Design and Analysis of Algorithms

Presented by Dr. Li Ning

Shenzhen Institutes of Advanced Technology, Chinese Academy of Science
Shenzhen, China

Turing Machines

Finite State Machines

Automatic Machines

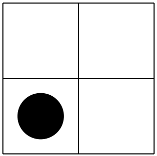
States: the static configurations of a system

Actions: the input symbols

Transitions: the mappings from (state, action) to state

Automatic machine, also called **automata**: the next state of the system is totally determined by the current state and the transition to perform.

state: bottom-left



Automatic Machines

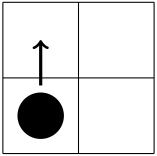
States: the static configurations of a system

Actions: the input symbols

Transitions: the mappings from (state, action) to state

Automatic machine, also called **automata**: the next state of the system is totally determined by the current state and the transition to perform.

action: move up



Automatic Machines

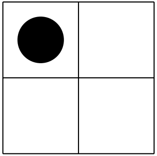
States: the static configurations of a system

Actions: the input symbols

Transitions: the mappings from (state, action) to state

Automatic machine, also called **automata**: the next state of the system is totally determined by the current state and the transition to perform.

state: top-left



Automatic Machines

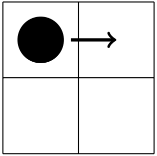
States: the static configurations of a system

Actions: the input symbols

Transitions: the mappings from (state, action) to state

Automatic machine, also called **automata**: the next state of the system is totally determined by the current state and the transition to perform.

action: move right



Automatic Machines

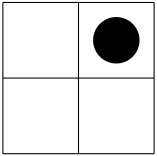
States: the static configurations of a system

Actions: the input symbols

Transitions: the mappings from (state, action) to state

Automatic machine, also called **automata**: the next state of the system is totally determined by the current state and the transition to perform.

state: top-right



Automatic Machines

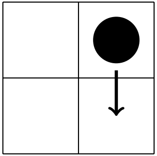
States: the static configurations of a system

Actions: the input symbols

Transitions: the mappings from (state, action) to state

Automatic machine, also called **automata**: the next state of the system is totally determined by the current state and the transition to perform.

action: move down



Automatic Machines

States: the static configurations of a system

Actions: the input symbols

Transitions: the mappings from (state, action) to state

Automatic machine, also called **automata**: the next state of the system is totally determined by the current state and the transition to perform.

state: bottom-right

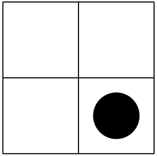
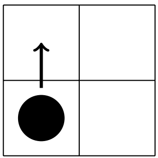


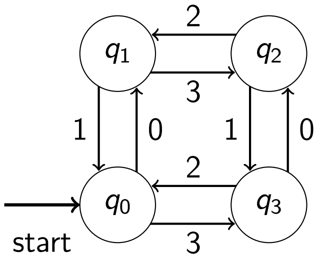
Diagram of Automatic Machines

q_0	bottom-left	q_1	top-left
q_2	top-right	q_3	bottom-right

0	up	1	down
2	left	3	right

bottom-left; move up



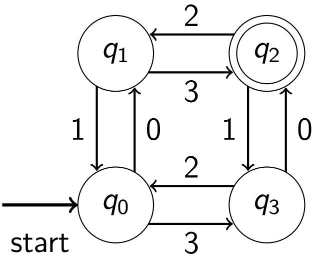


Finite State Machines

Finite State Machine, also called Finite Automata

- **states**
  - **start** state
  - **accepting** state
- **transitions**

- at a time, the machine is in **one** state
- handle the input symbols one by one
- **decide** if the machine stops at the accepting state



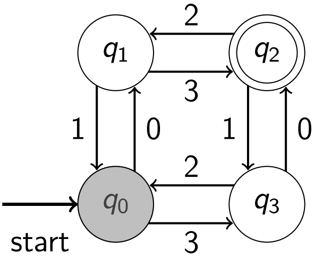
Finite State Machines

q_0	bottom-left	q_1	top-left
q_2	top-right	q_3	bottom-right

0	up	1	down
2	left	3	right

- at a time, the machine is in **one** state
- handle the input symbols one by one
- **decide** if the machine stops at the accepting state

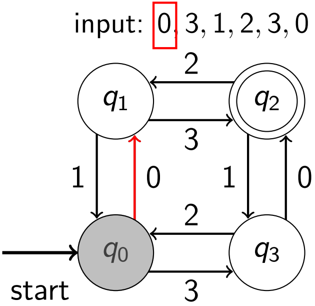
input: $0, 3, 1, 2, 3, 0$



Finite State Machines

q_0	bottom-left	q_1	top-left		0	up	1	down
q_2	top-right	q_3	bottom-right		2	left	3	right

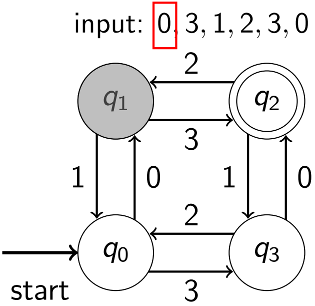
- at a time, the machine is in **one** state
- handle the input symbols one by one
- **decide** if the machine stops at the accepting state



Finite State Machines

q_0	bottom-left	q_1	top-left		0	up	1	down
q_2	top-right	q_3	bottom-right		2	left	3	right

- at a time, the machine is in **one** state
- handle the input symbols one by one
- **decide** if the machine stops at the accepting state



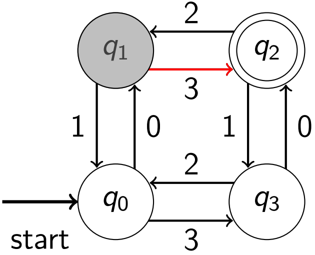
Finite State Machines

q_0	bottom-left	q_1	top-left
q_2	top-right	q_3	bottom-right

0	up	1	down
2	left	3	right

- at a time, the machine is in **one** state
- handle the input symbols one by one
- **decide** if the machine stops at the accepting state

input: 0, $\boxed{3}$, 1, 2, 3, 0



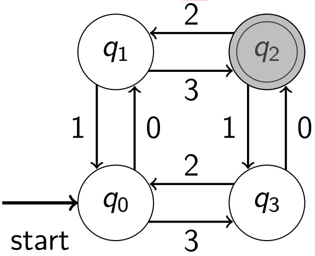
Finite State Machines

q_0	bottom-left	q_1	top-left
q_2	top-right	q_3	bottom-right

0	up	1	down
2	left	3	right

- at a time, the machine is in **one** state
- handle the input symbols one by one
- **decide** if the machine stops at the accepting state

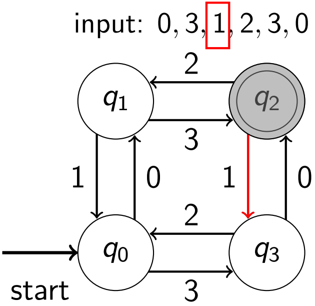
input: 0, $\boxed{3}$, 1, 2, 3, 0



Finite State Machines

q_0	bottom-left	q_1	top-left		0	up	1	down
q_2	top-right	q_3	bottom-right		2	left	3	right

- at a time, the machine is in **one** state
- handle the input symbols one by one
- **decide** if the machine stops at the accepting state



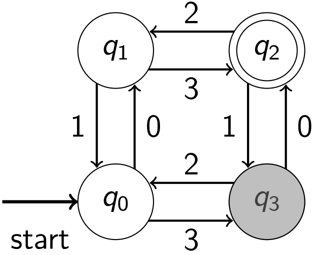
Finite State Machines

q_0	bottom-left	q_1	top-left
q_2	top-right	q_3	bottom-right

0	up	1	down
2	left	3	right

- at a time, the machine is in **one** state
- handle the input symbols one by one
- **decide** if the machine stops at the accepting state

input: $0, 3, \boxed{1}, 2, 3, 0$

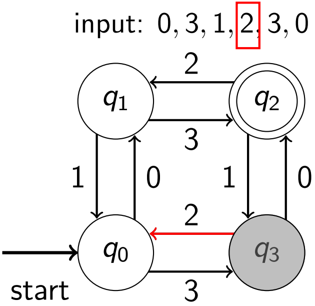


Finite State Machines

q_0	bottom-left	q_1	top-left
q_2	top-right	q_3	bottom-right

0	up	1	down
2	left	3	right

- at a time, the machine is in **one** state
- handle the input symbols one by one
- **decide** if the machine stops at the accepting state



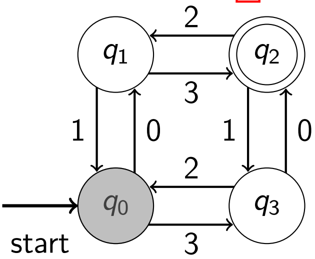
Finite State Machines

q_0	bottom-left	q_1	top-left
q_2	top-right	q_3	bottom-right

0	up	1	down
2	left	3	right

- at a time, the machine is in **one** state
- handle the input symbols one by one
- **decide** if the machine stops at the accepting state

input: $0, 3, 1, \boxed{2}, 3, 0$

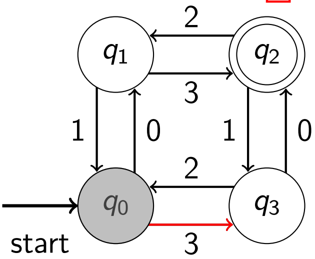


Finite State Machines

q_0	bottom-left	q_1	top-left	0	up	1	down
q_2	top-right	q_3	bottom-right	2	left	3	right

- at a time, the machine is in **one** state
- handle the input symbols one by one
- **decide** if the machine stops at the accepting state

input: 0, 3, 1, 2, $\boxed{3,}$ 0



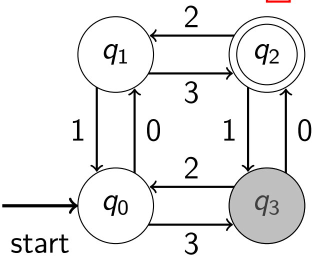
Finite State Machines

q_0	bottom-left	q_1	top-left
q_2	top-right	q_3	bottom-right

0	up	1	down
2	left	3	right

- at a time, the machine is in **one** state
- handle the input symbols one by one
- **decide** if the machine stops at the accepting state

input: 0, 3, 1, 2, $\boxed{3,}$ 0



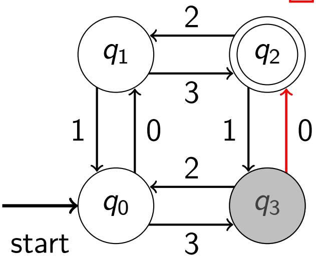
Finite State Machines

q_0	bottom-left	q_1	top-left
q_2	top-right	q_3	bottom-right

0	up	1	down
2	left	3	right

- at a time, the machine is in **one** state
- handle the input symbols one by one
- **decide** if the machine stops at the accepting state

input: 0, 3, 1, 2, 3, $\boxed{0}$



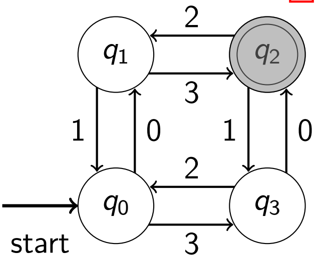
Finite State Machines

q_0	bottom-left	q_1	top-left
q_2	top-right	q_3	bottom-right

0	up	1	down
2	left	3	right

- at a time, the machine is in **one** state
- handle the input symbols one by one
- **decide** if the machine stops at the accepting state

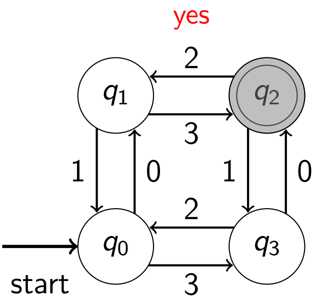
input: $0, 3, 1, 2, 3, \boxed{0}$



Finite State Machines

q_0	bottom-left	q_1	top-left	0	up	1	down
q_2	top-right	q_3	bottom-right	2	left	3	right

- at a time, the machine is in **one** state
- handle the input symbols one by one
- **decide** if the machine stops at the accepting state



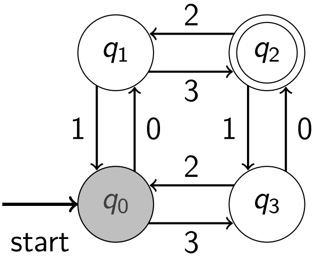
Finite State Machines

q_0	bottom-left	q_1	top-left
q_2	top-right	q_3	bottom-right

0	up	1	down
2	left	3	right

- at a time, the machine is in **one** state
- handle the input symbols one by one
- **decide** if the machine stops at the accepting state

input: $3, 0, 2, 1, 0$

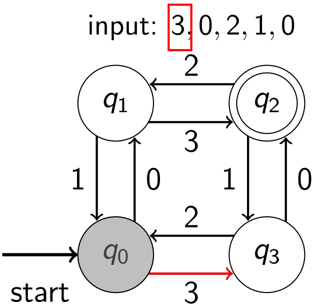


Finite State Machines

q_0	bottom-left	q_1	top-left
q_2	top-right	q_3	bottom-right

0	up	1	down
2	left	3	right

- at a time, the machine is in **one** state
- handle the input symbols one by one
- **decide** if the machine stops at the accepting state

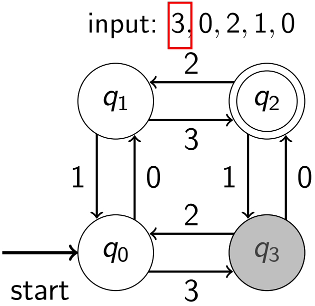


Finite State Machines

q_0	bottom-left	q_1	top-left
q_2	top-right	q_3	bottom-right

0	up	1	down
2	left	3	right

- at a time, the machine is in **one** state
- handle the input symbols one by one
- **decide** if the machine stops at the accepting state

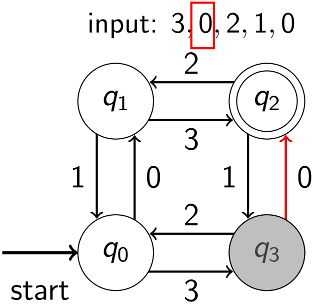


Finite State Machines

q_0	bottom-left	q_1	top-left
q_2	top-right	q_3	bottom-right

0	up	1	down
2	left	3	right

- at a time, the machine is in **one** state
- handle the input symbols one by one
- **decide** if the machine stops at the accepting state



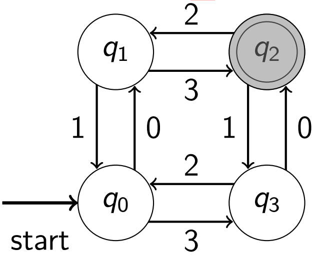
Finite State Machines

q_0	bottom-left	q_1	top-left
q_2	top-right	q_3	bottom-right

0	up	1	down
2	left	3	right

- at a time, the machine is in **one** state
- handle the input symbols one by one
- **decide** if the machine stops at the accepting state

input: $3, \boxed{0}, 2, 1, 0$

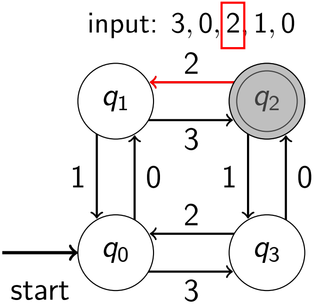


Finite State Machines

q_0	bottom-left	q_1	top-left
q_2	top-right	q_3	bottom-right

0	up	1	down
2	left	3	right

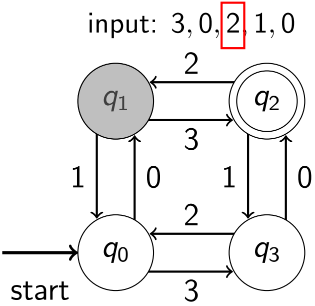
- at a time, the machine is in **one** state
- handle the input symbols one by one
- **decide** if the machine stops at the accepting state



Finite State Machines

q_0	bottom-left	q_1	top-left		0	up	1	down
q_2	top-right	q_3	bottom-right		2	left	3	right

- at a time, the machine is in **one** state
- handle the input symbols one by one
- **decide** if the machine stops at the accepting state

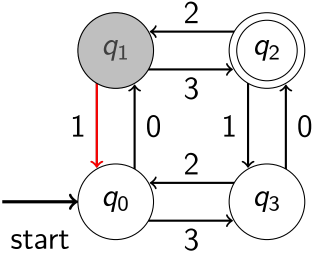


Finite State Machines

q_0	bottom-left	q_1	top-left		0	up	1	down
q_2	top-right	q_3	bottom-right		2	left	3	right

- at a time, the machine is in **one** state
- handle the input symbols one by one
- **decide** if the machine stops at the accepting state

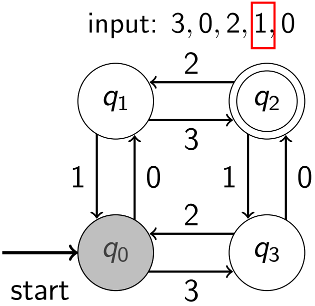
input: $3, 0, 2, \boxed{1,} 0$



Finite State Machines

q_0	bottom-left	q_1	top-left		0	up	1	down
q_2	top-right	q_3	bottom-right		2	left	3	right

- at a time, the machine is in **one** state
- handle the input symbols one by one
- **decide** if the machine stops at the accepting state



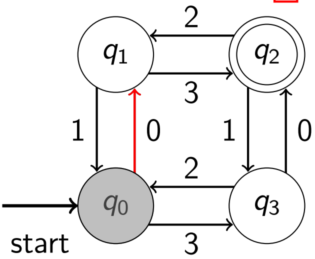
Finite State Machines

q_0	bottom-left	q_1	top-left
q_2	top-right	q_3	bottom-right

0	up	1	down
2	left	3	right

- at a time, the machine is in **one** state
- handle the input symbols one by one
- **decide** if the machine stops at the accepting state

input: $3, 0, 2, 1, \boxed{0}$



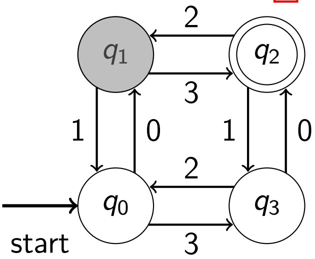
Finite State Machines

q_0	bottom-left	q_1	top-left
q_2	top-right	q_3	bottom-right

0	up	1	down
2	left	3	right

- at a time, the machine is in **one** state
- handle the input symbols one by one
- **decide** if the machine stops at the accepting state

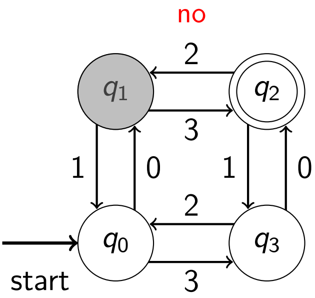
input: $3, 0, 2, 1, \boxed{0}$



Finite State Machines

q_0	bottom-left	q_1	top-left		0	up	1	down
q_2	top-right	q_3	bottom-right		2	left	3	right

- at a time, the machine is in **one** state
- handle the input symbols one by one
- **decide** if the machine stops at the accepting state



Finite State Machines

Input: a finite sequence of **symbols** from Σ.

Lanuage of the automata: the set of inputs that stops at the **accepting** state(s).

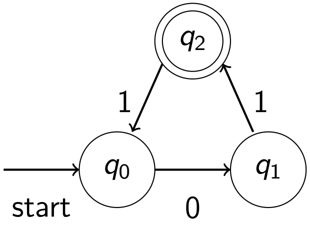
$$\mathcal{L}(D) = \{w \in \Sigma^* | \text{autumata } D \text{ accepts } w\}$$

Deterministic FSM: for each state-symbol pair (q_i, s), $s \in \Sigma$

- there is one transition
- there is only one transition

Finite State Machines

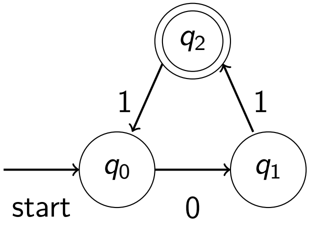
Is the diagram valid?



Finite State Machines

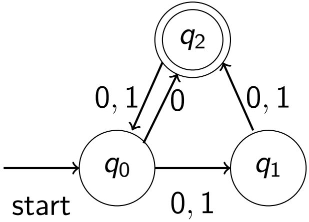
Is the diagram valid?

No! What is the transition for $(q_1, 0)$?



Finite State Machines

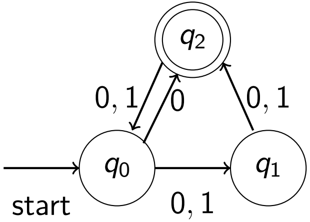
Is the diagram valid?



Finite State Machines

Is the diagram valid?

No! What is the transition for $(q_0, 0)$?



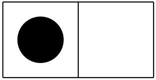
Memory of The Machine

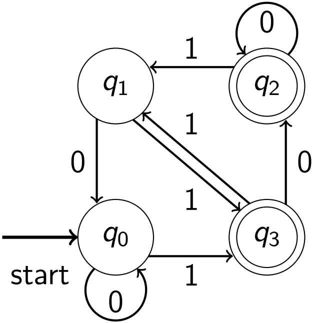
Finite state machine:

- the number of states are finite.
- can FSM remember the last performed transition?

q_0	left-0	q_1	left-1
q_2	right-0	q_3	right-1

0	stay	1	switch





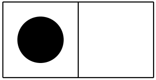
Memory of The Machine

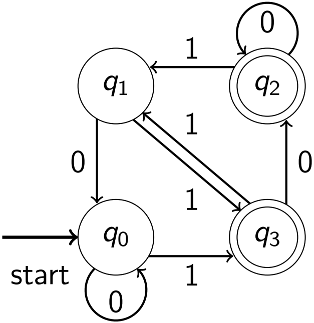
Finite state machine:

- the number of states are finite.
- can FSM remember the last performed transition?
- infinite memory?

q_0	left-0	q_1	left-1
q_2	right-0	q_3	right-1

0	stay	1	switch



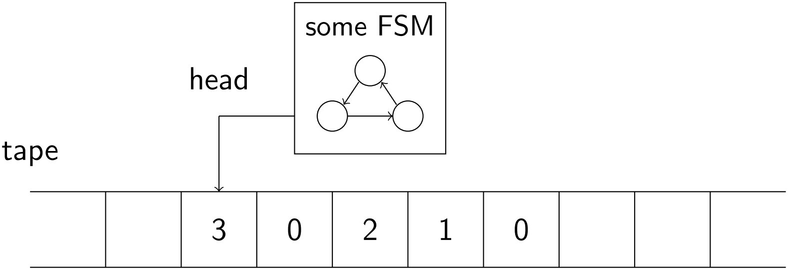


Turing Machines

Turing Machines

Turing machine: finite state machine + infinite tape

- input is written in the tape
- a tape head
  - start from the cell containing the first symbol of input
  - can move right or left along the tape
  - can read and write on a single memory cell



Alphabets

A Turing machine has **two** alphabets

- **Input alphabet** Σ: symbols in the inputs
- **Tape alphabet** Γ: symbols in the tape; always contains the blank symbol $\square$.

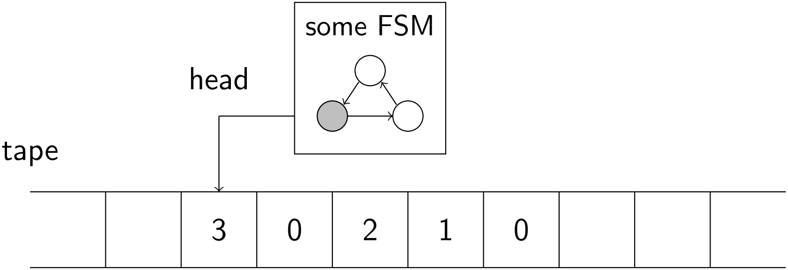
Guarantee: $\Sigma \subseteq \Gamma$; and $\square \notin \Sigma$

Initially,

- input is written in the tape
- the other cells are left blank $\square$
- head is positioned at the start of the input

Step by Step

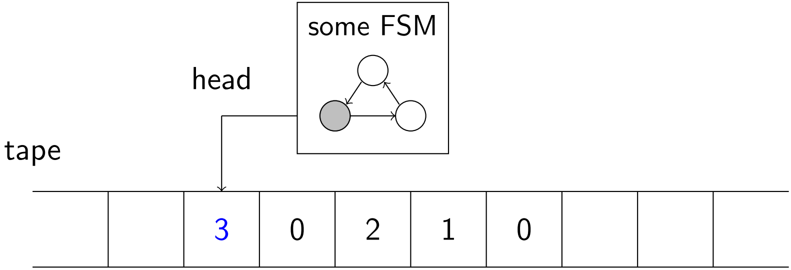
Turing machine processes the input in steps. In each step



Step by Step

Turing machine processes the input in steps. In each step

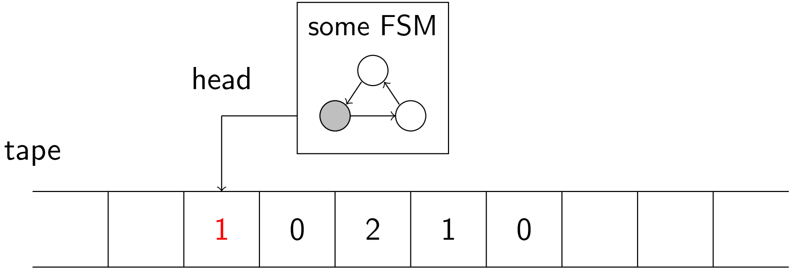
- read the symbol in the tape cell **under the tape head**



Step by Step

Turing machine processes the input in steps. In each step

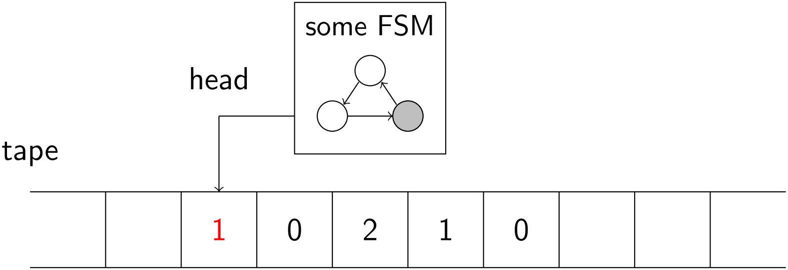
- read the symbol in the tape cell **under the tape head**
- write a symbol to the tape cell **under the tape head**



Step by Step

Turing machine processes the input in steps. In each step

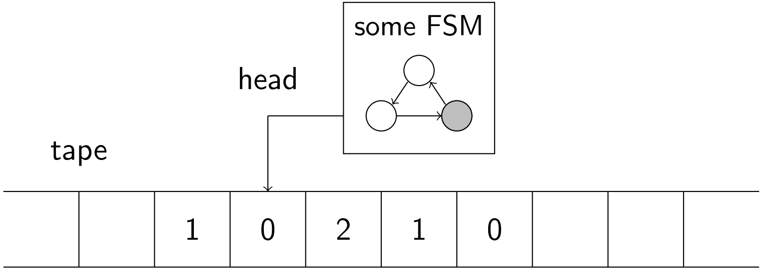
- read the symbol in the tape cell **under the tape head**
- write a symbol to the tape cell **under the tape head**
- change the state of FSM



Step by Step

Turing machine processes the input in steps. In each step

- read the symbol in the tape cell **under the tape head**
- write a symbol to the tape cell **under the tape head**
- change the state of FSM
- move the head to the left or to the right



Transitions

Transition has form $x \to y, D$, which means "upon reading x, replace it with symbol y, and move the tape head in direction D", where $D = \{L, R\}$.

When to finish? Turing machine does not stop processing the input when they finish reading it. It gives the result when arriving at

- **accept** state; or
- **reject** state

OR it may never stop.

Language Recognization

Language of a Turing machine M is defined as

$$\mathcal{L}(M) = \{w \in \Sigma^* | \text{Turing machine } M \text{ accepts } w\}$$

Consider language $\mathcal{L} = \{0^n 1^n | n \in \mathbb{N}\}$. How might we build a Turing machine M, such that

$$\mathcal{L}(M) = \mathcal{L}$$

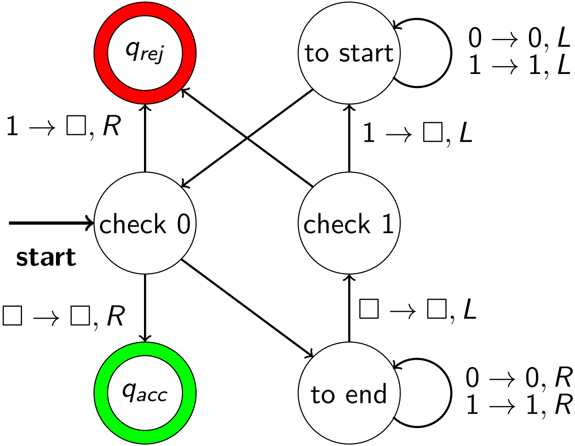
Language Recognization

Construct M s.t. $\mathcal{L}(M) = \mathcal{L} = \{0^n1^n | n \in \mathbb{N}\}$.

Let $\epsilon \in \Sigma$ be the empty string, and we know

- $\epsilon \in \mathcal{L}$
- $0w1 \in \mathcal{L} \Leftrightarrow w \in \mathcal{L}$
- $1w \notin \mathcal{L}$
- $w0 \notin \mathcal{L}$

Language Recognization

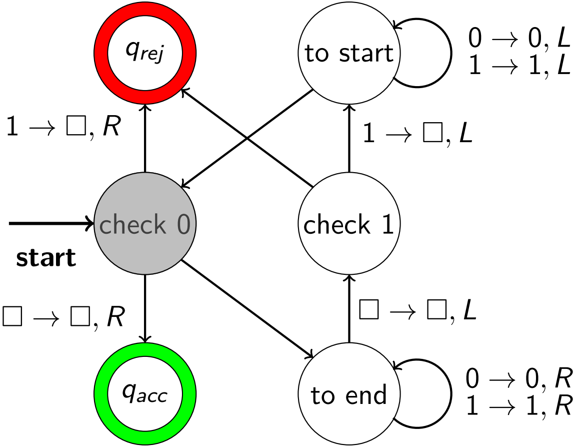


$$\text{check 0} \xrightarrow{0 \to \square, R} \text{to end}$$

$$\text{check 1} \xrightarrow[\square \to \square, R]{0 \to 0, R} q_{rej}$$

$$\text{to start} \xrightarrow[\square \to \square, R]{} \text{check 0}$$

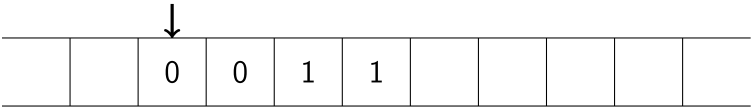
Language Recognization



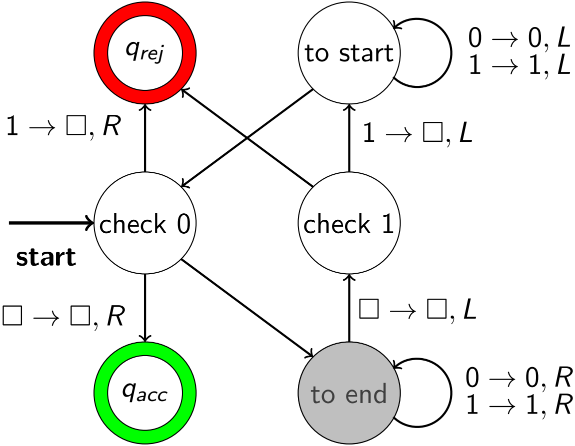
check 0 $\xrightarrow{0 \to \square, R}$ to end

check 1 $\xrightarrow[\square \to \square, R]{0 \to 0, R}$ q_{rej}

to start $\xrightarrow[\square \to \square, R]{}$ check 0



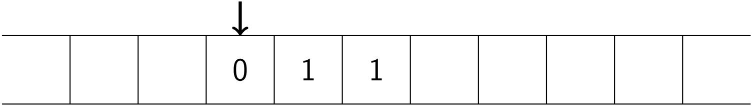
Language Recognization



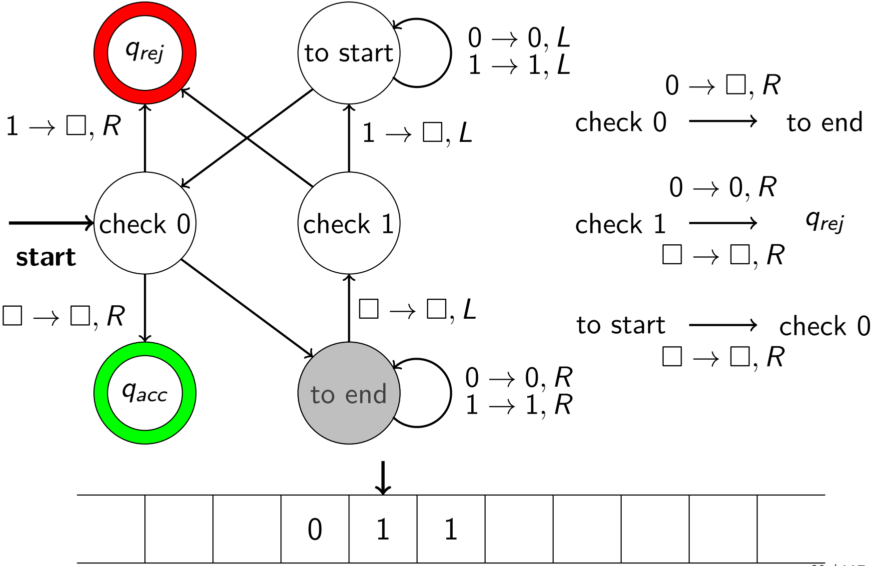
$$\text{check 0} \xrightarrow{0 \to \square, R} \text{to end}$$

$$\text{check 1} \xrightarrow[\square \to \square, R]{0 \to 0, R} q_{rej}$$

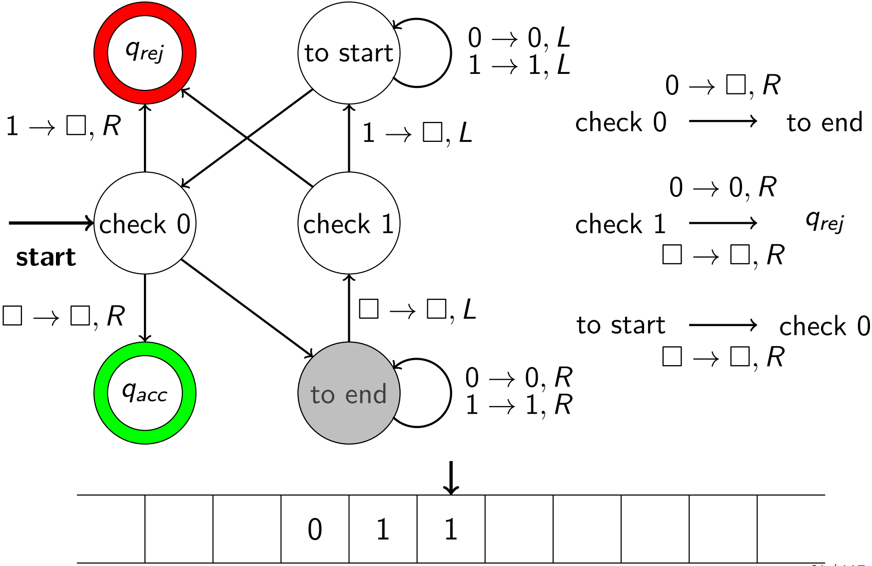
$$\text{to start} \xrightarrow[\square \to \square, R]{} \text{check 0}$$



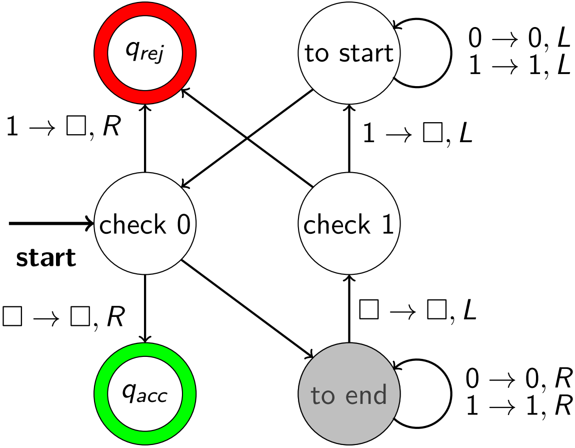
Language Recognization



Language Recognization



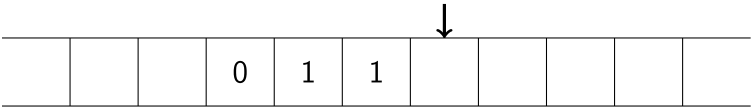
Language Recognization



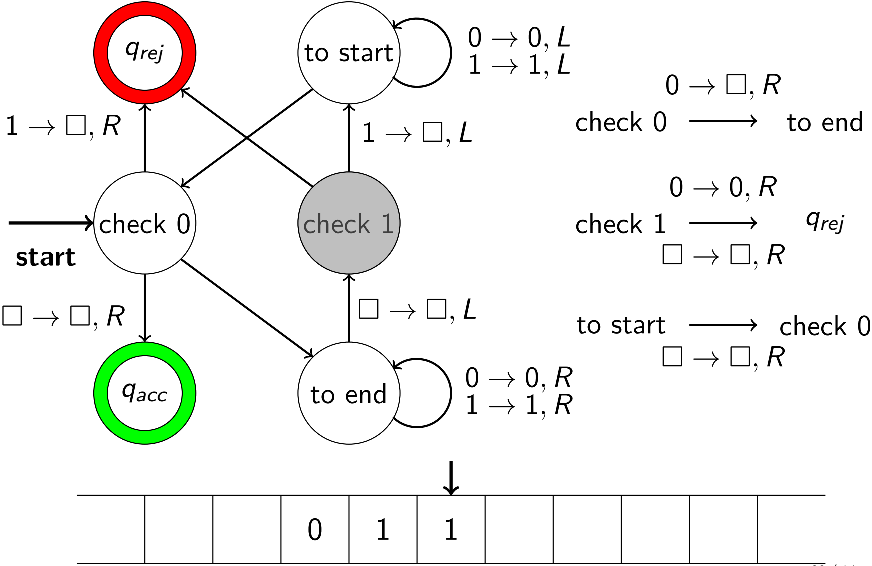
check 0 $\xrightarrow{0 \to \square, R}$ to end

check 1 $\xrightarrow[\square \to \square, R]{0 \to 0, R}$ q_{rej}

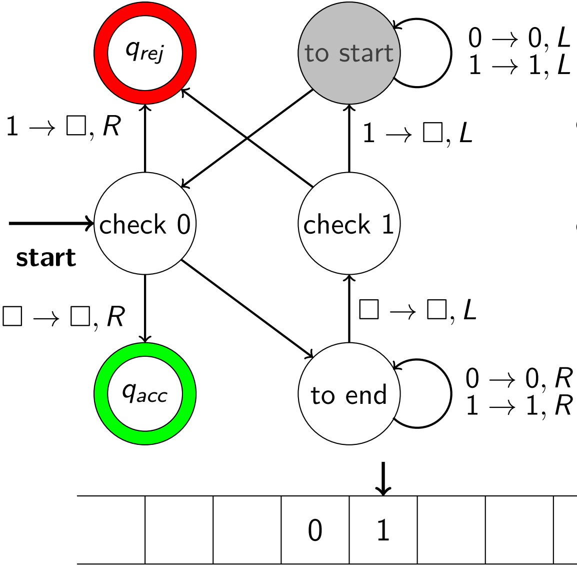
to start $\xrightarrow[\square \to \square, R]{}$ check 0



Language Recognization



Language Recognization

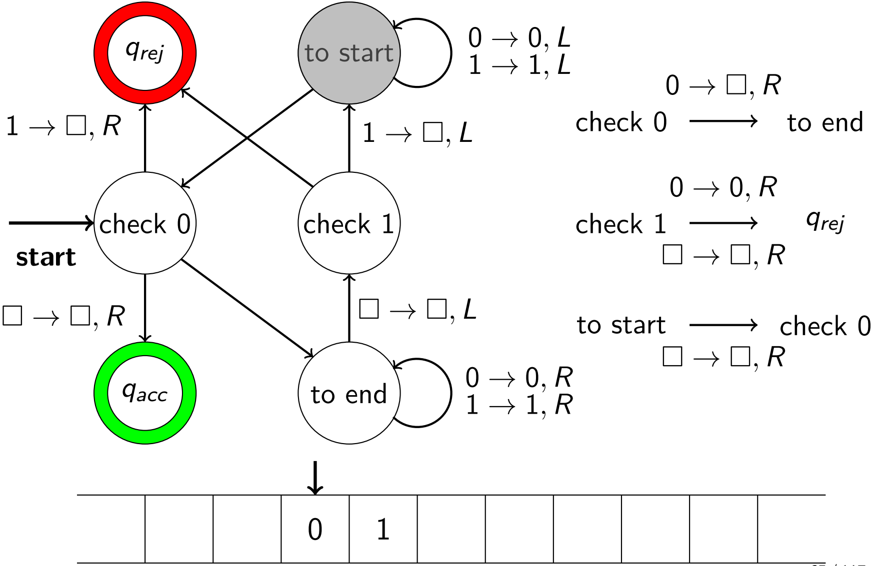


$$\text{check 0} \xrightarrow{0 \to \square, R} \text{to end}$$

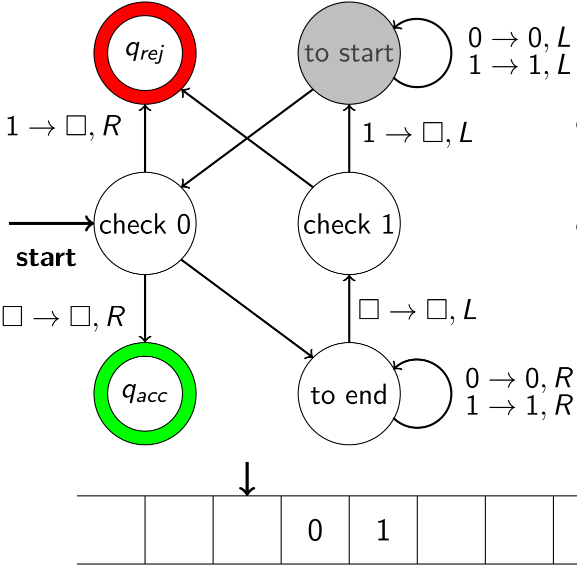
$$\text{check 1} \xrightarrow[\square \to \square, R]{0 \to 0, R} q_{rej}$$

$$\text{to start} \xrightarrow[\square \to \square, R]{} \text{check 0}$$

Language Recognization



Language Recognization

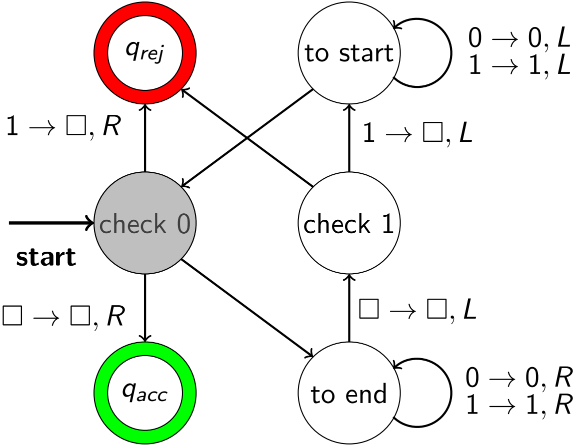


$$\text{check 0} \xrightarrow{0 \to \square, R} \text{to end}$$

$$\text{check 1} \xrightarrow[\square \to \square, R]{0 \to 0, R} q_{rej}$$

$$\text{to start} \xrightarrow[\square \to \square, R]{} \text{check 0}$$

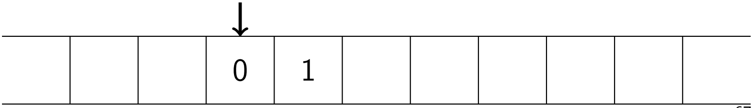
Language Recognization



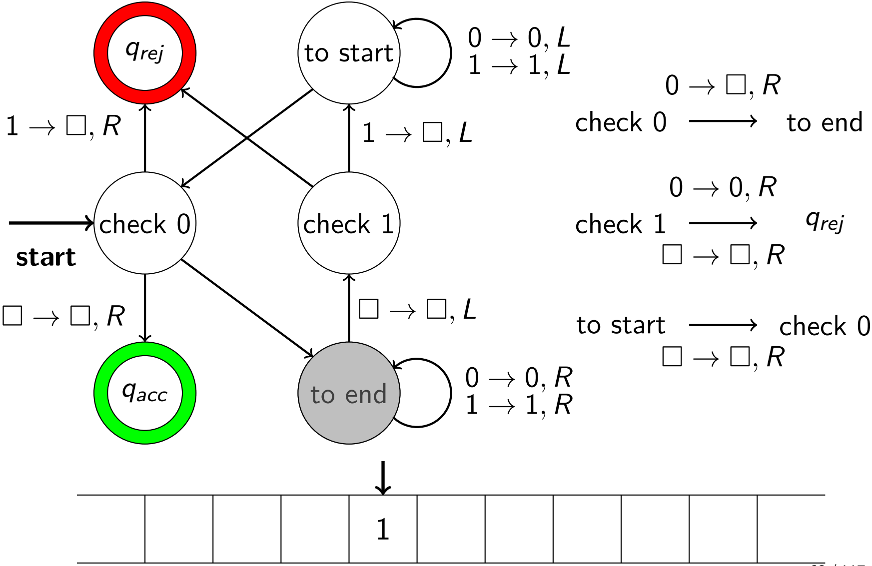
check 0 $\xrightarrow{0 \to \square, R}$ to end

check 1 $\xrightarrow[\square \to \square, R]{0 \to 0, R}$ q_{rej}

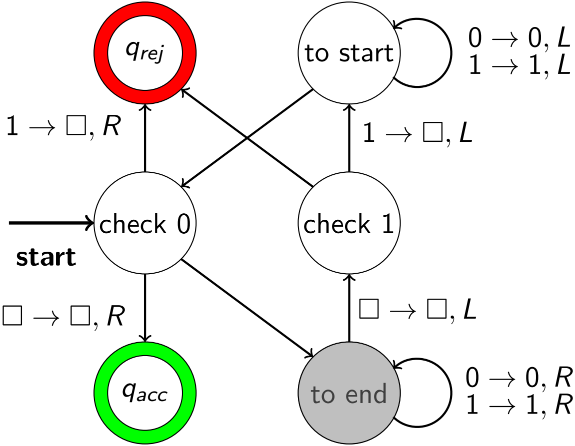
to start $\xrightarrow[\square \to \square, R]{}$ check 0



Language Recognization



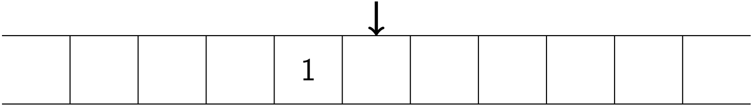
Language Recognization



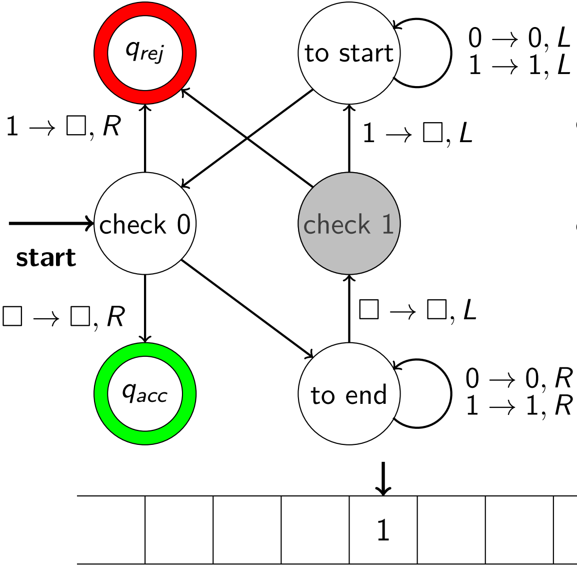
check 0 $\xrightarrow{0 \to \square, R}$ to end

check 1 $\xrightarrow[\square \to \square, R]{0 \to 0, R}$ q_{rej}

to start $\xrightarrow[\square \to \square, R]{}$ check 0



Language Recognization

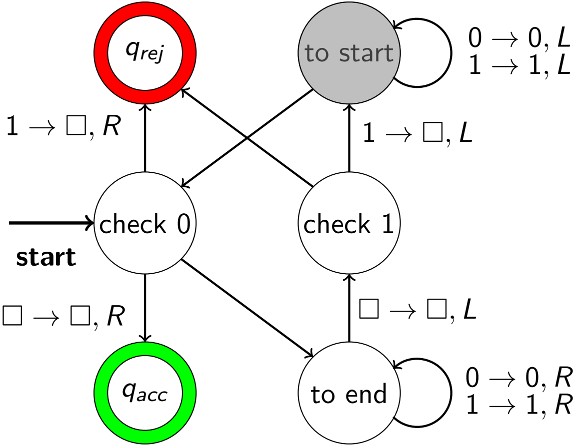


check 0 $\xrightarrow{0 \to \square, R}$ to end

check 1 $\xrightarrow[\square \to \square, R]{0 \to 0, R}$ q_{rej}

to start $\xrightarrow[\square \to \square, R]{}$ check 0

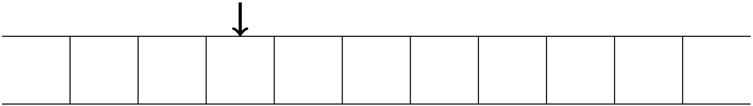
Language Recognization



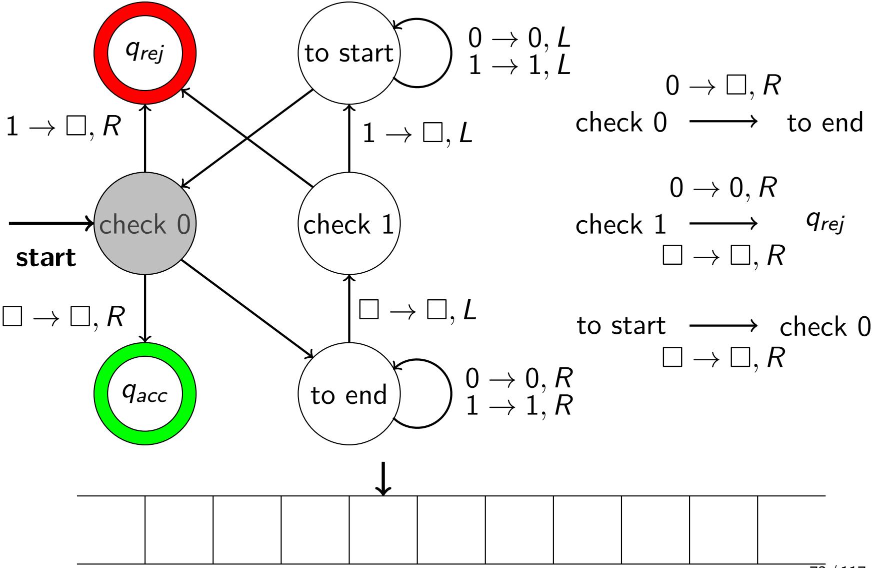
check 0 $\xrightarrow{0 \to \square, R}$ to end

check 1 $\xrightarrow[\square \to \square, R]{0 \to 0, R}$ q_{rej}

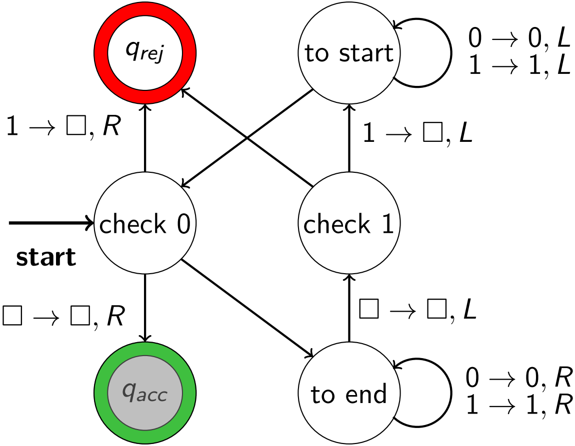
to start $\xrightarrow[\square \to \square, R]{}$ check 0



Language Recognization



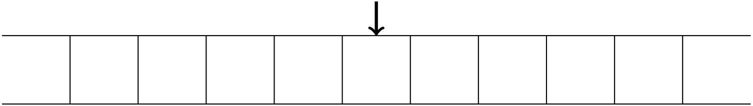
Language Recognization



check 0 $\xrightarrow{0 \to \square, R}$ to end

check 1 $\xrightarrow[\square \to \square, R]{0 \to 0, R}$ q_{rej}

to start $\xrightarrow[\square \to \square, R]{}$ check 0



Example: Take Odds

Problem: given a sequence of :-separated words $w_1 : w_2 : \ldots : w_{2n}$, extract the words

$$w_1 : w_3 : \ldots : w_{2n-1}$$

	1	0	:	0					

						1	0		

Example: Take Odds

Problem: given a sequence of :-separated words $w_1 : w_2 : \ldots : w_{2n}$, extract the words

$$w_1 : w_3 : \ldots : w_{2n-1}$$

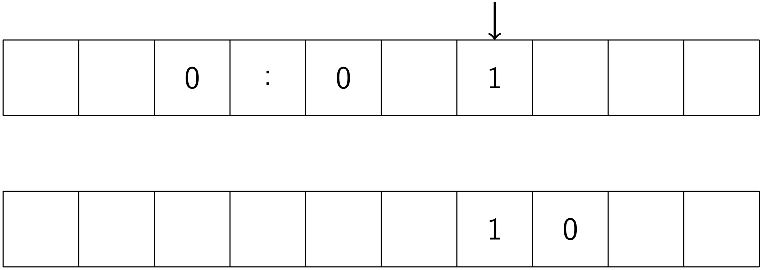
	1	0	:	0					

						1	0		

Example: Take Odds

Problem: given a sequence of :-separated words
$w_1 : w_2 : \ldots : w_{2n}$, extract the words

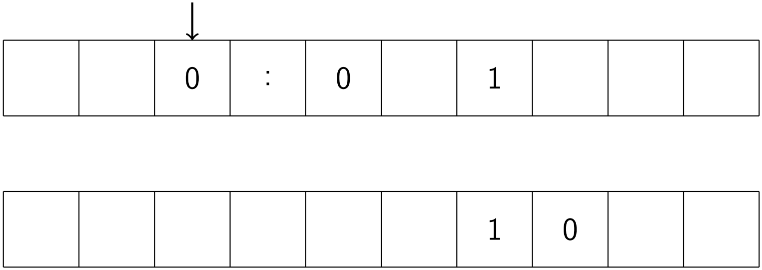
$$w_1 : w_3 : \ldots : w_{2n-1}$$



Example: Take Odds

Problem: given a sequence of :-separated words
$w_1 : w_2 : \ldots : w_{2n}$, extract the words

$$w_1 : w_3 : \ldots : w_{2n-1}$$



Example: Take Odds

Problem: given a sequence of :-separated words $w_1 : w_2 : \ldots : w_{2n}$, extract the words

$$w_1 : w_3 : \ldots : w_{2n-1}$$

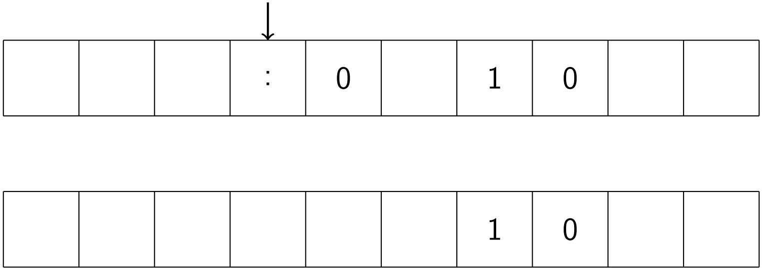
			:	0		1	0		

						1	0		

Example: Take Odds

Problem: given a sequence of :-separated words $w_1 : w_2 : \ldots : w_{2n}$, extract the words

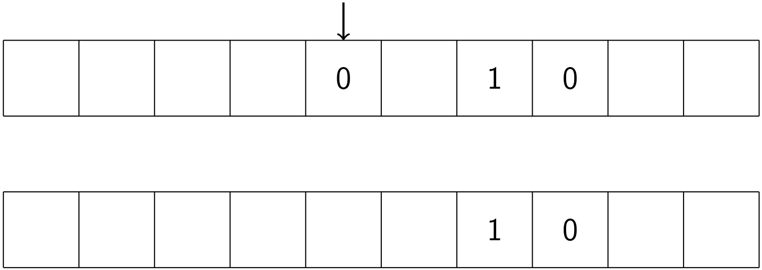
$$w_1 : w_3 : \ldots : w_{2n-1}$$



Example: Take Odds

Problem: given a sequence of :-separated words $w_1 : w_2 : \ldots : w_{2n}$, extract the words

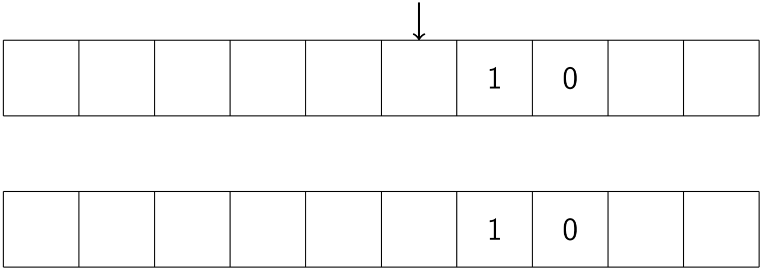
$$w_1 : w_3 : \ldots : w_{2n-1}$$



Example: Take Odds

Problem: given a sequence of :-separated words $w_1 : w_2 : \ldots : w_{2n}$, extract the words

$$w_1 : w_3 : \ldots : w_{2n-1}$$



Nondeterministic Turing Machines

Nondeterminism

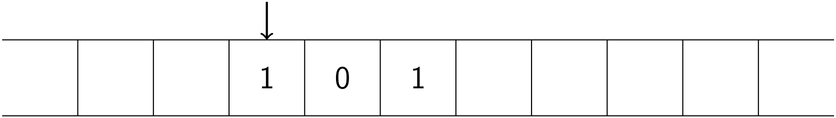
Determinism: in face of the context and the choices, to find the best one, try with every choices or some choices.

Nondeterminism: in face of the same context and the choices, make a **guess** and go with the guessed choice.

Nondeterministic Turing Machine (NTM)

Nondeterministic Turing machine

- finite state machine
- infinite tape
- tape head: read, write, move (L/R)
- for a given state-symbol pair, there can be more than one transitions



Nondeterministic Turing Machine (NTM)

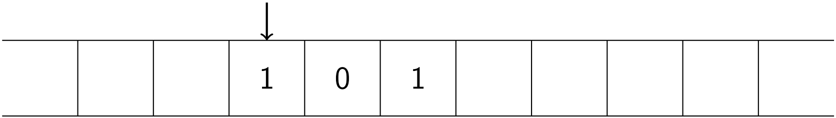
Nondeterministic Turing machine

- finite state machine
- infinite tape
- tape head: read, write, move (L/R)
- for a given state-symbol pair, there can be more than one transitions

make a choice

$1 \to 1, R$

$1 \to \square, L$



Nondeterministic Turing Machine (NTM)

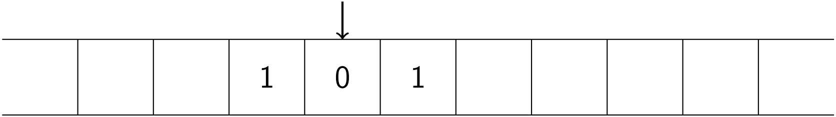
Nondeterministic Turing machine

- finite state machine
- infinite tape
- tape head: read, write, move (L/R)
- for a given state-symbol pair, there can be more than one transitions

make a choice

$1 \to 1, R$

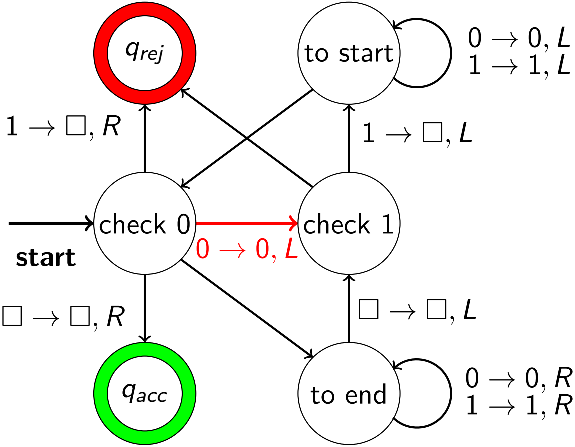
$1 \to \square, L$



Nondeterministic Turing Machine

Language of a nondeterministic Turing machine M is defined

$$\mathcal{L}(M) = \{w \in \Sigma^* | M \text{ can make a series of choices to accept } w\}$$



$$\text{check 0} \xrightarrow{0 \to \square, R} \text{to end}$$

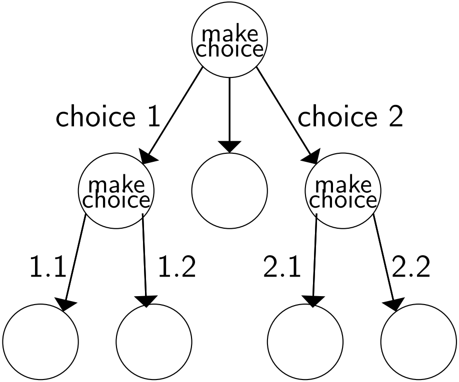
$$\text{check 1} \xrightarrow[\square \to \square, R]{0 \to 0, R} q_{rej}$$

$$\text{to start} \xrightarrow[\square \to \square, R]{} \text{check 0}$$

Tree Computation

NTM: make a choice

1. choice 1: make another choice
   1. choice 1.1
   2. choice ...
2. choice 2: make another choice
   1. choice 2.1
   2. choice ...
3. choice ...

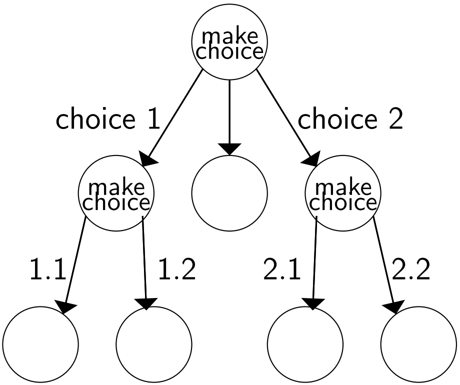


Tree Computation

NTM: make a choice

1. choice 1: make another choice
   1. choice 1.1
   2. choice . . .
2. choice 2: make another choice
   1. choice 2.1
   2. choice . . .
3. choice . . .

DTM: try all cases!

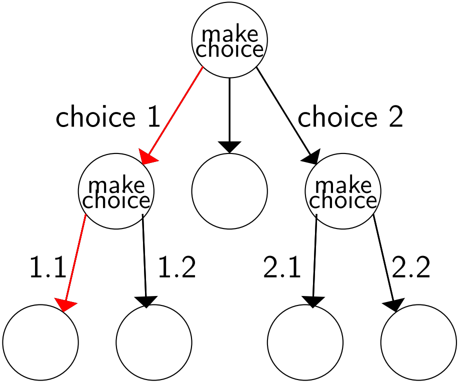


Tree Computation

NTM: make a choice

1. choice 1: make another choice
   1. choice 1.1
   2. choice ...
2. choice 2: make another choice
   1. choice 2.1
   2. choice ...
3. choice ...

DTM: try all cases!

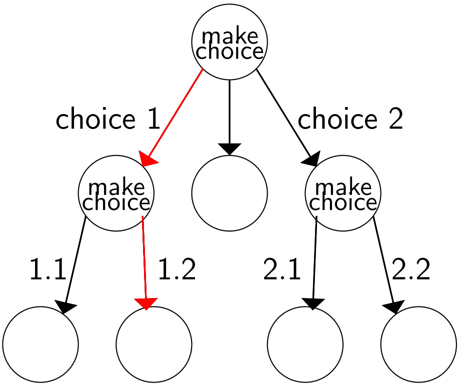


Tree Computation

NTM: make a choice

1. choice 1: make another choice
   1. choice 1.1
   2. choice ...
2. choice 2: make another choice
   1. choice 2.1
   2. choice ...
3. choice ...

DTM: try all cases!

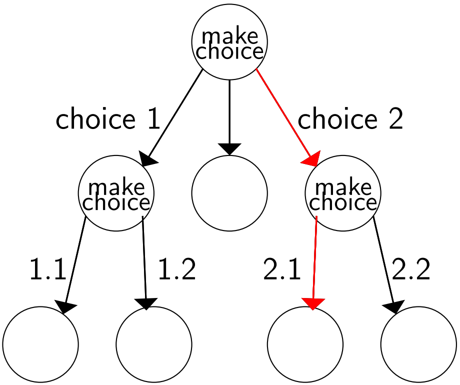


Tree Computation

NTM: make a choice

1. choice 1: make another choice
   1. choice 1.1
   2. choice . . .
2. choice 2: make another choice
   1. choice 2.1
   2. choice . . .
3. choice . . .

DTM: try all cases!

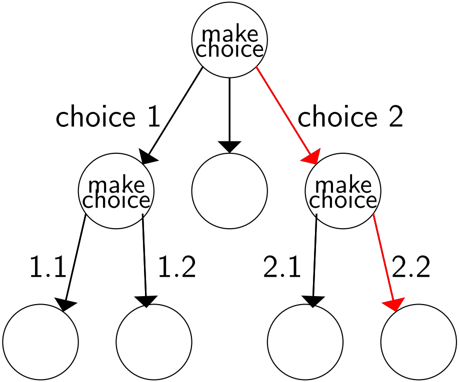


Tree Computation

NTM: make a choice

1. choice 1: make another choice
   1. choice 1.1
   2. choice ...
2. choice 2: make another choice
   1. choice 2.1
   2. choice ...
3. choice ...

DTM: try all cases!

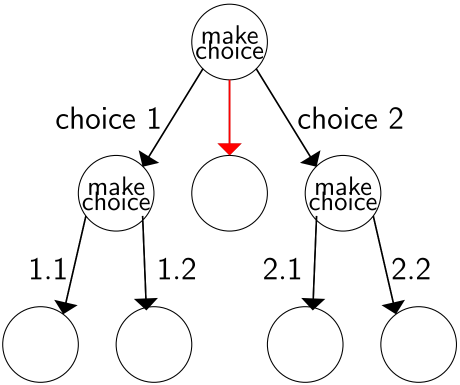


Tree Computation

NTM: make a choice

1. choice 1: make another choice
   1. choice 1.1
   2. choice . . .
2. choice 2: make another choice
   1. choice 2.1
   2. choice . . .
3. choice . . .

DTM: try all cases!



Proof: A Series of Choices

Language of a nondeterministic Turing machine M is defined

$\mathcal{L}(M) = \{w \in \Sigma^* | M \text{ can make a series of choices to accept } w\}$

Claim: For any NTM N, there is a DTM D such that

$$\mathcal{L}(D) = \mathcal{L}(N)$$

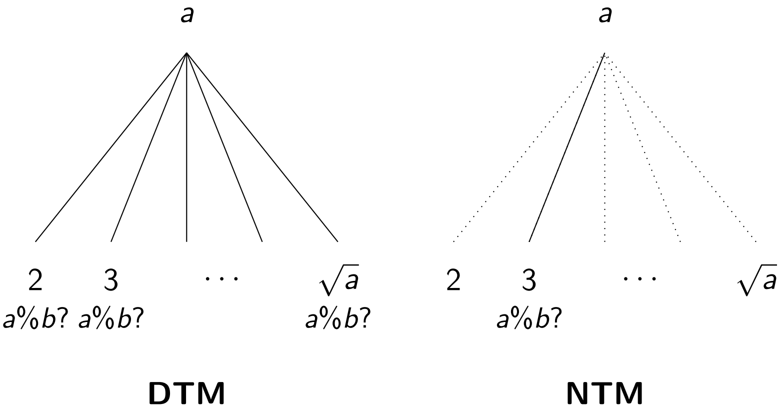
To prove $w \in \mathcal{L}(N) = \mathcal{L}(D)$

- DTM: try a lot of cases, if not all
- NTM: much easier given a series of good choices

Example: Composite Number

Problem: given a positive integer a, decide if $a = b \cdot c$, with

- $b \in \mathbb{Z}^+$, $1 < b < a$
- $c \in \mathbb{Z}^+$, $1 < c < a$



Decision Problems

Decision problem: answer yes or no.

Example

- given x, y and z, decide if $x + y = z$.
- given a number x, decide if x is prime.

P v.s. NP

P: the problem that can be decided (accept or reject) by **DTM** in polynomial time.

NP: the problem that can be decided (accept or reject) by **NTM** in polynomial time, **given a series of choices**.

P v.s. NP

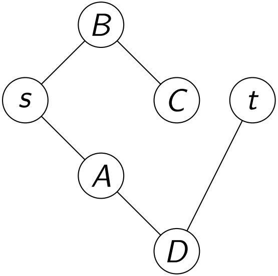
P: the problem that can be decided (accept or reject) by **DTM** in polynomial time.

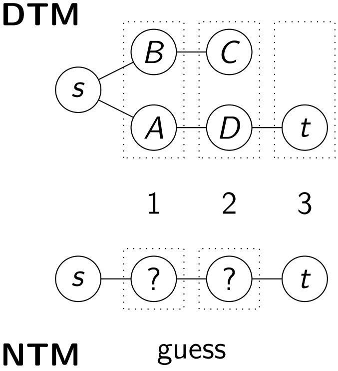
NP: the problem that can be decided (accept or reject) by **NTM** in polynomial time, **given a series of choices**.

$P \subseteq NP$!

Example: Reachability

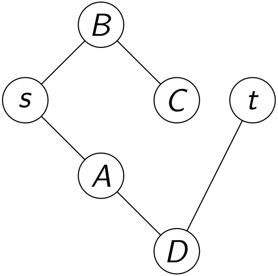
Problem: given a graph $G = (V, E)$, and $s, t \in V$, decide if there is a path from s to t, with at most k hops.

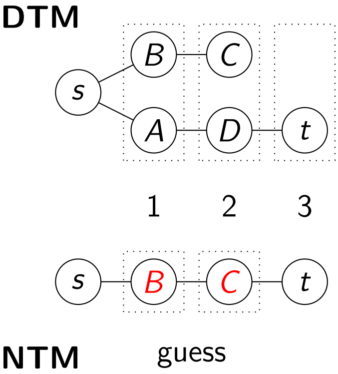




Example: Reachability

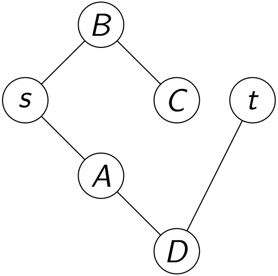
Problem: given a graph $G = (V, E)$, and $s, t \in V$, decide if there is a path from s to t, with at most k hops.



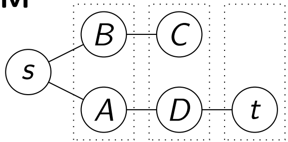


Example: Reachability

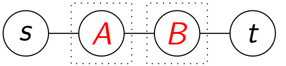
Problem: given a graph $G = (V, E)$, and $s, t \in V$, decide if there is a path from s to t, with at most k hops.



DTM



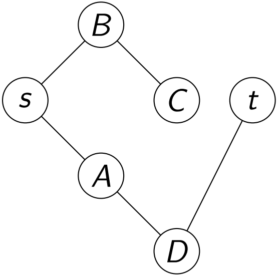
1 2 3

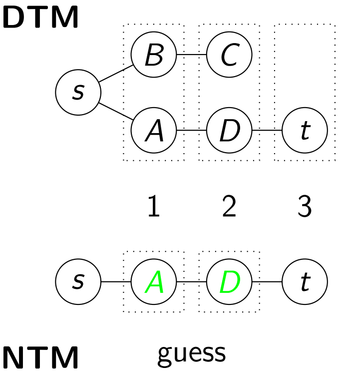


NTM guess

Example: Reachability

Problem: given a graph $G = (V, E)$, and $s, t \in V$, decide if there is a path from s to t, with at most k hops.





Solve NP Problem with A Certifier

P v.s. NP

P: the problem that can be decided by **DTM** in polynomial time.

NP: the problem that can be decided by **NTM** in polynomial time, **given a series of choices**.

Certification: **a series of choices**.

P v.s. NP

P: the instances can be solved by a polynomial-time algorithm.

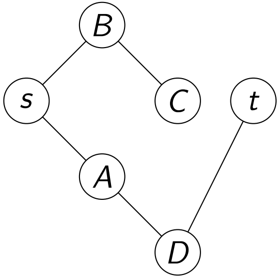
NP: any instance can be checked by a polynomial-time certifier.

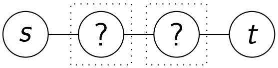
Certifier: given an instance α and a certification t, check if $\alpha \in \mathcal{L}$ can be proved by t.

Certifier Example

Problem: given a graph $G = (V, E)$, and $s, t \in V$, decide if there is a path from s to t, with at most k hops.

Certifier: C(α, t)

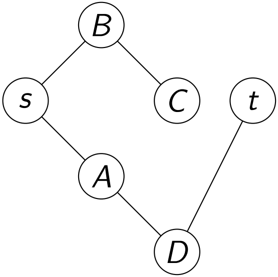




NTM

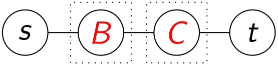
Certifier Example

Problem: given a graph $G = (V, E)$, and $s, t \in V$, decide if there is a path from s to t, with at most k hops.



Certifier: C(α, t)

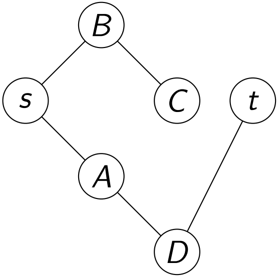
t: $s - B - C - t$



NTM

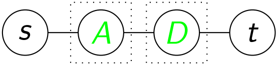
Certifier Example

Problem: given a graph $G = (V, E)$, and $s, t \in V$, decide if there is a path from s to t, with at most k hops.



Certifier: C(α, t)

t: $s - A - D - t$



NTM

Certifier Example

Problem: given an integer a, decide if $a = b \cdot c$, such that

- $b \in \mathbb{Z}, 1 < b < a$
- $c \in \mathbb{Z}, 1 < c < a$

Algorithm: C(a, b)

---

if $b \leq 1$ *or* $b \geq a$ **then return** false;
if $a\%b = 0$ **then**
  return true;
end
else
  return false;
end

Undecidability: Halting Problem

Undecidable Problems

Undecidable problem can not be decided by a Turing machine.

The Halting Problem

Problem: given a program $\mathcal{P}$ and an input instance α, can we decide if $\mathcal{P}(\alpha)$ halts (finally)?

- Program $\mathcal{P}$, input α
  - $\mathcal{P}(\alpha)$ return 0 if $\alpha \in \mathbb{Z}$ and $\alpha \leq 0$;
  - $\mathcal{P}(\alpha)$ return 1 if $\alpha \in \mathbb{Z}$ and $\alpha > 0$;
  - $\mathcal{P}(\alpha)$ loops forever (for thinking) if $\alpha \notin \mathbb{Z}$;
- HM($\mathcal{P}$, α)
  - yes, if $\alpha \in \mathbb{Z}$
  - no, if $\alpha \notin \mathbb{Z}$

The Halting Machine

- **Assume** we have program HM($\cdot$, $\cdot$), for any $\mathcal{P}$ and α,
  - HM($\mathcal{P}$, α) $=$ yes if $\mathcal{P}(\alpha)$ halts
  - HM($\mathcal{P}$, α) $=$ no if $\mathcal{P}(\alpha)$ does not halt

The Halting Machine

- **Assume** we have program HM($\cdot$, $\cdot$), for any $\mathcal{P}$ and α,
  - HM($\mathcal{P}$, α) $=$ yes if $\mathcal{P}(\alpha)$ halts
  - HM($\mathcal{P}$, α) $=$ no if $\mathcal{P}(\alpha)$ does not halt
- **Then construct** CM($\cdot$), such that
  - CM($\mathcal{P}$) loops forever if HM($\mathcal{P}$, $\mathcal{P}$) says yes.
  - CM($\mathcal{P}$) halts immediately if HM($\mathcal{P}$, $\mathcal{P}$) says no.

The Halting Machine

- **Assume** we have program HM$(\cdot, \cdot)$, for any $\mathcal{P}$ and α,
  - HM$(\mathcal{P}, \alpha) =$ yes if $\mathcal{P}(\alpha)$ halts
  - HM$(\mathcal{P}, \alpha) =$ no if $\mathcal{P}(\alpha)$ does not halt
- **Then construct** CM$(\cdot)$, such that
  - CM$(\mathcal{P})$ loops forever if HM$(\mathcal{P}, \mathcal{P})$ says yes.
  - CM$(\mathcal{P})$ halts immediately if HM$(\mathcal{P}, \mathcal{P})$ says no.
- Does CM(CM) halt?
  - CM(CM) halts? HM(CM, CM) says yes, and thus CM(CM) loops forever.
  - CM(CM) does not halt? HM(CM, CM) says no, and thus CM(CM) halts.

THANK YOU

中国科学院深圳先进技术研究院
SHENZHEN INSTITUTES OF ADVANCED TECHNOLOGY
CHINESE ACADEMY OF SCIENCES