Learning Symbolic Automata

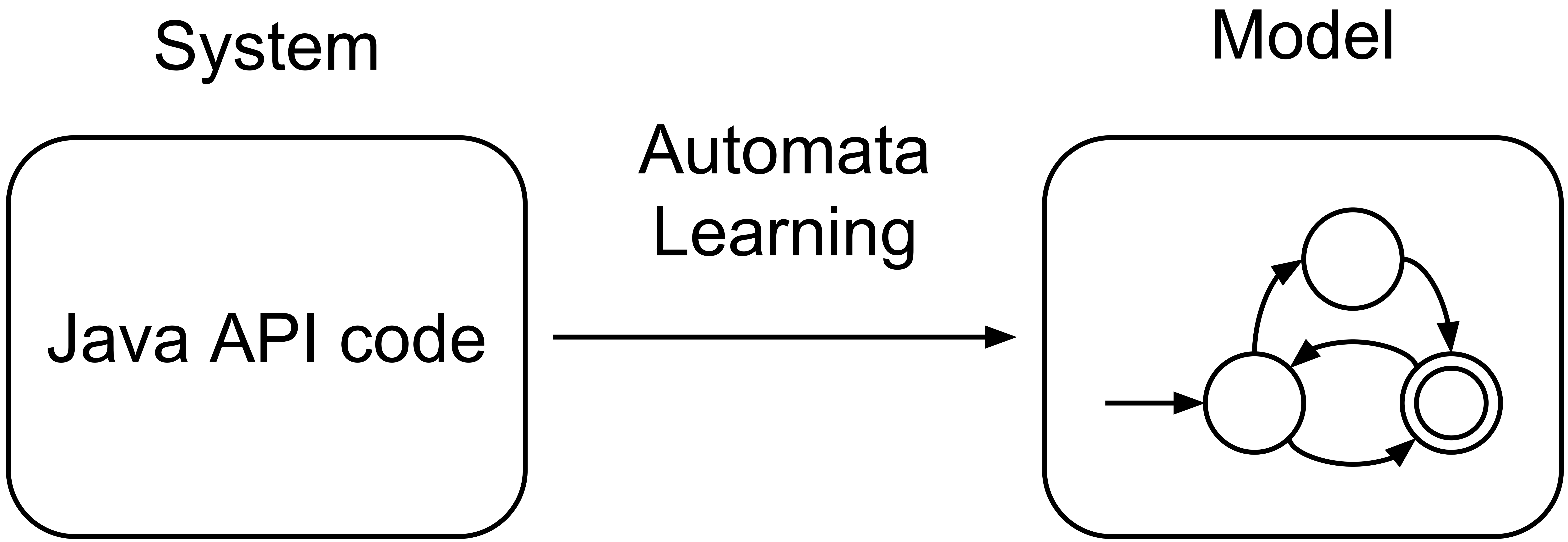
Samuel Drews

Loris D'Antoni

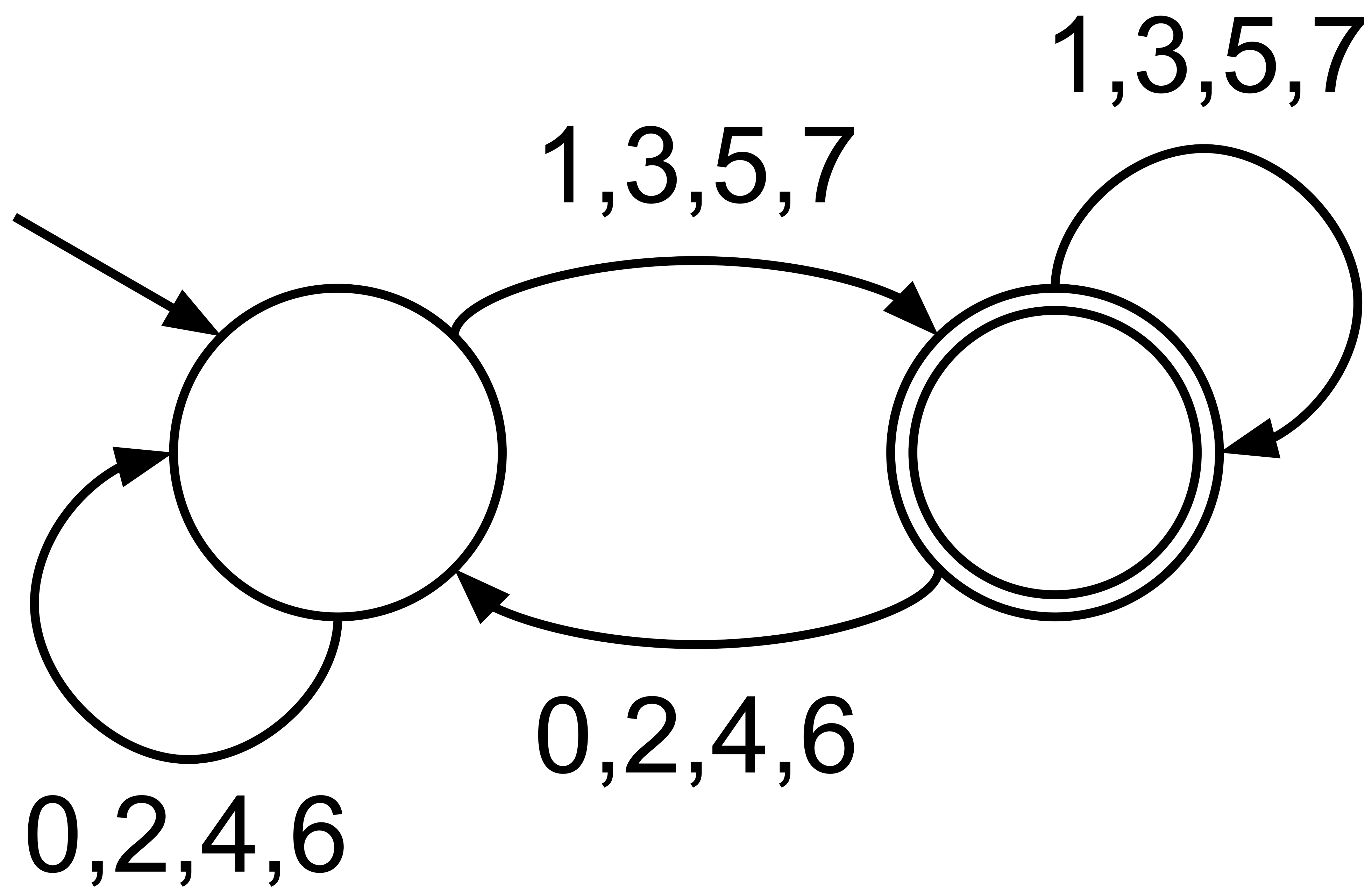
University of Wisconsin-Madison

madPL

Motivation



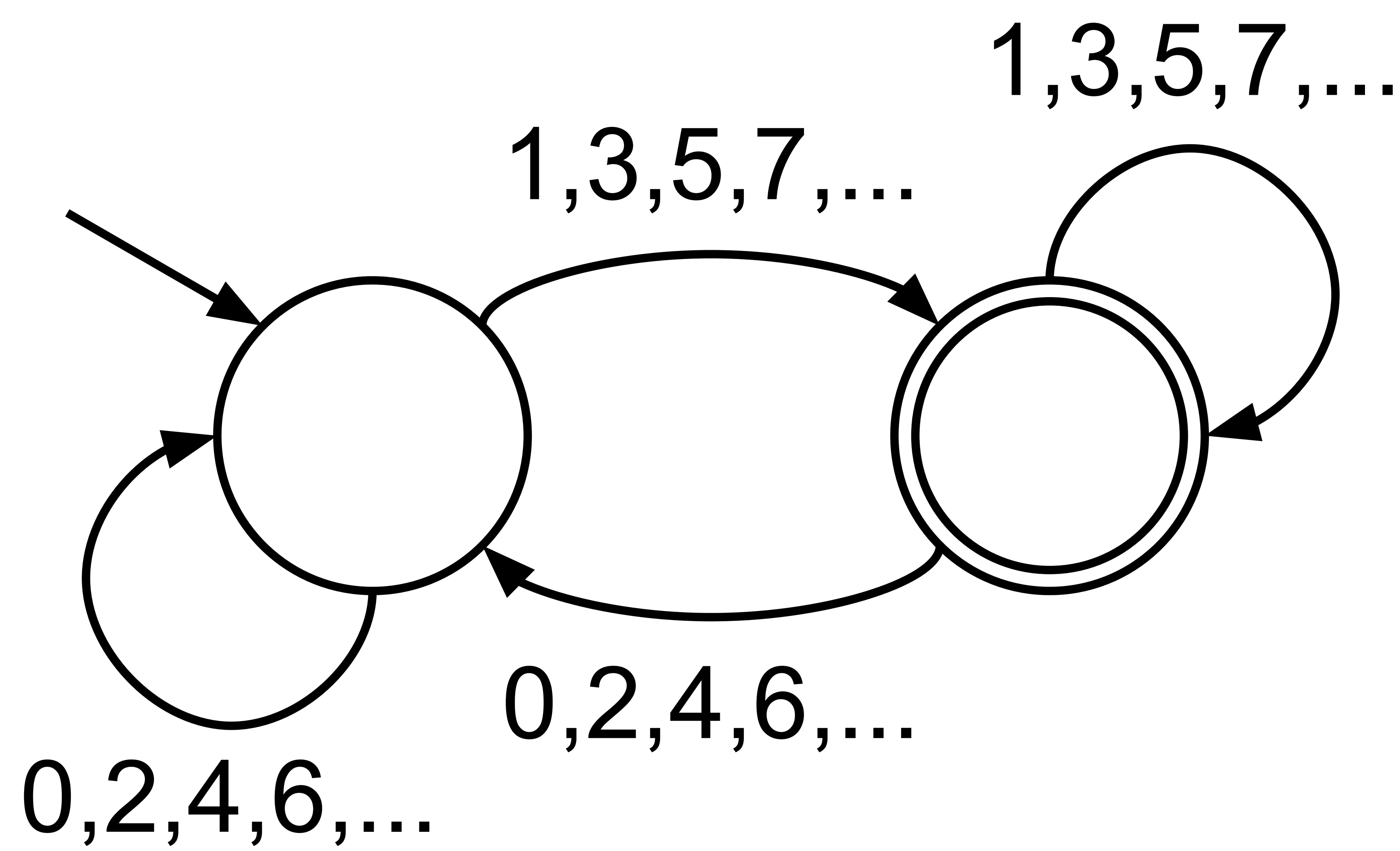
Classic Automata



Alphabet $\Sigma = \{0,1,2,3,4,5,6,7\}$

Transition $\delta : Q \times \Sigma \rightarrow Q$

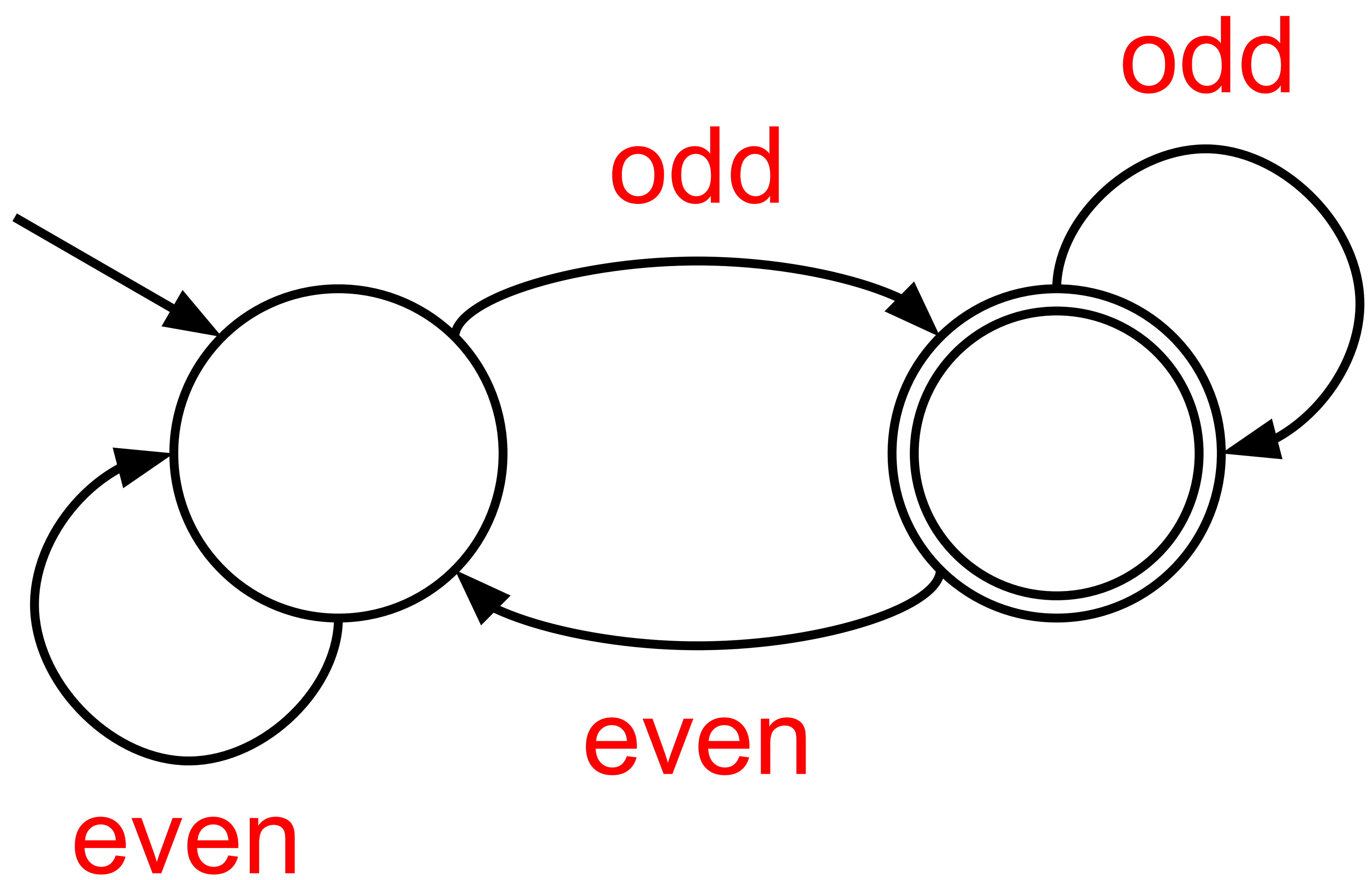
Classic Automata



Alphabet $\Sigma = \{0,1,2,3,4,5,6,7,...\}$

Transition $\delta : Q \times \Sigma \rightarrow Q$

Symbolic Automata

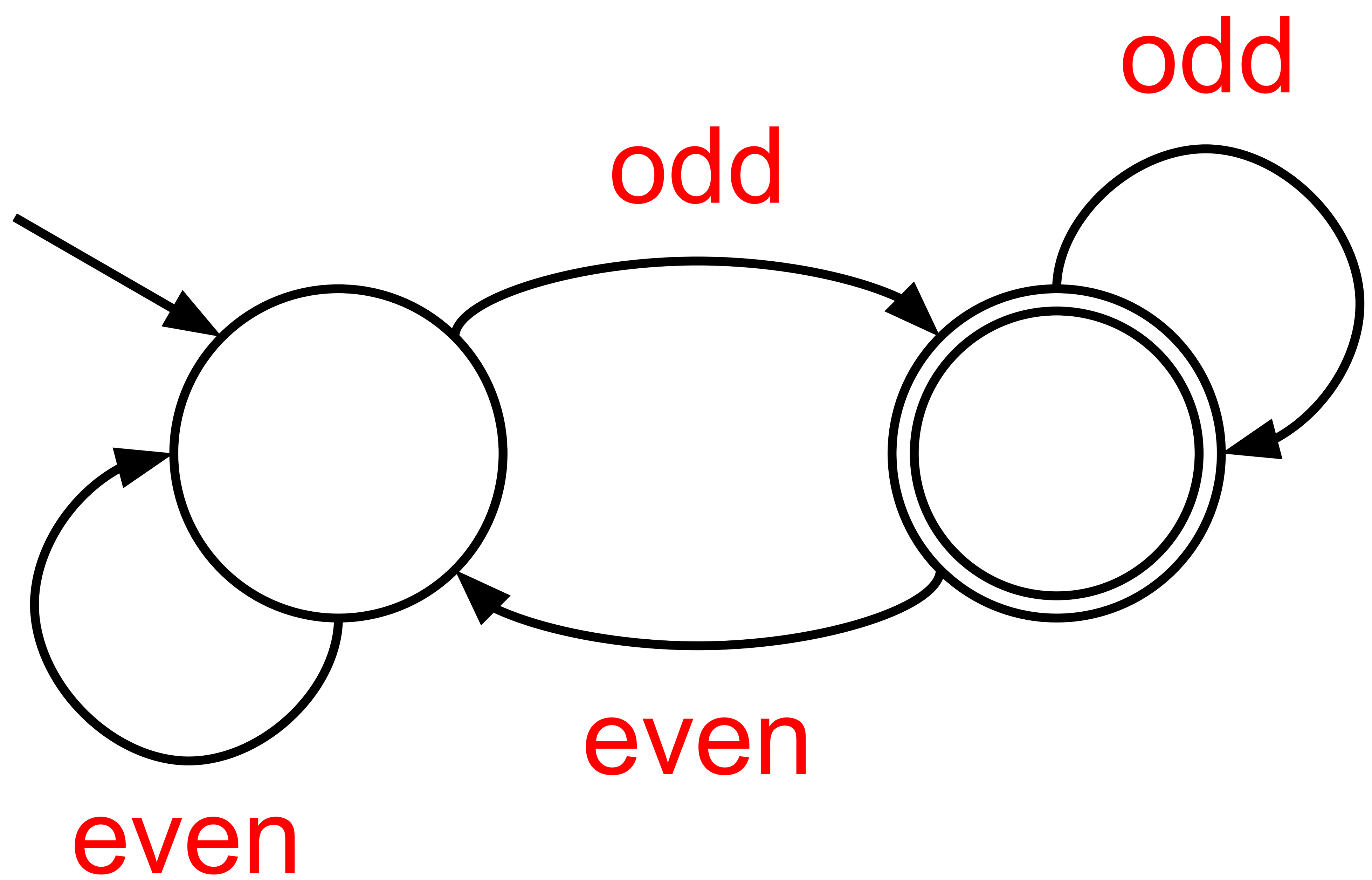


Alphabet $\Sigma = \{0,1,2,3,4,5,6,7,...\}$

Boolean Algebra $BA = \{\bot, odd, even, \top\}$

Transition $\delta : Q \times BA \rightarrow Q$

Symbolic Automata



Boolean Algebra

$\varphi \in BA \rightarrow \neg\varphi \in BA$

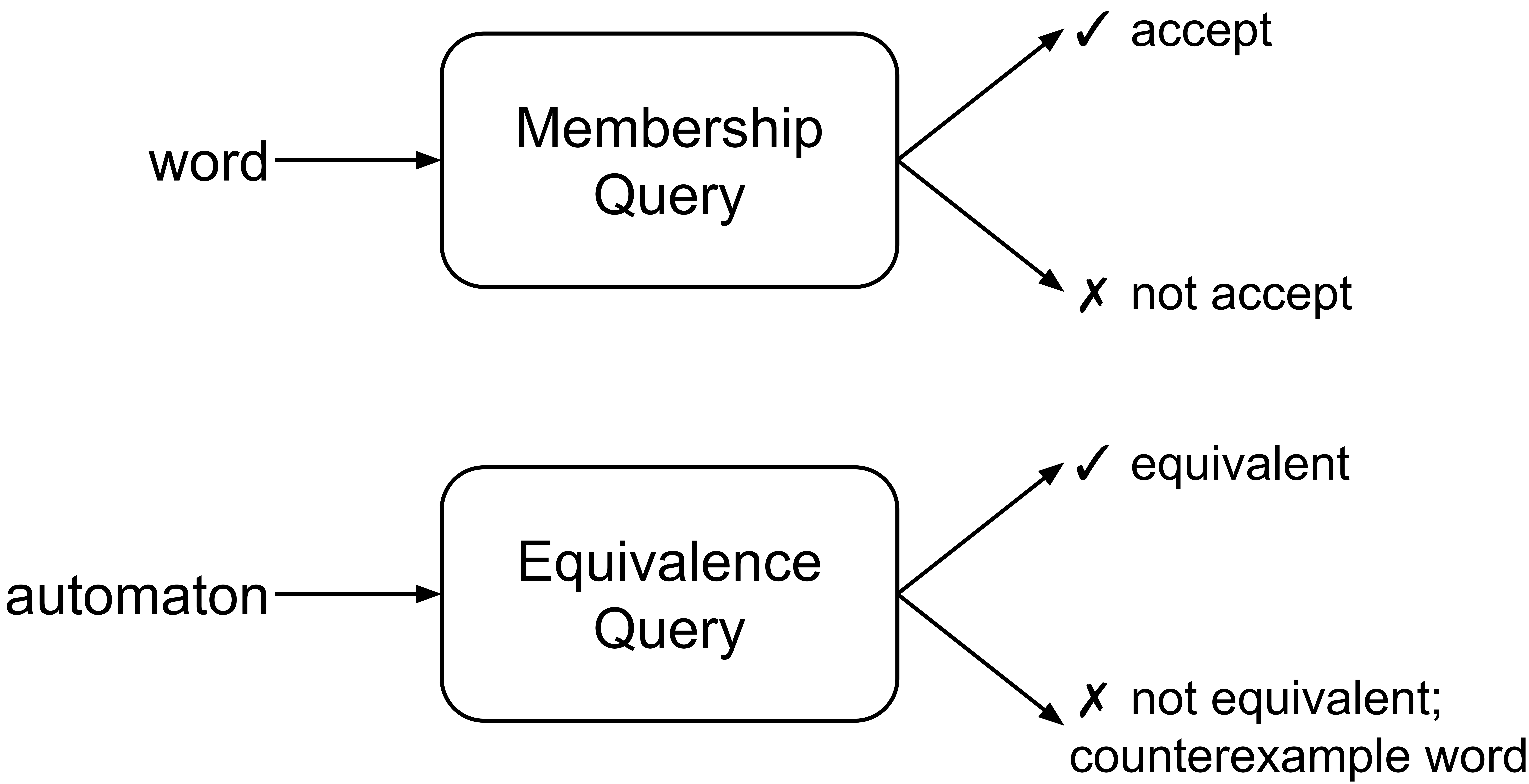
$\varphi, \psi \in BA$
$\rightarrow \varphi \wedge \psi \in BA$

Alphabet $\Sigma = \{0,1,2,3,4,5,6,7,...\}$

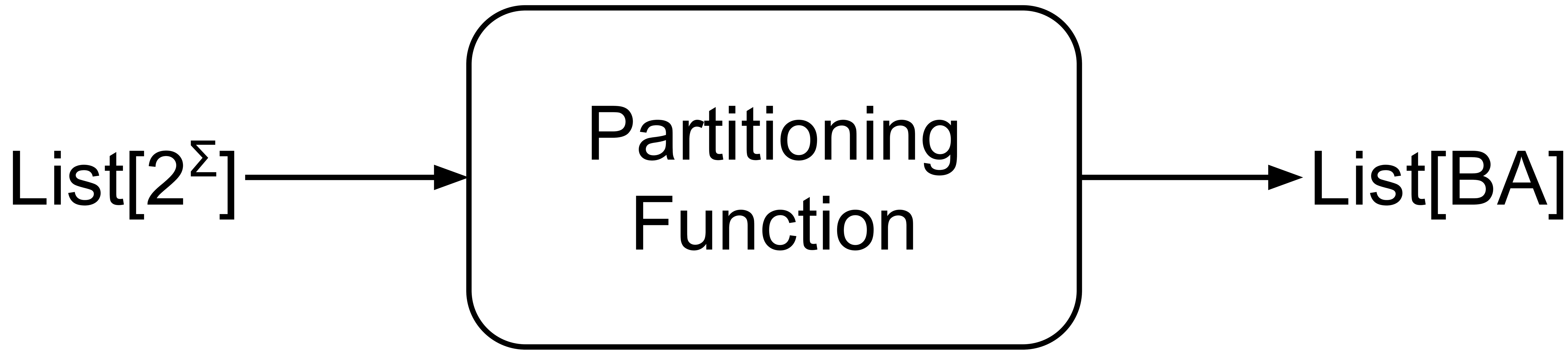
Boolean Algebra $BA = \{\bot, odd, even, \top\}$

Transition $\delta : Q \times BA \rightarrow Q$

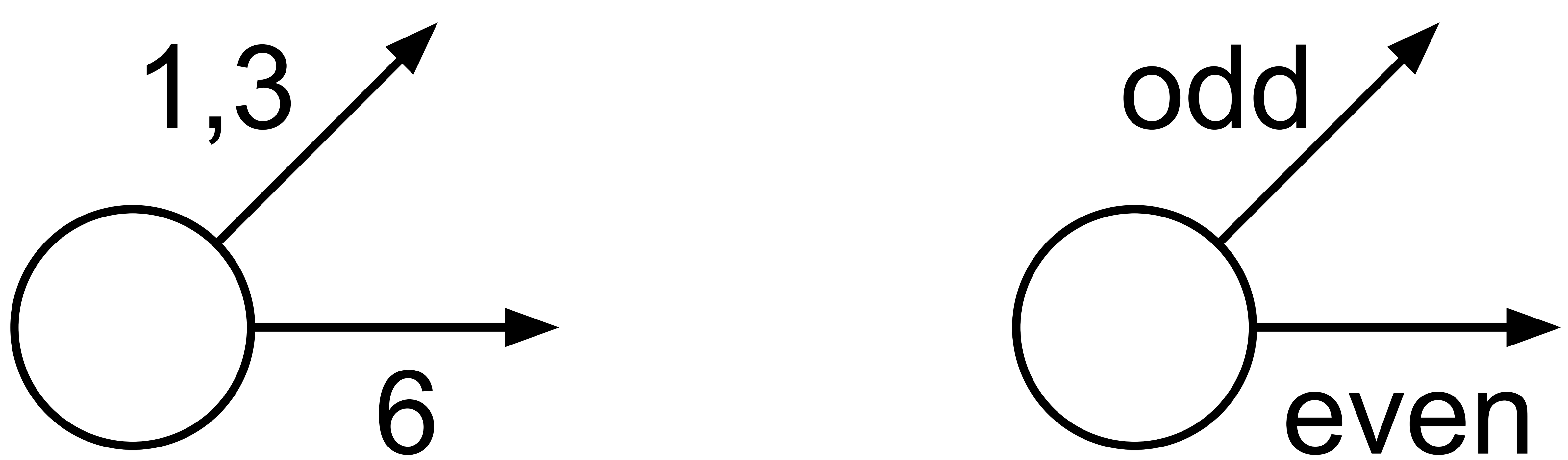
Λ* Oracle Queries



Λ^* Partitioning Function



Ex:



P([{1,3},{6}]) = [odd, even]

Angluin's L* (classic automata)

Λ^*
Sparse
Membership
Queries
Intermediary
Automaton
Partitioning
Function
Refine
Observation
Table
Conjecture
Symbolic
Automaton
Equivalence Query
+ Counterexample

Anatomy of the Observation Table

	ε	0
ε	✓	✓
5	✗	✗
5,0	✗	✓
0	✓	✓
5,0,0	✓	✓

Rows: strings that lead to states (representatives above divider)

Columns: suffixes that tell states apart

Body: whether automaton accepts word

Anatomy of the Observation Table

	ε	0
ε	✓	✓
5	✗	✗
5,0	✗	✓
0	✓	✓
5,0,0	✓	✓

Rows: strings that lead to states
(representatives above divider)

Columns: suffixes that tell states apart

Body: whether automaton accepts word

Anatomy of the Observation Table

	ε	0
ε	✓	✓
5	✗	✗
5,0	✗	✓
0	✓	✓
5,0,0	✓	✓

Rows: strings that lead to states
(representatives above divider)

Columns: suffixes that tell states apart

Body: whether automaton accepts word

Anatomy of the Observation Table

	ε	0
ε	✓	✓
5	✗	✗
5,0	✗	✓
0	✓	✓
5,0,0	✓	✓

Rows: strings that lead to states (representatives above divider)

Columns: suffixes that tell states apart

Body: whether automaton accepts word

Anatomy of the Observation Table

	ε	0
ε	✓	✓
5	✗	✗
5,0	✗	✓
0	✓	✓
5,0,0	✓	✓

Rows: strings that lead to states (representatives above divider)

Columns: suffixes that tell states apart

Body: whether automaton accepts word

Anatomy of the Observation Table

	ε	0
ε	✓	✓
5	✗	✗
5,0	✗	✓
0	✓	✓
5,0,0	✓	✓

Rows: strings that lead to states (representatives above divider)

Columns: suffixes that tell states apart

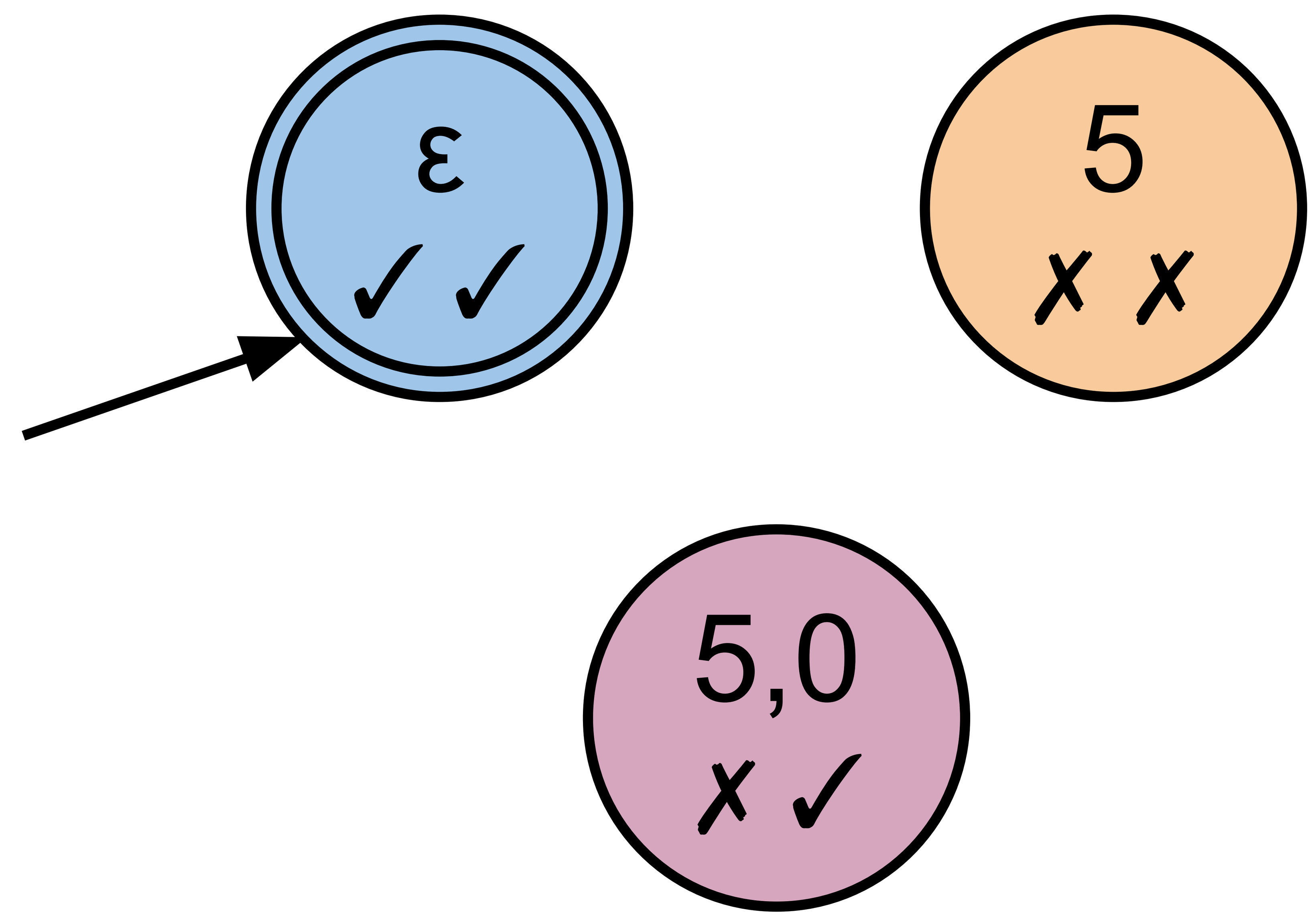
Body: whether automaton accepts word

does not accept 5,0·ε

accepts 5,0·0

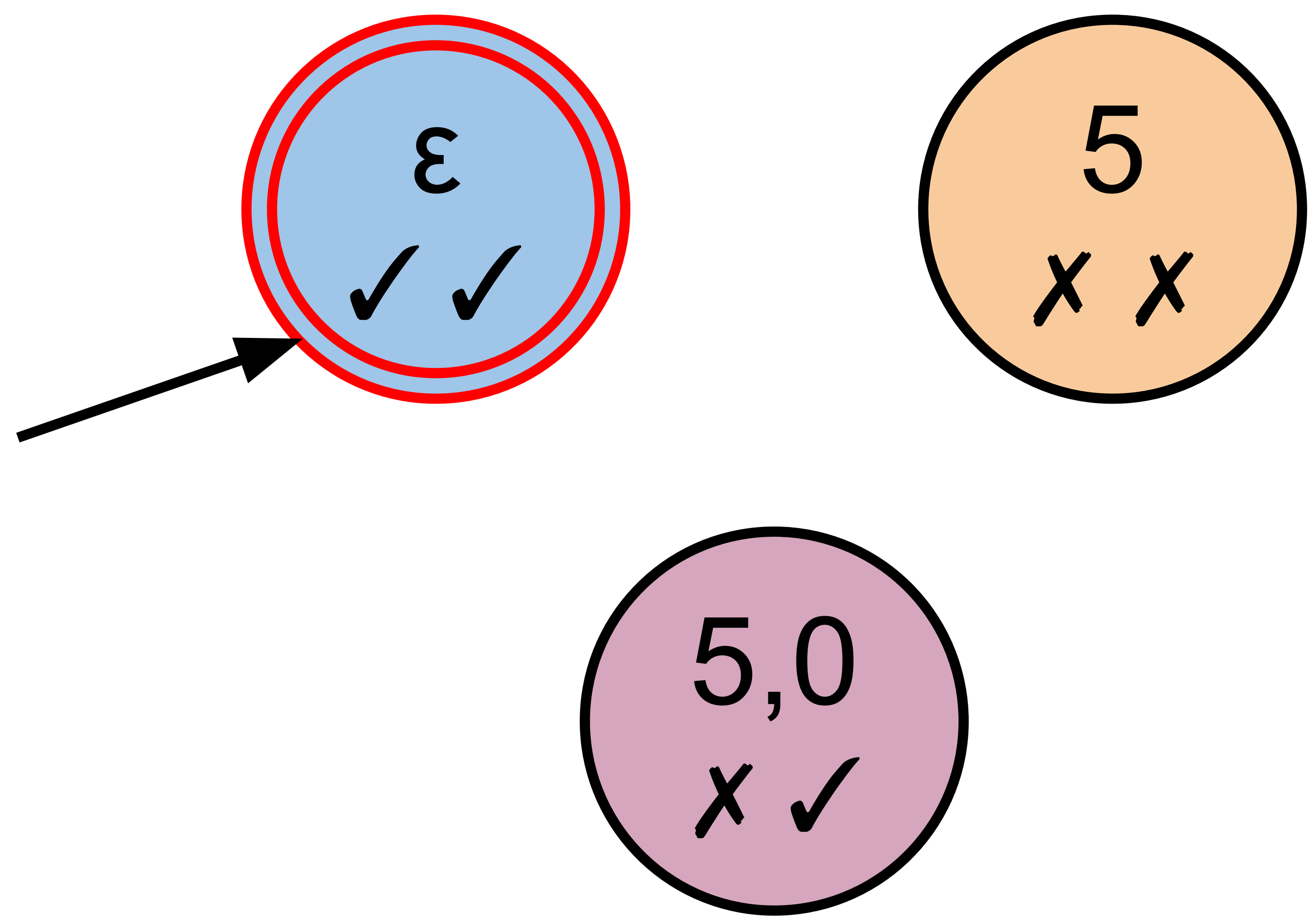
Observation Table to Intermediary Automaton

	ε	0
ε	✓	✓
5	✗	✗
5,0	✗	✓
0	✓	✓
5,0,0	✓	✓



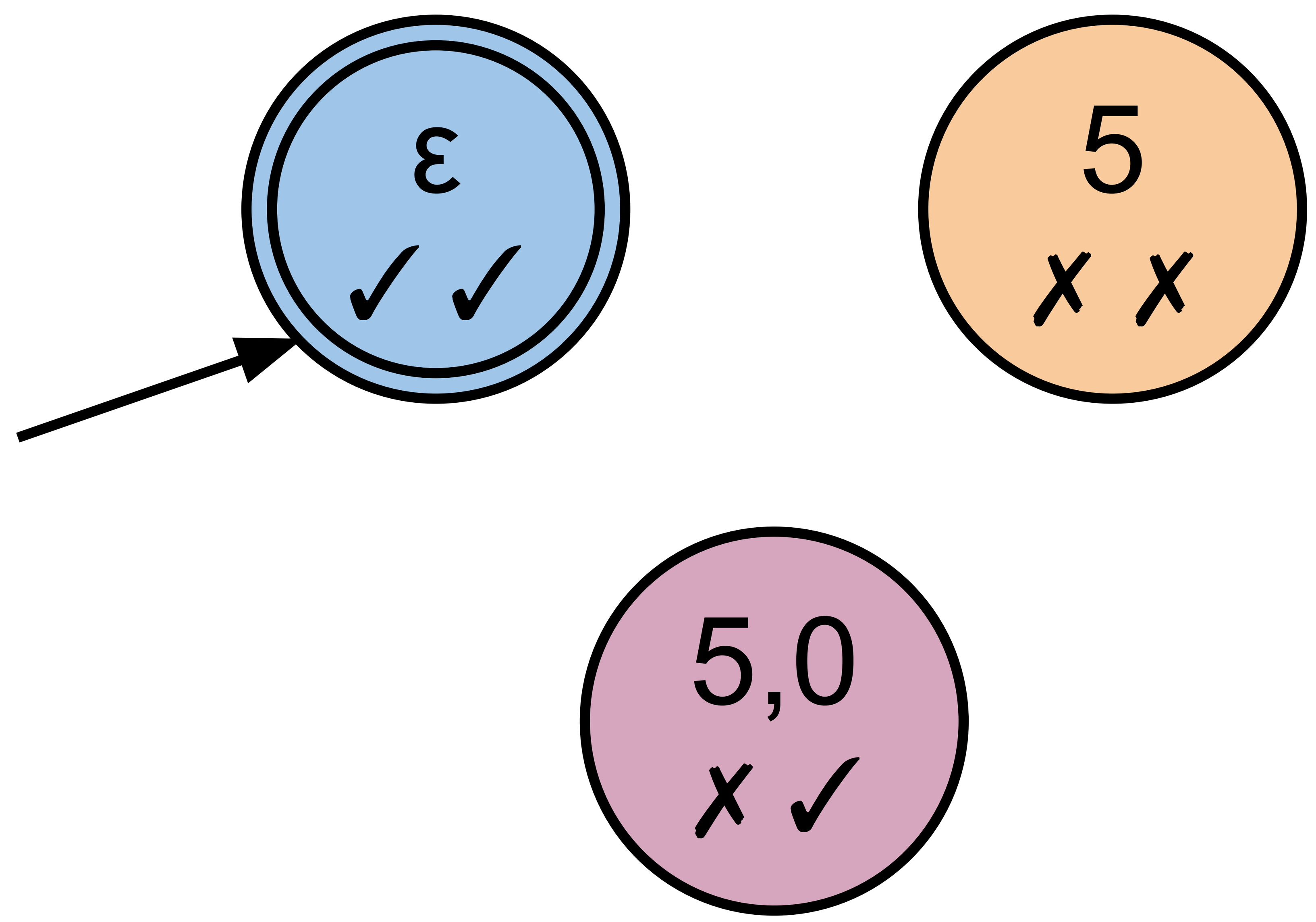
Observation Table to Intermediary Automaton

	ε	0
ε	✓	✓
5	✗	✗
5,0	✗	✓
0	✓	✓
5,0,0	✓	✓



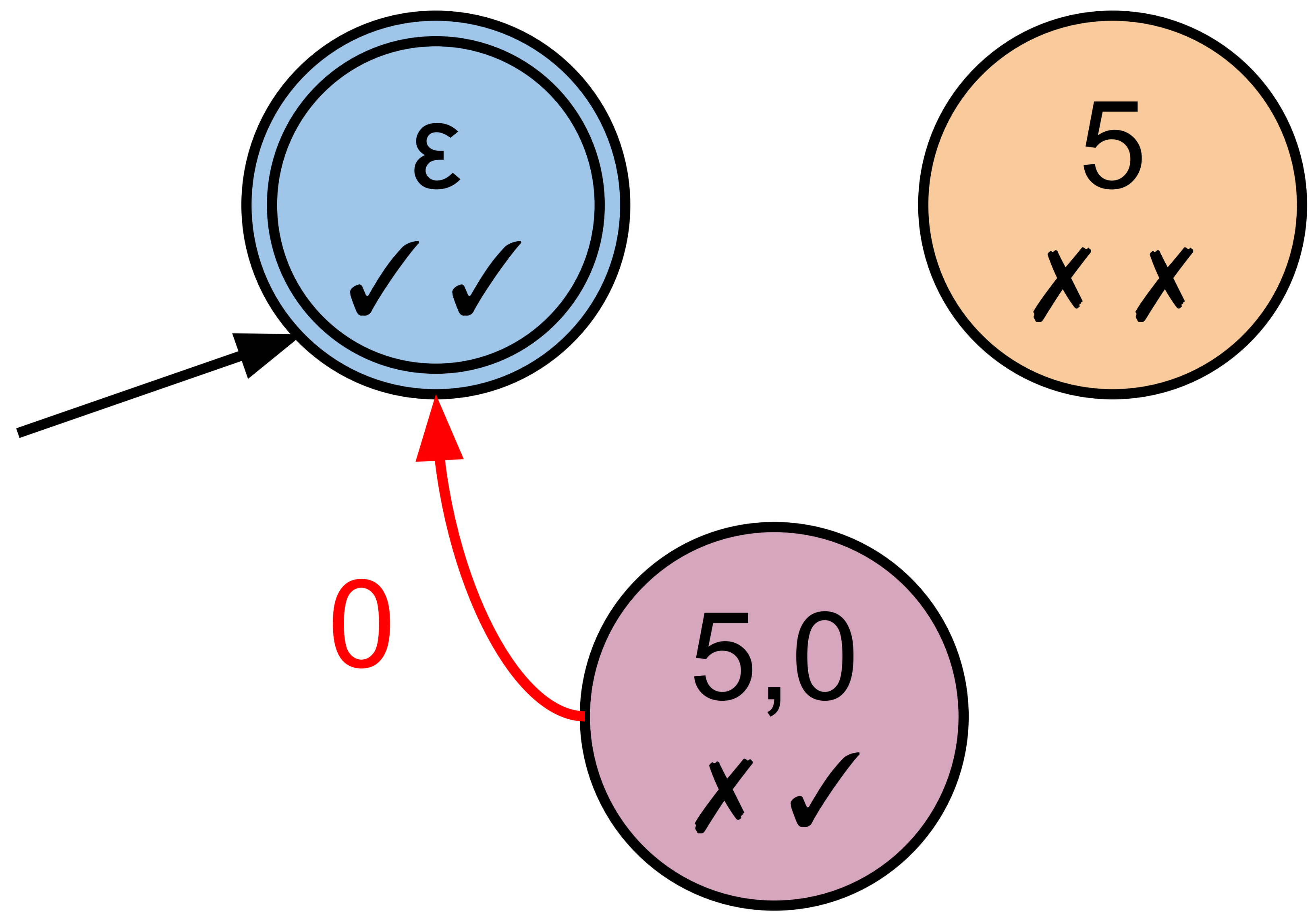
Observation Table to Intermediary Automaton

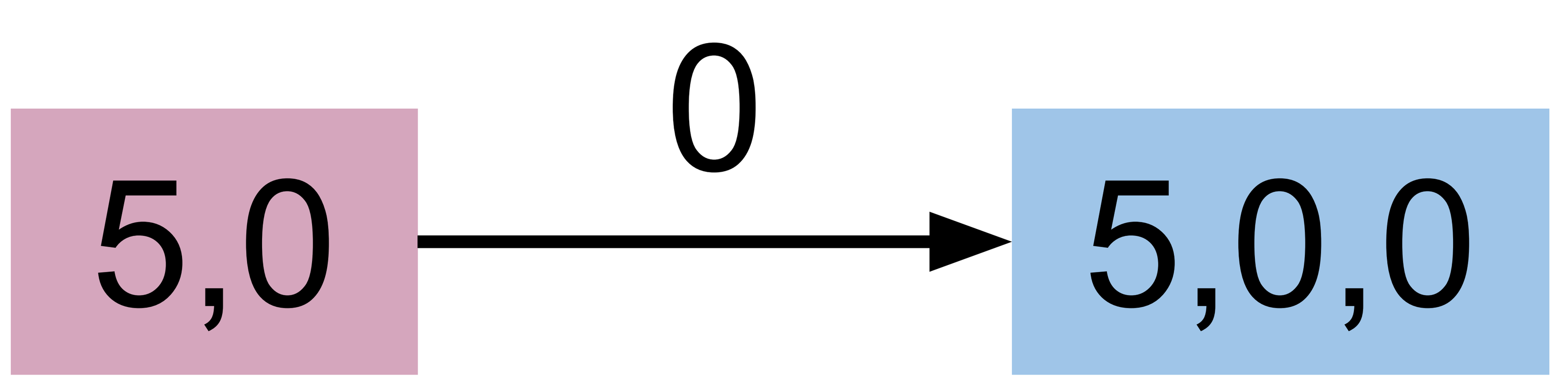
	ε	0
ε	✓	✓
5	✗	✗
5,0	✗	✓
0	✓	✓
5,0,0	✓	✓



Observation Table to Intermediary Automaton

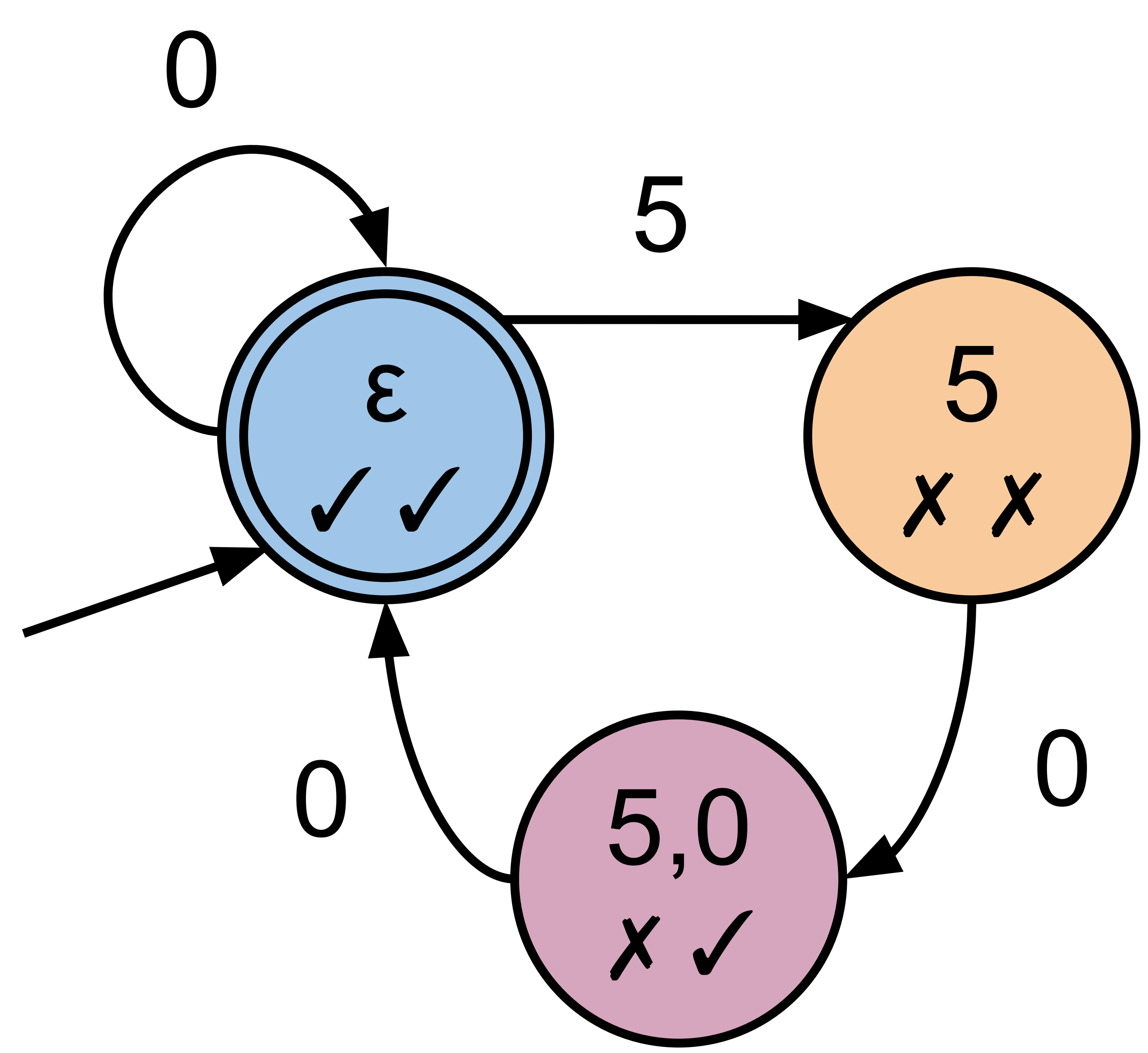
	ε	0
ε	✓	✓
5	✗	✗
5,0	✗	✓
0	✓	✓
5,0,0	✓	✓



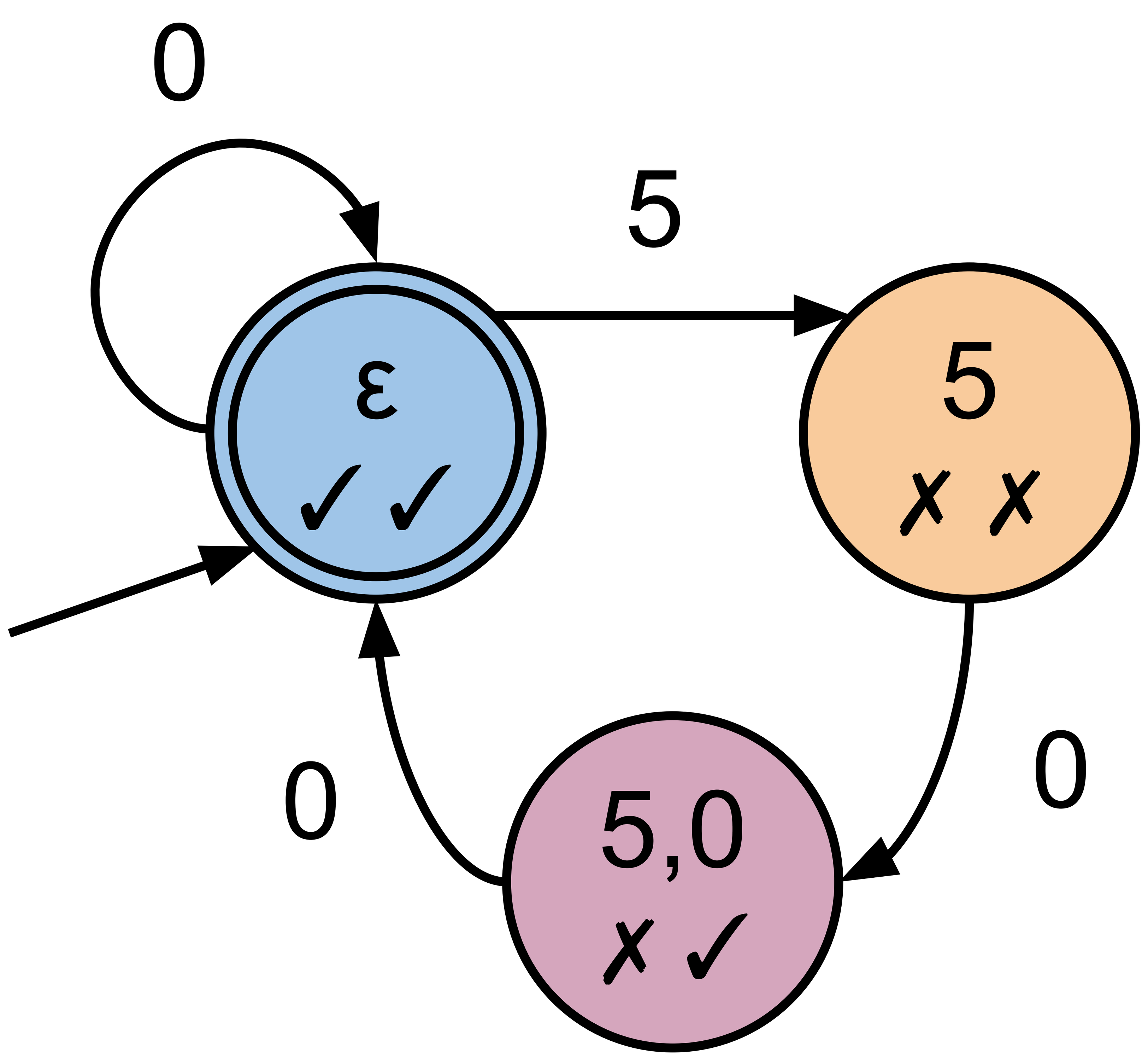


Observation Table to Intermediary Automaton

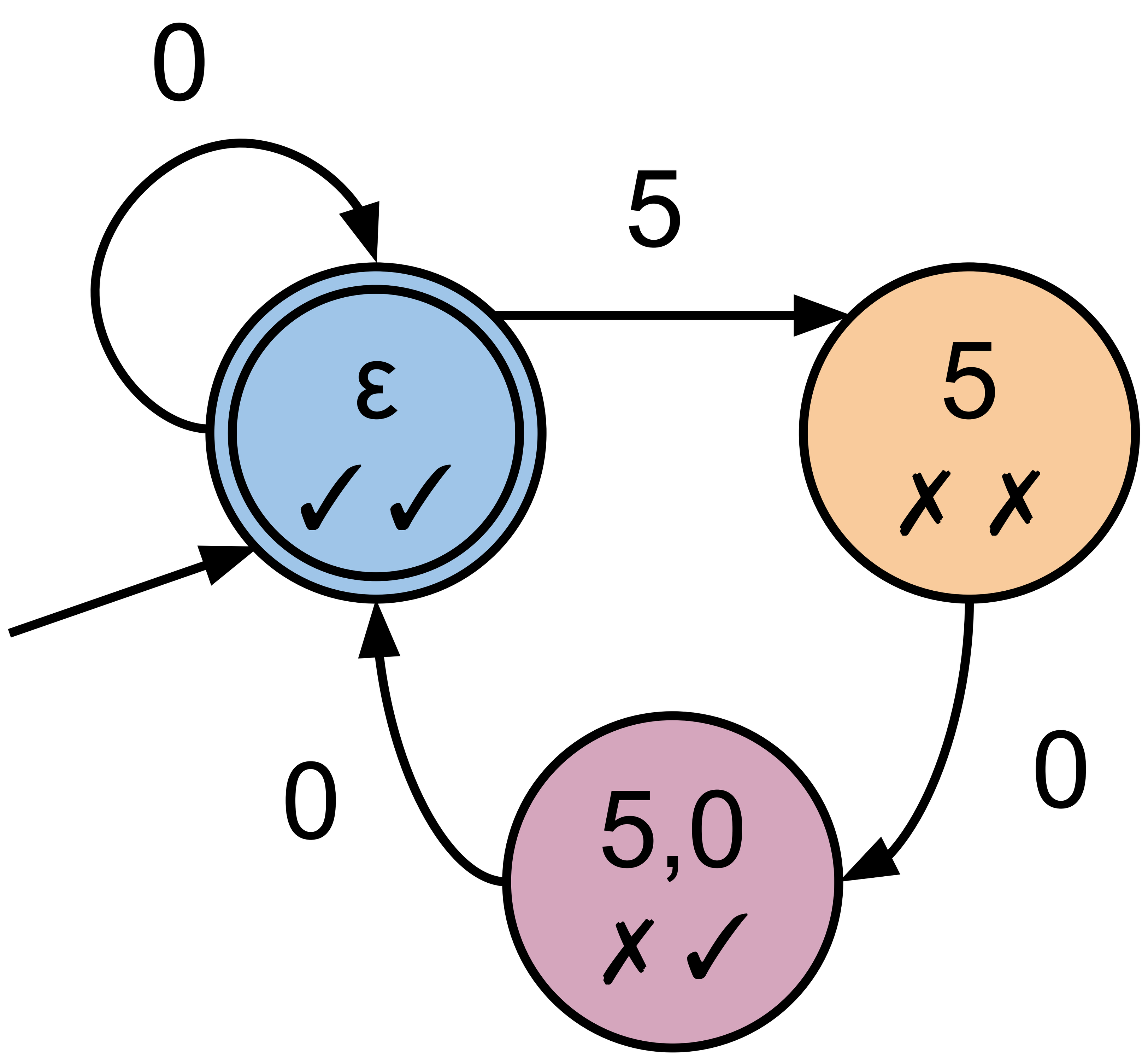
	ε	0
ε	✓	✓
5	✗	✗
5,0	✗	✓
0	✓	✓
5,0,0	✓	✓

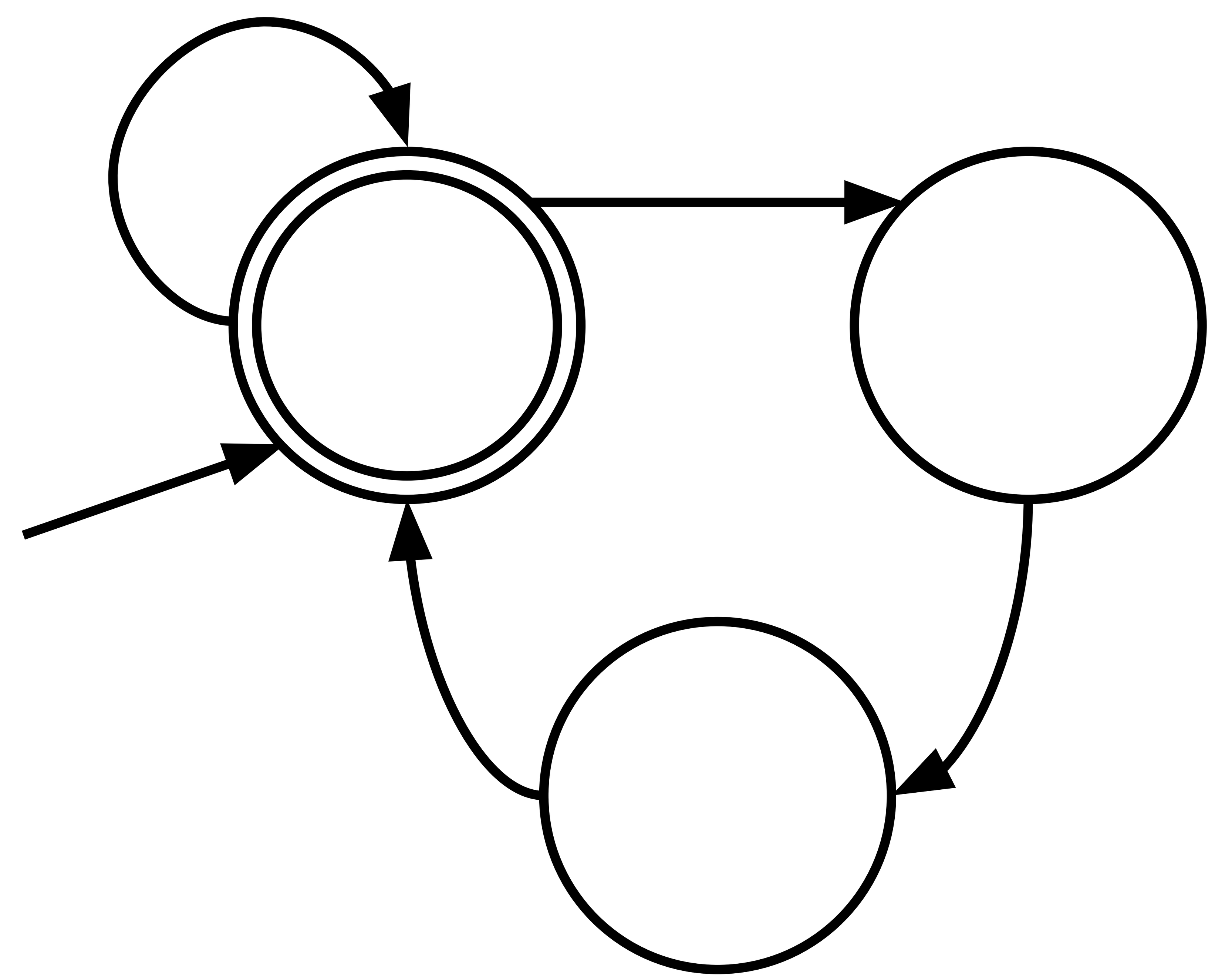


… to Symbolic Automaton



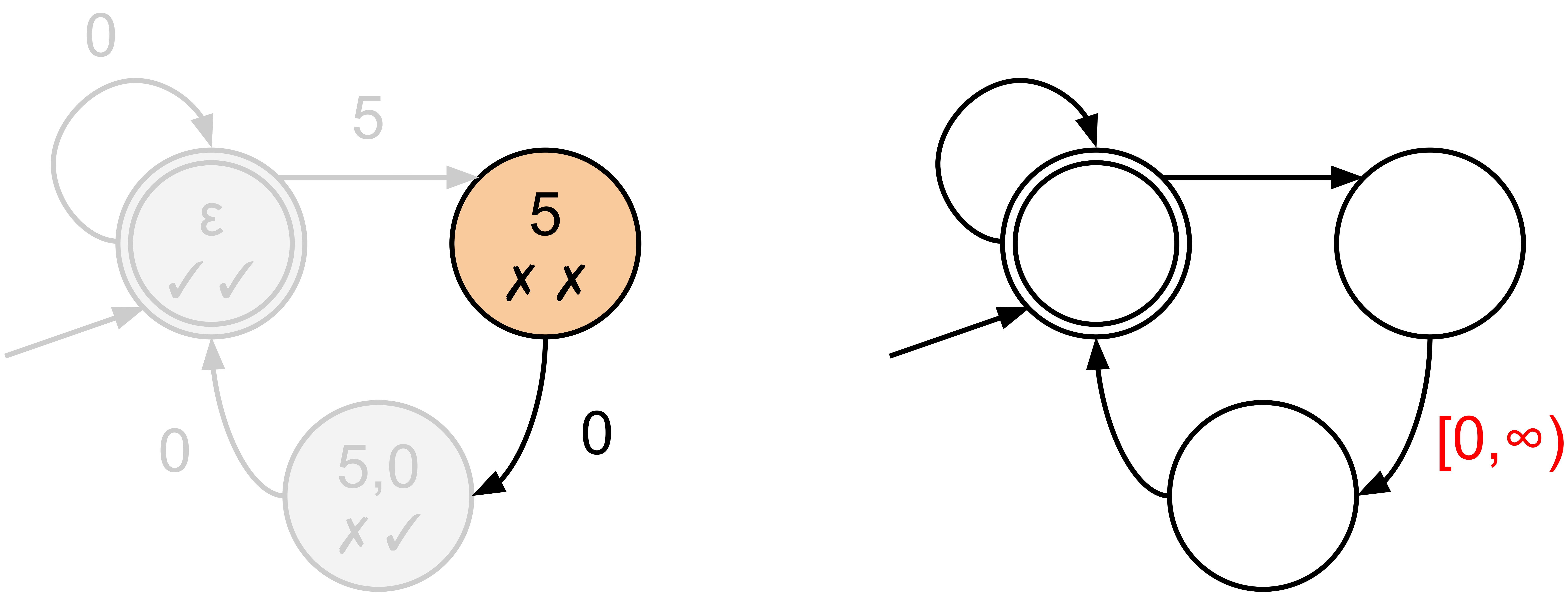
… to Symbolic Automaton





… to Symbolic Automaton

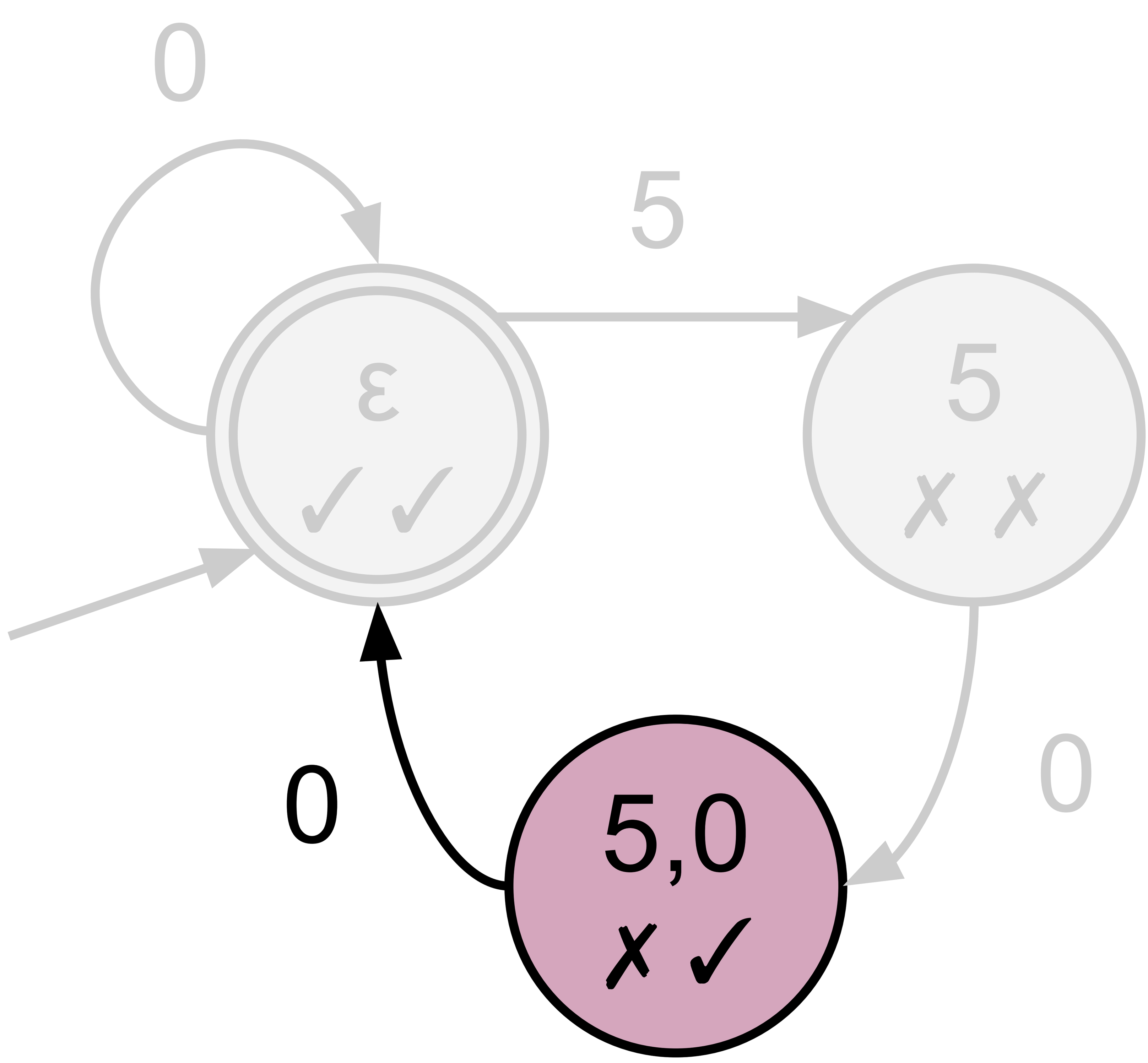
BA = intervals over $Z_{\geq 0}$

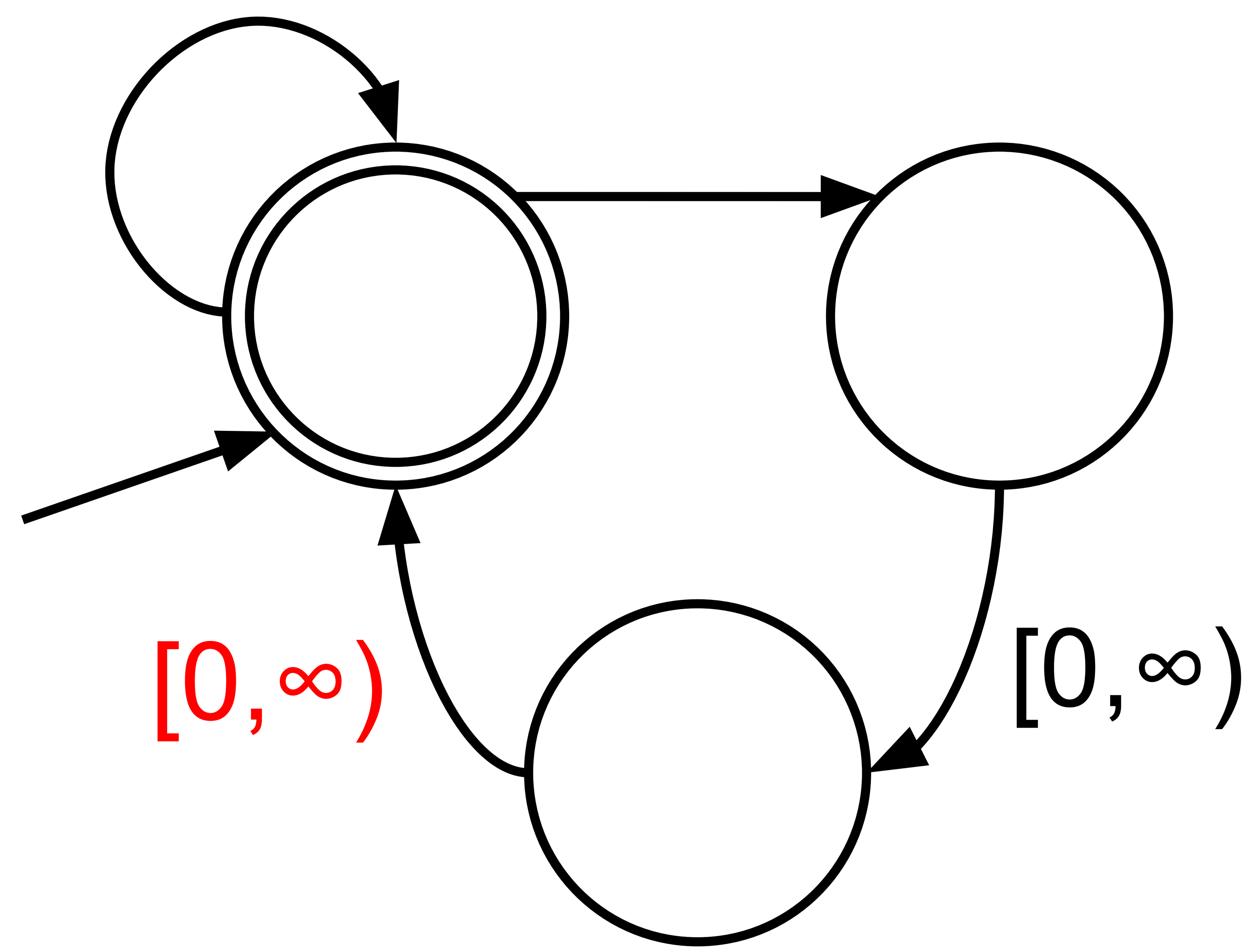


Use partitioning function: P([{0}]) = [0,∞)

… to Symbolic Automaton

BA = intervals over $Z_{\geq 0}$

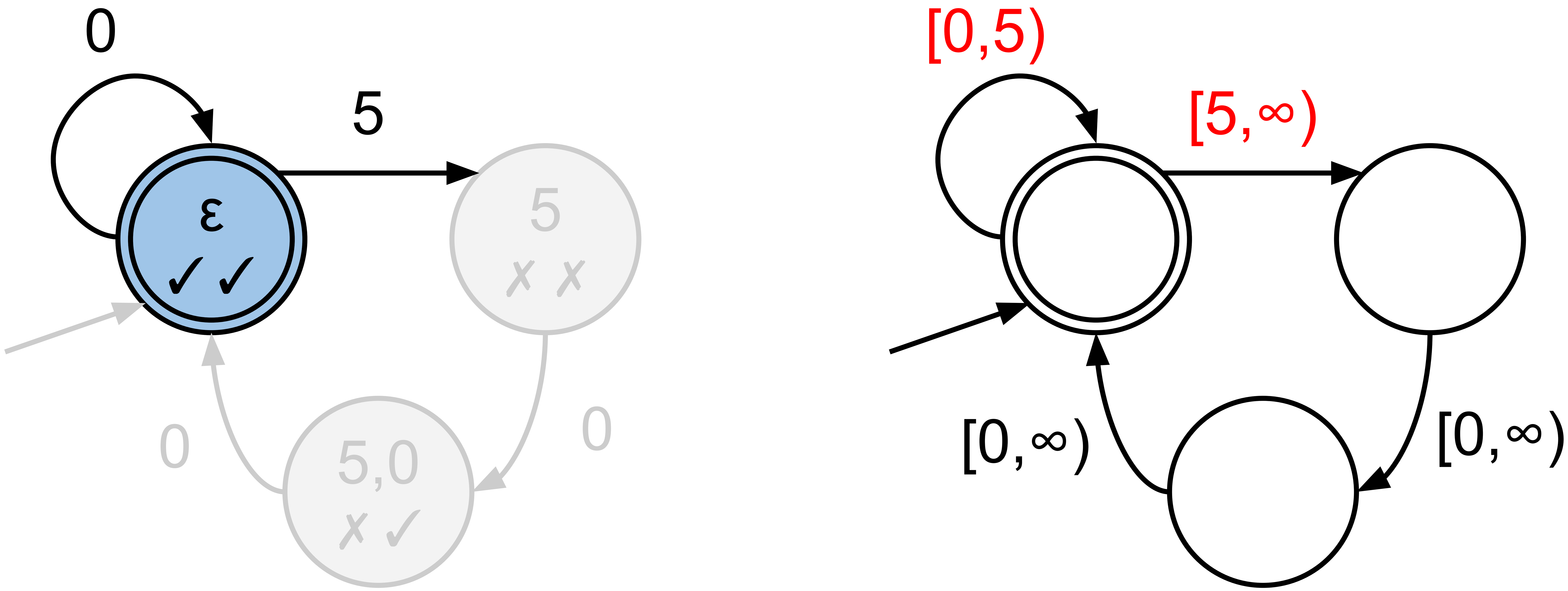




Use partitioning function: $P([\{0\}]) = [0,\infty)$

… to Symbolic Automaton

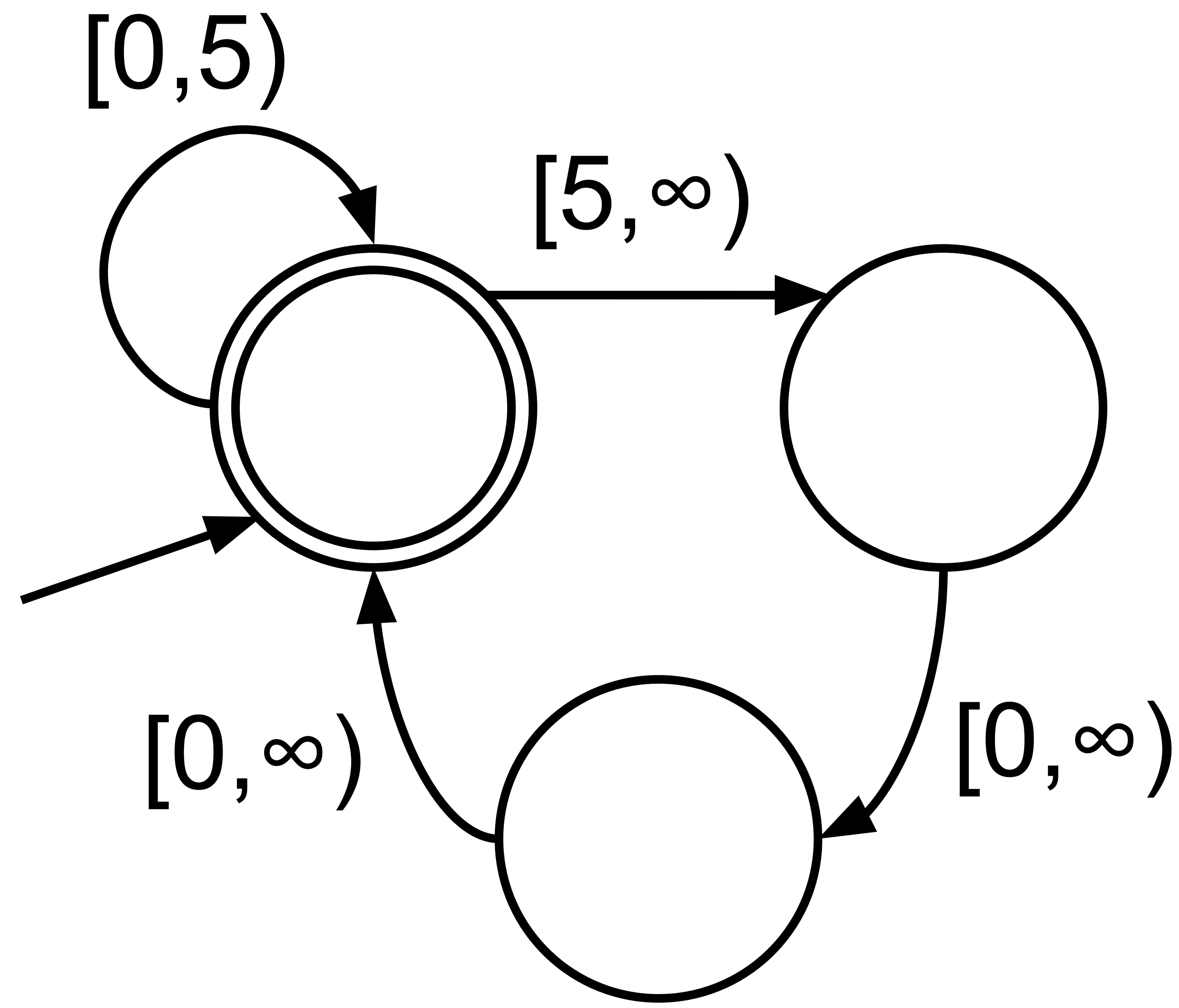
BA = intervals over $Z_{\geq 0}$



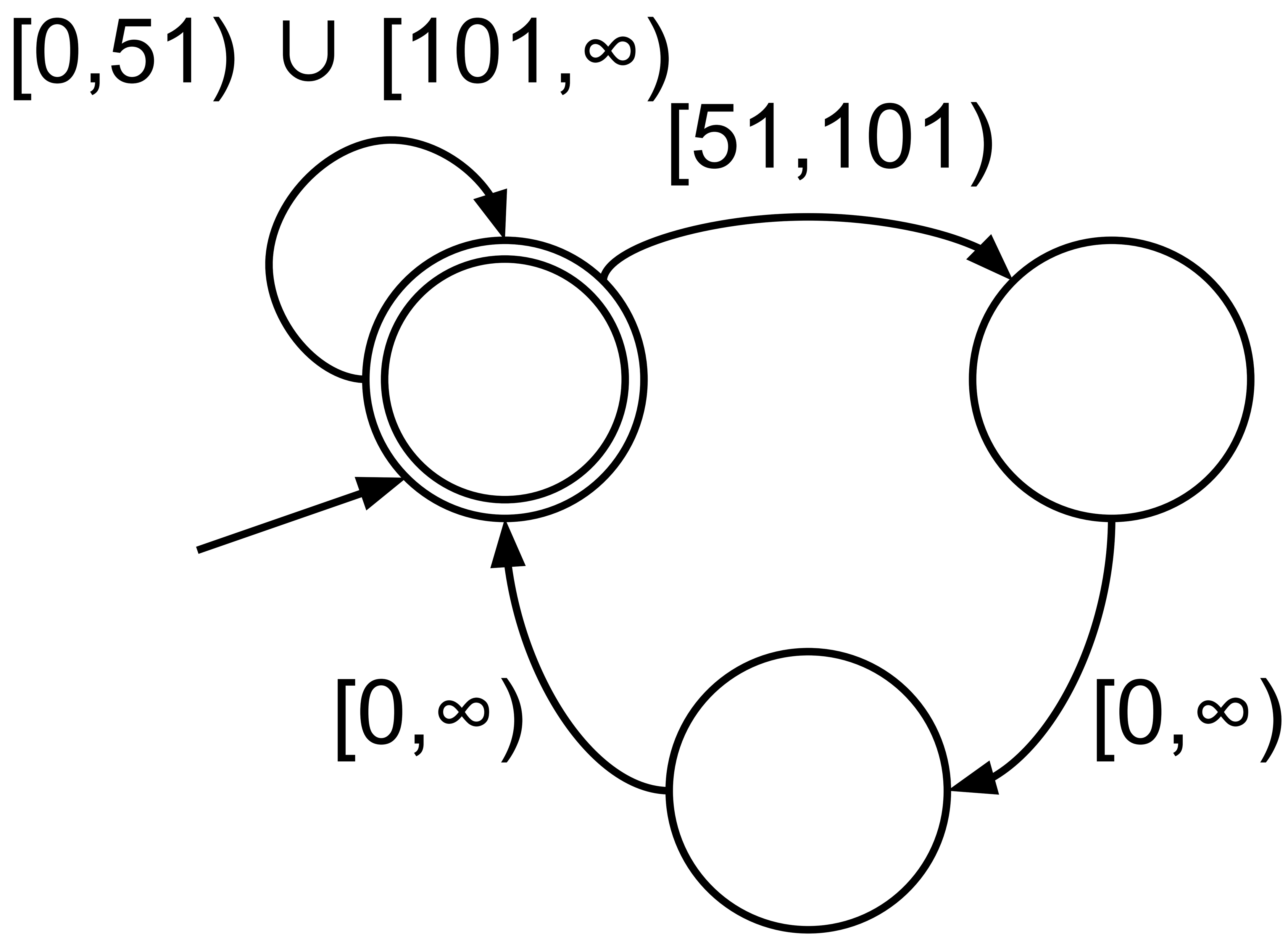
Use partitioning function: P([{0},{5}]) = [0,5), [5,∞)

… to Symbolic Automaton

	ε	0
ε	✓	✓
5	✗	✗
5,0	✗	✓
0	✓	✓
5,0,0	✓	✓



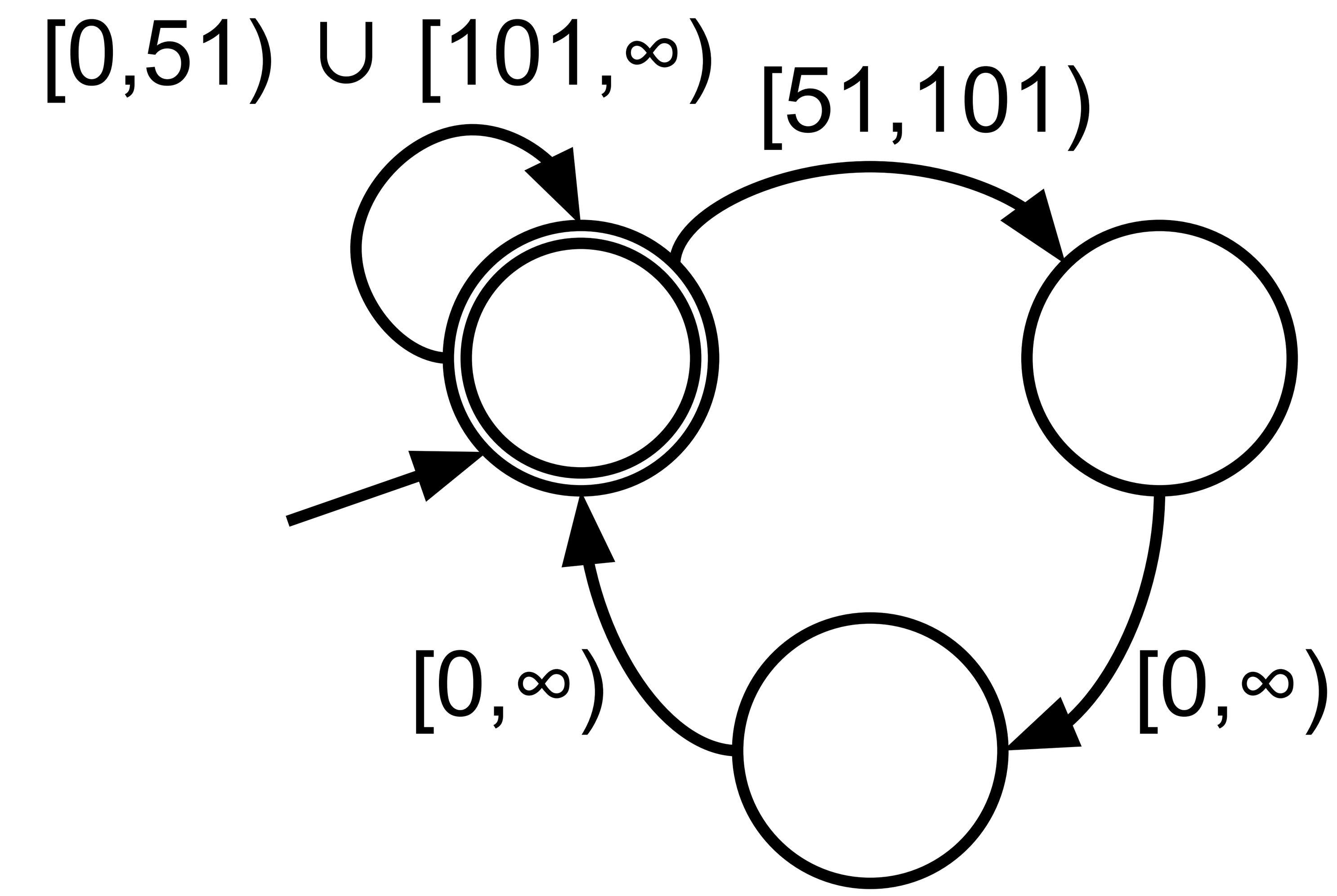
Λ^* by Example



Σ = non-negative integers

BA = unions of intervals over Σ

Λ* by Example



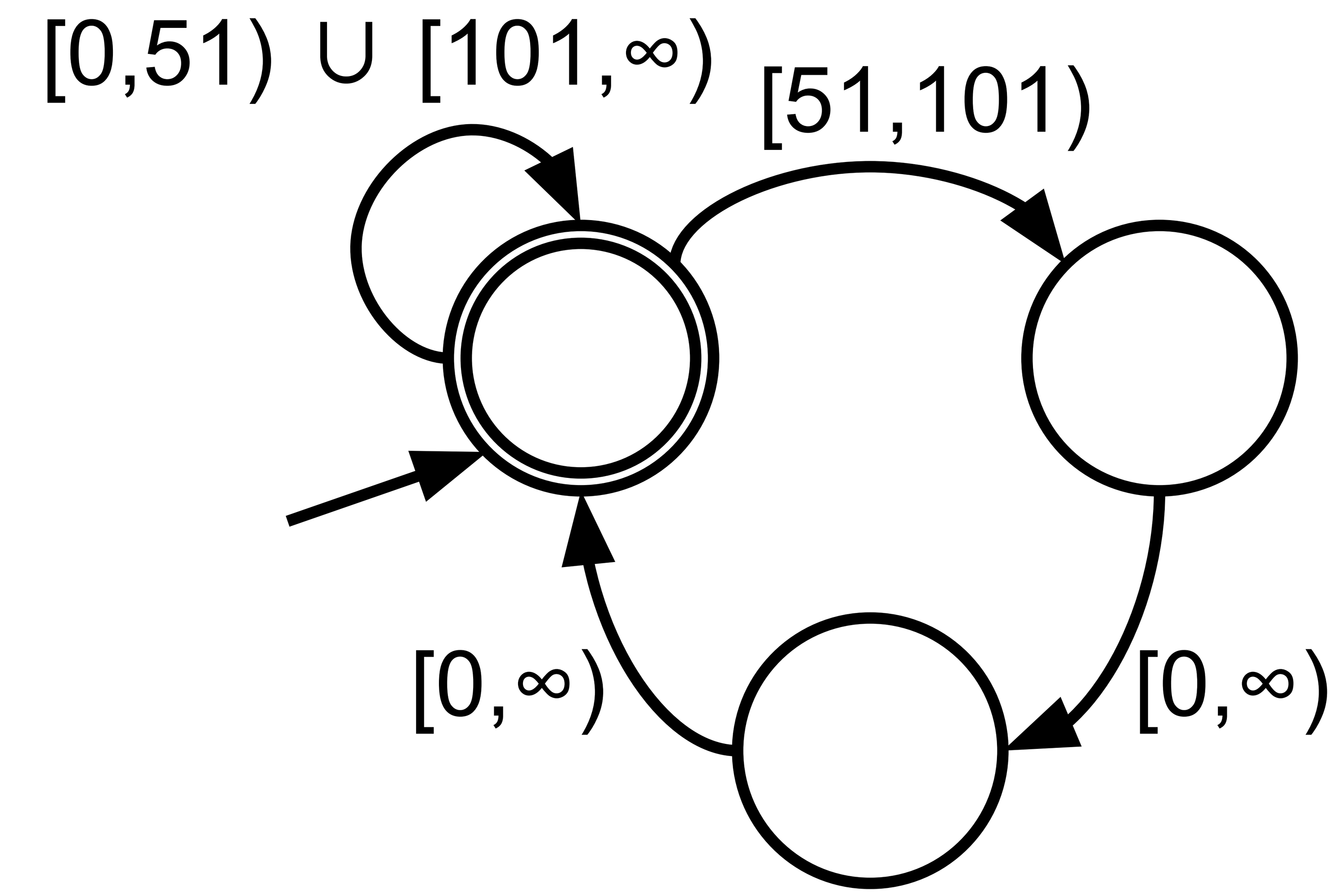
	ε
ε	✓
0	✓

Initialize table:

Membership query for ε

Membership query for 0 (arbitrary)

Λ* by Example



	ε
ε	✓
0	✓

Initialize table:

Membership

Membershi

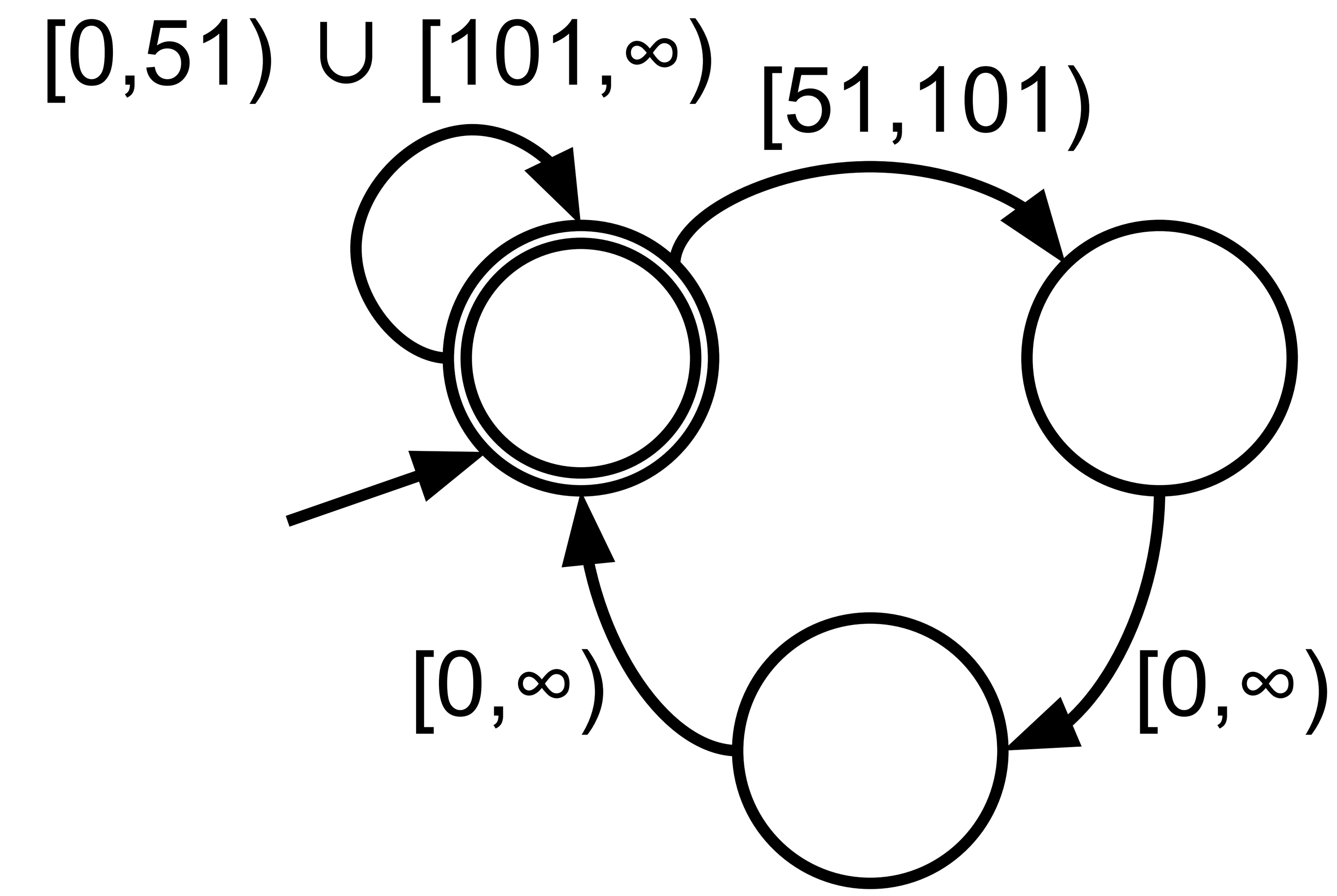
Λ* : query for single element

L* : queries for all of Σ

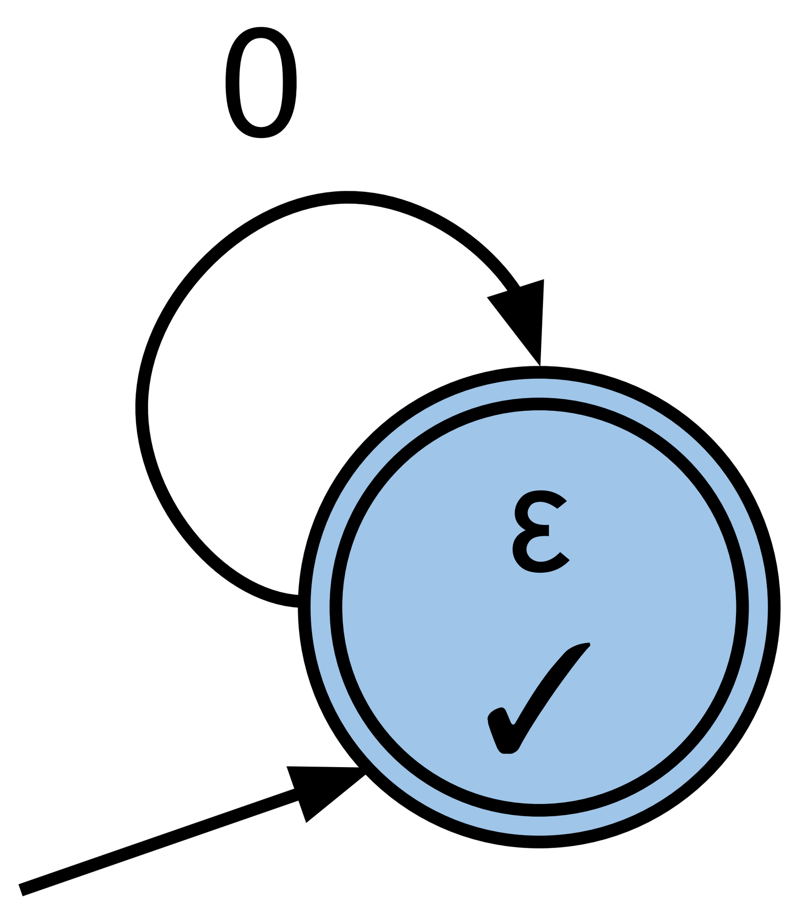
	ε
ε	✓
0	✓
1	✓
2	✓

…

Λ^* by Example



	ε
ε	✓
0	✓

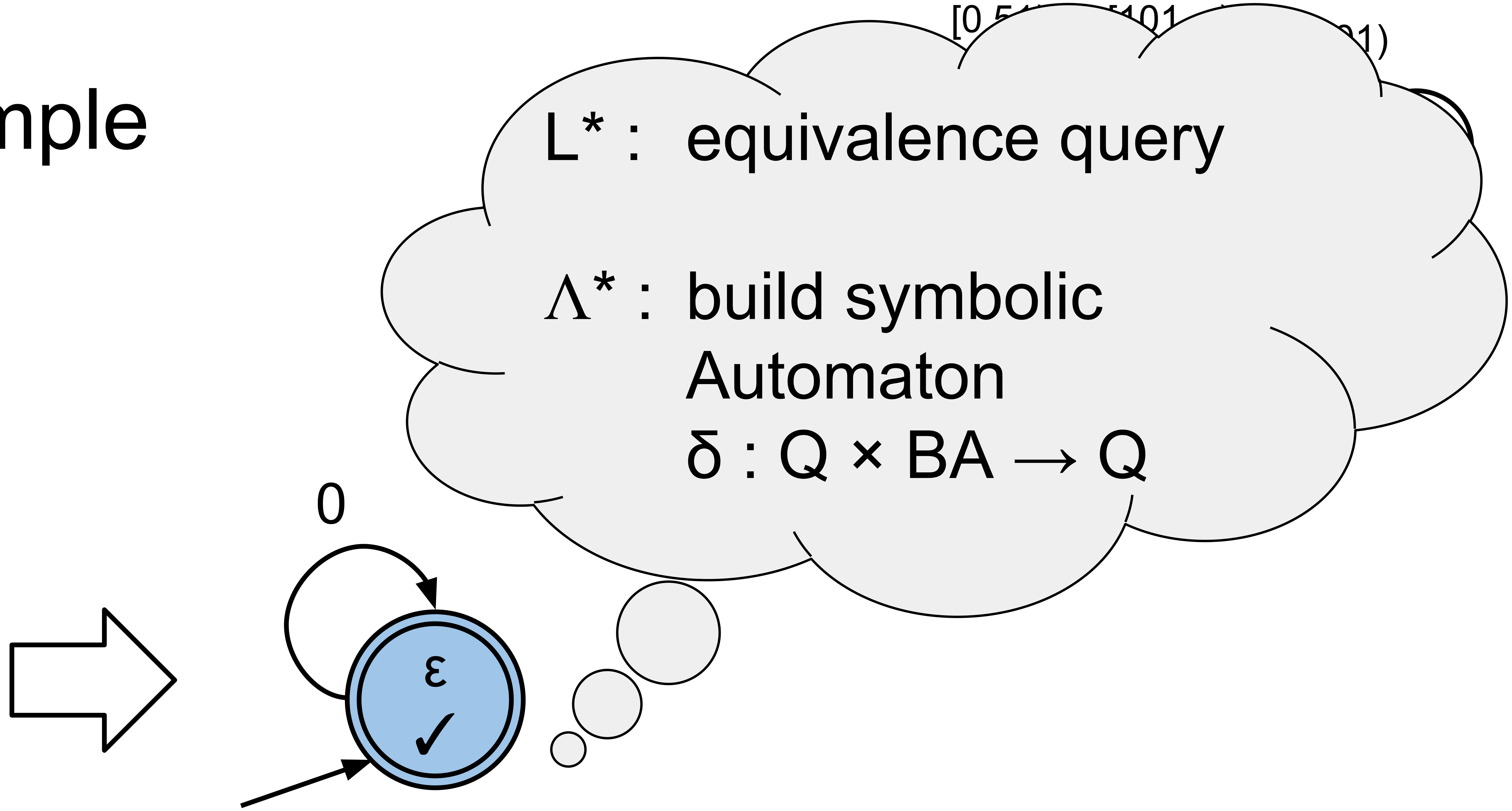


Build “sparse” automaton from table

$\delta : Q \times \Sigma \rightarrow Q$

Λ* by Example

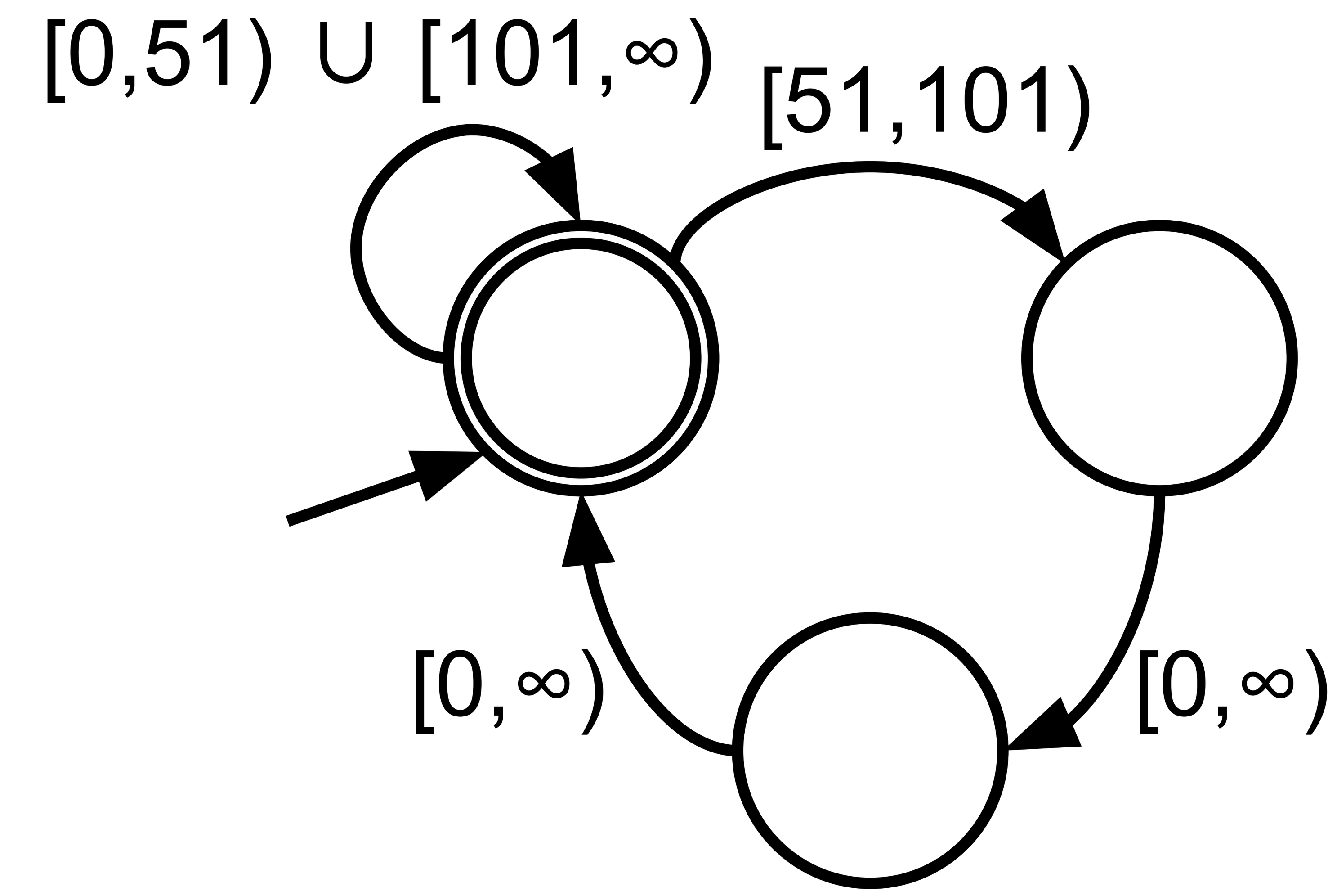
	ε
ε	✓
0	✓



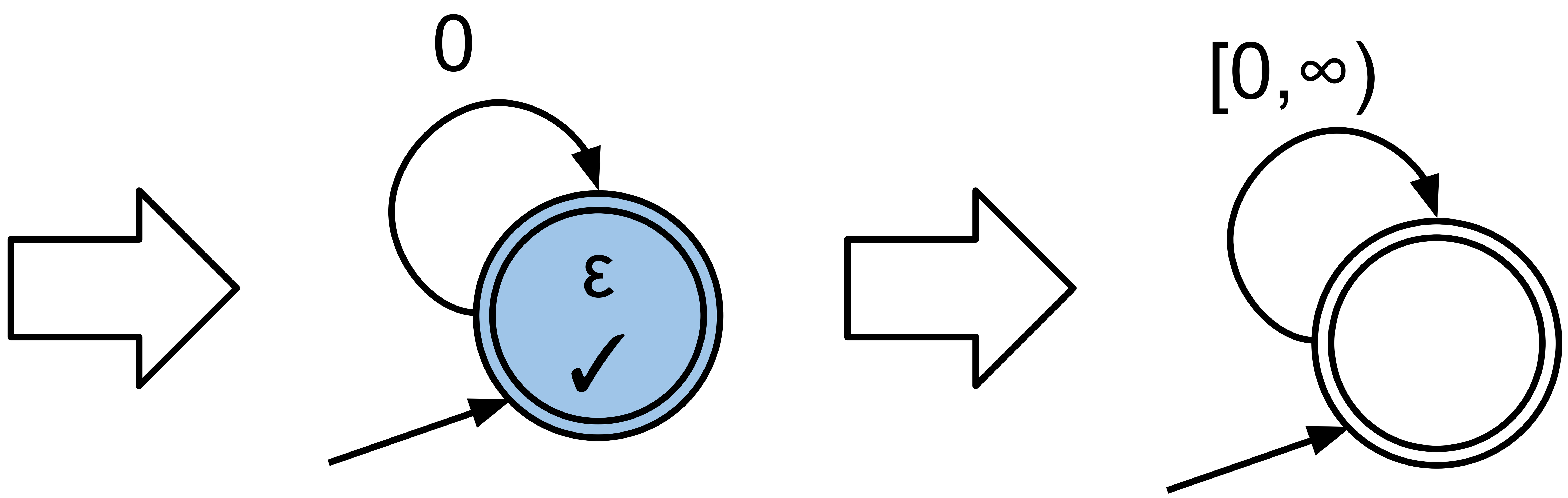
Build "sparse" automaton from table

$\delta : Q \times \Sigma \rightarrow Q$

Λ* by Example

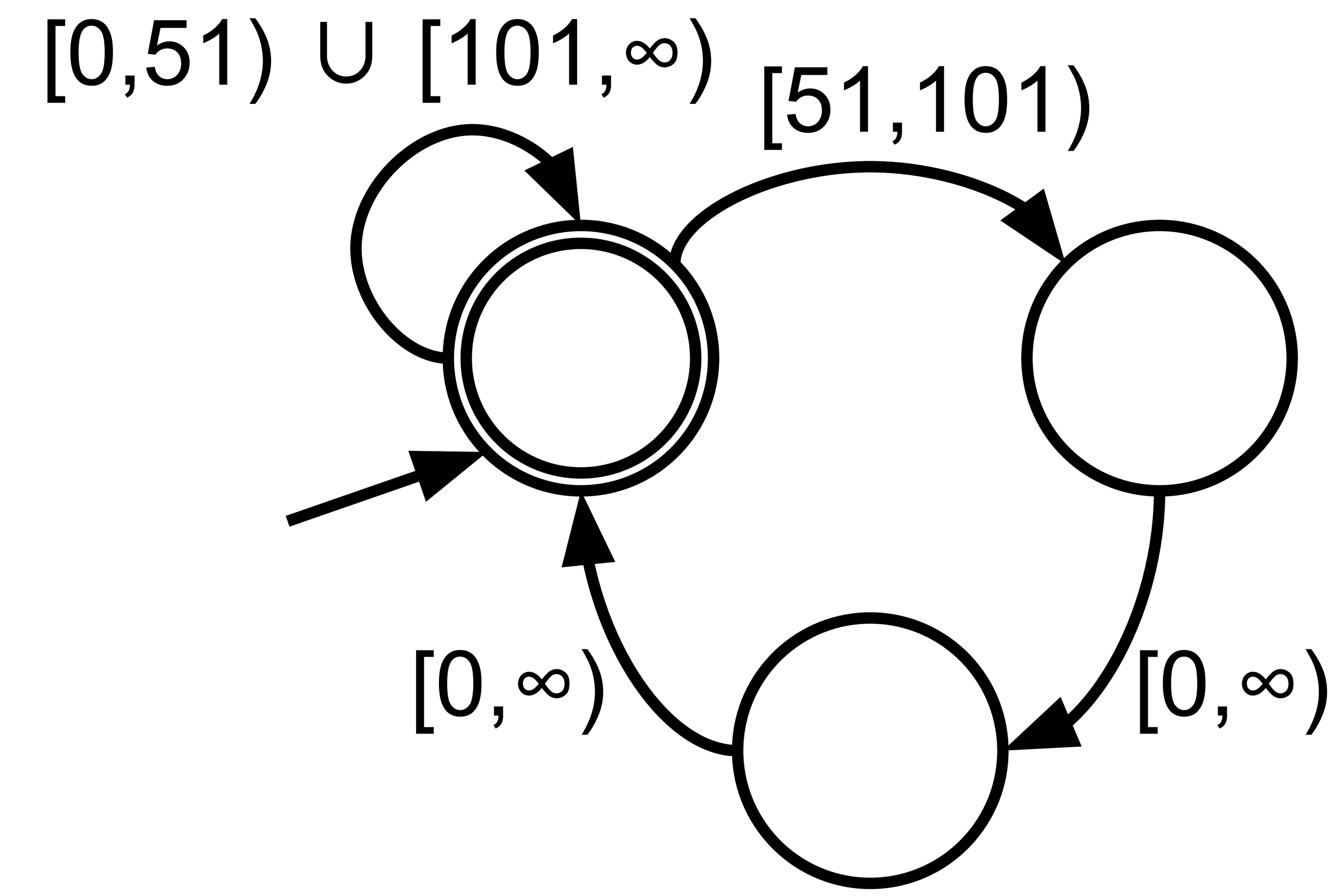


	ε
ε	✓
0	✓

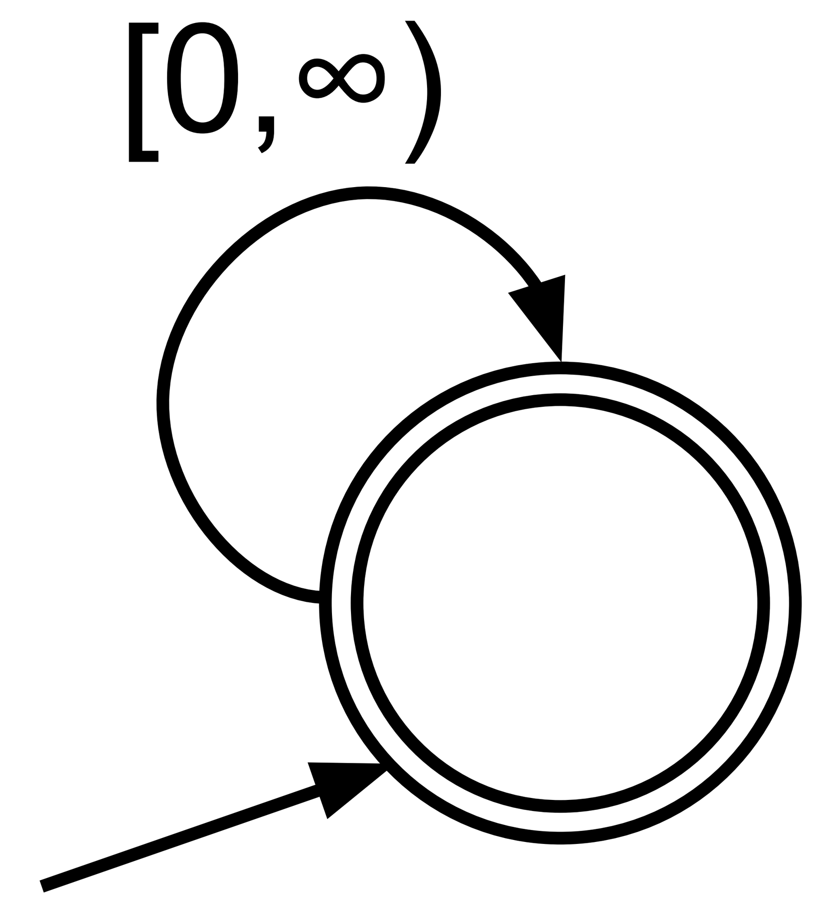


Build symbolic automaton using partitioning function:
suppose $P(\{0\}) = [0,\infty)$

Λ* by Example



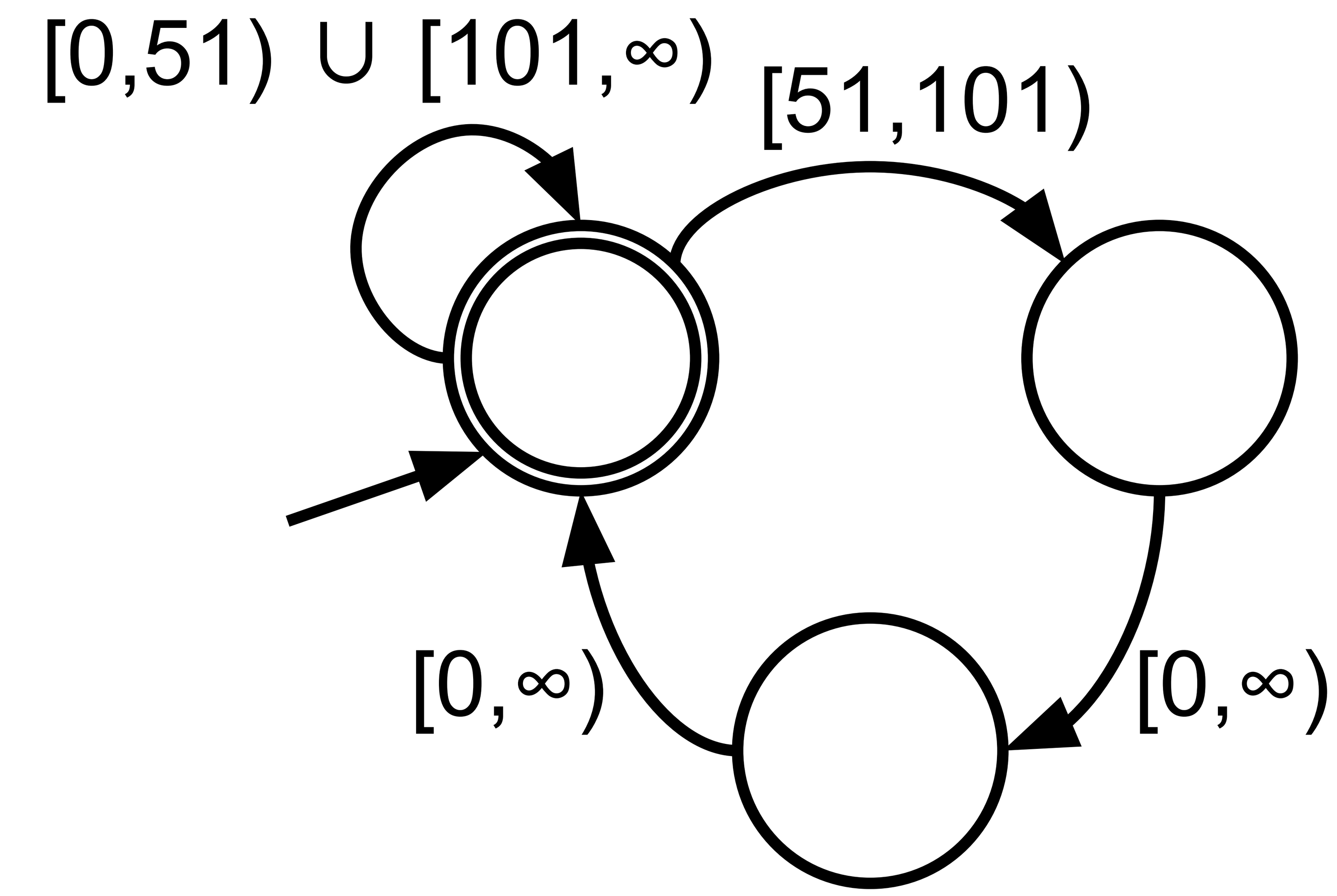
	ε
ε	✓
0	✓
51	✗



Equivalence query:

Not equivalent! cex (51, ✗)

Λ* by Example



	ε
ε	✓
0	✓
51	✗

Not *closed*:
51 leads to a new state

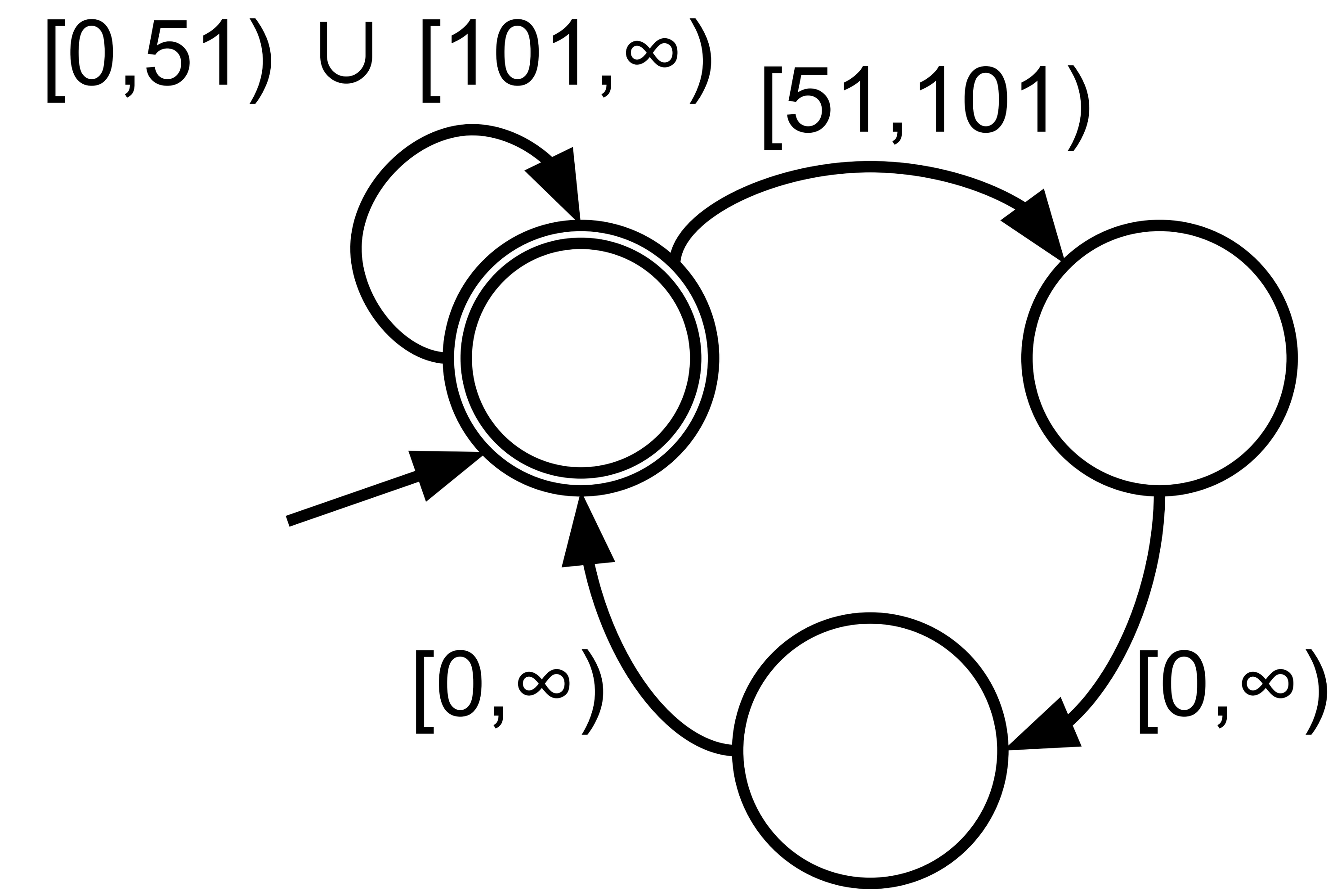
Move 51 to top

	ε
ε	✓
51	✗
0	✓

	ε
ε	✓
51	✗
0	✓
51,0	✗

Membership query
on 51,0

Λ* by Example



Move 51 to top

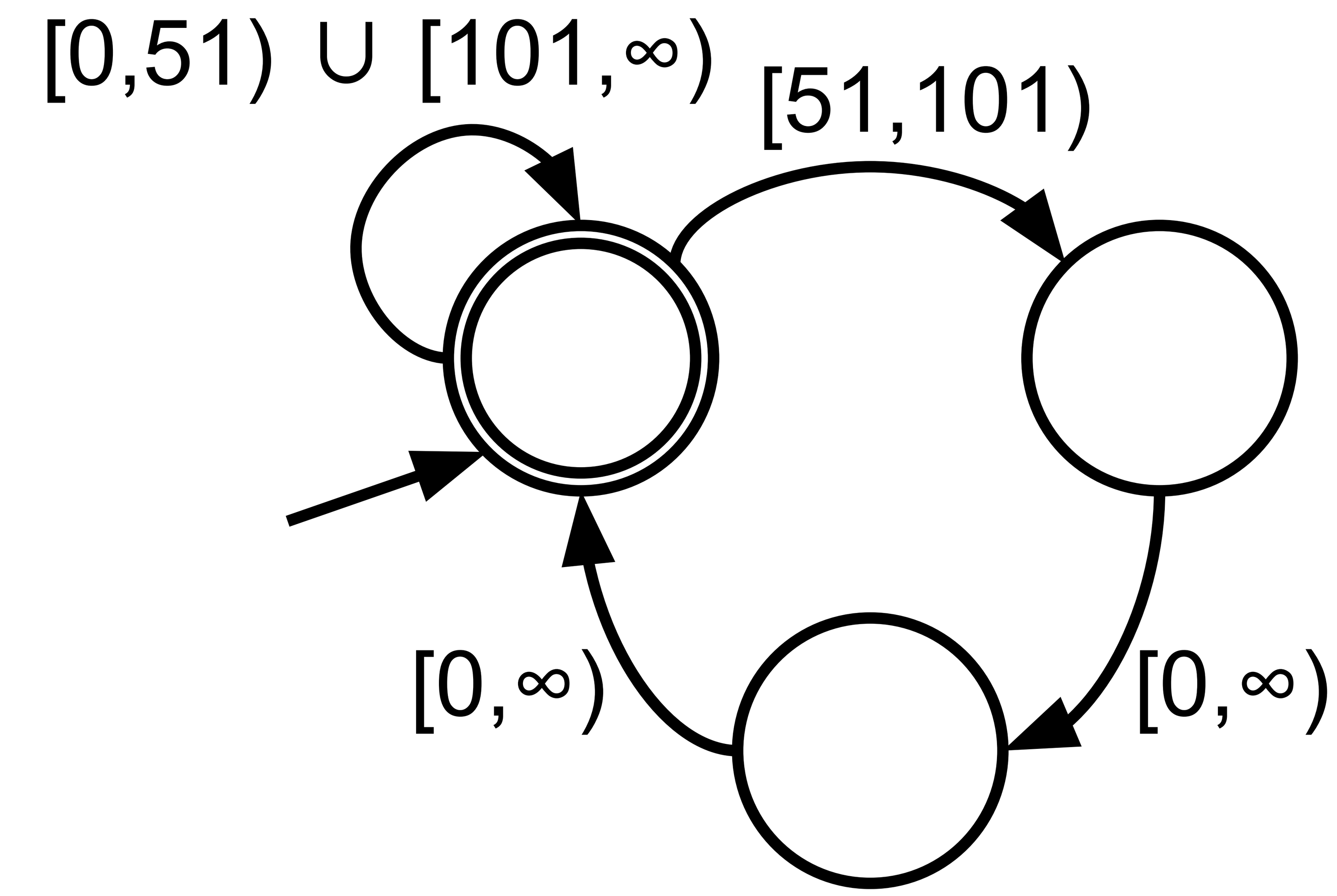
	ε
ε	✓
0	
51	

L* queries all of 51·Σ

	ε
ε	✓
51	✗
0	✓
51,0	✗

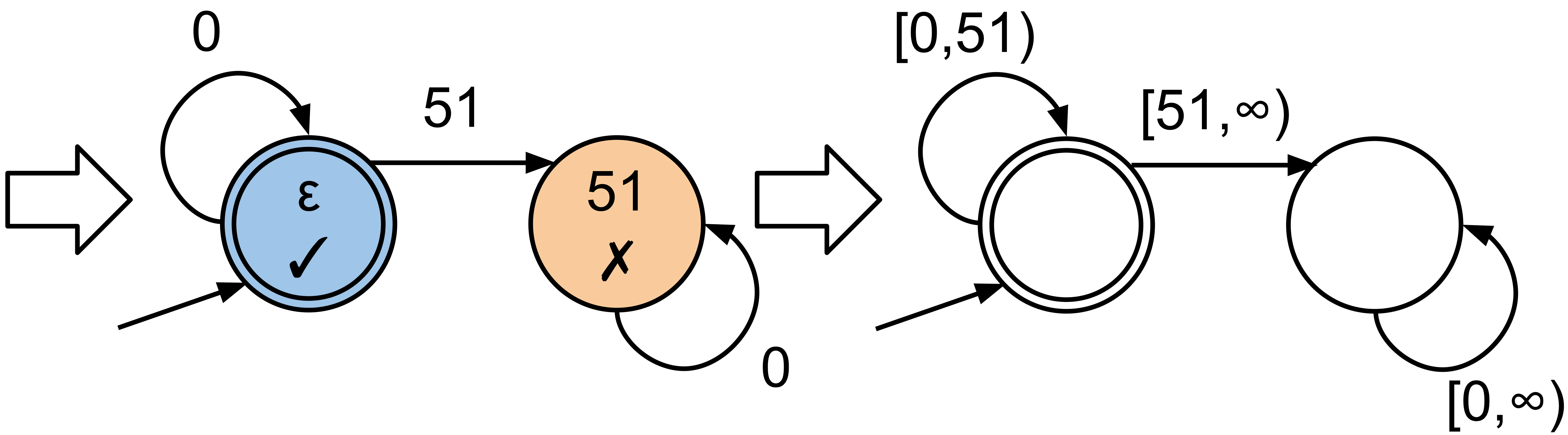
Not *closed*:
51 leads to a new state

Membership query on 51,0



Λ* by Example

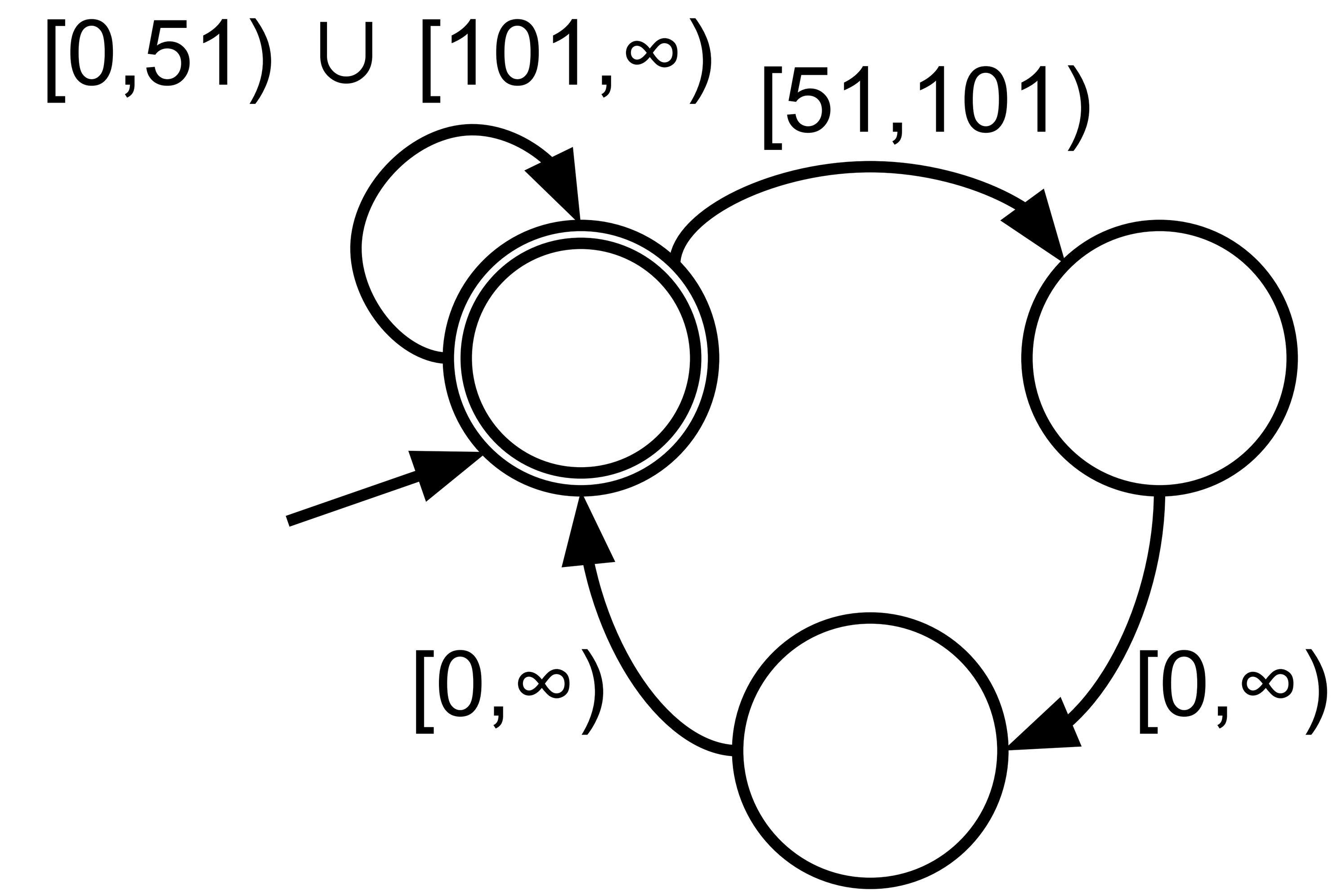
	ε
ε	✓
51	✗
0	✓
51,0	✗



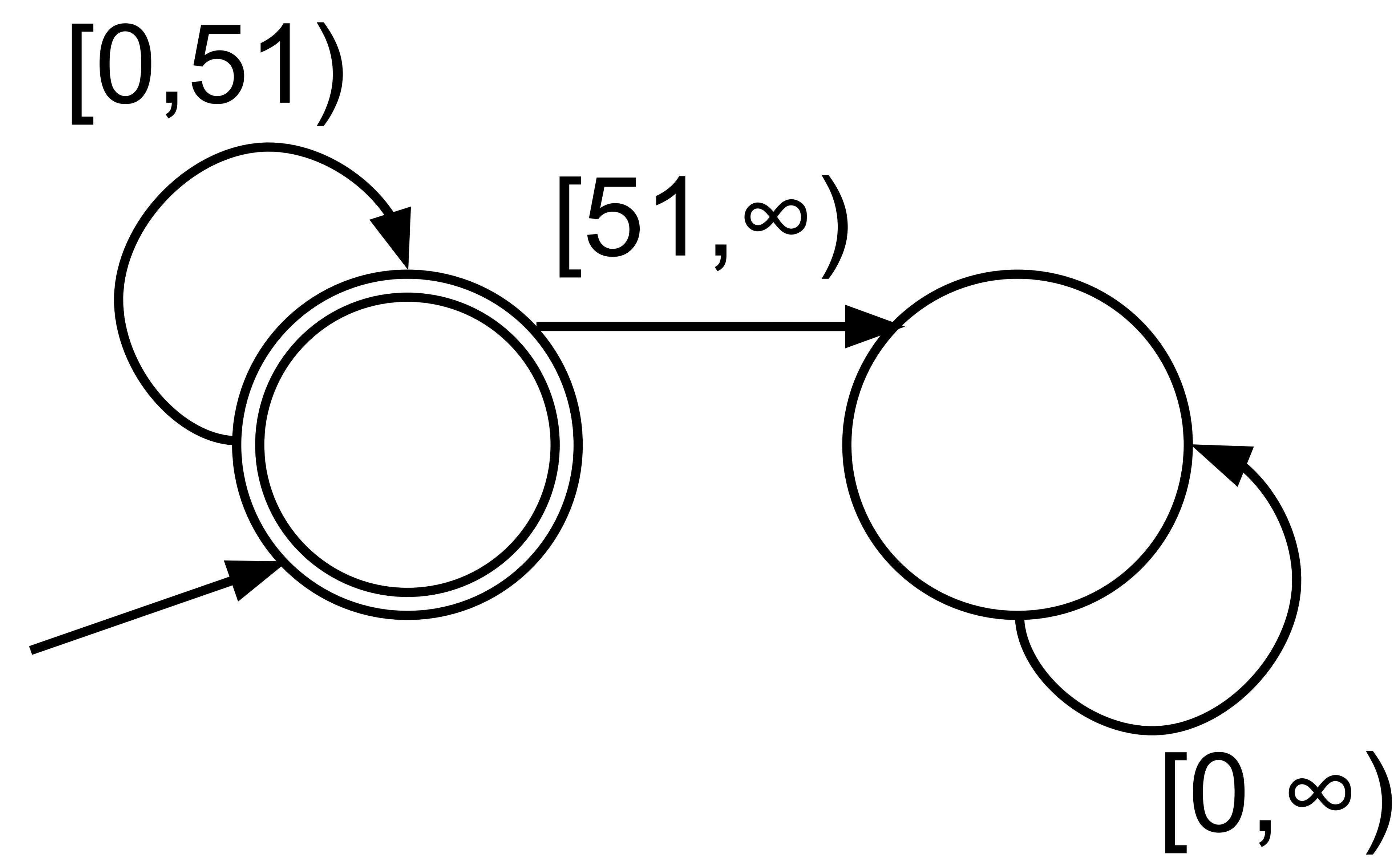
suppose P({0},{51}) = [0,51) , [51,∞)

P({0}) = [0,∞)

Λ* by Example



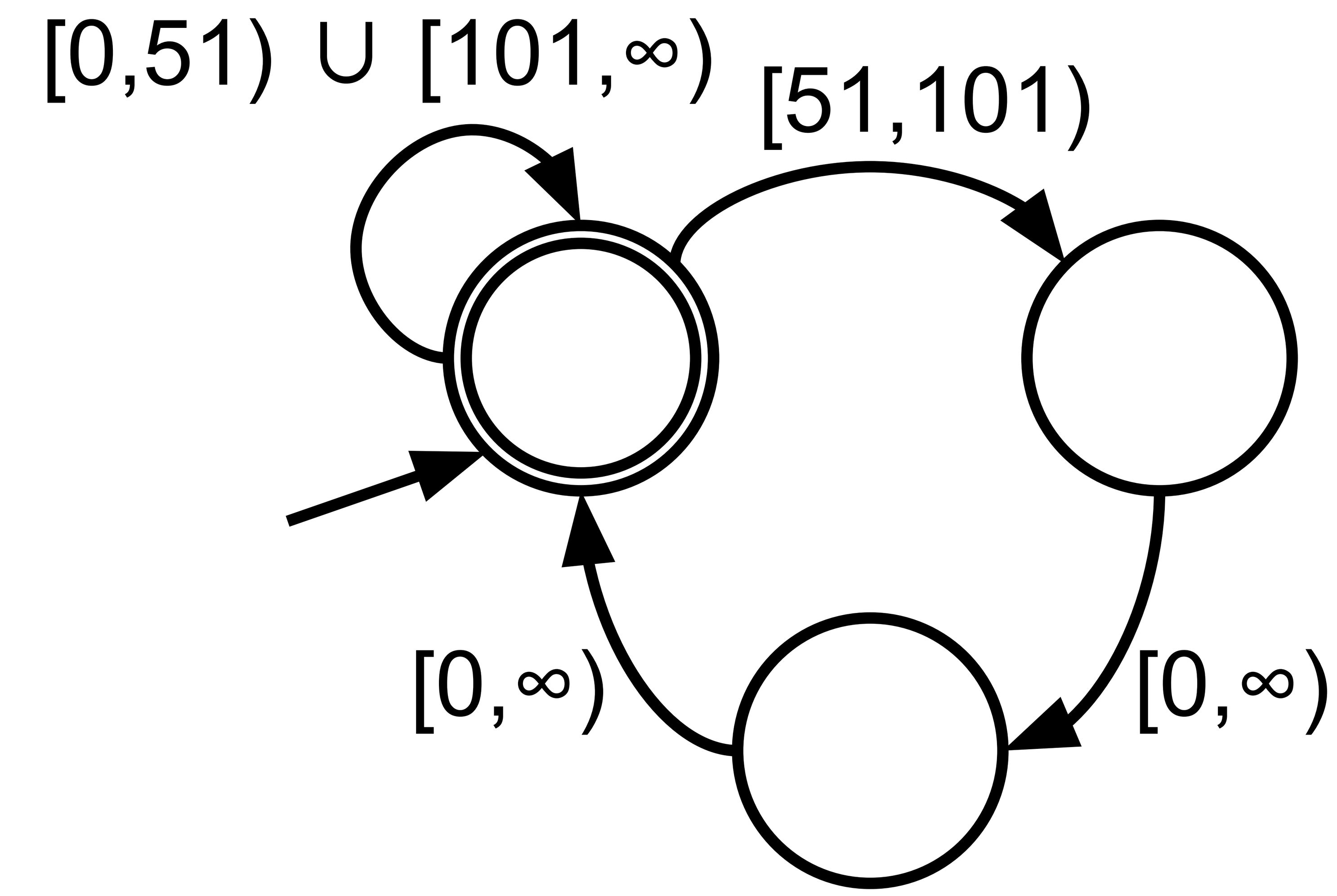
	ε
ε	✓
51	✗
0	✓
51,0	✗



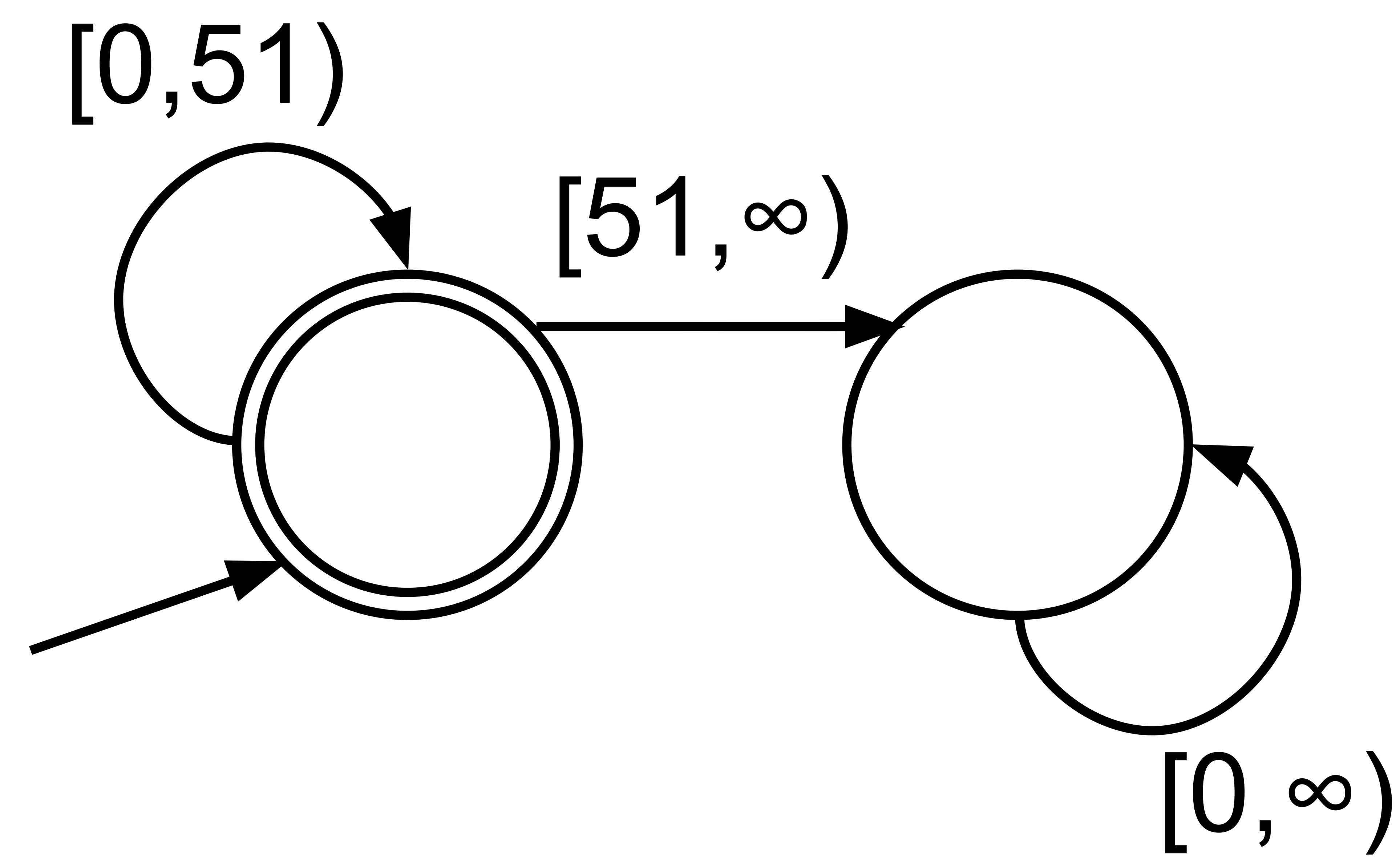
Equivalence query:

Not equivalent! cex (101; ✓)

Λ* by Example



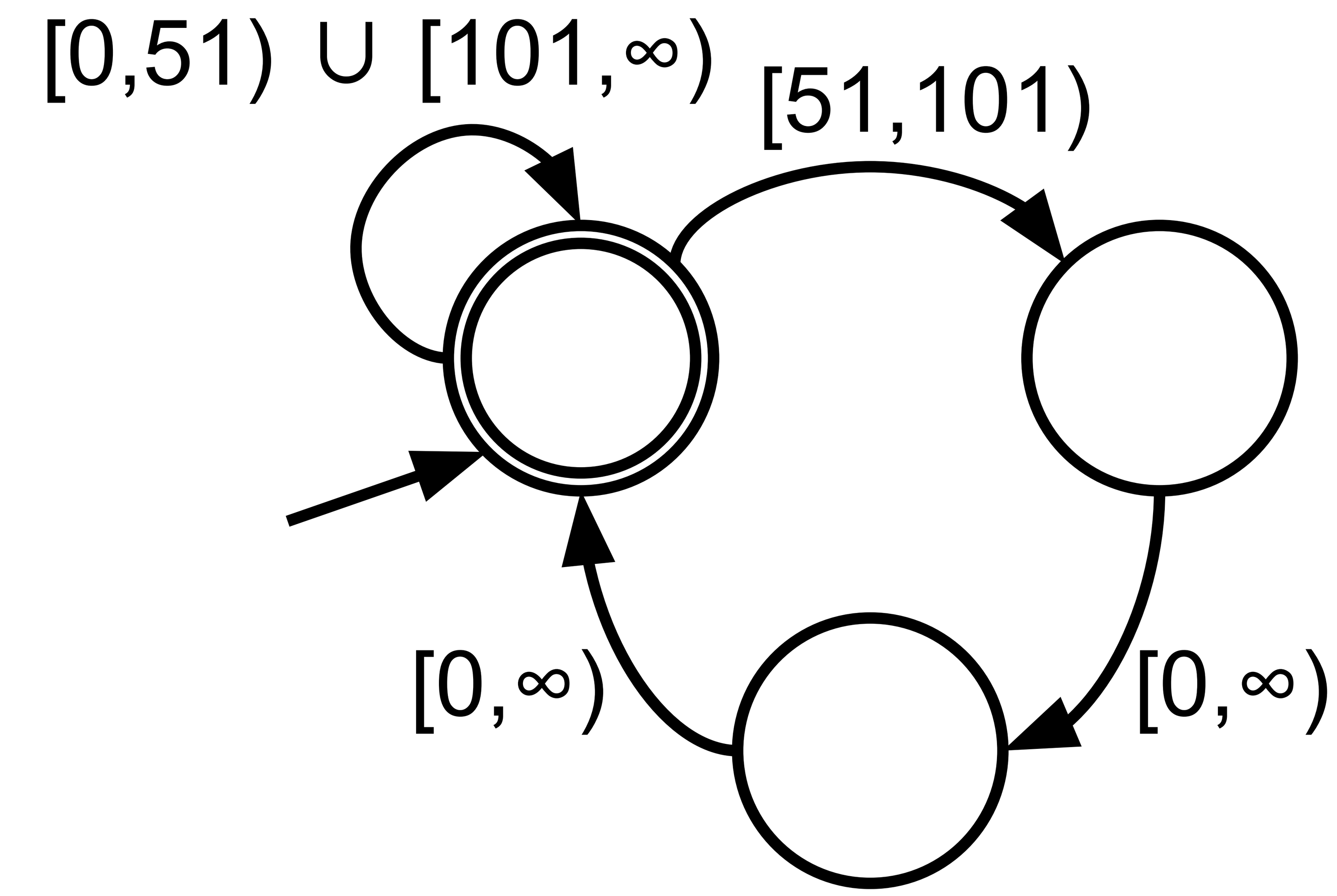
	ε
ε	✓
51	✗
0	✓
51,0	✗
101	✓



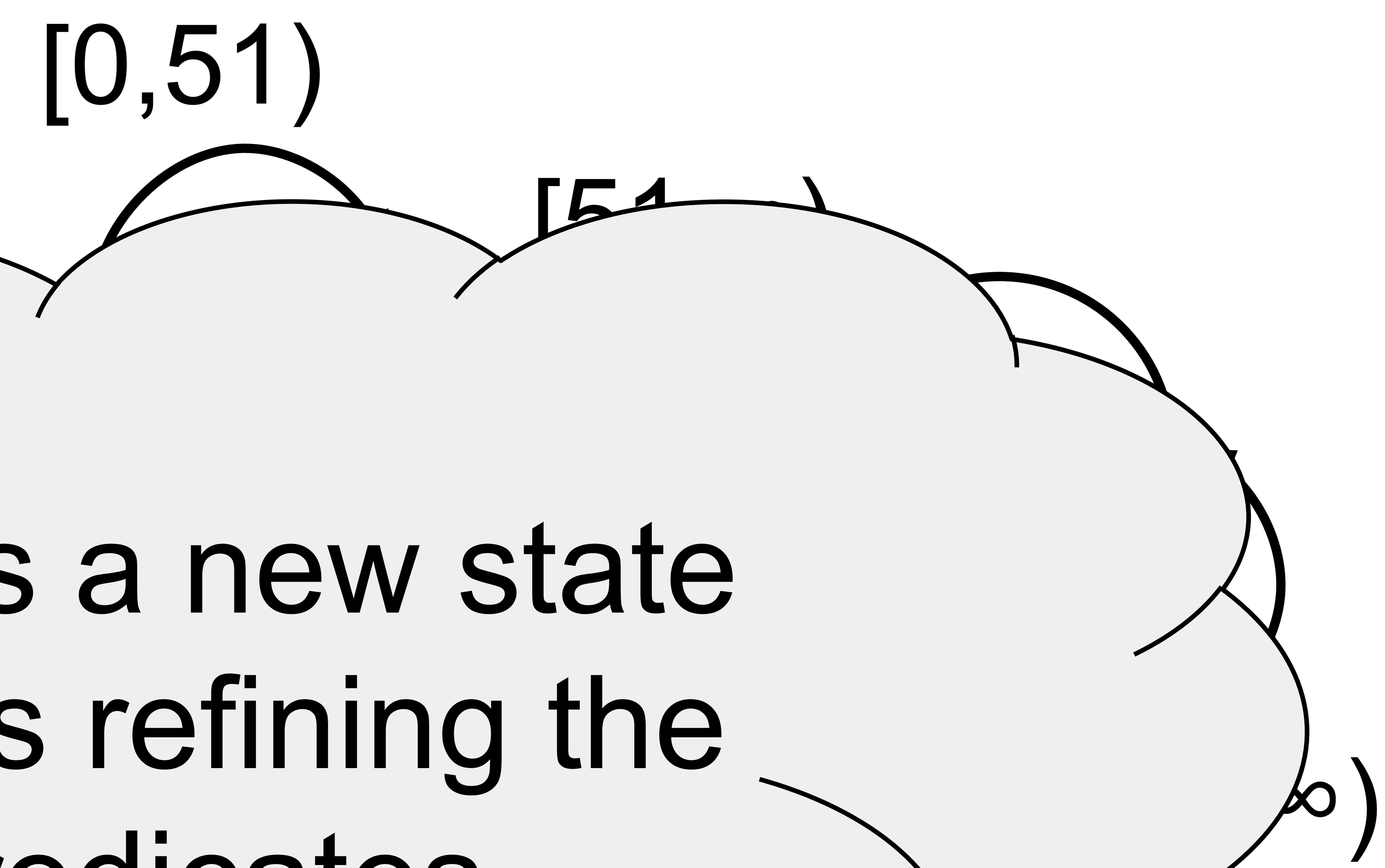
Equivalence query:

Not equivalent! cex (101; ✓)

Λ* by Example

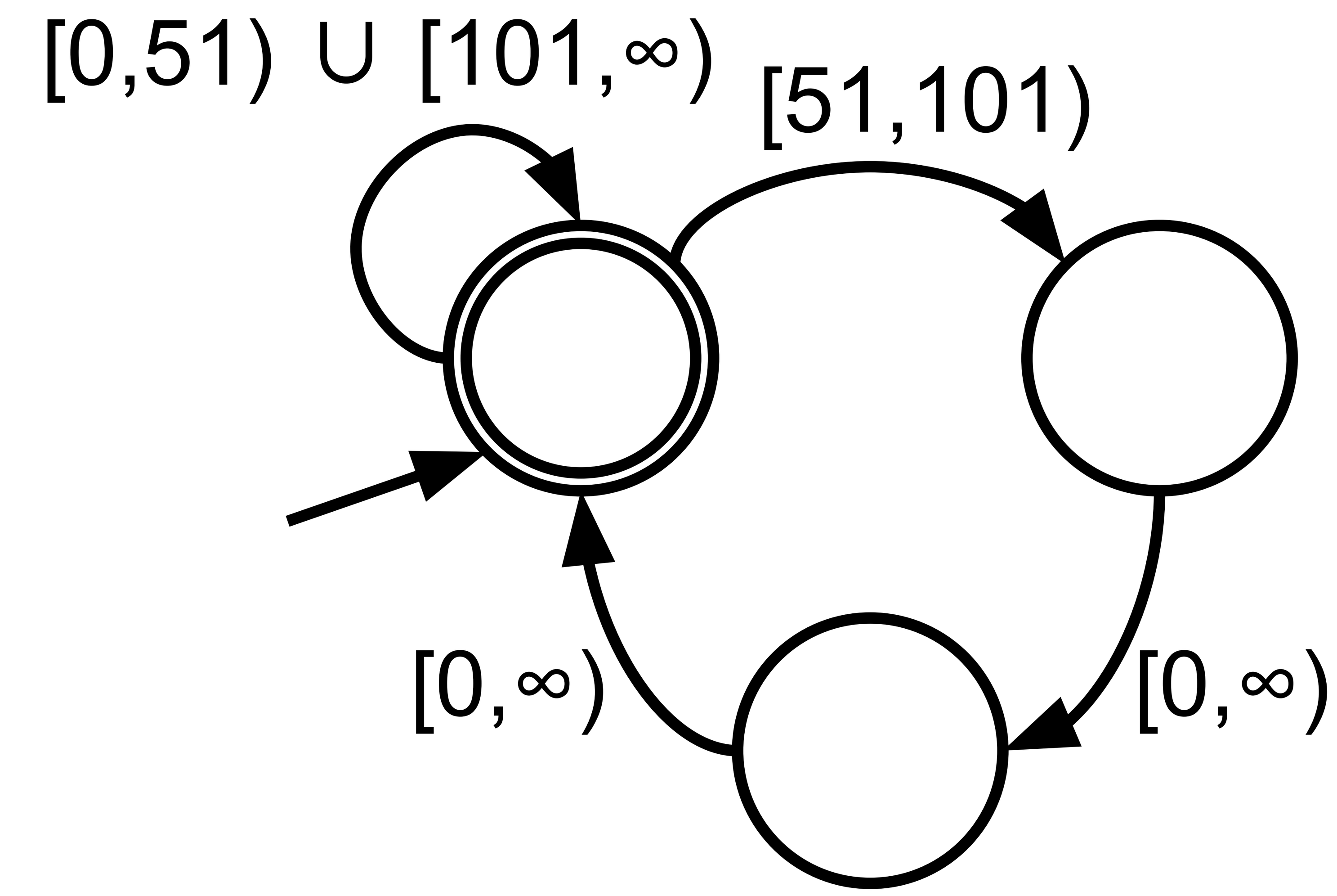


	ε
ε	✓
51	✗
0	✓
51,0	✗
101	✓

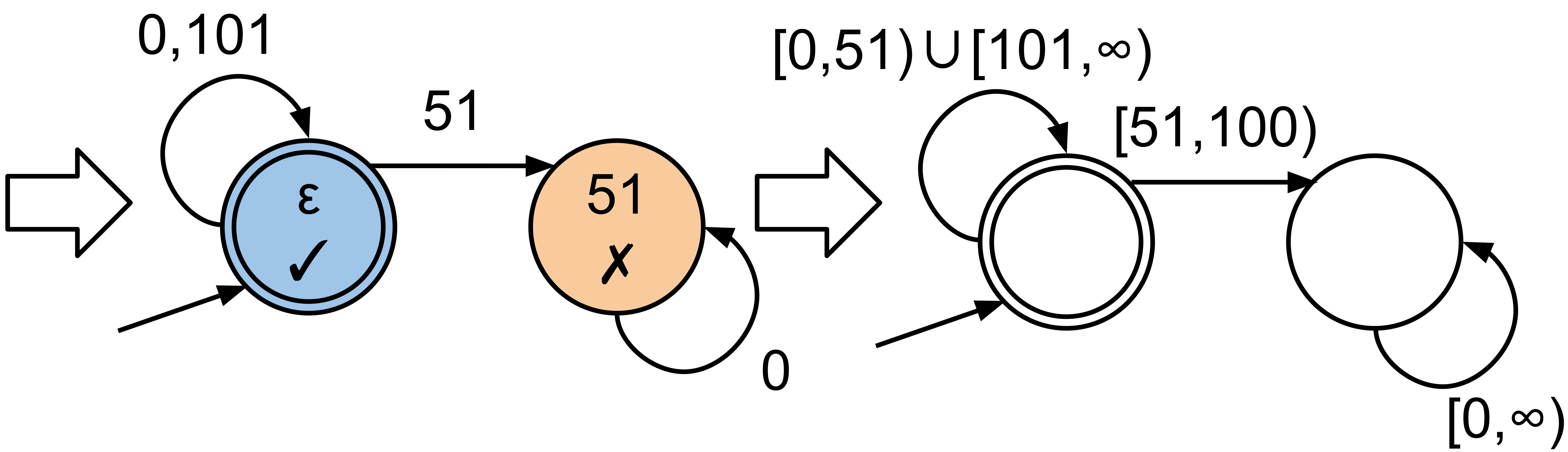


L* : every cex is a new state
Λ* : some cex is refining the outgoing predicates

Λ^* by Example



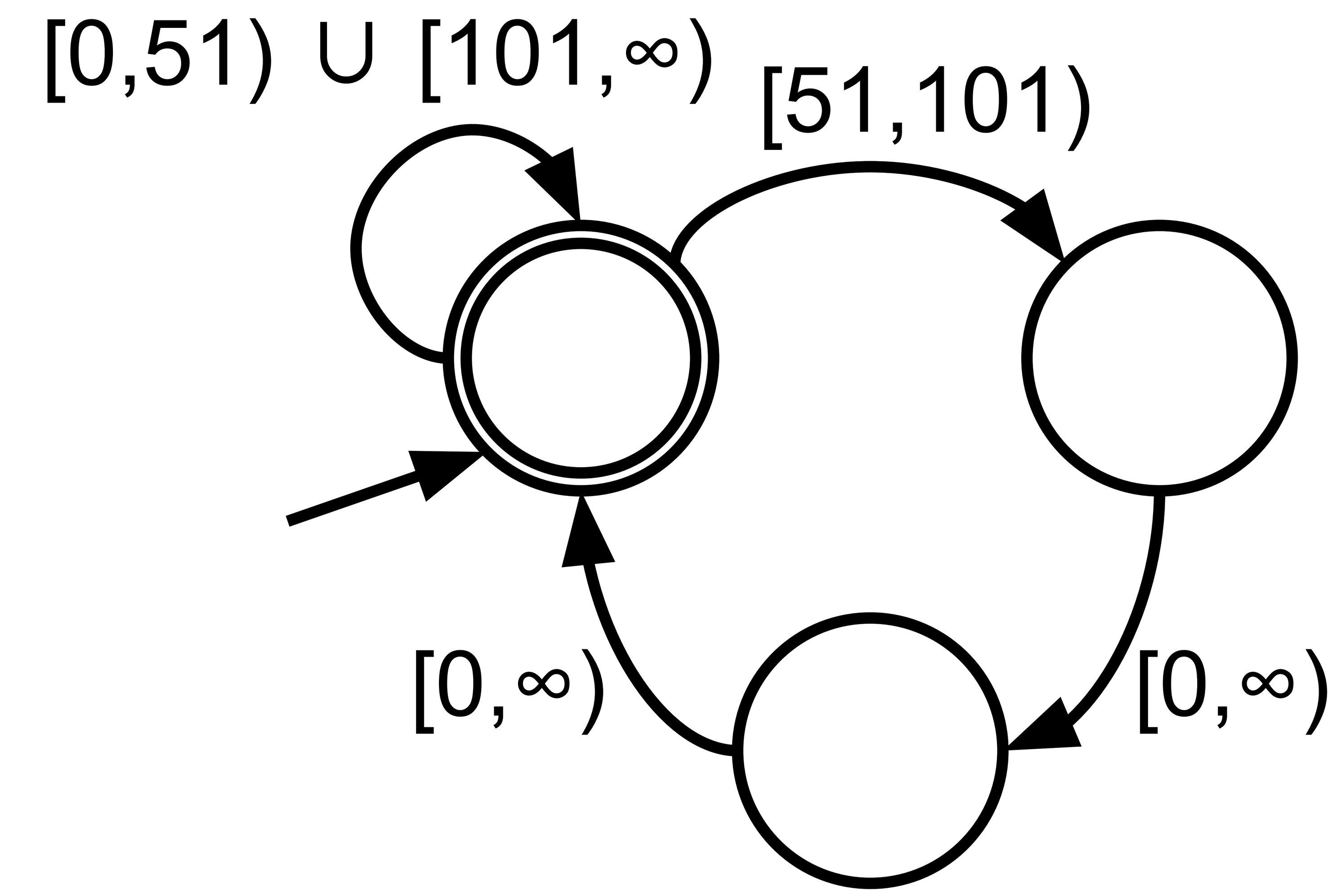
	ε
ε	✓
51	✗
0	✓
51,0	✗
101	✓



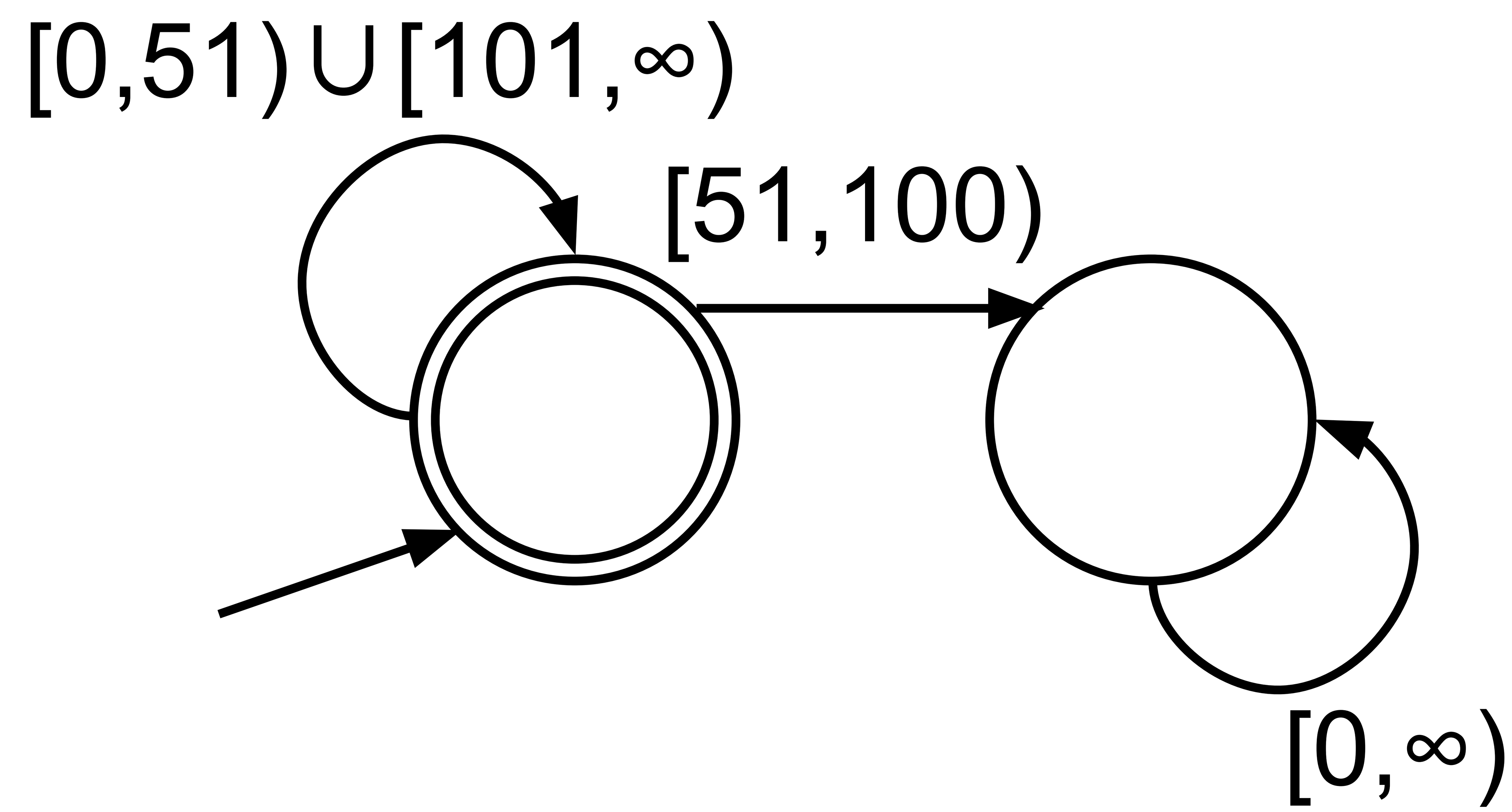
suppose $P(\{0,101\},\{51\}) = [0,51) \cup [101,\infty)\ ,\ [51,\infty)$

$P(\{0\}) = [0,\infty)$

Λ* by Example



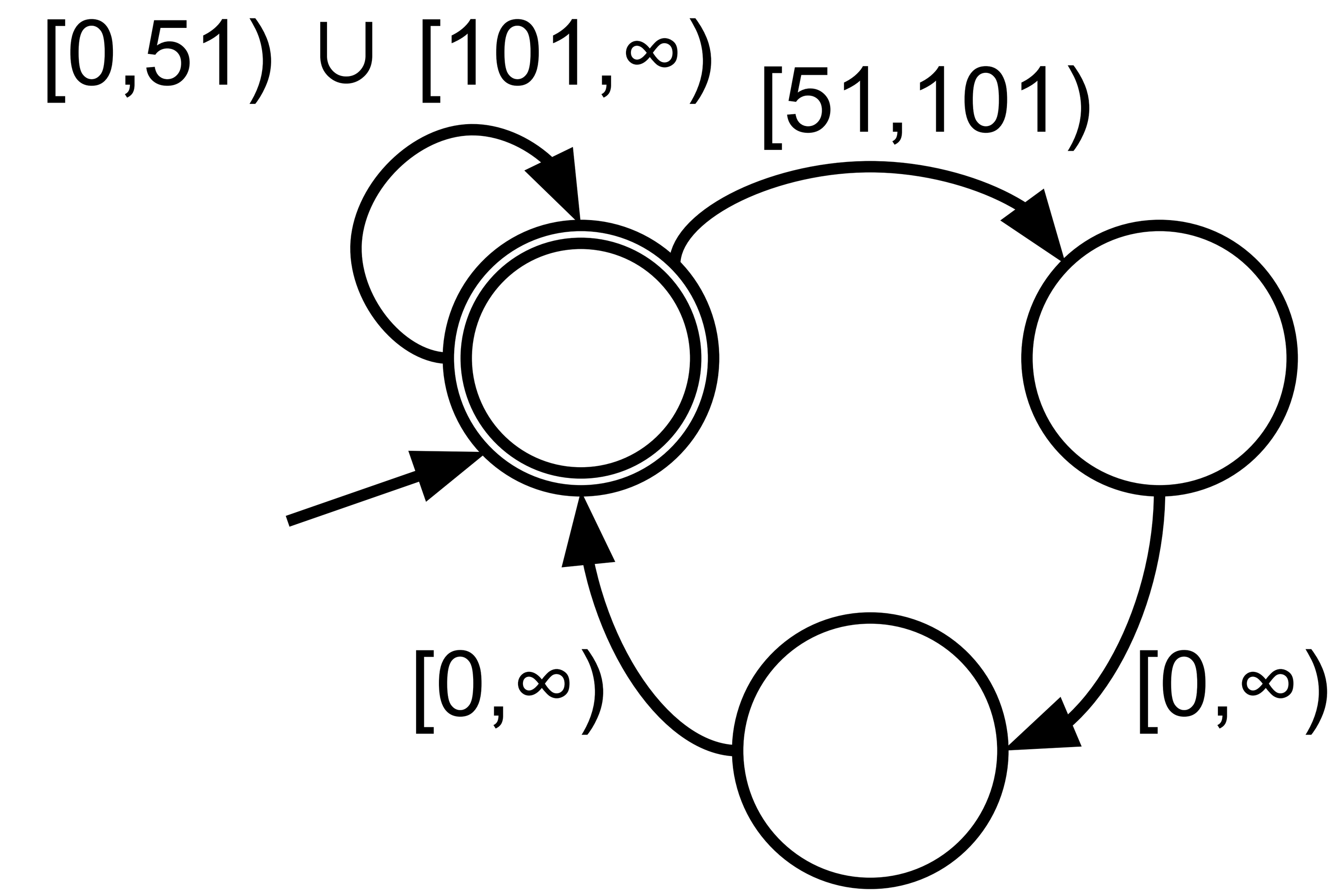
	ε
ε	✓
51	✗
0	✓
51,0	✗
101	✓
51,0,0	✓



Equivalence query:

Not equivalent! cex (51,0,0; ✓)

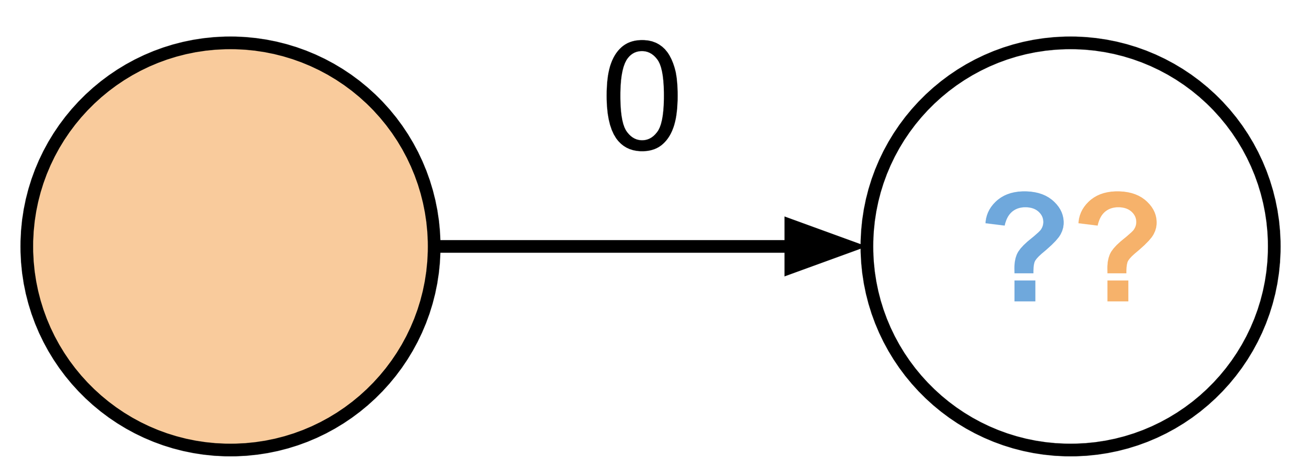
Λ* by Example



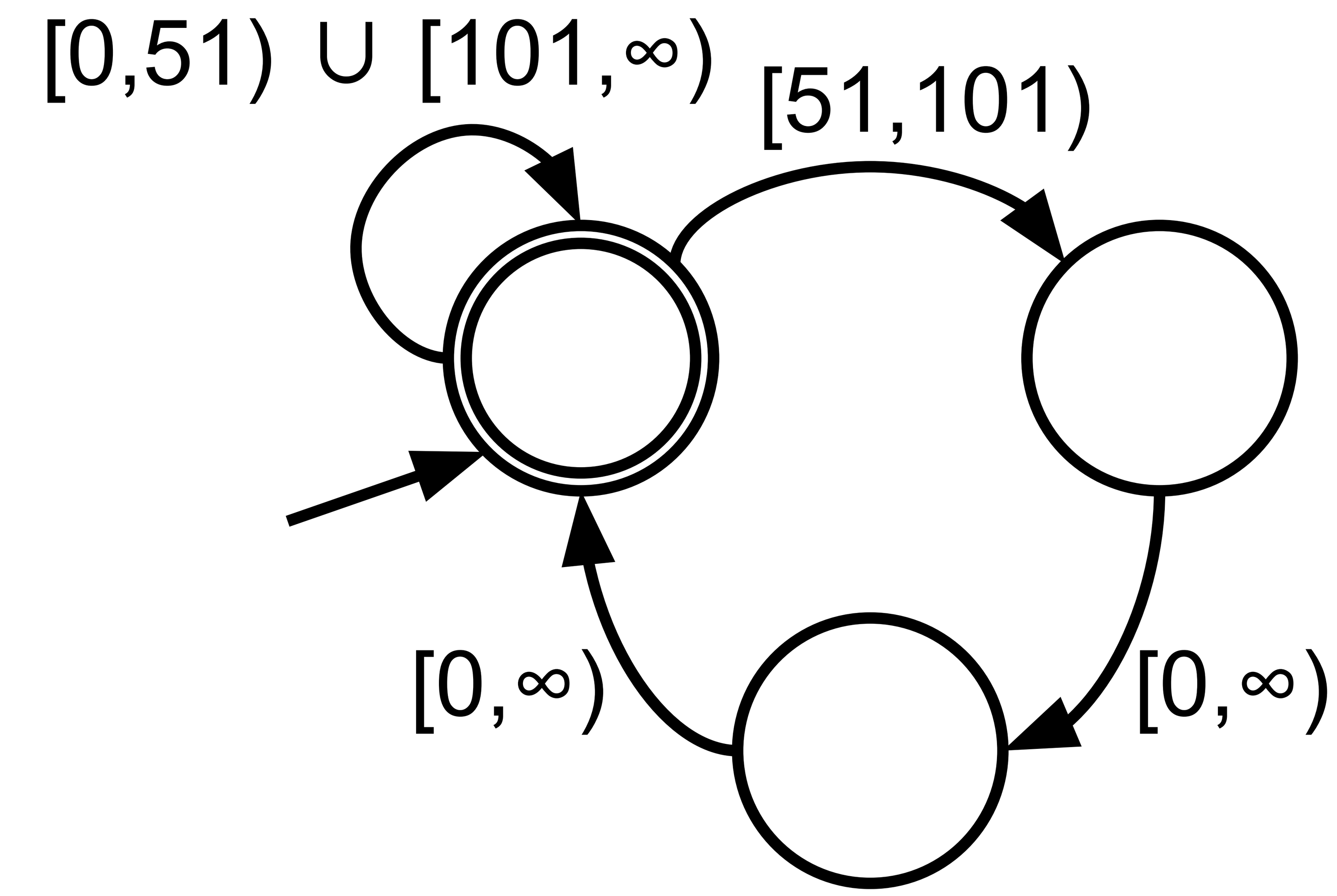
	ε
ε	✓
51	✗
0	✓
51,0	✗
101	✓
51,0,0	✓

51 and **51,0** seem like same state

51·0 and **51,0·0** are different states



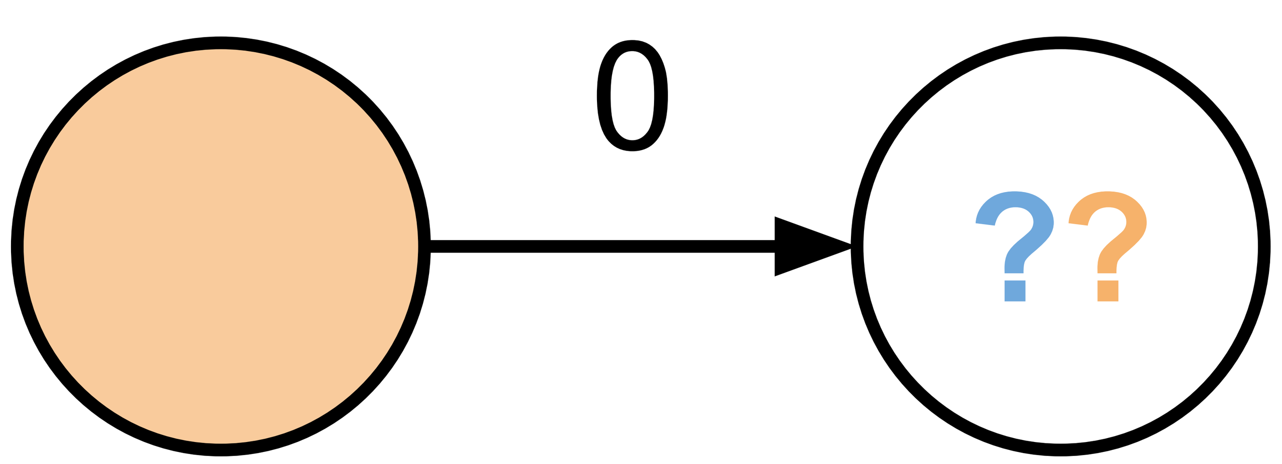
Λ* by Example



	ε
ε	✓
51	✗
0	✓
51,0	✗
101	✓
51,0,0	✓

51 and **51,0** seem like same state

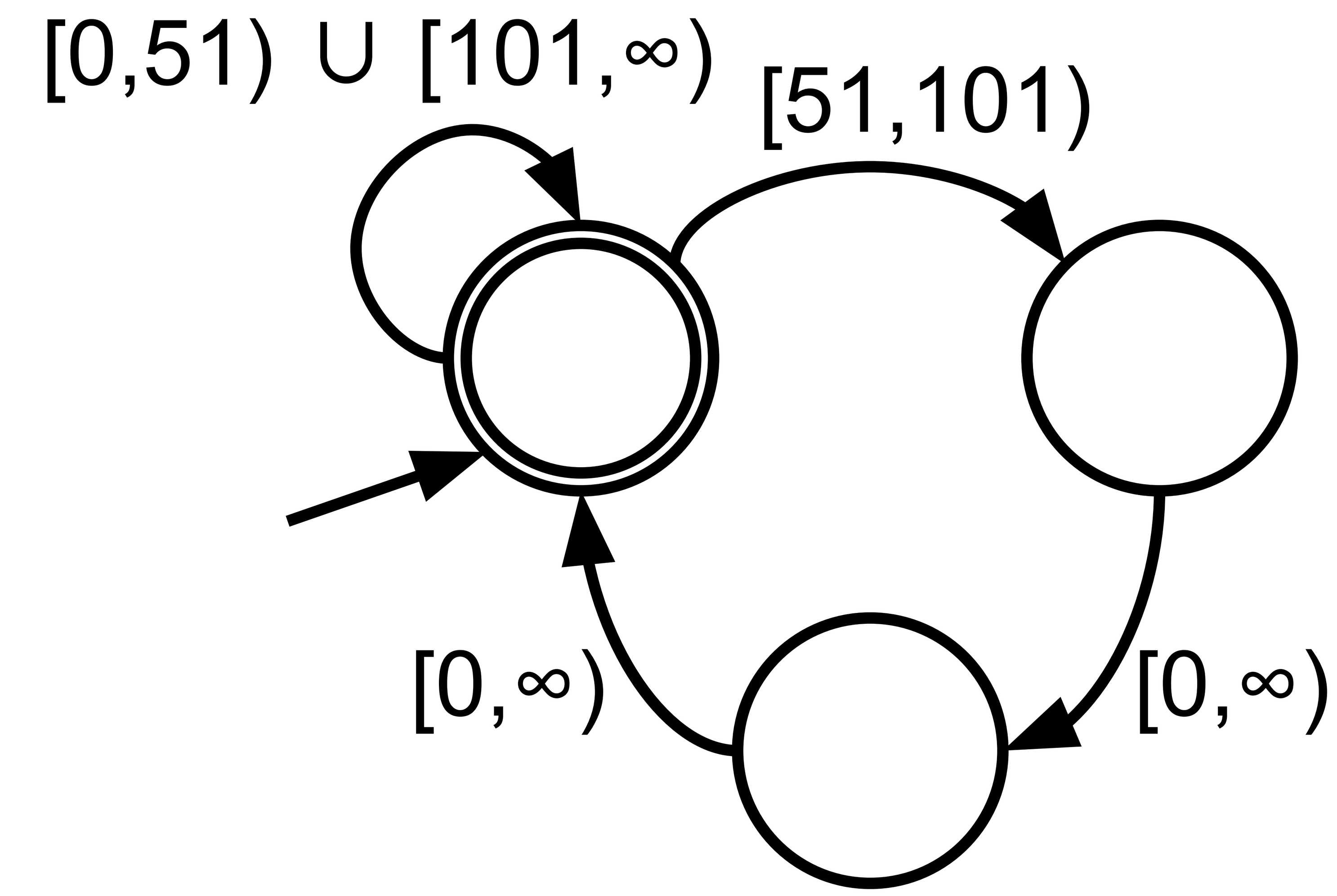
51·0 and **51,0·0** are different states



	ε	0
ε	✓	✓
51	✗	✗
0	✓	✓
51,0	✗	✓
101	✓	✓
51,0,0	✓	✓

Inconsistent: add 0 to E

Λ* by Example

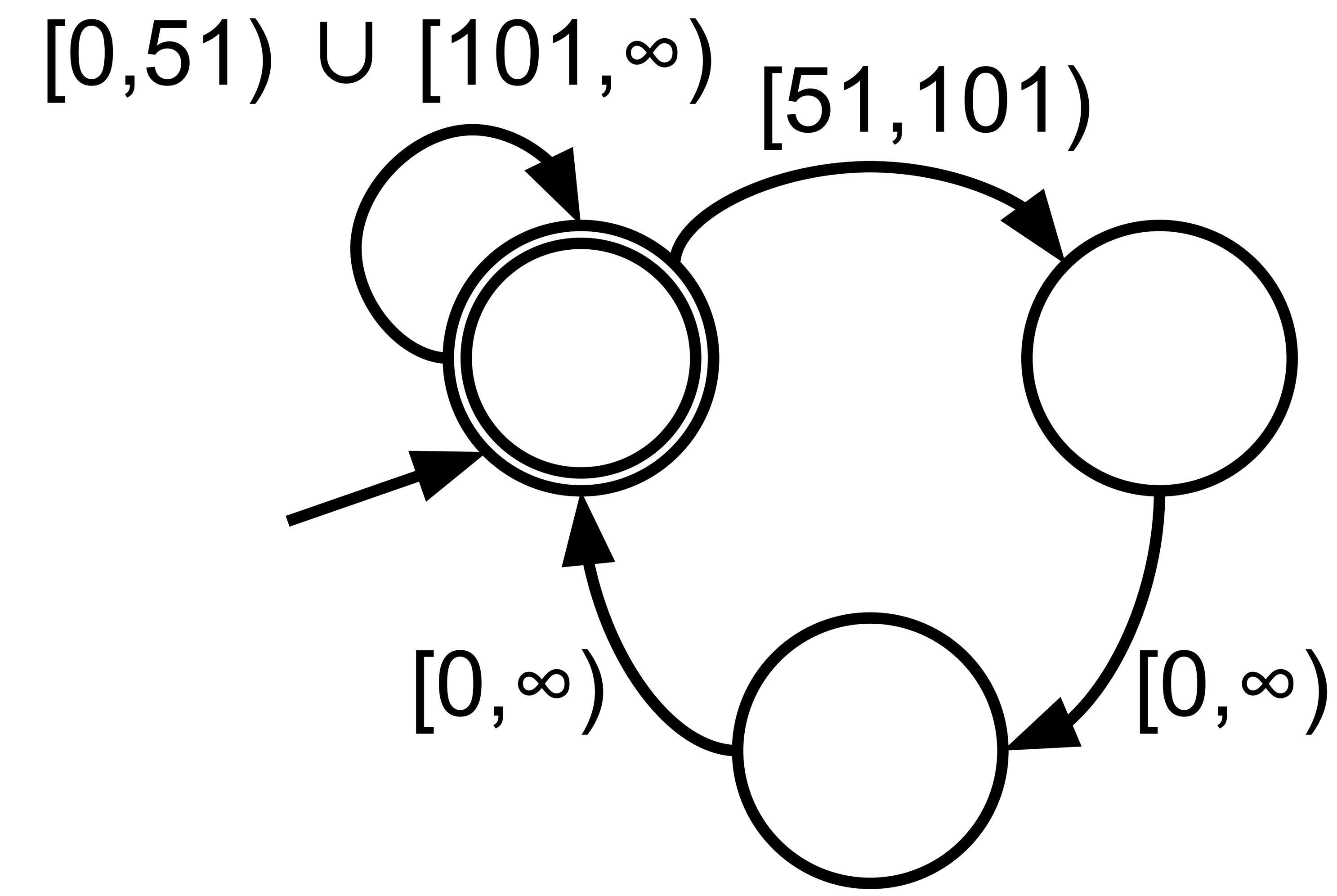


	ε	0
ε	✓	✓
51	✗	✗
0	✓	✓
51,0	✗	✓
101	✓	✓
51,0,0	✓	✓

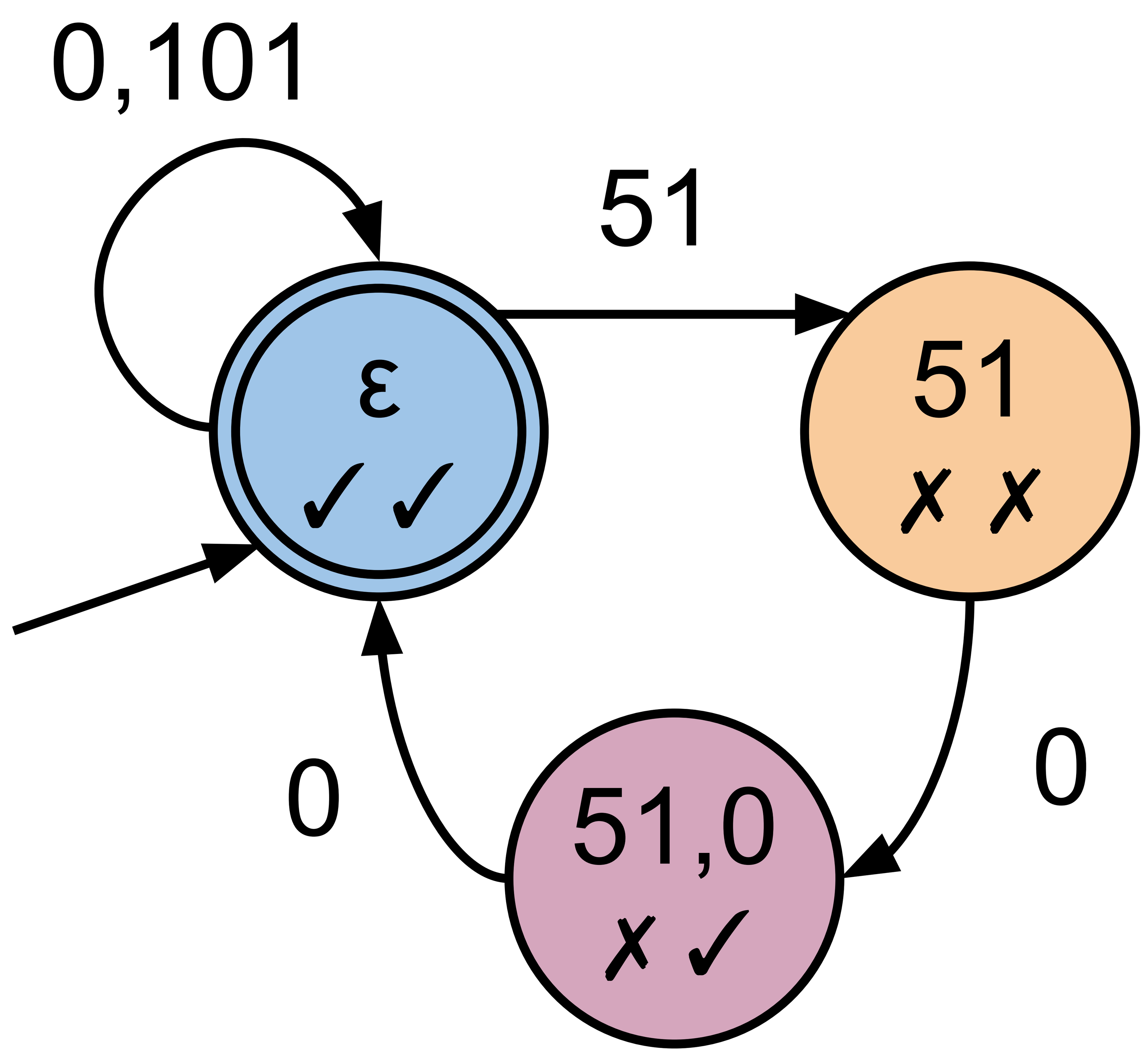
make *closed*
move 51,0 to top

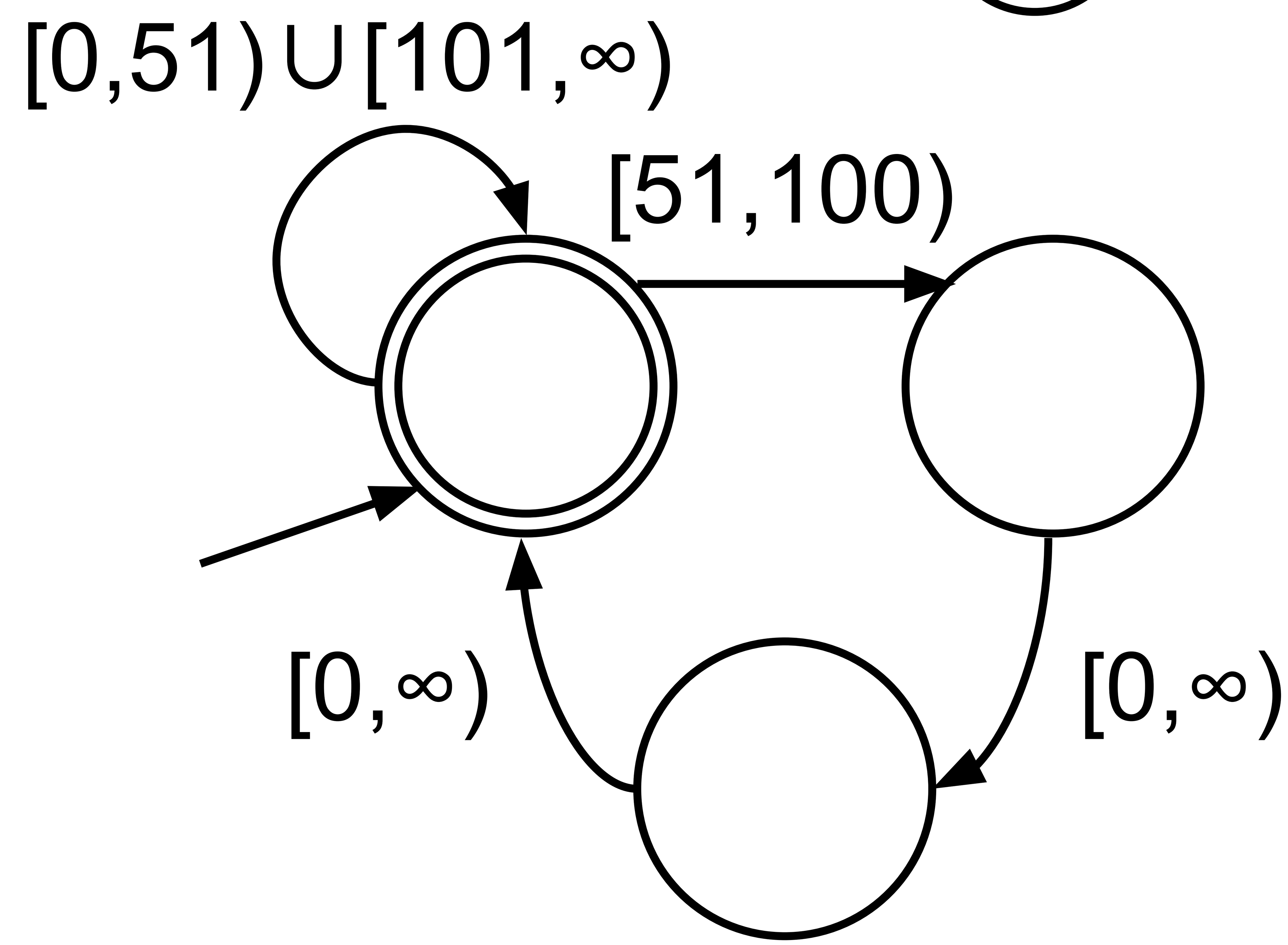
	ε	0
ε	✓	✓
51	✗	✗
51,0	✗	✓
0	✓	✓
101	✓	✓
51,0,0	✓	✓

Λ^* by Example



	ε	0
ε	✓	✓
51	✗	✗
51,0	✗	✓
0	✓	✓
101	✓	✓
51,0,0	✓	✓

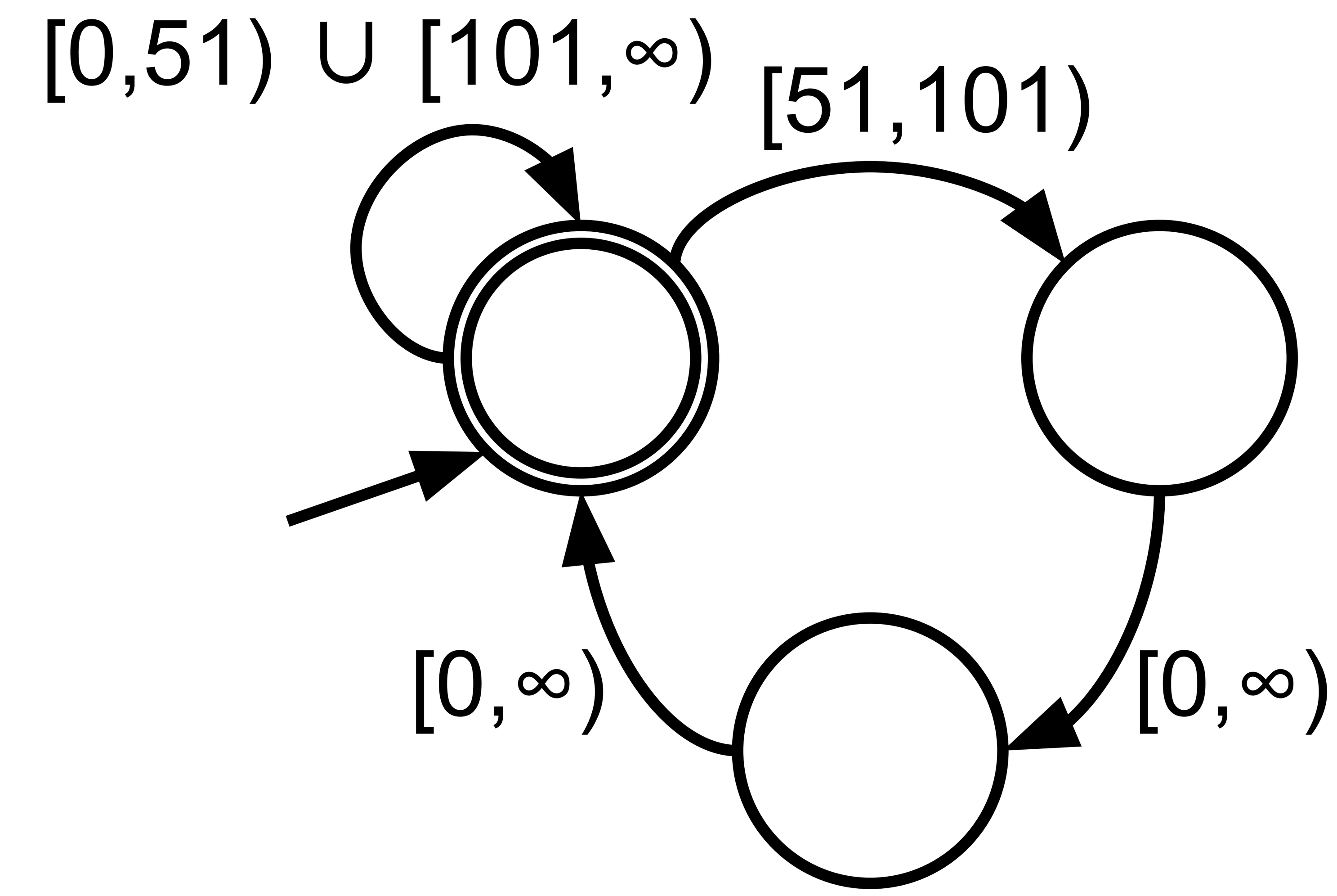




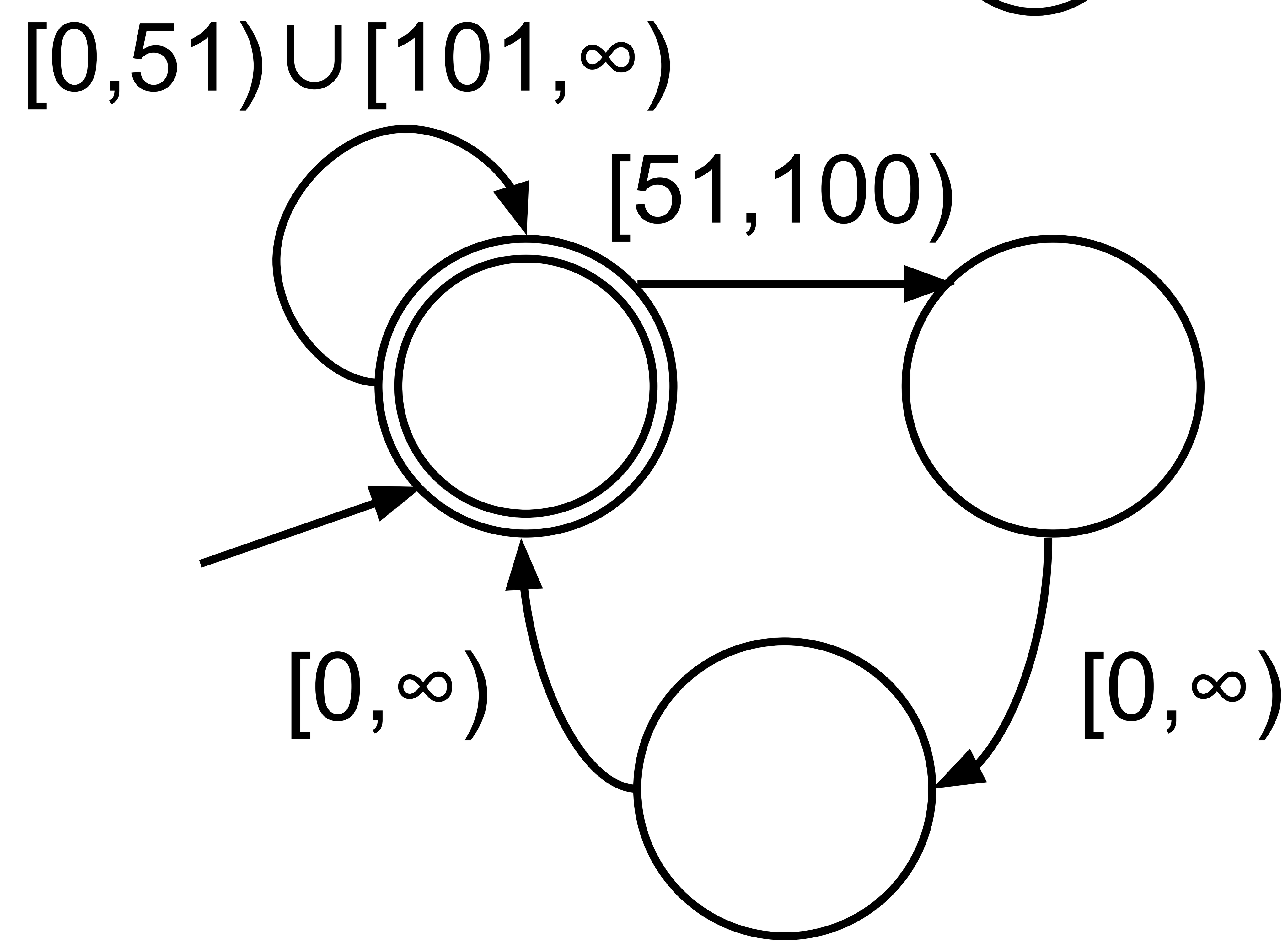
$P(\{0,101\},\{51\}) = [0,51) \cup [101,\infty)\ ,\ [51,101)$

$P(\{0\}) = [0,\infty)$

Λ* by Example

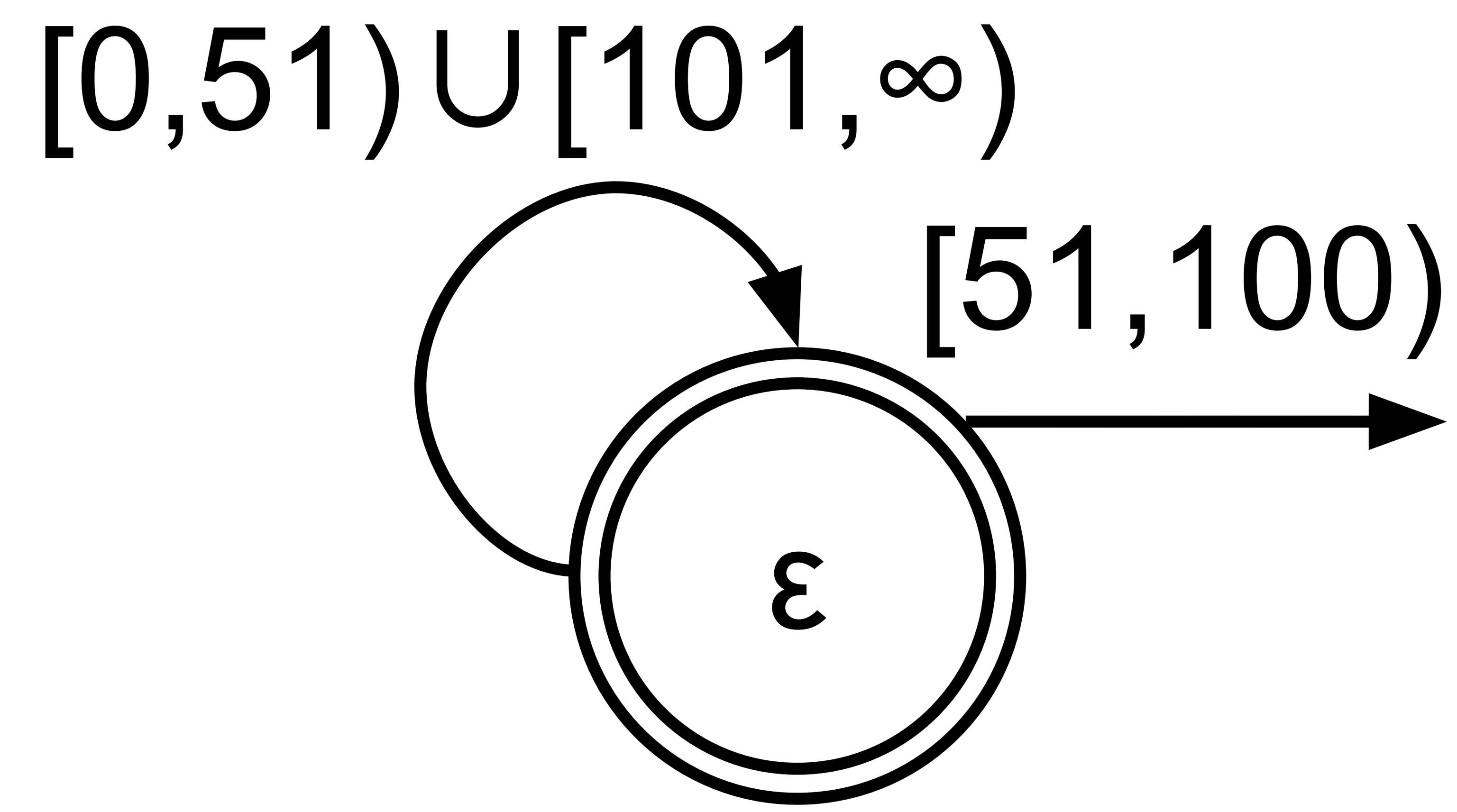


	ε	0
ε	✓	✓
51	✗	✗
51,0	✗	✓
0	✓	✓
101	✓	✓
51,0,0	✓	✓



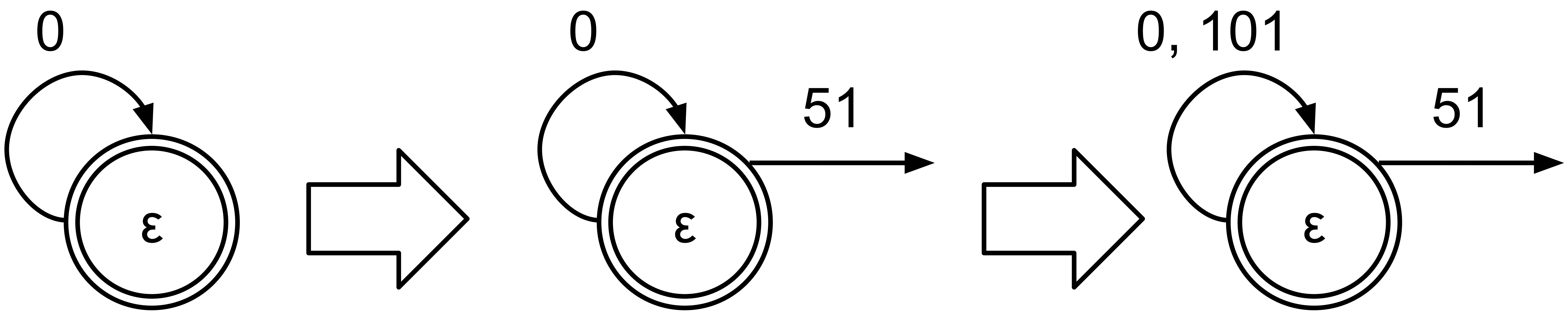
Equivalence query:
Equivalent!

Why did this work?



Infinite alphabet, but finite examples

Oracle gave us good counterexamples



Call this projection of the oracle a *generator*:

[{0}] → [{0},{51}] → [{0,101},{51}]

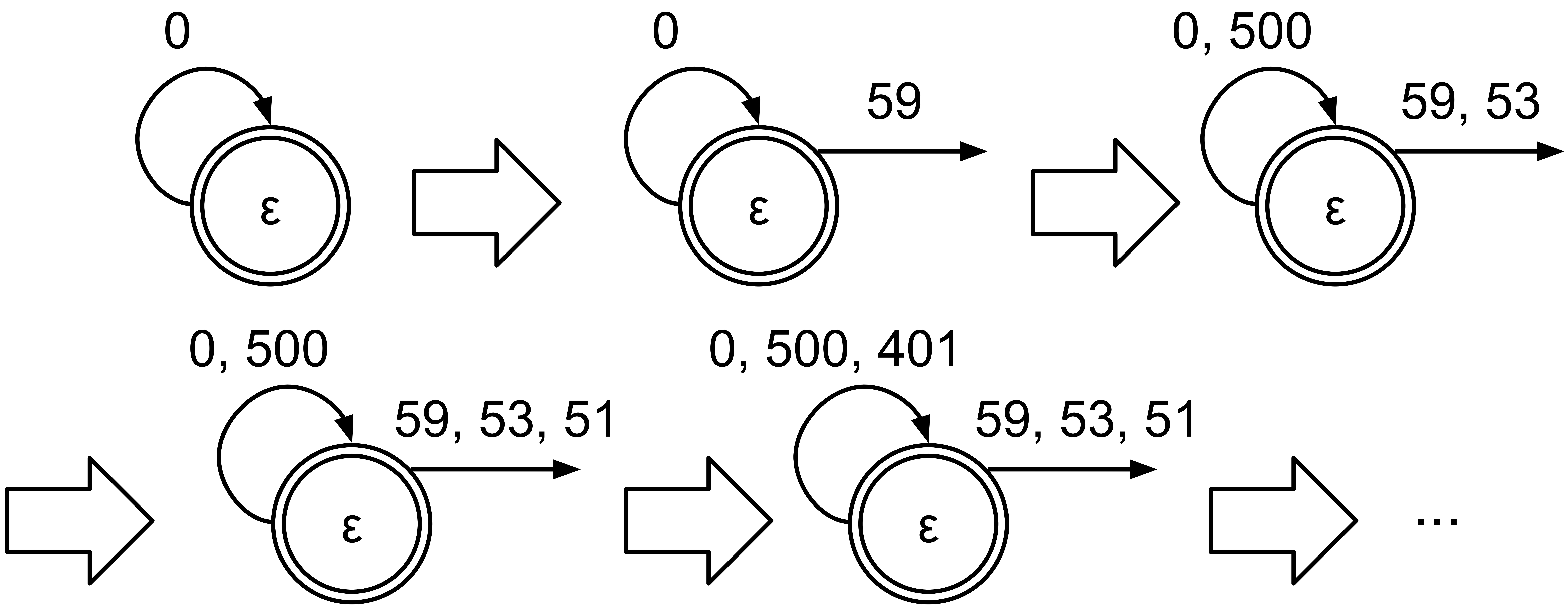
Learnability of Boolean Algebra

Learn automaton with oracle providing Σ^* examples

Learn partition in BA with *generator* providing Σ examples

“Bad” Oracle

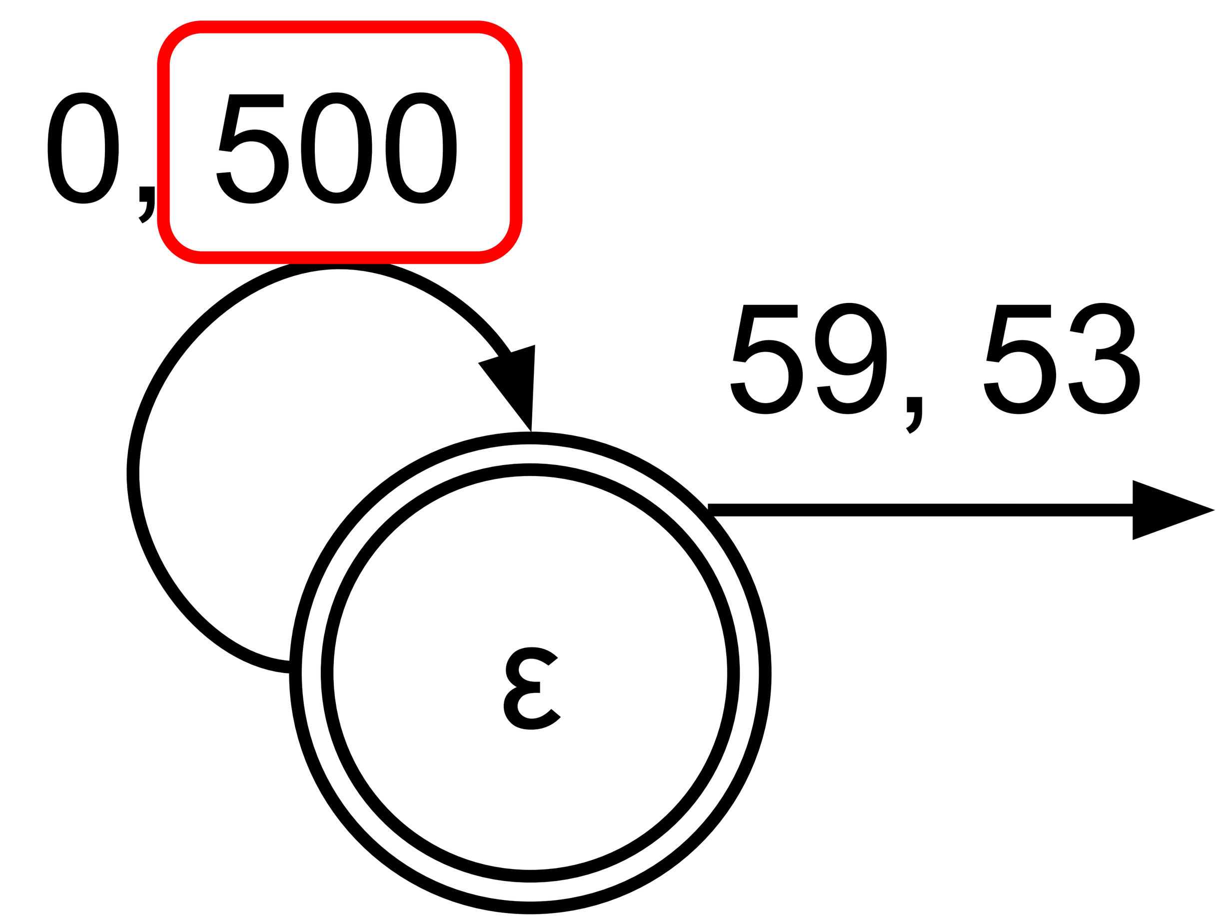
Suppose the oracle does not provide optimal counterexamples



generator: [{0}] → [{0},{59}] → [{0,500},{59,53}] → …

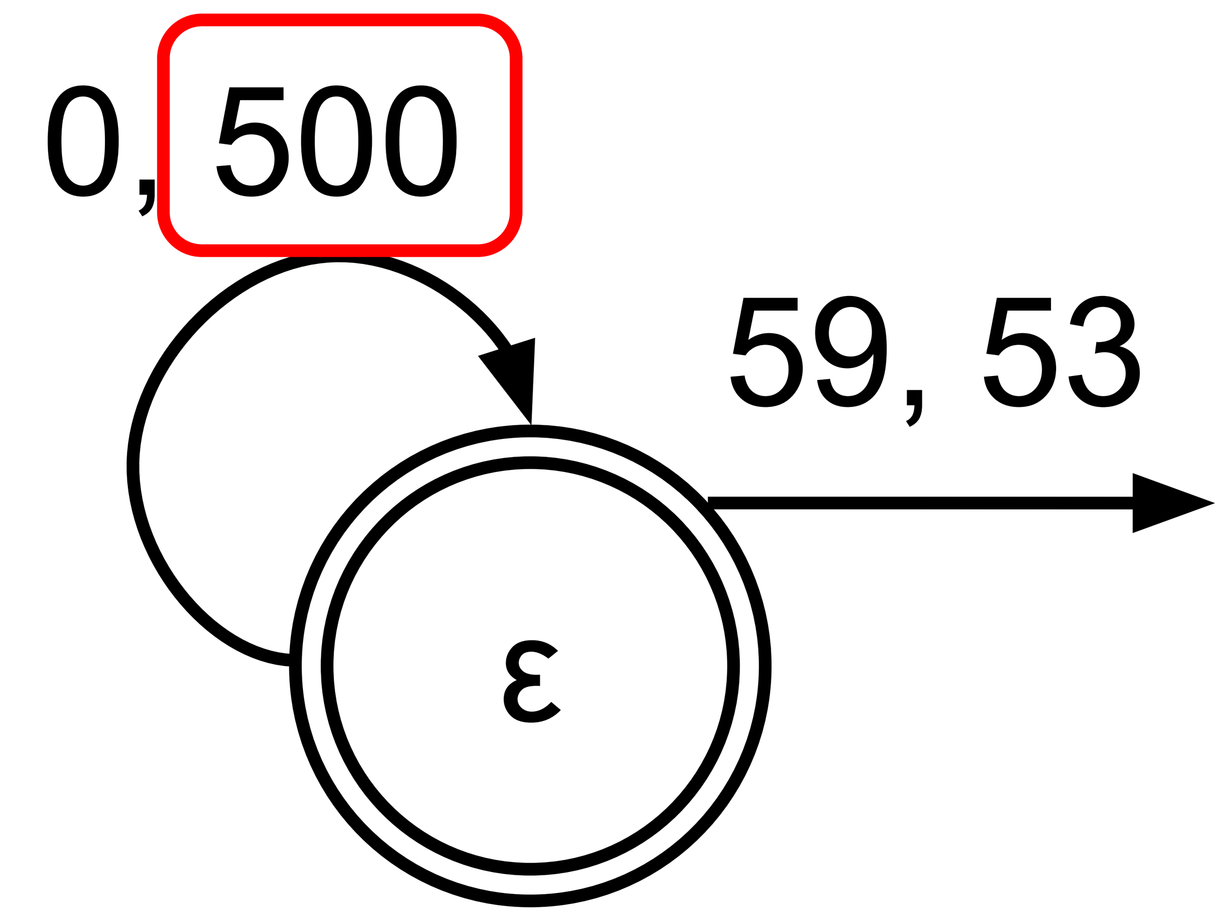
“Bad” Oracle

Suppose the oracle does not provide optimal counterexamples

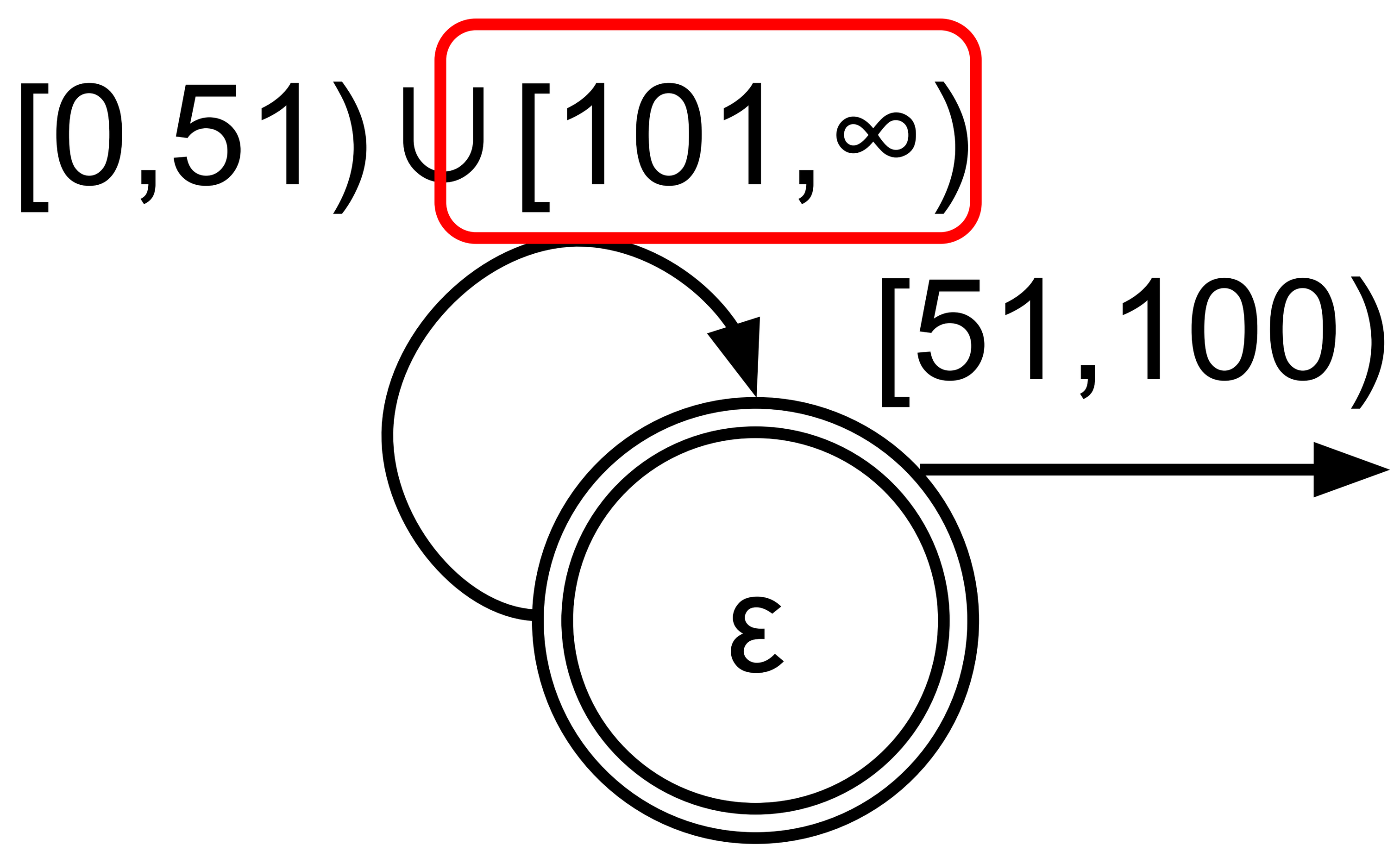


“Bad” Oracle

Suppose the oracle does not provide optimal counterexamples



Partitioning function assumes everything >500 behaves the same as 500



Since 500 > 101, we will never see another example >500

s_g-learnability of Boolean Algebra

c - partition in BA, *g* - generator

Fix a partitioning function P:

define $s_g(c)$ = # examples from *g* needed for P to produce *c*

Ex: $c = [0,51) \cup [101,\infty)\ ,\ [51,101)$

Good examples: $s_g(c) = 3$

Bad examples: $s_{g'}(c) < \infty$

s_g-learnability

\# Equivalence queries to learn symbolic automata *M*

$$\leq n^2 \underbrace{\textstyle\sum_{g,c} s_g(c)}_{\text{oracle examples}}$$

Learning Classes

$$
\begin{array}{ccccc}
C^{\forall}_{\text{constant}} & \subseteq & C^{\forall}_{\text{size}} & \subseteq & C^{\forall}_{\text{finite}} \\
\rotatebox{90}{$\subseteq$} & & \rotatebox{90}{$\subseteq$} & & \rotatebox{90}{$\subseteq$} \\
C^{\exists}_{\text{constant}} & \subseteq & C^{\exists}_{\text{size}} & \subseteq & C^{\exists}_{\text{finite}}
\end{array}
$$

Learning Classes

$$
\begin{array}{ccccc}
C^{\forall}_{\text{constant}} & \subseteq & C^{\forall}_{\text{size}} & \subseteq & C^{\forall}_{\text{finite}} \\
\rotatebox{90}{$\subseteq$} & & \rotatebox{90}{$\subseteq$} & & \rotatebox{90}{$\subseteq$} \\
C^{\exists}_{\text{constant}} & \subseteq & C^{\exists}_{\text{size}} & \subseteq & C^{\exists}_{\text{finite}}
\end{array}
$$

There exists a generator: any partition is learned from a constant # examples

Learning Classes

For every generator:
any partition is learned from
a # examples based on the
size of the partition

$$
\begin{array}{ccccc}
C^{\forall}_{\text{constant}} & \subseteq & C^{\forall}_{\text{size}} & \subseteq & C^{\forall}_{\text{finite}} \\
\rotatebox{-90}{$\subseteq$} & & \rotatebox{-90}{$\subseteq$} & & \rotatebox{-90}{$\subseteq$} \\
C^{\exists}_{\text{constant}} & \subseteq & C^{\exists}_{\text{size}} & \subseteq & C^{\exists}_{\text{finite}}
\end{array}
$$

There exists a generator:
any partition is learned from
a constant # examples

Learning Classes

For every generator:
any partition is learned from
a # examples based on the
size of the partition

$$\begin{array}{ccccc} C^{\forall}_{constant} & \subseteq & C^{\forall}_{size} & \subseteq & C^{\forall}_{finite} \\ \rotatebox{-90}{$\subseteq$} & & \rotatebox{-90}{$\subseteq$} & & \rotatebox{-90}{$\subseteq$} \\ C^{\exists}_{constant} & \subseteq & C^{\exists}_{size} & \subseteq & C^{\exists}_{finite} \end{array}$$

There exists a generator:
any partition is learned from
a constant # examples

Λ^* example

Composition of Boolean Algebras

We have a non-negative integer partitioning function in $C^{\exists}_{size}$

Can we learn partitions over *all* integers?

Disjoint union: $Z \cong Z_{<0} \uplus Z_{\geq 0}$

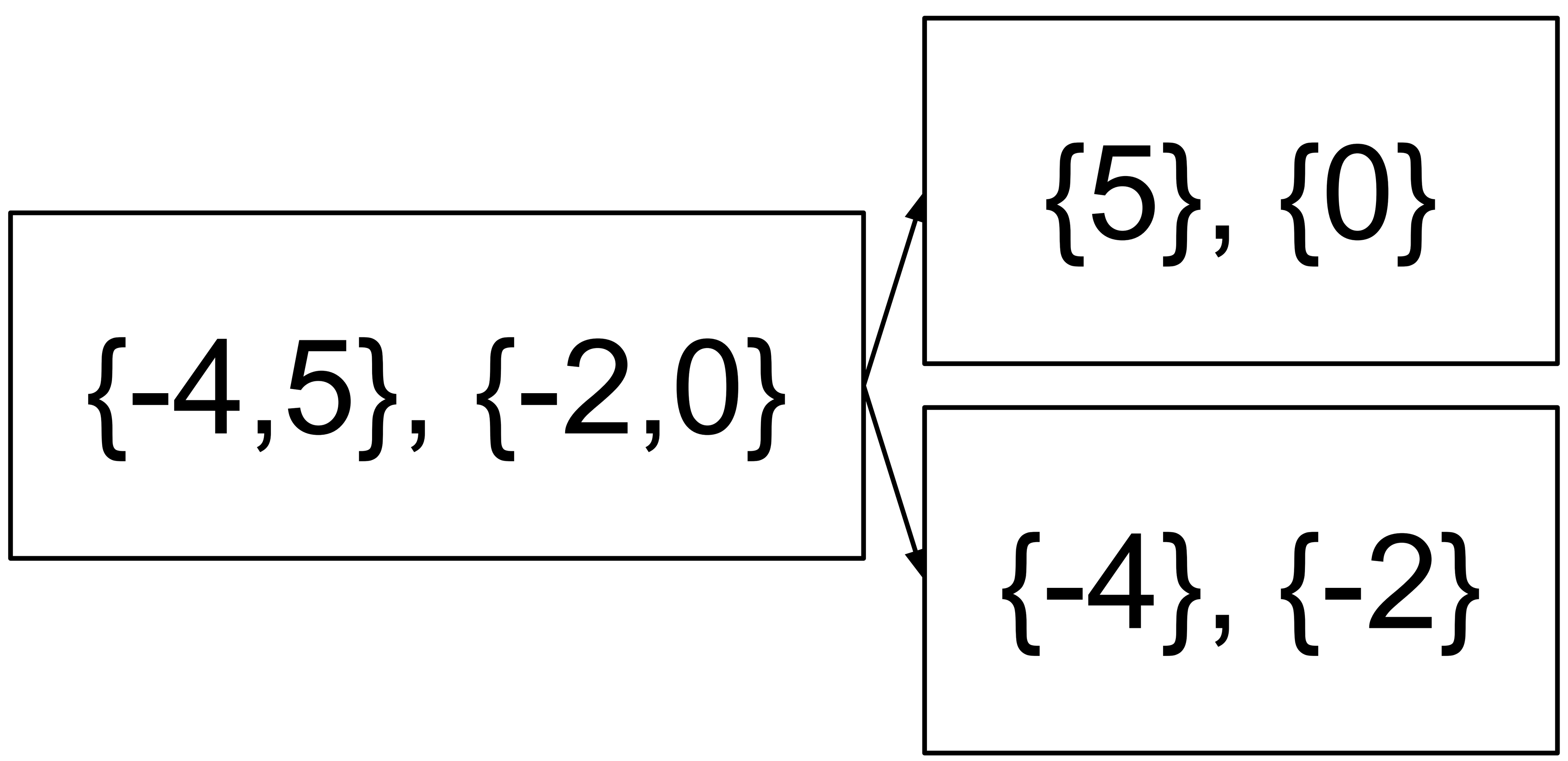
{-4,5}, {-2,0}

Composition of Boolean Algebras

We have a non-negative integer partitioning function in $C^{\exists}_{size}$

Can we learn partitions over *all* integers?

Disjoint union: $Z \cong Z_{<0} \uplus Z_{\geq 0}$

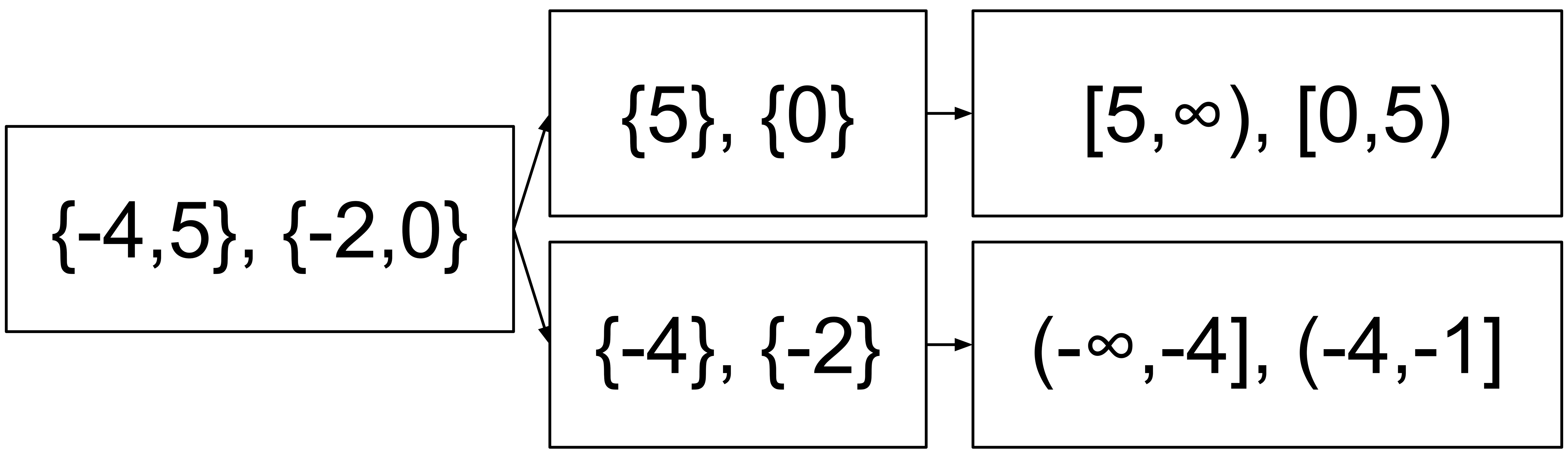


Composition of Boolean Algebras

We have a non-negative integer partitioning function in $C^{\exists}_{size}$

Can we learn partitions over *all* integers?

Disjoint union: $Z \cong Z_{<0} \uplus Z_{\geq 0}$



Composition of Boolean Algebras

We have a non-negative integer partitioning function in $C^{\exists}_{size}$

Can we learn partitions over *all* integers?

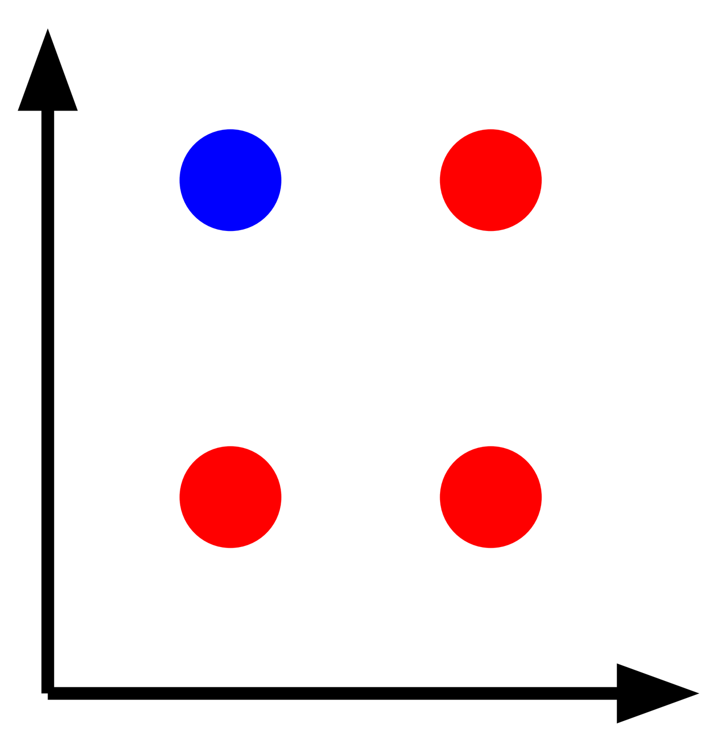
Disjoint union: $Z \cong Z_{<0} \uplus Z_{\geq 0}$

Composition of Boolean Algebras

We can learn partitions over all integers Z

Can we learn partitions over Z^2?

Cartesian product: $Z^2 \cong Z \times Z$

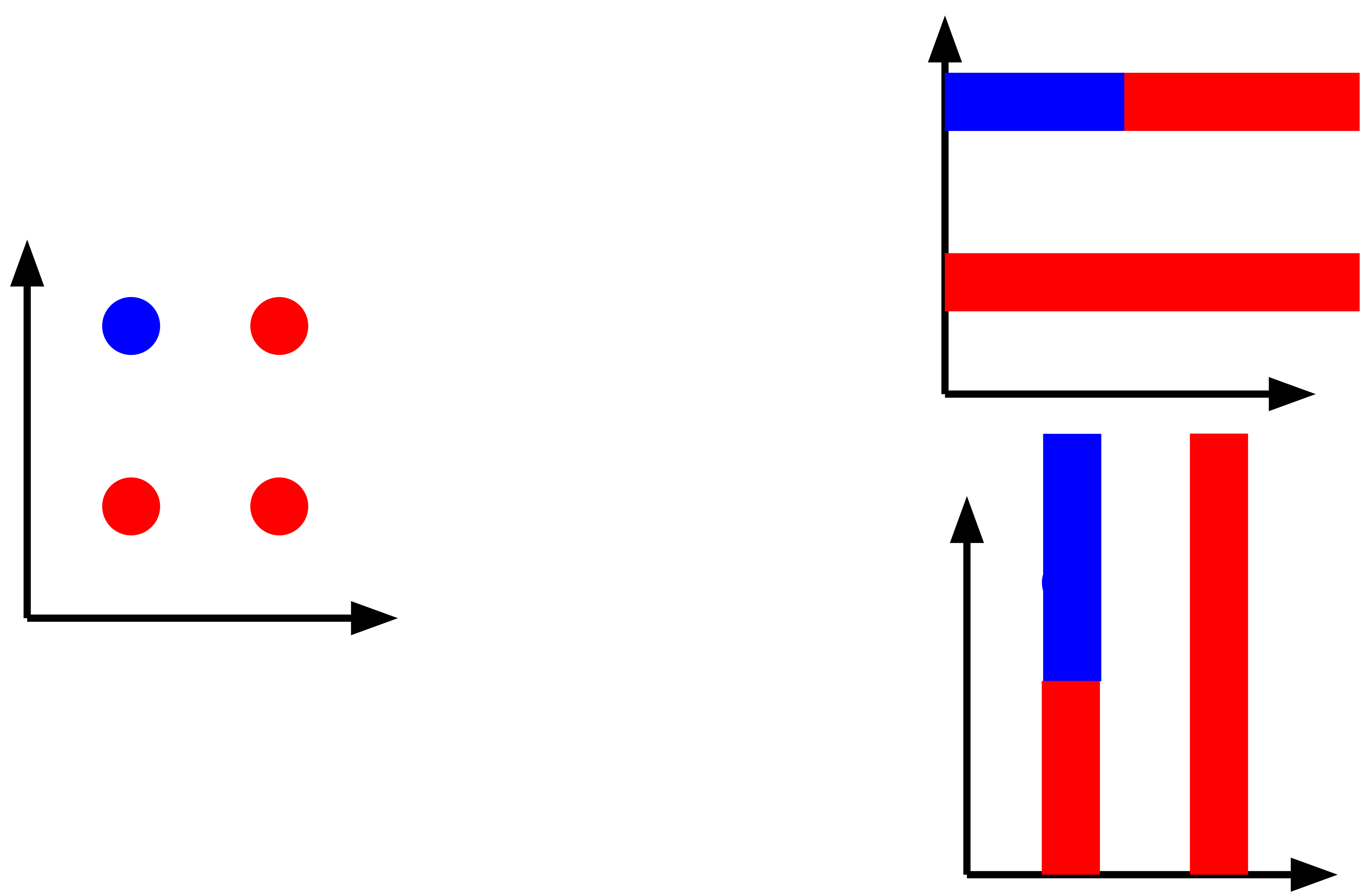


Composition of Boolean Algebras

We can learn partitions over all integers Z

Can we learn partitions over Z^2?

Cartesian product: $Z^2 \cong Z \times Z$

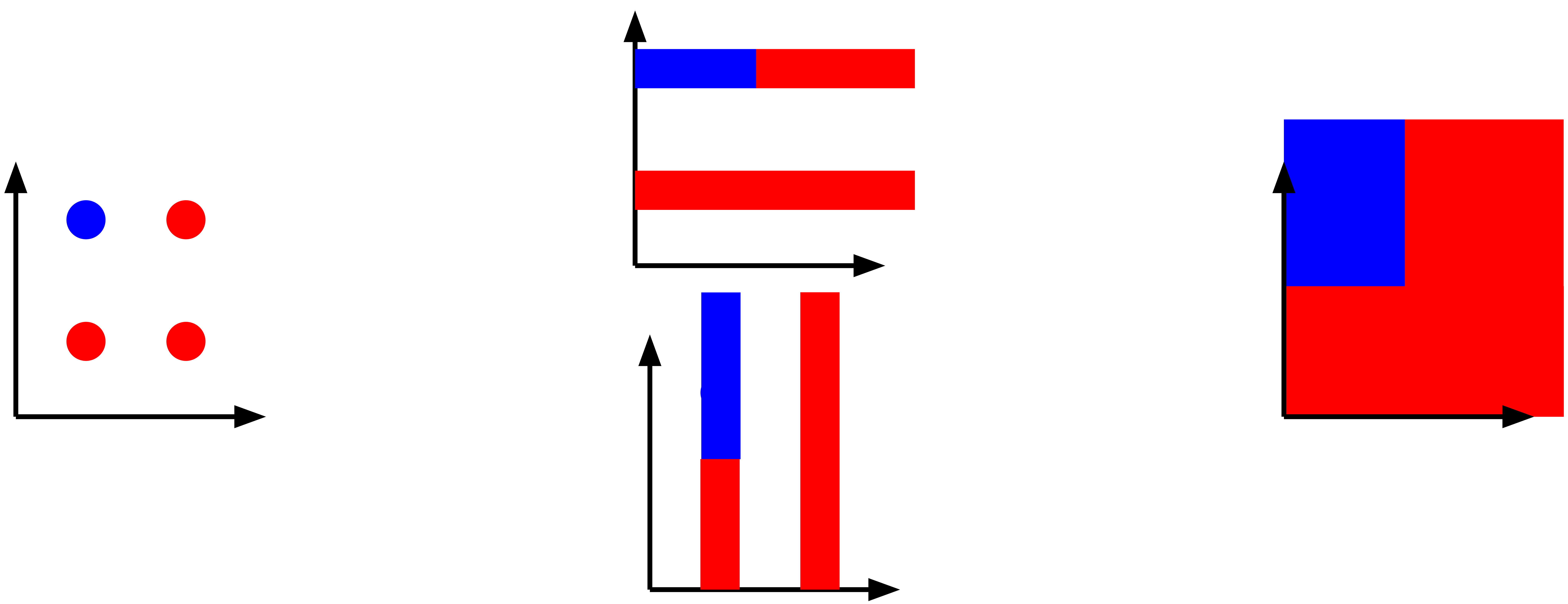


Composition of Boolean Algebras

We can learn partitions over all integers Z

Can we learn partitions over Z^2?

Cartesian product: $Z^2 \cong Z \times Z$



Composition of Boolean Algebras

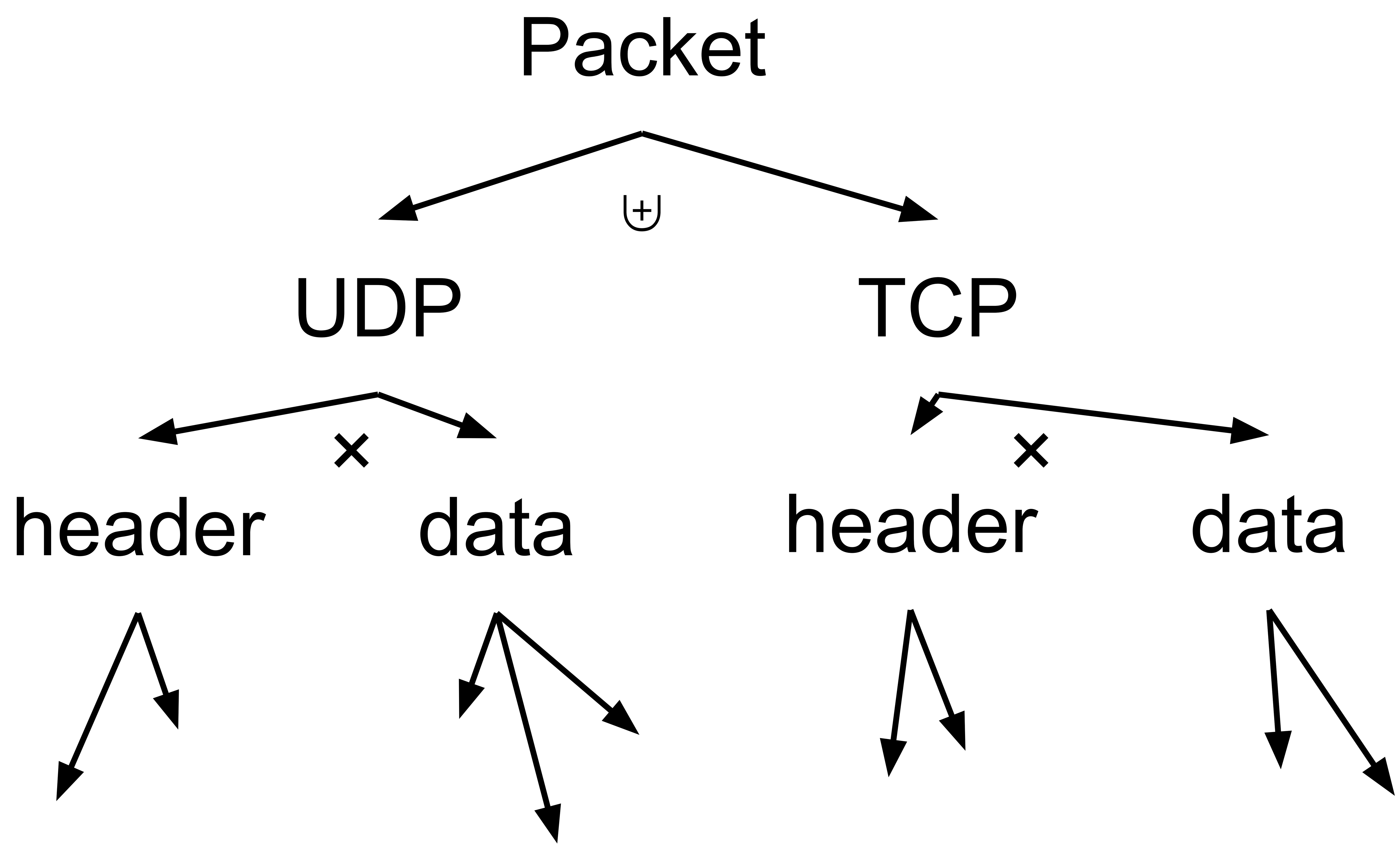
If BA_1 and BA_2 are Boolean Algebras in learning class *C*

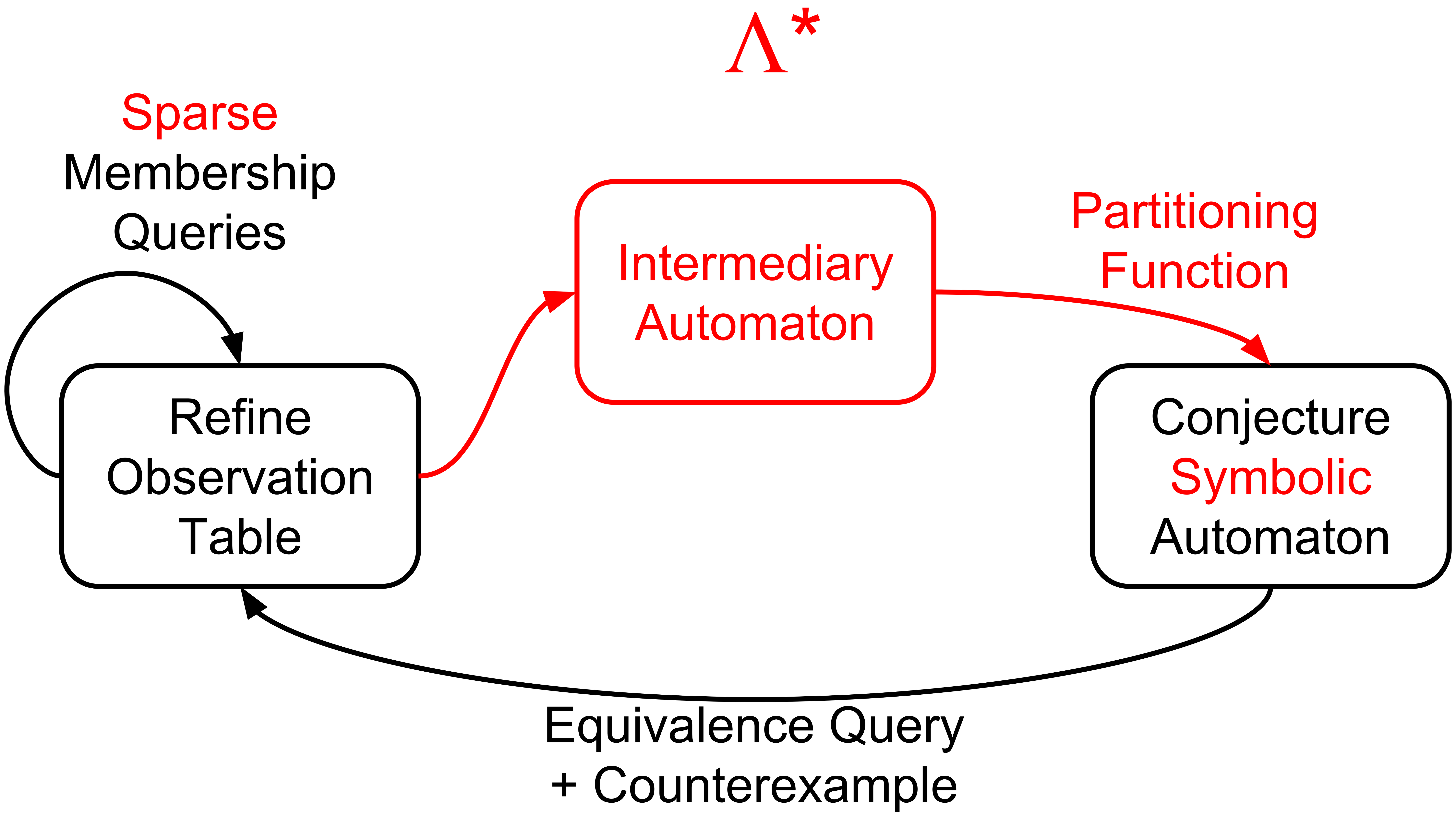
$BA_1 \uplus BA_2$ is in *C*

$BA_1 \times BA_2$ is in *C*

Composition of Boolean Algebras

Learning an automaton over strings of network packets





$$
\begin{array}{ccccc}
C^{\forall}_{\text{constant}} & \subseteq & C^{\forall}_{\text{size}} & \subseteq & C^{\forall}_{\text{finite}} \\
\rotatebox{90}{$\supseteq$} & & \rotatebox{90}{$\supseteq$} & & \rotatebox{90}{$\supseteq$} \\
C^{\exists}_{\text{constant}} & \subseteq & C^{\exists}_{\text{size}} & \subseteq & C^{\exists}_{\text{finite}}
\end{array}
$$