

# COMP2022|2922

## Models of Computation

### **Deterministic Finite Automata (DFA)**

Sasha Rubin

August 10, 2022



# Questions we will study (see week 1)

Q. How can we specify languages?

1. In English.
2. In mathematics/set-theoretic notation, and recursive definitions.
3. By regular expressions  $R$
4. By (context-free) grammars  $G$
5. By automata  $M$ , and more generally by machine models of computation.

# Automata in a nutshell

An automaton is like a symbol-processing program that:

- Takes a string as input.
- Can only use variables with finite domains: state taking finitely many values.
- Can read the input string character by character: `get_char()`
- Can test for end of input: `end_of_input()`
- Must decide to "Accept" or "Reject" the input string.

# Why study such a simple model of computation?

1. It is part of computing culture.
  - first appeared in McCulloch and Pitt's model of a neural network (1943)
  - then formalised by Kleene (American mathematician) as a model of stimulus and response
2. It has numerous practical applications.
  - scanner (aka lexical analyser)
  - pattern matching
  - communication protocols with bounded memory
  - circuits with feedback
  - finite-state reactive systems
  - finite-state controllers
  - non-player characters in computer games
  - ...
3. It is simple to implement/code.

# Program representation of automata

What strings over alphabet  $\Sigma = \{a, b\}$  does the following code accept?

---

```
1 state = 1
2 while not end_of_input():
3     x = get_char()
4     if state==1 and x=="b" then state=2
5     else
6         if state==2 and x=="a" then state=3
7 if state==2 return "Accept"
8 else return "Reject"
```

---

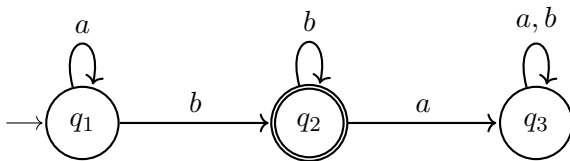
Vote now! (on mentimeter)

1. All strings that match the RE  $(a|b)^*$ .
2. All strings that do contain a  $b$  but not an  $a$ .
3. All strings that match the RE  $a^*bb^*$ .

# Graphical representation of automata

Such programs have a graphical representation as a directed edge-labeled graph:

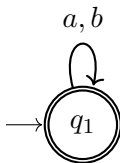
- Vertices represent states.
- Labeled-edges  $q \xrightarrow{a} q'$  represent transitions between states.
- The initial state is marked with an incoming arrow.
- Accepting states are marked with an extra circle.



The automaton **accepts** the strings that label a path from the initial state to a final state.

# Example

Exactly which strings does this automaton accept?

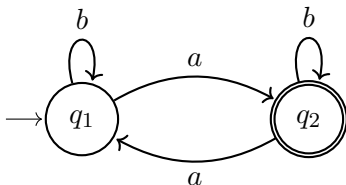


Vote now! (on mentimeter)

1. All strings
2. All strings that end in an  $a$ .
3. All strings with an odd number of  $a$ 's.
4. All strings that do not contain an  $a$ .

# Example

Exactly which strings does this automaton accept?



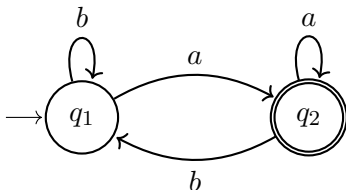
Vote now! (on mentimeter)

1. All strings
2. All strings that end in an  $a$ .
3. All strings with an odd number of  $a$ 's.
4. All strings that do not contain an  $a$ .



# Example

Exactly which strings does this automaton accept?



Vote now! (on mentimeter)

1. All strings
2. All strings that end in an  $a$ .
3. All strings with an odd number of  $a$ 's.
4. All strings that do not contain an  $a$ .

# DFA

## Definition

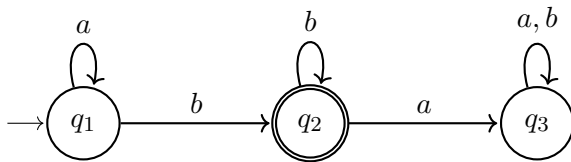
A **deterministic finite automaton (DFA)**  $M$  consists of 5 items

$$(Q, \Sigma, \delta, q_0, F)$$

where

1.  $Q$  is a finite set of **states**,
2.  $\Sigma$  is the **alphabet** (aka **input alphabet**),
3.  $\delta : Q \times \Sigma \rightarrow Q$  is the **transition function**,
  - If  $\delta(q, a) = q'$  we write  $q \xrightarrow{a} q'$ , called a **transition**.
4.  $q_0 \in Q$  is the **start state**, and
5.  $F \subseteq Q$  is the set of **accept states** (aka **final states**).

# Example



1. the states are
2. the alphabet is
3. the transitions are
4. the initial state is
5. the accept states are

# The language recognised by $M$

- The automaton **accepts** the strings that label a path from the initial state to a final state.
- The **language recognised by  $M$**  (aka **the language of  $M$** ) written  $L(M)$  is the set of strings the automaton accepts.

Formally:

## Definition

- A **run** (aka **computation**) of  $M$  on  $w = w_1w_2 \cdots w_n$  is a sequence of transitions  $q_0 \xrightarrow{w_1} q_1 \xrightarrow{w_2} q_2 \xrightarrow{w_3} \cdots \xrightarrow{w_n} q_n$
- The run is **accepting** if  $q_n \in F$ .
- If  $w$  has an accepting run then we say that  $M$  **accepts**  $w$ .
- The set  $L(M) = \{w \in \Sigma^* : M \text{ accepts } w\}$  is the **language recognised by  $M$**  (aka **language of  $M$** ).

# Regular languages

The languages of DFAs are so important, we give them a name:

## Definition

A language  $L$  is called **regular** if  $L = L(M)$  for some DFA  $M$ .

# Designing automata tips (i)

Imagine you are the automaton reading the string symbol by symbol:

- what **information** about the string read so far do you need to make a decision on whether to accept or reject.
- can you update this information if another input symbol arrives?
- the states will store this information.

# Designing automata

Draw an automaton for the language of strings over  $\Sigma = \{a, b\}$  that contain *aab* as a substring.

## Designing automata tips (ii)

Build automata out of other automata using closure properties of the regular languages.

If you have DFA for  $L_1, L_2$  then you can get DFA for

1.  $\Sigma^* \setminus L_1$
2.  $L_1 \cup L_2$  (and thus, by DeMorgan!, also  $L_1 \cap L_2$ )
3.  $L_1 L_2$
4.  $(L_1)^*$

Let's see how to do complement (union is in the tutorial, concatenation and star are trickier and we will do it in a future lecture).



# Regular languages closed under complementation

## Theorem

If  $L$  is regular, then  $\Sigma^* \setminus L$  is regular.

## Idea

Swap accept and reject states in a DFA for  $L$

# Regular languages closed under complementation

## Theorem

If  $L$  is regular, then  $\Sigma^* \setminus L$  is regular.

## Construction: "swap accept and reject states"

- Given DFA  $M = (Q, \Sigma, \delta, q_0, F)$  recognising  $L$
- We build a DFA  $M'$  recognising  $\Sigma^* \setminus L$  as follows:  
Define  $M' = (Q, \Sigma, \delta, q_0, F')$  where  $F' = Q \setminus F$ .

# Regular languages closed under complementation

## Theorem

If  $L$  is regular, then  $\Sigma^* \setminus L$  is regular.

## Example

Give DFA for the language of strings over  $\{a, b\}$  that do **not** contain  $aab$  as a substring.

# Important questions about DFA

- Which languages can be described by DFAs? All languages?
  - No! (we will come back to this in a future lecture)
- There are natural decision problems associated with DFAs. Are there programs that solve them?

## 1. Membership problem

Input: DFA  $M$ , string  $w$ .

Output: decide if  $w \in L(M)$ .

## 2. Non-emptiness problem (tutorial)

Input: DFA  $M$ .

Output: decide if  $L(M) \neq \emptyset$ .

## 3. Equivalence problem (tutorial)

Input: DFAs  $M_1, M_2$ .

Output: decide if  $L(M_1) = L(M_2)$ .

# Membership problem

Input: DFA  $M$ , string  $w$ .

Output: decide if  $w \in L(M)$ .

---

```
1 def membership(M,w):
2     state = q_0
3     while not end_of_input(w):
4         x = get_char(w)
5         state =  $\delta$ (state,x)
6     if state in F:
7         return "Accept"
8     else:
9         return "Reject"
```

---

# COMP2022|2922

## Models of Computation

### **Nondeterministic finite automata (NFA)**

Sasha Rubin

August 10, 2022



THE UNIVERSITY OF  
**SYDNEY**



## Agenda

1. Introduction to nondeterminism.
  - It refers to situations that in which the next state of the computation is not uniquely determined by the current state.
2. Nondeterministic finite automata

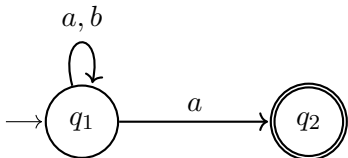
Let's introduce nondeterminism into automata.

- A **nondeterministic finite automaton (NFA)** is like a DFA except that states can have more than one outgoing transition on the same input symbol.
- A string is **accepted by an NFA** if it labels **some path** from the initial state to a final state.



# NFA

Exactly which strings over alphabet  $\Sigma = \{a, b\}$  are accepted by this NFA?



Vote now! (on mentimeter)

The set of

1. all strings.
2. strings that have at least one  $a$ .
3. strings that end in an  $a$ .
4. strings that start with an  $a$ .

# Pattern matching made easy!

- NFAs are good for specifying languages of the form "the string has  $x$  as a substring".
- E.g.,  $x = aab$

# Definition of NFA

## Definition

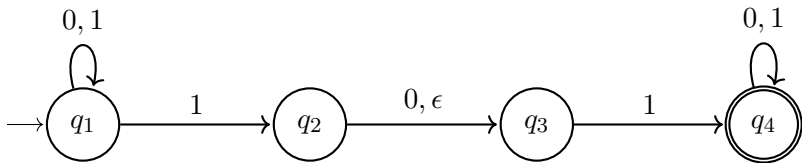
A **nondeterministic finite automaton (NFA)**  $M = (Q, \Sigma, \delta, q_0, F)$  is the same as a DFA except that

$$\delta : Q \times \Sigma_\epsilon \rightarrow P(Q),$$

called the **transition relation**.

- If  $q' \in \delta(q, a)$  we write  $q \xrightarrow{a} q'$ , called a **transition**.
- Here  $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$ .
- So, we also allow **epsilon-transitions**
  - This amounts to transitions that do not consume the next input symbol.

# NFA with Epsilon transitions



# Definition of NFA

## In words:

An NFA  $M$  describes the language  $L(M)$  of all strings that label paths from the start state to an accepting state.

Formally:

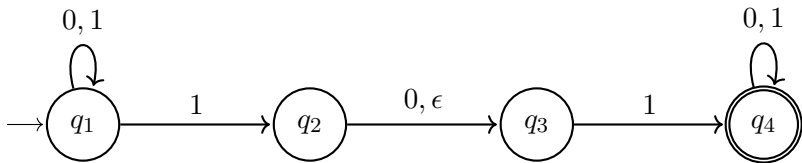
## Definition

- A **run** (aka **computation**) of an NFA  $M$  on string  $w$  is a sequence of transitions  $q_0 \xrightarrow{y_1} q_1 \xrightarrow{y_2} q_2 \dots \xrightarrow{y_m} q_m$  such that each  $y_i \in \Sigma_\epsilon$  and  $w = y_1 y_2 \dots y_m$ .<sup>1</sup>
- The run is **accepting** if  $q_m \in F$ .
- If  $w$  has at least one accepting run, then we say that  $w$  is **accepted** by  $M$ .
- The language **recognised** by  $M$  is  $L(M) = \{w \in \Sigma^* : w \text{ is accepted by } M\}$ .

---

<sup>1</sup>Recall that  $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$ , and  $\epsilon x = x\epsilon = x$  for all strings  $x$ .

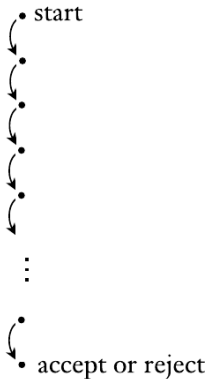
# NFA with Epsilon transitions



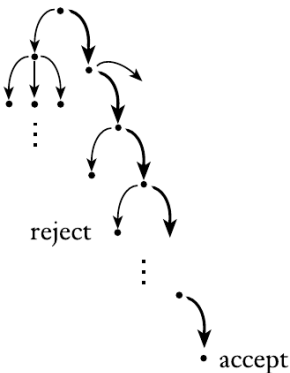
input word 011

# Comparing NFAs and DFAs

Deterministic  
computation



Nondeterministic  
computation



# Comparing NFAs and DFAs

1. In a DFA, every input has exactly one run.<sup>2</sup> The input string is accepted if this run is accepting.
2. In an NFA, an input may have zero, one, or more runs. The input string is accepted if at least one of its runs is accepting.
3. For every DFA  $M$  there is an NFA  $N$  such that  $L(M) = L(N)$ .
  - Idea:  $q' = \delta(q, a)$  in  $M$  becomes  $\{q'\} = \delta(q, a)$  in  $N$ .
  - You can think of a DFA as an NFA in which there is no nondeterminism.

---

<sup>2</sup>With our convention of not drawing rejecting sinks, an input of a DFA may have no runs too.



# Where are we going?

- We are going to show that DFA, NFA and regular expressions specify the same set of languages!
- We will do this with a series of transformations:
  1. From Regular Expressions to NFAs
  2. From NFAs to NFAs without  $\epsilon$ -transitions
  3. From NFAs without  $\epsilon$ -transitions to DFAs
  4. From DFAs to Regular Expressions

# From Regular Expressions to NFAs

## Theorem

*For every RE  $R$  there is an NFA  $N$  such that  $L(R) = L(N)$ .*

The construction is by recursion on the structure of  $R$ .

1. The base cases are  $R = \emptyset$ ,  $R = \epsilon$ ,  $R = a$  for  $a \in \Sigma$ 
  - We must show that each of these languages is recognised by some NFA.
2. The recursive cases are  $R = (R_1 \mid R_2)$ ,  $R = (R_1 R_2)$ , and  $R = R_1^*$ 
  - We must show that if  $N_1, N_2$  are NFAs, then there are NFAs recognising  $L(N_1 \cup N_2)$ ,  $L(N_1)L(N_2)$ , and  $L(N_1)^*$ .

# From Regular Expressions to NFAs

The base cases are  $R = \emptyset$ ,  $R = \epsilon$ ,  $R = a$  for  $a \in \Sigma$ .

# NFAs are closed under union

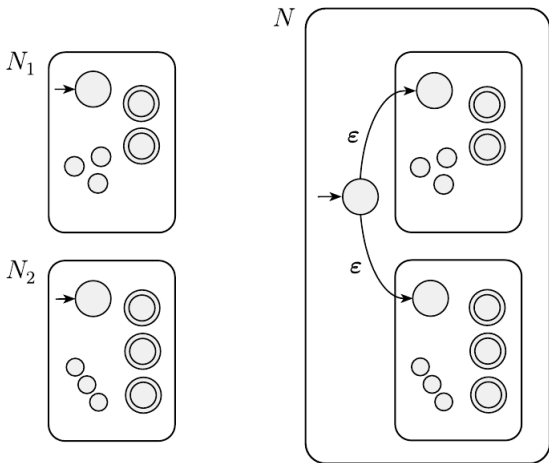
## Lemma

*If  $N_1, N_2$  are NFAs, there is an NFA  $N$  recognising  $L(N_1) \cup L(N_2)$*

## Idea: "Simulate $N_1$ or $N_2$ "

- Given NFAs  $N_i = (Q_i, \Sigma, \delta_i, q_i, F_i)$  construct NFA  $N$  that guesses which of  $N_1$  or  $N_2$  to simulate. How?
- $N$  has states  $Q_1 \cup Q_2 \cup \{q_0\}$  so that it can simulate  $N_1, N_2$ .
- $N$  guesses from  $q_0$  whether to go to the start state of  $N_1$  or  $N_2$ .

# NFAs are closed under union



# NFAs are closed under concatenation

## Lemma

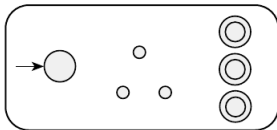
*If  $N_1, N_2$  are NFAs, there is an NFA  $N$  recognising  $L(N_1)L(N_2)$*

**Idea: "Simulate  $N_1$  followed by  $N_2$ "**

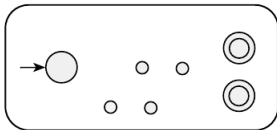
- Given NFAs  $N_i = (Q_i, \Sigma, \delta_i, q_i, F_i)$  construct NFA  $N$  that **guesses** how to break the input into two pieces, the first accepted by  $N_1$ , the second by  $N_2$ . How?
- $N$  has states  $Q_1 \cup Q_2$  so that it can simulate  $N_1, N_2$ .
- At some point when  $N_1$  is in a final state, **guess** that it is time to move to the initial state of  $N_2$ .

# NFAs are closed under concatenation

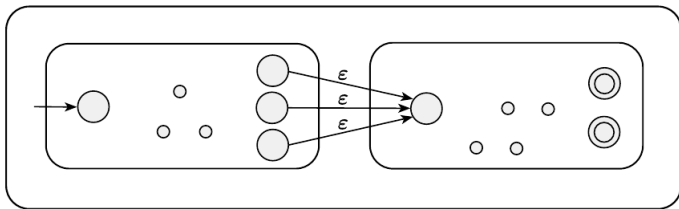
$N_1$



$N_2$



$N$



# NFAs are closed under star

## Lemma

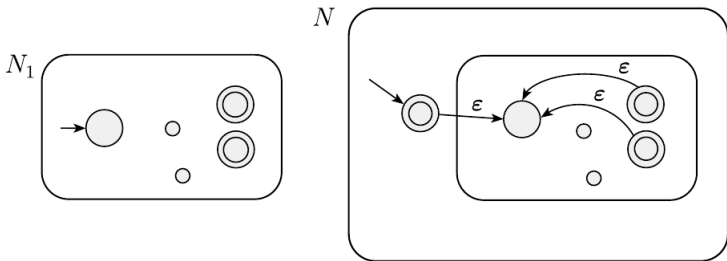
*If  $N_1$  is an NFA, there is an NFA  $N$  recognising  $L(N_1)^*$*

## Idea: "Repeatedly simulate $N_1$ "

- Given NFA  $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$  construct NFA  $N$  that guesses how to break the input into pieces, each of which is accepted by  $N_1$ . How?
- $N$  has states  $Q \cup \{q_0\}$ , and extra transitions from final states of  $N_1$  to the initial state of  $N_1$ .
- The new state  $q_0$  is ensure that  $\epsilon$  is accepted.



# NFAs are closed under star



# From RE to NFA: example

Convert the regular expression  $(ab|a)^*$  to an NFA.

# Where are we going?

- We are going to show that DFA, NFA and regular expressions specify the same set of languages!
- We will do this with a series of transformations:
  1. From Regular Expressions to NFAs
  2. From NFAs to NFAs without  $\epsilon$ -transitions
  3. From NFAs without  $\epsilon$ -transitions to DFAs
  4. From DFAs to Regular Expressions