Fundamental Computer Science
Lecture 3: first steps in complexity
Complexity classes

Denis Trystram
MoSIG1 and M1Info – University Grenoble-Alpes

February, 2021

Summary of the previous lecture

- Turing Machines
  - universal computational model
  - non-determinism
    decide the same languages as the deterministic TM... but not using the same number of steps
  - all variants of the model are equivalent w.r.t. *decidability*
  - the RAM is a good trade-off

Agenda

- A focus on decidability
- Classifying the problems in complexity classes
  - time complexity:
  - space complexity

Focus on *decidable* languages (that correspond to *solvable* problems)

Time complexity class

Let $f : \mathbb{N} \to \mathbb{N}$ be a function. We define the **time complexity class**

$\mathrm{TIME}(f(n)) = \{L \mid L$ is a language *decided* by a Turing Machine in $O(f(n))$ time, where n is the size of the input$\}$

Time complexity class

Let $f : \mathbb{N} \to \mathbb{N}$ be a function. We define the **time complexity class**

$\mathrm{TIME}(f(n)) = \{L \mid L$ is a language *decided* by a Turing Machine in $O(f(n))$ time, where n is the size of the input$\}$

Example: $L = \{0^k 1^k \mid k \geq 0\}$

Example (1)

$L = \{0^k 1^k \mid k \geq 0\}$

$M_1 =$ "On input w:

1. Scan the tape and *reject* if a "0" is found on the right of a "1".
2. Repeatedly scan the tape deleting each time a single 0 and a single 1.
3. If no 0's and no 1's remain in the tape then *accept*, else *reject*."

Example (1)

$L = \{0^k 1^k \mid k \geq 0\}$

$M_1 =$ "On input w:

1. Scan the tape and *reject* if a "0" is found on the right of a "1".
2. Repeatedly scan the tape deleting each time a single 0 and a single 1.
3. If no 0's and no 1's remain in the tape then *accept*, else *reject*."

$L \in \mathrm{TIME}(n^2)$

The length of the input is $n = 2k$

Example (2)

$L = \{0^k 1^k \mid k \geq 0\}$

An enhanced Turing Machine:

$M_2 =$ "On input w:

1. Scan the tape and *reject* if a "0" is found on the right of a "1".
2. Repeat:
   - scan the tape deleting every second 0 and then every second 1.
3. If no 0's and no 1's remain in the tape then *accept*, else *reject*."

Example (2)

$L = \{0^k 1^k \mid k \geq 0\}$

An enhanced Turing Machine:

$M_2 =$ "On input w:

1. Scan the tape and *reject* if a "0" is found on the right of a "1".
2. Repeat:
   - scan the tape deleting every second 0 and then every second 1.
3. If no 0's and no 1's remain in the tape then *accept*, else *reject*."

$L \in \mathrm{TIME}(n \log_2 n)$

The class $\mathcal{P}$

A Turing Machine $M = (K, \Sigma, \Gamma, \delta, s, H)$ is called **polynomially bounded** if there is a polynomial p and for any input w there is no configuration C such that $(s, \sqcup w) \vdash_M^{p(|w|)} C$.

A language is called **polynomially decidable** if there is a polynomially bounded Turing Machine that *decides* it.

The class $\mathcal{P}$

A Turing Machine $M = (K, \Sigma, \Gamma, \delta, s, H)$ is called **polynomially bounded** if there is a polynomial p and for any input w there is no configuration C such that $(s, \sqcup w) \vdash_M^{p(|w|)} C$.

A language is called **polynomially decidable** if there is a polynomially bounded Turing Machine that *decides* it.

$\mathcal{P}$ is the class of *polynomially decidable* languages.

$$\mathcal{P} = \bigcup_k \mathrm{TIME}(n^k)$$

Recall: languages versus problems

- Decision problem: a problem whose answer is yes/no.
- Example
  - PATH
  - Input: Given a graph $G = (V, E)$ and two nodes $s, t \in V$
  - Question: is there a path from s to t?

[1]defined at a polynomal

Recall: languages versus problems

- Decision problem: a problem whose answer is yes/no.
- Example
  - PATH
  - Input: Given a graph $G = (V, E)$ and two nodes $s, t \in V$
  - Question: is there a path from s to t?
- Is PATH a language?

[1] defined at a polynomal

Recall: languages versus problems

- Decision problem: a problem whose answer is yes/no.
- Example
  - PATH
  - Input: Given a graph $G = (V, E)$ and two nodes $s, t \in V$
  - Question: is there a path from s to t?
- Is PATH a language? **No**
- How to define the language corresponding to PATH?

[1] defined at a polynomal

Recall: languages versus problems

- Decision problem: a problem whose answer is yes/no.
- Example
  - PATH
  - Input: Given a graph $G = (V, E)$ and two nodes $s, t \in V$
  - Question: is there a path from s to t?
- Is PATH a language? **No**

- How to define the language corresponding to PATH?

  $\mathrm{PATH} = \{\langle G, s, t\rangle \mid G$ is a graph that has a path from s to $t\}$

  - $\langle G, s, t\rangle$ is the input string
  - $|\langle G, s, t\rangle|$ is the size of the input[1]

[1] defined at a polynomal

Recall: languages versus problems

- Decision problem: a problem whose answer is yes/no.
- Example
  - PATH
  - Input: Given a graph $G = (V, E)$ and two nodes $s, t \in V$
  - Question: is there a path from s to t?
- Is PATH a language? **No**
- How to define the language corresponding to PATH?

  $\mathrm{PATH} = \{\langle G, s, t\rangle \mid G \text{ is a graph that has a path from } s \text{ to } t\}$

  - $\langle G, s, t\rangle$ is the input string
  - $|\langle G, s, t\rangle|$ is the size of the input[1]
- $\mathrm{PATH} \in \mathcal{P}$?

[1] defined at a polynomal

Recall: languages versus problems

- Decision problem: a problem whose answer is yes/no.
- Example
  - PATH
  - Input: Given a graph $G = (V, E)$ and two nodes $s, t \in V$
  - Question: is there a path from s to t?
- Is PATH a language? **No**
- How to define the language corresponding to PATH?

  $\mathrm{PATH} = \{\langle G, s, t\rangle \mid G$ is a graph that has a path from s to $t\}$

  - $\langle G, s, t\rangle$ is the input string
  - $|\langle G, s, t\rangle|$ is the size of the input[1]
- $\mathrm{PATH} \in \mathcal{P}$?
  - **Yes** (i.e., Breadth First Search in $O(|V| + |E|)$)

[1] defined at a polynomal

Enhanced Turing Machine models

- Does the definition of the class $\mathcal{P}$ remains the same if we use multiple tapes?

Enhanced Turing Machine models

- Does the definition of the class $\mathcal{P}$ remains the same if we use multiple tapes? **YES**
  - Recall: if a multiple tape Turing Machine *halts* on input w after t steps, then the corresponding single tape Turing Machine *halts* after $O\big(t(|w|+t)\big)$ steps.

Enhanced Turing Machine models

- Does the definition of the class $\mathcal{P}$ remains the same if we use multiple tapes? **YES**
  - Recall: if a multiple tape Turing Machine *halts* on input w after t steps, then the corresponding single tape Turing Machine *halts* after $O\big(t(|w| + t)\big)$ steps.

Example: $L = \{0^k 1^k \mid k \geq 0\}$

Enhanced Turing Machine models

- Does the definition of the class $\mathcal{P}$ remains the same if we use multiple tapes? **YES**
  - Recall: if a multiple tape Turing Machine *halts* on input w after t steps, then the corresponding single tape Turing Machine *halts* after $O\big(t(|w|+t)\big)$ steps.

Example: $L = \{0^k 1^k \mid k \geq 0\}$

$M_3 =$ "On input w:

1. Scan the tape and *reject* if a "0" is found on the right of a "1".
2. Copy the 0's in tape 2.
3. Scan tapes 1 & 2 simultaneously and delete a single 0 from tape 2 and a single 1 from tape 1.
4. If no 0's and no 1's remain then *accept*, else *reject*."

Enhanced Turing Machine models

- Does the definition of the class $\mathcal{P}$ remains the same if we use multiple tapes? **YES**
  - Recall: if a multiple tape Turing Machine *halts* on input w after t steps, then the corresponding single tape Turing Machine *halts* after $O\big(t(|w| + t)\big)$ steps.

Example: $L = \{0^k 1^k \mid k \geq 0\}$

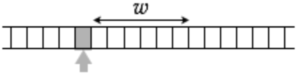
$M_3 =$ "On input w:

1. Scan the tape and *reject* if a "0" is found on the right of a "1".
2. Copy the 0's in tape 2.
3. Scan tapes 1 & 2 simultaneously and delete a single 0 from tape 2 and a single 1 from tape 1.
4. If no 0's and no 1's remain then *accept*, else *reject*."

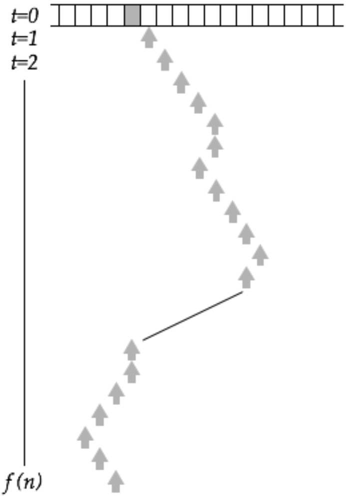
- complexity: $O(n) \quad \Rightarrow \quad L \in \mathrm{TIME}(n^2) \quad \Rightarrow \quad L \in \mathcal{P}$

A pictorial definition of Complexity

Let consider a Turing Machine:



A pictorial definition of Complexity



Synthesis and Extension to space complexity

The time-complexity is the number of elementary transitions before reaching a halting state ($f(n)$) where n is the size of the input.

Synthesis and Extension to space complexity

The time-complexity is the number of elementary transitions before reaching a halting state $(f(n))$ where n is the size of the input.

- Memory is also a critical resource.
- In complexity theory, memory is often referred as *space*.

- How to measure the memory used by an algorithm?

Synthesis and Extension to space complexity

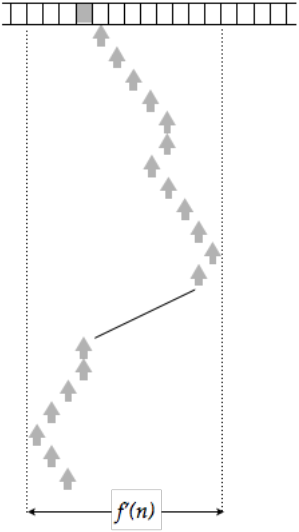
The time-complexity is the number of elementary transitions before reaching a halting state ($f(n)$) where n is the size of the input.

- Memory is also a critical resource.
- In complexity theory, memory is often referred as *space*.

- How to measure the memory used by an algorithm?

The space-complexity is the number of distinct elementary cells involved before reaching a halting state ($f'(n)$).

A pictorial view of space complexity



PSPACE

Definition

Let $f : \mathbb{N} \to \mathbb{N}$ be a function. We define the **space complexity class**

$\mathrm{SPACE}(f(n)) = \{L \mid L$ is a language *decided* by a Turing Machine using $O(f(n))$ cells on a TM, where n is the size of the input$\}$

We are interested in the characterization of the languages –problems– that are decided in polynomial space.

PSPACE is defined similarly to $\mathcal{P}$

PSPACE is the class of *polynomially decidable* languages.

$$PSPACE = \bigcup_k \mathrm{SPACE}(n^k)$$

PolyLog Space classes

The class PSPACE is very large.

We are looking for a restricted class where the number of cells visited during an execution of the TM is bounded by a poly-logarithmic function. We do not count the cells used for coding the input word.

Let us change slightly the definition.

Definition

Let $f : \mathbb{N} \to \mathbb{N}$ be a function.

$\mathrm{SPACE}(f(n)) = \{L \mid L$ is a language *decided* by a 2-tapes TM –where the first tape can not be modified (read only) and the second one is the working tape– using $O(f(n))$ cells on the second tape of this TM, where n is the size of the input$\}$

This does not change the previous definition of PSPACE but it is important for the more refined space classes.

LogSPACE

$LogSPACE$ is defined as $PSPACE(logn)$

We will come back later on this class.

Non-deterministic time complexity class

Let $f : \mathbb{N} \to \mathbb{N}$ be a function. We define the **non-deterministic time complexity class**

$\mathrm{NTIME}(f(n)) = \{L \mid L$ is a language *decided* by a non-deterministic Turing Machine in $O(f(n))$ time, where n is the size of the input$\}$

Example:

$\mathrm{HPATH} = \{\langle G, s, t\rangle \mid G$ is a graph with a Hamiltonian path from s to $t\}$

Non-deterministic time complexity class

Let $f : \mathbb{N} \to \mathbb{N}$ be a function. We define the **non-deterministic time complexity class**

$$\mathrm{NTIME}(f(n)) = \{L \mid L \text{ is a language } \textit{decided} \text{ by a non-deterministic Turing Machine in } O(f(n)) \text{ time, where } n \text{ is the size of the input}\}$$

Example:

$$\mathrm{HPATH} = \{\langle G, s, t\rangle \mid G \text{ is a graph with a Hamiltonian path from } s \text{ to } t\}$$

$M =$ "On input $\langle G, s, t\rangle$:

1. Non-deterministically generate a permutation of the vertex set, $v_1, v_2, \ldots, v_n$.
2. If $v_1 = s$, $v_n = t$ and $(v_i, v_{i+1}) \in E$ for each $i = 1, 2, \ldots n-1$, then *accept*, else *reject*."

Non-deterministic time complexity class

Let $f : \mathbb{N} \to \mathbb{N}$ be a function. We define the **non-deterministic time complexity class**

$\mathrm{NTIME}(f(n)) = \{L \mid L$ is a language *decided* by a non-deterministic Turing Machine in $O(f(n))$ time, where n is the size of the input$\}$

Example:

$\mathrm{HPATH} = \{\langle G, s, t\rangle \mid G$ is a graph with a Hamiltonian path from s to $t\}$

$M =$ "On input $\langle G, s, t\rangle$:

1. Non-deterministically generate a permutation of the vertex set, $v_1, v_2, \ldots, v_n$.
2. If $v_1 = s$, $v_n = t$ and $(v_i, v_{i+1}) \in E$ for each $i = 1, 2, \ldots n-1$, then *accept*, else *reject*."

- M decides HPATH
- $f(n) = O(n^2) \quad \Rightarrow \quad \mathrm{HPATH} \in \mathrm{NTIME}(n^2)$

Certificates and Verifiers

- "*non-deterministically generate*" a string
- check if the generated string has a certain property of the language
- if this input is in the language, then at least one such string exists
- we call this string a **certificate**

Certificates and Verifiers

- "*non-deterministically generate*" a string
- check if the generated string has a certain property of the language
- if this input is in the language, then at least one such string exists
- we call this string a **certificate**
- Examples of certificates
  - $\mathrm{COMPOSITES}$: $\langle p, q \rangle$ such that $x = p \cdot q$
  - HPATH: $\langle v_1, v_2, \ldots, v_n \rangle$ such that $s = v_1 \to v_2 \to \ldots \to v_n = t$ is a Hamiltonian path from s to t

Certificates and Verifiers

- "*non-deterministically generate*" a string
- check if the generated string has a certain property of the language
- if this input is in the language, then at least one such string exists
- we call this string a **certificate**
- Examples of certificates
  - $\mathrm{COMPOSITES}$: $\langle p, q \rangle$ such that $x = p \cdot q$
  - HPATH: $\langle v_1, v_2, \ldots, v_n \rangle$ such that $s = v_1 \to v_2 \to \ldots \to v_n = t$ is a Hamiltonian path from s to t
- A **verifier** for a language L is an algorithm $\mathcal{V}$ where

$$L = \{w \mid \mathcal{V} \text{ accepts } \langle w, c \rangle \text{ for each certificate } c\}$$

Certificates and Verifiers

- "*non-deterministically generate*" a string
- check if the generated string has a certain property of the language
- if this input is in the language, then at least one such string exists
- we call this string a **certificate**

- Examples of certificates
  - $\mathrm{COMPOSITES}$: $\langle p, q \rangle$ such that $x = p \cdot q$
  - HPATH: $\langle v_1, v_2, \ldots, v_n \rangle$ such that $s = v_1 \to v_2 \to \ldots \to v_n = t$ is a Hamiltonian path from s to t

- A **verifier** for a language L is an algorithm $\mathcal{V}$ where

$$L = \{w \mid \mathcal{V} \text{ accepts } \langle w, c \rangle \text{ for each certificate } c\}$$

- A **polynomial time verifier** runs in polynomial time with respect to the length of the input w

Equivalence of Verifiers and Non-deterministic TM

Theorem

A language L has a polynomial time verifier $\mathcal{V}$ if and only if there is a polynomial time Non-deterministic Turing Machine $NDTM$ which decides it.

Proof: $(\Rightarrow)$ Consider a polynomial time verifier $\mathcal{V}$ for L

$NDTM =$

Equivalence of Verifiers and Non-deterministic TM

Theorem

A language L has a polynomial time verifier $\mathcal{V}$ if and only if there is a polynomial time Non-deterministic Turing Machine $NDTM$ which decides it.

Proof: ($\Rightarrow$) Consider a polynomial time verifier $\mathcal{V}$ for L

$NDTM =$ "On input w of length n:

1. Non-deterministically generate a string c of length n^k.
2. Run $\mathcal{V}$ on input $\langle w, c \rangle$.
3. If $\mathcal{V}$ accepts, then *accept*, else *reject*."

Equivalence of Verifiers and Non-deterministic TM

Theorem

A language L has a polynomial time verifier $\mathcal{V}$ if and only if there is a polynomial time Non-deterministic Turing Machine $NDTM$ which decides it.

Proof: ($\Leftarrow$) Consider a polynomial time Non-deterministic Turing Machine $NDTM$ that *decides* L

$\mathcal{V} =$

Equivalence of Verifiers and Non-deterministic TM

Theorem

A language L has a polynomial time verifier $\mathcal{V}$ if and only if there is a polynomial time Non-deterministic Turing Machine $NDTM$ which decides it.

Proof: ($\Leftarrow$) Consider a polynomial time Non-deterministic Turing Machine $NDTM$ that *decides* L

$\mathcal{V}$ = "On input $\langle w, c\rangle$:

1. Simulate $NDTM$ on input w using each symbol of c as the non-deterministically choice in order to decide the next step.
2. If this branch of computation accepts, then *accept*, else *reject*."

The class $\mathcal{NP}$

A non-deterministic Turing Machine $M = (K, \Sigma, \Gamma, \Delta, s, H)$ is called **polynomially bounded** if there is a polynomial p and for any input w there is no configuration C such that $(s, \sqcup w) \vdash_M^{p(|w|)} C$.

A language is called **non-deterministically polynomially decidable** if there is a polynomially bounded Turing Machine that *decides* it.

The class $\mathcal{NP}$

A non-deterministic Turing Machine $M = (K, \Sigma, \Gamma, \Delta, s, H)$ is called **polynomially bounded** if there is a polynomial p and for any input w there is no configuration C such that $(s, \underline{\sqcup} w) \vdash_M^{p(|w|)} C$.

A language is called **non-deterministically polynomially decidable** if there is a polynomially bounded Turing Machine that *decides* it.

$\mathcal{NP}$ is the class of *non-deterministic polynomially decidable* languages.

$$\mathcal{NP} = \bigcup_k \mathrm{NTIME}(n^k)$$

The class $\mathcal{NP}$

A non-deterministic Turing Machine $M = (K, \Sigma, \Gamma, \Delta, s, H)$ is called **polynomially bounded** if there is a polynomial p and for any input w there is no configuration C such that $(s, \underline{\sqcup} w) \vdash_M^{p(|w|)} C$.

A language is called **non-deterministically polynomially decidable** if there is a polynomially bounded Turing Machine that *decides* it.

$\mathcal{NP}$ is the class of *non-deterministic polynomially decidable* languages.

$$\mathcal{NP} = \bigcup_k \mathrm{NTIME}(n^k)$$

equivalently

$\mathcal{NP}$ is the class of languages that have a polynomial time verifier.

$\mathcal{P}$ versus $\mathcal{NP}$

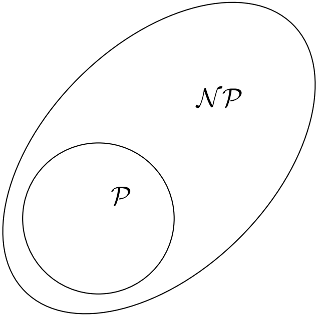
Be careful !!

$\mathcal{NP}$ means "non-deterministic polynomial" and NOT "non-polynomial"

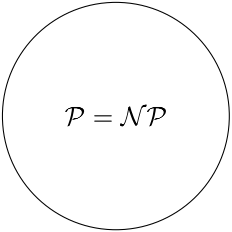
$\mathcal{P}$ versus $\mathcal{NP}$

Be careful !!

$\mathcal{NP}$ means "non-deterministic polynomial" and NOT "non-polynomial"



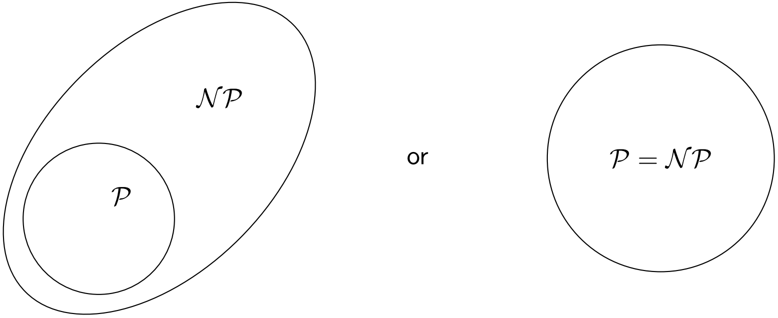
or



$\mathcal{P}$ versus $\mathcal{NP}$

Be careful !!

$\mathcal{NP}$ means "non-deterministic polynomial" and NOT "non-polynomial"



What do we know? $\mathcal{NP} \subseteq \mathrm{EXPTIME} = \bigcup_k \mathrm{TIME}(2^{n^k})$

Coming back to the space complexity: NPSPACE

Definition

Let $f : \mathbb{N} \to \mathbb{N}$ be a function. We define the **space complexity class**

$\mathrm{NSPACE}(f(n)) = \{L \mid L$ is a language *decided* using $O(f(n))$ cells on a NDTM, where n is the size of the input$\}$

Coming back to the space complexity: NPSPACE

Definition

Let $f : \mathbb{N} \to \mathbb{N}$ be a function. We define the **space complexity class**

$\mathrm{NSPACE}(f(n)) = \{L \mid L$ is a language *decided* using $O(f(n))$ cells on a NDTM, where n is the size of the input$\}$

We are interested in the characterization of the languages –problems– that are decided in polynomial space in non-deterministic TMs.

NPSPACE is the class of *polynomially decidable* languages.

$$NPSPACE = \bigcup_{k} \mathrm{NSPACE}(n^k)$$

Some important properties

$$NTIME(f(n)) \subset \bigcup_k \mathrm{TIME}(c^{f(n)})$$

where c is the cardinal of the alphabet.

Some important properties

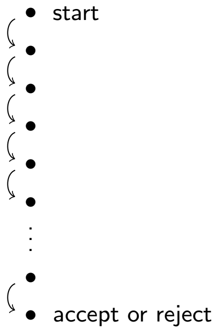
$$NTIME(f(n)) \subset \bigcup_k \mathrm{TIME}(c^{f(n)})$$

where c is the cardinal of the alphabet.

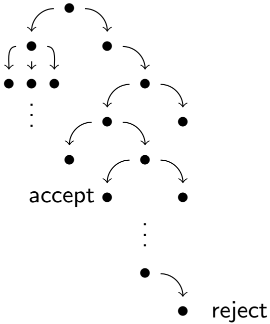
$$NTIME(f(n)) \subset \bigcup_k \mathrm{SPACE}(f(n))$$

Both proofs are intuitive and they are skipped.

Non-deterministic Turing Machines

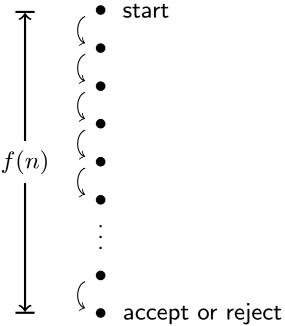


deterministic computation

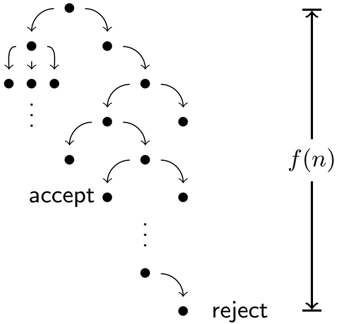


non-deterministic computation

Non-deterministic Turing Machines



deterministic computation



non-deterministic computation

The **running time** of a non-deterministic Turing Machine which *decides* a language is a function $f : \mathbb{N} \to \mathbb{N}$, where $f(n)$ is the maximum number of steps on any branch of the computation on any input of length n.

non-deterministic space vs time (1)

Let consider again the problem PATH (also called REACH):

- REACH
- Input: a graph G and two vertices s and t
- Question: Is there a path between s and t in G?

non-deterministic space vs time (1)

Let consider again the problem PATH (also called REACH):

- REACH
- Input: a graph G and two vertices s and t
- Question: Is there a path between s and t in G?

Clearly, REACH $\in \mathcal{P}$

It will play a major role in the class LogSPACE.

non-deterministic space vs time (2)

- The idea is to associate a graph to all possible transitions where ω is the input word.
  This graph has $O(c^{f(n)})$ vertices.
- Then apply REACH on (G_ω, K_ω, H)
  Let us assume WLOG that there is only one acceptance state.
- Accept ω if and only if there exists a path between the original configuration and H

REACH $\in O(n^2)$ in time

It will play a major role in the class LogSPACE.

deterministic vs non-deterministic space

- REACH $\in$ SPACE$(log^2(n))$
  The proof is based on a very smart way to progress in the paths.

Savitch Theorem

$$NTIME(f(n)) \subset \bigcup_k \mathrm{SPACE}(f(n))$$

deterministic vs non-deterministic space

- REACH $\in$ SPACE$(log^2(n))$
  The proof is based on a very smart way to progress in the paths.

Savitch Theorem

$$NTIME(f(n)) \subset \bigcup_k \mathrm{SPACE}(f(n))$$

Corollary

PSPACE = NPSPACE

Synthesis

We have the following series of inclusions:

$$LogSPACE \subseteq \mathcal{P} \subseteq \mathcal{NP} \subseteq PSPACE = NPSPACE \subseteq EXPTIME$$

Co-$\mathcal{NP}$

Definition

This is the class of languages L whose complements $(\Sigma^* - L)$ are in $\mathcal{NP}$

Let consider the following problem:

- PRIME
- **Instance:** an integer N
- **question:** Is N a prime?

Coding the input

- There exists a well-known algorithm for solving this problem
- The Erathostenes' sieve
  whose running time is in $O(\sqrt{N})$

Is the problem clearly in $\mathcal{P}$?

Coding the input

- There exists a well-known algorithm for solving this problem
- The Erathostenes' sieve
  whose running time is in $O(\sqrt{N})$

Is the problem clearly in $\mathcal{P}$? **Not clearly!**

A *natural* coding of $\textsc{Prime}$ consists in writing N in binary using $\log_2(N)$ bits.
This means that the complexity of this algorithm is exponential!

Coding the input

- There exists a well-known algorithm for solving this problem
- The Erathostenes' sieve
  whose running time is in $O(\sqrt{N})$

Is the problem clearly in $\mathcal{P}$? **Not clearly!**

A *natural* coding of $\textsc{Prime}$ consists in writing N in binary using $\log_2(N)$ bits.
This means that the complexity of this algorithm is exponential!

$\textsc{Prime}$ is an example of a problem that is hard to give a certifier...
Hard to prove that it is in $\mathcal{NP}$ (but its complement –recognize that an integer is NOT a prime– is very simple by multiplying its divisers).

Co-$\mathcal{NP}$

Let us remark that if a problem is in $\mathcal{NP}$, its complement is not necessarily in $\mathcal{NP}$...

Why?

Co-$\mathcal{NP}$

Let us remark that if a problem is in $\mathcal{NP}$, its complement is not necessarily in $\mathcal{NP}$...

Why?
We are dealing here with non-deterministic machines and thus, it is not obvious to invert the answer given by such a machine!

- A NDTM accepts a language if there exists an execution that is accepted.
- It refuses if all the executions refuse, that is not verifiable by a NDTM.

Co-$\mathcal{NP}$

Let us remark that if a problem is in $\mathcal{NP}$, its complement is not necessarily in $\mathcal{NP}$...

Why?
We are dealing here with non-deterministic machines and thus, it is not obvious to invert the answer given by such a machine!

- A NDTM accepts a language if there exists an execution that is accepted.
- It refuses if all the executions refuse, that is not verifiable by a NDTM.

For deterministic TM, we have $\mathcal{P} =$ co-$\mathcal{P}$

An open question

$$\mathcal{P} = ?\ \mathcal{NP} \bigcap \text{co-}\mathcal{NP}$$