CS420

Introduction to the Theory of Computation

Lecture 19: Turing Machines

Tiago Cogumbreiro

Today we will learn...

- Recap exercises
- TM configuration and configuration history
- TM acceptance
- Variants of Turing Machines
  - Multi-tape
  - Nondeterministic

> Section 3.1, 3.2, and 3.3

Supplementary material

- Professor Harry Porter's video
- Professor Dan Gusfield's video
- Turing Machines, Stanford Encyclopedia of Philosophy

Exercises

Exercise 1

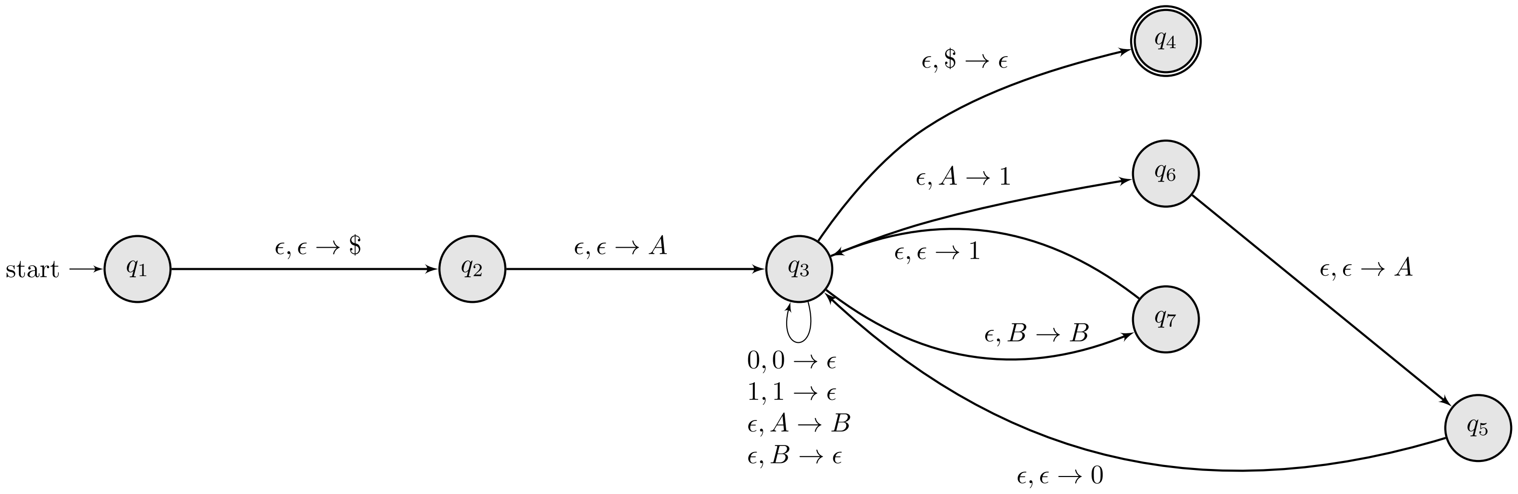
Convert the following grammar into a PDA

$$A \rightarrow 0A1 \mid B$$
$$B \rightarrow 1B \mid \epsilon$$

Exercise 1

Convert the following grammar into a PDA

$$A \to 0A1 \mid B$$
$$B \to 1B \mid \epsilon$$



Exercise 2

Show that if $L_1 \cup L_2$ is not context-free, then either L_1 is not context-free or L_2 is not context free.

Proof.

Exercise 2

Show that if $L_1 \cup L_2$ is not context-free, then either L_1 is not context-free or L_2 is not context free.

Proof.

1. We know that if L_1 is CF and L_2 is CF, then $L_1 \cup L_2$ is CF.
2. Apply the contrapositive to (1) and we conclude our proof.

Exercise 3

We know that $L_2 = \{w \mid w = a^n b^n c^n \vee |w| \text{ is even}\}$ is not context free.

Show that $L_3 = \{a^n b^n c^n \mid n \geq 0\}$ is not context-free without using the Pumping Lemma for CF or the Theorem of non-CF from Lecture 11.

Proof.

Exercise 3

We know that $L_2 = \{w \mid w = a^n b^n c^n \vee |w| \text{ is even}\}$ is not context free.

Show that $L_3 = \{a^n b^n c^n \mid n \geq 0\}$ is not context-free without using the Pumping Lemma for CF or the Theorem of non-CF from Lecture 11.

Proof.

1. It is easy to see that $L_2 = L_3 \cup L_4$ where $L_4 = \{w \mid \ |w| \text{ is even}\}$.

Exercise 3

We know that $L_2 = \{w \mid w = a^n b^n c^n \vee |w| \text{ is even}\}$ is not context free.

Show that $L_3 = \{a^n b^n c^n \mid n \geq 0\}$ is not context-free without using the Pumping Lemma for CF or the Theorem of non-CF from Lecture 11.

Proof.

1. It is easy to see that $L_2 = L_3 \cup L_4$ where $L_4 = \{w \mid \ |w| \text{ is even}\}$.
2. We apply the previous theorem of Exercise 1 and get that either L_3 is not context free, or L_4 is not-context free.

Exercise 3

We know that $L_2 = \{w \mid w = a^n b^n c^n \vee |w| \text{ is even}\}$ is not context free.

Show that $L_3 = \{a^n b^n c^n \mid n \geq 0\}$ is not context-free without using the Pumping Lemma for CF or the Theorem of non-CF from Lecture 11.

Proof.

1. It is easy to see that $L_2 = L_3 \cup L_4$ where $L_4 = \{w \mid |w| \text{ is even}\}$.
2. We apply the previous theorem of Exercise 1 and get that either L_3 is not context free, or L_4 is not-context free.
3. But we know that L_4 is regular and therefore context-free.

Exercise 3

We know that $L_2 = \{w \mid w = a^n b^n c^n \vee |w| \text{ is even}\}$ is not context free.

Show that $L_3 = \{a^n b^n c^n \mid n \geq 0\}$ is not context-free without using the Pumping Lemma for CF or the Theorem of non-CF from Lecture 11.

Proof.

1. It is easy to see that $L_2 = L_3 \cup L_4$ where $L_4 = \{w \mid \ |w| \text{ is even}\}$.
2. We apply the previous theorem of Exercise 1 and get that either L_3 is not context free, or L_4 is not-context free.
3. But we know that L_4 is regular and therefore context-free.
4. Thus, L_2 is not CF.

Turing Machine:

configuration & configuration history

Turing Machines

Definition 3.3

A Turing machine is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$

1. Q set of states
2. Σ input alphabet not containing the blank symbol $\sqcup$
3. Γ the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$
4. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{\text{L}, \text{R}\}$ transition function
5. $q_0 \in Q$ is the start state
6. q_{accept} is the accept state
7. q_{reject} is the reject state ($q_{reject} \neq q_{accept}$)

To ponder..

- What is the minimum number of states?
- Can the input and the tape alphabets be the same?
- Write a Turing machine with the minimum number of states that recognizes $\emptyset$
- Write a Turing machine with the minimum number of states that recognizes $\Sigma^\star$

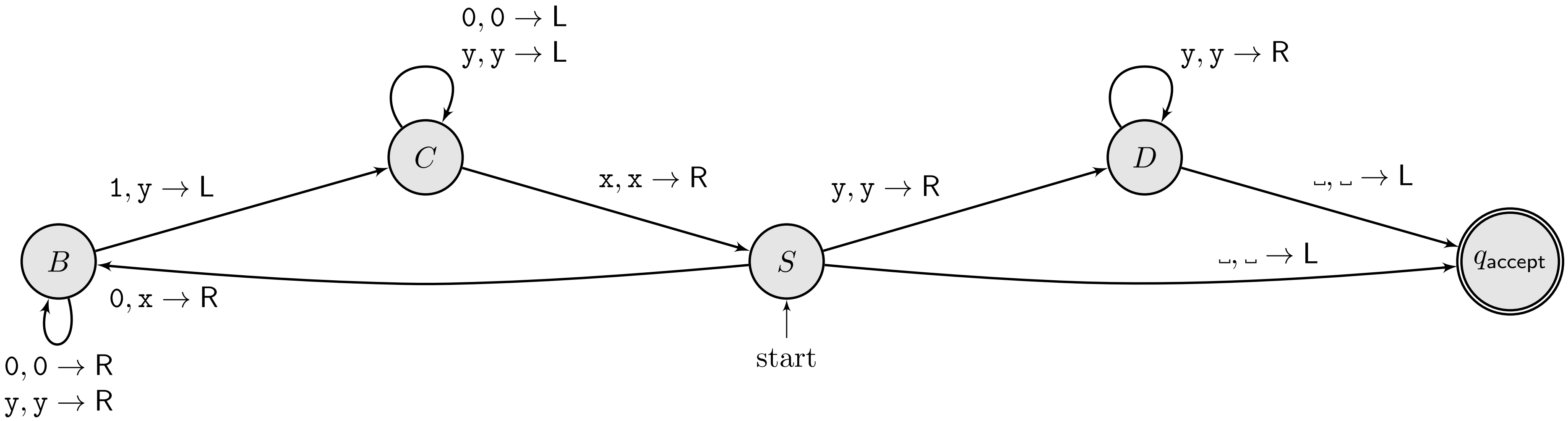
Configuration

> A configuration is a snapshot of a computation. That is, it contains all information necessary to resume (or replay) a computation from any point in time.

A configuration consists of

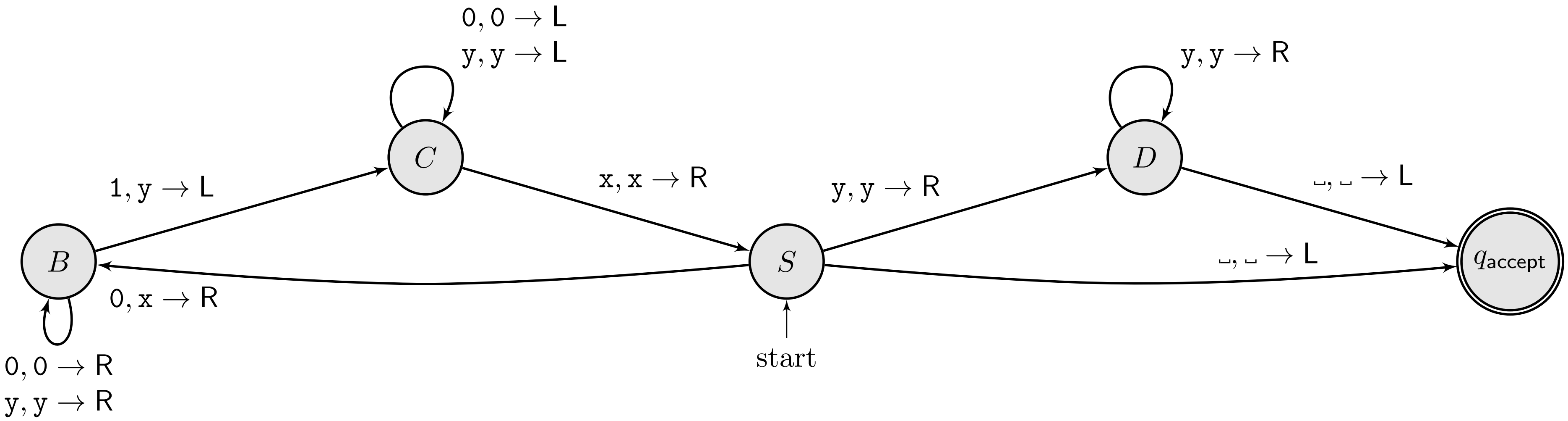
- the tape
- the head of the tape
- the current state

Example 2



State	Tape
S	0011
B	X011
B	X011
C	X0Y1

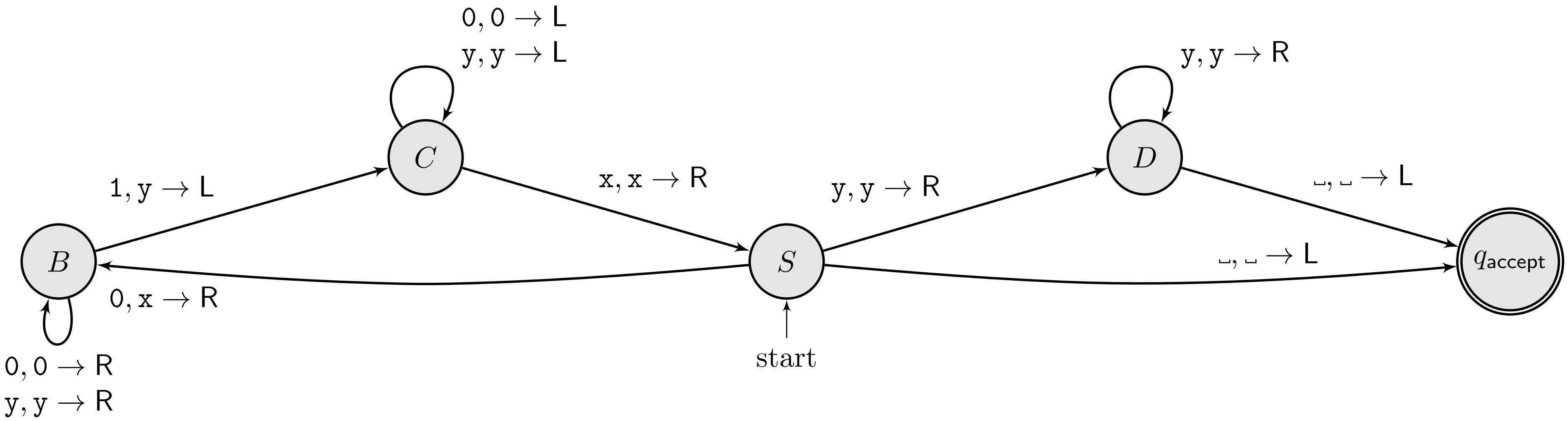
Example 2



State	**Tape**
S	0011
B	X011
B	X011
C	X0Y1

State	**Tape**
C	X0Y1
S	X0Y1
B	XXY1
B	XXY1

Example 2

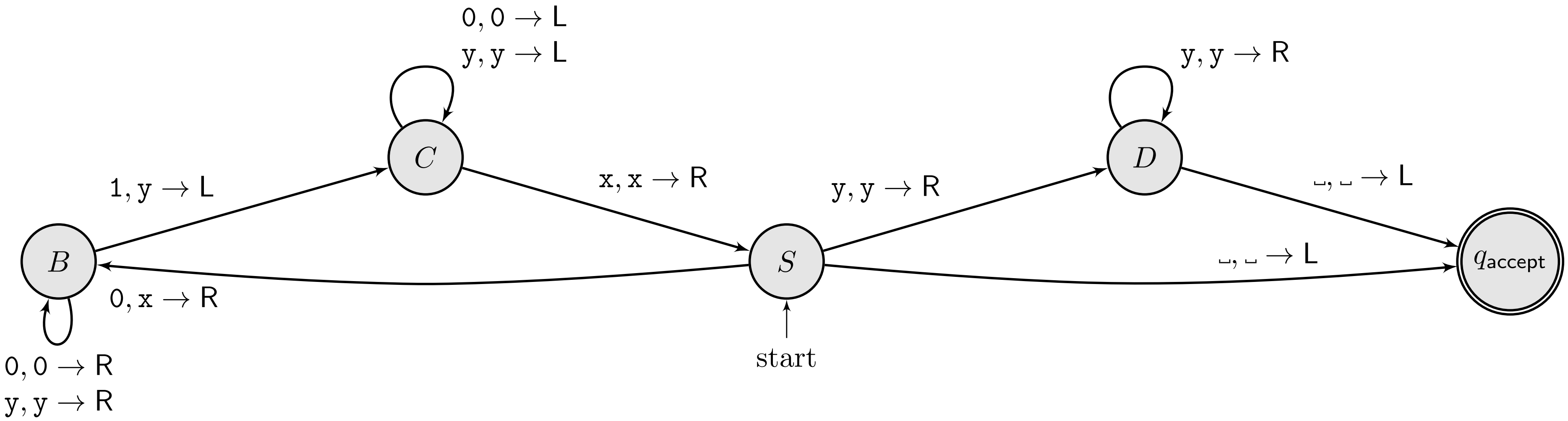


State	Tape
S	0011
B	X011
B	X011
C	X0Y1

State	Tape
C	X0Y1
S	X0Y1
B	XXY1
B	XXY1

State	Tape
C	XXYY
C	XXYY
S	XXYY
D	XXYY

Example 2



State	Tape
S	0011
B	X011
B	X011
C	X0Y1

State	Tape
C	X0Y1
S	X0Y1
B	XXY1
B	XXY1

State	Tape
C	XXYY
C	XXYY
S	XXYY
D	XXYY

State	Tape
D	XXYY␣

Accept!

Simulate

Example 1 configuration

State	Tape	Configuration
S	01110	S 01110
B	x1110	x B 1110
B	xy110	xy B 110
B	xyy10	xyy B 10
B	xyyy0	xyyy B 0
B	xyyyx_	xyyyx B

Configuration history

> The configuration history (sequence of configurations), describes all configurations from the initial state until a current state.

Definition

We say that C_1 **yields** C_2

Example

Configuration history
`S 01110`
`x B 1110`
`xy B 110`
`xyy B 10`
`xyyy B 0`
`xyyyx B`

Acceptance

A Turing machine

- **accepts** a string if there is a configuration history that reaches the `accept` state.
- **rejects** a string if there is a configuration history that reaches the `reject` state.
- **rejects** a string if it never reaches an `accept` or `reject` states
  This means that for any configuration of any length, there is no accept nor a reject state.

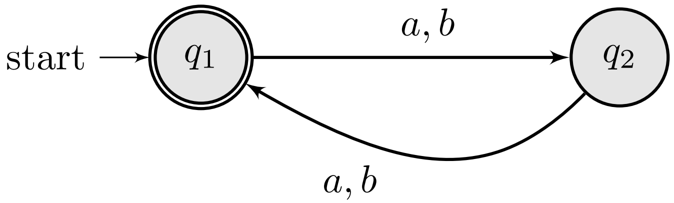
The acceptance algorithm

- **halts** when the machine is in an `accept` or `reject` state
  This is different than NFAs/PDAs which can enter and leave the accept state.

Exercise

Give a Turing Machine that recognizes words of an even length.

NFA



TM that recognizes words of an even length

(online)

Exercise

What language does this TM recognize? (online)

The Church-Turing thesis

Alan Turing and the Turing Machine

- No computers at the time (1936)
- Alan Turing was researching into the foundations of mathematics
- Original intent: capture all possible processes which can be carried out in computing a number†
- What about non-numerical problems?
- How do Turing machines capture all general and effective procedures which determine whether something is the case or not.

Section 3.3

---

†: Devise an algorithm that tests whether a polynomial has an integral root.

The Church-Turing thesis

- Any algorithm can be represented by an equivalent Turing machine
- A problem is computable if, and only if, there exists a Turing Machine that recognizes it.
- Turing Machines are equivalent to λ-terms

The Universal Turing Machine

Or, How do we study the limits of computability

The Universal Turing Machine

A Turing Machine that is capable of simulating any other Turing Machine

- Let U be a TM.
- Given some TM M and some input w, we can encoded as an input string, which we represent as $\langle M, w\rangle$

$$U \text{ accepts } \langle M, w\rangle \text{ iff } M \text{ accepts } w$$

Note that the Universal TM is a regular TM. This computability model is expressive enough to simulate itself.

Alan Turing's impact on modern computers

- Modern computers: von Neumann's EDVAC design
- Fundamental idea of the EDVAC design: stored-programs
  Manipulation of programs as data
- **Universal Turing Machines pioneer the idea of stored programs**

> TM are used to reflect on the limits and potentials of general-purpose computers by engineers, mathematicians and logicians (Module 3)

A single machine simulates all possible machine designs!

Without this idea, computers would have limited scope.

Multi-tape Turing Machine

The TM tape only grows to the right

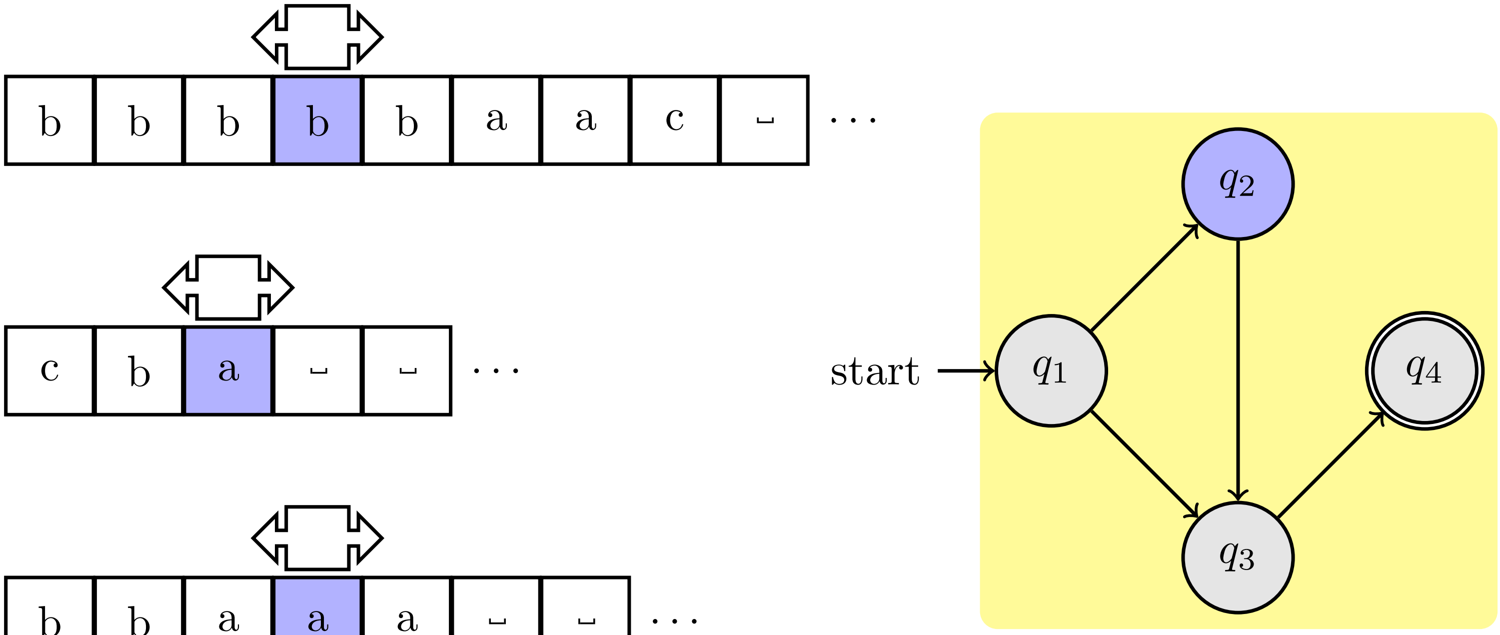
- An important thing to note is that TMs have a tape that grows only to the right
- In `turingmachine.io`, the tape actually grows both ways

Are Turing machines that grow both sides more expressive?

Generalizing, are TM with multi-tapes more expressive?

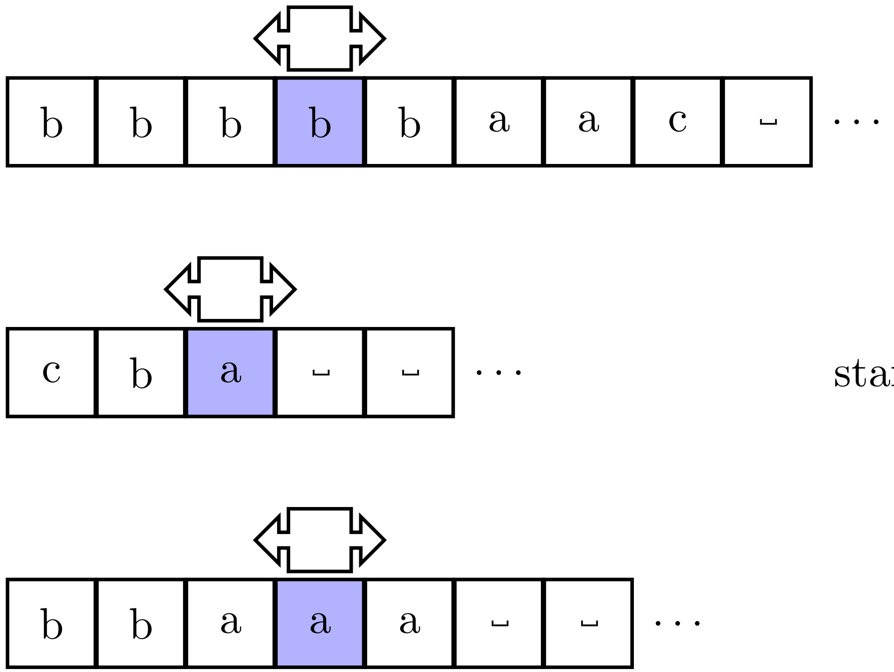
Multi-tape Turing Machine

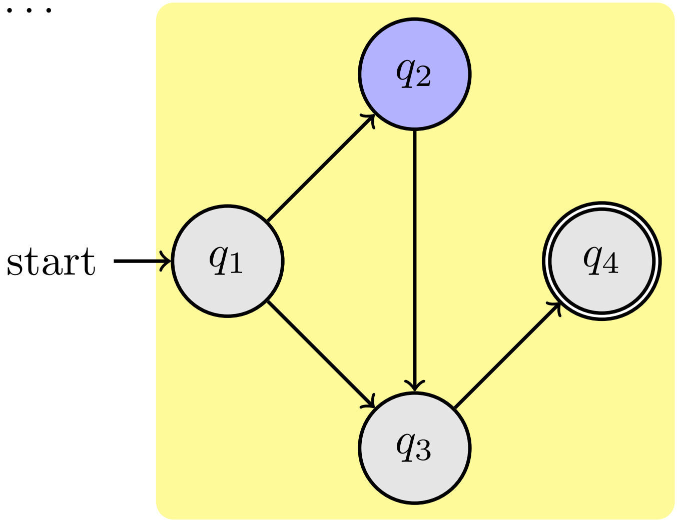
- A variation of the Turing Machine with multiple tapes
- The control may issue each head to move: forward, backward, or skip



Multihead Turing Machine

Are Turing Machines less expressive than Multitape Turing Machines?





Multihead Turing Machine

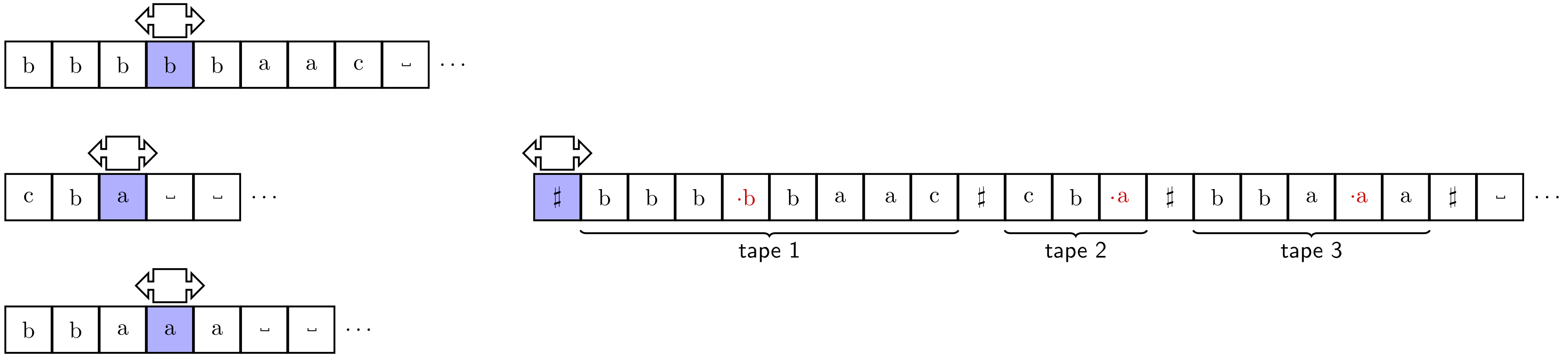
Turing Machines $\iff$ Multitape Turing Machines

- ($\Leftarrow$) Multitape Turing Machines trivially recognize the same language as Turing Machine (let the number of tapes be 1)
- ($\Rightarrow$) How can a single tape encode multitape?

Simulating a multitape

Tape encoding

- Concatenate the three tapes together
- Delimit each tape with a character that is not in the alphabet $\sharp$
- "Tag" the character to encode each tape head (virtual heads), eg $\cdot a$
- The tape head always sits in the beginning of the tape



Simulating a multitape

Operation

- To move the i-th head, read the tape from the beginning until you read $\sharp$ a total of i times and then seek until you find the marked character
- If the virtual head i hits the end of the tape $\sharp$, then shift the rest of the tape to the right and insert a blank character $\sqcup$

Nondeterministic Turing Machines

Nondeterministic Turing Machines (NTM)

A machine can follow more than one transitions for the same input:

$$\delta\colon Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{\mathsf{L}, \mathsf{R}\})$$

Consequence

> Deterministic can only have one outgoing edge ***per*** character read, a nondeterministic machine can have multiple edges

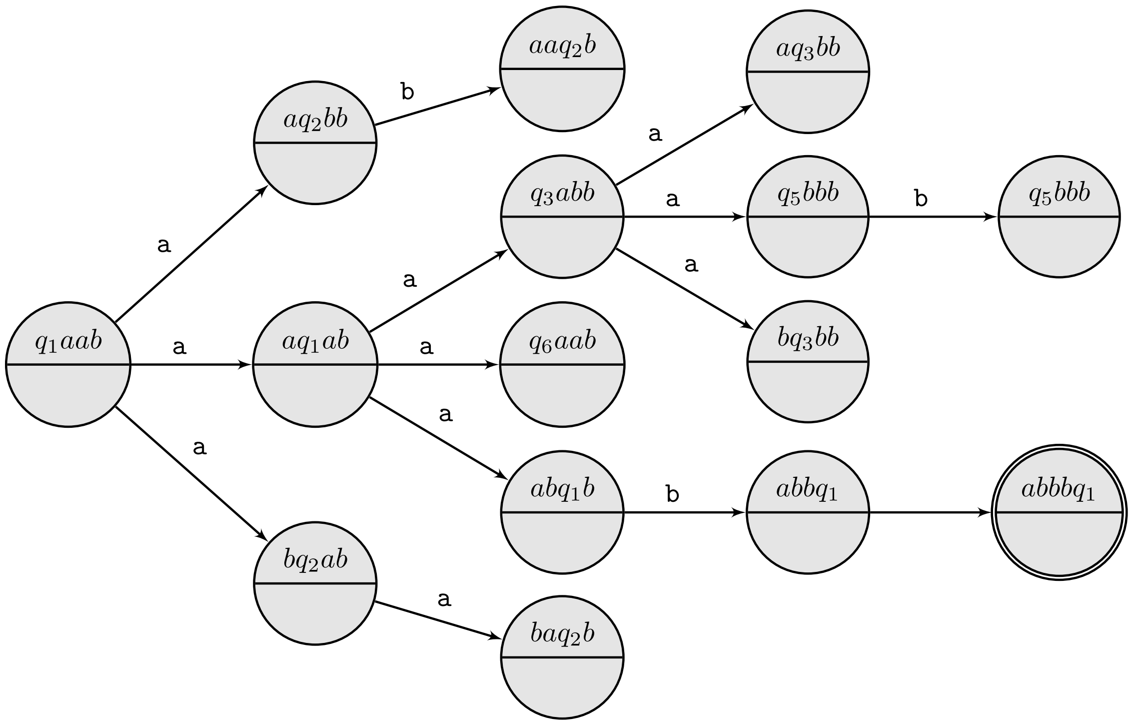
Configurations in a TM

- In a deterministic TM, a configuration history is ***linear***

`abc q1 aac → abcx q2 ac → abcxa q2 c → abcx q2 ay`

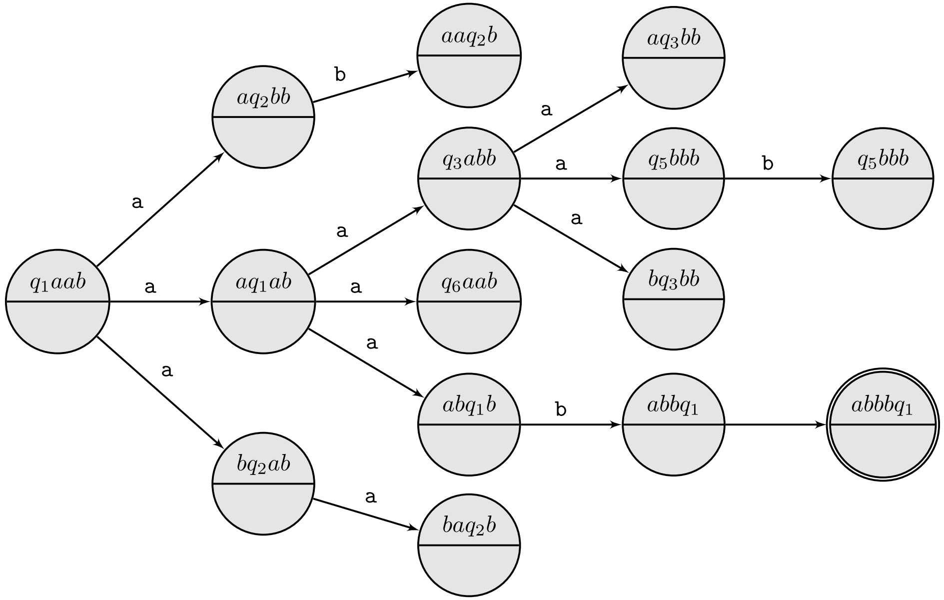
Configurations in a NTM

- In a nondeterministic TM, a configuration history is a **tree**!



Nondeterministic Turing Machines

- **Accept:** when **any** branch reaches q_{start}
- **Reject:** when **all** branches reach q_{reject}
- To find a single acceptance state we need to search the computation tree



Are Turing Machines less expressive than Nondeterministic Turing Machines?

TM $\iff$ NTM

- Given an NTM, say N we show how to construct a TM, say D
- If N accepts on any branch, then D halts and accepts
- If N rejects on every branch, then D halts and rejects

Intuition

> Simulate all branches of the computation; search for any node with an accept state.

Attention!

Question: If we are searching a search tree, and there may exist infinite branches (due to loops), how should we search the tree: DFS or BFS?

TM $\iff$ NTM

- Given an NTM, say N we show how to construct a TM, say D
- If N accepts on any branch, then D halts and accepts
- If N rejects on every branch, then D halts and rejects

Intuition

> Simulate all branches of the computation; search for any node with an accept state.

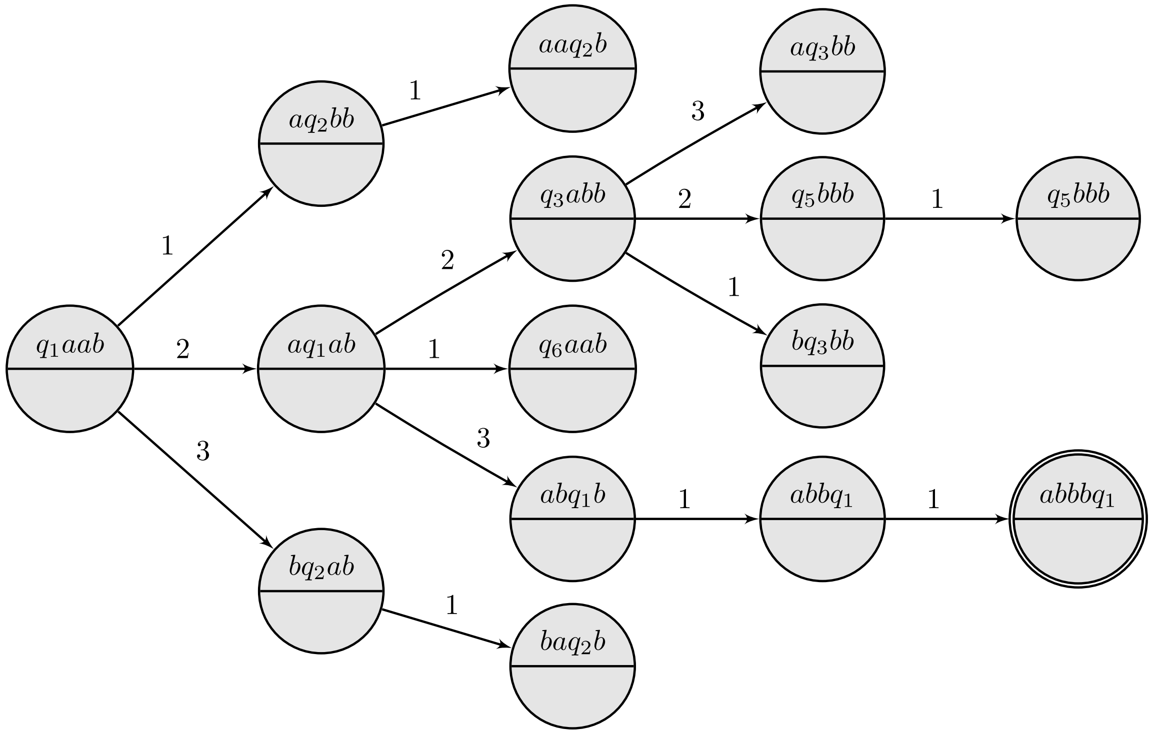
Attention!

Question: If we are searching a search tree, and there may exist infinite branches (due to loops), how should we search the tree: DFS or BFS?

Bread-First Search will ensure our search is not caught in a never-ending branch.

Addressing configuration history

- We can use a sequence of numbers to uniquely identify each node of the configuration history



Unique paths

- 11
- 223
- 2221
- 221
- 21
- 2311
- 31

Using a TM to simulate a NTM

Use 3 tapes

1. **Initial input:** One tape for the input
2. **Simulation tape:** Where we will be executing an address
3. **Address tape:** An ever growing number that uniquely identifies where we are in the tree

How many choices at each step?

1. Copy tape 1 to tape 2
2. Simulate TM with address from the address tape; if it reaches an accepted state, then ACCEPT, otherwise continue
3. Increment address (next BFS-wise) and go to 1