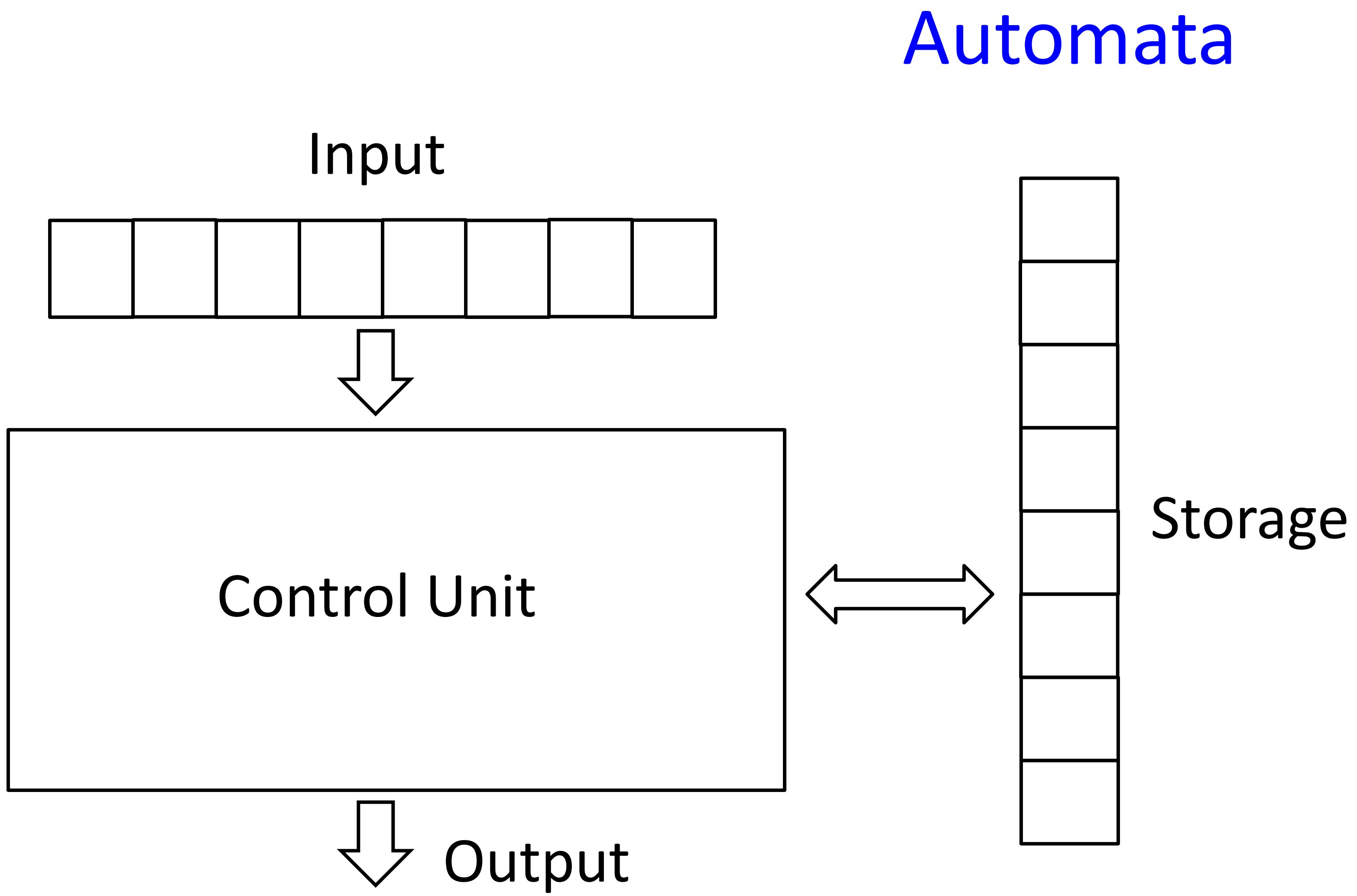
Finite Automata

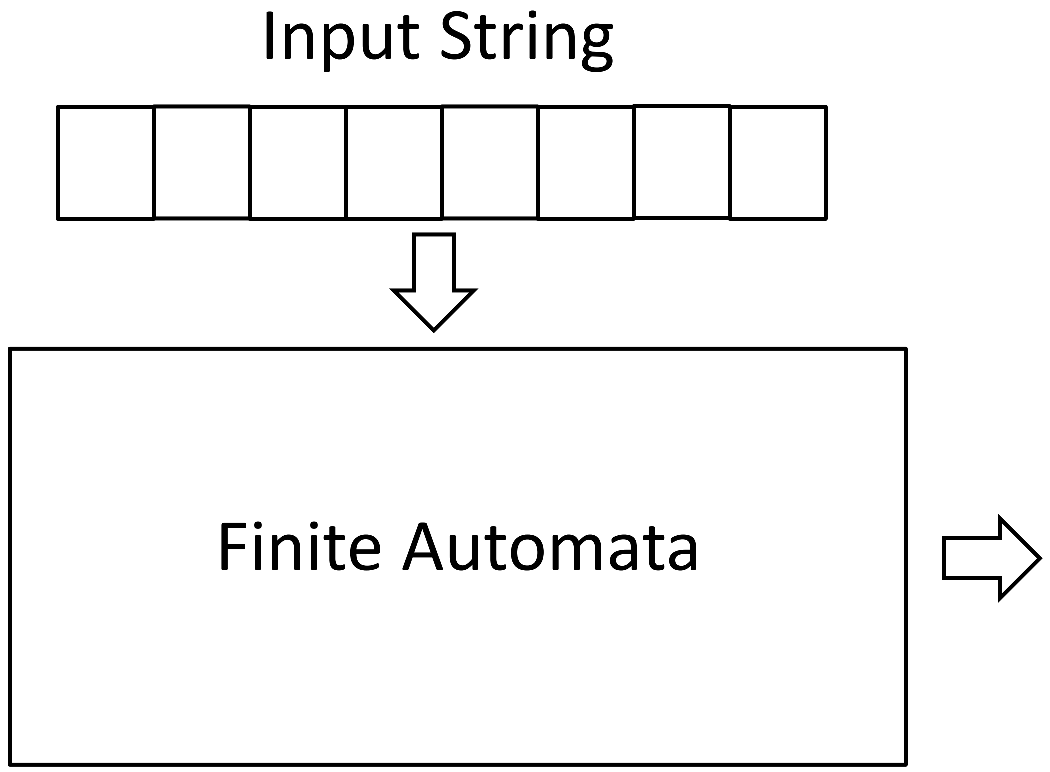
Theory of Computation

CISC 603, Spring 2020, Daqing Yun

Automata Theory



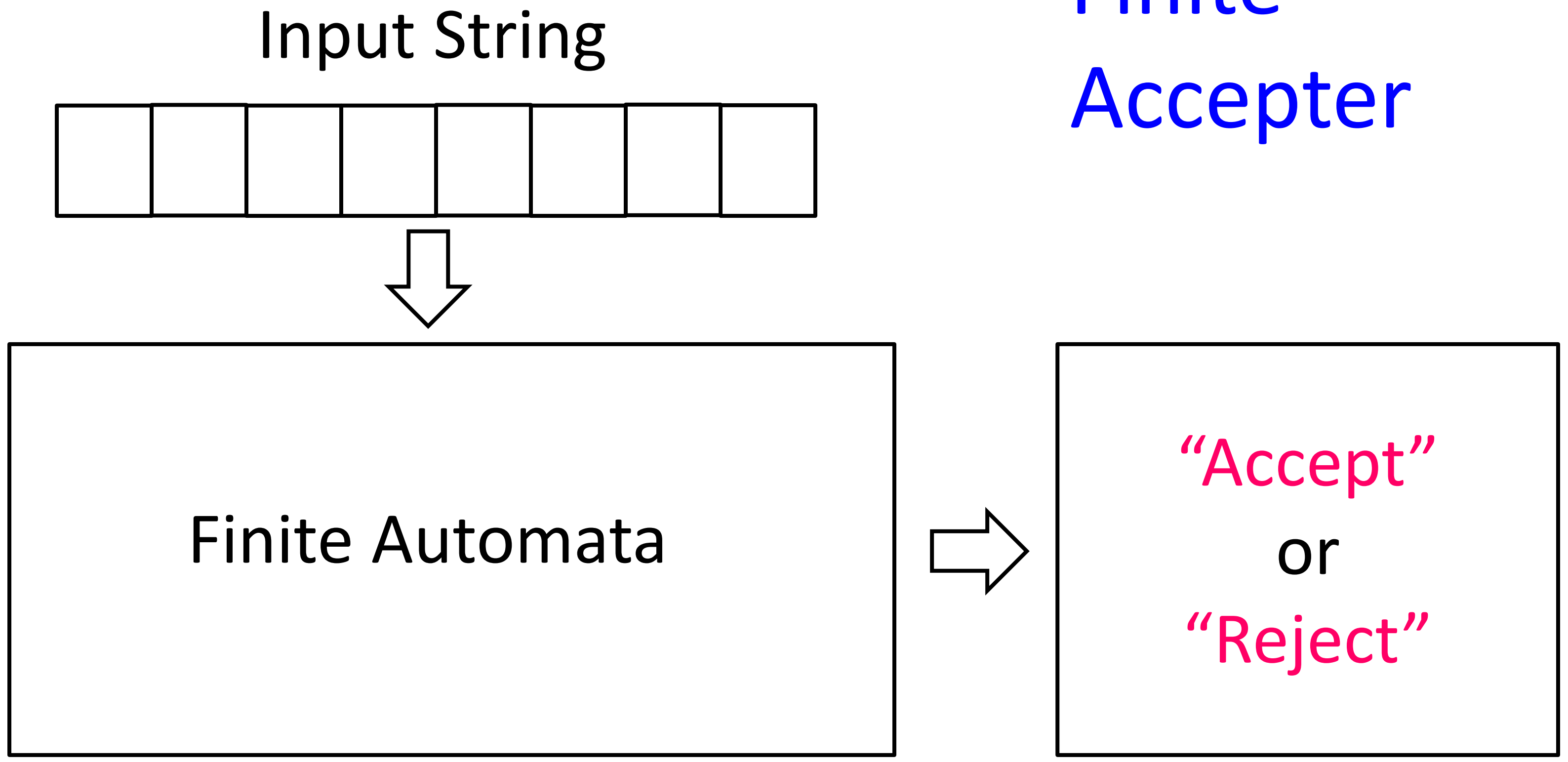
Automata Theory



Finite Automata

Automata Theory

Finite
Accepter



The Simplest Computers

- Computers are now everywhere
- How well do we understand the way they work?
- Real computers have a lot of complexity
  - Many subsystems and their interactions
  - Impractical to analyze real computers directly
  - Use simplified models of computers
- A computing machine to its barest essentials
  - What it can be used for
  - Explore the limits

Finite Automata

- Finite Automata (FA) are ideally suited for lexical analysis, the first stage in compiling a computer program
- A lexical analyzer takes a string of characters and provides a string of "tokens"
- Tokens have a simple structure: e.g., "41.3", "main"

#	i	n	c	l	u	d	e	<sp>	<	s	t	d	i	o	.
35	105	110	99	108	117	100	101	32	60	115	116	100	105	111	46
h	>	\n	\n	i	n	t	<sp>	m	a	i	n	(	)	\n	{
104	62	10	10	105	110	116	32	109	97	105	110	40	41	10	123
\n	<sp>	<sp>	<sp>	<sp>	p	r	i	n	t	f	(	"	h	e	l
10	32	32	32	32	112	114	105	110	116	102	40	34	104	101	108
l	o	,	<sp>	w	o	r	l	d	\	n	"	)	;	\n	}
108	111	44	32	119	111	114	108	100	92	110	34	41	59	10	125

```
#include <stdio.h>

int main()
{
    printf("hello, world\n");
}
```

Deterministic Finite Automata States, Rules, and Input

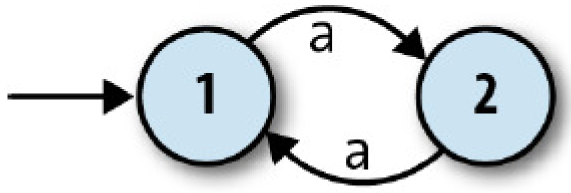
- A *finite state machine* (FSM), also known as a *finite automaton*, is a drastically simplified model of a computer that throws out all of these features in exchange for being easy to understand, easy to reason about, and easy to implement in hardware or software
- Conceived as an *abstract* machine that can be in one of a *finite* number of states
- FSM can change from one state to another when initiated by a triggering event or condition, this is called *transition*

Deterministic Finite Automata States, Rules, and Input

- Finite automata have no permanent storage and virtually no RAM
- A machine with a handful of possible *states* and the ability to keep track of which one of those states it is currently in
- They do not have a keyboard, mouse, or network interface for receiving input, just a single external stream of input characters that they can read one at a time

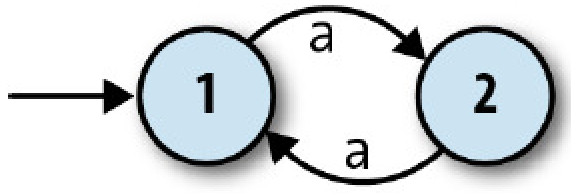
Deterministic Finite Automata States, Rules, and Input

- Each finite automaton has a *hardcoded* collection of *rules* that determine how it should move from one state to another in response to input
- The automaton starts in one particular state and reads individual characters from its input stream, following a rule each time it reads a character



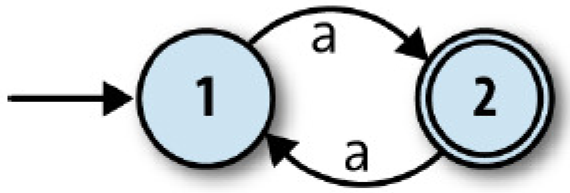
Deterministic Finite Automata States, Rules, and Input

- How the machine processes a stream of inputs:
  - It starts in state 1
  - It only has rules for reading the character a from its input stream
  - The only thing that can happen, when it reads an a, it moves from state 1 into state 2
  - When the machine reads another a, it moves back into state 1



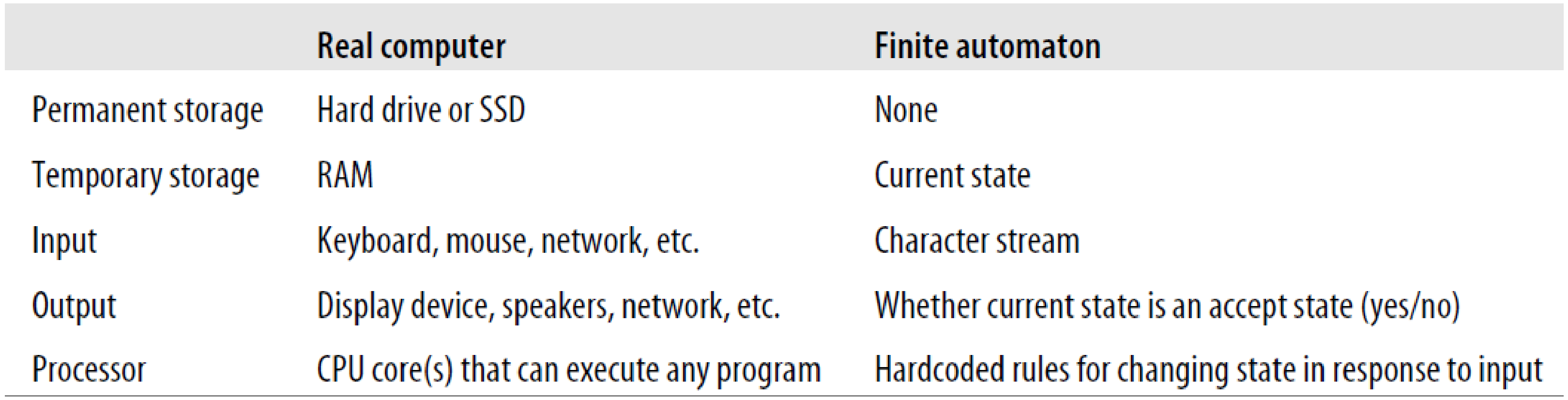
Deterministic Finite Automata States, Rules, and Input

- Output: just mark some of the states as being special (call them *accept* states), say state 2, and show that with a double circle
- This machine accepts any string of a's whose length is an odd number: a, aaa, aaaaa are all accepted, while aa, aaaa, and the *empty* string are rejected



Deterministic Finite Automata States, Rules, and Input

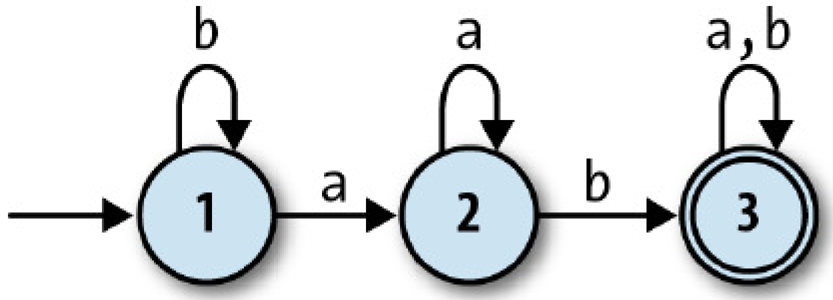
- A machine can read a sequence of characters and provide a yes/no output to indicate whether that sequence has been accepted
- The DFA is performing a *computation*, because we can ask it a question—"is the length of this string an odd number?"—and get a meaningful reply back
- This is arguably enough to call it a simple computer, and we can see how its features stack up against a real computer:



	Real computer	Finite automaton
Permanent storage	Hard drive or SSD	None
Temporary storage	RAM	Current state
Input	Keyboard, mouse, network, etc.	Character stream
Output	Display device, speakers, network, etc.	Whether current state is an accept state (yes/no)
Processor	CPU core(s) that can execute any program	Hardcoded rules for changing state in response to input

Deterministic Finite Automata States, Rules, and Input

- This specific automaton does not do anything sophisticated or useful, and we can build more complex automata that have more states and can read more than one character
- For example, this is one with three states and the ability to read the inputs a and b, and it accepts strings like ab, baba, and aaaab, and rejects strings like a, baa, and bbbba



Determinism

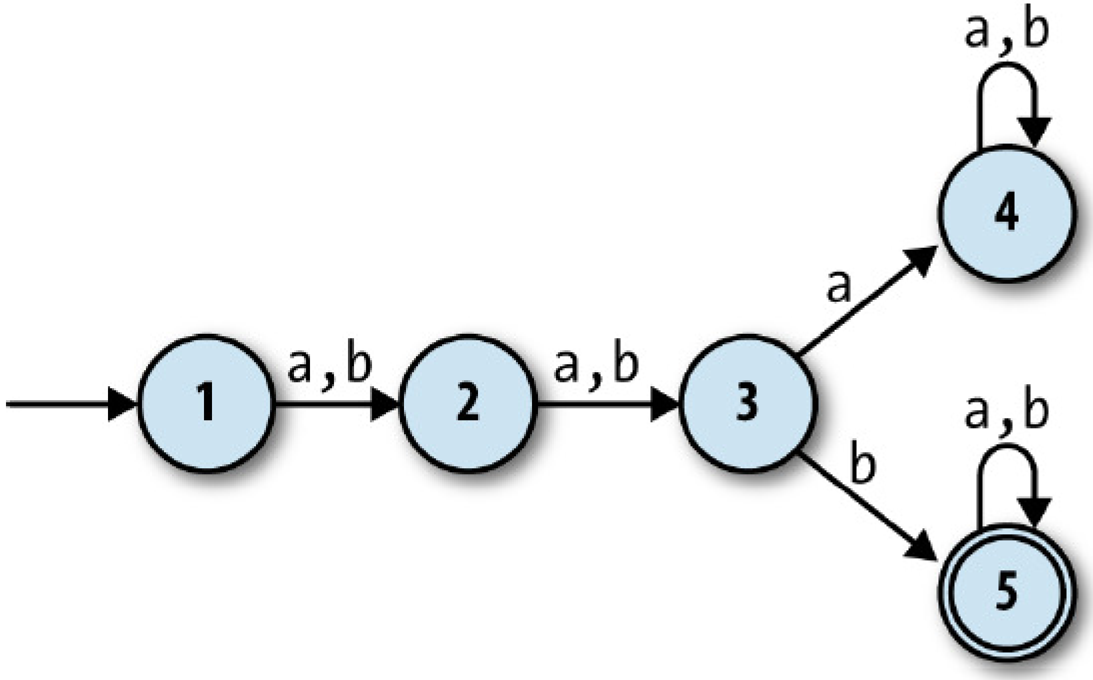
- This kind of automaton is *deterministic*: it is always absolutely certain which state it will end up in
- It is guaranteed if we respect two constraints:
  - No contradictions
  - No omissions
- The machine must have exactly one rule for each combination of state and input
- The technical name for a machine that obeys the determinism constraints is a *deterministic finite automaton* (DFA)

Nondeterministic Finite Automata

- Determinism constraints seem restrictive
- Leave out rules for characters we do not care
- Let the machine can go into a generic failure state when something unexpected happens
- Allow the machine to have contradictory rules, so that more than one execution path is possible?
- What if the machine could change state without having to read anything?

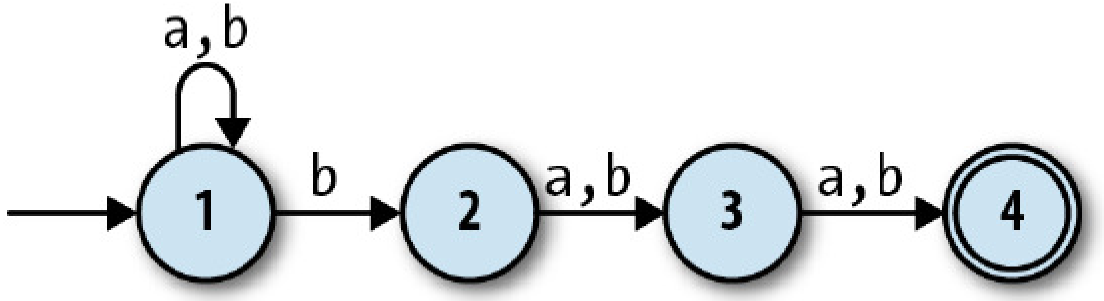
Nondeterminism

- A finite automaton that would accept any string of a’s and b’s as long as the third character was b



Nondeterminism

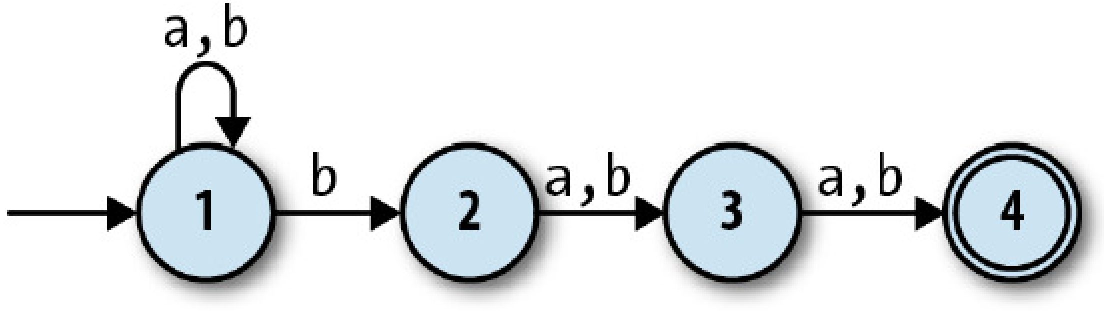
- Relax the determinism constraints
- Allow the rulebook to contain multiple rules (or no rules at all) for a given state and input, we can design a machine that does the job:



- This state machine is a *nondeterministic finite automata* (NFA)

Nondeterministic Finite Automata

- What does it mean for an NFA to accept or reject a string?
- A string is accepted if there is *some* way for the NFA to end up in an accept state by following some of its rules—that is, if finishing in an accept state is *possible*, even if it is not inevitable

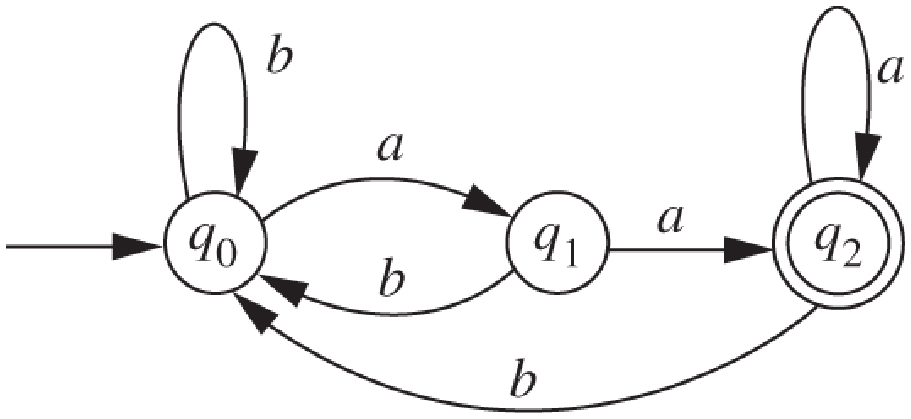


Languages

- The collection of strings that are accepted by a particular machine is called a *language*: we say that the machine *recognizes* that language
- Not all possible languages have a DFA or NFA that can recognize them, but those languages that can be recognized by finite automata are called *regular languages*

Example

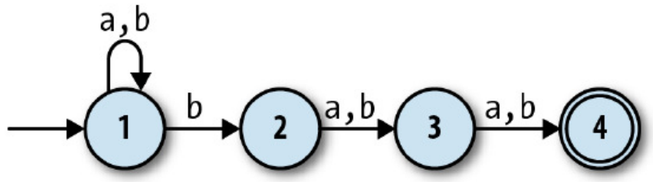
- This FA accepts the language of strings that end in aa
  - The three states represent strings that end with no a’s, one a, and two a’s, respectively
  - From each state, if the input is anything but an a, go back to the initial state, because now the current string does not end with a



$$\begin{tikzcd} \arrow[r] & q_0 \arrow[loop above, "b"] \arrow[r, bend left, "a"] & q_1 \arrow[l, bend left, "b"] \arrow[r, "a"] & q_2 \arrow[loop above, "a"] \arrow[ll, bend left=50, "b"] \end{tikzcd}$$

Nondeterministic Finite Automata

- An NFA deals in possibilities rather than certainties; its behavior is in terms of what *can* happen rather than what *will* happen
- How can such a machine work in the real world?
- To stand a chance of accepting a string, our example NFA must stay in state 1 until it reads the third-from-last character
- It has no way of knowing how many more characters it will receive
- How can we simulate such a machine like this deterministically?



Nondeterministic Finite Automata

- The key to simulating an NFA on a deterministic computer is to find a way to *explore all possible executions* of the machine
- When an NFA reads a character, there are only ever a *finite* number of possibilities for what it can do next
- Simulate the nondeterminism by somehow trying all possibilities and seeing whether any of them ultimately allows it to reach an accept state

Nondeterministic Finite Automata

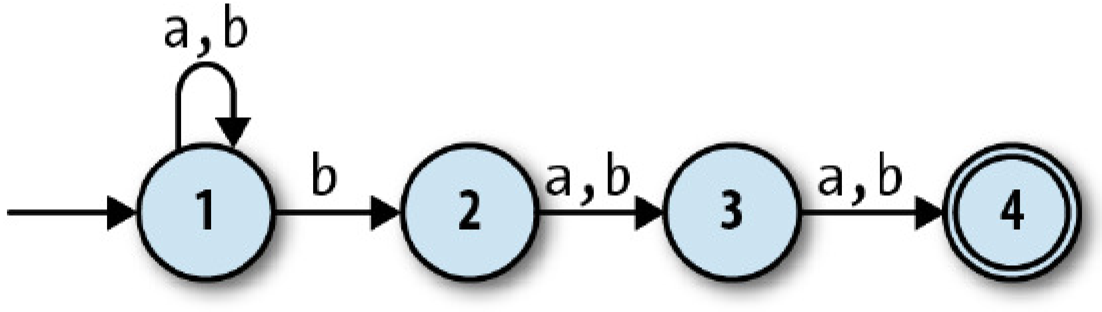
- Recursively trying all possibilities
  1. Each time the simulated NFA reads a character and there are multiple applicable rules, follow one of those rules and try reading the rest of the input
  2. If that does not leave the machine in an accept state, go back into the earlier state, rewind the input to its earlier position, and try again by following a different rule
  3. Repeat until some choice of rules leads to an accept state, or until all possible choices have been tried without success
- This approach is complicated and highly inefficient

Nondeterministic Finite Automata

- Simulate a single NFA by keeping track of all its *possible* current states
- Two identical machines in the same state are interchangeable for any purpose
- Nothing is lost if collapsing all those possibilities down into a single machine and asking “which states *could* it be in by now?” instead

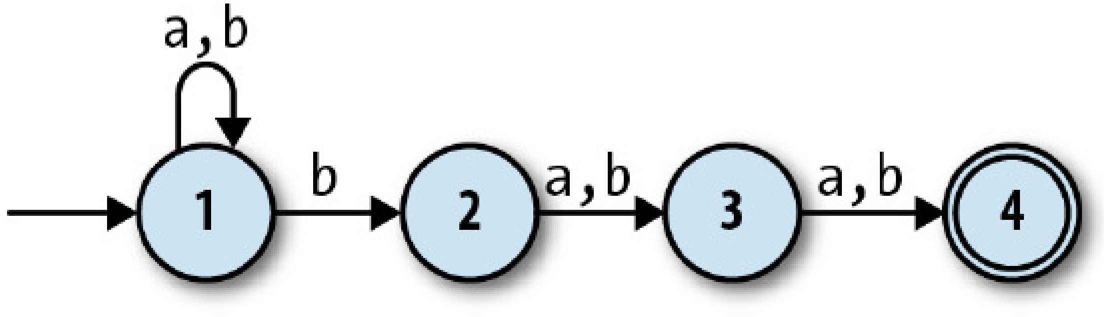
Example Walk Through – "bab"

- Before the NFA has read any input, it is definitely in state 1, its start state



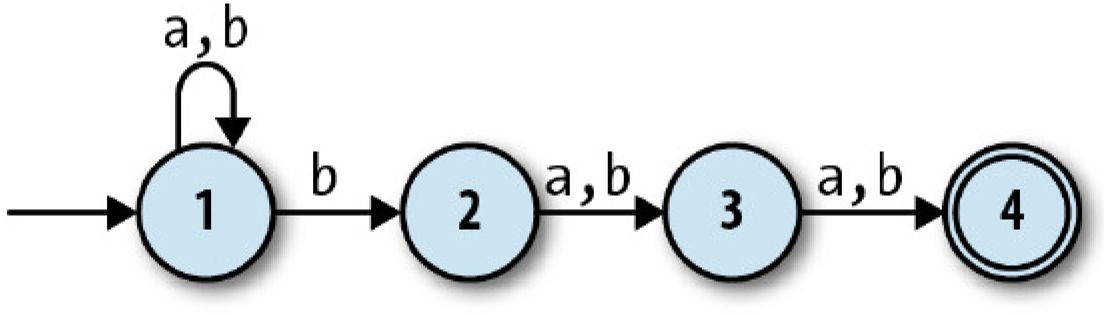
Example Walk Through – “bab”

- It reads the first character, b. From state 1, there is one b rule that lets the NFA stay in state 1 and another b rule that takes it to state 2, so we know it can be in either state 1 or 2 afterward
- Neither of those is an accept state, which tells us there is no possible way for the NFA to reach an accept state by reading the string b



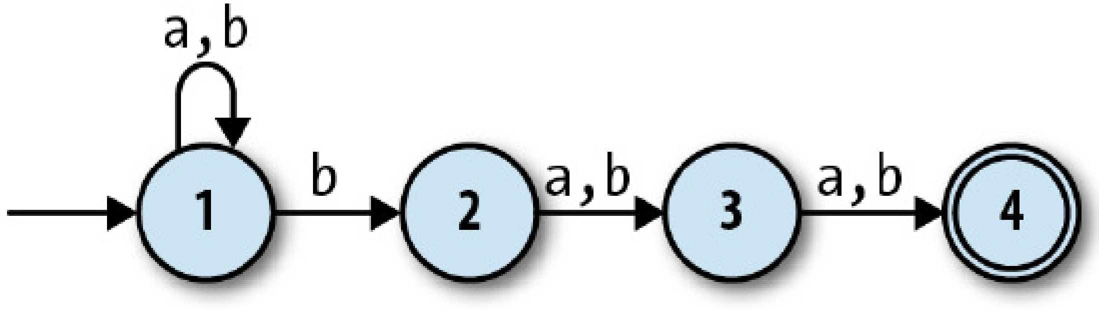
Example Walk Through – “bab”

- It reads the second character, a. If it is in state 1 then there is only one a rule it can follow, which will keep it in state 1
- If it is in state 2, it will have to follow the a rule that leads to state 3
- It must end up in state 1 or 3, and again, these are not accept states, so there is no way the string ba can be accepted by this machine



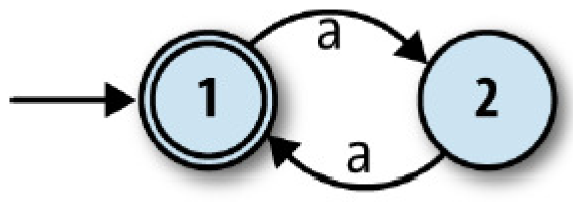
Example Walk Through – "bab"

- It reads the third character, b. If it is in state 1 then, as before it can stay in state 1 or go to state 2; if it is in state 3, then it must go to state 4
- It is possible for the NFA to be in states 1, 2, or 4 after reading the whole input string
- State 4 is an accept state, and our simulation shows that there must be *some* way for the machine to reach state 4 by reading that string, so the NFA *does* accept bab



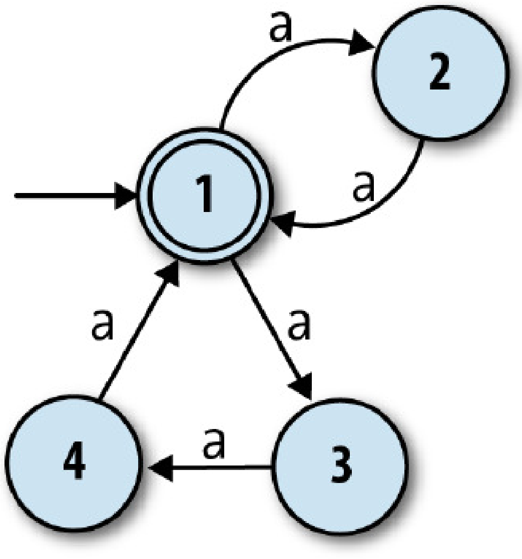
Free Moves

- What else can we safely relax to give ourselves more design freedom?
- Design a DFA that accepts strings of a’s whose length is a multiple of two (aa, aaaa, …):



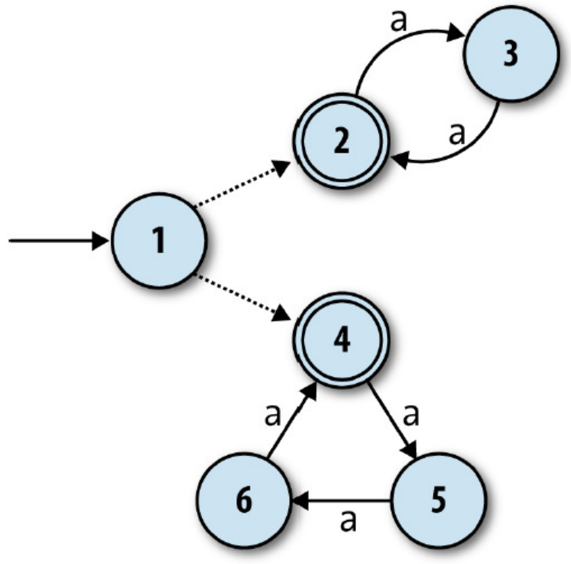
Free Moves

- How can we design a machine that accepts strings whose length is a multiple of *two* or *three*?
- Design an NFA that has one "multiple of two" path and one "multiple of three" path



Free Moves

- Not obvious whether an NFA can do the job at all
- Let us try something called *free moves*
- Machine can spontaneously follow without reading any input
- Now the NFA has an initial choice between two separate groups of states:



Terminology

- The characters read by finite automata are usually called *symbols*
- The rules for moving between states are called *transitions*
- The collection of rules making up a machine is called a *transition function* (or sometimes *transition relation* for NFAs)
- The math symbol for the empty string is the Greek letter *ε* (epsilon), an NFA with free moves is known as an NFA-*ε*, and free moves themselves are usually called *ε-transitions*

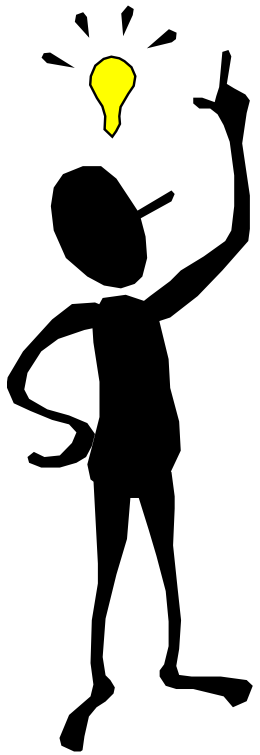
Finite Automata

- Definition: a *finite automaton* is a 5-tuple $M = (Q, \Sigma, q_0, A, \delta)$, where:
  - Q is a finite set of *states*
  - Σ is a finite *input alphabet*
  - $q_0 \in Q$ is the *initial state*
  - $A \subseteq Q$ is the set of *accepting* states
  - $\delta : Q \times \Sigma \rightarrow Q$ is the *transition* function
- From state q the machine will move to state $\delta(q, \sigma)$ if it receives input symbol σ

Next

- Regular Expressions

RegEx



Thanks ! ☺

Questions ?