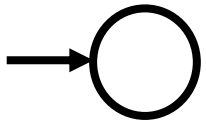
CSC 6011: Theory of Computation

Lecture 1



Time: Mon & Wed, 4:00 - 5:20 PM

Location: Teaching A, 107

Instructor: Tao LIN (林涛） lintao@cuhk.edu.cn

TA: Xingyu WANG (王星宇) 225040497@link.cuhk.edu.cn

Central Question:

What are the fundamental capabilities and limitations of computers?

A question since the 1930s

Still relevant in the present days and the future

CSC 6011: Course Outline

Computability Theory (1930s – 1950s)

- What is computable… or not?
- Examples:
  halting problem, mathematical truth
- Models of Computation:
  Finite automata, Turing machines, …

Other topics:

- Approximation algorithms
- Communication complexity
- Connections with modern AI
- …

Complexity Theory (1960s – present)

- What is computable in practice?
- Example: *factoring problem*
- P versus NP problem
- Measures of complexity: *Time* & *Space*
- Models: *Probabilistic* & *Interactive computation*

An Outstanding Paper on ICLR 2026

(International Conference on Learning Representation)

TRANSFORMERS ARE INHERENTLY SUCCINCT

Pascal Bergsträßer
RPTU Kaiserslautern-Landau
Kaiserslautern, Germany

Ryan Cotterell
ETH Zürich
Zurich, Switzerland

Anthony W. Lin
RPTU Kaiserslautern-Landau and MPI-SWS
Kaiserslautern, Germany

ABSTRACT

We study *succinctness* as a measure of the expressive power of transformers. Succinctness—how compactly a formalism can describe a language relative to other formalisms—is a classical notion in logic and automata theory. We prove that fixed-precision transformers are remarkably succinct: they can be exponentially more succinct than both linear temporal logic (LTL) and recurrent neural networks, and, by extension, state-space models, and doubly exponentially more succinct than finite automata. In other words, there exist families of languages describable by polynomial-size transformers whose smallest equivalent LTL formula or recurrent neural network is exponentially large, and whose smallest equivalent automaton is doubly exponentially large. We also establish matching upper bounds, showing that any fixed-precision transformer can be converted to an LTL formula with at most an exponential blow-up—improving a prior doubly exponential translation. As a consequence of this succinctness, we show that basic verification problems for transformers, such as emptiness and equivalence, are provably intractable: specifically, EXPSPACE-complete.

Tentative outline

Week	Content/ topic/ activity
1 ~ 2	Introduction.
	Finite automata (DFA and NFA), regular languages.
3	Context-free languages, pushdown automata.
4	Turing machines, the Church-Turing thesis.
5	Decidability, diagonalization.
6	Reduction. Time complexity.
7	P and NP, NP-completeness.
8	Cook-Levin theorem. [Midterm Exam]
9 ~ 10	Space complexity, PSPACE, L, NL, Savitch's theorem.
	[ICLR'26 paper]: Transformers & EXPSPACE-completeness.
11	Randomized computation, RP, BPP.
12	Interactive proofs, IP = PSPACE.
13	Approximation algorithms, hardness of approximation, PCP theorem.
14	Communication complexity.

Course Logistics

Prerequisites:

Prior experience and comfort with mathematical concepts, theorems, and proofs.

Reference books

- ***Introduction to the Theory of Computation*** Michael Sipser, 3rd Edition.
- *Introduction to Automata Theory, Languages, and Computation*, by John Hopcroft, Rajeev Motwani, and Jeffrey Ullman.
- *Communication Complexity*, by Eyal Kushilevitz and Noam Nisan.
- *Computational Complexity: A Modern Approach*, by Sanjeev Arora and Boaz Barak.

Homework – 30%

- Biweekly (tentative)
- AI usage is allowed.
- But try yourself first.
- Write up your own solutions.

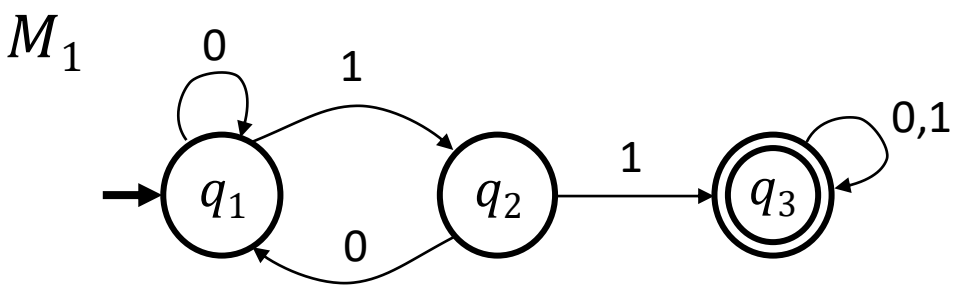
Midterm (30%) and Final exam (40%)

- Open book and notes

Office hour (tentative)

- Tao Lin: Thursday 5 – 6 pm, Daoyuan Building (道远楼) 420c
- Xingyu Wang: Wed 7 – 9 pm, Zhixin Building (知新楼) 247

Let's begin: Finite Automata



States: q_1 q_2 q_3

Transitions: 1 →

Start state: →○

Accept states: ◎

Input: finite string

Output: Accept or Reject

Computation process:

- Begin at start state,
- read input symbols,
- follow corresponding transitions,
- Accept if end with accept state, Reject if not.

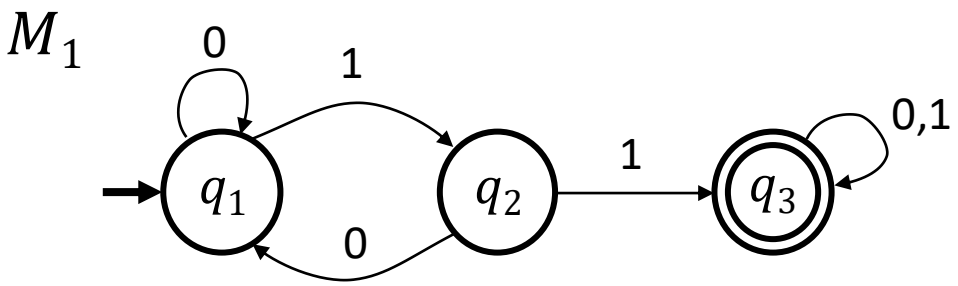
Examples: 01101 → Accept

00101 → Reject

Question:

What is the *set of strings* that M_1 accepts?

Let's begin: Finite Automata



States: q_1 q_2 q_3

Transitions: 1 →

Start state: →○

Accept states: ◎

Input: finite string

Output: Accept or Reject

Computation process:

- Begin at start state,
- read input symbols,
- follow corresponding transitions,
- Accept if end with accept state, Reject if not.

Examples: 01101 → Accept

00101 → Reject

Answer: M_1 accepts exactly the strings in the set

$A = \{w \mid w \text{ contains substring } 11\}$.

Terminology: We say that A is the language of M_1 or M_1 recognizes A or $A = L(M_1)$.

Finite Automata – Formal Definition

Defn: A finite automaton M is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$

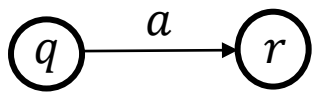
Q finite set of states

Σ alphabet: finite set of symbols

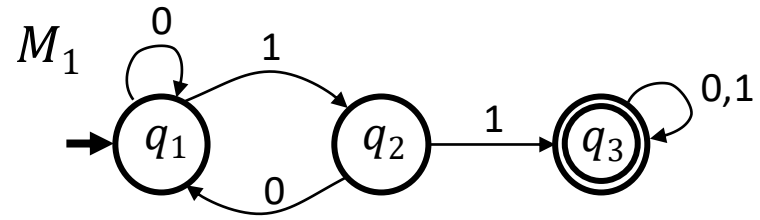
δ transition function $\delta: Q \times \Sigma \rightarrow Q$

q_0 start state

F set of accept states

$\delta(q, a) = r$ means 

Example:



$M_1 = (Q, \Sigma, \delta, q_1, F)$

$Q = \{q_1, q_2, q_3\}$

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

$F = \{q_3\}$

$\delta =$

	0	1
q_1	q_1	q_2
q_2	q_1	q_3
q_3	q_3	q_3

Finite Automata – Computation

Strings and languages

- A string is a finite sequence of symbols in Σ
- A language is a (finite or infinite) set of strings
- The empty string ε is the string of length 0
- The empty language $\emptyset$ is the set with no strings:
  - Do not confuse with ε and $\{\varepsilon\}$

Recognizing languages

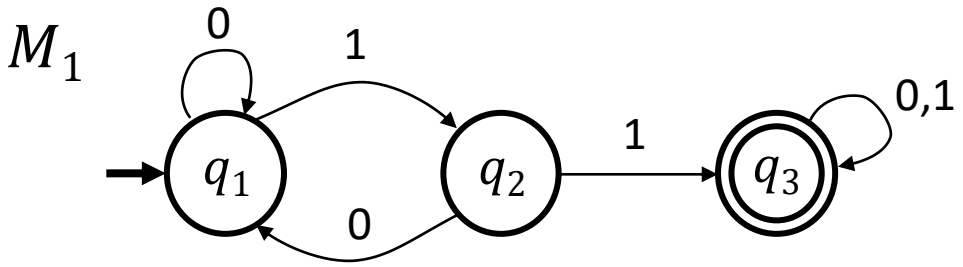
- $L(M) = \{w \mid M \text{ accepts } w\}$
- $L(M)$ is the language of M
- M recognizes $L(M)$

Defn: M accepts string $w = w_1 w_2 \ldots w_n$ each $w_i \in \Sigma$ if there is a sequence of states $r_0, r_1, r_2, , \ldots, r_n \in Q$ where:

- $r_0 = q_0$
- $r_i = \delta(r_{i-1}, w_i)$ for $1 \leq i \leq n$
- $r_n \in F$

Defn: A language is regular if some finite automaton recognizes it.

Regular Languages – Examples



$L(M_1) = \{w \mid w \text{ contains substring } 11\} = A$

Therefore A is regular

More examples:

Let $B = \{w \mid w \text{ has an even number of 1s}\}$
B is regular (make automaton for practice).

Let $C = \{w \mid w \text{ has equal numbers of 0s and 1s}\}$
C is not regular (we will prove).

Goal: Understand regular languages

- What languages are regular (i.e., recognized by finite automata)?
- What properties do they have?

Regular Expressions

Regular operations. Let A, B be languages:

- **Union:** $A \cup B = \{w | \ w \in A \text{ or } w \in B\}$
- **Concatenation:** $A \circ B = \{xy | \ x \in A \text{ and } y \in B\} = AB$
- **(Kleene) Star:** $A^* = \{x_1 \ldots x_k | \text{ each } x_i \in A \text{ for } k \geq 0\}$

  Note: $\varepsilon \in A^*$ always

Example. Let $A = \{\text{good, bad}\}$ and $B = \{\text{boy, girl}\}$.

- $A \cup B = \{\text{good, bad, boy, girl}\}$
- $A \circ B = AB = \{\text{goodboy, goodgirl, badboy, badgirl}\}$
- $A^* = \{\varepsilon, \text{good, bad, goodgood, goodbad, badgood, badbad, goodgoodgood, goodgoodbad, } \ldots\}$

Regular expressions:

- Σ, members in $\Sigma, \emptyset, \varepsilon$ [Atomic]
- By using $\cup, \circ, *$ [Composite]

Examples:

- $(0 \cup 1)^* = \Sigma^*$ gives all strings over Σ
- $\Sigma^* 1$ gives all strings that end with 1
- $\Sigma^* 11 \Sigma^*$ = all strings that contain 11
  $= L(M_1)$

Kleene's Theorem: Finite automata are equivalent to regular expressions.

We will prove this in the next two lectures.

Closure Properties of Regular Languages

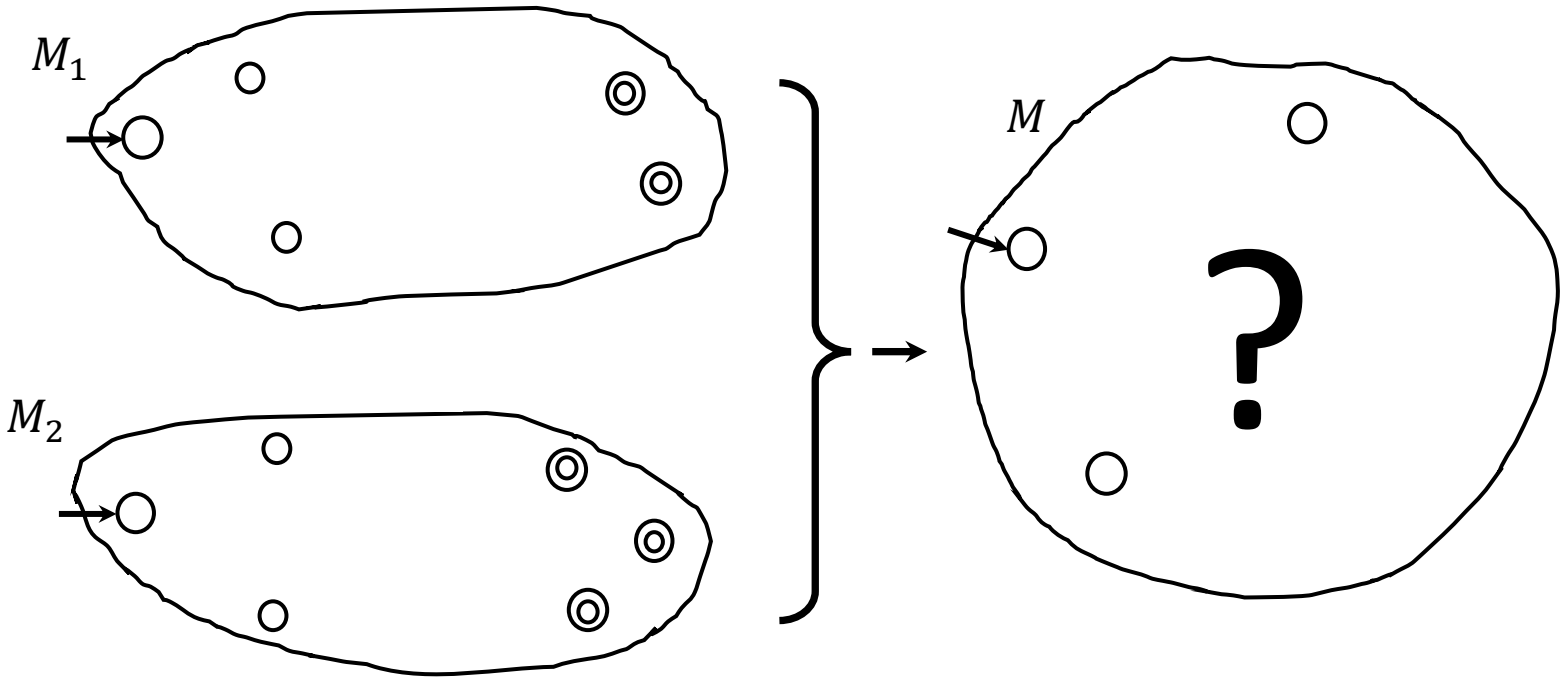
Theorem (closure under ∪)**:** If A_1, A_2 are regular languages, so is $A_1 \cup A_2$

Proof: Let $M_1 = (Q_1, \Sigma, \delta_1, q_1^s, F_1)$ recognize A_1

$M_2 = (Q_2, \Sigma, \delta_2, q_2^s, F_2)$ recognize A_2

Goal: Construct $M = (Q, \Sigma, \delta, q_0, F)$ recognizing $A_1 \cup A_2$

M should accept input w if either M_1 or M_2 accept w.



Closure Properties of Regular Languages

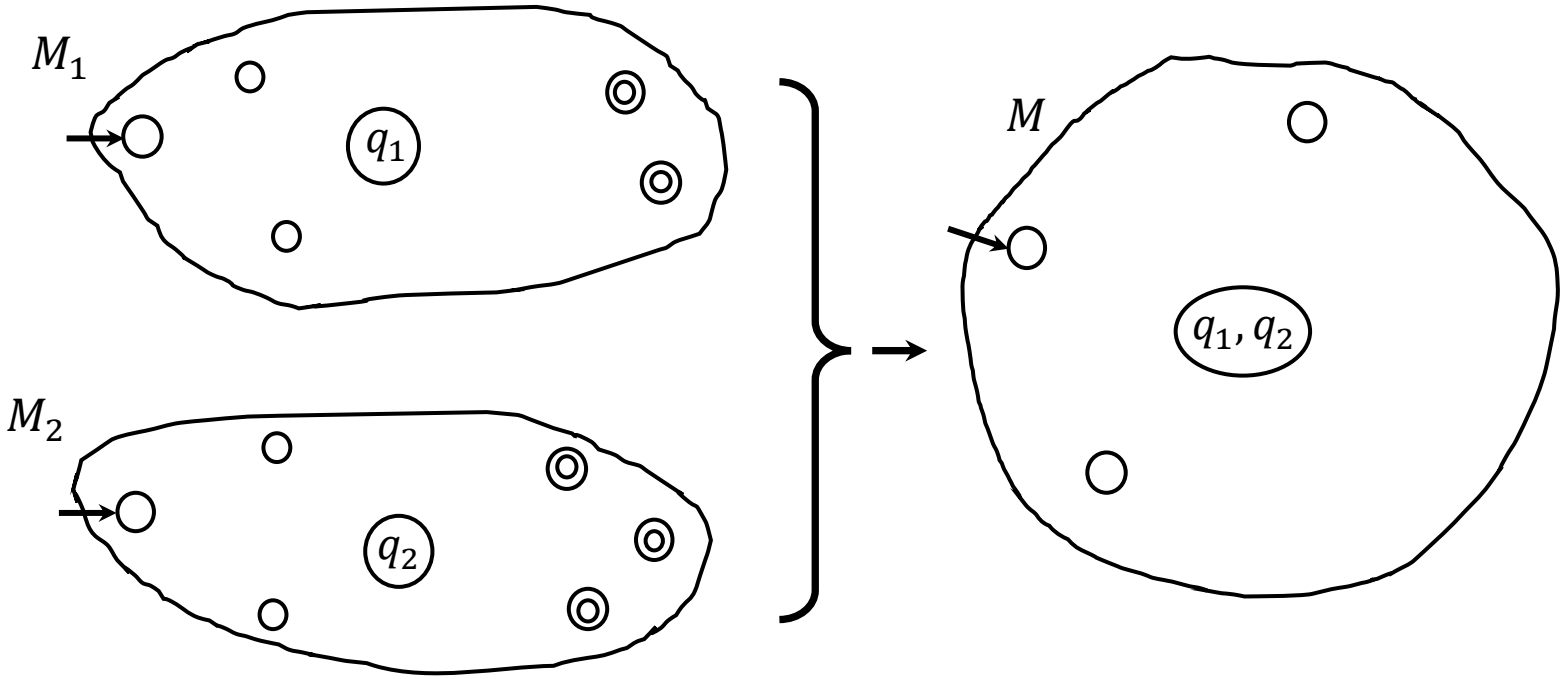
Theorem (closure under ∪)**:** If A_1, A_2 are regular languages, so is $A_1 \cup A_2$

Proof: Let $M_1 = (Q_1, \Sigma, \delta_1, q_1^s, F_1)$ recognize A_1

$M_2 = (Q_2, \Sigma, \delta_2, q_2^s, F_2)$ recognize A_2

Goal: Construct $M = (Q, \Sigma, \delta, q_0, F)$ recognizing $A_1 \cup A_2$

M should accept input w if either M_1 or M_2 accept w.



Components of M:

$Q = Q_1 \times Q_2$

$= \{(q_1, q_2) \mid q_1 \in Q_1 \text{ and } q_2 \in Q_2\}$

$q_0 = (q_1^s, q_2^s)$

$\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$

$F = \cancel{F_1 \times F_2}$ NO! [gives intersection]

$F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$

Closure Properties of Regular Languages

Theorem (closure under ∪)**:** If A_1, A_2 are regular languages, so is $A_1 \cup A_2$

Proof: Let $M_1 = (Q_1, \Sigma, \delta_1, q_1^s, F_1)$ recognize A_1
$M_2 = (Q_2, \Sigma, \delta_2, q_2^s, F_2)$ recognize A_2

Goal: Construct $M = (Q, \Sigma, \delta, q_0, F)$ recognizing $A_1 \cup A_2$
M should accept input w if either M_1 or M_2 accept w.

Mini-quiz:

In the proof, if M_1 has k_1 states and M_2 has k_2 states, then how many states does M have?

(a) $k_1 + k_2$
(b) $(k_1)^2 + (k_2)^2$
(c) $k_1 \times k_2$

Components of M:

$Q = Q_1 \times Q_2$
$= \{(q_1, q_2) \mid q_1 \in Q_1 \text{ and } q_2 \in Q_2\}$

$q_0 = (q_1^s, q_2^s)$

$\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$

$F = \cancel{F_1 \times F_2}$ NO! [gives intersection]

$F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$

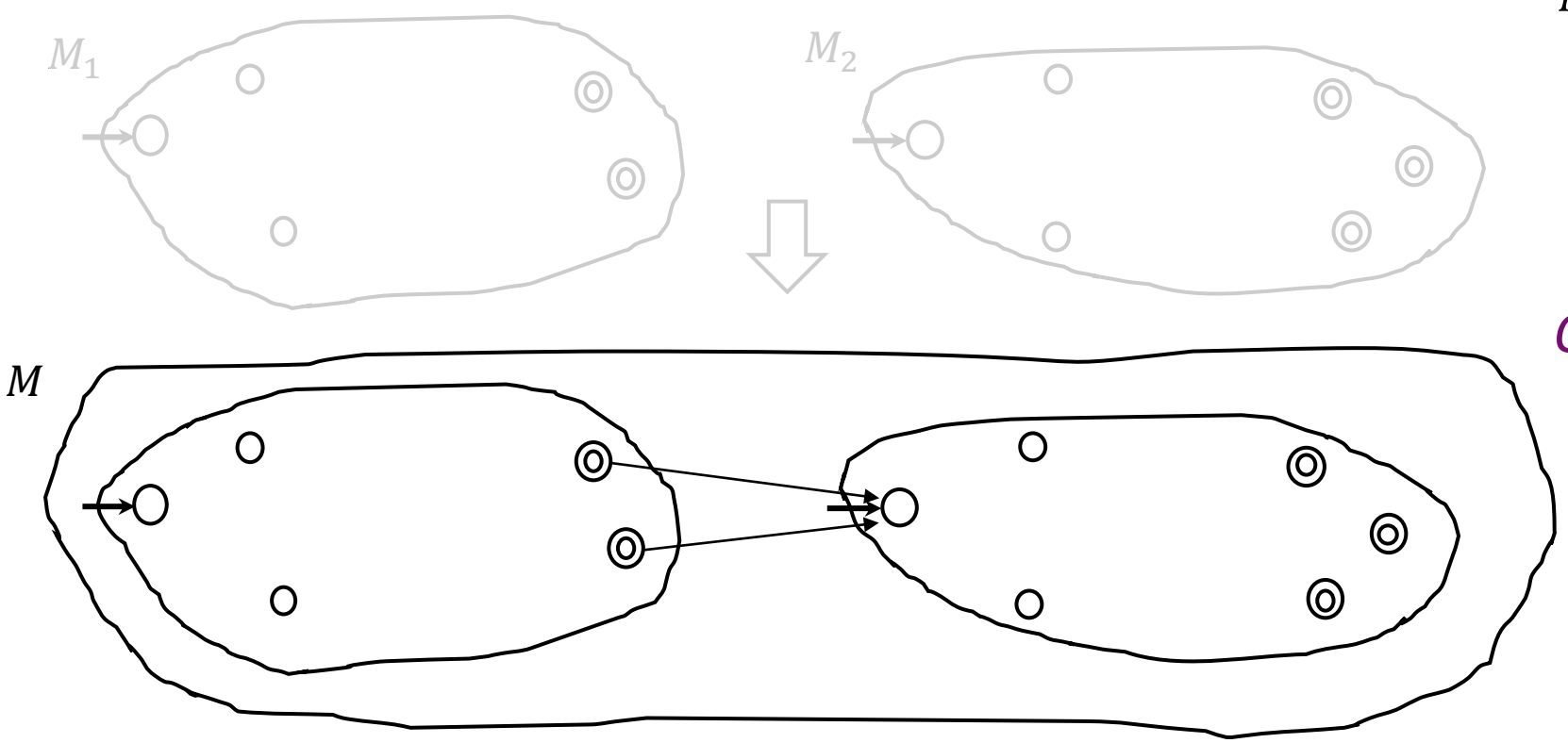
Closure Properties continued

Theorem (closure under ∘): If A_1, A_2 are regular languages, so is $A_1 A_2$

Proof: Let $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1

$M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2

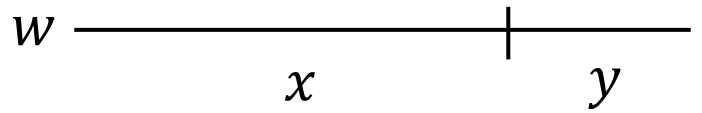
Construct $M = (Q, \Sigma, \delta, q_0, F)$ recognizing $A_1 A_2$



M should accept input w

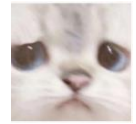
if $w = xy$ where

M_1 accepts x and M_2 accepts y.

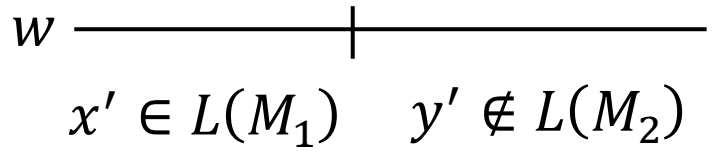


Question: Does this M work?

No!

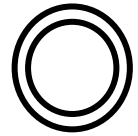


It might split w in a wrong place:



Quick review of today

- Introduction, outline, logistics.
- Finite automata, formal definition, regular languages.
- Regular operations and regular expressions.
- Proved: the class of regular languages is closed under $\cup$
- To prove:
  - Closure under $\circ$
  - Kleene's Theorem: Finite automata = regular expressions.



These slides are adapted from Michael Sipser, *18.404J Theory of Computation*, Fall 2020. Massachusetts Institute of Technology:

- Source: MIT OpenCourseWare https://ocw.mit.edu