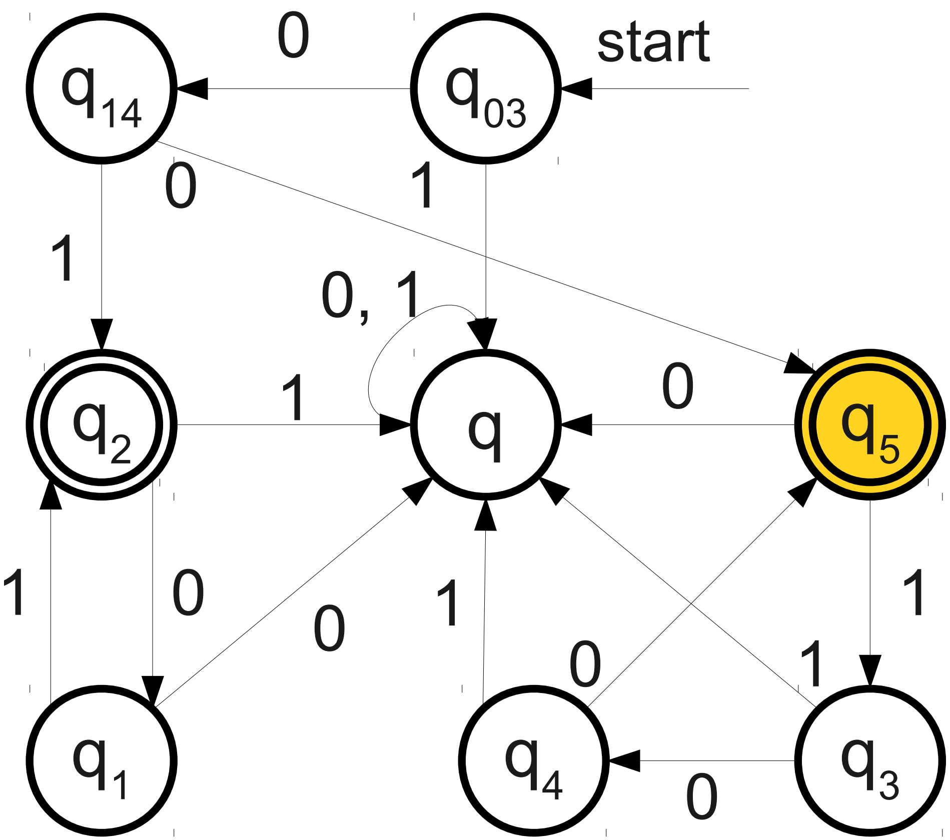


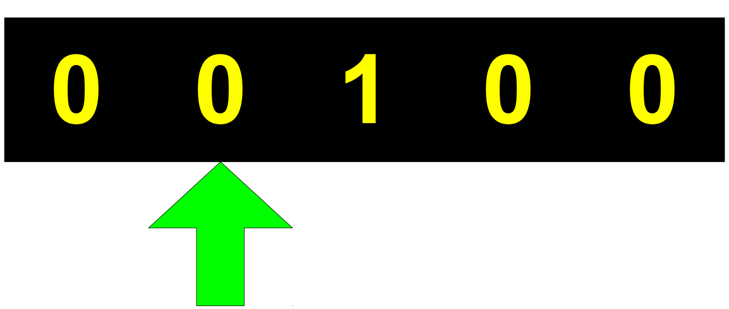


0 0 1 0 0

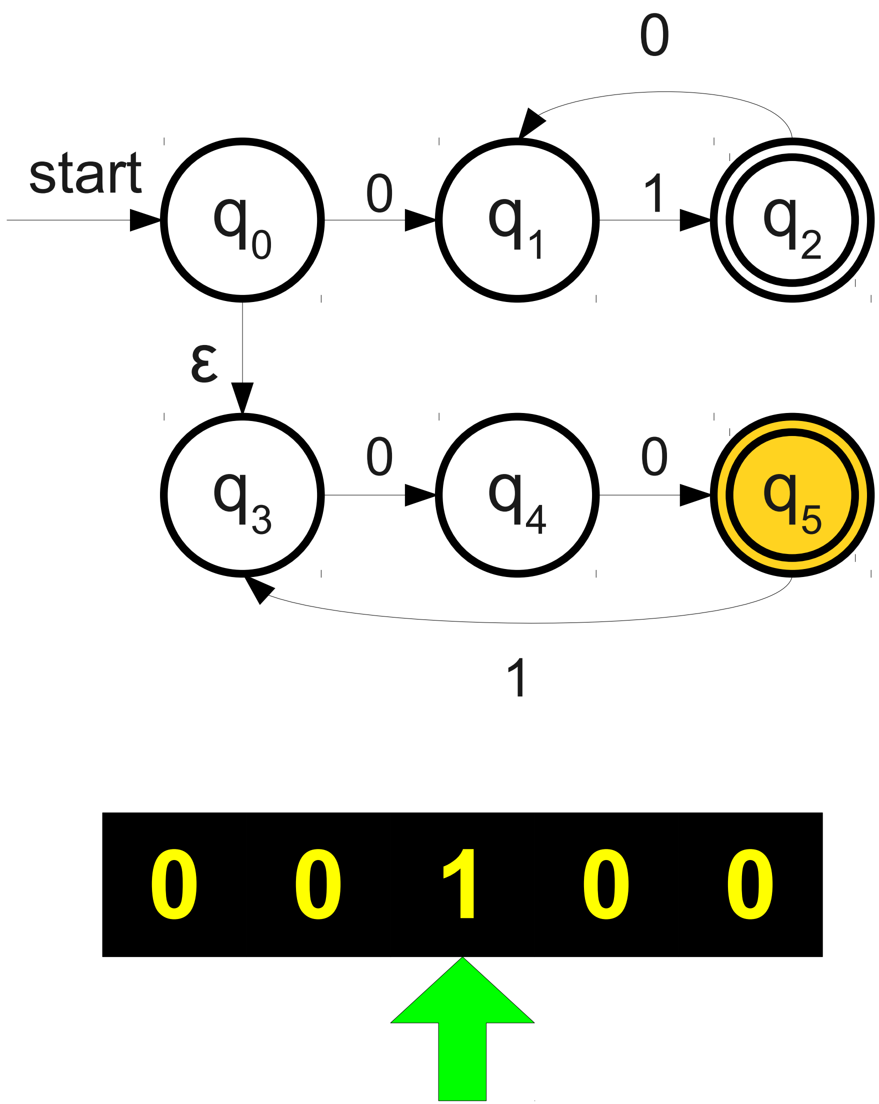
Simulating an NFA with a DFA

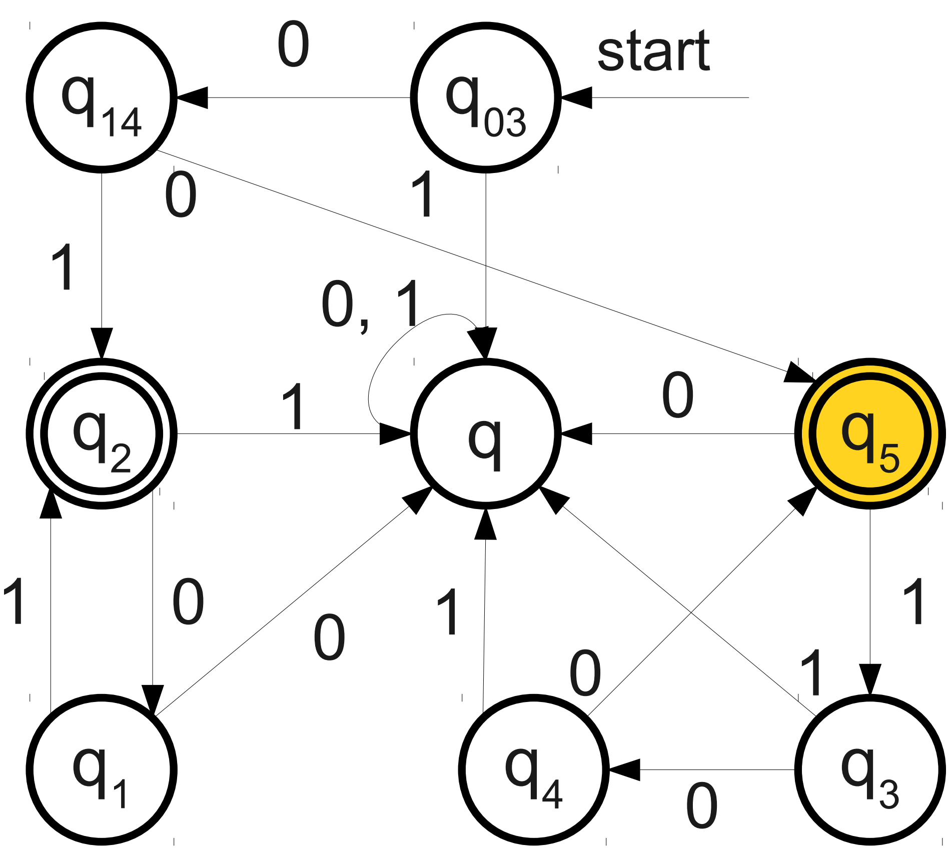




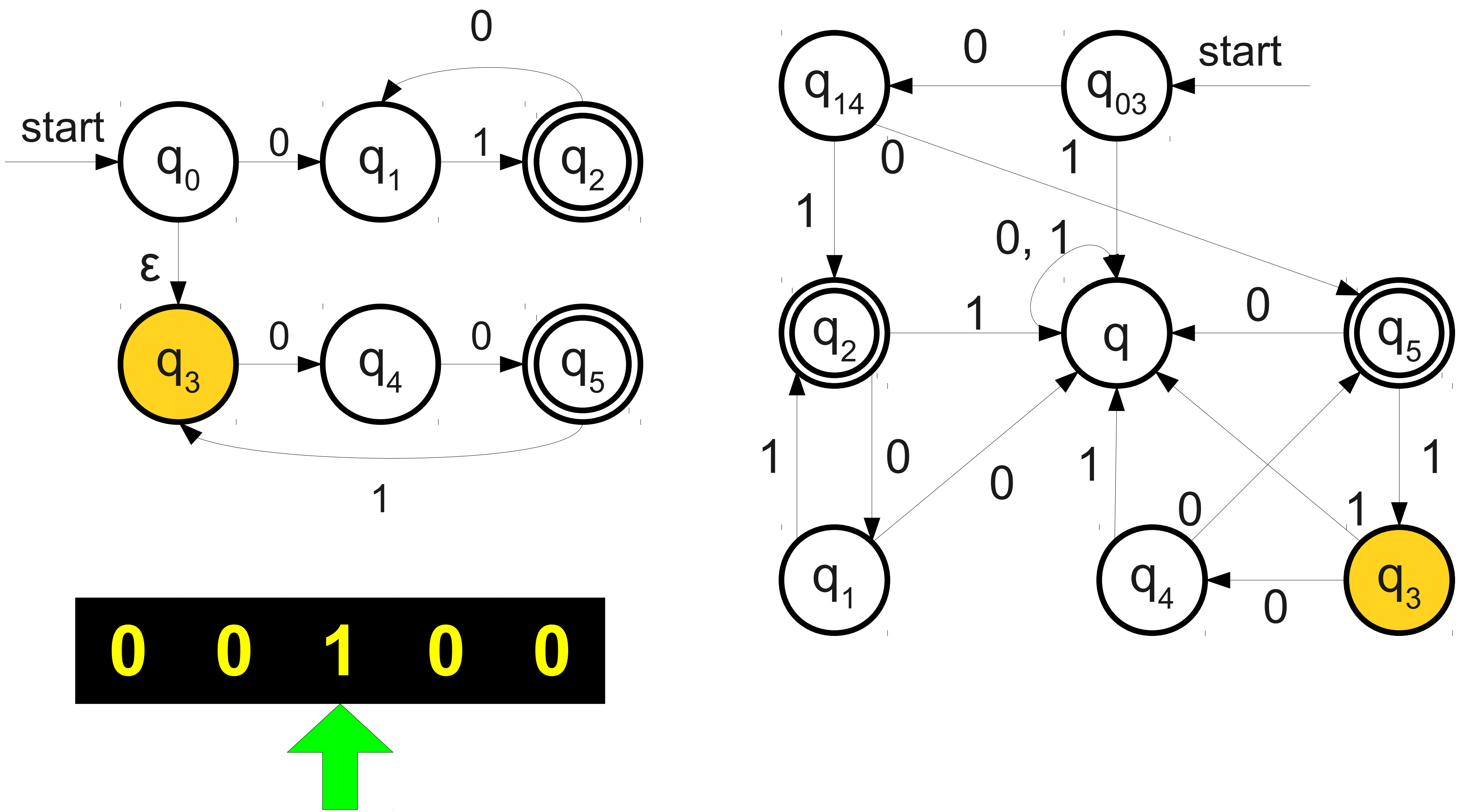


Simulating an NFA with a DFA

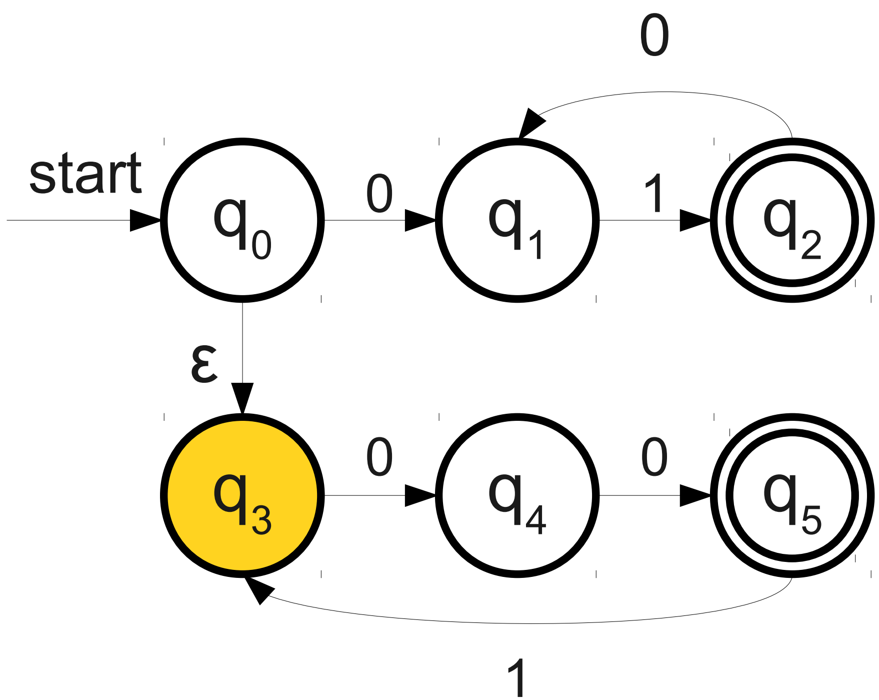


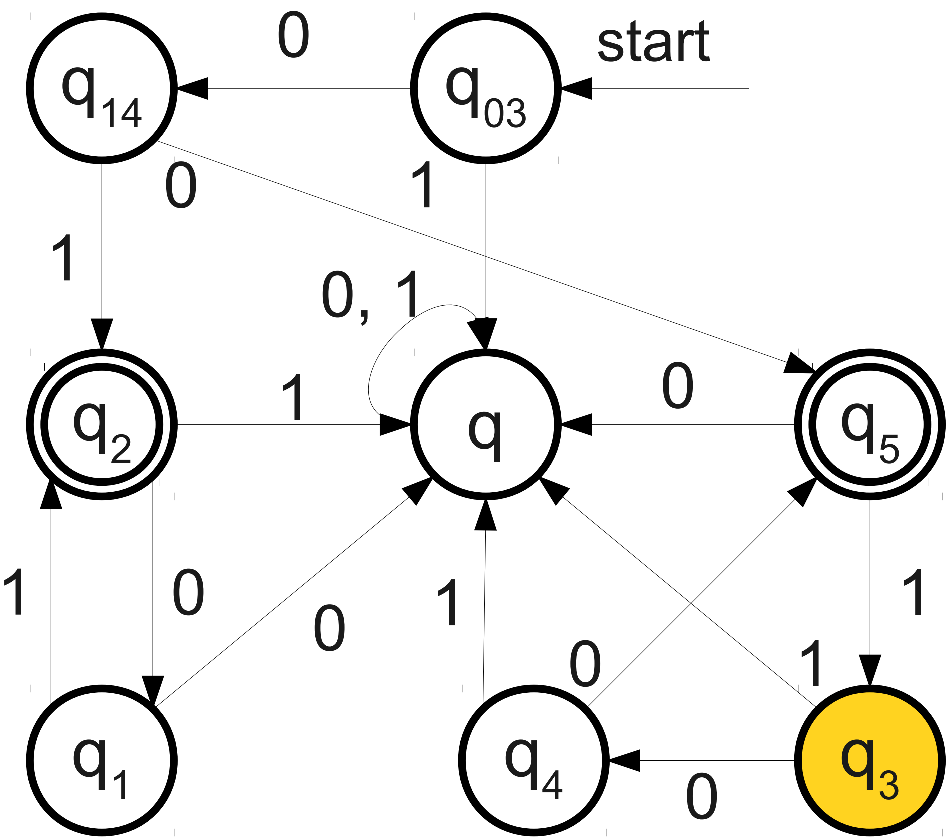


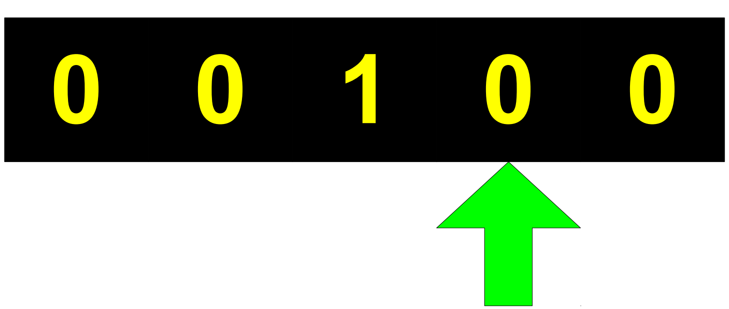
Simulating an NFA with a DFA



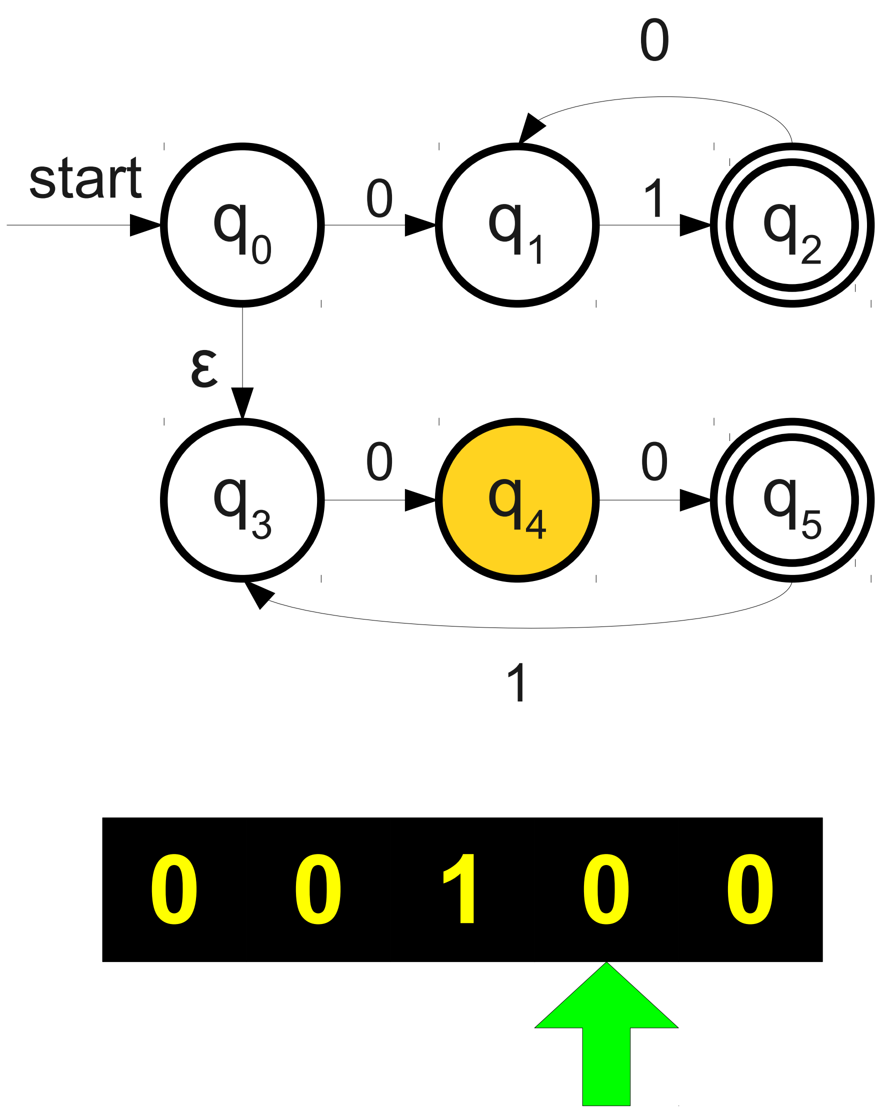
Simulating an NFA with a DFA

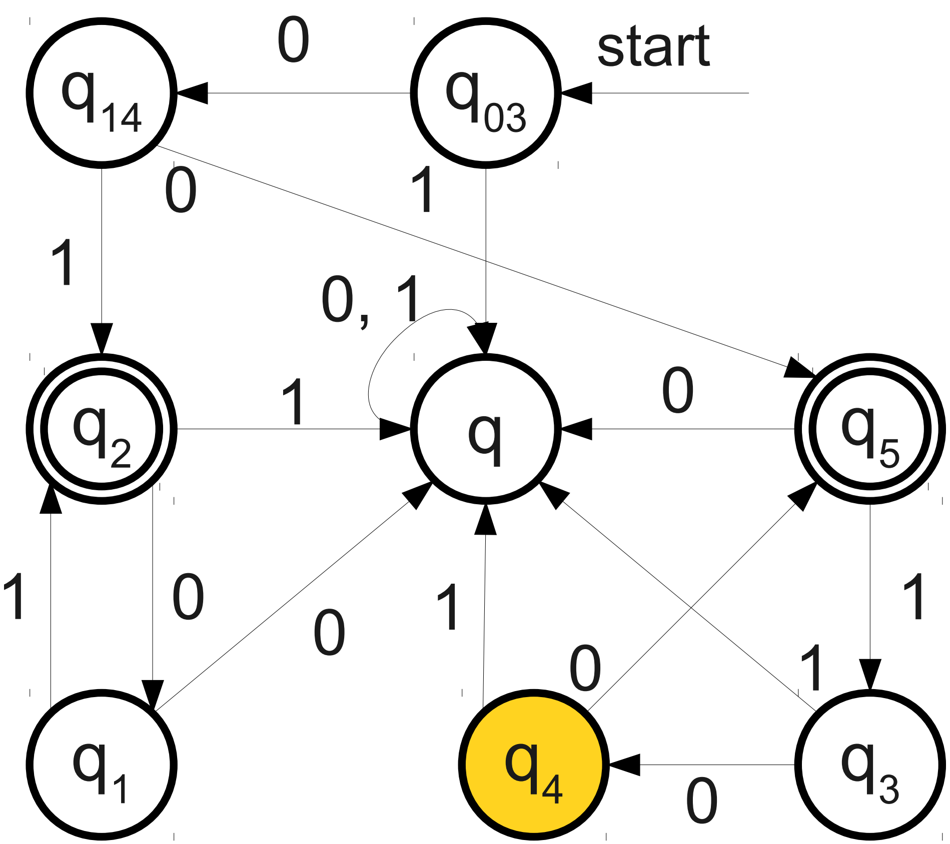




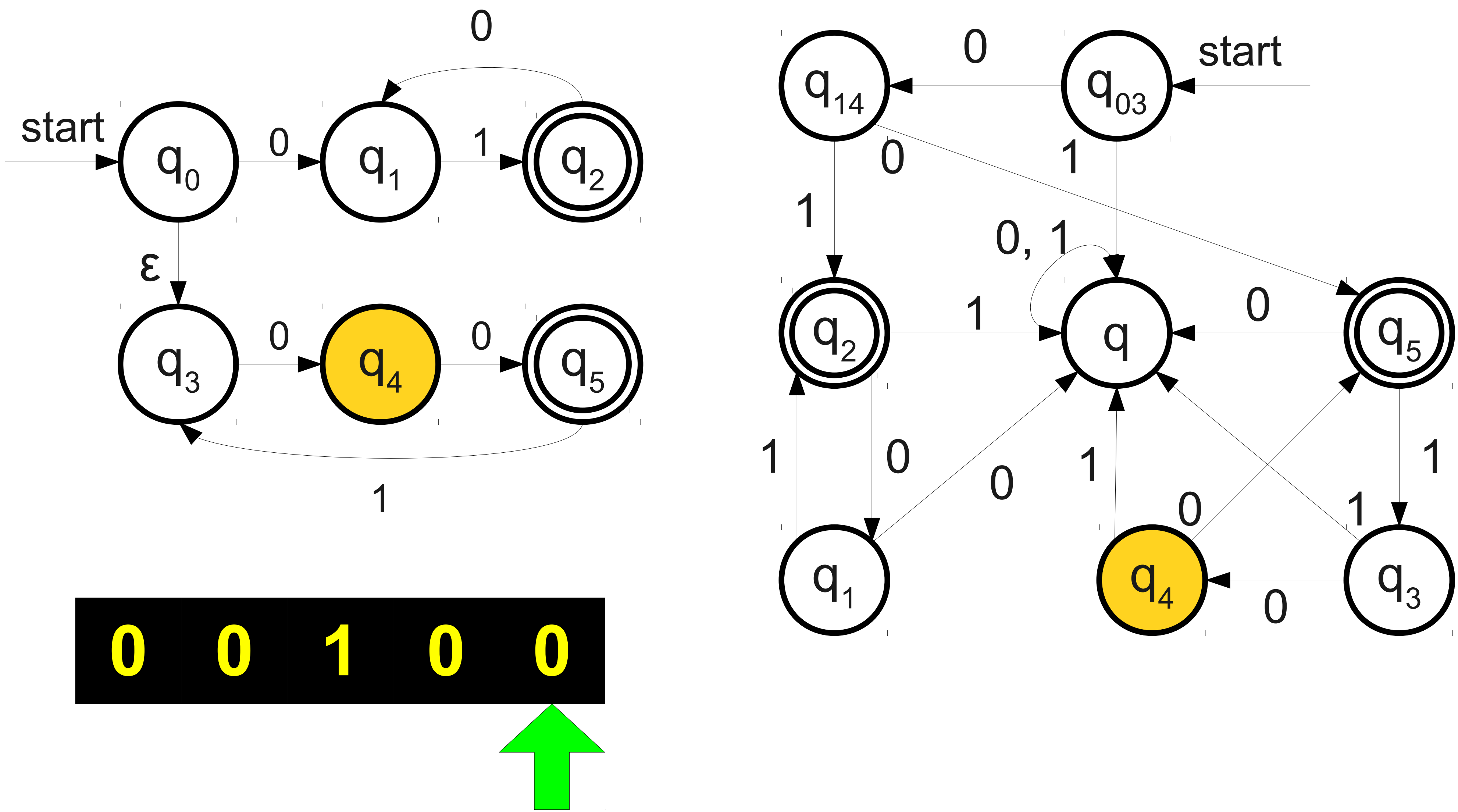


Simulating an NFA with a DFA

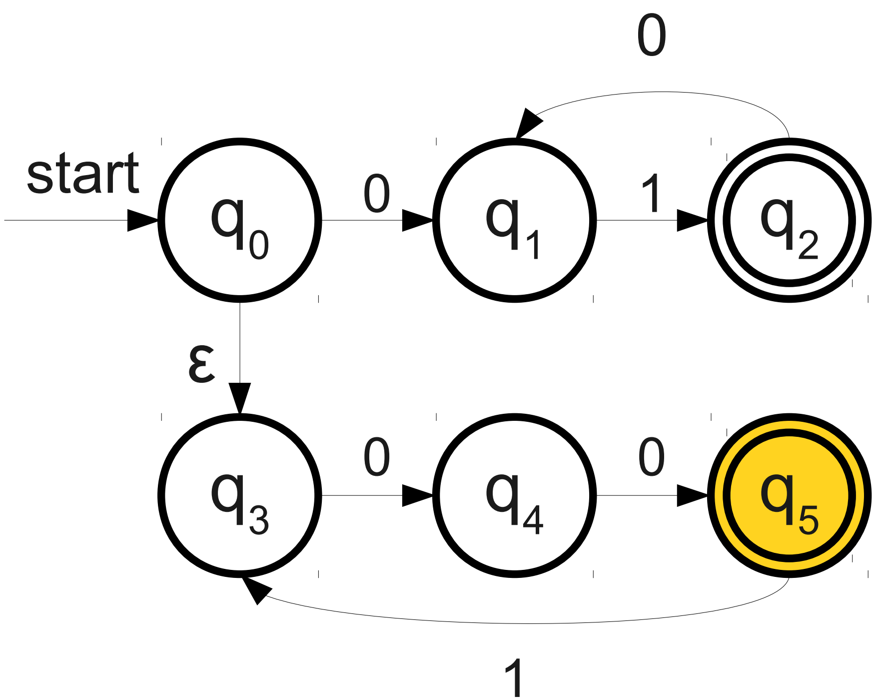


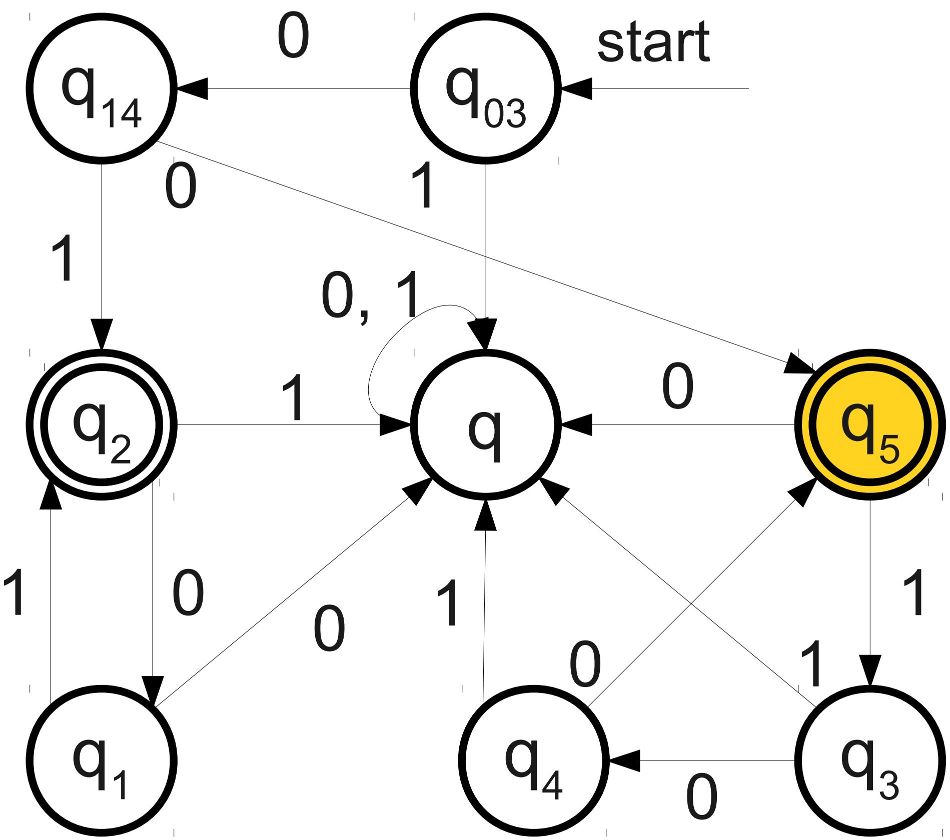


Simulating an NFA with a DFA



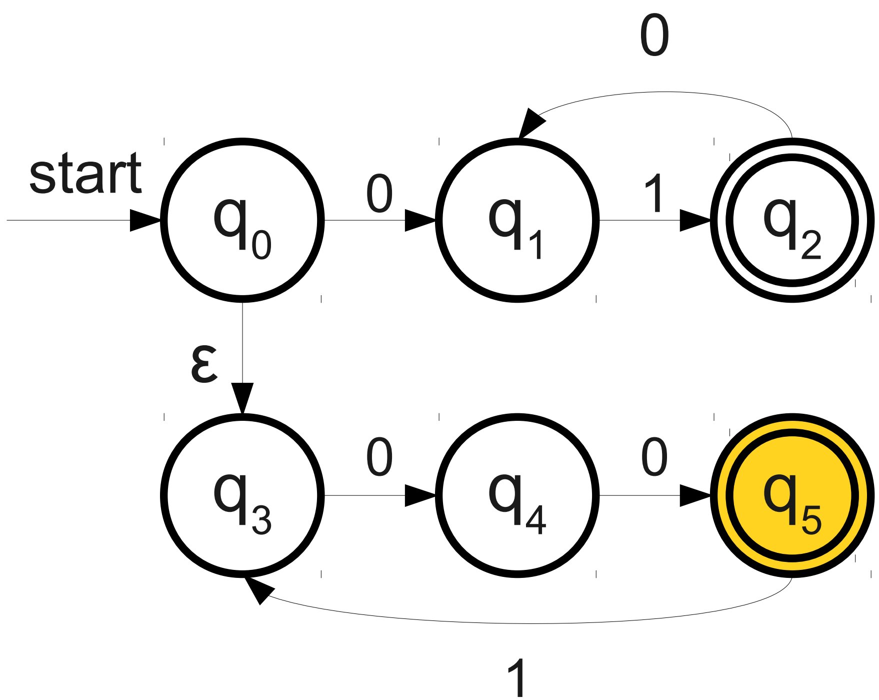
Simulating an NFA with a DFA

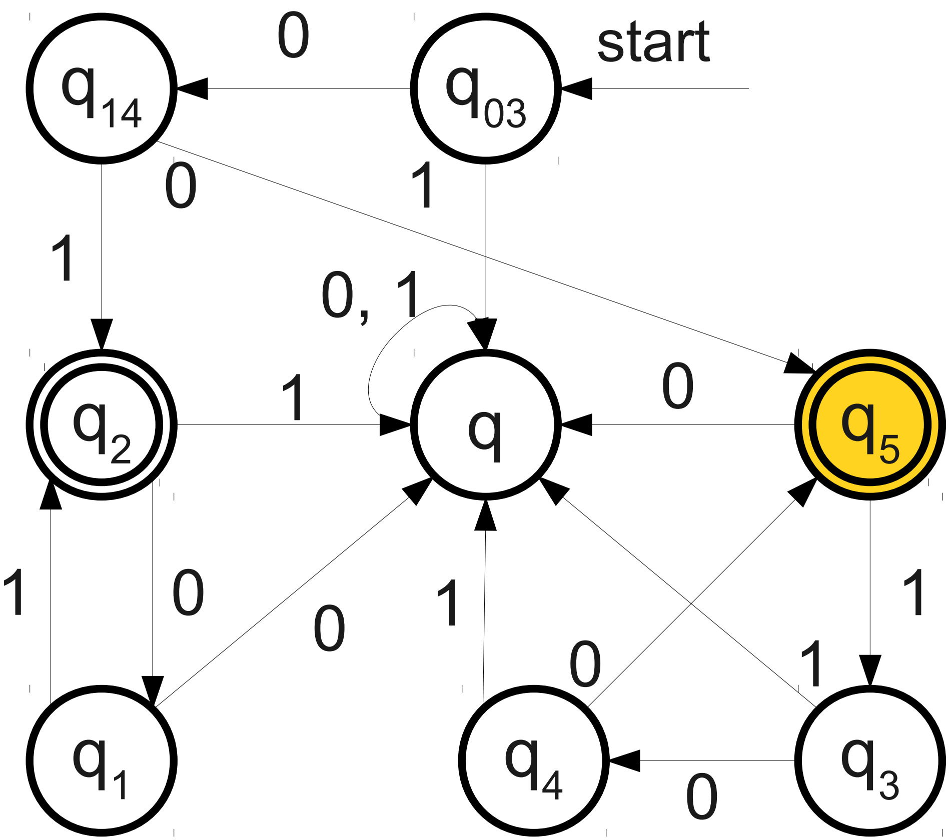




0 0 1 0 0

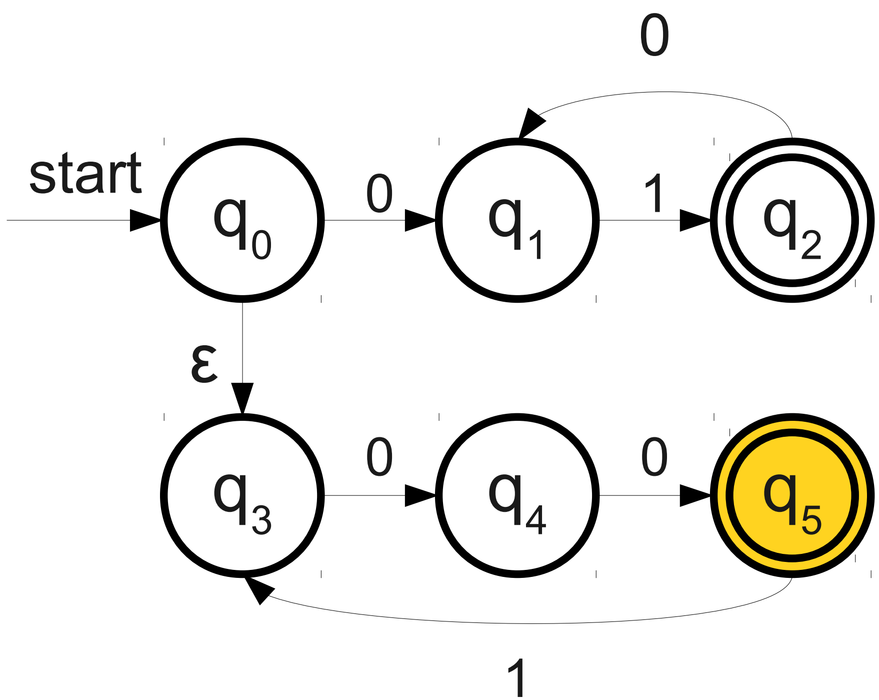
Simulating an NFA with a DFA



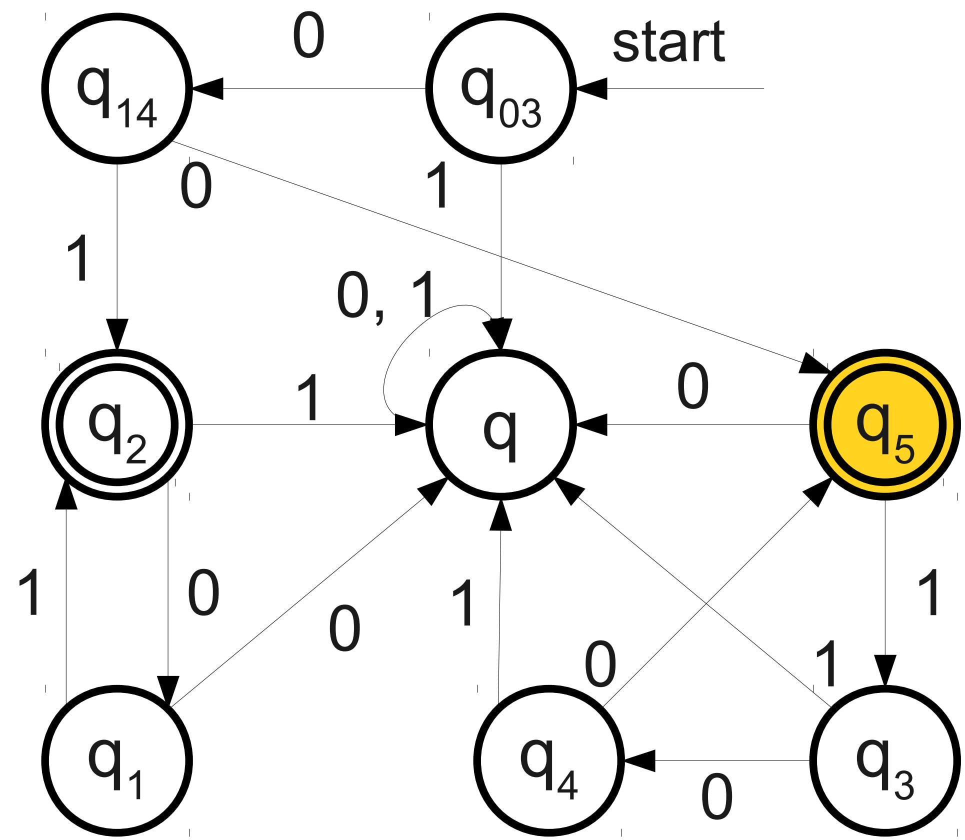


0 0 1 0 0

Simulating an NFA with a DFA



0 0 1 0 0



The Subset Construction

- This construction for transforming an NFA into a DFA is called the **subset construction** (or sometimes the **powerset construction**).
- Intuitively:
  - States of the new DFA correspond to *sets of states* of the NFA.
  - The initial state is the start state, plus all states reachable from the start state via ε-transitions.
  - Transition on state *S* on character **`a`** is found by following all possible transitions on **`a`** for each state in *S*, then taking the set of states reachable from there by ε-transitions.
  - Accepting states are any set of states where *some* state in the set is an accepting state.
- **Read Sipser for a formal account.**

The Subset Construction

- In converting an NFA to a DFA, the DFA's states correspond to sets of NFA states.
- Fact: $|\wp(S)| = 2^{|S|}$ for any finite set S.
- In the worst-case, the construction can result in a DFA that is *exponentially larger* than the original NFA.
- Interesting challenge: Find a language for which this worst-case behavior occurs (there are infinitely many of them!)

A language L is called a **regular language** iff there exists a DFA D such that $\mathscr{L}(D) = L$.

An Important Result

Theorem: A language L is regular iff there is some NFA N such that $\mathscr{L}(N) = L$.

An Important Result

Theorem: A language L is regular iff there is some NFA N such that $\mathscr{L}(N) = L$.

Proof Sketch:

An Important Result

Theorem: A language L is regular iff there is some NFA N such that $\mathscr{L}(N) = L$.

Proof Sketch: If L is regular, there exists some DFA for it, which we can easily convert into an NFA.

An Important Result

Theorem: A language L is regular iff there is some NFA N such that $\mathscr{L}(N) = L$.

Proof Sketch: If L is regular, there exists some DFA for it, which we can easily convert into an NFA. If L is accepted by some NFA, we can use the subset construction to convert it into a DFA that accepts the same language, so L is regular.

An Important Result

Theorem: A language L is regular iff there is some NFA N such that $\mathscr{L}(N) = L$.

Proof Sketch: If L is regular, there exists some DFA for it, which we can easily convert into an NFA. If L is accepted by some NFA, we can use the subset construction to convert it into a DFA that accepts the same language, so L is regular. ■

Why This Matters

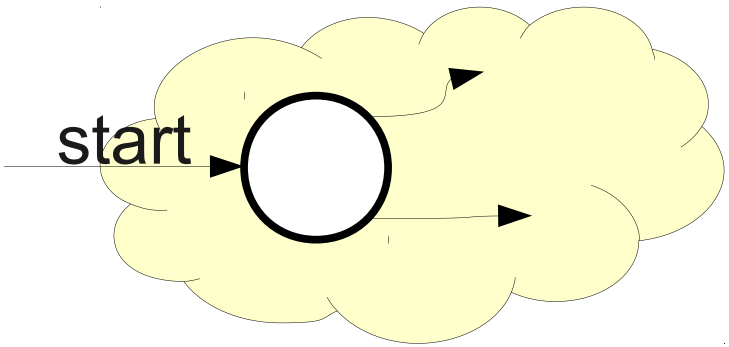
- We now have two perspectives on regular languages:
  - Regular languages are languages accepted by DFAs.
  - Regular languages are languages accepted by NFAs.
- We can now reason about the regular languages in two different ways.

The Union of Two Languages

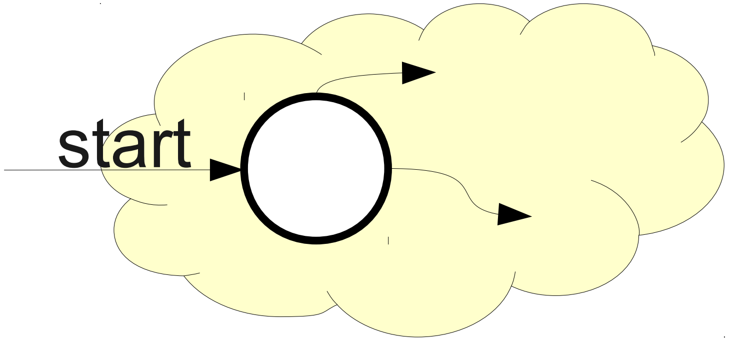
- If L_1 and L_2 are languages over the alphabet Σ, the language $L_1 \cup L_2$ is the language of all strings in at least one of the two languages.
- If L_1 and L_2 are regular languages, is $L_1 \cup L_2$?

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 $L_1 \cup L_2$?



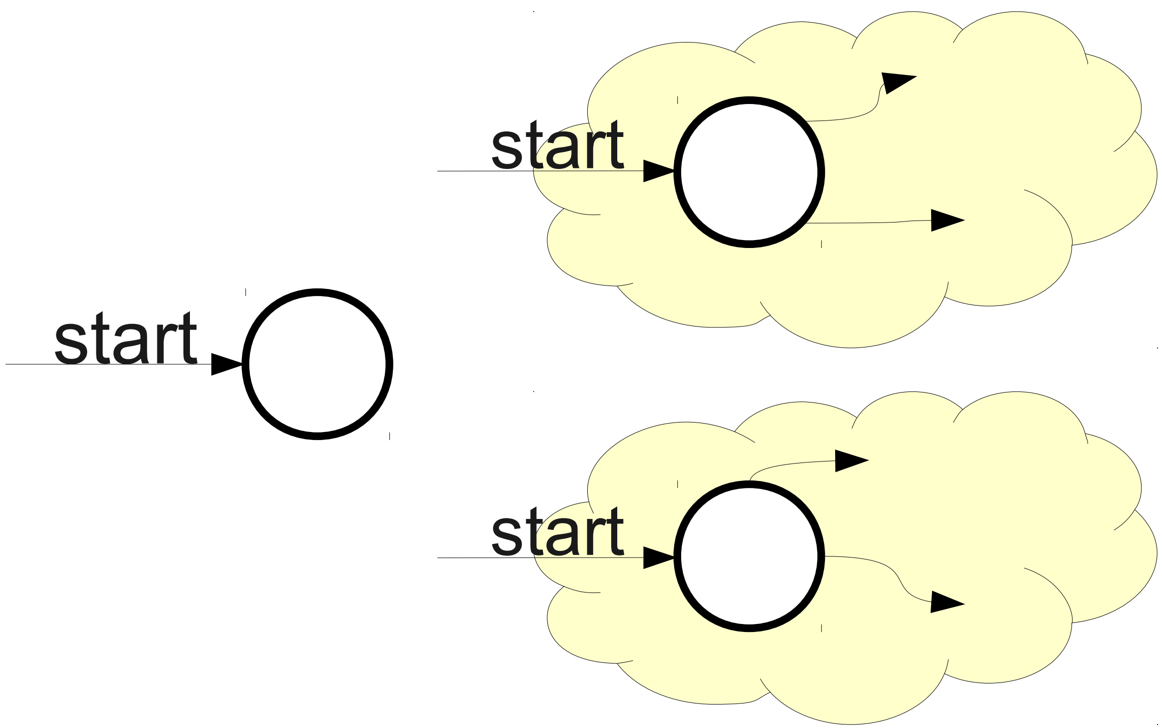
Machine for L_1



Machine for L_2

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 $L_1 \cup L_2$?

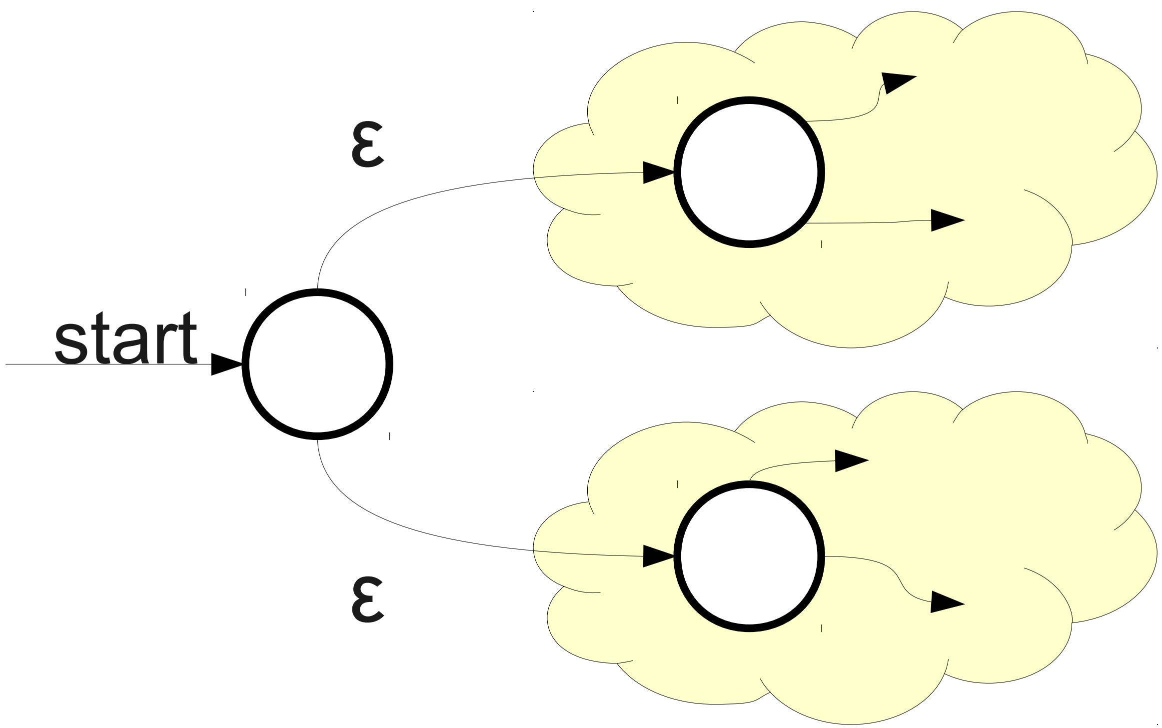


Machine for L_1

Machine for L_2

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 $L_1 \cup L_2$?

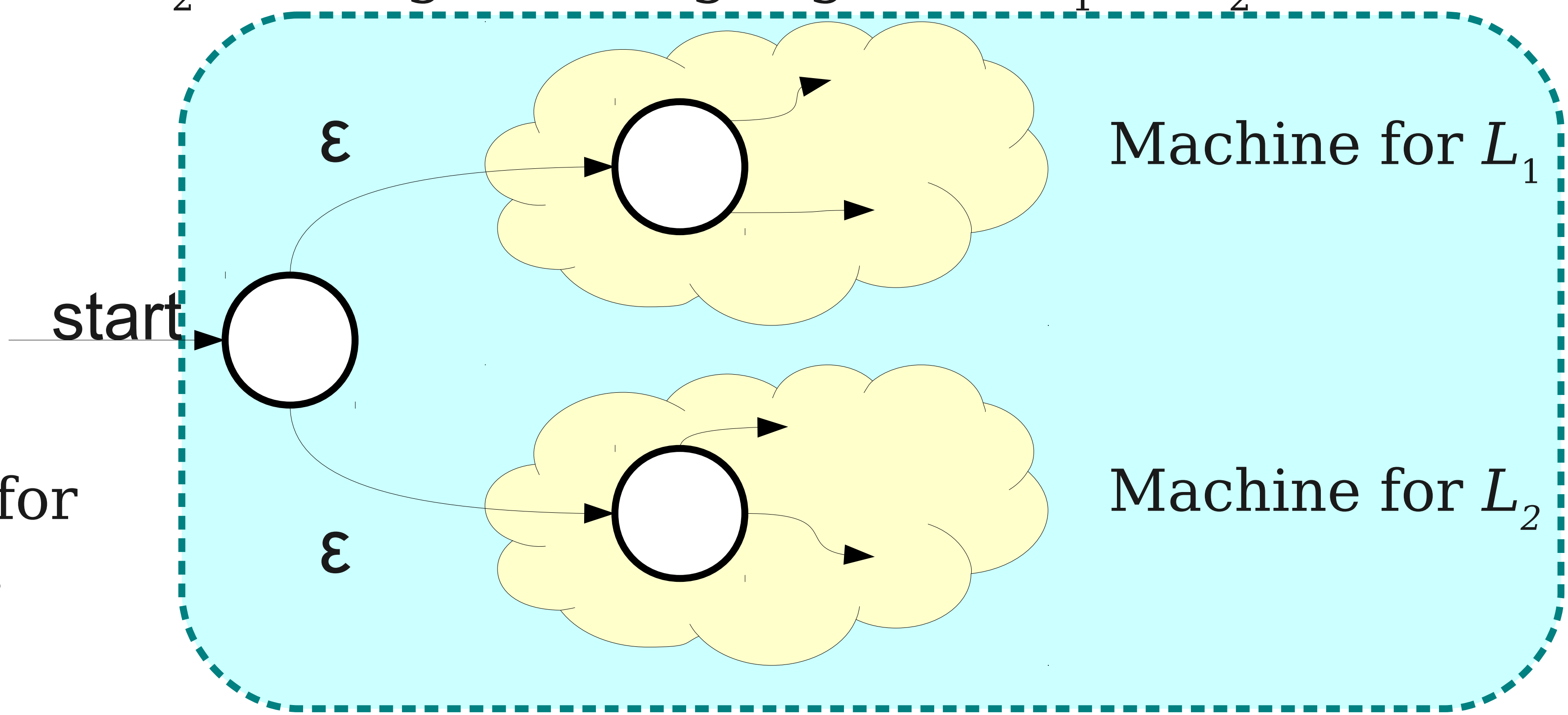


Machine for L_1

Machine for L_2

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 $L_1 \cup L_2$?



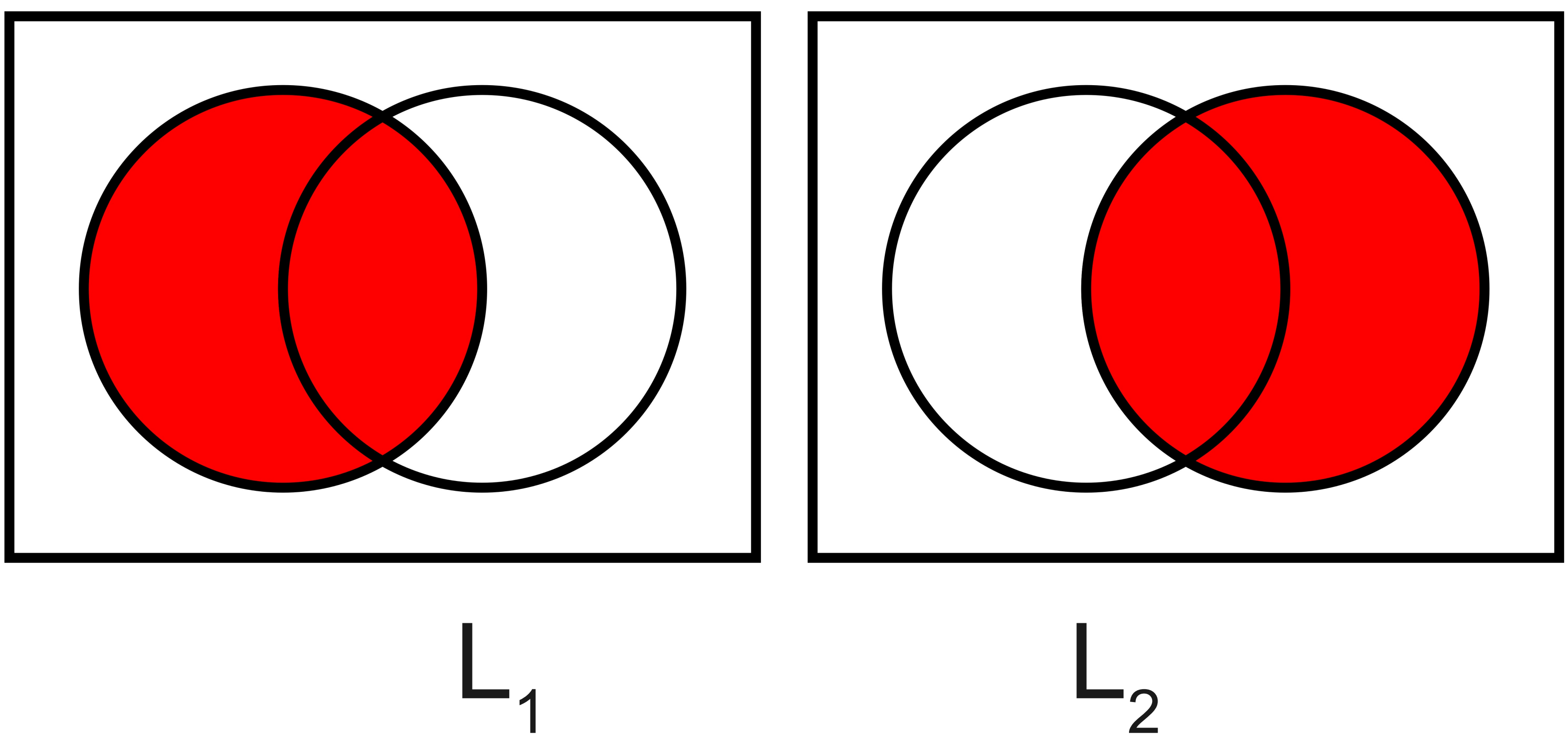
Machine for $L_1 \cup L_2$

The Intersection of Two Languages

- If L_1 and L_2 are languages over Σ, then $L_1 \cap L_2$ is the language of strings in both L_1 and L_2.
- Question: If L_1 and L_2 are regular, is $L_1 \cap L_2$ regular as well?

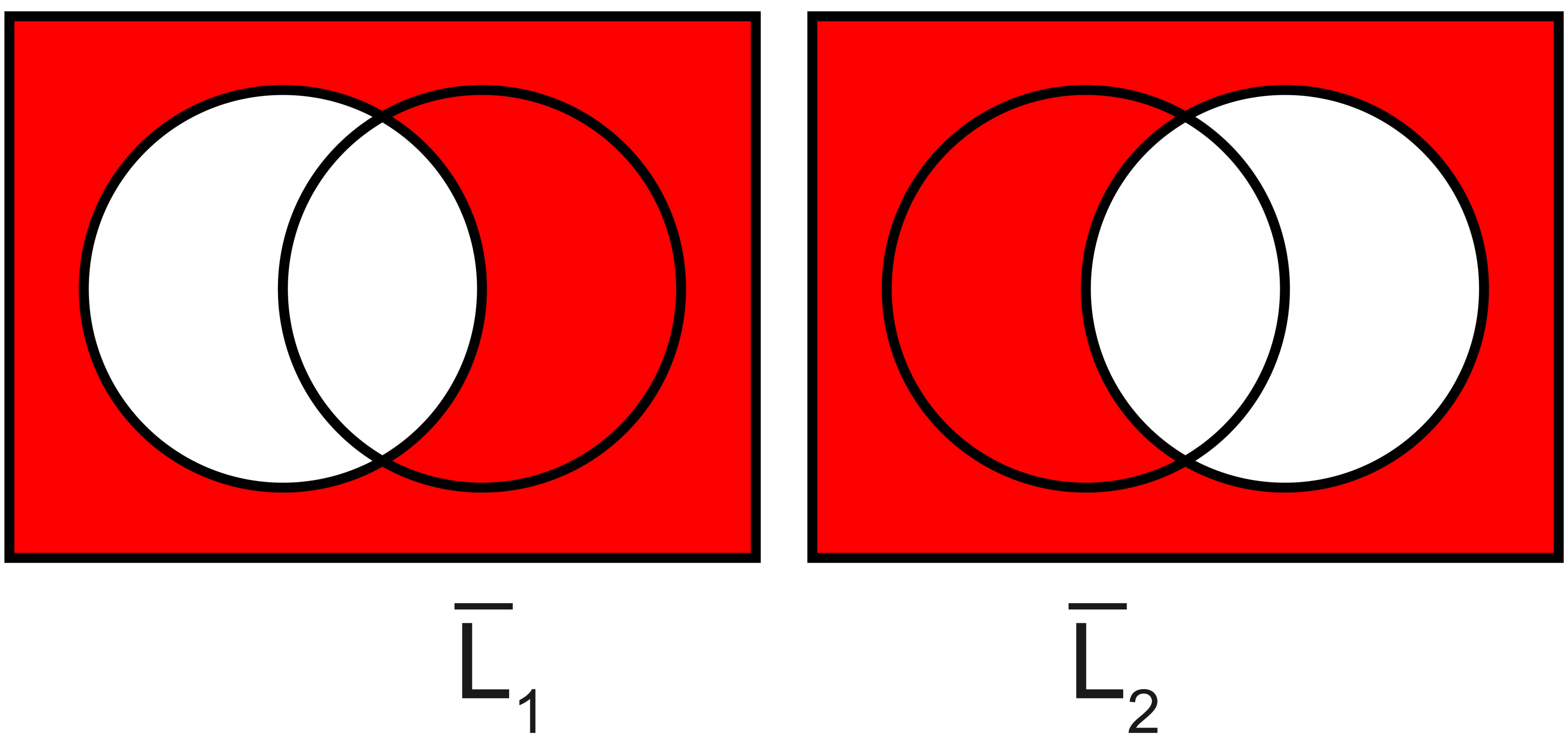
The Intersection of Two Languages

- If L_1 and L_2 are languages over Σ, then $L_1 \cap L_2$ is the language of strings in both L_1 and L_2.
- Question: If L_1 and L_2 are regular, is $L_1 \cap L_2$ regular as well?



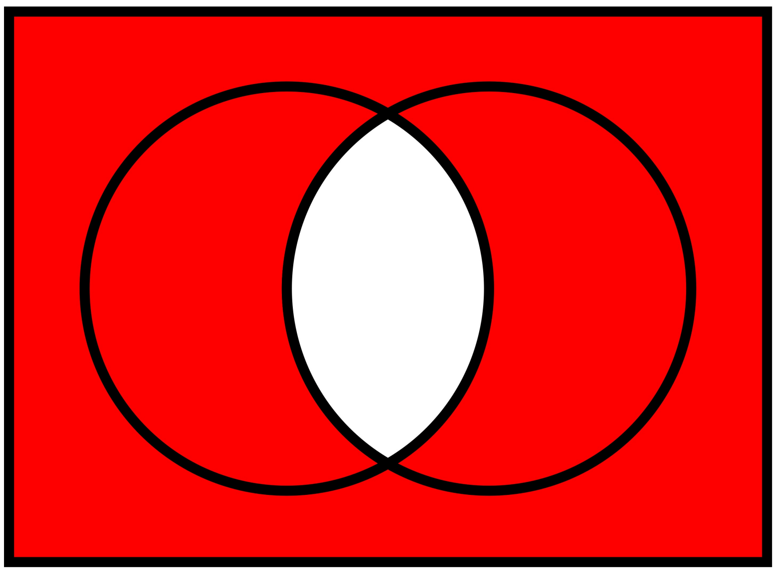
The Intersection of Two Languages

- If L_1 and L_2 are languages over Σ, then $L_1 \cap L_2$ is the language of strings in both L_1 and L_2.
- Question: If L_1 and L_2 are regular, is $L_1 \cap L_2$ regular as well?



The Intersection of Two Languages

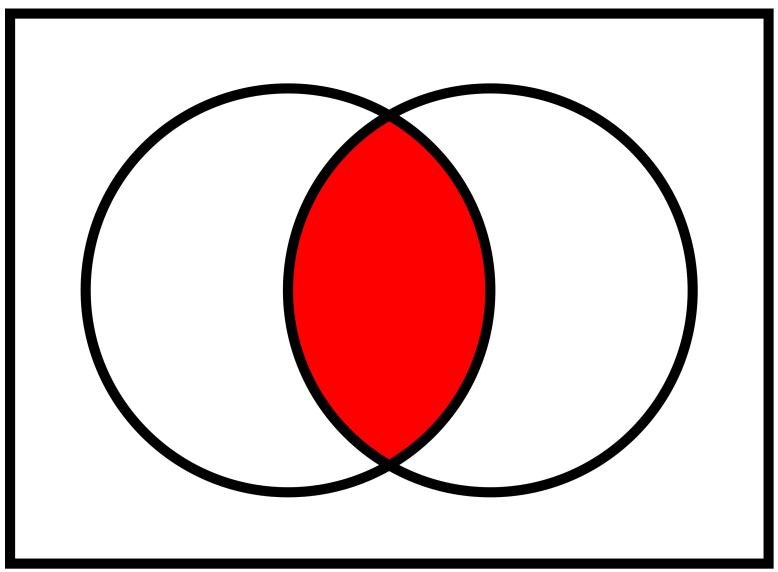
- If L_1 and L_2 are languages over Σ, then $L_1 \cap L_2$ is the language of strings in both L_1 and L_2.
- Question: If L_1 and L_2 are regular, is $L_1 \cap L_2$ regular as well?



$\overline{L}_1 \cup \overline{L}_2$

The Intersection of Two Languages

- If L_1 and L_2 are languages over Σ, then $L_1 \cap L_2$ is the language of strings in both L_1 and L_2.
- Question: If L_1 and L_2 are regular, is $L_1 \cap L_2$ regular as well?



$$\overline{\overline{L_1} \cup \overline{L_2}}$$

Hey, it's De Morgan's laws!

Concatenation

- The **concatenation** of two languages L_1 and L_2 over the alphabet Σ is the language

$$\boldsymbol{L_1 L_2 = \{\, wx \in \Sigma^* \mid w \in L_1 \land x \in L_2 \,\}}$$

- The set of strings that can be split into two pieces: a piece from L_1 and a piece from L_2.
- Conceptually similar to the Cartesian product of two sets, only with strings.

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

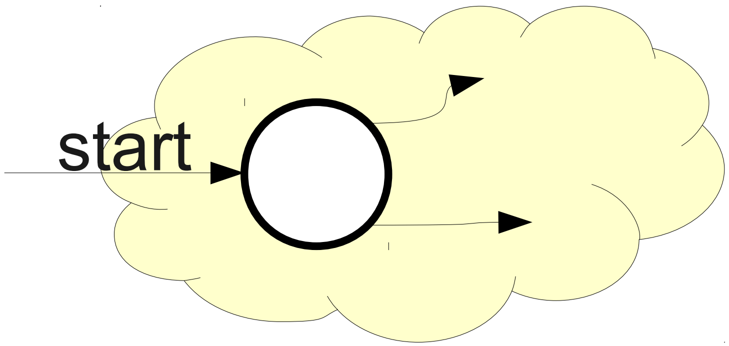
  { **ThePuppyHugsTheWhale**,
  TheWhaleLovesTheRainbow,
  TheRainbowJugglesTheRainbow, … }

Concatenating Regular Languages

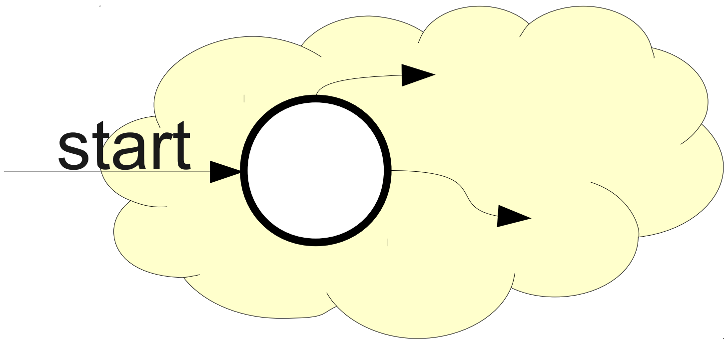
- If L_1 and L_2 are regular languages, is L_1L_2?
- Intuition – can we split a string w into two strings xy such that $x \in L_1$ and $y \in L_2$?

Concatenating Regular Languages

- If L_1 and L_2 are regular languages, is L_1L_2?
- Intuition – can we split a string w into two strings xy such that $x \in L_1$ and $y \in L_2$?



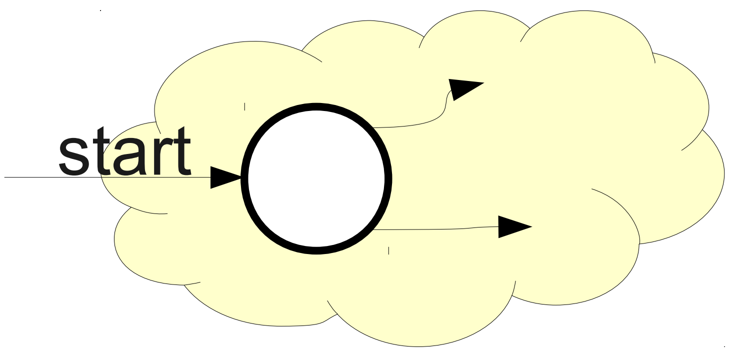
Machine for L_1



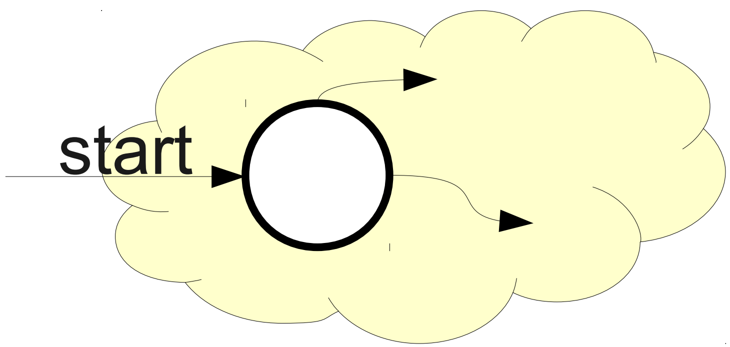
Machine for L_2

Concatenating Regular Languages

- If L_1 and L_2 are regular languages, is L_1L_2?
- Intuition – can we split a string w into two strings xy such that $x \in L_1$ and $y \in L_2$?



Machine for L_1

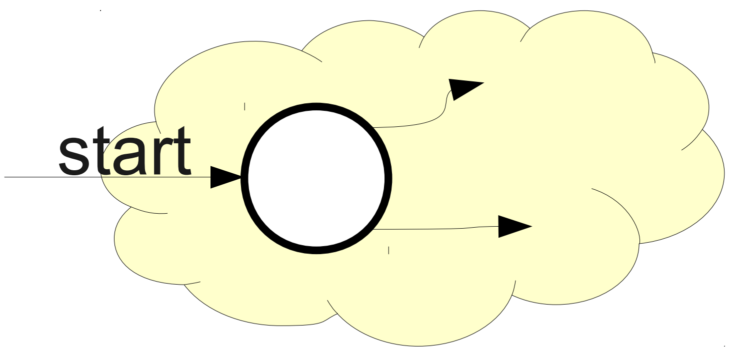


Machine for L_2

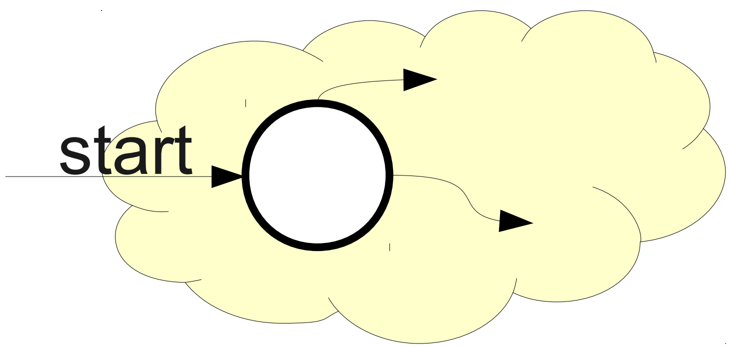
b	**o**	**o**	**k**	**k**	**e**	**e**	**p**	**e**	**r**

Concatenating Regular Languages

- If L_1 and L_2 are regular languages, is L_1L_2?
- Intuition – can we split a string w into two strings xy such that $x \in L_1$ and $y \in L_2$?



Machine for L_1

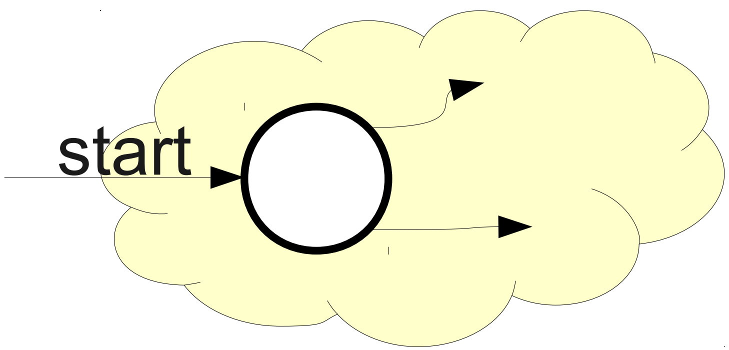


Machine for L_2

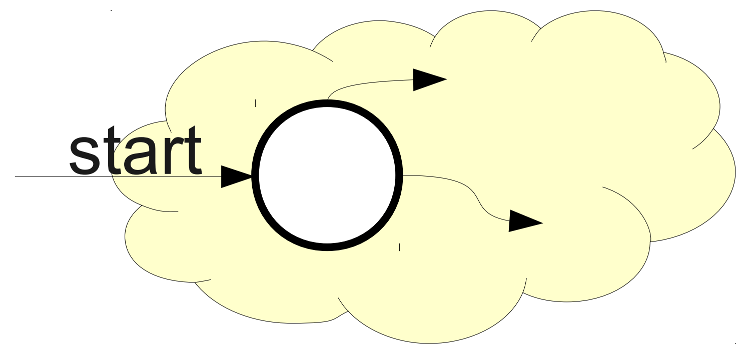
b	**o**	**o**	**k**	**k**	**e**	**e**	**p**	**e**	**r**

Concatenating Regular Languages

- If L_1 and L_2 are regular languages, is L_1L_2?
- Intuition – can we split a string w into two strings xy such that $x \in L_1$ and $y \in L_2$?



Machine for L_1



Machine for L_2

b	**o**	**o**	**k**

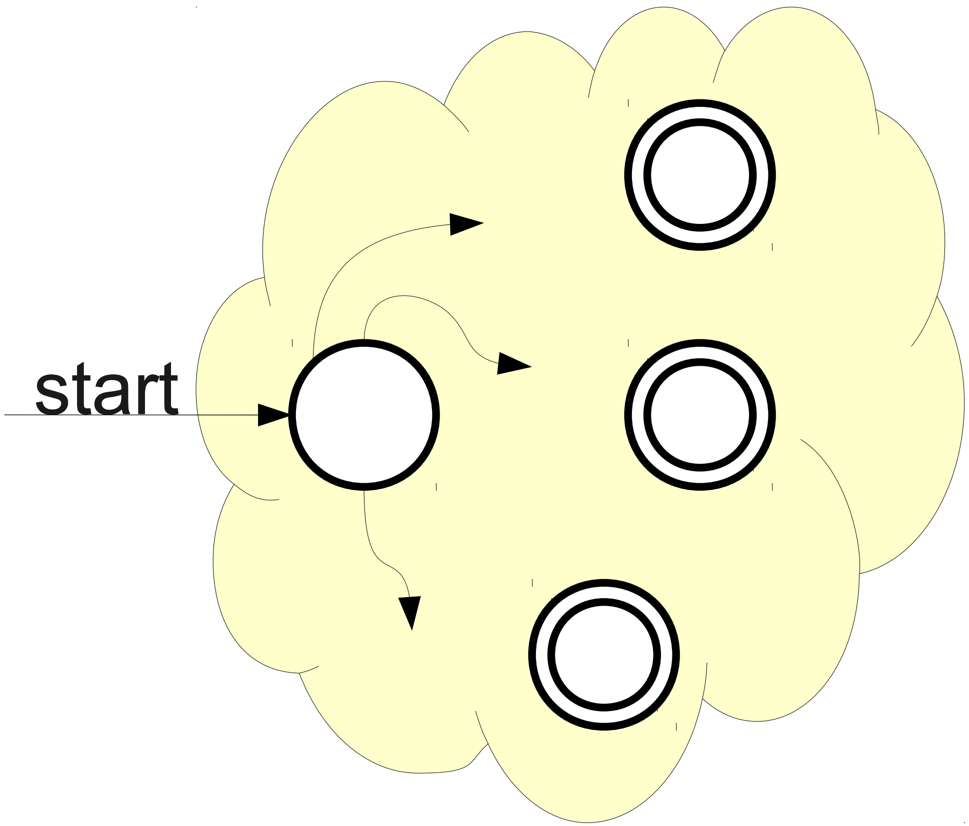
k	**e**	**e**	**p**	**e**	**r**

Concatenating Regular Languages

- If L_1 and L_2 are regular languages, is L_1L_2?
- Intuition – can we split a string w into two strings xy such that $x \in L_1$ and $y \in L_2$?
- **Idea**: Run the automaton for L_1 on w, and whenever L_1 reaches an accepting state, optionally hand the rest off w to L_2.
  - If L_2 accepts the remainder, then L_1 accepted the first part and the string is in L_1L_2.
  - If L_2 rejects the remainder, then the split was incorrect.

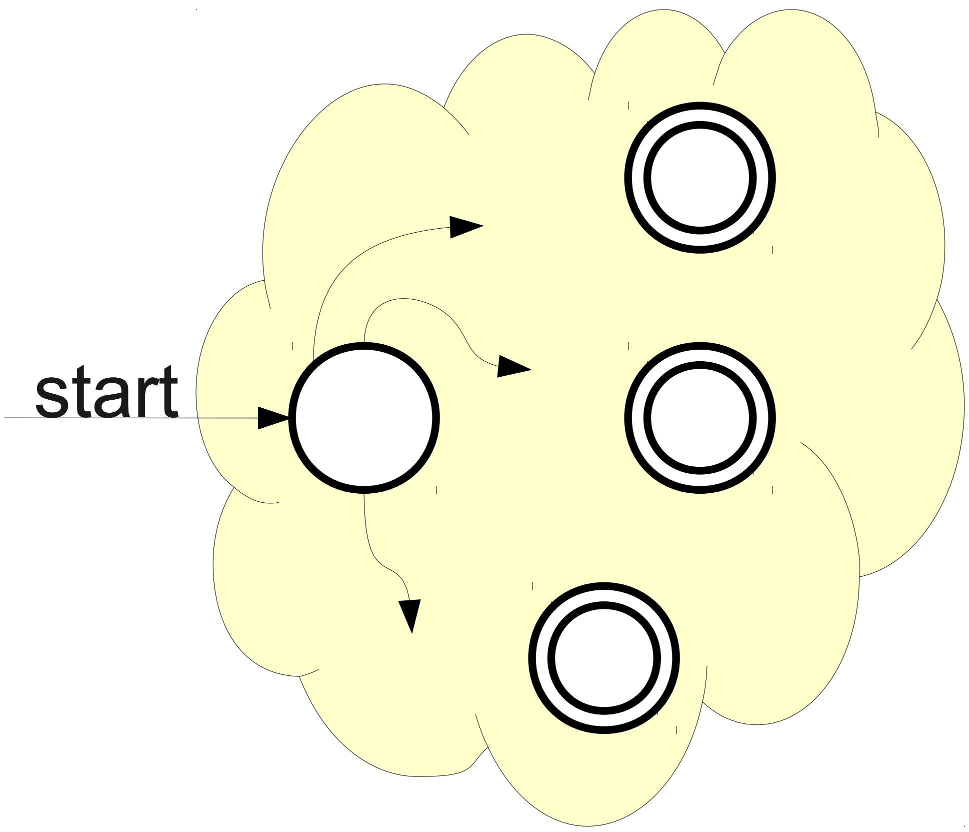
Concatenating Regular Languages

Concatenating Regular Languages

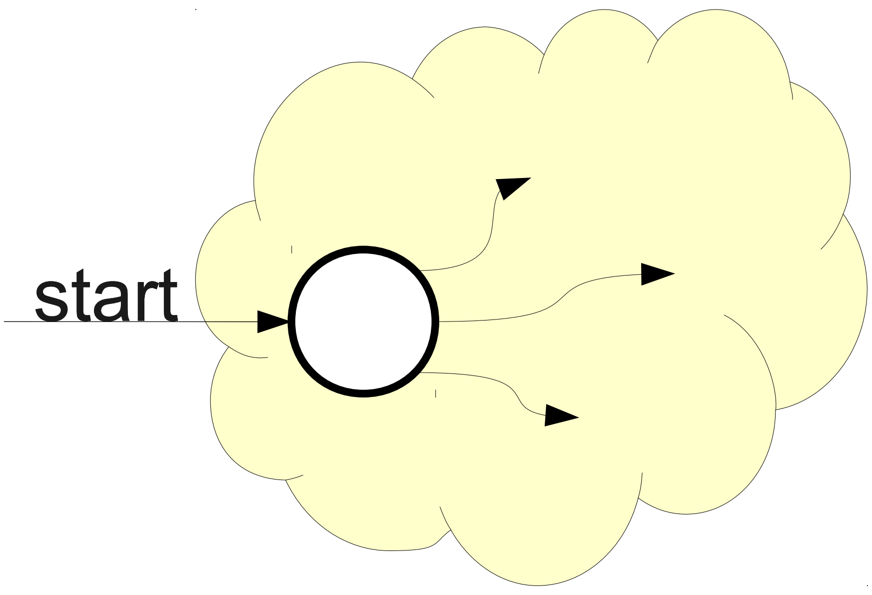


Machine for
L_1

Concatenating Regular Languages

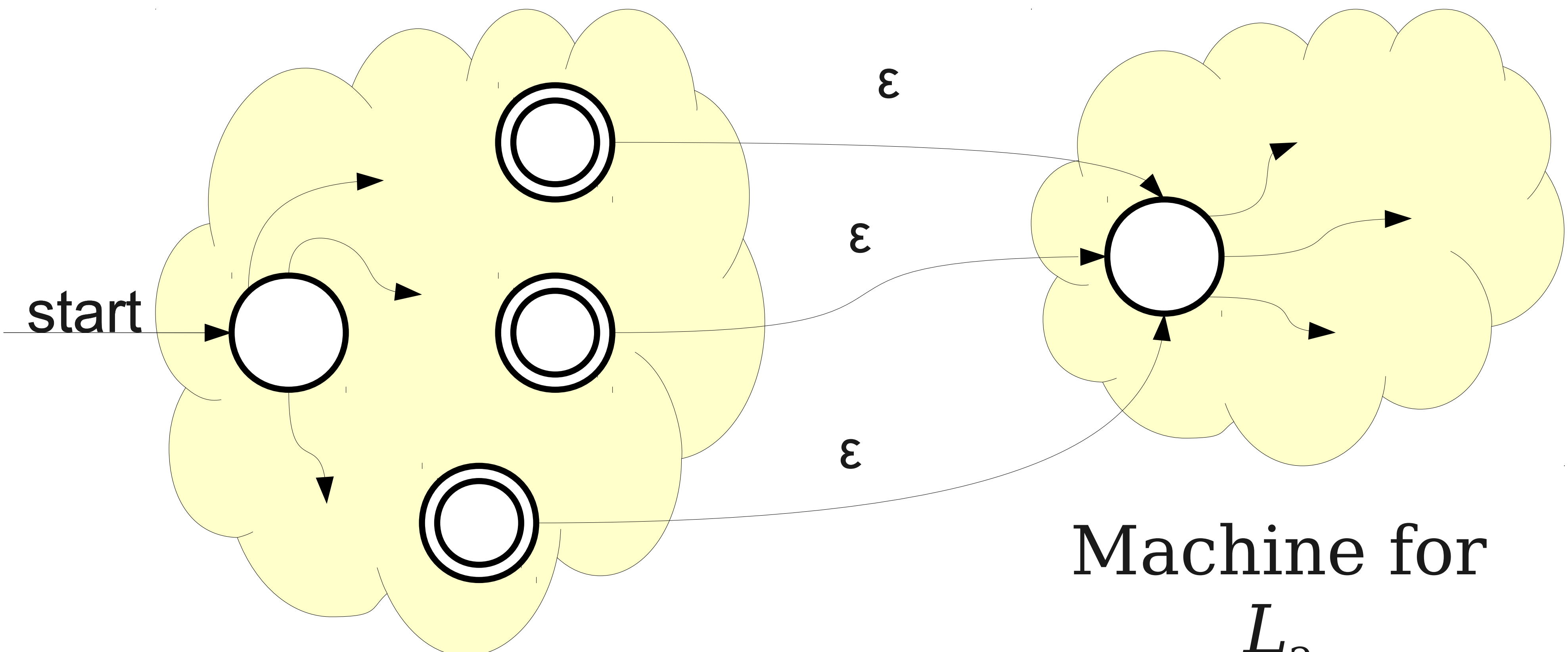


Machine for L_1



Machine for L_2

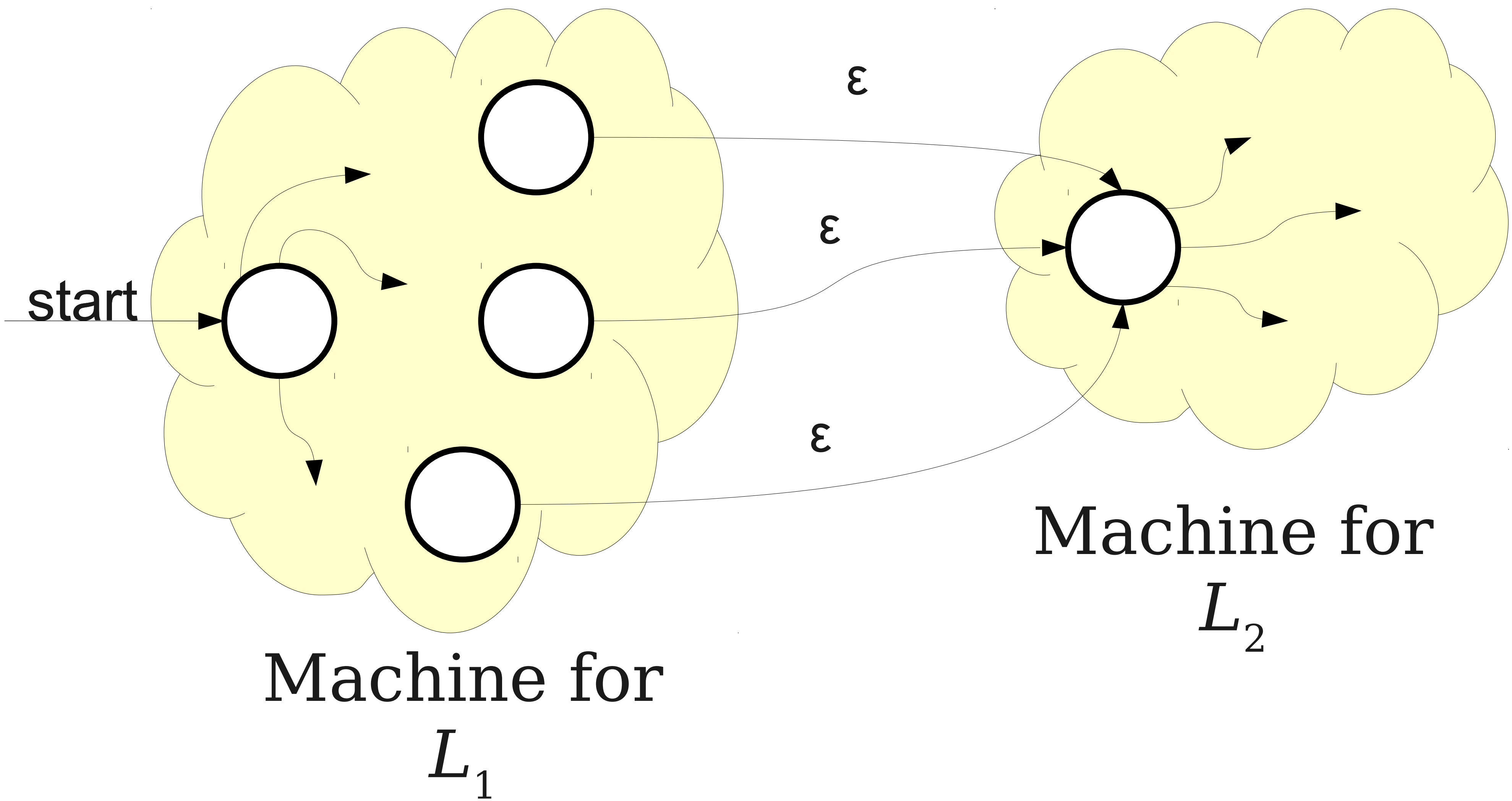
Concatenating Regular Languages



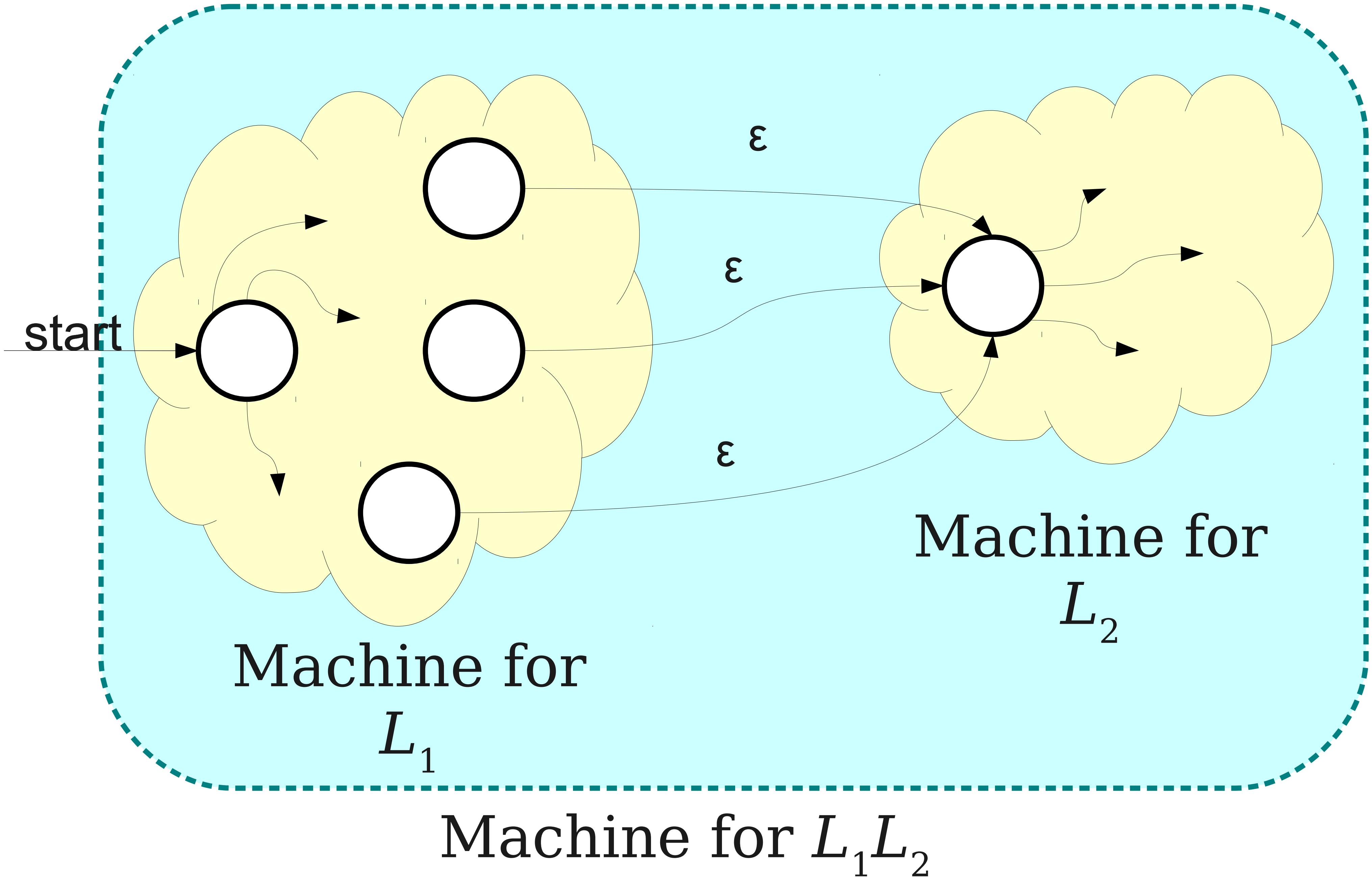
Machine for L_1

Machine for L_2

Concatenating Regular Languages



Concatenating Regular Languages



Lots and Lots of Concatenation

- Consider the language L = { `aa`, `b` }
- LL is the set of strings formed by concatenating pairs of strings in L.

{ `aaaa`, `aab`, `baa`, `bb` }

- LLL is the set of strings formed by concatenating triples of strings in L.

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

- $LLLL$ is the set of strings formed by concatenating quadruples of strings in L.

{ `aaaaaaaa`, `aaaaaab`, `aaaabaa`, `aaaabb`, `aabaaaa`, `aabaab`, `aabbaa`, `aabbb`, `baaaaaa`, `baaaab`, `baabaa`, `baabb`, `bbaaaa`, `bbaab`, `bbbaa`, `bbbb`}

Language Exponentiation

- We can define what it means to "exponentiate" a language as follows:
- $L^0 = \{ \varepsilon \}$
  - The set containing just the empty string.
  - Idea: Any string formed by concatenating zero strings together is the empty string.
- $L^{n+1} = LL^n$
  - Idea: Concatenating $(n + 1)$ strings together works by concatenating n strings, then concatenating one more.

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 Closure

If L = { **a**, **bb** }, then L^* = {

ε,

a, **bb**,

aa, **abb**, **bba**, **bbbb**,

aaa, **aabb**, **abba**, **abbbb**, **bbaa**, **bbabb**, **bbbba**, **bbbbbb**,

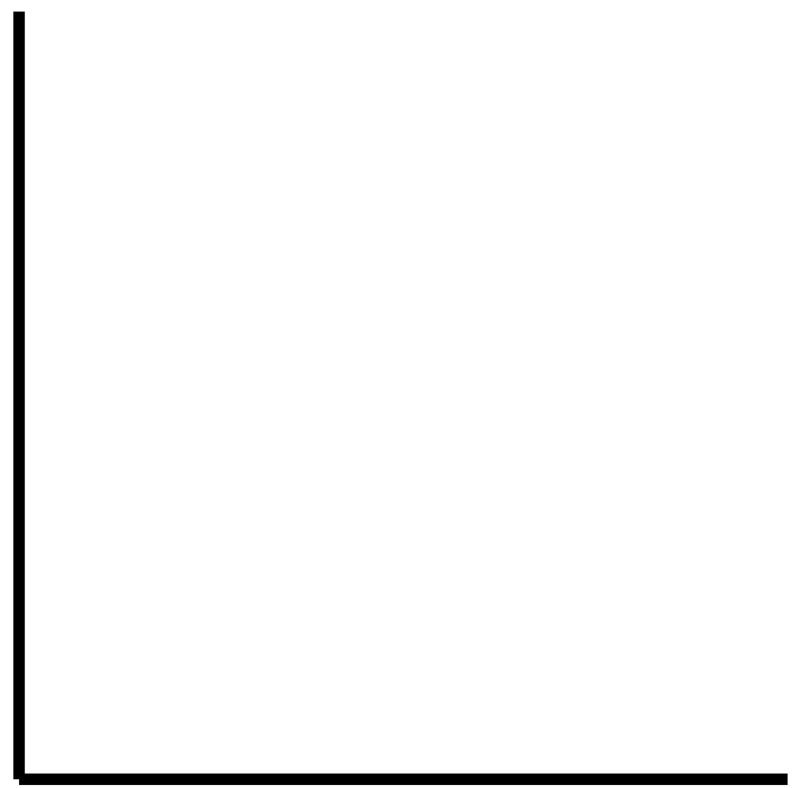
…

}

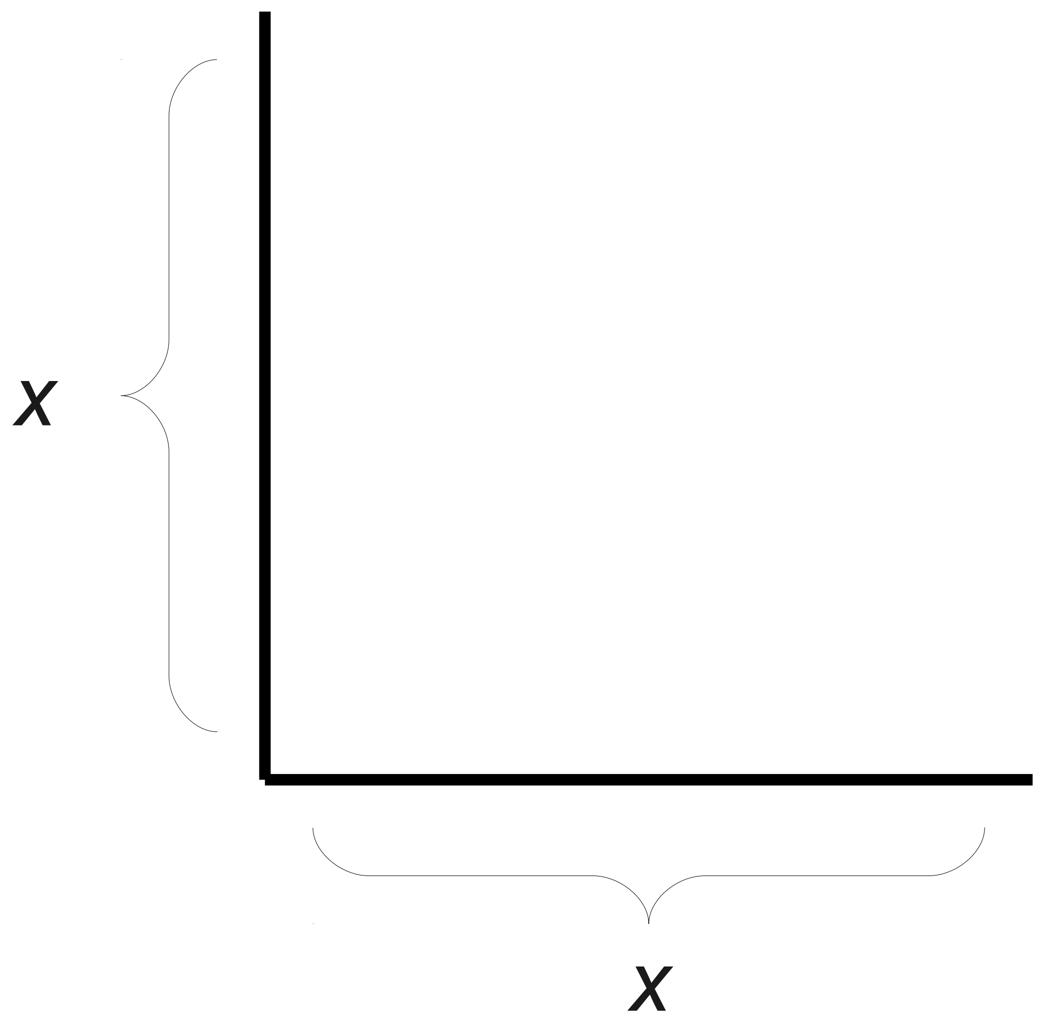
Reasoning about Infinity

- If L is regular, is L^* necessarily regular?
- **A Bad Line of Reasoning**:
  - $L^0 = \{ \varepsilon \}$ is regular.
  - $L^1 = L$ is regular.
  - $L^2 = LL$ is regular
  - $L^3 = L(LL)$ is regular
  - …
  - Regular languages are closed under union.
  - So the union of all these languages is regular.

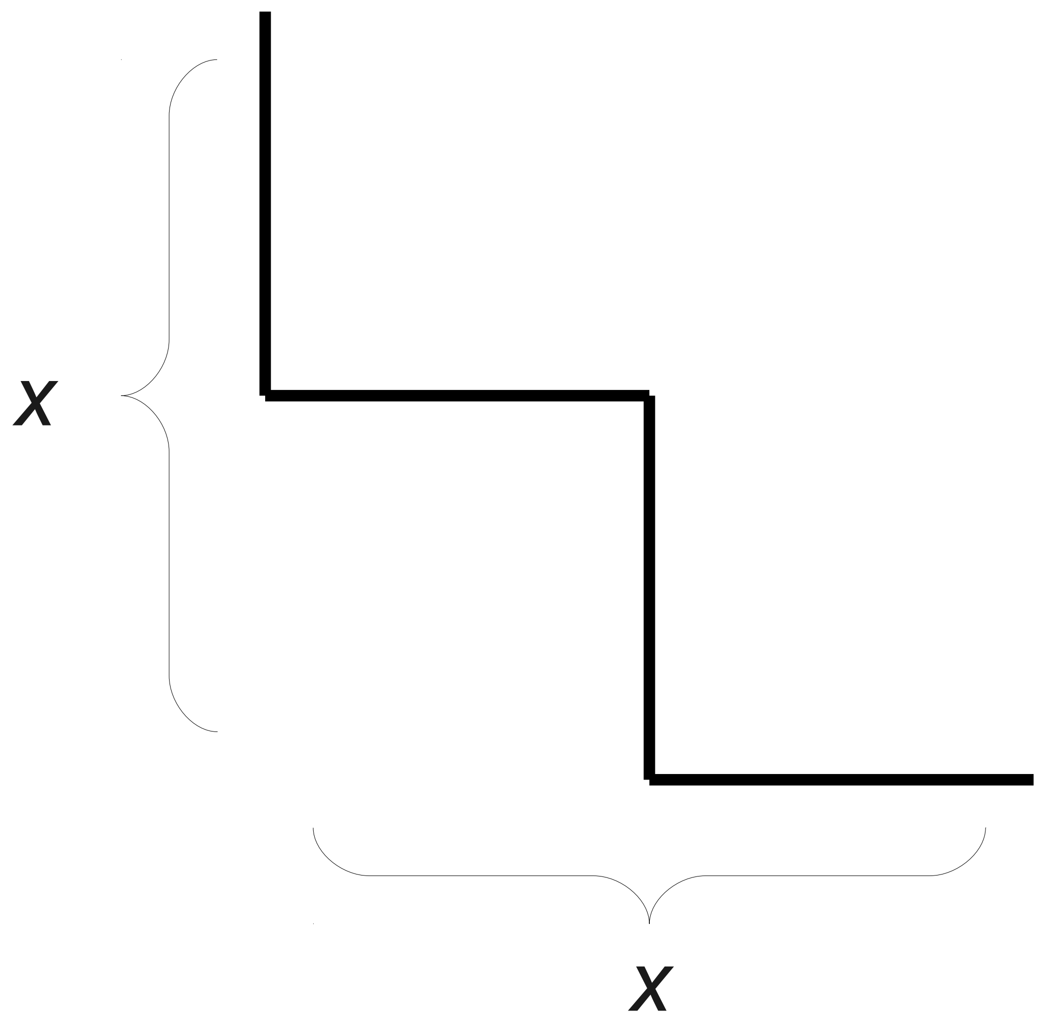
Reasoning about Infinity



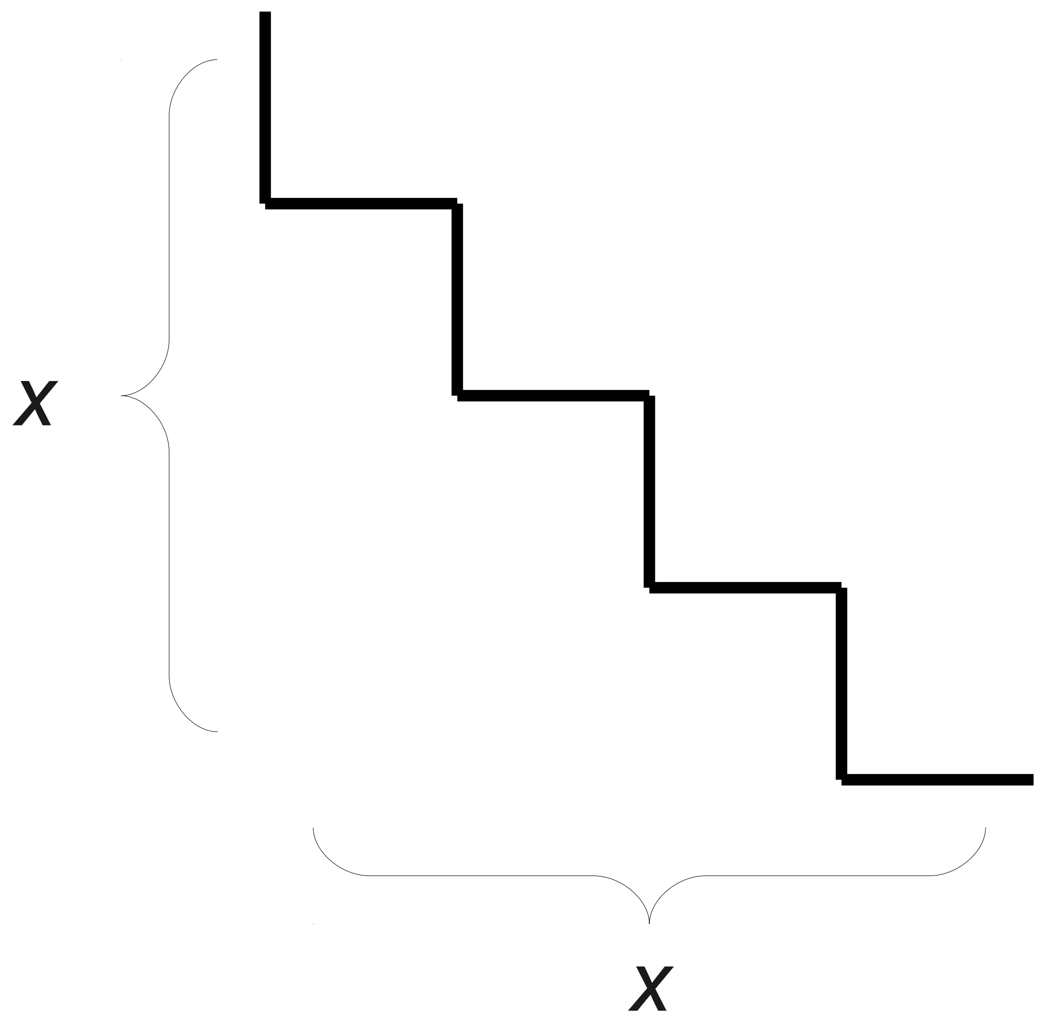
Reasoning about Infinity



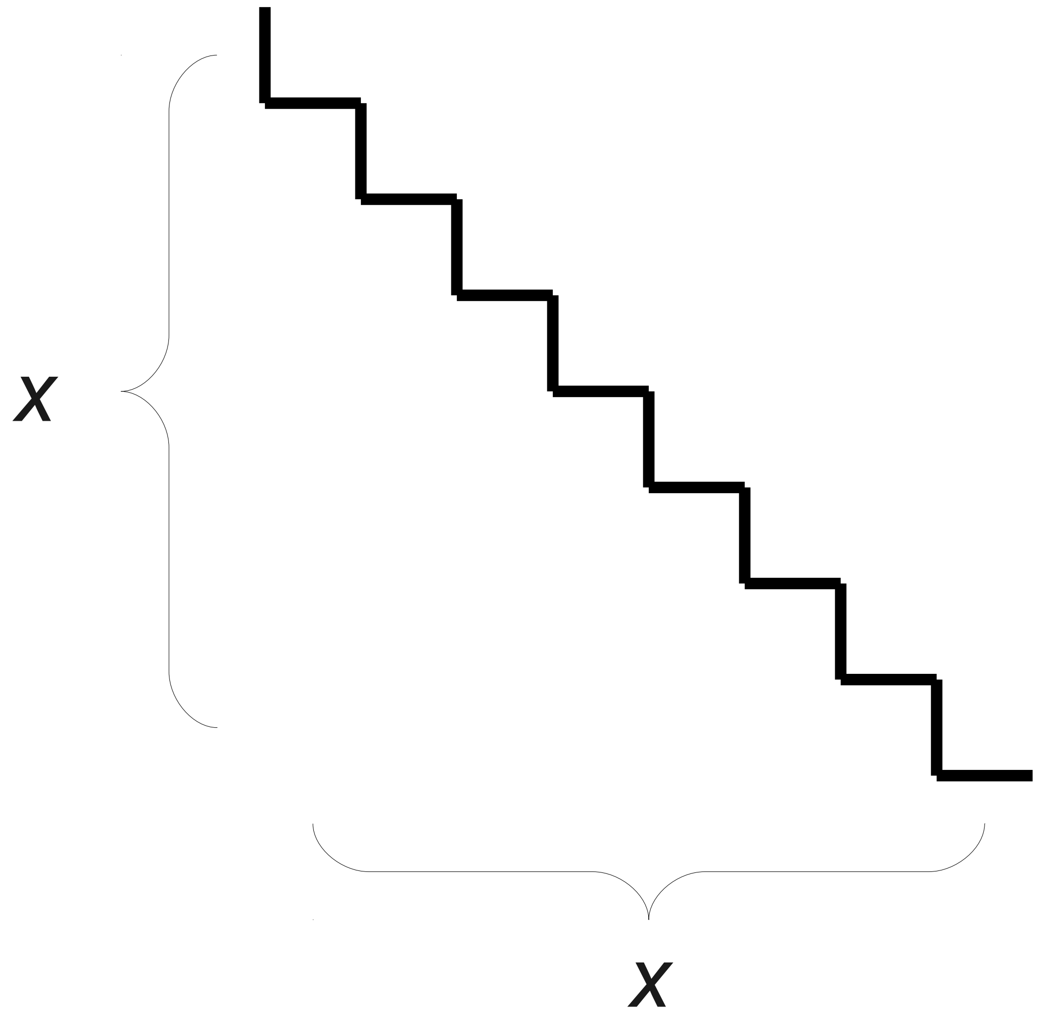
Reasoning about Infinity



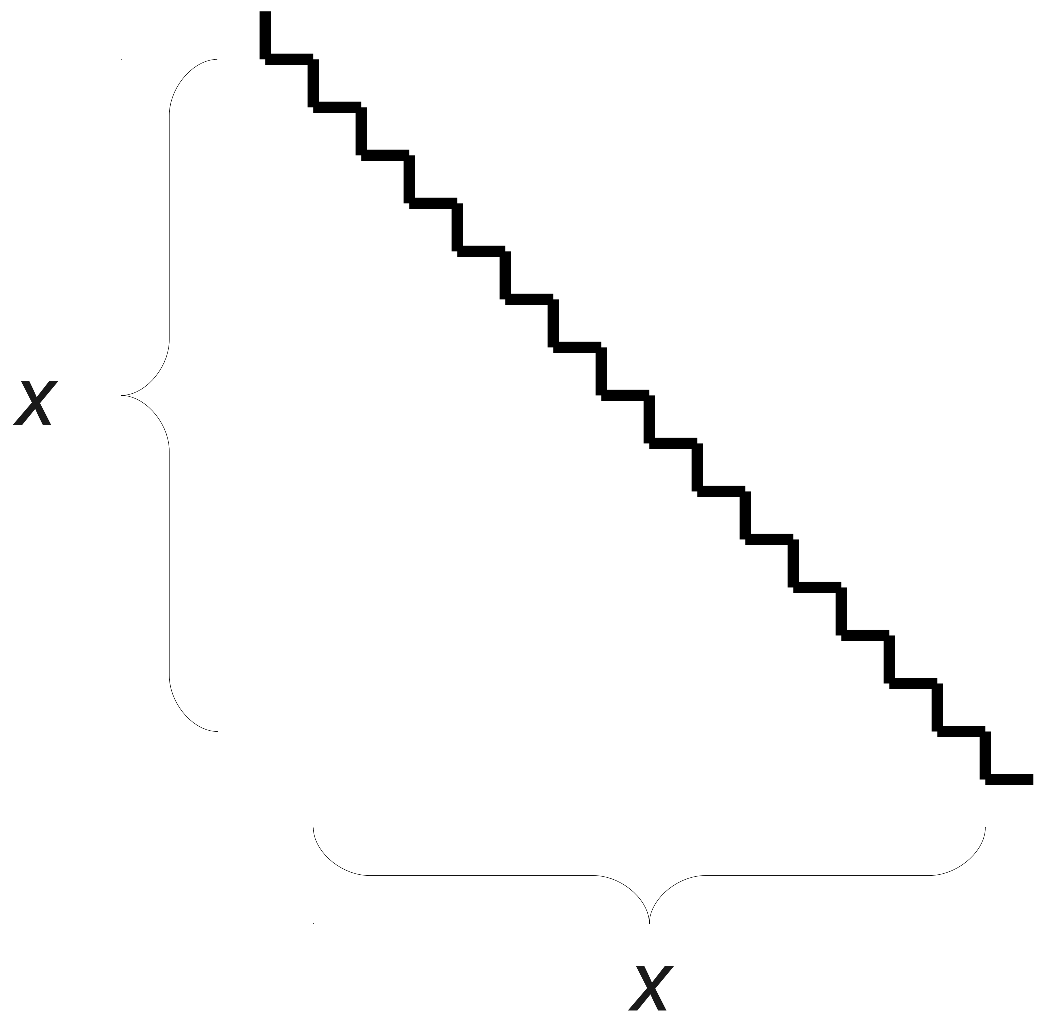
Reasoning about Infinity



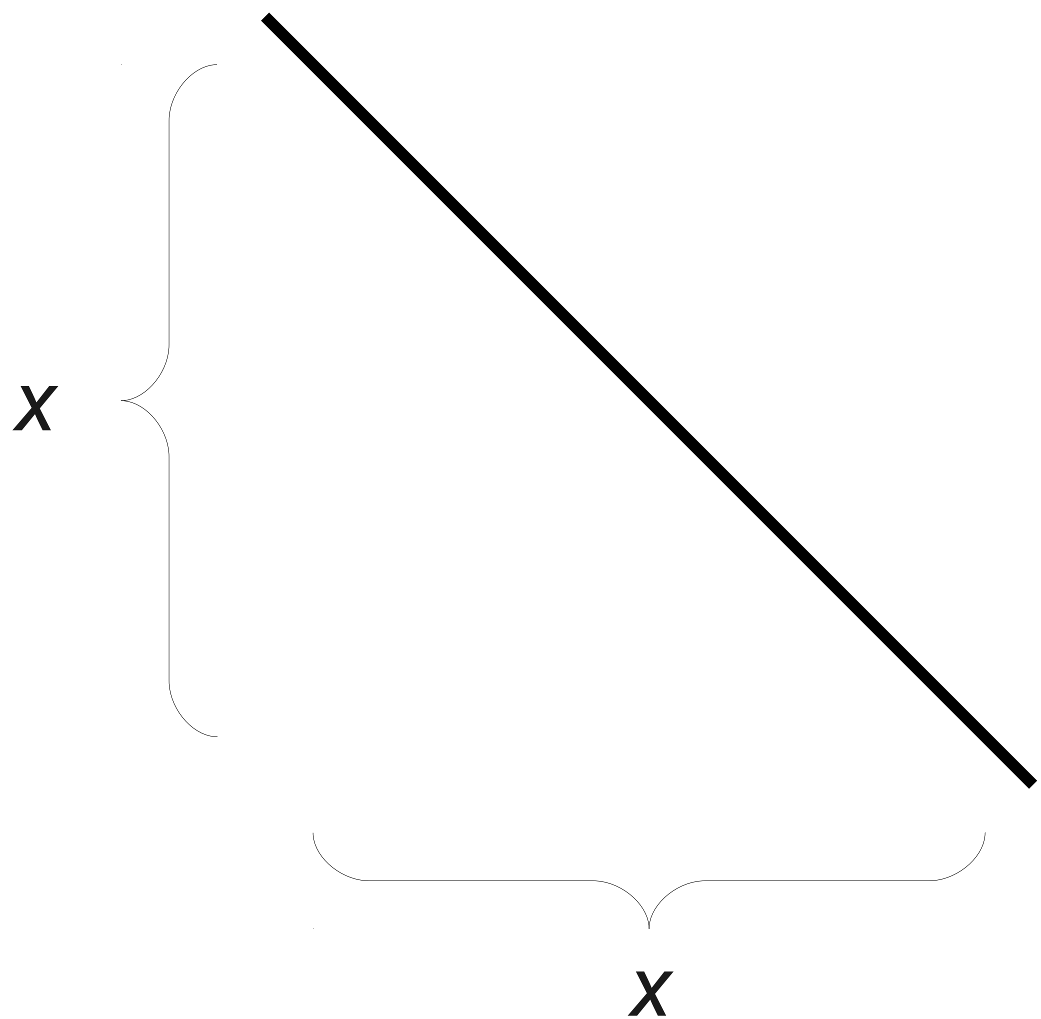
Reasoning about Infinity



Reasoning about Infinity



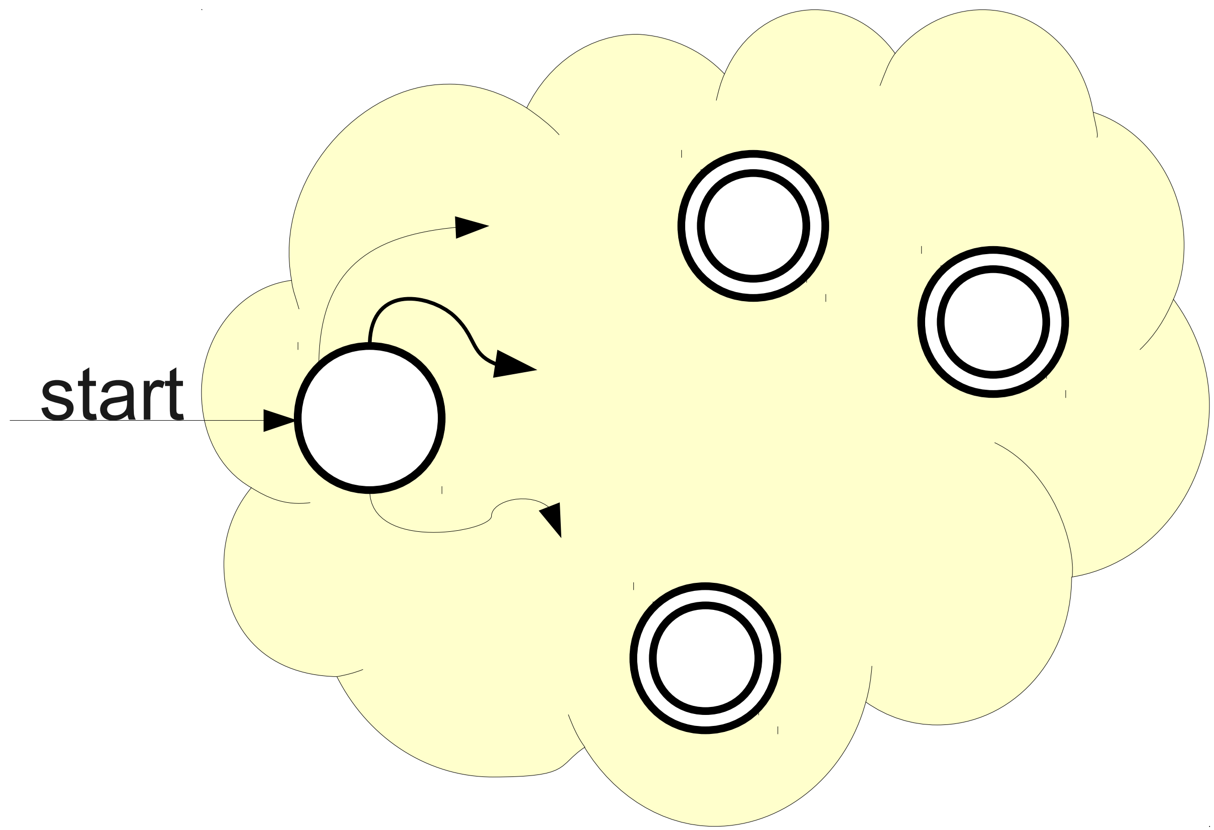
Reasoning about Infinity



Reasoning About the Infinite

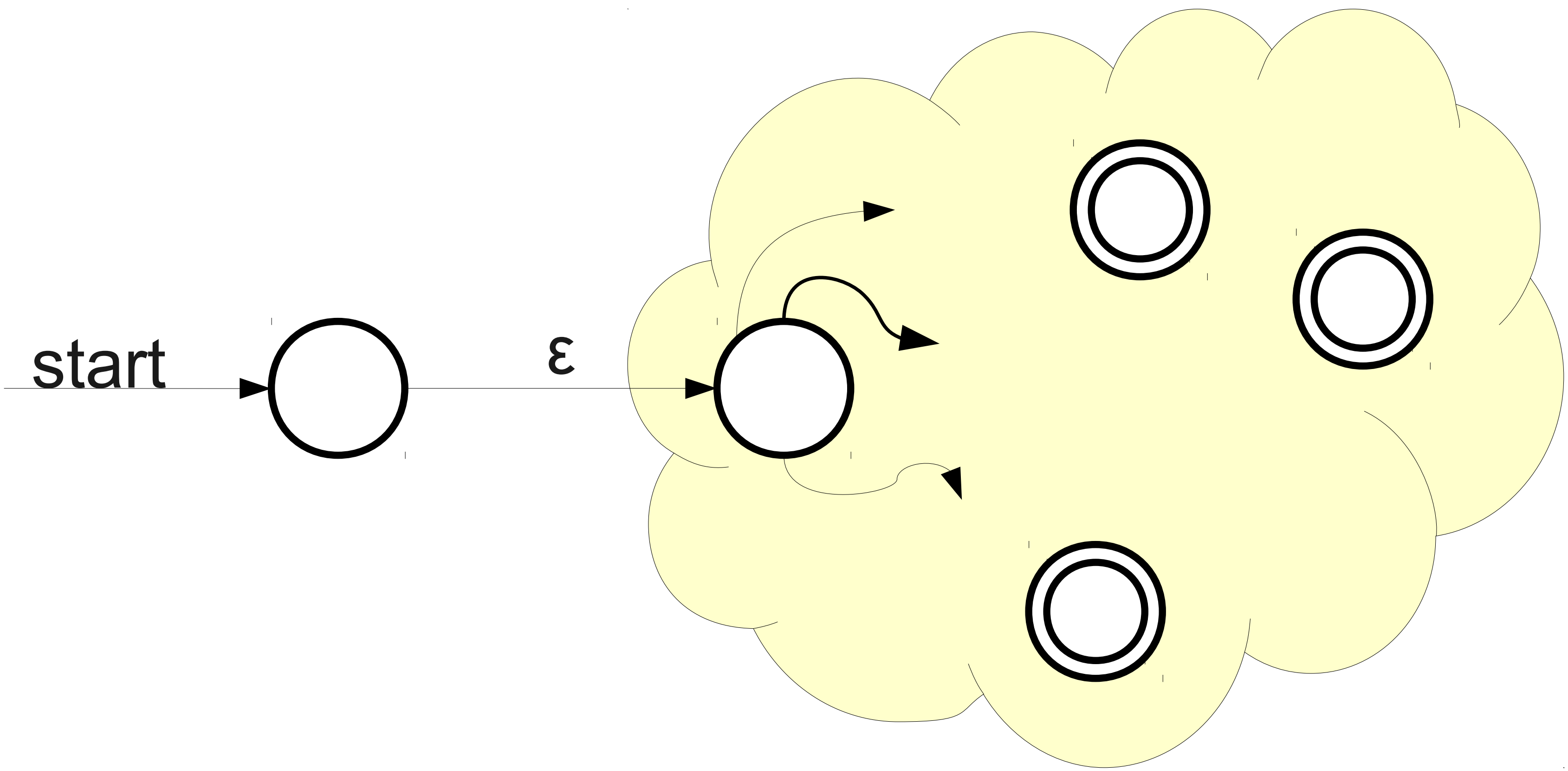
- If a series of finite objects all have some property, their infinite union **does not** necessarily have that property!
  - No matter how many times we zigzag that line, it's never straight.
  - Concluding that it must be equal "in the limit" is not mathematically precise.
  - (This is why calculus is interesting).
- **A better intuition:** Can we convert an NFA for the language L to an NFA for the language L^*?

The Kleene Star



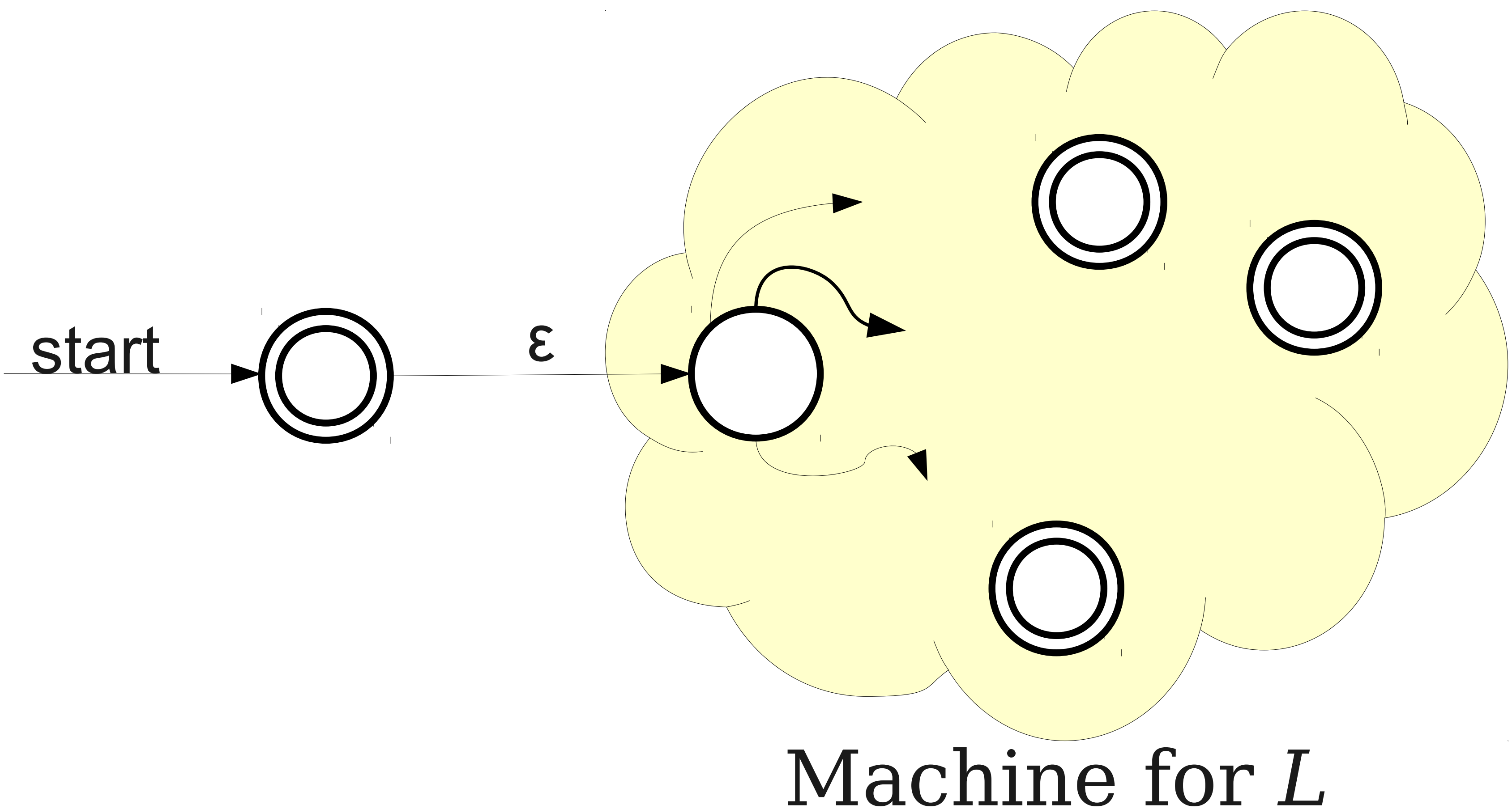
Machine for L

The Kleene Star

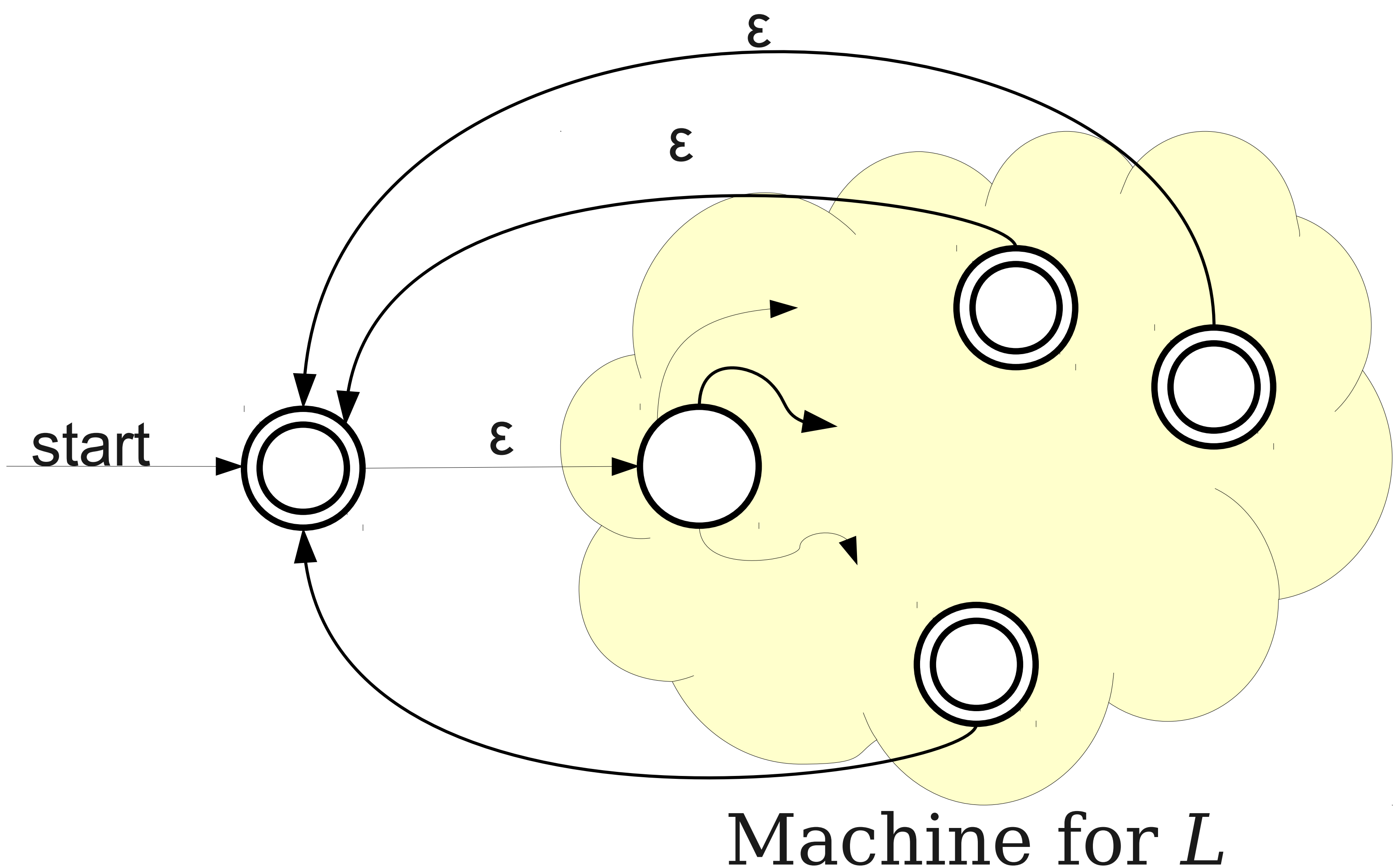


Machine for L

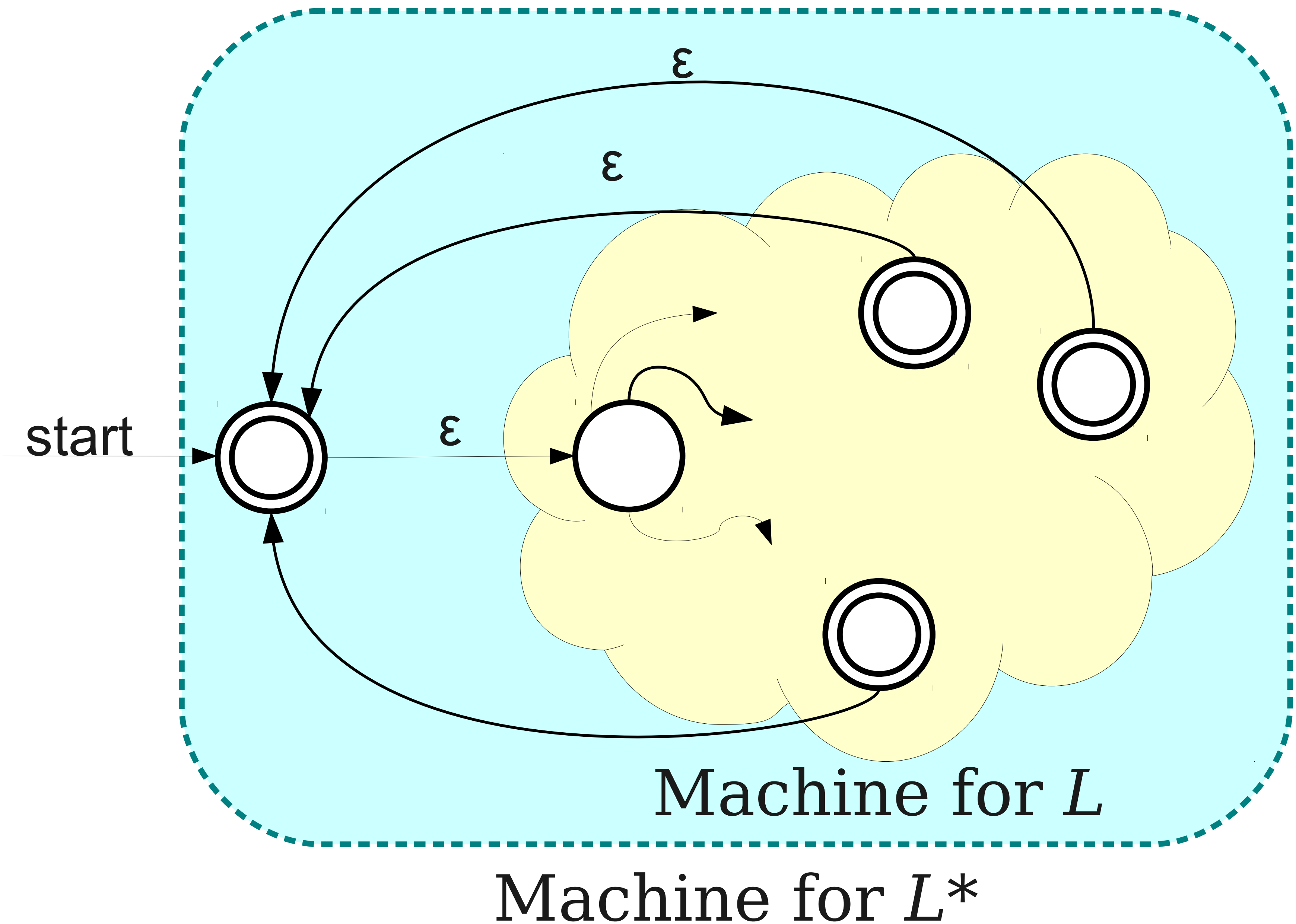
The Kleene Star



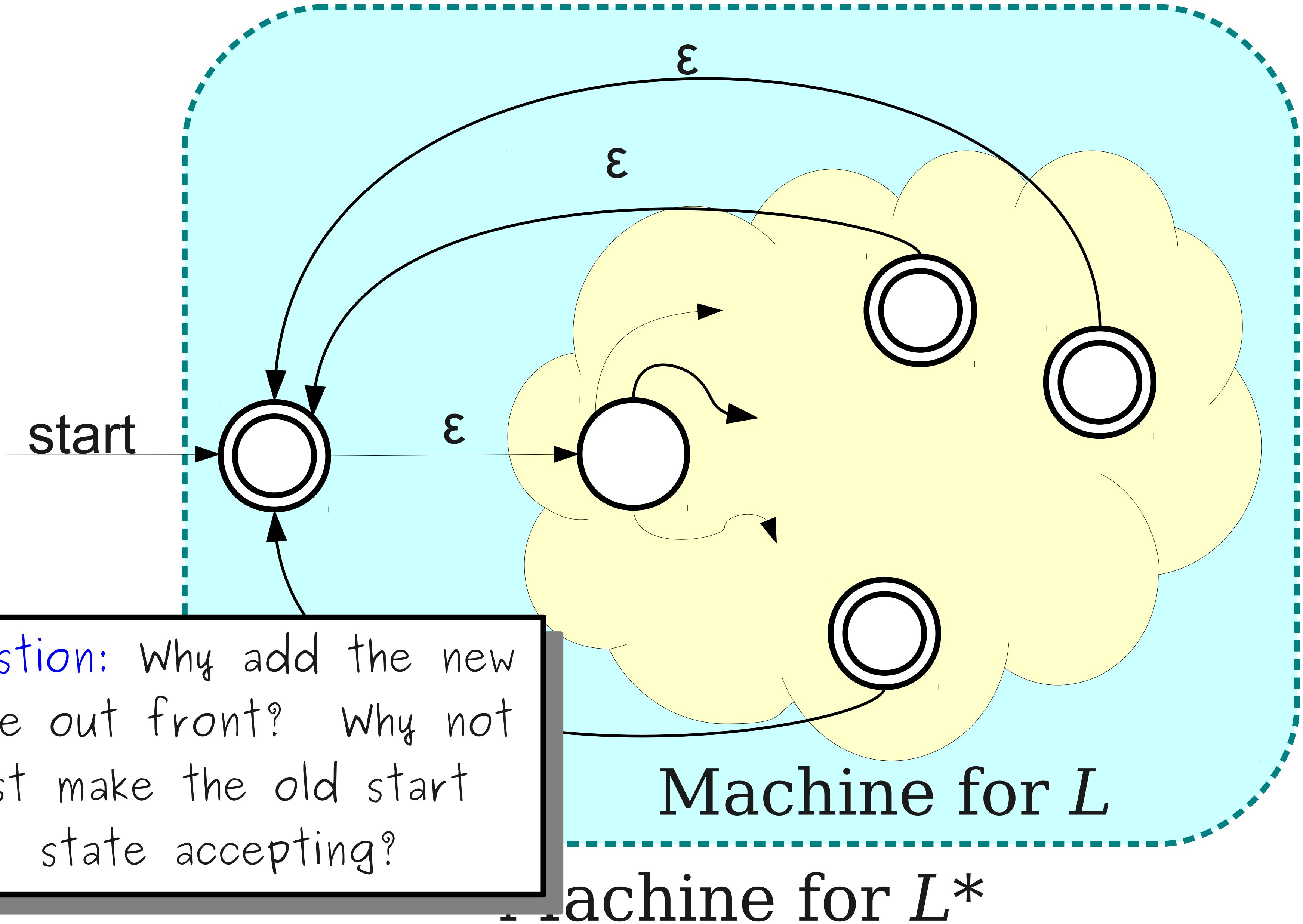
The Kleene Star



The Kleene Star



The Kleene Star



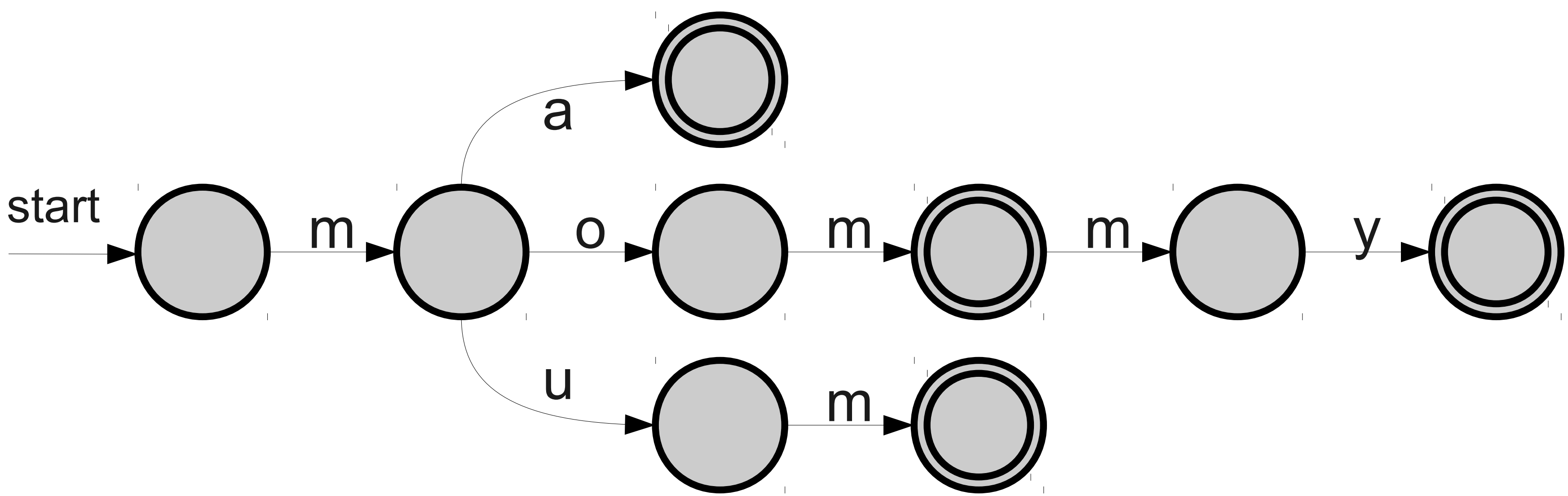
Question: Why add the new state out front? Why not just make the old start state accepting?

Kleene Star in Action

$L = \{$ ma, mom, mommy, mum $\}$

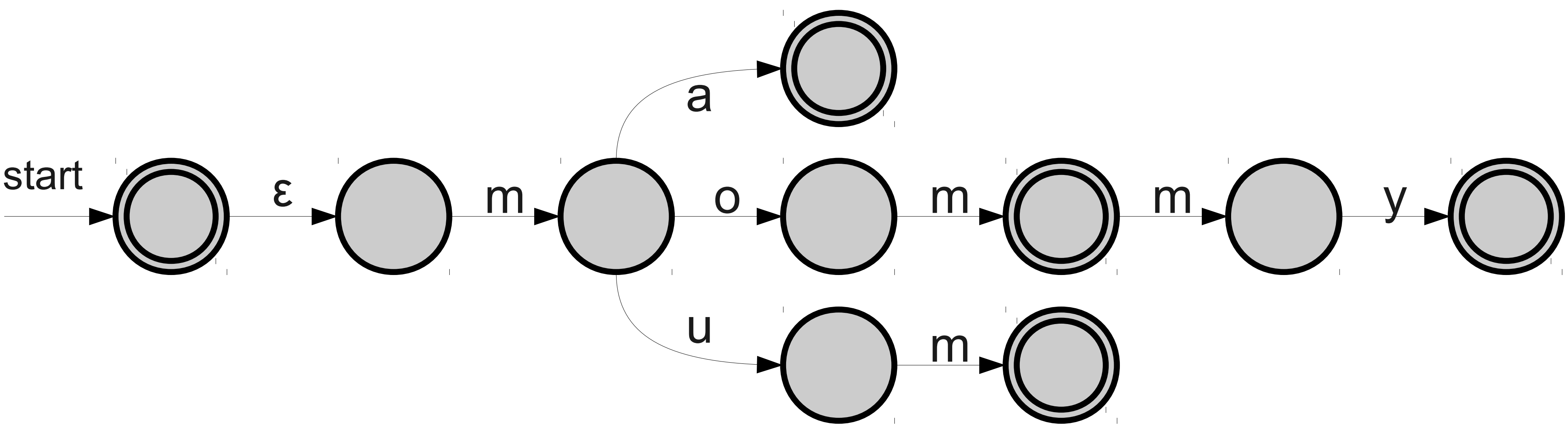
Kleene Star in Action

L = { ma, mom, mommy, mum }



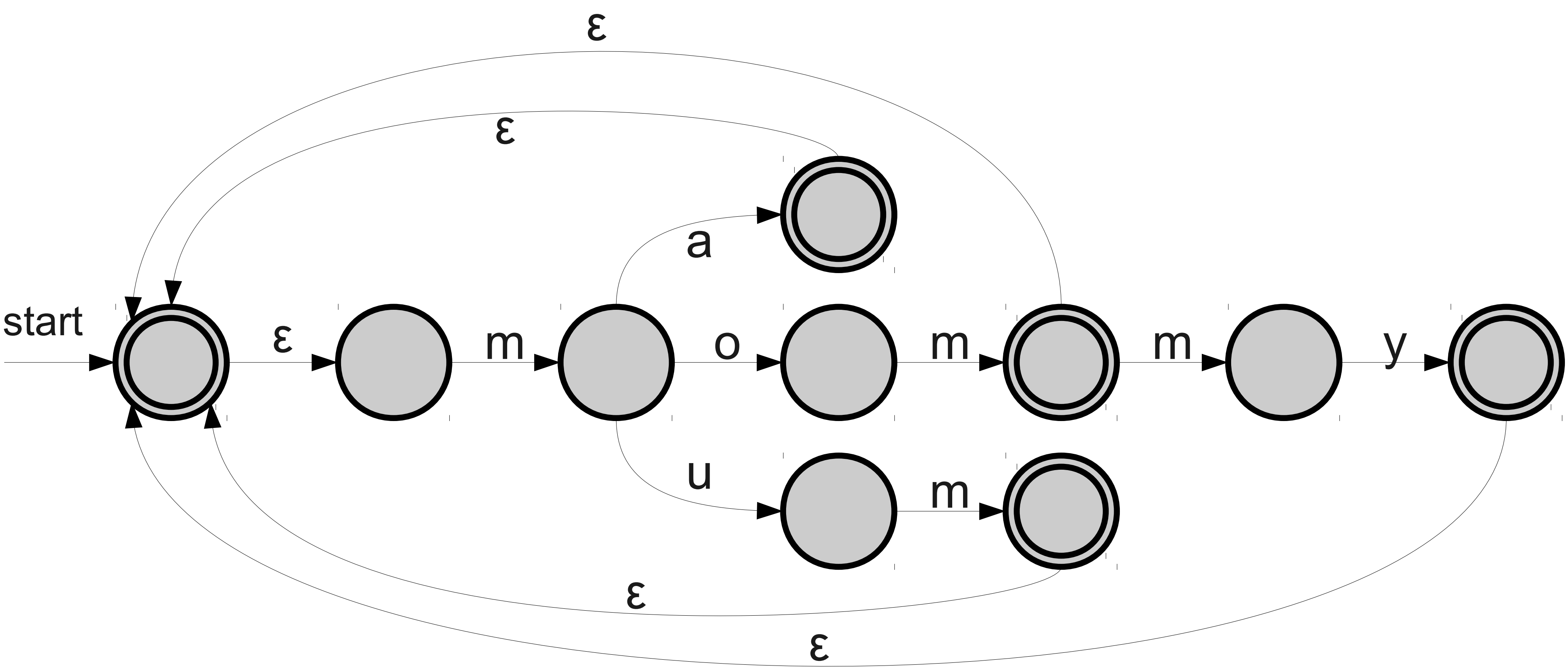
Kleene Star in Action

L = { ma, mom, mommy, mum }



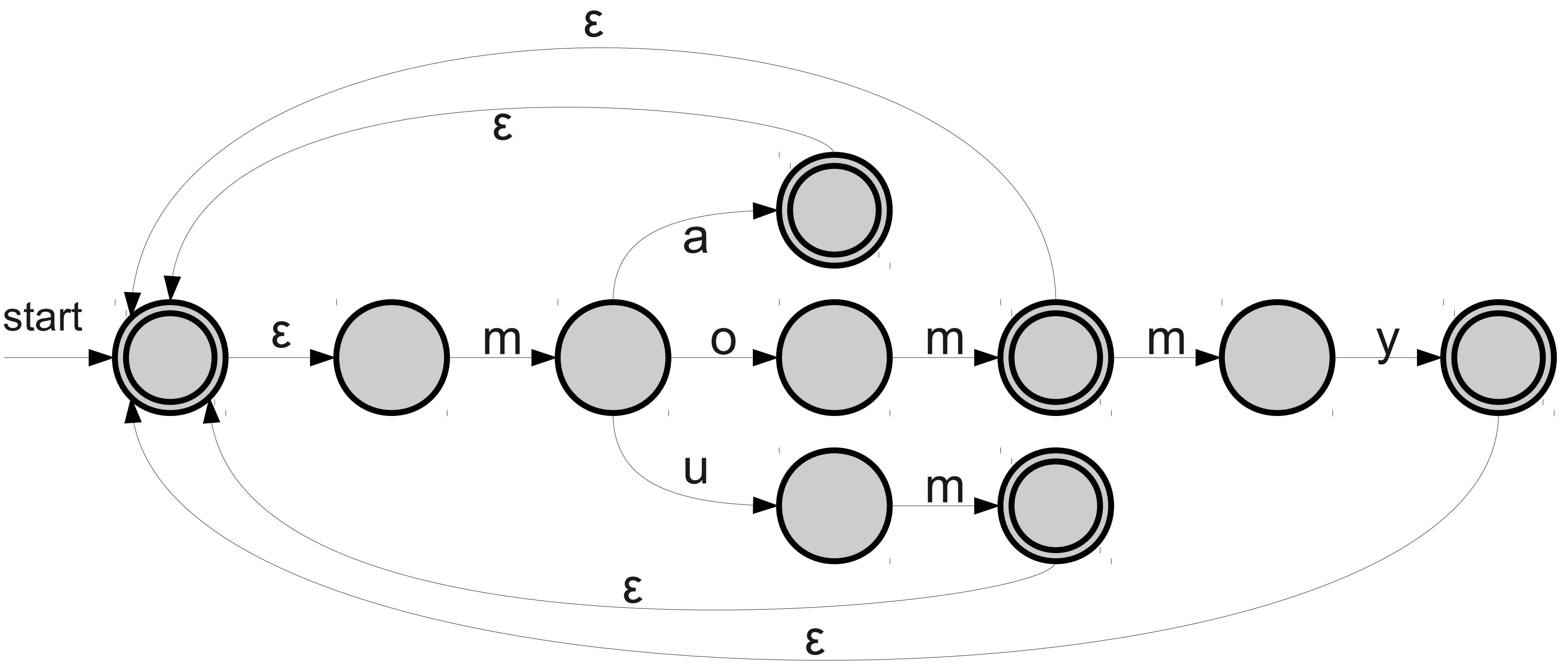
Kleene Star in Action

$$L = \{ \text{ma, mom, mommy, mum} \}$$



Kleene Star in Action

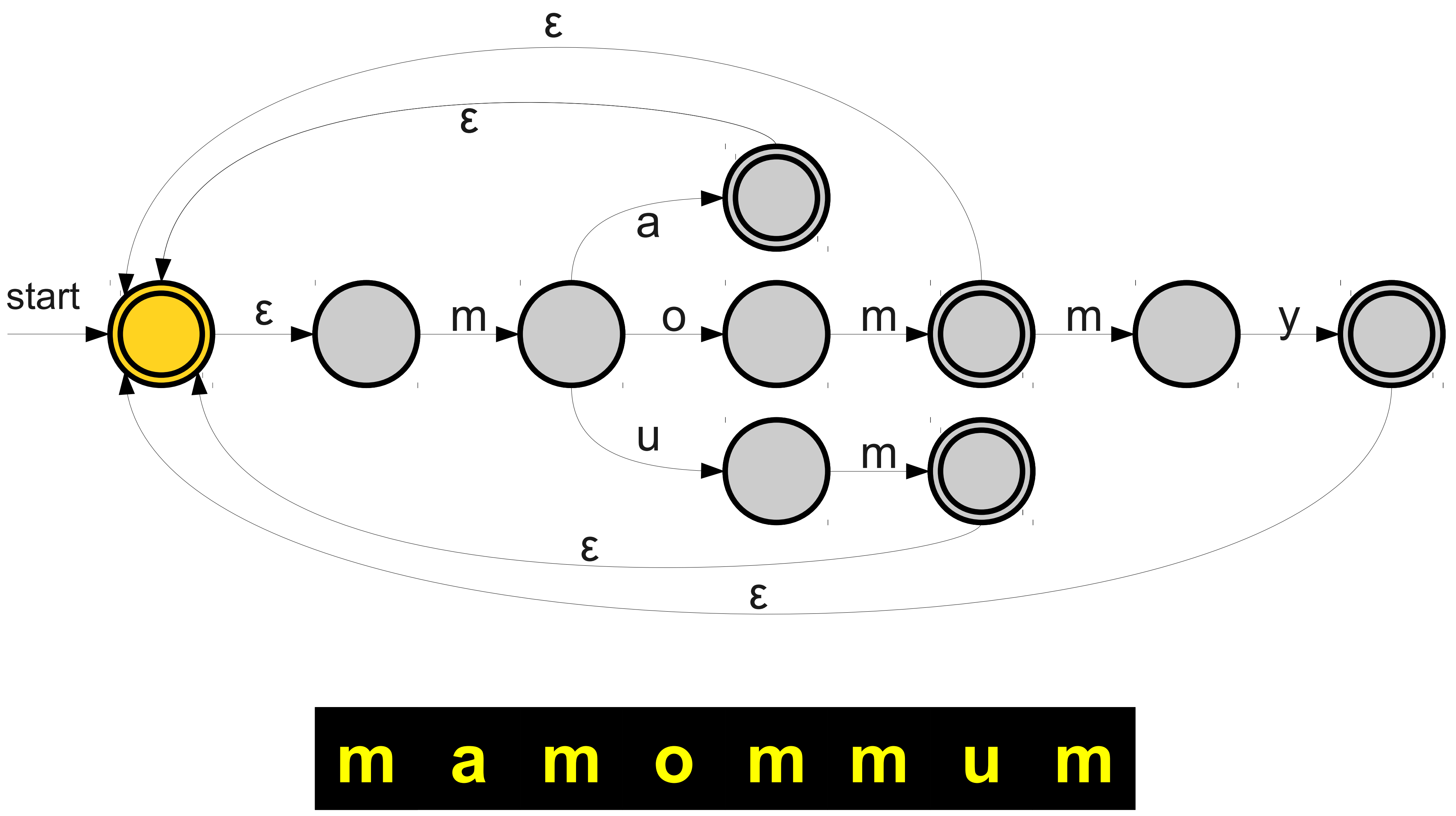
$L = \{$ ma, mom, mommy, mum $\}$



m a m o m m u m

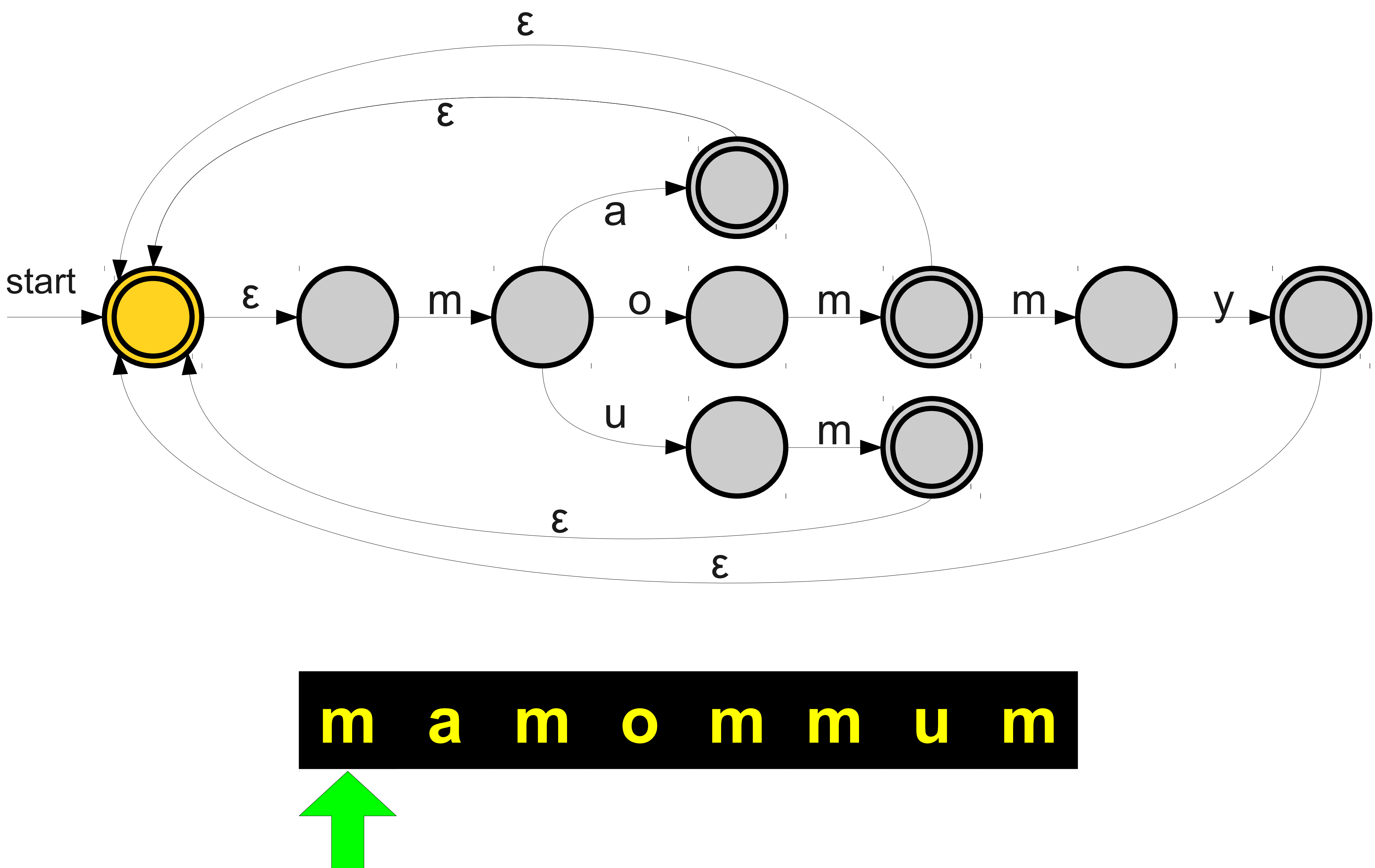
Kleene Star in Action

$L = \{$ ma, mom, mommy, mum $\}$



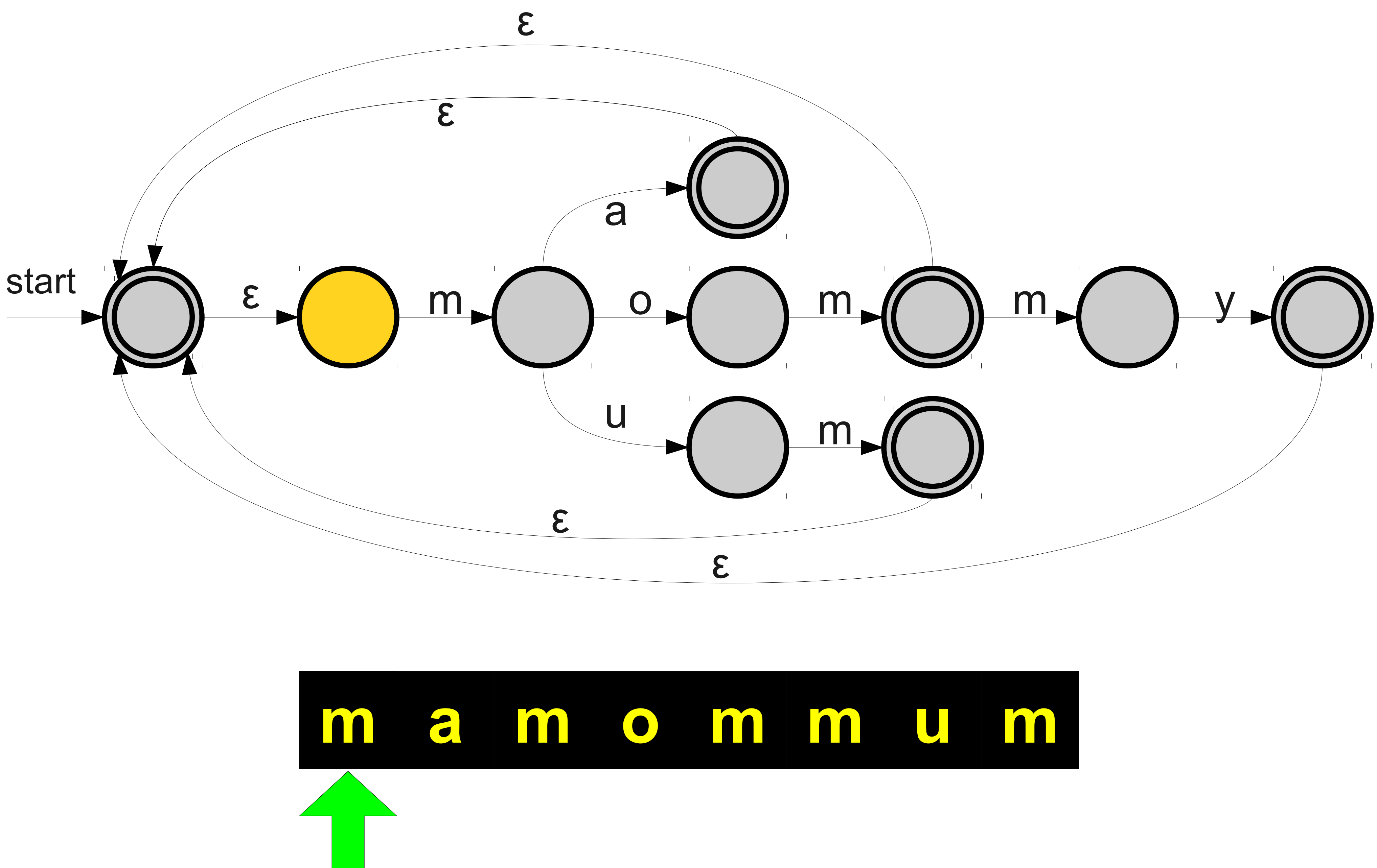
Kleene Star in Action

$L = \{ \text{ma, mom, mommy, mum} \}$



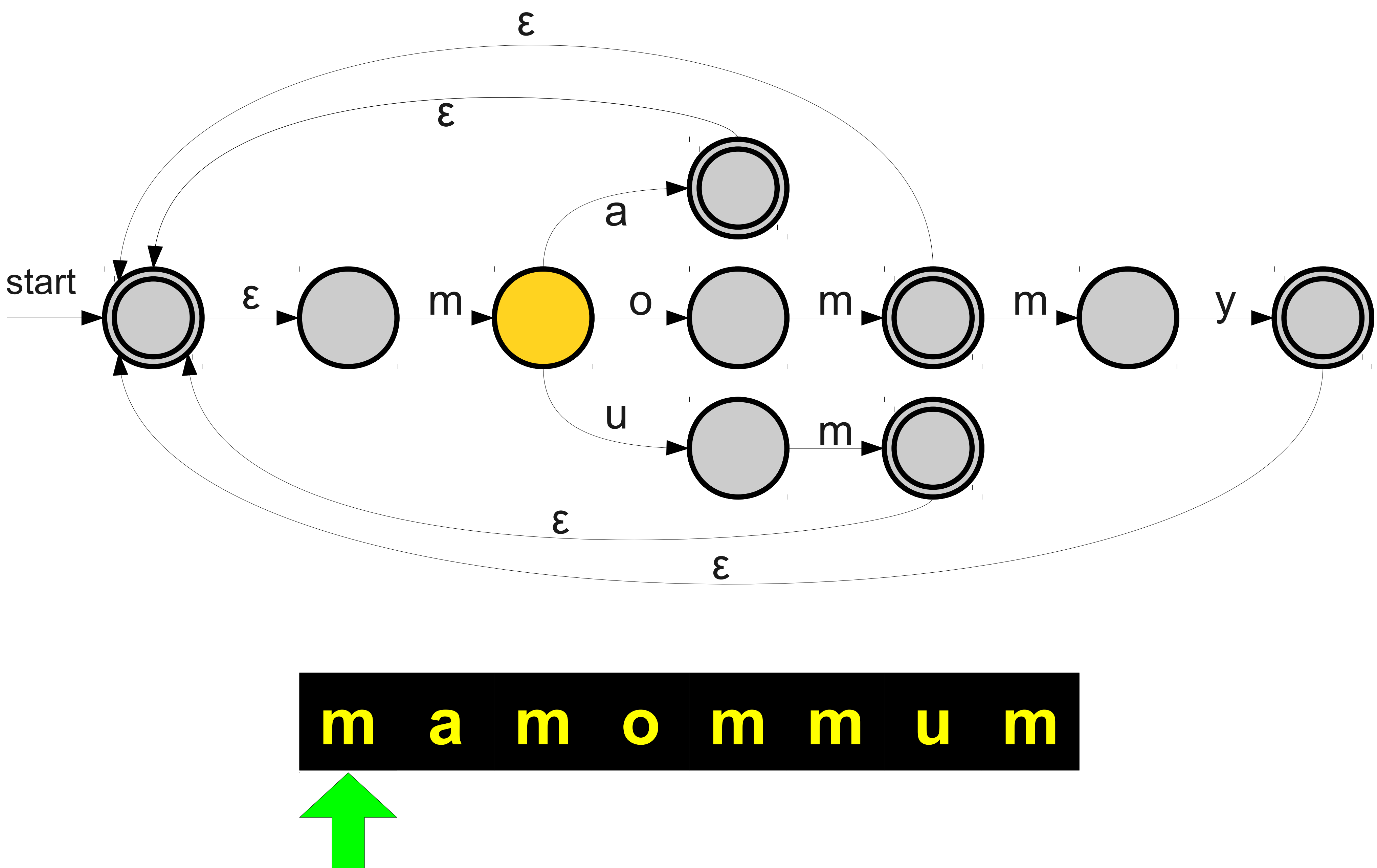
Kleene Star in Action

L = { ma, mom, mommy, mum }



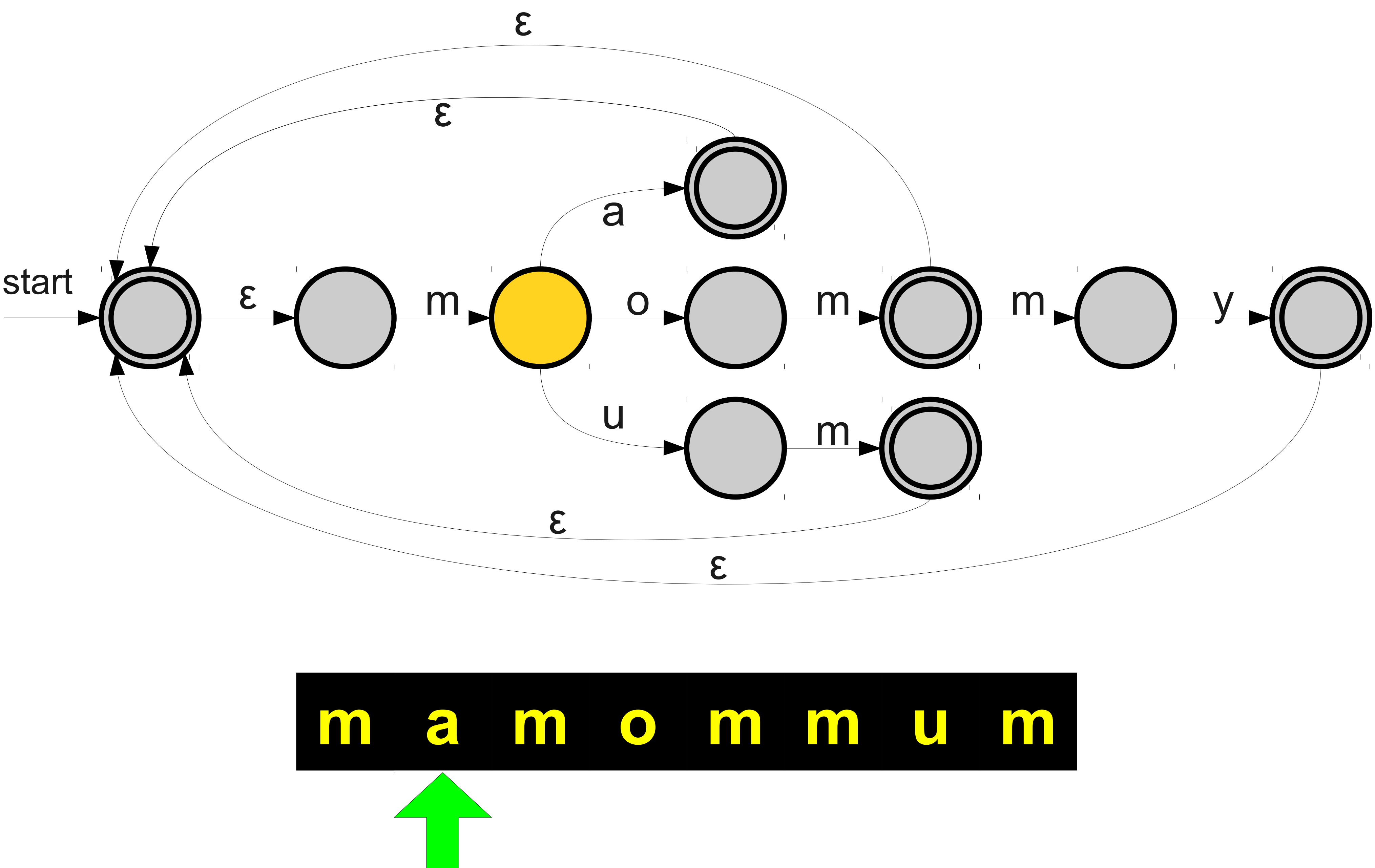
Kleene Star in Action

$L = \{ \text{ma, mom, mommy, mum} \}$



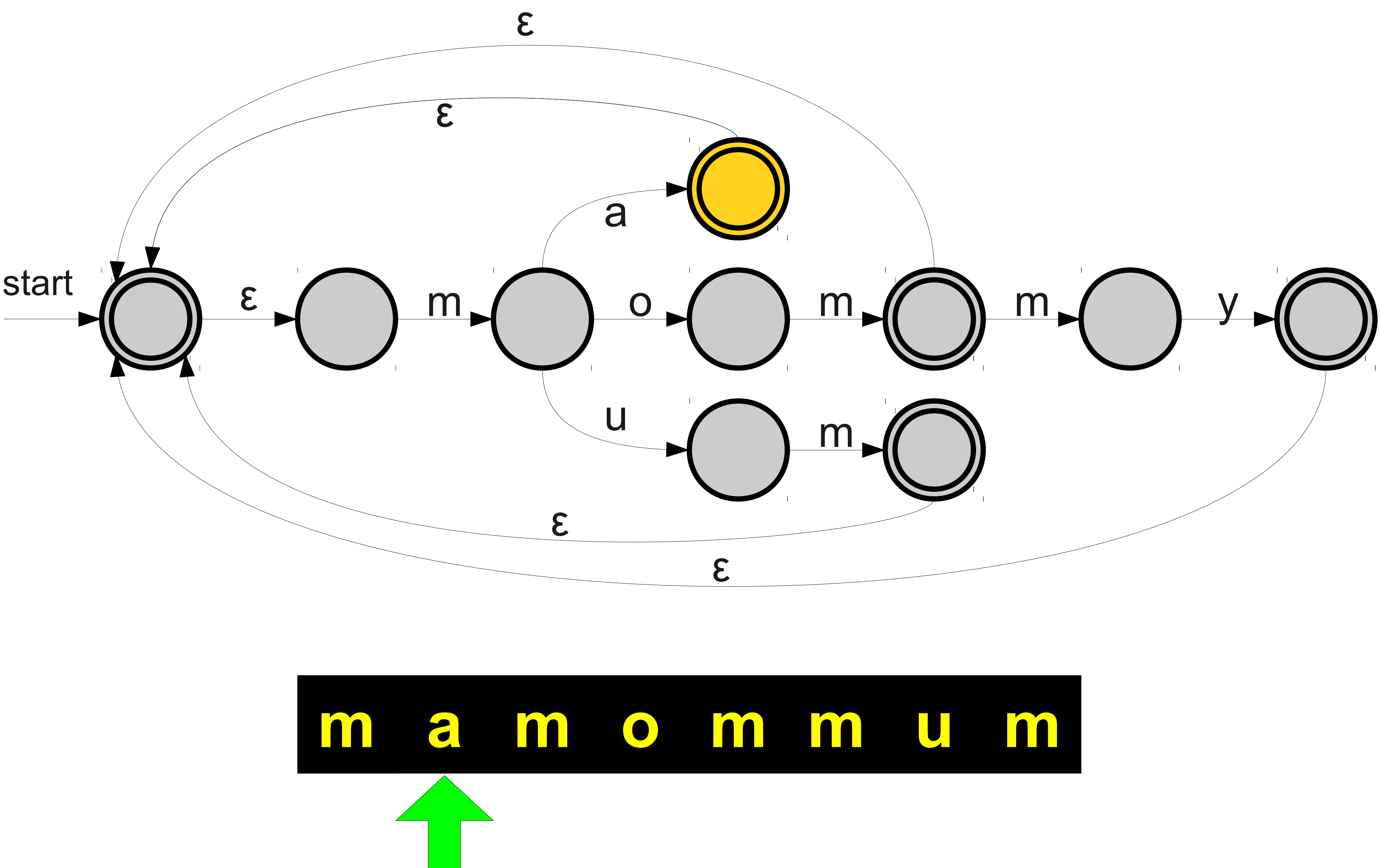
Kleene Star in Action

L = { ma, mom, mommy, mum }



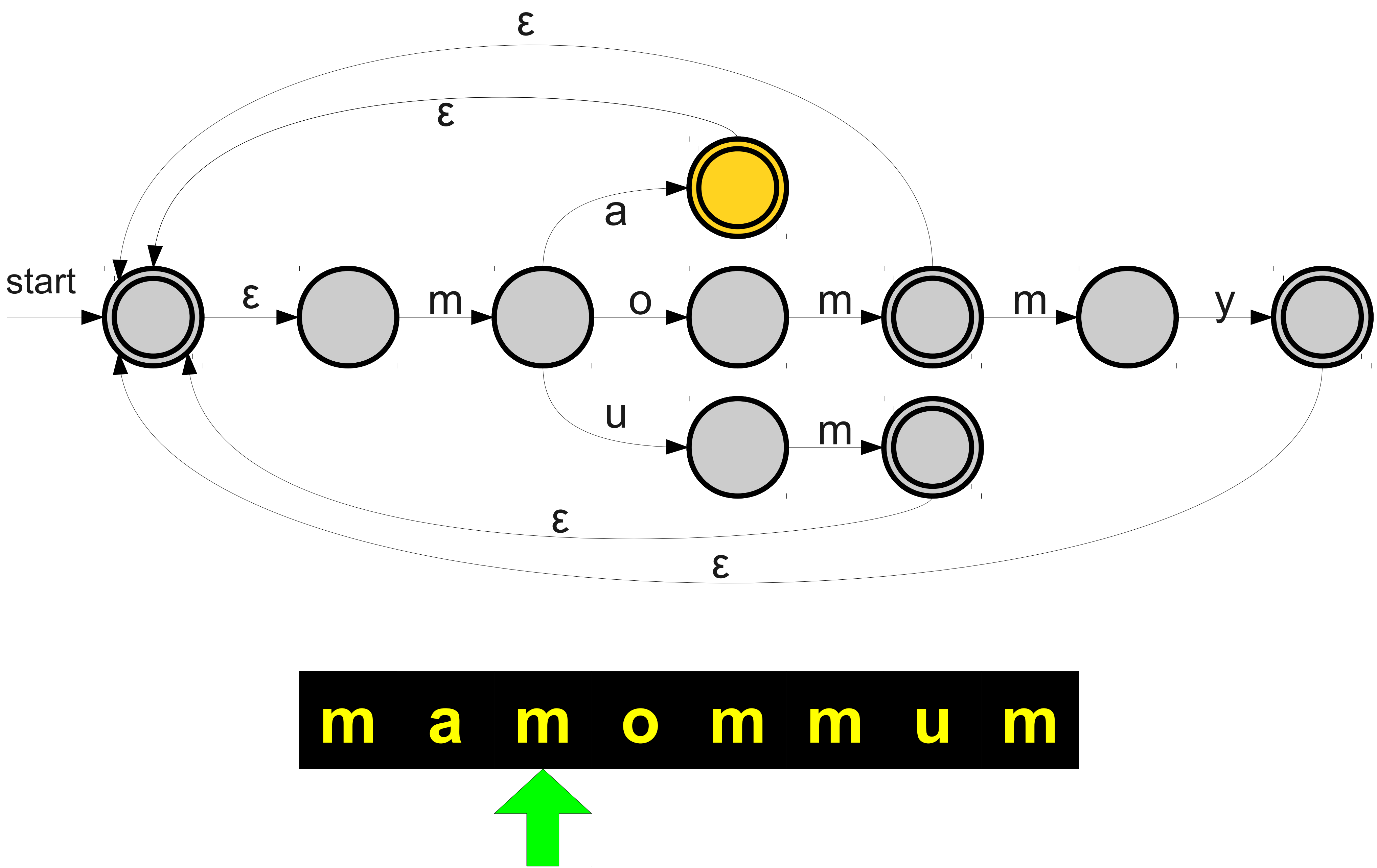
Kleene Star in Action

$L = \{ \text{ma, mom, mommy, mum} \}$



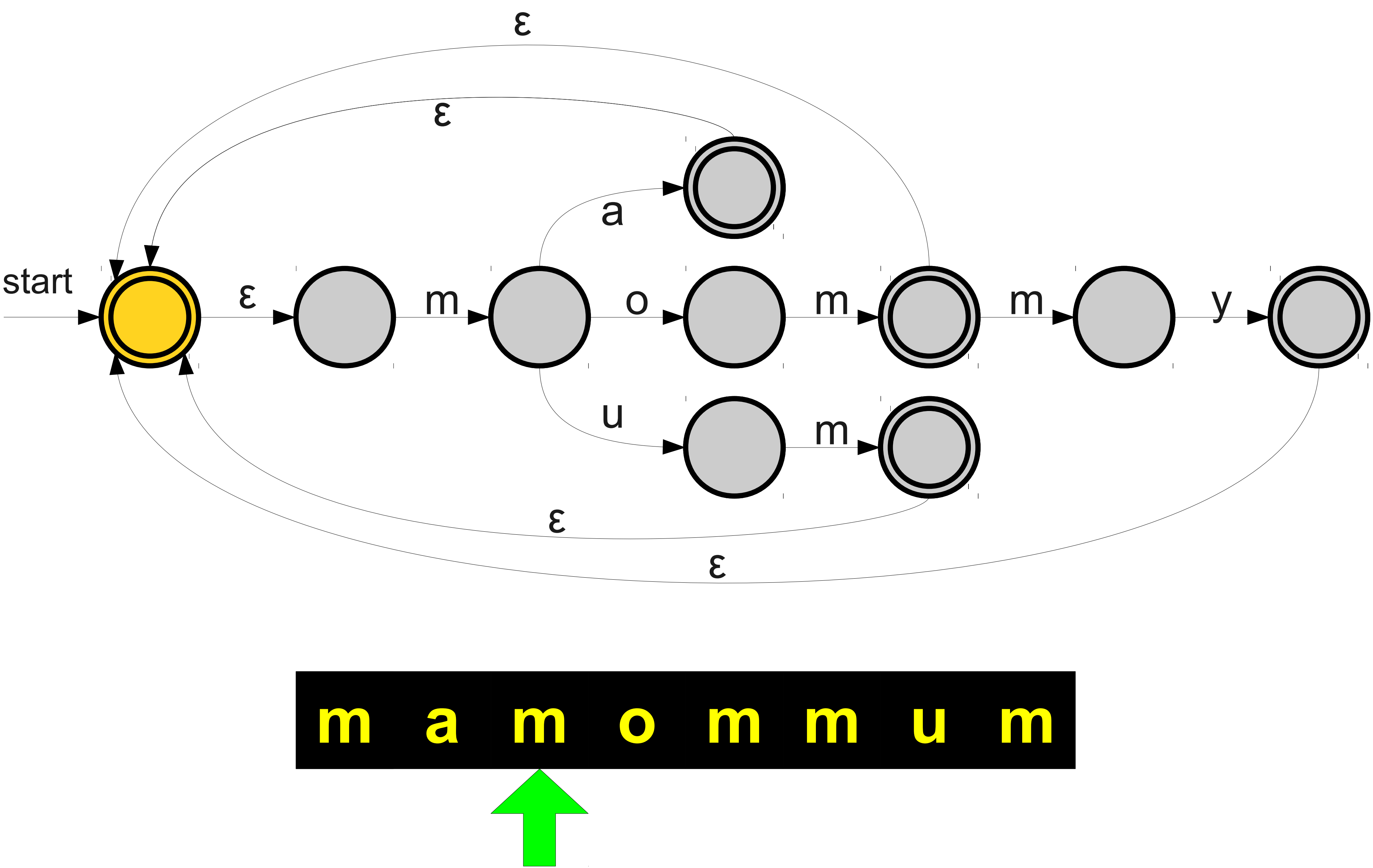
Kleene Star in Action

$L = \{$ ma, mom, mommy, mum $\}$



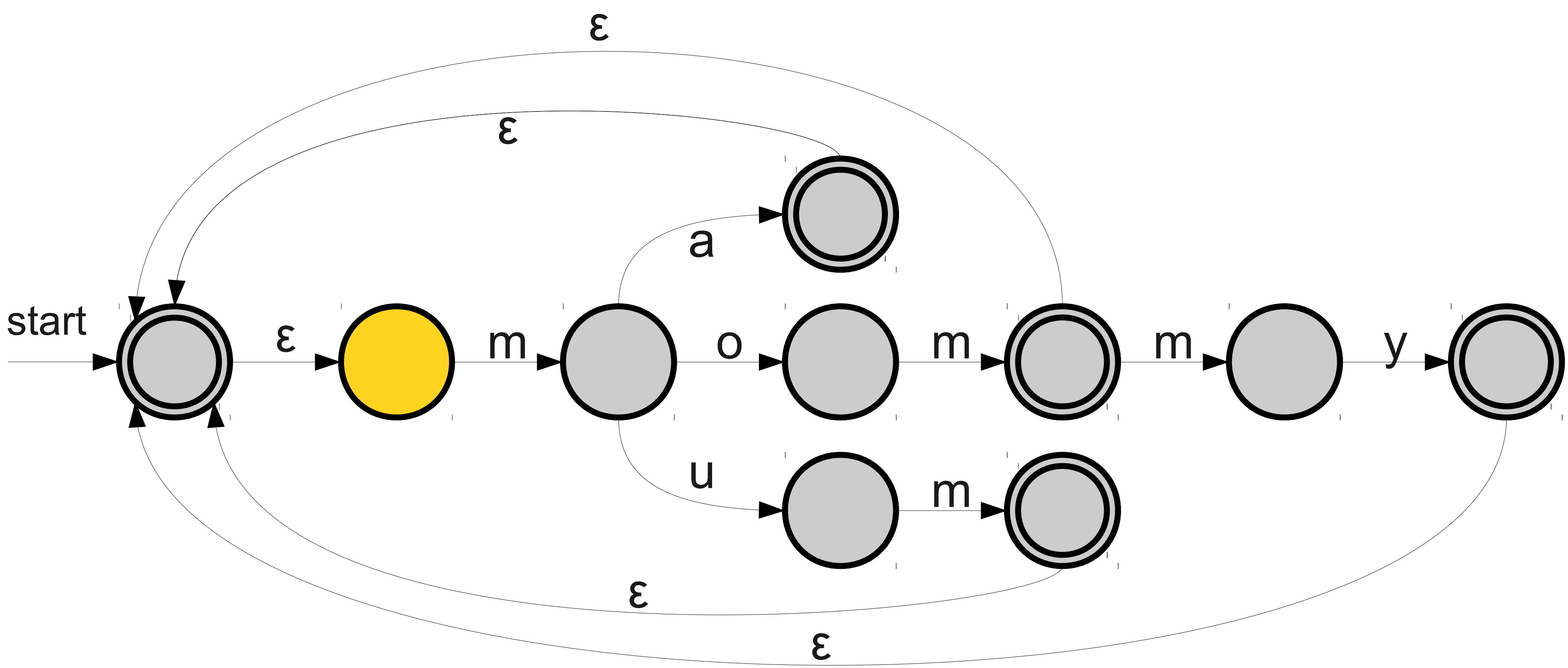
Kleene Star in Action

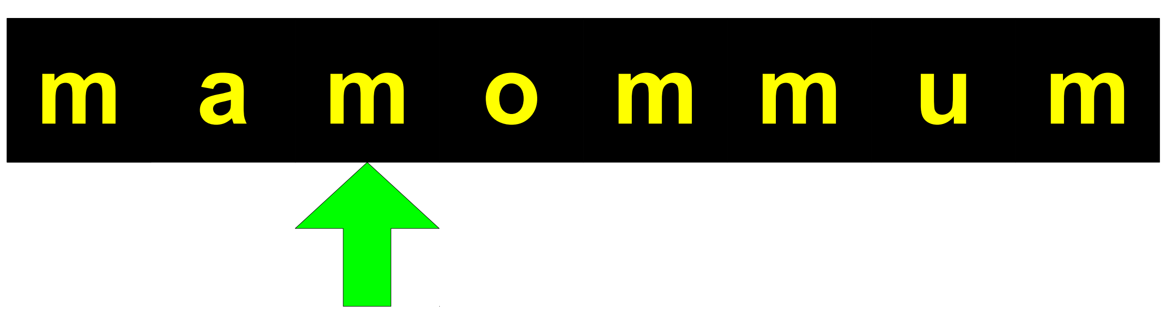
$L = \{ \text{ma}, \text{mom}, \text{mommy}, \text{mum} \}$



Kleene Star in Action

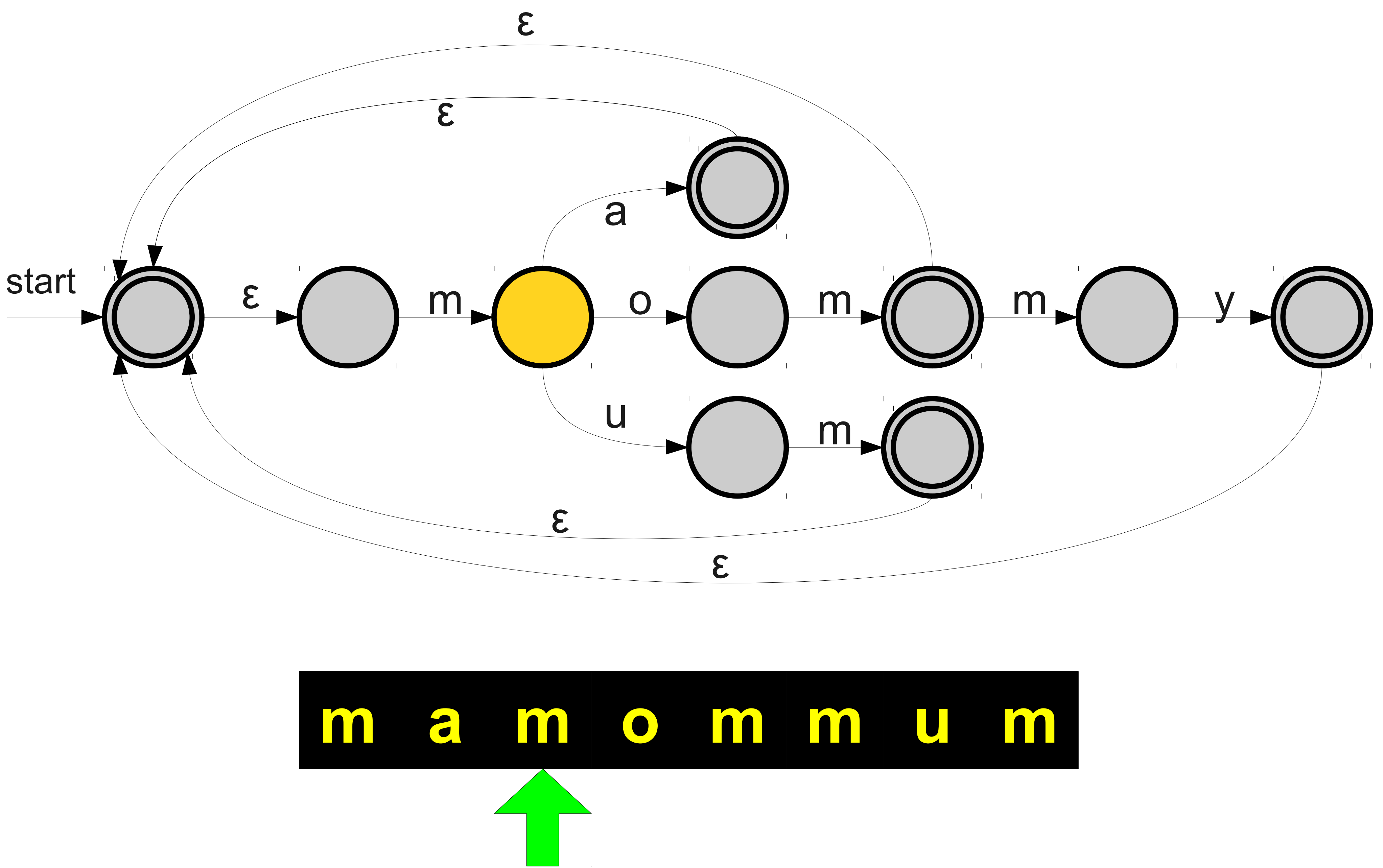
$L = \{$ ma, mom, mommy, mum $\}$





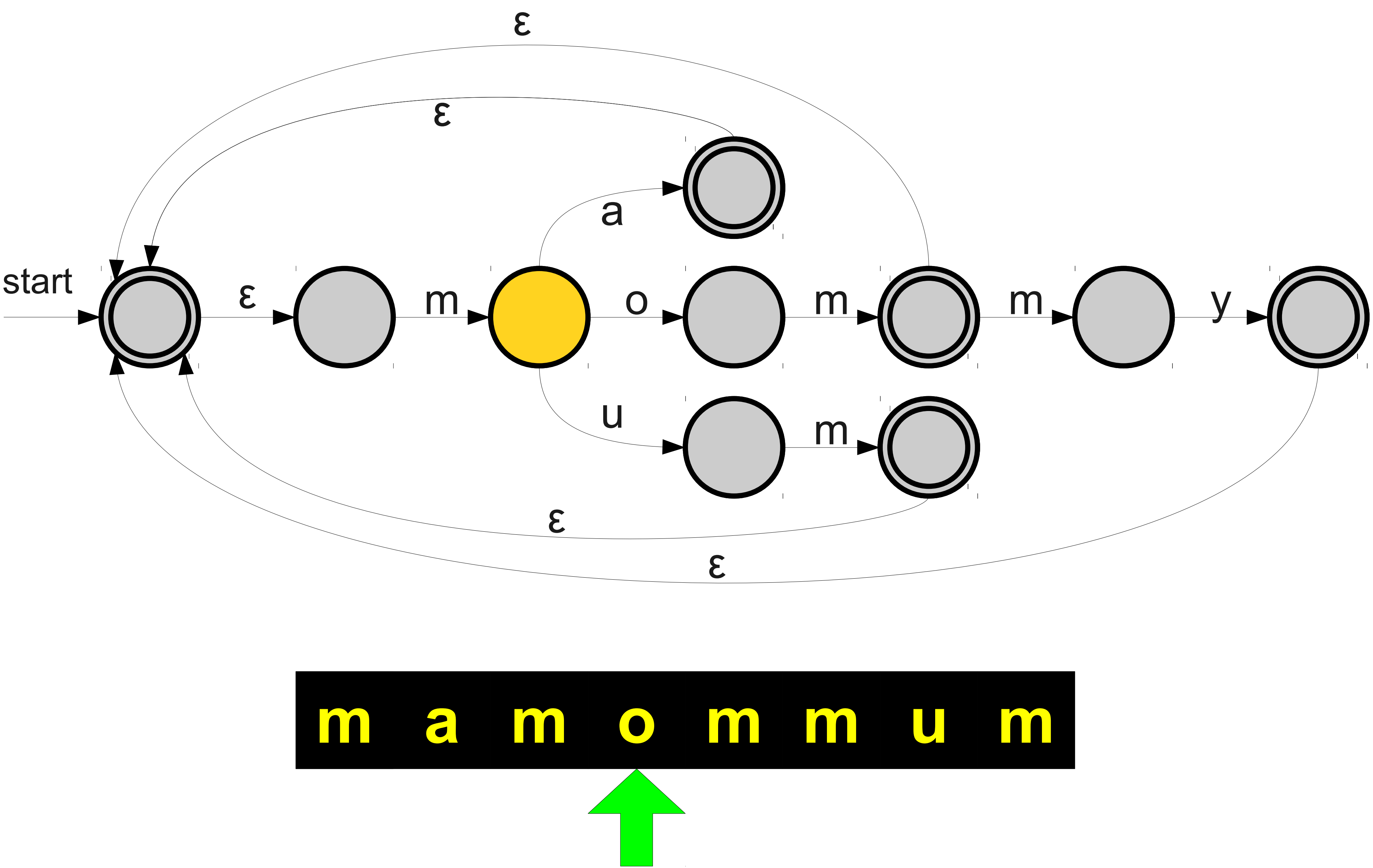
Kleene Star in Action

$L = \{ \text{ma, mom, mommy, mum} \}$



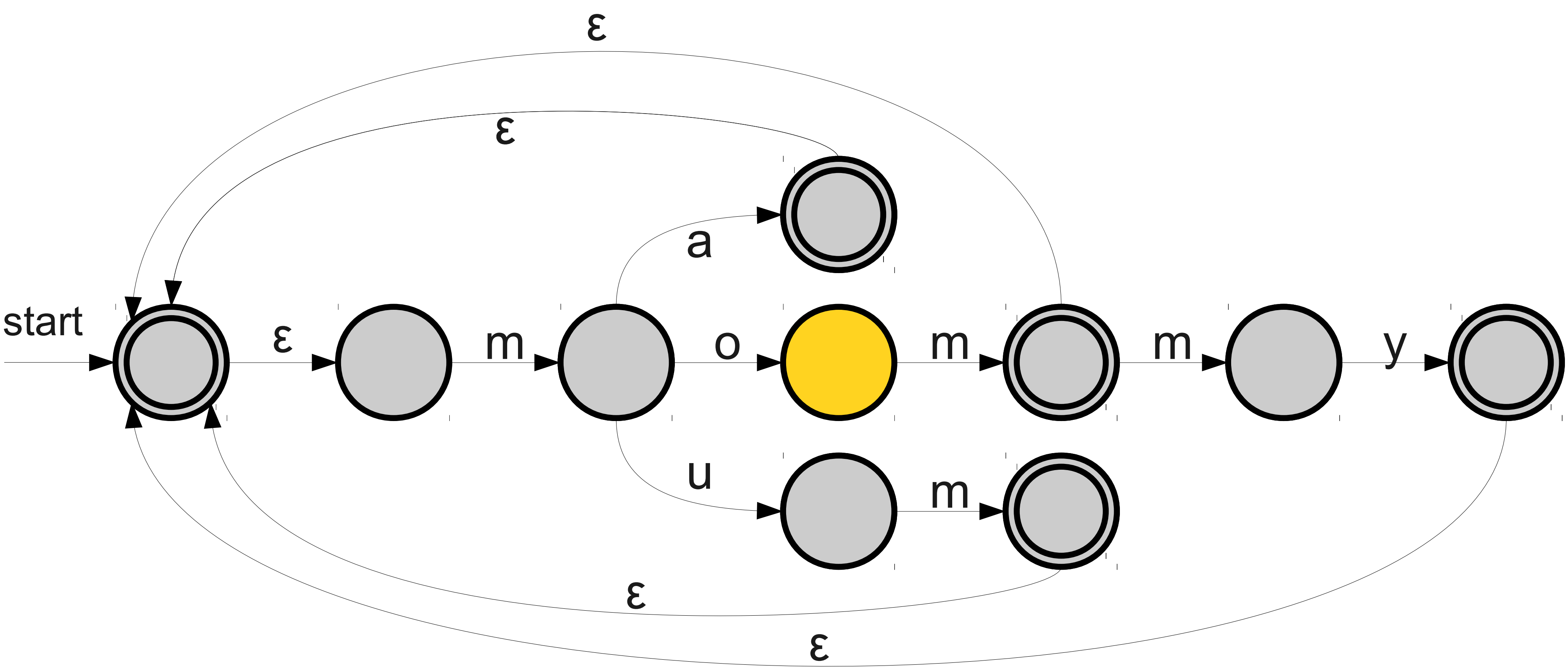
Kleene Star in Action

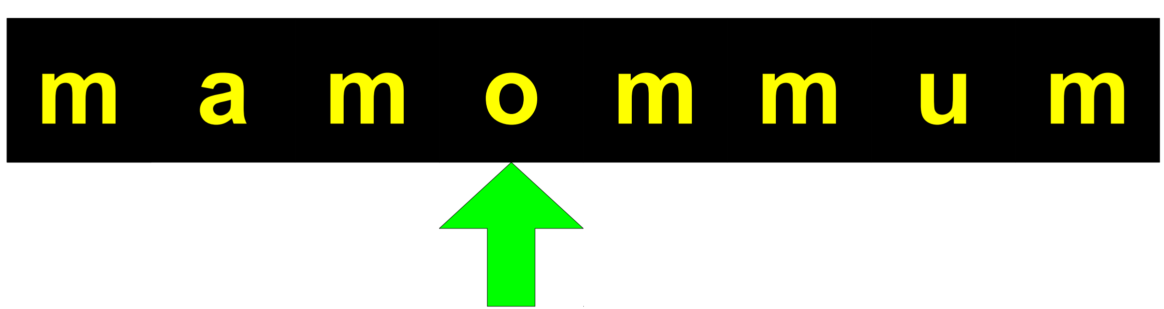
$L = \{$ ma, mom, mommy, mum $\}$



Kleene Star in Action

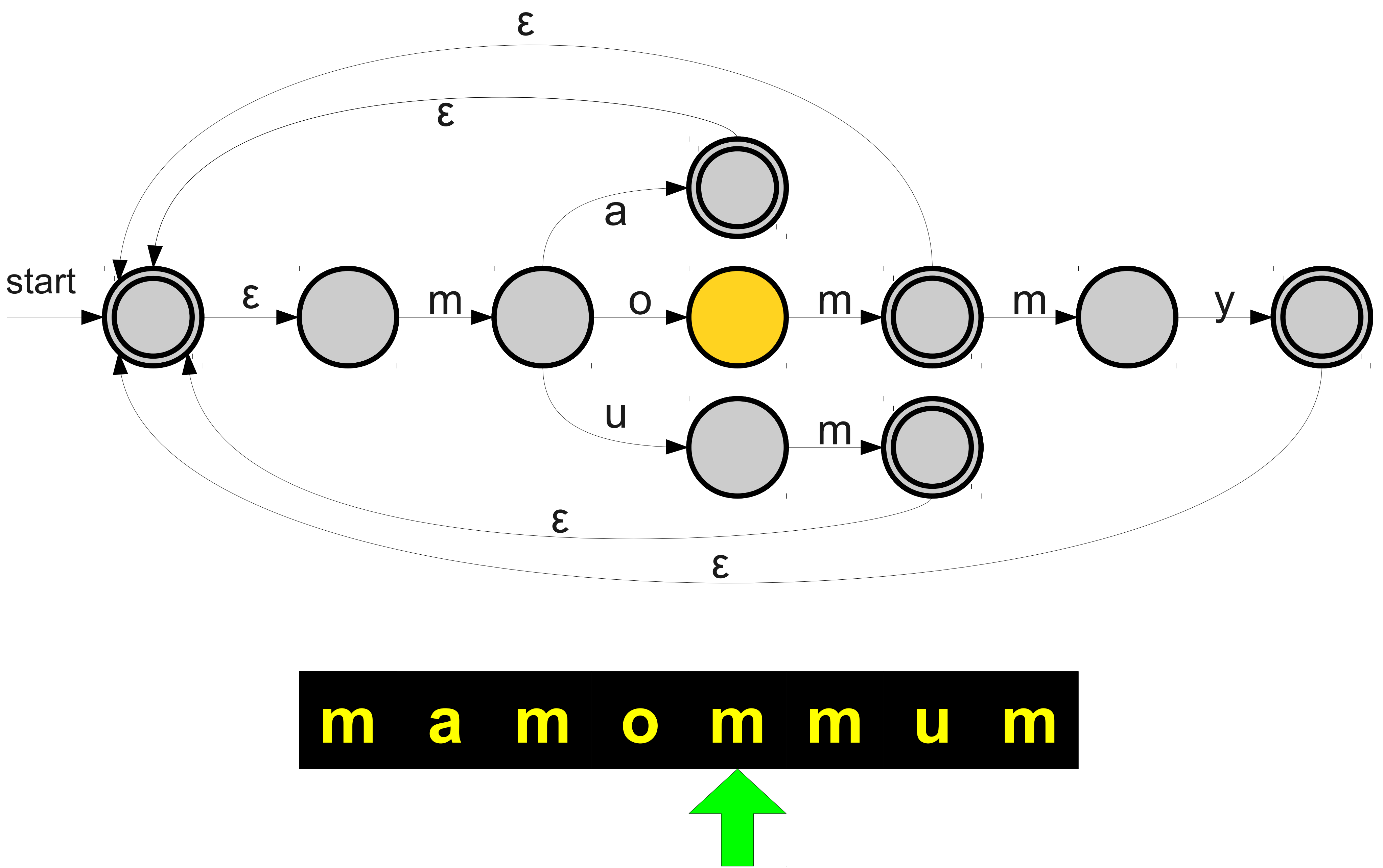
L = { ma, mom, mommy, mum }





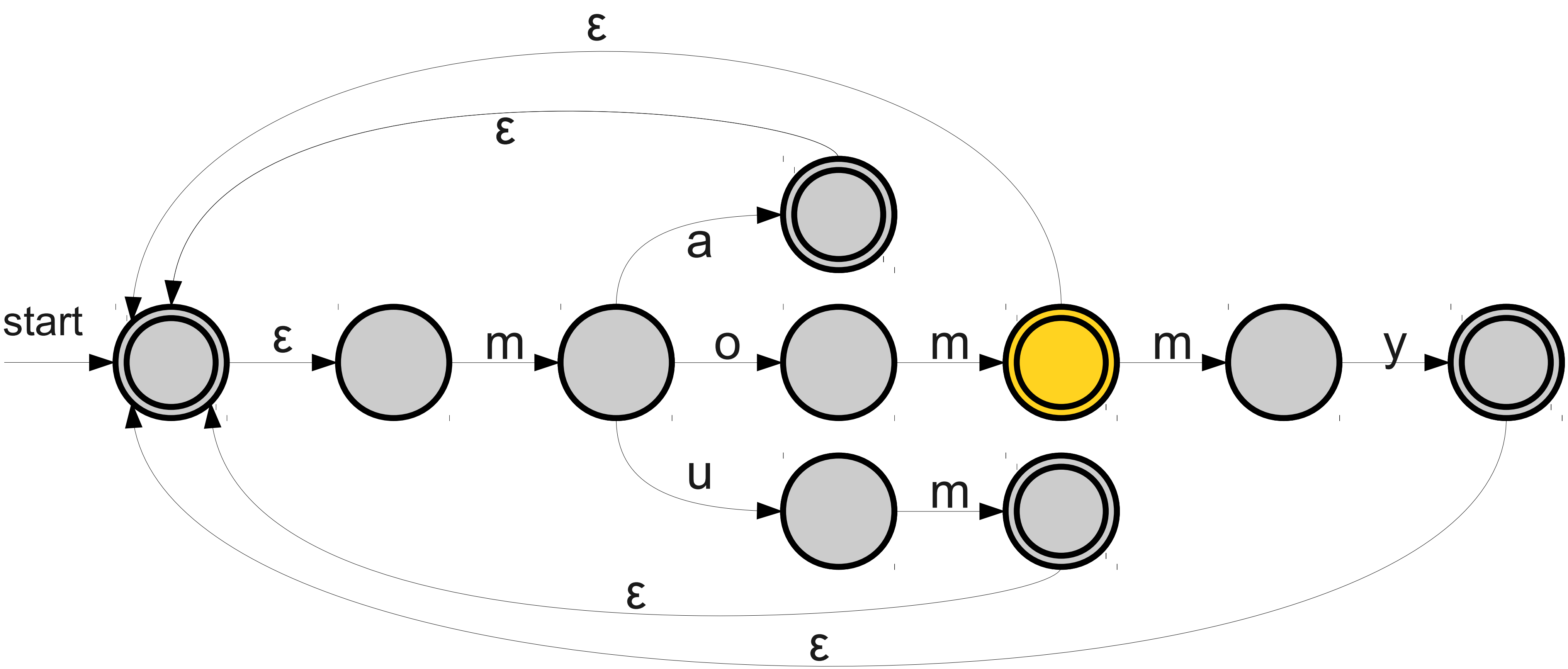
Kleene Star in Action

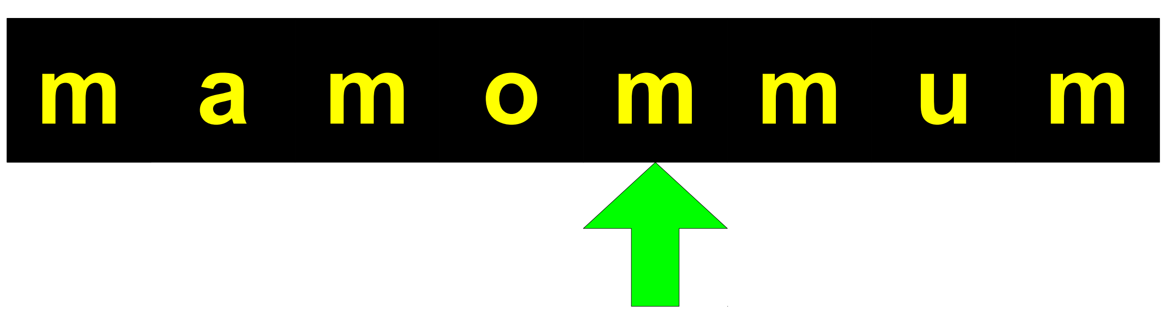
L = { ma, mom, mommy, mum }



Kleene Star in Action

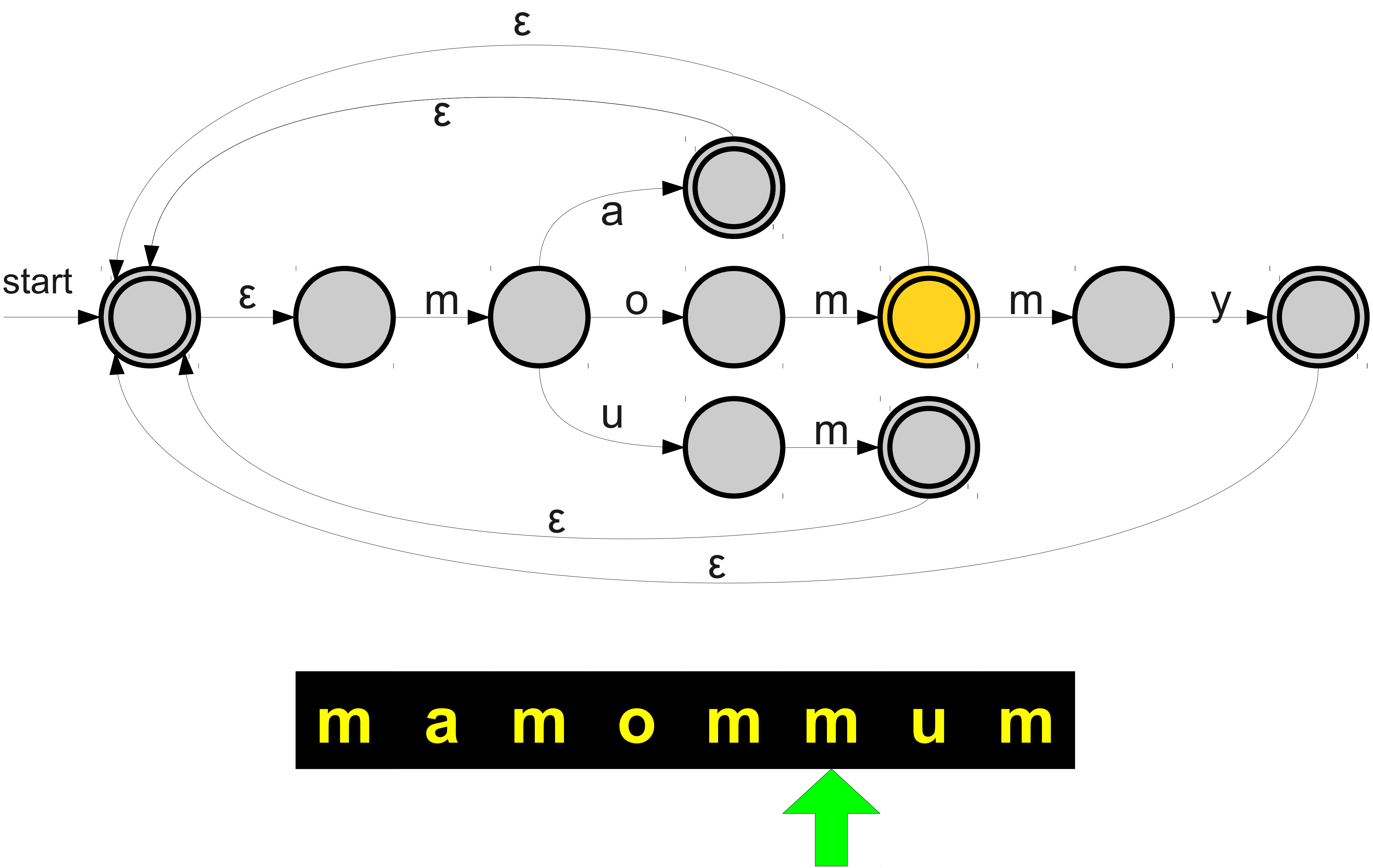
$L = \{$ ma, mom, mommy, mum $\}$





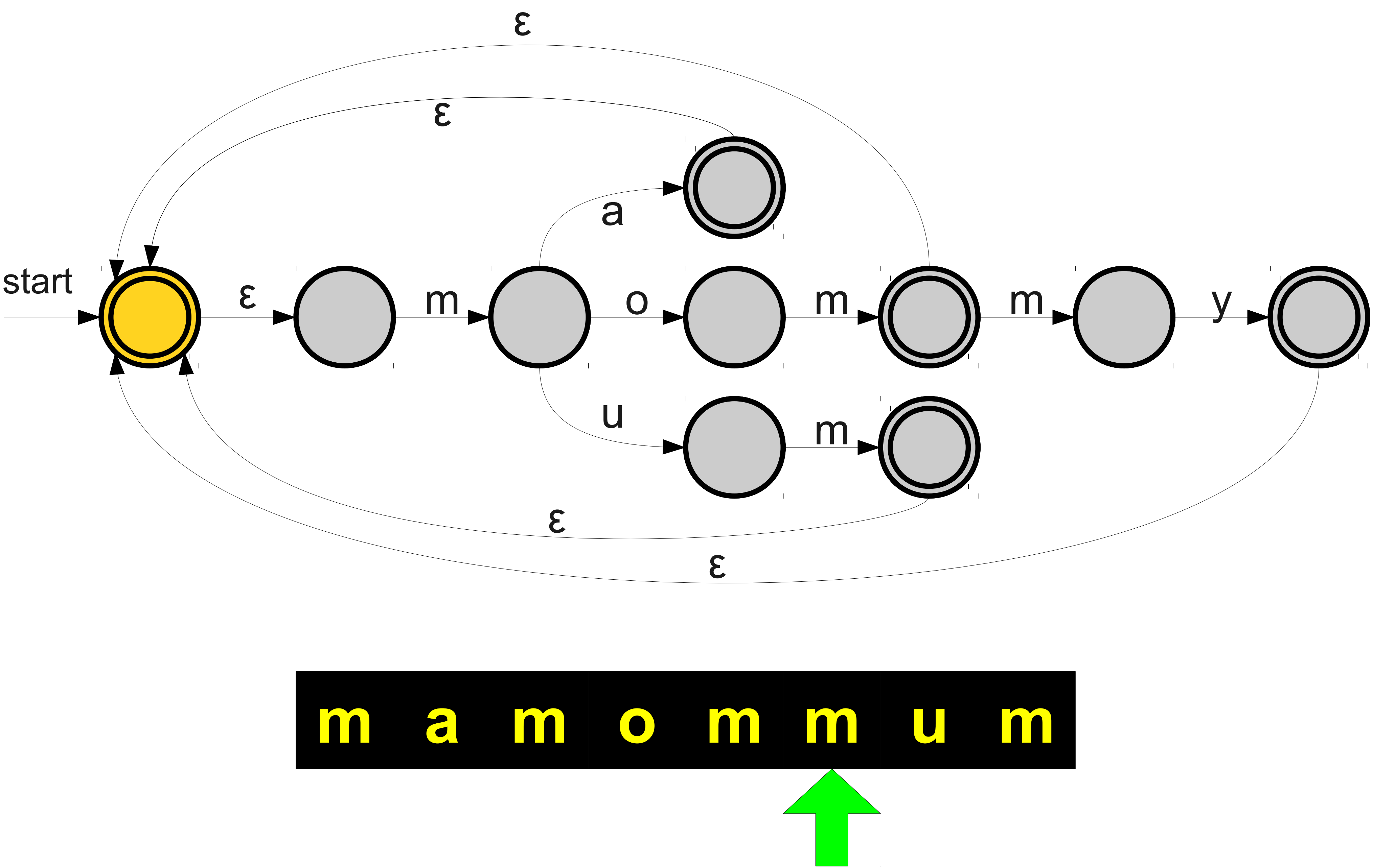
Kleene Star in Action

$$L = \{ \text{ma, mom, mommy, mum} \}$$



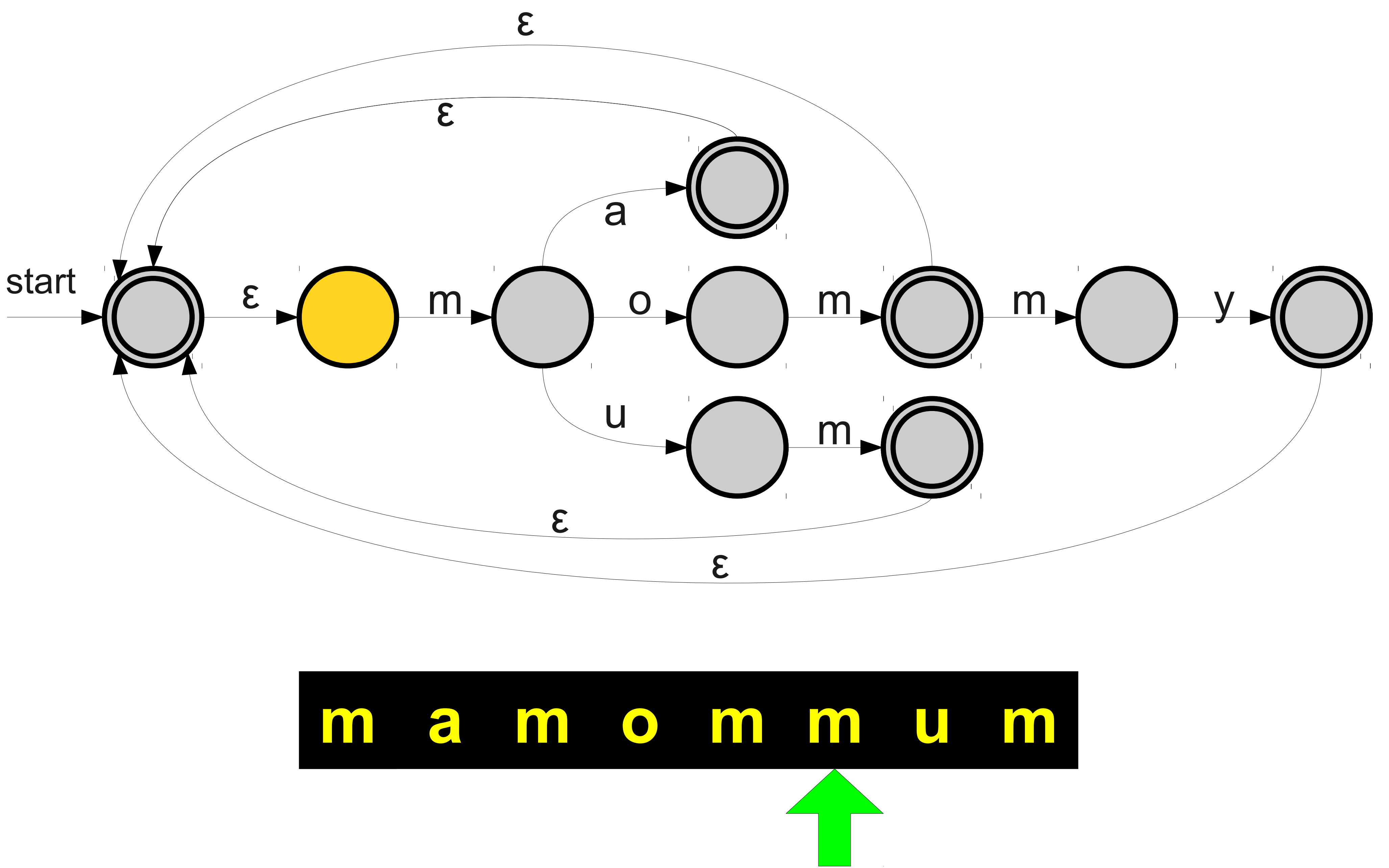
Kleene Star in Action

$L = \{ \text{ma, mom, mommy, mum} \}$



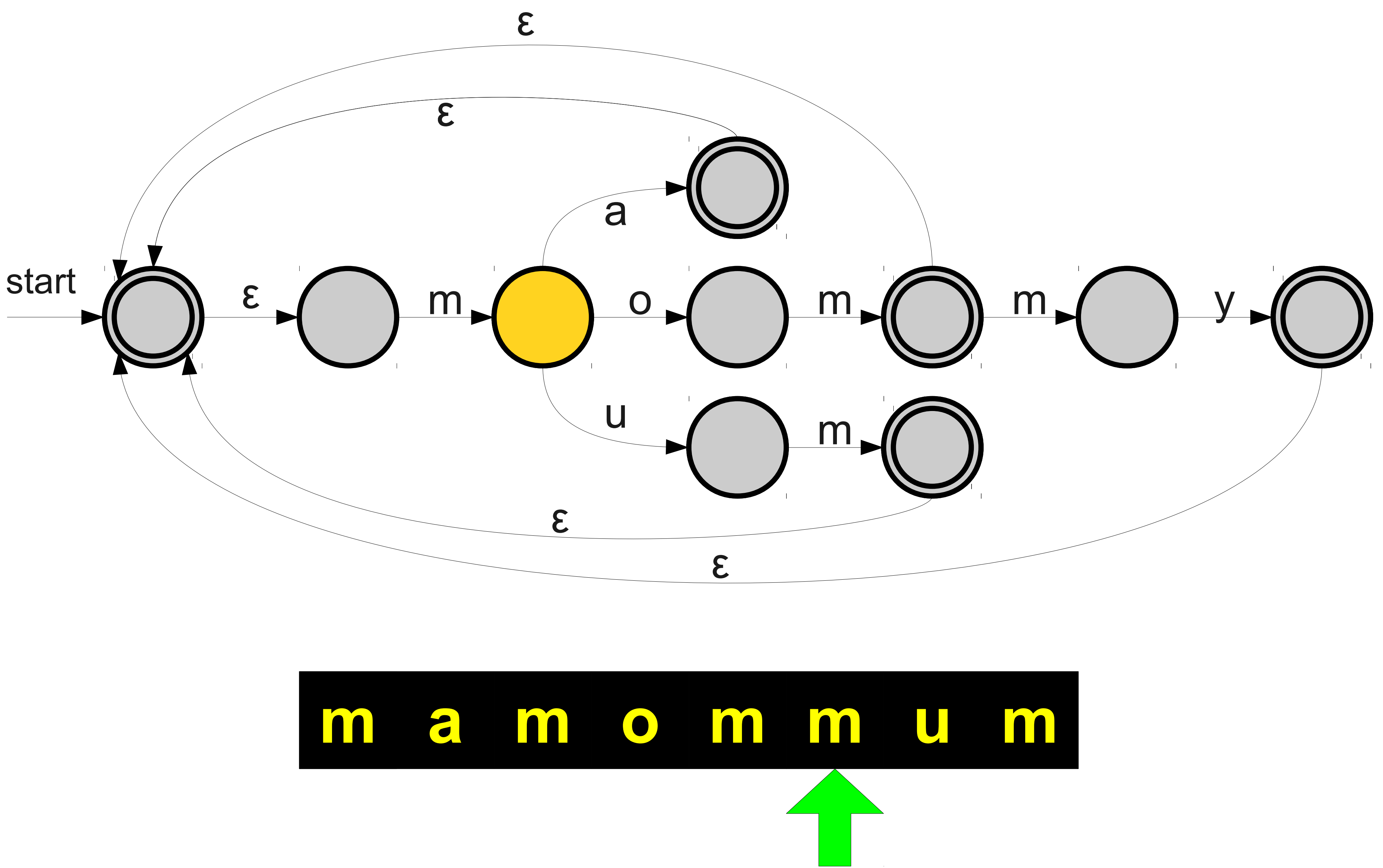
Kleene Star in Action

$L = \{ \text{ma}, \text{mom}, \text{mommy}, \text{mum} \}$



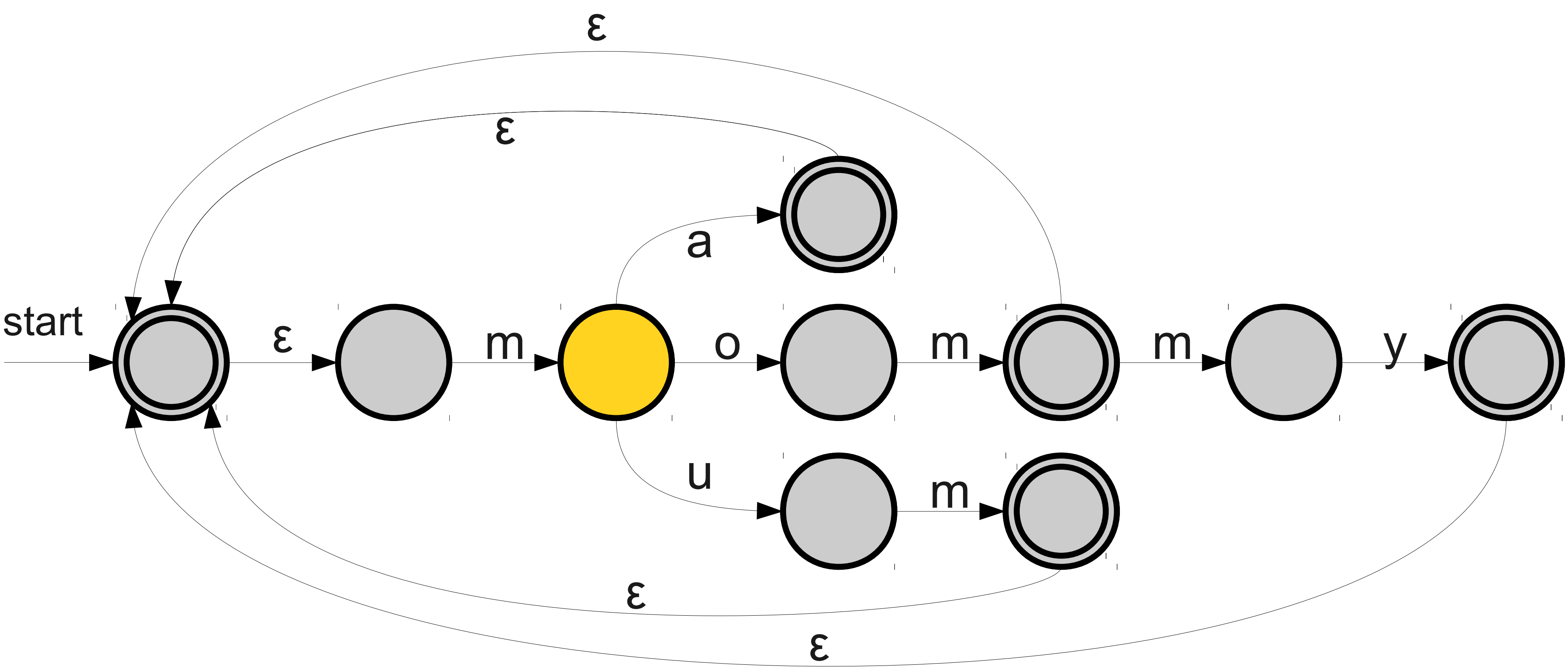
Kleene Star in Action

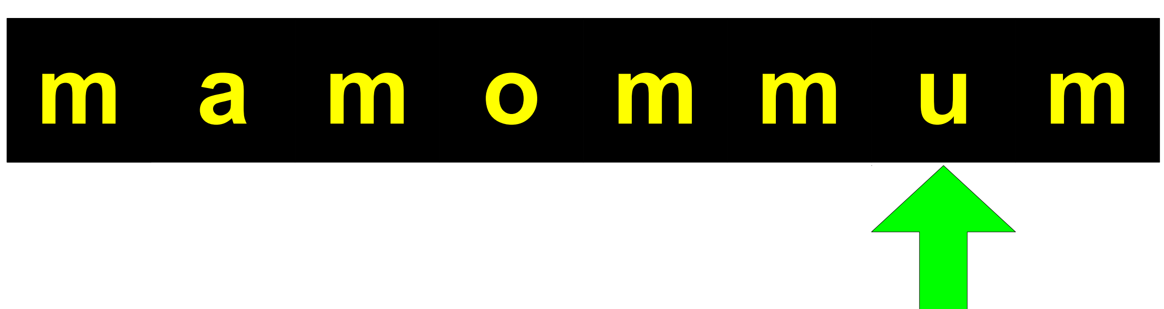
$L = \{ \text{ma, mom, mommy, mum} \}$



Kleene Star in Action

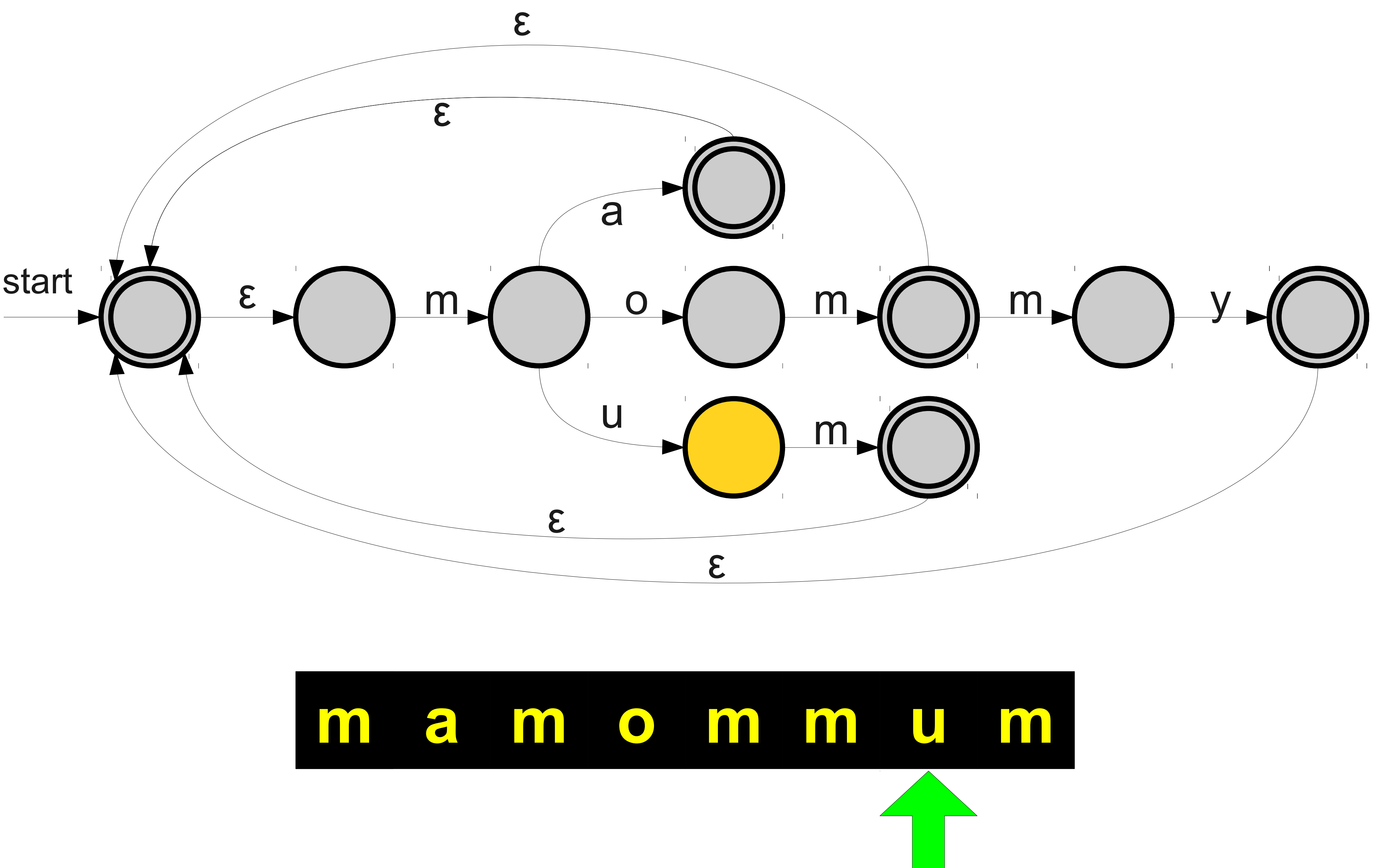
$L = \{$ ma, mom, mommy, mum $\}$





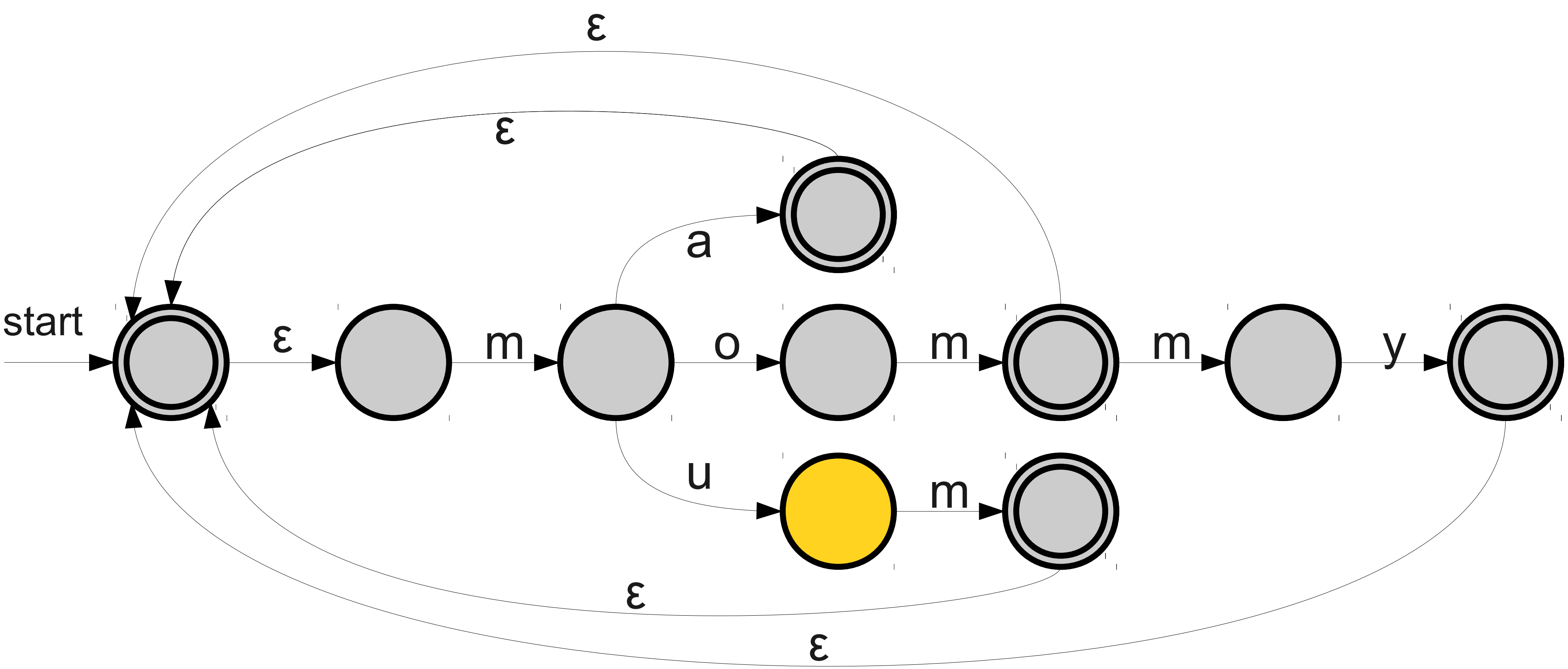
Kleene Star in Action

$L = \{$ ma, mom, mommy, mum $\}$



Kleene Star in Action

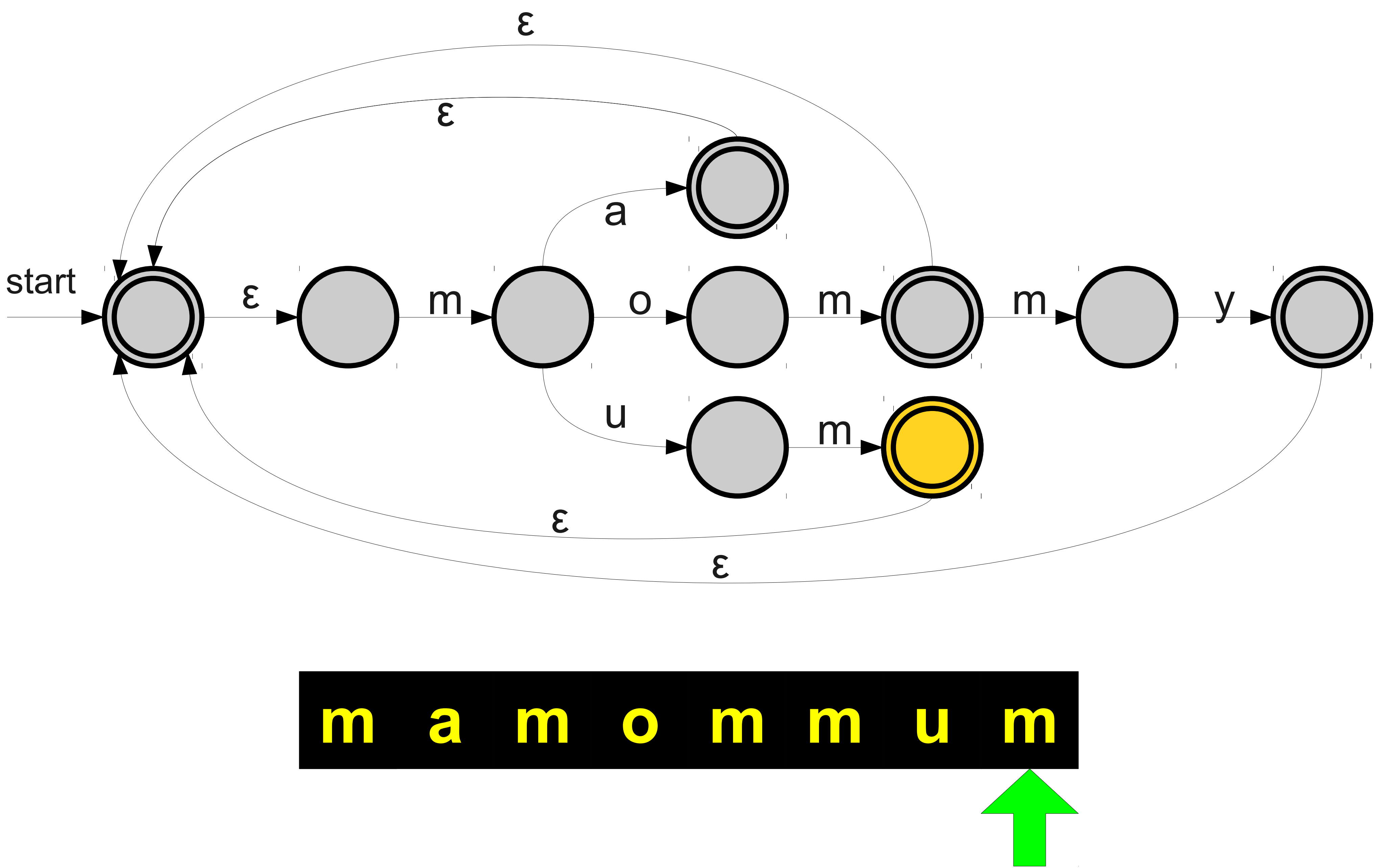
L = { ma, mom, mommy, mum }



m a m o m m u m

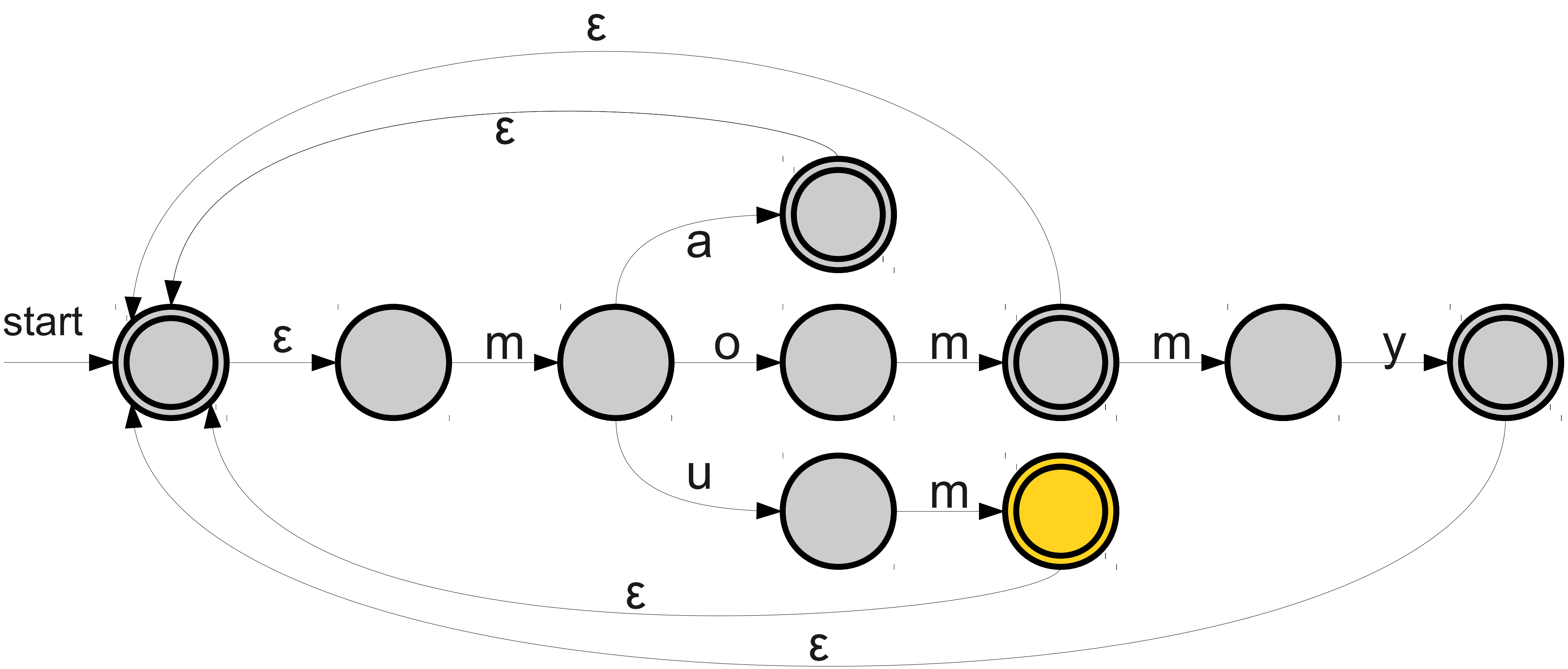
Kleene Star in Action

$L = \{$ ma, mom, mommy, mum $\}$



Kleene Star in Action

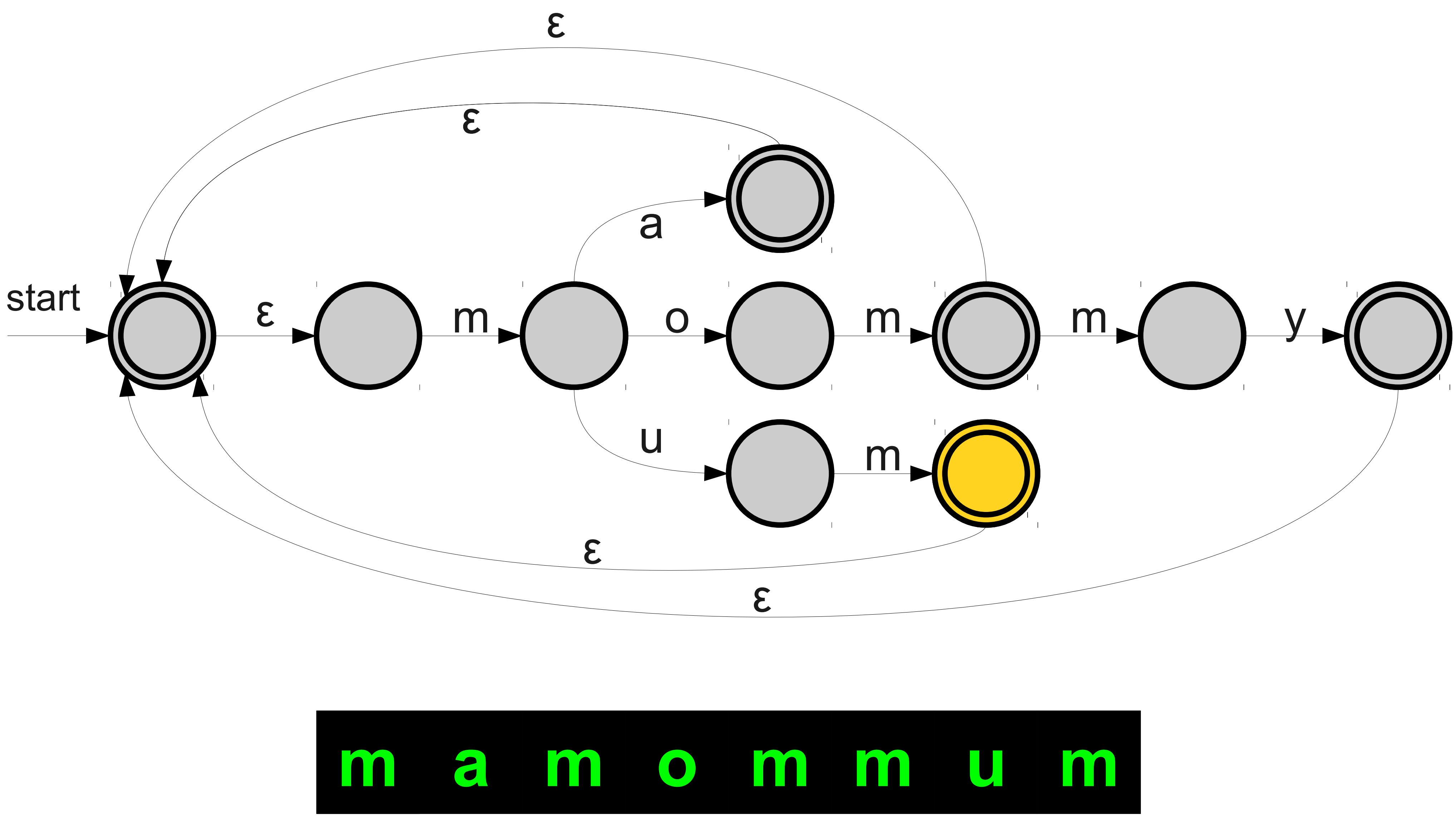
$L = \{\ \text{ma, mom, mommy, mum}\ \}$



m	a	m	o	m	m	u	m

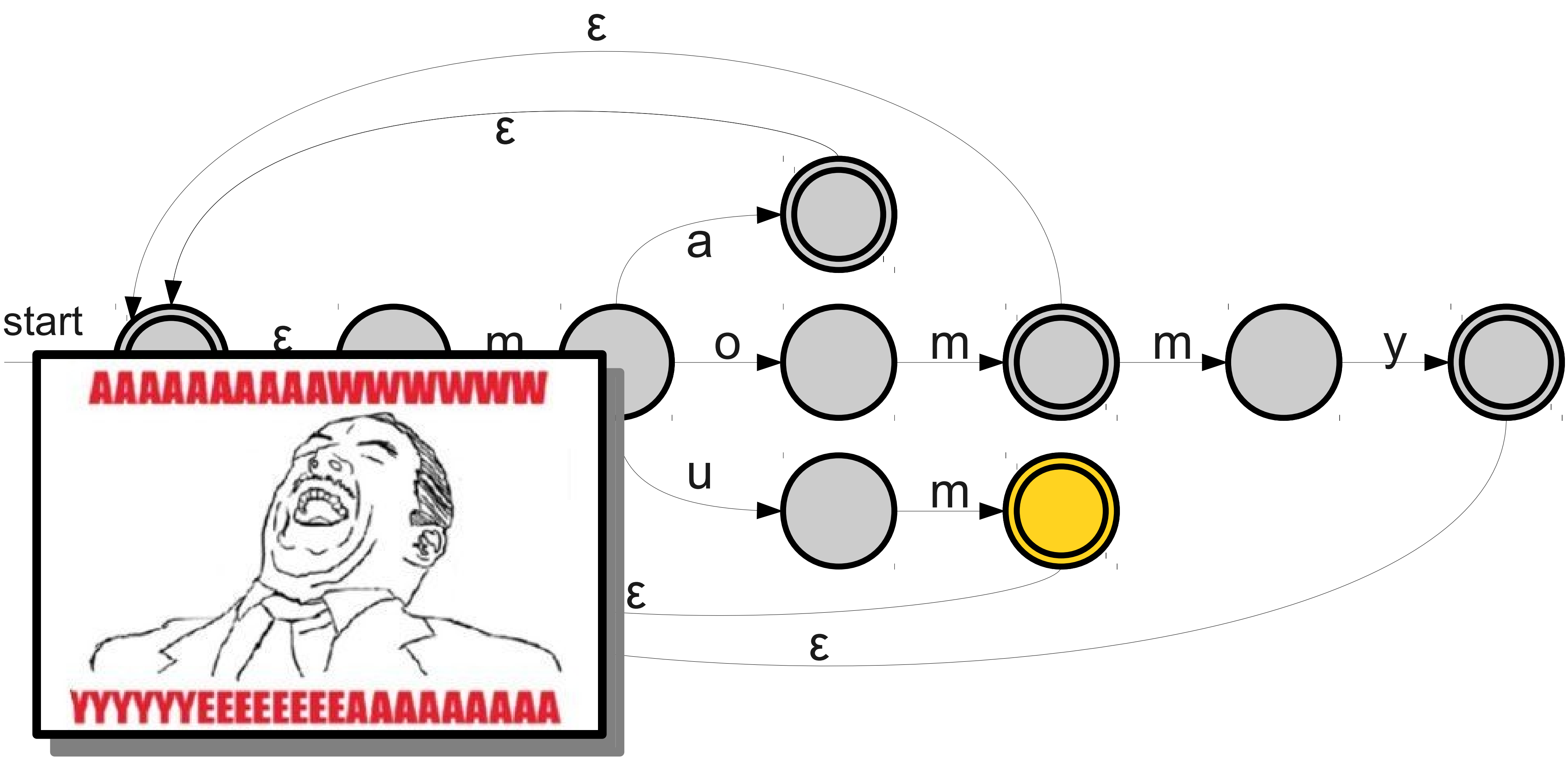
Kleene Star in Action

L = { ma, mom, mommy, mum }



Kleene Star in Action

$L = \{$ ma, mom, mommy, mum $\}$



m a m o m m u m

Summary

- NFAs are a powerful type of automaton that allows for **nondeterministic** choices.
- NFAs can also have **ε-transitions** that move from state to state without consuming any input.
- The **subset construction** shows that NFAs are not more powerful than DFAs, because any NFA can be converted into a DFA that accepts the same language.
- The union, intersection, complement, concatenation, and Kleene closure of regular languages are all 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 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.)

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, $\boldsymbol{R_1 R_2}$ 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 is

$$(R)$$

$$R^*$$

$$R_1R_2$$

$$R_1 \mid R_2$$

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

Worthwhile activity: Apply this recursive definition to

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

and see what you get.

Regular Expressions are Awesome

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

(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 $\}$

(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 $\}$

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

11011100101
0000
11111011110011111

Regular Expressions are Awesome

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

Regular Expressions are Awesome

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

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

Regular Expressions are Awesome

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

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

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

Regular Expressions are Awesome

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

Regular Expressions are Awesome

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

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

Regular Expressions are Awesome

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

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

Regular Expressions are Awesome

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

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

0000
1010
1111
1000

Regular Expressions are Awesome

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

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

0000
1010
1111
1000

Regular Expressions are Awesome

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

$$(0|1)^4$$

0000
1010
1111
1000

Regular Expressions are Awesome

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

$(0|1)^4$

0000
1010
1111
1000

Regular Expressions are Awesome

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

Regular Expressions are Awesome

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

1*(0 | ε)1*

Regular Expressions are Awesome

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

$$1^*(0 \mid \varepsilon)1^*$$

Regular Expressions are Awesome

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

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 | ε)1*

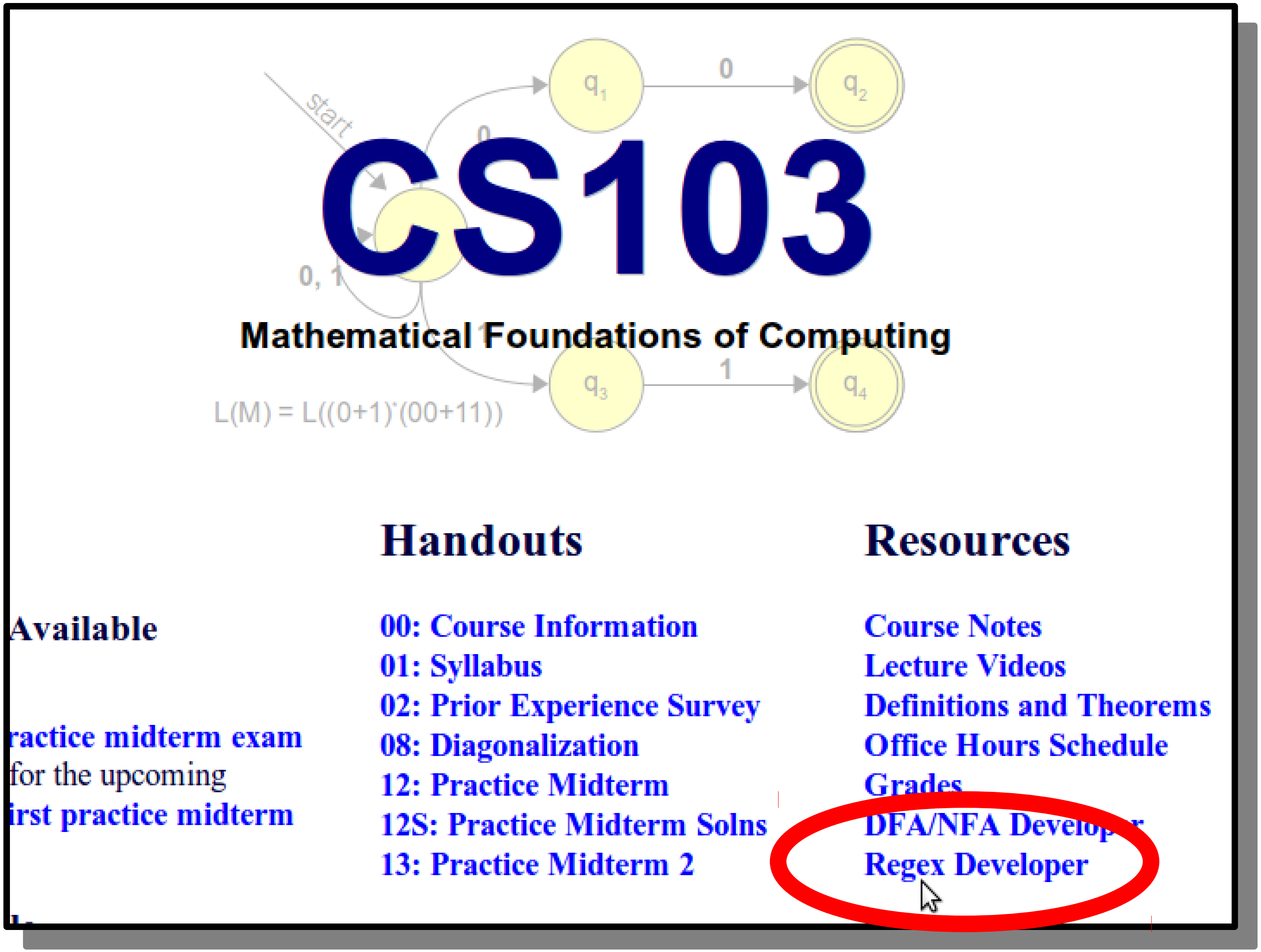
11110111
111111
0111
0

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


start
0
0, 1
1
L(M) = L((0+1)*(00+11))
CS103
Mathematical Foundations of Computing
Handouts
Resources
Available
00: Course Information
01: Syllabus
02: Prior Experience Survey
08: Diagonalization
12: Practice Midterm
12S: Practice Midterm Solns
13: Practice Midterm 2
ractice midterm exam
for the upcoming
irst practice midterm
Course Notes
Lecture Videos
Definitions and Theorems
Office Hours Schedule
Grades
Regex Developer