Turing Machines

IIC3242

Computational Complexity

Objective: Measure computational complexity of a problem.

That is: We want to measure computational resources needed to solve a problem.

- Time
- Space
- ...

But first, let us define what a problem is.

Decision problems

Alphabet Σ: a finite set of symbols.

- Example: $\Sigma = \{0, 1\}$

Word w: a finite sequence of symbols from Σ.

- Example: $w = 01101$

Σ^*: The set of all words over the alphabet Σ.

Language L: a set of words.

- Example: $L = \{0^n 1^n \mid n \in \mathbb{N}\}$

Decision problems

Decision problem associated to a language L: Given $w \in \Sigma^*$, decide if $w \in L$ (or $w \notin L$).

Example

We can view SAT as a decision problem. Assume that $P = \{p, q\}$:

- $\Sigma = \{p, q, \neg, \wedge, \vee, \rightarrow, \leftrightarrow, (,)\}$

  Some words in Σ^* represent formulas, while others like $\neg\neg$ and $p\neg q \wedge \wedge \vee q$ do not.

- SAT $= \{w \in \Sigma^* \mid w$ represents a formula and w is satisfiable$\}$

Complexity of a problem

The complexity of a language L is the complexity of a decision problem associated to L.

When can we say that L can be solved efficiently?

- When there is an efficient algorithm that decides L

Exercise

Show that $L = \{w \in \{0,1\}^* \mid w \text{ is a palindrome}\}$ can be solved efficiently.

When can we say that L is an (intrinsically) difficult problem?

- When **there is no** algorithm that decides L efficiently.

What is an algorithm?

How can we show that a problem is difficult?

- To do this we need to formalize the notion of an algorithm

What is an algorithm? Can we define this formally?

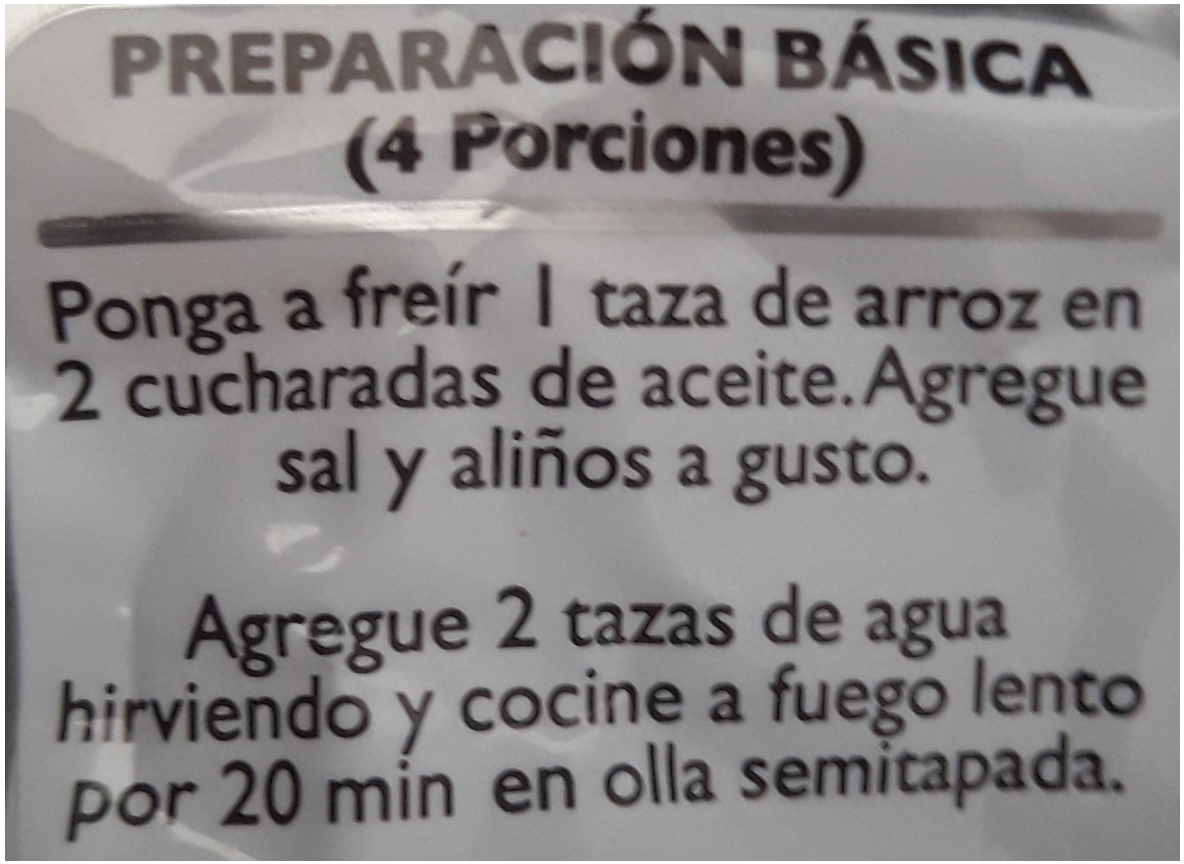
What is an algorithm?

Maybe this?

```
def fibR(n):
 if n==1 or n==2:
  return 1
 return fibR(n-1)+fibR(n-2)
print fibR(5)
```

What is an algorithm?

Or this?



What is an algorithm?

What is an algorithm? Can we define this formally?

- Turing machines: An intent to formalize this concept

Is it possible to show that Turing machines capture the notion of an algorithm?

- No way. Algorithm is an intuitive concept.

Turing machines

Why do we believe that Turing machines are a good formalisation of the notion of an algorithm?

Turing machines

Why do we believe that Turing machines are a good formalisation of the notion of an algorithm?

- All programs of a Turing machine can be implemented

Turing machines

Why do we believe that Turing machines are a good formalisation of the notion of an algorithm?

- All programs of a Turing machine can be implemented
- All known algorithms can be implemented on a Turing machine

Turing machines

Why do we believe that Turing machines are a good formalisation of the notion of an algorithm?

- All programs of a Turing machine can be implemented
- All known algorithms can be implemented on a Turing machine
- All other attempts to formalise the concept are reducible to Turing machines

Turing machines

Why do we believe that Turing machines are a good formalisation of the notion of an algorithm?

- All programs of a Turing machine can be implemented
- All known algorithms can be implemented on a Turing machine
- All other attempts to formalise the concept are reducible to Turing machines
  - The best attempts are equivalent to Turing machines
  - All "reasonable" formalisations are efficiently inter-reducible.

Turing machines

Why do we believe that Turing machines are a good formalisation of the notion of an algorithm?

- All programs of a Turing machine can be implemented
- All known algorithms can be implemented on a Turing machine
- All other attempts to formalise the concept are reducible to Turing machines
  - The best attempts are equivalent to Turing machines
  - All "reasonable" formalisations are efficiently inter-reducible.
- Church-Turing thesis: **Algorithm = Turing machine**

Turing machines: the idea

- An infinite tape to read from and write to
- A read/write head that can move along the tape
- Accept/reject state to tell us if the word belongs to the language or not

Note that the machine can go on forever

Definition of a Turing machine

Definition

A (deterministic) Turing machine: $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$

- Q is a finite set of states
- Σ is the input alphabet, where $\vdash, \mathtt{B} \notin \Sigma$
- Γ is the tape alphabet, where $\Sigma \cup \{\vdash, \mathtt{B}\} \subseteq \Gamma$
- $q_0 \in Q$ is the initial state
- $q_{accept} \in Q$ is the accepting state
- $q_{reject} \in Q$ is the rejecting state ($q_{accept} \neq q_{reject}$)
- δ is a partial function:

$$\delta \quad : \quad Q \times \Gamma \to Q \times \Gamma \times \{\leftarrow, \square, \rightarrow\}$$

δ is called the transition function

Computation of a Turing machine

The tape of a Turing machine is unbounded to the right

- The symbol $\vdash$ marks the position 0 (leftmost position) of the tape

We assume that:

- If $\delta(q, \vdash)$ is defined: $\delta(q, \vdash) = (q', \vdash, X)$, with $X \in \{\rightarrow, \square\}$
- If $a \in (\Gamma \backslash \{\vdash\})$ and $\delta(q, a)$ is defined: $\delta(q, a) = (q', b, X)$, with $b \in (\Gamma \backslash \{\vdash\})$

Computation of a Turing machine

Σ is the input alphabet and Γ is the tape alphabet.

- An input $w \in \Sigma^*$ of length n is placed in the positions 1, ..., n of the tape
- The positions $(n+1, n+2, \ldots)$ contain the symbol $\mathtt{B}$

Computation of a Turing machine: configurations

The machine always starts in the state q_0 and with the read/write head over the position 1 of the tape.

As the machine computes, changes occur in the:

- Current state of the machine
- Current position of the head
- The content of the tape

A setting of these three components is called the configuration of the machine.

Computation of a Turing machine: representing configurations

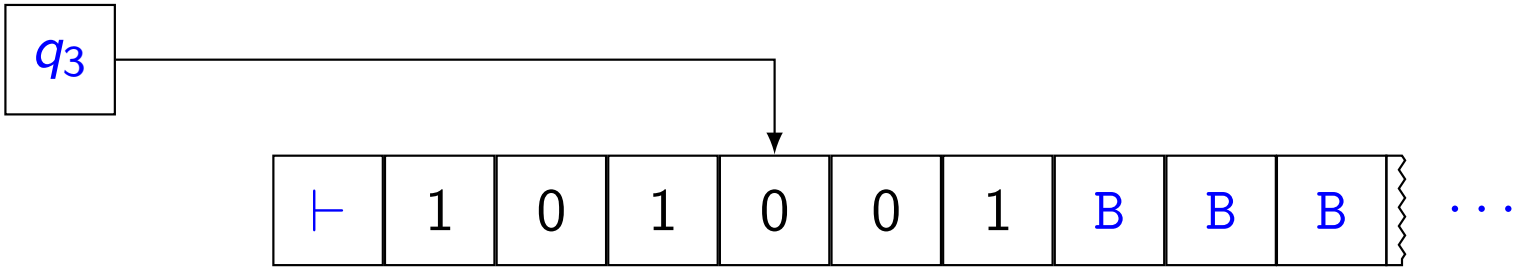
Assume that:

- The current state of the machine is q
- The tape content is equal to uv, with u and v strings over Γ (and all symbols after the last letter of v are $\mathtt{B}$)
- The machine head is located over the first symbol of v

This configuration is then represented as $C = uqv$

For instance

If we have the following situation:



Then this configuration is described using $u = \vdash 101$ and $v = 001$

Thus resulting in a configuration $C = \vdash 101q_3001$

How does the computation work?

A configuration C_1 **yields** a configuration C_2:

We have $a, b, c \in \Gamma$, and $u, v \in \Gamma^*$ and $q_i, q_j \in Q$

uaq_ibv yields uq_jacv, if we have $\delta(q_i, b) = (q_j, c, \leftarrow)$;

uaq_ibv yields $uacq_jv$, if we have $\delta(q_i, b) = (q_j, c, \rightarrow)$;

uaq_ibv yields uaq_jcv, if we have $\delta(q_i, b) = (q_j, c, \square)$.

How does the computation work?

The **start configuration** on input w is always $q_0 w$

In **accepting configuration** the state is q_{accept}

In **rejecting configuration** the state is q_{reject}

The states q_{accept} and q_{reject} are **halting states**

Halting states do not yield any further configurations

Accepting a string

A Turing machine M **accepts** an input w if:

There is a sequence of configurations $C_1, C_2, \ldots, C_k$ such that:

1. C_1 is the start configuration of M on w,
2. C_i yields C_{i+1} and
3. C_k is an accepting configuration.

Definition

The language accepted by the Turing machine M:

$$L(M) = \{w \in \Sigma^* \mid M \text{ accepts } w\}.$$

Is this enough?

A language is called **Turing recognizable (recursively enumerable)** if it is accepted by some Turing machine.

On an input a Turing machine can: *accept*, *reject*, or **loop** (i.e. run forever)

When a machine loops this is not very useful for deciding if string is in the language or not

Namely, this does not give us an algorithm

First notion of algorithm = Turing machine that halts on every input

Deciders

A Turing machine is a **decider** if it halts on every input

A language is **Turing-decidable (recursive)** if it is accepted by some decider

Remark

From now on all the Turing machines we consider are going to be deciders!

Example of a Turing machine

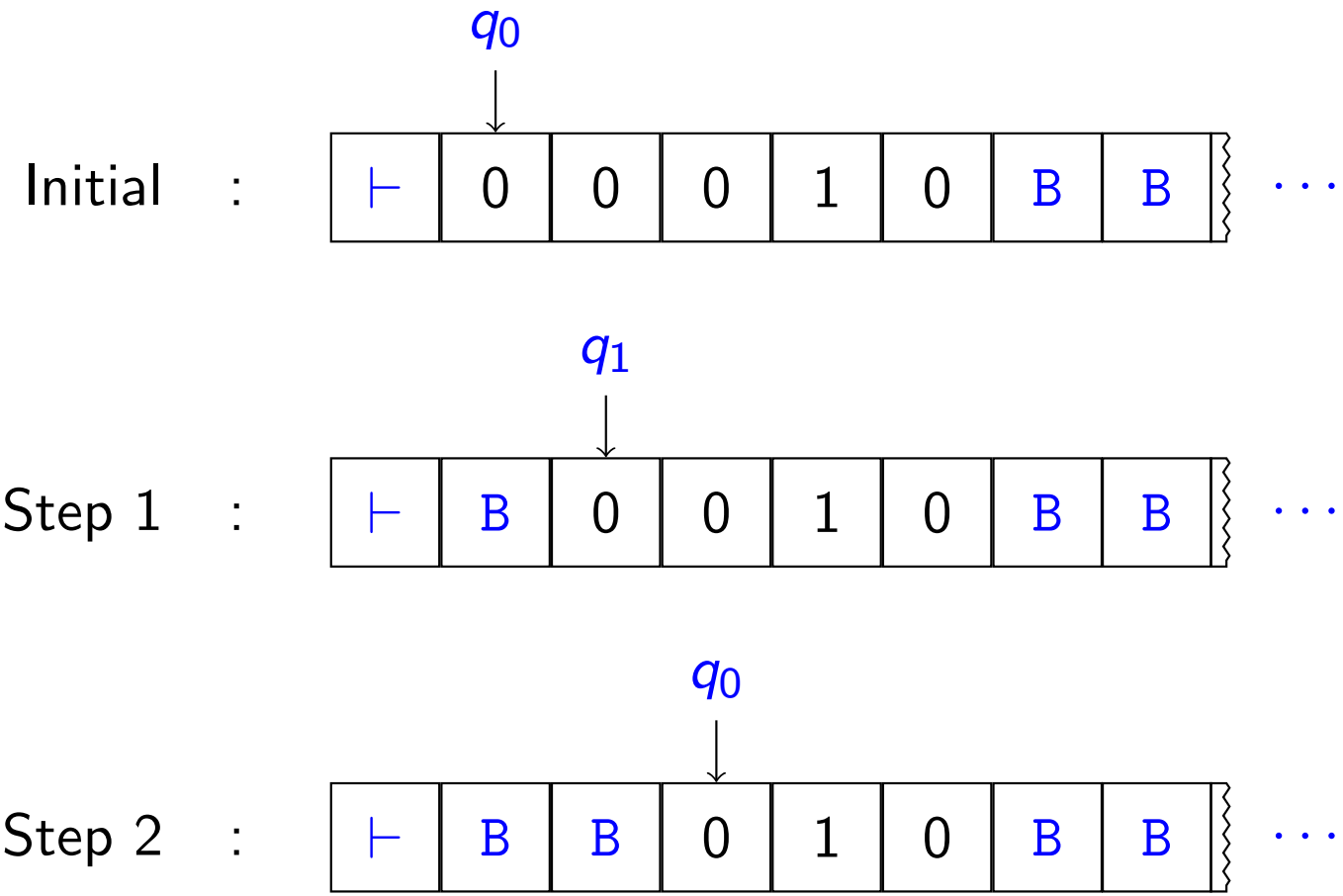
We will construct a Turing machine that checks if the number of 0s in a word is even: $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$

- $Q = \{q_0, q_1, q_{accept}, q_{reject}\}$
- $\Sigma = \{0, 1\}$
- $\Gamma = \{0, 1, \vdash, \mathtt{B}\}$
- δ is defined as follows:

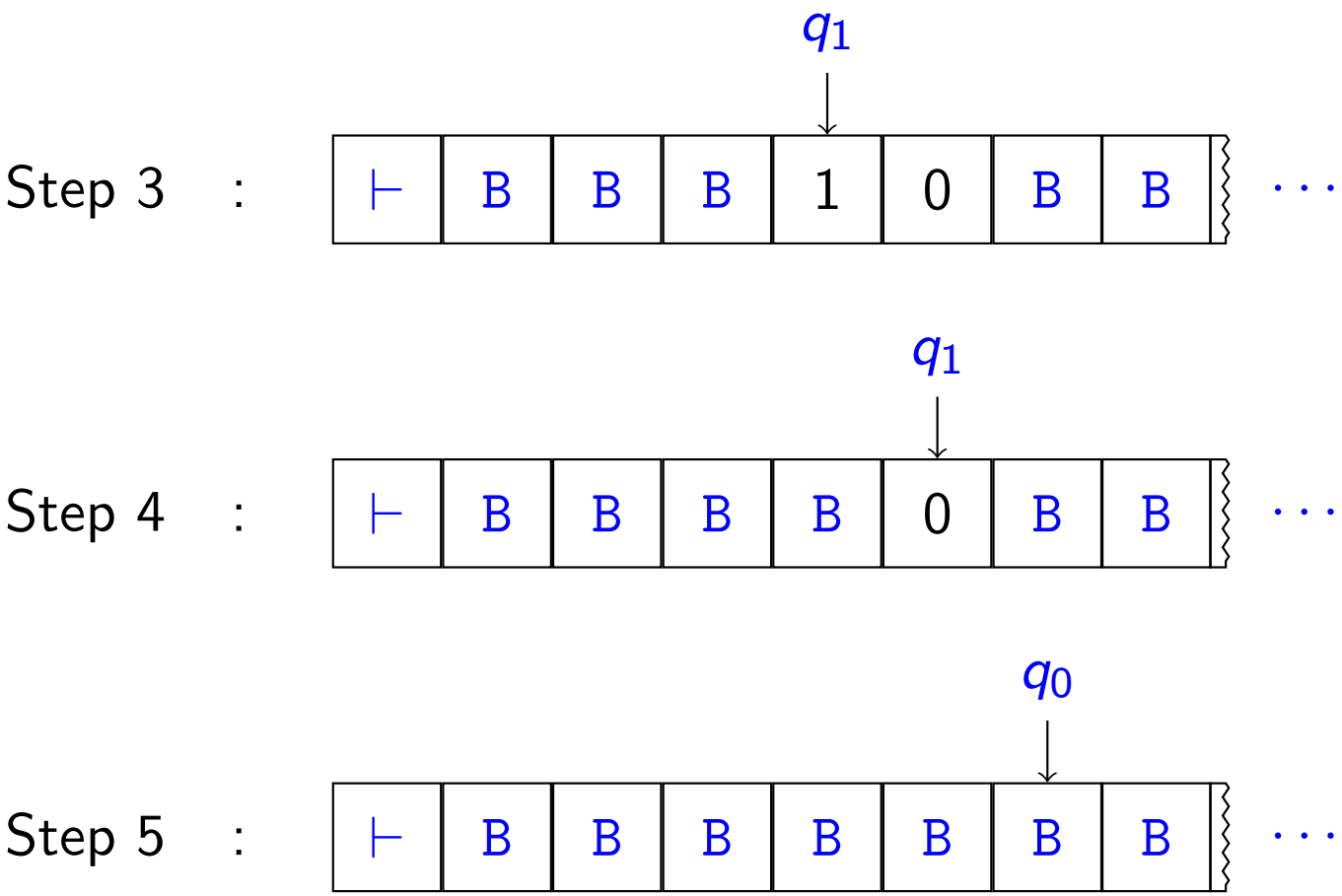
$$
\begin{aligned}
\delta(q_0, 0) &= (q_1, \mathtt{B}, \rightarrow) \\
\delta(q_0, 1) &= (q_0, \mathtt{B}, \rightarrow) \\
\delta(q_1, 0) &= (q_0, \mathtt{B}, \rightarrow) \\
\delta(q_1, 1) &= (q_1, \mathtt{B}, \rightarrow) \\
\delta(q_0, \mathtt{B}) &= (q_{accept}, \mathtt{B}, \rightarrow) \\
\delta(q_1, \mathtt{B}) &= (q_{reject}, \mathtt{B}, \rightarrow)
\end{aligned}
$$

Run of this machine

Assume that $w = 00010$:



Run of this machine



In the next step we go to q_{accept}

Conclusion: The machine accepts $w = 00010$

Language accepted by a TM

Example

For the machine M of previous three slides:

$$L(M) = \{w \in \{0,1\}^* \mid w \text{ has an even number of } 0\}$$

Language accepted by a TM

Example

For the machine M of previous three slides:

$$L(M) = \{w \in \{0,1\}^* \mid w \text{ has an even number of } 0\}$$

Exercise

Construct a Turing machine that decides the language

$$L = \{w \in \{0,1\}^* \mid w \text{ is a palindrome}\}.$$

Complexity of an algorithm

Algorithm = TM that halts on every input

How do we measure the time taken by the algorithm?

For a TM with alphabet Σ:

- Step of M: Execute one instruction of the transition function (one configuration yield)
- $time_M(w)$: Number of steps that M executes on input $w \in \Sigma^*$

Complexity of an algorithm

Definition

The **time complexity** of a TM M in the worst case is defined by the function t_M:

$$t_M(n) \quad = \quad \max\{\ time_M(w) \ \mid \ w \in \Sigma^* \text{ and } |w| = n\ \}.$$

Exercise

Construct a TM with time complexity $O(n^2)$ that decides the language

$$L = \{w \in \{0,1\}^* \mid w \text{ is a palindrome}\}.$$

The model does not matter: TM with multiple tapes

Definition

TM (deterministic) with k tapes: $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$

- Q is a finite set of states
- Σ is the input alphabet, with $\vdash, \mathtt{B} \notin \Sigma$
- Γ is the tape alphabet, with $\Sigma \cup \{\vdash, \mathtt{B}\} \subseteq \Gamma$
- $q_0 \in Q$ is the initial state
- $q_{accept}, q_{reject} \in Q$ are the accepting and rejecting state
- δ is a partial function:

$$\delta \quad : \quad Q \times \Gamma^k \to Q \times \Gamma^k \times \{\leftarrow, \square, \rightarrow\}^k.$$

δ is called the transition function.

Multitape TM: how do they work

The machine has k tapes infinite to the right.

- We use $\vdash$ to mark the position 0 of each tape
- No transition can rewrite this symbol, or move left from it

Σ is the input alphabet and Γ is the alphabet of the tapes.

- An input $w \in \Sigma^*$ of length n is placed in positions 1, ..., n of the first tape
- The positions $(n+1, n+2, \ldots)$ of the first tape have $\mathtt{B}$
- The remaining tapes have $\mathtt{B}$ in positions 1, 2, 3, ...

Multitape TM: how do they work

The machine has one read/write head per tape.

- At start the state of the machine is q_0, and all the heads are over the position 1 of their tape

The transition of the form

$$\delta(q, a_1, \ldots, a_k) = (q', b_1, \ldots, b_k, X_1, \ldots, X_k)$$

means that:

- The machine is in the state q,
- The heads 1 through k are reading symbols a_1 through a_k,
- The state changes to q',
- The machine writes b_1 to tape 1, b_2 to tape 2, etc.
- The head i moves as indicated by X_i

Multitape TM: configurations

Configurations are defined as before, but now we have k tapes

Thus we will need k pairs $u_i, v_i \in \Gamma^*$ to describe their content

The symbol $\#$ is used to denote the start of the tape

If the machine is in the state q and the tape contents are $u_i v_i$ a configuration is:

$$\#u_1 q v_1 \#u_2 q v_2 \# \cdots \#u_k q v_k,$$

where each head is over the first symbol of v_i

Yield relation and acceptance are defined as for single-tape machines

Multitape TM: complexity

As before:

$$L(M) = \{w \in \Sigma^* \mid M \text{ accepts } w\}.$$

(Recall that M is a decider for us.)

For a k-tape machine M with alphabet Σ:

- Step of M: Execute an instruction of the transition function (change configuration)
- $time_M(w)$: Number of steps M takes on input $w \in \Sigma^*$
- Worst case complexity of M:

$$t_M(n) \quad = \quad \max\{\ time_M(w) \ \mid \ w \in \Sigma^* \text{ and } |w| = n\ \}$$

Multitape TM: example

Example

There is a 2-tape TM M that runs in time $O(n)$ and accepts the language $L = \{w \in \{0,1\}^* \mid w \text{ is a palindrome } \}$.

Multitape TM: example

Example

There is a 2-tape TM M that runs in time $O(n)$ and accepts the language $L = \{w \in \{0,1\}^* \mid w \text{ is a palindrome } \}$.

Solution: Our machine M works as follows:

1. First it copies the input from tape 1 to the tape 2
2. Then it moves the head of the first tape to the beginning of the input (the second head stays at the last symbol of the second tape=input)
3. We then move one step forwards by the first head and one backwards by the second
4. If the symbols are different reject, otherwise repeat step 3
5. If we come to the end of the input accept

Multitape TM: example

Exercise

Define the TM from the previous slide formally. That is, describe its states and the transition function.

Hint: Use different states to know which phase of the algorithm you are in.

Did we get any power?

A language L is decided by a TM M if $L = L(M)$.

- Can we decide more languages with additional tapes?

Did we get any power?

A language L is decided by a TM M if $L = L(M)$.

- Can we decide more languages with additional tapes?

Theorem

If a language L is decided by a k-tape TM M, then it is also decided by a single tape TM S.

Simulating multitape machine with a single tape

Proof: k tapes of M are stored in the single tape of S

We use $\#$ to separate the content of different tapes

To know the head position, S will use $\tilde{}$ over the tape symbol

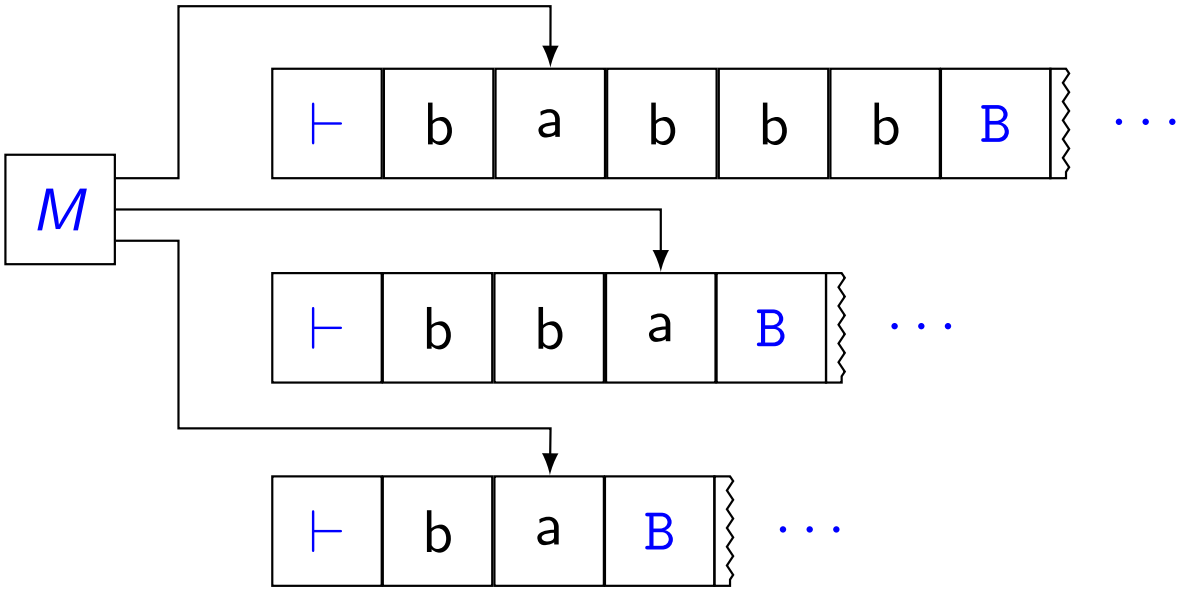
E.g. if head is over a we will use $\tilde{a}$ to denote this

That is, S has $\tilde{a}$ in its tape alphabet, for a in tape alphabet of M

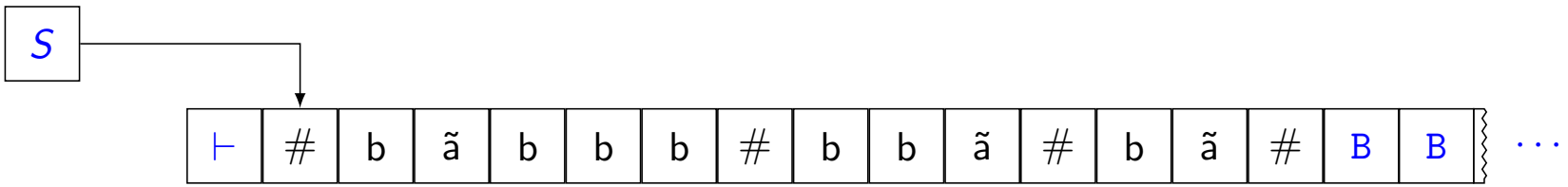
Think of this as virtual tapes and heads

We use $\#$ instead of $\vdash$ to denote the first symbol of each tape

Simulating multitape machine with a single tape



Is the same as:



Simulating multitape machine with a single tape

$S =$ on input $w = w_1 \cdots w_n$:

1. Simulate the initial configuration of M on w:

$$\vdash \#\tilde{w}_1 w_2 \cdots w_n \#\tilde{\mathtt{B}}\#\tilde{\mathtt{B}} \cdots \#\tilde{\mathtt{B}}\#$$

2. To simulate a move of M our machine S scans the tape in order to see where each head is. Then S does a second pass to update the tapes according to the transition function of M.
3. If a virtual head moves *right* to $\#$ this means that M moves its head to previously unread portion of the corresponding tape (a $\mathtt{B}$). So S writes $\mathtt{B}$ and shifts the content of its tape from this position to the end one cell towards right.

□

Complexity of different models

A language L is decided by a TM M in time $O(t(n))$ if $L = L(M)$ and $t_M(n)$ is $O(t(n))$.

- The definition is identical for $\Omega(t(n))$ and $o(t(n))$

This definition considers the TM to have k tapes, with $k \geq 1$.

Notation reminder

- **Big-O ($\leq$)**: $f(n) = O(g(n))$ if:

$$\exists c, n_0 \text{ s.t. } \forall n \geq n_0 \; f(n) \leq c \cdot g(n)$$

- **Small-o ($<$)**: $f(n) = o(g(n))$ if:

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0; \text{ that is,}$$

$$\forall c > 0 \exists n_0 \text{ s.t. } \forall \; n \geq n_0 f(n) < c \cdot g(n)$$

Notation reminder

- **Big-Omega ($\geq$)**: $f(n) = \Omega(g(n))$ if:

$$g(n) = O(f(n)), \text{ that is}$$

$$\exists c, n_0 \text{ s.t. } \forall n \geq n_0 \; g(n) \leq c \cdot f(n), \text{ that is}$$

$$\exists d, n_0 \text{ s.t. } \forall n \geq n_0 \; f(n) \geq d \cdot g(n)$$

Complexity of different models

Is the complexity smaller when we use more than one tape?

Complexity of different models

Is the complexity smaller when we use more than one tape?

Theorem

If a language L is decided by a TM M with k tapes $(k \geq 2)$ in time $O(t(n))$, where $t(n) \geq n$, then L is decided by a single-tape TM S in time $O(t(n)^2)$.

Complexity of different models

Is the complexity smaller when we use more than one tape?

Theorem

If a language L is decided by a TM M with k tapes $(k \geq 2)$ in time $O(t(n))$, where $t(n) \geq n$, then L is decided by a single-tape TM S in time $O(t(n)^2)$.

Exercise

Prove the theorem (see previous slides).

- Is it possible to reduce the complexity difference between M and S?

No, not really!

Let $L = \{w \in \{0, 1, \#\}^* \mid w \text{ is a palindrome}\}$.

- L is decided by a 2-tape TM in time $O(n)$
- Can L be decided in linear time by a one-tape TM?

No, not really!

Let $L = \{w \in \{0,1,\#\}^* \mid w \text{ is a palindrome}\}$.

- L is decided by a 2-tape TM in time $O(n)$
- Can L be decided in linear time by a one-tape TM?

Proposition

Let M be a one-tape TM. If $L = L(M)$, then M runs in time $\Omega(n^2)$.

No, not really!

Let $L = \{w \in \{0,1,\#\}^* \mid w \text{ is a palindrome}\}$.

- L is decided by a 2-tape TM in time $O(n)$
- Can L be decided in linear time by a one-tape TM?

Proposition

Let M be a one-tape TM. If $L = L(M)$, then M runs in time $\Omega(n^2)$.

Proof: Assume that $L = L(M)$, with M a one-tape TM.

- Let Q be the set of states of M

Complexity of different models

Wlog, assume that M always reads the entire input word.

- Why can we assume this?

For $w \in \{0, 1, \#\}^*$, let w^r be w written in reverse.

Define L_n as the following language ($n > 0$ and divisible by 4):

$$L_n \quad = \quad \{w\#^{\frac{n}{2}} w^r \mid w \in \{0,1\}^{\frac{n}{4}}\}.$$

Clearly $L_n \subseteq L$.

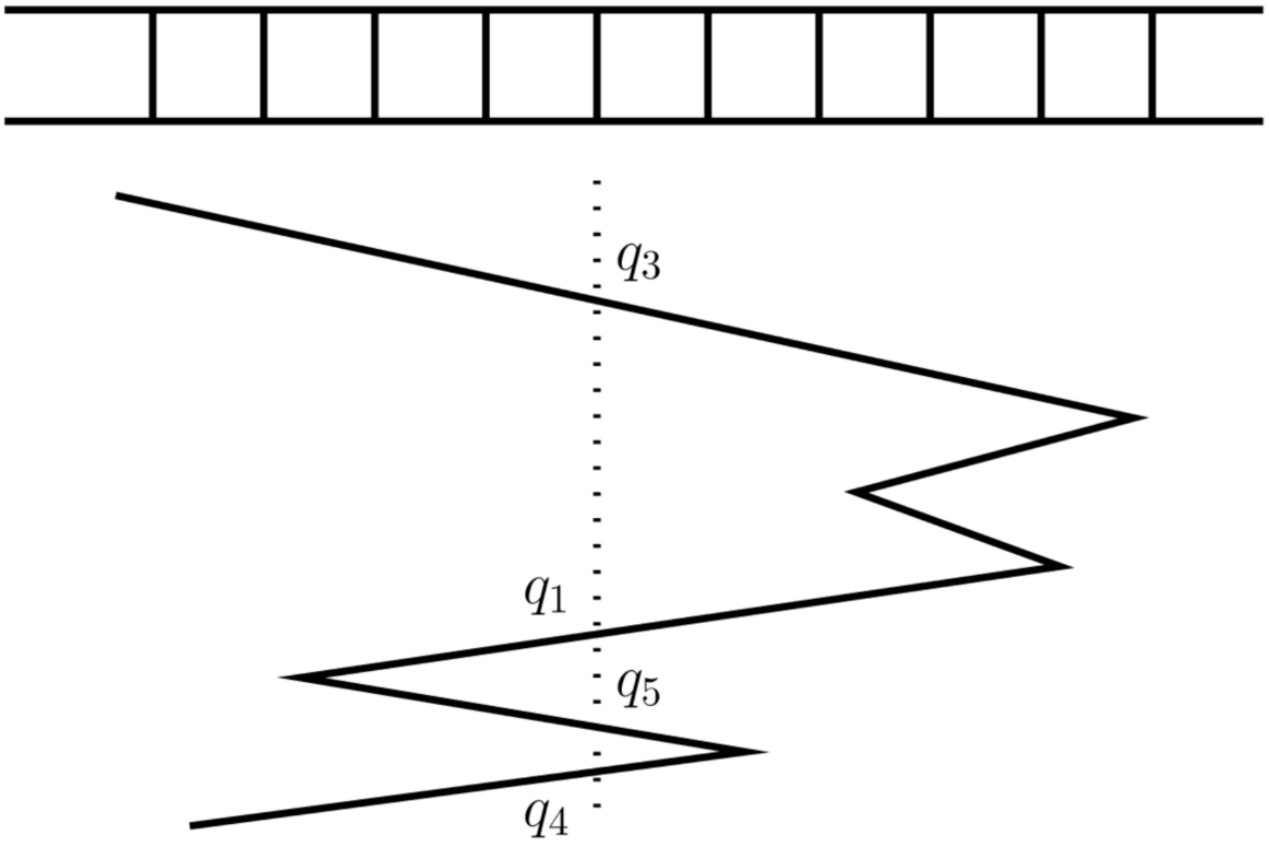
Crossing sequences

Let $w \in L_n$ and $\frac{n}{4} \leq i \leq \frac{3n}{4}$. Denote by $C_i(w)$ the sequence of states $[q_1, \ldots, q_k]$ that M is in while passing the line between the position i and $i+1$ (in either direction) while running with w as the input.

Let $C(w) = \{C_i(w) \mid \frac{n}{4} \leq i \leq \frac{3n}{4}\}$

Crossing sequences

Just to make sure what a crossing sequence is:



A little lemma

Lemma

If $w_1, w_2 \in L_n$ *and* $w_1 \neq w_2$, *then* $C(w_1) \cap C(w_2) = \emptyset$.

Proof: Assume that the lemma is false. Then there are $i, j \in \{\frac{n}{4}, \ldots, \frac{3n}{4}\}$ such that $C_i(w_1) = C_j(w_2)$.

Let u_1 and v_2 be the words consisting of the first i symbols of w_1 and the last $n - j$ symbols of w_2, respectively.

Since $C_i(w_1) = C_j(w_2)$, we have that $u_1 v_2$ is accepted by M.

- How do we prove this?

But $u_1 v_2$ is not a palindrome, thus giving us a contradiction. □

Almost there (don't fall asleep)

For $w \in L_n$, let s_w be the **shortest sequence** in $C(w)$.

- $S_n = \{s_w \mid w \in L_n\}$

From the lemma it follows that $s_{w_1} \neq s_{w_2}$ if $w_1 \neq w_2$.

- Therefore: $|S_n| = |L_n| = 2^{\frac{n}{4}}$

Let m the length of the longest sequence in S_n.

- The number of sequences of length at most m is:

$$\sum_{i=0}^{m} |Q|^i = \frac{|Q|^{m+1} - 1}{|Q| - 1}$$

We finish here

From this we conclude that: $\frac{|Q|^{m+1}-1}{|Q|-1} \geq 2^{\frac{n}{4}}$.

- Why? (Lemma $\Rightarrow$ all the shortest sequences are different)

By taking logarithms we get that m is $\Omega(n)$.

- So there exists $w_0 \in L_n$ such that $|s_{w_0}|$ is $\Omega(n)$.

Therefore: All the sequences in $C(w_0)$ are of length $\Omega(n)$ (since the shortest one is).

Conclusion: With input w_0, the machine M takes time $\Omega(n^2)$.

- Given that M has to generate $\frac{n}{2}$ sequences of states of length $\Omega(n)$ (one for each crossing between $\frac{n}{4}$ and $\frac{3n}{4}$)

Different models: nondeterminism

Definition

Nondeterministic Turing machine: $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$

- Q *is a finite set of states*
- Σ *is the input alphabet, with* $\vdash, \mathtt{B} \notin \Sigma$
- Γ *is the tape alphabet, with* $\Sigma \cup \{\vdash, \mathtt{B}\} \subseteq \Gamma$
- $q_0 \in Q$ *is the initial state*
- $q_{accept}, q_{reject} \in Q$ *are the accepting and rejecting states*
- δ *is a* **relation**:

$$\delta \subseteq Q \times \Gamma \times Q \times \Gamma \times \{\leftarrow, \square, \rightarrow\}$$

Computing with nondeterminism

Initial state, configuration and tape content are same as before

But now we can have a choice where to go next:

Assume that the head is reading a and the state is q:

If $(q, a, q', b, \square)$ and $(q, a, q'', a, \rightarrow)$ are both in δ

- We can write b, change the state to q' and keep the head as it was; *or*
- Keep the a, change the state to q'' and move to the right

Computing with nondeterminism: formally

A configuration C_1 **yields** a configuration C_2:

We have $a, b, c \in \Gamma$, and $u, v \in \Gamma^*$ and $q_i, q_j \in Q$

uaq_ibv yields uq_jacv, if we have $(q_i, b, q_j, c, \leftarrow) \in \delta$;

uaq_ibv yields $uacq_jv$, if we have $(q_i, b, q_j, c, \rightarrow) \in \delta$;

uaq_ibv yields uaq_jcv, if we have $(q_i, b, q_j, c, \square) \in \delta$.

Accepting a string

A nondeterministic TM M **accepts** an input w if:

There is a sequence of configurations $C_1, C_2, \ldots, C_k$ such that:

1. C_1 is the start configuration of M on w,
2. C_i yields C_{i+1} and
3. C_k is an accepting configuration.

Definition

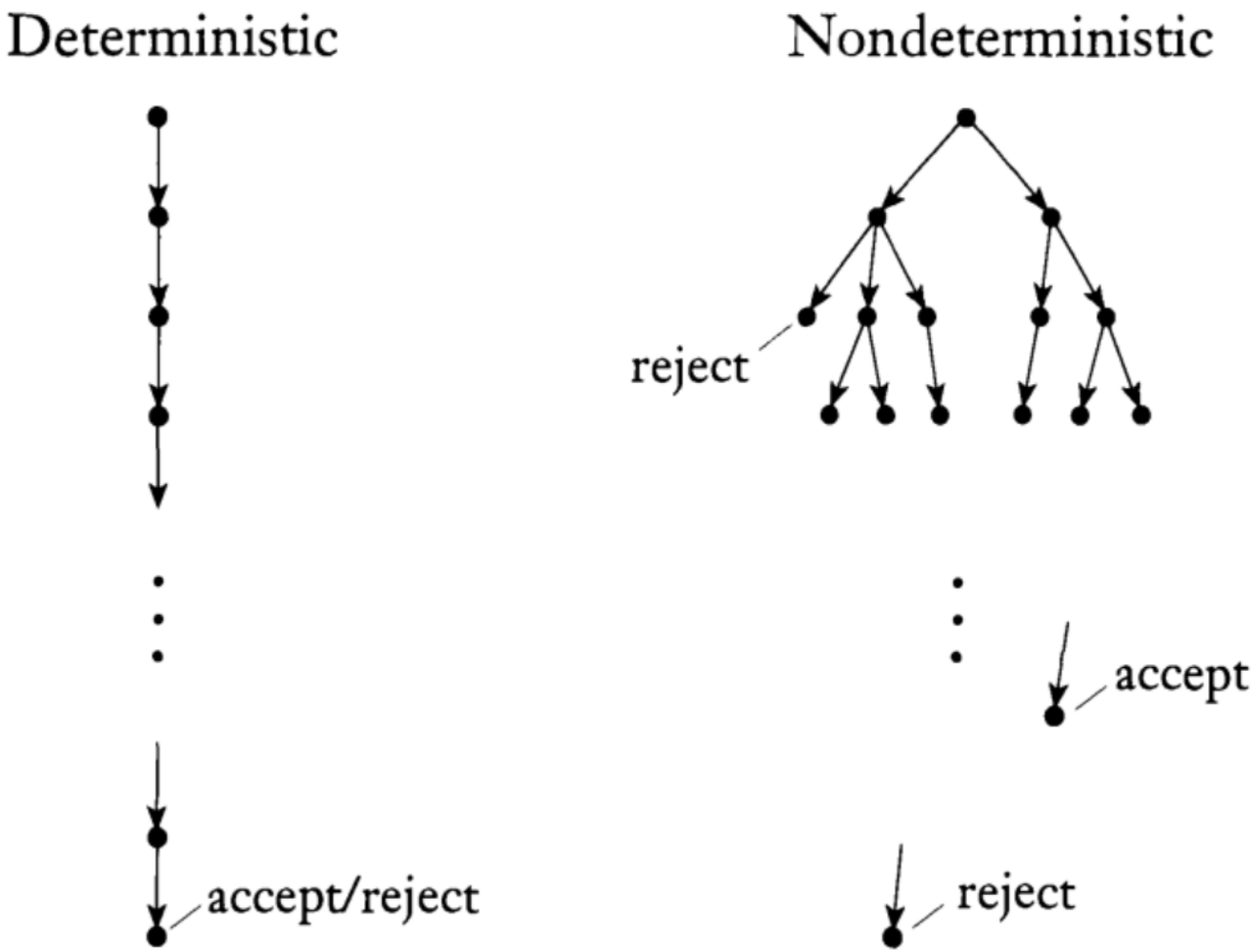
The language accepted by the Turing machine M:

$$L(M) = \{w \in \Sigma^* \mid M \text{ accepts } w\}.$$

Note that some sequences can reject and some accept.

We only need one accepting sequence!

How to think about nondeterminism



Nondeterministic algorithms

Recall that we want to know if a string is or is not in the language.

With nondeterminism it can happen that some branch accepts, but we don't know when this happens.

A nondeterministic TM is a **decider** if all of its branches halt on all inputs.

Nondeterministic algorithm = a nondeterministic TM that is a decider

Complexity of nondeterministic algorithms

Let M be a NTM that is a decider

- Step of M: Execute one instruction of the transition relation (one "change" of configuration)
- $time_M(w)$: Maximum number of steps that M takes on any branch when run with the input w

Definition

The **time complexity** of a NTM M is defined by the function t_M:

$$t_M(n) \quad = \quad \max\{\ time_M(w) \ \mid \ w \in \Sigma^* \text{ and } |w| = n\ \}.$$

Complexity of nondeterministic algorithms

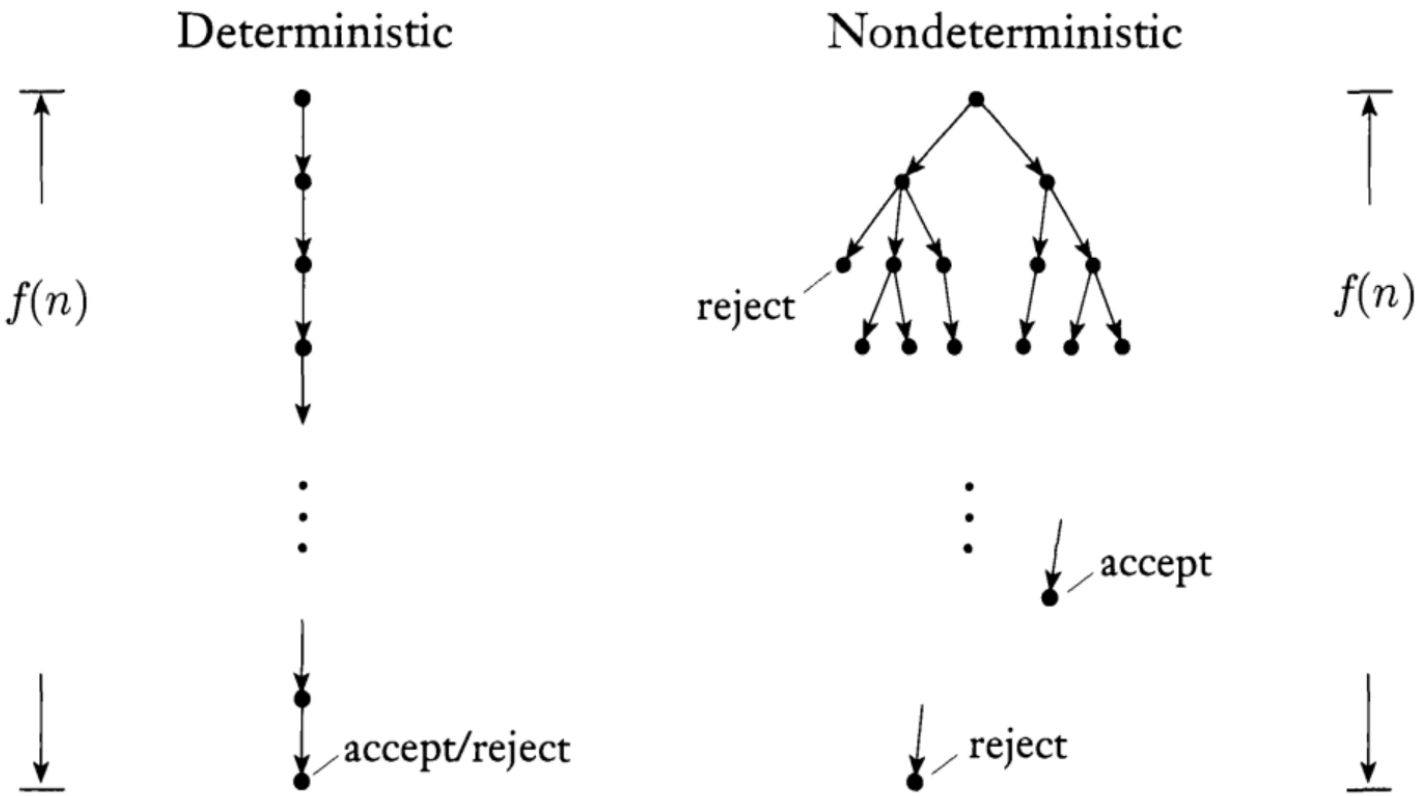
Equivalently, we say that:

Definition

The **running time** of a NTM M is a function $t_M : \mathbf{N} \to \mathbf{N}$, where $t_M(n)$ is the maximum number of steps that M uses on any branch of computation on any input of length n.

The way to think of this

For any word of length n the following holds:



Can we accept more languages with NTM?

No, but we can do it more efficiently.

Theorem

Let $t(n)$ be a function with $t(n) \geq n$. Then for any nondeterministic TM N running in time $t(n)$ there is an equivalent deterministic TM D running in time $2^{O(t(n))}$.

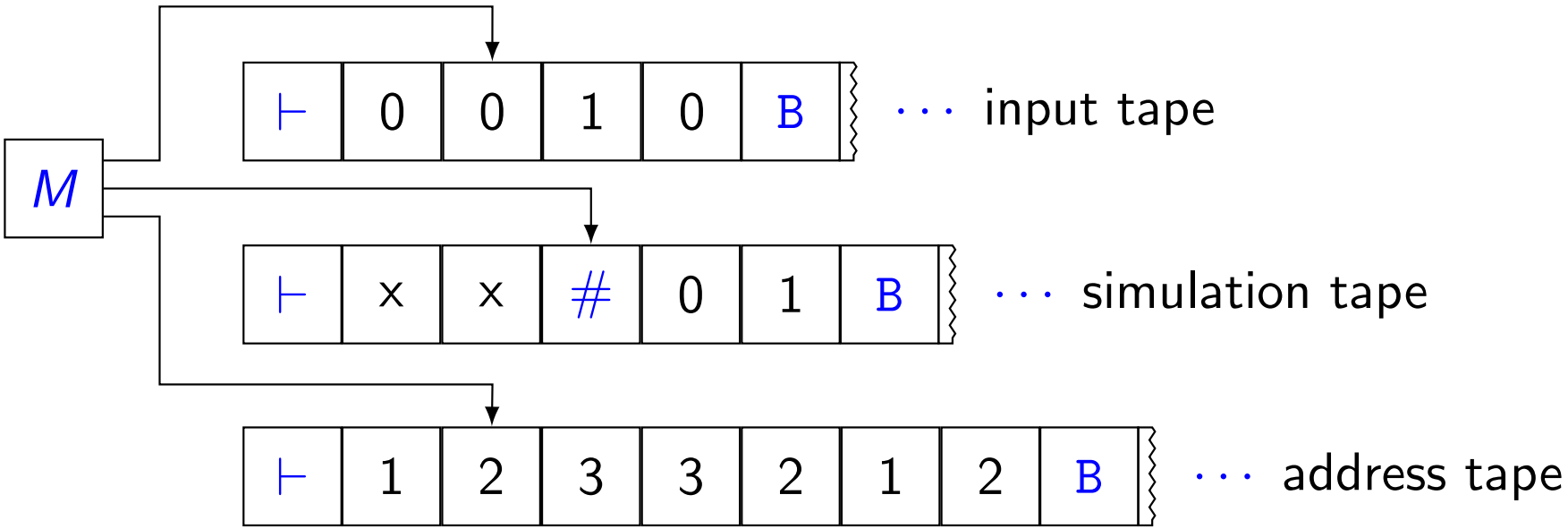
Let's see how

Proof: The idea is to examine the computation tree of the NTM using a deterministic one. If we stumble upon a branch that accepts we do as well. If not, we keep on going.

We will simulate N using a 3-tape deterministic machine D:

- Tape 1 always contains the input w
- Tape 2 simulates one branch of N's computation on w
- Tape 3 tells us which branch we are examining

Visually this means:



Let's see how this is done

We will explore the computation tree using BFS (why?)

Let b be the largest number of choices we can make according to N's δ

Tape 3 will enumerate (lexicographically) strings over $\{1, 2, \ldots, b\}$

This tells us where to go in the computation tree

Note that some addresses are meaningless

On Tape 2 we simulate this branch

Let's see how

D works as follows:

1. Start with w on Tape 1. The other tapes are empty.
2. Copy Tape 1 to Tape 2.
3. Simulate N's run according to choices from Tape 3. If it accepts accept. Otherwise goto 4.
4. Replace the string with the lexicographically next string. Goto 2.

Almost good. But recall that we want a decider.

Since N is a decider we can do this easily. How?

Let's see how

Next we analyse the complexity.

Note that going down each branch takes at most $t(n)$ time.

The number of configuration we check is at most

$$\sum_{i=1}^{t(n)} b^i = O(b^{t(n)})$$

So we take $O(t(n)b^{t(n)}) = 2^{O(t(n))}$ time