Exercises in Class

@February 28, 2023

▼ *What is the cardinality of S^n namely the set of all strings of length n in S? Prove your claim*

Solution: $|S^n| = |S|^n$

This can be proved by induction on the value of the parameter n. In other words we prove $\forall n.P(n)$ by proving that $P(0)$ and $\forall n.P(n) \rightarrow P(n+1)$. Of course, $P(n)$ is $|S^n| = |S|^n$

Base Case ($P(0)$ is $|S^0| = |S|^0$)

$|S^0| = |\{\varepsilon\}| = 1 = |S|^0$

Inductive Case: we have to derive $|S^{n+1}| = |S|^{n+1}$ from $|S^n| = |S|^n$

$|S^{n+1}| = |S \cdot S^n|$ which is the set of strings obtained as xy where $x \in S$ and $y \in S^n$.

The cardinality of $|S \cdot S^n|$ is $|S| \cdot |S^n|$. From here, we can use the inductive hypothesis. $|S| \cdot |S|^n = |S|^{n+1}$

The Proof by induction is fairly common in CS

@March 6, 2023

▼ *Relate the following pairs of functions by way of the asymptotic operator O, Ω, Θ*

- $f_1(n) = n \log n \quad g_1(n) = 10n \log(\log n)$

Let us consider the first pair of functions. We compute the limit of the ratio between the two functions

$\lim_{n\to\infty} \frac{f_1(n)}{g_1(n)} = \lim_{n\to\infty} \frac{\log(n)}{10\log(\log(n))}$

The logarithm is a function that goes slower, then by nesting logarithms means that we will go more slowly. So the result of the limit is $+\infty$ because $\log(n)$ grows asymptotically faster than $\log(\log(n))$. We can therefore conclude that f_1 is strictly faster than $g_1 \Rightarrow$

1. $f_1(n) \notin \Theta(g_1(n))$ (of course, you can also say that $g_1(n) \in O(f_1(n))$.
2. $f_1(n) \in \Omega(g_1(n))$
3. $f_1(n) \notin O(g_1(n))$

- $f_2(n) = 1000n \quad g_2(n) = \frac{1}{100} n \log n$

$\lim_{n\to\infty} \frac{100000}{\log n}$. In this case I have a constant on the numerator (which doesn't depend on n) and a logarithm in the denominator. The result will be 0, and therefore I can be able to say that f_2 is strictly slower than $g_2 \Rightarrow$

1. $f_2(n) \notin \Theta(g_2(n))$
2. $f_2(n) \in O(g_2(n))$
3. $f_2(n) \notin \Omega(g_2(n))$

▼ *We would like to find appropriate encodings (into binary strings) for the following discrete sets*

- **The set $\mathbb{Q}$ of rational numbers**

We can write $\mathbb{Q}$ as follows: $\mathbb{Q} = \{\frac{z}{n} | z \in \mathbb{Z}, n \in \mathbb{N}, n > 0\}$

It is then sufficient to observe that $\mathbb{Q}$ can be seen as a set of pairs in the form (z, n). Each of the two numbers can then be encoded as strings in one of the ways we know.

$\llcorner z/n \lrcorner = s\ m_z \#\ m_n \in \{0, 1, \#\}^* \rightarrow$ sign of z, modulus of z and modulus of n.

Becomes completely satisfactory if you impose that z and n are co-prime ($gcd(z, n) = 1$).
We should be careful about how we represent m_z and m_n since the encoding must be reasonably succint (not too verbose), e.g. the following is not acceptable:

$1 \mapsto 10,\ 2 \mapsto 100,\ 3 \mapsto 1000, ...$
(we aim for $|\llcorner x \lrcorner| = \Theta(\log(n))$ while this is $|\llcorner x \lrcorner| = O(n)$)

- **Disjoint union of $\mathbb{N}$ and $\{0, 1\}^*$ (keep recurring occurrences)**

The solution is actually quite simple because:
the disjoint union $X \uplus Y$ is $\{(l, x) | x \in X\} \cup \{(r, y) | y \in Y\}$ and this can make us immediately realized that l and r can be

encoded as one little prefix consisting in a bit.
So $\mathbb{N} \uplus \{0,1\}^*$ can be encoded as follows:
$(l,n) \mapsto 0 \llcorner n \lrcorner$
$(r,s) \mapsto 1s$

- **Graphs, namely pairs in the form (V,E) such that V is a finite set of nodes and E is a subset of the cartesian product $V \times V$ finite set of edges, e.g.**

  $G = (\{a,b,c,d\}, \{(a,b),(a,c),(a,d),(b,d),(c,c)\})$

Graphs are usually taken as equivalent when they are isomorphic, i.e. their sets of nodes are in bijective correspondence and edges are coherent with that correspondence. E.g.

$G_1 = (\{d,b,c\}, \{(d,b),(b,d),(d,c),(c,b)\})$ and $G_2 = (\{3,2,1\}, \{(3,1),(1,3),(2,3),(1,2)\})$. The correspondence is $(d,3),(b,1),(c,2)$

The set of nodes, then, can be taken to be $\{1,2,3,...,n\}$ for a certain $n \in \mathbb{N}$. This way, the set of nodes can be represented as the binary string $\llcorner n \lrcorner$.

The edges instead are not so easy to be represented. There are at least two ways of representing the edges of a graph:

1. Adjacency Matrix

   The set E of edges becomes a $n \times n$ matrix, where each component is a binary digit. $M_{ij} \in \{0,1\}$. It is 0 if $(i,j) \notin E$, 1 otherwise. The dimension of the matrix would be n^2: $|M| = \Theta(n^2)$.
   If the proportion between nodes and edges is skewed I have sparse graph: not many edges given a high number of nodes.

2. Adjacency Lists

   You represent all the edges whose first component is a given node i as a list, the whole set of edges thus becoming a list of lists, e.g.

   $N = \{1,2,3\}, E = \{(1,2),(1,3),(2,3),(3,3)\} \mapsto \llcorner 3 \lrcorner \# \llcorner 2 \lrcorner \beta \llcorner 3 \lrcorner \# \llcorner 3 \lrcorner \# \llcorner 3 \lrcorner$ (β is another separator, distinct from $\#$ and meant to keep the encodings of edges in the same adjacency list separated.

@March 13, 2023

▼ *Prove the un-decidability of the halting problem, namely the fact that the function*

$$halt(\alpha, x) = \begin{cases} 1 \text{ if } \mathcal{M}_\alpha \text{ terminates} \\ 0 \text{ otherwise} \end{cases}$$ *is uncomputable*

We show that from an hypothetical machine computing the function $halt$, call it $\mathcal{M}_{halt}$, we can get another machine, call it $\mathcal{M}_{uc}$, which computes the function uc that we know from the slides, being uncomputable. It is again a proof by contradiction. As a consequence, we can get that $\mathcal{M}_{halt}$ cannot exist and $halt$ is uncomputable.

Let us see how we can turn $\mathcal{M}_{halt}$ into $\mathcal{M}_{uc}$. We stay into an informal level, being careful not to "cheat" with subroutines.

→ on an input α, $\mathcal{M}_{uc}$ proceeds by calling $\mathcal{M}_{halt}$ seen as an auxiliary algorithm, and feeding it with (α, α).

→ $\mathcal{M}_{uc}$ then waits until $\mathcal{M}_{halt}$ returns a result and then:

⇒ In the case $\mathcal{M}_{halt}$ returns 0 (meaning that $\mathcal{M}(\alpha,\alpha)$ diverges), $\mathcal{M}_{uc}$ returns 1

⇒ Instead, if $\mathcal{M}_{halt}$ returns 1, meaning that $\mathcal{M}_\alpha(\alpha)$ terminates, then $\mathcal{M}_{uc}$ knows it can safely call $\mathcal{U}$ on input (α,α), because it knows that the machine $\mathcal{M}_\alpha$ converges on α. Let $b \in \{0,1\}^*$ be the result produced in output by $\mathcal{U}$ on input (α,α). Then $\mathcal{M}_{uc}$:

→ returns 0 if $b = 1$

→ returns 1 otherwise

$\mathcal{M}_{uc}$ as we have defined it is indeed a Turing Machine computing the function uc. But since we know that such a machine cannot exist, the only hypothesis we did, namely the existence of $\mathcal{M}_{halt}$ must be false. □

▼ *Show that the function $inc : \mathbb{N} \to \mathbb{N}$ such that $inc(n) = n + 1$ can be computed in linear time by explicitly constructing a Turing Machine for it.*

inc acts on $\mathbb{N}$, so we need to find a way to encode elements of $\mathbb{N}$ as binary strings. Classic binary encoding might be a way to complicate things; therefore, we use a "reversed version" (e.g. 12 = 0011 = 0*1+0*2+1*4+1*8 → the head of the only tape is on the least significant bit, which means one single pass is enough).

The Turing Machine at hand has four states, $q_{init}, q_0, q_1, q_{halt}$, one tape which acts as an input-work-output tape, and Γ= $\{\triangleright, \square, 0, 1\}$. The transition function is the following:

$\rightarrow (q_{init}, \triangleright) \overset{\delta}{\mapsto} (q_0, \triangleright, R)$

$\rightarrow (q_0, 0) \overset{\delta}{\mapsto} (q_1, 1, L)$

$\rightarrow (q_0, 1) \overset{\delta}{\mapsto} (q_0, 0, R)$

$\rightarrow (q_0, \square) \overset{\delta}{\mapsto} (q_1, 1, L)$

$\rightarrow (q_1, 1) \overset{\delta}{\mapsto} (q_1, 1, L)$

$\rightarrow (q_1, 0) \overset{\delta}{\mapsto} (q_1, 0, L)$

$\rightarrow (q_1, \triangleright) \overset{\delta}{\mapsto} (q_{halt}, \triangleright, S)$

Coming back to the starting point is not actually necessary, but it is a convention often followed.

@March 14, 2023

▼ *Classify the following languages as either decidable or undecidable:*

▼ *The set of (Encodings of) Turing Machines which compute the inc function (see previous toggle)*

The first language, call it $\mathcal{L}_{inc}$, is semantic. Moreover, $\mathcal{L}_{inc}$ is not trivial, because in the previous exercise we showed $\mathcal{L}_{inc}$ to be non-empty and $\mathcal{L}_{inc} \neq \{0,1\}^*$: the code of any Turing Machine computing the Identity Function $id : \{0,1\}^* \to \{0,1\}^*$ s.t. $id(x) = x$ cannot be in $\mathcal{L}_{inc}$. We know, however, that Semantic non-trivial sets/languages cannot be decidable, thanks to Rice's Theorem. $\square$

▼ *The set of (Encodings of) Turing Machines whose set of states includes a number of states between 10 and 27*

The second language, call it $\mathcal{L}_{10to27}$, is indeed decidable. This is because extracting the number of states from a Turing Machine encoding is of course possible, because states are part of the definition of a Turing Machine.

▼ *The set of Terminating Turing Machines (machines that never diverge) satisfying the following property: whenever the machine finds itself in front of three consecutive zeros in the work tape, it halts immediately* [this is not required for the exam]

The third language, call it $\mathcal{L}_{000}$, looks quite challenging and very likely to turn out to be undecidable. On the other hand, $\mathcal{L}_{000}$ is not semantic, so we cannot apply Rice's Theorem. How can we proceed then? The only other tool we know is Reduction $\phi : \{0,1\}^* \to \{0,1\}^*$ s.t. $\phi(x) \in \mathcal{L}_{000} \iff x \in \mathcal{I}$, where $\mathcal{I}$ must be a language we already know is undecidable.

Let us take $\mathcal{I}$ as the halting problem, namely the set of pairs (α, x) s.t. $\mathcal{M}_\alpha(x)$ converges. We are looking for ϕ s.t. $x \overset{\phi}{\mapsto} ?$ (x is an instance of the halting problem and $?$ is an instance of $\mathcal{L}_{000}$. We can proceed by defining ϕ as follows: the code α is manipulated by making sure that the machine α encodes never writes the symbol 0 in the working tape. Instead, such machine uses a new symbol $\bar{0}$ which was not part of the alphabet of α. This way, if (α, x) - this machine $\phi(\alpha, x)$ simulates $\mathcal{M}_\alpha$ on x throwing away its input - is in the halting language, of course, $\phi(\alpha, x)$ is in $\mathcal{L}_{000}$. Instead, if (α, x) is not a halting pair, then α is not a machine which always converges and as such $\phi(\alpha) \notin \mathcal{L}_{000}$.

@March 20, 2023

▼ *Prove that the function minmax which given the encoding of a sequence of natural numbers $(a_1, ..., a_n)$ returns both the least and the greatest elements in the sequence is in $\mathbf{FP}$. As an example, on input $(11, 23, 3, 7, 2)$ the function minmax should produce in output the pair $(2, 23)$.*

Complete freedom to choose between using pseudocode or Turing Machines; however, it might become bothersome to use TMs in this case, no matter how easy the problem seems. We will use pseudocode:

Input: a sequence $(a_1, ..., a_n)$, $a_i \in \mathbb{N}$, appropriately encoded.
Output: (b, c) where b is the least and c is the greatest of the a_is.

```
min <- a1
max <- a1
for i <- 2 to n do:
  if ai < min then
    min <- ai
  if ai > max then
    max <- ai
  end
endfor
return (min, max)
```

What would happen if we use the $\geq$ or $\leq$? Absolutely nothing, it would be fine anyway. Here, we change only if we find a strict difference, we avoid the useless computation of switching a number with itself.

Since the algorithm above exhaustively look for the least and greatest elements in $(a_1, ..., a_n)$ it is correct by construction.

Let us then consider the three requirements we have to check to be sure that the algorithm above works in polynomial time:

1. **Number of Instructions**:
   2 instructions at the beginning, $O(n)$ - the body of the for linear amount of time, the for body contains at most 4 instructions (it won't always execute all four, but we are pessimistic)

   Therefore $2 + O(n) \cdot 4 + 1 \in O(n)$. This means it is polynomially bounded.
2. **Size of intermediate results**:
   Besides the input itself, we have to look at the size of auxiliary variables, namely $\min$ and $\max$. Both of them at any time contain one of the elements in the input list. In other words, their sizes is always bounded by the size of the input.
3. **Time necessary to execute each instruction**:
   The only instructions which deserve to be considered are the comparisons, but we know that comparing natural numbers can be done in time polynomial in their sizes (in principle, should be proven. In practice, basic operators, logic operators, arithmetical comparisons can be used as primitives).

We can then conclude that minmax is in $\mathbf{FP}$.

▼ *Prove that the wedding plan problem as we have described it at the beginning is in the class* $\mathbf{FEXP}$.

First of all, let us recall how the problem is defined: given a list of candidate invitees and a list of incompatibility constraints between invitees, one wants to build a sublist of invitees which are compatible with each other, and which has maximum length.

Let us then show how this problem is an instance of a problem about graphs (Maximum Independent Set Problem).

The problem then refers to indirected graphs, namely pairs (V, E), where V is a finite set of vertices /nodes and E is a subset of $\{\{l, m\} | l, m \in V, l \neq m\}$

In an undirected graph $\mathbb{G} = (V, E)$, whenever $v \in V$, we write $N(v) = \{\{v, w\} \in E\}$

The wedding problem then can be spelled out as follows: given an undirected graph $\mathbb{G} = (V, E)$ determine a subset of V, call it W such that:

1. for all $v \in W$, $N(w) \cap W = \emptyset$
2. $|W|$, namely the number of elements in W, is maximum among that of all subsets of V satisfying 1.

The function we want to prove in $\mathbf{FEXP}$, then, is a function, call it $\text{findset} : \text{graphs} \to \text{finite-sets}$.

The algorithm we are looking for, thus, looks as follows:
Input: an undirected graph $\mathbb{G} = (V, E)$
Output: $W \subseteq V$ satisfying 1. and 2. above.

```
Z <- \empty
foreach W in V do:
  ind <- True
  foreach w in W do:
    if N(w) intersect W != \empty then:
      ind <- False
  endfor
  if ind then:
    if |Z|<|W| then:
      Z <- W
endfor
return Z
```

Now we have to perform the checks:

1. Number of instructions: $2 + O(2^{|V|}) \cdot (4 + O(|V|) \cdot 2) \in O(2^{|V|}) \cdot O(|V|) \in O(2^{|V|^2})$.
2. Size of intermediate results: we have that $|W| \leq |V|$ (trivially, $W \subseteq V$), and also $|Z| \leq |V|$. All intermediate results are polynomial in size.
3. Time necessary to solve instructions (Complexity of instructions): as for the first instruction, it's okay. The rest is Exponential in time.

Class with Tutor

@March 21, 2023

▼ Mathematical Preliminaries

f is $O(g)$ when $\exists c$ s.t. $f(n) \leq g(n) \cdot c$. f is $\Omega(g)$ when ...$f(n) \geq g(n) \cdot c$. f is $\Theta(g)$ if it is $\Omega(g)$ and $O(g)$

▼ $f(n) = n^2,\ g(n) = 4n + 100\log(n)$

$\lim \frac{f(n)}{g(n)} \overset{n\to\infty}{\to} \infty$

This means $f \in \Omega(g)$

▼ $f(n) = 10^{-3}n^3,\ g(n) = 10^4 n^3 + 10^5 n^2$

Still a polynomial function.

$\frac{f(n)}{g(n)} \overset{+\infty}{\to} 10^{-7}$

Therefore $f \in \Theta(g)$

▼ $f(n) = (2.99)^n n^{100},\ g(n) = \frac{3^n}{n^2}$

$\frac{f(n)}{g(n)} = (\frac{(2,99)}{3})^n n^{102} \overset{+\infty}{\to} 0$

f is in a sense smaller than g. $f \in O(g)$.

▼ Turing Machines

A Turing Machine is $TM = (\Gamma, Q, \delta)$ where Γ is the alphabet, with $\square, \triangleright, 0, 1 \in \Gamma$, Q is a finite set of states (with special states q_{init}, q_{halt}) and $\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{L, S, R\}^k$, with k number of tapes

▼ *Parity problem*

Input: A word $w \in \{0,1\}^*$

Output: returns 1 if the number of 1 in w is odd, 0 if it is even

What you do is read the input from left to right, keep "in memory" the parity of the substring you just read.

$\Gamma = \{\square, \triangleright, 0, 1\}$. $Q = \{q_{init}, q_{halt}, q_0, q_1\}$. $\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{L, S, R\}^k$. We will use two tapes.

$\delta(q_{init}, (\triangleright, \triangleright)) = (q_0, \triangleright, (R, R))$

$\delta(q_0, (0, \square)) = (q_0, \square, (R, S))$

$\delta(q_0, (1, \square)) = (q_1, \square, (R, S))$

$\delta(q_0, (\square, \square)) = (q_{halt}, 0, (S, S))$ (we're writing an output)

$\delta(q_1, (0, \square)) = (q_1, \square, (R, S))$

$\delta(q_1, (1, \square)) = (q_0, \square, (R, S))$

$\delta(q_1, (\square, \square)) = (q_{halt}, 1, (S, S))$

$\delta(q_{halt}, (\sigma_1, \sigma_2)) = (q_{halt}, \sigma_2, (S, S))$ to maintain the principles of TMs

This TM always halts, the complexity is $n + 2$ instructions ($\Theta(n)$)

▼ *"Set of Palindromes" Problem*

Input: A word $w \in \{0,1\}^*$

Output: returns 1 if w is a palindrome.

With a TM with 3 tapes.

1. Copy the input tape to the work tape

   $\delta(q_{copy}, (0, \square, \square)) = (q_{copy}, (0, \square), (R, R, S))$

2. Move back the header of the input tape at the beginning

   $\delta(q_{left}, (\sigma, \sigma', \square)) = (q_{left}, (\sigma', \square), (L, S, S))$

   $\delta(q_{left}, (\triangleright, \sigma, \square)) = (q_{step3}, (\sigma, \square), (R, L, S))$

3. Move simultaneously the two headers, Right for the input, Left for the working tape. If you find something different, halt and output 0, else arrive to the end, halt and output 1.

   $\delta(q_{step3}, (\sigma, \sigma, \square)) = (q_{step3}, (\sigma, \square), (R, L, S))$

   $\delta(q_{step3}, (0, 1, \square)) = (q_{halt}, (1, 0), (S, S, S))$

$Q = \{q_{init}, q_{halt}, q_{copy}, q_{left}, q_{step3}\}$

▼ *Reverse Problem*

Input: A word $w \in \{0,1\}^*$

Output: the reverse of w

1. Go to the far right of the input tape
2. Go back to the left on the input tape and start the copy on the output tape.
3. Halt when you see the $\triangleright$ on the input tape.

▼ Undecidability

> 👩 **Theorem**
> Any decidable semantic language $\mathcal{L}$ is trivial ($\mathcal{L} = \emptyset, \mathcal{L} = \{0,1\}^*$) $\iff$ If $\mathcal{L}$ is semantic and non-trivial, then $\mathcal{L}$ is undecidable

▼ $\mathcal{L} = \{\llcorner \mathcal{M} \lrcorner | \mathcal{M}(111) = 111\}$

$\mathcal{L}$ is semantic:

- language of encodings $\llcorner \mathcal{M} \lrcorner$
- if $\mathcal{N}$ computes the same function as $\mathcal{M}$

then if $\mathcal{M} \in \mathcal{L} \iff \mathcal{M}(111) = 111 \iff \mathcal{N}(111) = 111 \iff \mathcal{N} \in \mathcal{L}$

$\mathcal{L}$ is non-trivial

- there exists $\mathcal{M}$ such that $\llcorner \mathcal{M} \lrcorner \in \mathcal{L}$
- there exists $\mathcal{N}$ s.t. $\llcorner \mathcal{N} \lrcorner \notin \mathcal{L}$

The copy machine is in $\mathcal{L}$. The zero-constant machine is not in $\mathcal{L}$.

Thus, $\mathcal{L}$ is undecidable.

▼ $\mathcal{L} = \{\llcorner \mathcal{M} \lrcorner | \mathcal{M}$ *is a Turing Machine*$\}$

Is $\mathcal{L}$ semantic:

- It is a language of encodings of TM
- If $\llcorner \mathcal{M} \lrcorner \in \mathcal{L}$ and $\mathcal{N} \approx \mathcal{M}$, then $\llcorner \mathcal{N} \lrcorner \in \mathcal{L}$.

$\mathcal{L}$ is trivial! $\mathcal{L} = \{0,1\}^*$

▼ *What is the encoding of a TM?*

Suppose you have an encoding function: $\mathcal{M} \to \llcorner \mathcal{M} \lrcorner$. Given this encoding, we can compute the reverse $\llcorner \mathcal{M} \lrcorner \mapsto \mathcal{M}$.

If w is not an encoding of a TM, then we decrypt w as the "zero-constant" TM. With this encoding E, then $\mathcal{L}$ is trivial.

$\mathcal{L} = \{E(\mathcal{M}) | \mathcal{M}$ is a turing machine$\}$ then $\mathcal{L} = \{0,1\}^*$

▼ $\mathcal{L} = \{\llcorner \mathcal{M} \lrcorner | \mathcal{M}$ *decides the language* $\emptyset\} \iff \mathcal{L} = \{\llcorner \mathcal{M} \lrcorner | \mathcal{M}(w) = 0$ *for any* $w \in \{0,1\}^*\}$

$\mathcal{L}$ is semantic: if $\mathcal{M} \approx \mathcal{N}$ then $\mathcal{N}(w) = 0$ if $\llcorner \mathcal{M} \lrcorner \in \mathcal{L}$

Non-trivial: the "zero-constant" machine is in $\mathcal{L}$. The "one-constant" machine is not in $\mathcal{L}$.

▼ $\mathcal{L} = \{\llcorner \mathcal{M} \lrcorner | \mathcal{M}$ *has more than 5 states*$\}$

It is non-trivial, it is not semantic. Indeed, if $\mathcal{M} \approx \mathcal{N}$ and $\llcorner \mathcal{M} \lrcorner \in \mathcal{L}$ there is no reason for $\llcorner \mathcal{N} \lrcorner \in \mathcal{L}$. $\mathcal{L}$ is decidable: just count the number of states.

▼ Complexity

▼ *The following problem is in* $\mathbf{FP}$.

Data: a string $s \in \{0,1\}^*$

```
p <- s
l <- |s|
i <- 1

while i < l do:
  p := p @ s
  i <- i + 1
end
return p
```

1. Number of instructions: $4 + 2 * (|s| - 1)$. So this is polynomial
2. Size of intermediate results: the variables we have defined are initially all polynomial in size; after the i^{th} while, p will be polynomial in size $|p| = (i+1)|s|$. Because $i < l$, at the end $|p| = |s| \times |s| = |s|^2$
3. Complexity of instructions: assignments are polynomials and increments are polynomial. The end is, we're still in $\mathbf{FP}$

▼ *Change a single line*

```
p <- s
l <- |s|
i <- 1

while i < l do:
  p := p @ p
  i <- i + 1
end
return p
```

Number of instructions and polynomial time computability stays true. But the intermediates are not anymore polynomial. Let us look at p. This time is exponential. After i loops $|p| = 2^i|s|$. At the end, $|p| = 2^{|s|-1}|s|$. Thus, the algorithm is not in $\mathbf{FP}$.

▼ *Take this algorithm: triangle-free*

Input: a graph $\mathbb{G} = (V, E)$

Output: Return 1 if $\mathbb{G}$ has no triangles.

```
forall (u,v) in E, u != v
  forall w in V
    if w!=u & w!=v & (u,w) in E && (w,v) in E
      return 0
return 1
```

At most n^2 instructions.

It is in $\mathbf{FP}$

@March 30, 2023

▼ Examples of NP-Complete Problems

▼ *Knapsack Problem*

Given a set of objects $\{1, ..., n\}$ each of them coming with a weight w_i and a price p_i, and given a maximum weight Z and a price R, determine whether there is a choice of objects $I \subseteq \{1, ..., n\}$ s.t. $\sum_{i \in I} w_i \leq Z$ and $\sum_{i \in I} p_i \geq R$.

▼ *Bin-Packing*

You have n objects $1, ..., n$, each of them having a size s_i, a bin size $B \in \mathbb{N}$ and a natural number $K \in \mathbb{N}$. You are asked to determine whether there is a partition $X_1, ..., X_k$ of $\{1, ..., n\}$ such that $\sum_{i \in X_j} s_i \leq B \; \forall j \in \{1, ..., k\}$

▼ *Clique*

A Clique in an undirected graph is a subset W of the set of nodes V s.t. each pair of distinct nodes $v, w \in W$ are such that there's an edge between v and w. This problem asks you to determine whether a graph $\mathbb{G}$ has a clique of cardinality at least K

▼ *Hamiltonian Path/Cycle*

A Hamiltonian path in a graph is a path (sequence of nodes linked by edges) that goes through each vertex of the graph exactly once (similarly for a cycle). The hamiltonian path problem consists in determining whether a given graph $\mathbb{G}$ has a hamiltonian path or not (similarly for cycles).

Similarly to these problems, one can consider Eulerian paths/cycles, namely paths/cycles going through each edge of the graph exactly once.

These problems are both in $\mathbf{P}$! This is because, e.g. $\mathbb{G} \in$ `EULCYCLE` iff each vertex has an even degree, namely an even number of outgoing edges

▼ *Traveling Salesman Problem*

Given an undirected graph $\mathbb{G}$ whose edges are labeled with weights in the form of natural numbers, you are asked to determine whether there is a cycle in the graph going through all the vertices exactly once and having total weight at most equal to $k \in \mathbb{N}$.

▼ *Vertex Cover*

Given an undirected graph $\mathbb{G}$ and a natural number K, the problem consists in checking whether $\mathbb{G}$ has a cover of at most K nodes, where a cover of a graph is a subset of its nodes s.t. every edge has one of its endpoints in the cover.

▼ *Theorem*: $\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{EXP}$

Proof Sketch: First of all, let us consider $\mathbf{P} \subseteq \mathbf{NP}$. Any $\mathcal{L} \in \mathbf{P}$ must then be proved to be in the form $\{x \in \{0,1\}^* | \exists y \in \{0,1\}^{p(|x|)}.\mathcal{M}(x,y) = 1\}$ where p is a polynomial and $\mathcal{M}$ is a polytime Turing Machine. We can however take p to be the polynomial which identically equal to 0 and $\mathcal{M}$ as the algorithm deciding $\mathcal{L}$ in polynomial time, which exists by hypothesis. Then:

$$\mathcal{L} = \{x \in \{0,1\}^* | \mathcal{N}(x) = 1\} = \{x \in \{0,1\}^* | \exists y \in \{0,1\}^{\emptyset}.\mathcal{M}(x,y) = 1\}$$

where $\mathcal{N}$ is a polytime TM for $\mathcal{L}$ and $\mathcal{M}(x,y) = \mathcal{N}(x)$. In other words, $\mathcal{L} \in \mathbf{NP}$

Let us now prove the second inclusion, namely $\mathbf{NP} \subseteq \mathbf{EXP}$. Let

$\mathcal{L} = \{x \in \{0,1\}^* | \exists y \in \{0,1\}^{p(|x|)}.\mathcal{M}(x,y) = 1\}$. We have to prove that $\mathcal{L} \in \mathbf{EXP}$. So, let us design an exponential-time algorithm for $\mathcal{L}$.

Input: $x \in \{0,1\}^*$
Output: 0 or 1, dependently on $x \in \mathcal{L}$

```
foreach * do:
  simulate \mathcal{M}(x,y) until it produces a result b
  if b=1
    return 1
endfor
return 0
```

The fact that the algorithm above is correct, i.e. it decides $\mathcal{L}$, is self-evident. It considers all possible certificates. It remains to show that it works in exponential time.

- The for loop is executed at most $2^{p(|x|)}$ times (the number of certificates).
- Each iteration of the for loop takes time $q(|x|)$, where q is a polynomial. Because $\mathcal{M}$ works in poly time and $|y|$ is not so big compared to $|x|$.
- The size of intermediate results is poly-bounded

The overall runtime is then $2^{p(|x|)} \cdot q(|x|) \leq 2^{p(|x|)} \cdot 2^{\lg q(|x|)} \leq 2^{p(|x|)+q(|x|)}$. This expression is upper bounded by $|x|^c$ if c is "big enough"

Class with Tutor

@April 3, 2023

The exercises of today are taken from previous exams. Focus on $\mathbf{P}$ and $\mathbf{NP}$ problems

$\mathcal{L} \in \mathbf{NP}$ if there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a polytime TM s.t. $\mathcal{L} = \{x \in \{0,1\}^* | \exists y \in \{0,1\}^{p(|x|)}.\mathcal{M}(\llcorner x, y \lrcorner) = 1\}$. y is the certificate, $\mathcal{M}$ is the verifier.

▼ *Show that the following language is in* $\mathbf{NP}$

$\mathcal{L} = \{(\mathbb{G}, k) | \exists f : V \mapsto \{1, ..., k\}.f(v) \neq f(w)$ if $(V,w) \in E\} \in \mathbf{NP}$?$\mathcal{L} = \{(\mathbb{G}, k) | \exists f : V \mapsto \{1, ..., k\}.f(v) \neq f(w)$ if$(V,w) \in E\} \in \mathbf{NP}$? ← Graph Coloring

Use algorithm method

Certificate: A function $f : V \to \{1, ..., k\}$ ← poly size?

Verifier: Input $(\mathbb{G}, k), f$. Output: return 1 if f is a correct coloring $\iff f(v) \neq f(w)$ for any $(v,w) \in E$ ← poly time

The certificate is polynomial in size since it has size $|V|$.

```
for each (v,w) in E:
  if f(v) = f(w) then return 0
return 1
```

This is a poly-time algorithm because you have $O(|E|)$ simple instructions.

Use a non-deterministic Turing Machine

Input: $(\mathbb{G}, k)$

```
f = []
for each v in V:
  choose i in {1,...,k}
  f.push((V,i))
for each (u, v) in E:
  if f(v) = f(w) return 0
return 1
```

If this algorithm is poly-time and there exists at least one non-deterministic path that returns 1, then $\mathcal{L} \in \mathbf{NP}$.

This algorithm is poly-time:

- $O(|V| + |E|)$ instructions
- f is polynomial in size $|V|$
- each instruction can be done in polynomial time

So $\mathcal{L} \in \mathbf{NP}$

▼ *Show that the following language is in* $\mathbf{NP}$

$\mathcal{L} = \{(n, m, k) | n, m \text{ are natural numbers with at least } k \text{ prime factors in common}\} \in \mathbf{NP}$?

Certificate: A list $p_1, ..., p_k$ of integers

Verifier: Input $(n, m, k), p_1, ..., p_k$. Output: return 1 $\iff$ $p_1, ..., p_k$ are prime numbers and $p_1, ..., p_k$ are factors of both n and m.

$p_1, ..., p_k$ is poly-size $\leq k|n|$.

```
for each 1 <= i <= k:
  verify that pi is prime (AKS algorithm)
  if pi is not prime return 0
if n % p1*p2*...*pk !=0 then return 0
if m % p1*p2*...*pk !=0 then return 0
return 1
```

This algorithm is poly-time:

- $O(k)$ instructions
- no intermediate results
- each instruction is poly-time (especially verifying that a number is prime).

$\mathcal{L} \in \mathbf{NP}$

▼ *Show that the following language is in* $\mathbf{NP}$

$\mathcal{L} = \{(\mathbb{G}, u, v) |$ There exists a path in $\mathbb{G}$ from u to $v\}$

Certificate: a sequence $u, w_1, ..., w_n, v$.

We ask that $n \leq |V|$ because if a path from u to v exists, then there is one with length $\leq |v|$.

Verifier: Input: $(\mathbb{G}, u, v), u, w_1, ..., w_n, v$

```
w0 = u, wn+1 = v
for 0 <= i <= n:
```

```
  if not ((wi, wi+1) in E) then return 0
return 1
```

The verifier is poly-time, thus we have indeed $\mathcal{L} \in \mathbf{NP}$

Alternative Solution (Applicable if you know an actual poly-time algorithm that solves the task): The reachability problem can be done in polynomial time (using for example depth-first search).Thus, we know that $\mathcal{L} \in \mathbf{P}$, and because $\mathbf{P}$ is included in $\mathbf{NP}$, we have $\mathcal{L} \in \mathbf{NP}$

▼ *You are given a language $\mathcal{L}$ and you are asked: "What is complexity class in which $\mathcal{L}$ is in? Justify your answer"*

$\mathcal{L} \leq_p \mathcal{H}$ if there exists a poly-time computable function $f : \{0,1\}^* \to \{0,1\}^* s.t.\ x \in \mathcal{L} \iff f(x) \in \mathcal{H}$ù

▼ $\mathcal{L} = \texttt{MULTISAT} = \{(\mathtt{n}, \varphi_1, \ldots, \varphi_\mathtt{n}) | \varphi_1, \ldots, \varphi_\mathtt{n} \text{ are CNF formulas s.t. for all } \mathtt{i}, \varphi_\mathtt{i} \text{ is satisfiable}\}$

We show that **MULTISAT** is $\mathbf{NP}$-complete.

- **MULTISAT** $\in \mathbf{NP}$.

  Certificate: n assignments ρ_1, ..., ρ_n, poly-size $\leq |\varphi_1| + ... + |\varphi_n|$

  Verifier: Input: $(n, \varphi_1, ..., \varphi_n), \rho_1, ..., \rho_n$

```
foreach 1 <= i <= n
  if [|phi_i|]^p = false then return 0
return 1
```

  This is a poly-time algorithm

- **MULTISAT** is $\mathbf{NP}$-hard

  To do this, we show that $\mathcal{L} \leq_p$ **MULTISAT** with $\mathcal{L}$ a known $\mathbf{NP}$-complete problem. We take $\mathcal{L} =$ **SAT** $= \{\varphi | \varphi \texttt{ is satisfiable}\}$ which is a $\mathbf{NP}$-complete problem.

  SAT $\leq_\mathtt{p}$ **MULTISAT** $\rightarrow$ we need to find $f : \{0,1\}^* \to \{0,1\}^*$s.t. $x \in$ **SAT** $\iff \mathtt{f(x)} \in$ **MULTISAT**

  we define $f(\llcorner \varphi \lrcorner) = \llcorner 1, \varphi \lrcorner$

  $\llcorner \varphi \lrcorner \in$ **SAT** $\iff \varphi$ **is a CNF satisfiable formula** $\iff (1, \varphi)$ **is a sequence of 1 CNF formula that is**

We can conclude that **MULTISAT** is $\mathbf{NP}$-Complete.

▼ 5*Clique Problem*

$\mathcal{L} = \{(\mathbb{G}, k) |$ there exists a clique in $\mathbb{G}$ of size at least $k\}$, $\mathbf{NP}$-Complete problem

5CLIQUE $= \{\mathbb{G} |$ There exists a clique of size $\geq 5\}$

- **5CLIQUE** $\in \mathbf{NP}$

```
for each V1 in V
  for each V2 in V
    for each V3 in V
      for each V4 in V
        for each V5 in V
          if (V1,...,V5) is a clique return 1
return 0
```

  This is a poly-time algorithm because you have $O(|V|^5)$ instructions. And this is over.

▼ $\mathcal{L} = \{(\mathbb{G}, X) | X \subseteq V, \mathbb{G}$ has a hamiltonian cycle for $X\}$

- $\mathcal{L}$ is $\mathbf{NP}$-hard

  To do this, we need to take a known $\mathbf{NP}$-complete problem $\mathcal{H}$ and show that $\mathcal{H} \leq \mathcal{L}$. We take $\mathcal{H} =$ **HAMCYCLE** which is $\mathbf{NP}$-Complete and we show that $\mathcal{H} \leq \mathcal{L}$. We need to define $f : \{0,1\}^* \to \{0,1\}^*$ s.t. $x \in$ **HAMCYCLE** $\iff \mathtt{f(x)} \in \mathcal{L}$

  We define f by $f(\mathbb{G}) = (\mathbb{G}, V)$. Obviously, $\mathbb{G} \in$ **HAMCYCLE** $\iff \mathbb{G}$ has a hamiltonian cycle on all vertices $\iff \mathbb{G}$ has a hamiltonian cycle for the subset V of $V \iff (\mathbb{G}, V) \in \mathcal{L}$

- $\mathcal{L} \in \mathbf{NP}$

To do this we give: *CERTIFICATE*: a path in X (the subset of vertices): $w_1, ..., w_n$. We ask that $n < |X| + 1$. *VERIFIER*: input $\rightarrow (\mathbb{G}, X), w_1, ..., w_n$, output $\rightarrow 1 \iff w_1, ..., w_n$ is a path in $\mathbb{G}$ and $\forall i,\ w_i \in X, w_1 = w_n$ and they are all different.

```
for each (w_i, w_i+1) in the sequence
  if (w_i, w_i+1) is not in E, return 0
define array of size |X| initialized with "F" everywhere
for each w_i in the sequence (i != n)
  if A[w_i] == T or undefined then return 0
  else A[w_i] = T
for each w in X
  if A[w] == F, return 0
if w_1 == w_n return 1
return 0
```

This is poly-time because you have $O(|x| + |x| + |x|)$.

@April 4, 2023

Theorem

1. $\leq_p$ is a preorder (reflexive and transitive relation)
2. If a language $\mathcal{L}$ is $\mathbf{NP}$-hard and in $\mathbf{P}$, then $\mathbf{P} = \mathbf{NP}$
3. if a language $\mathcal{L}$ is $\mathbf{NP}$-Complete, then $\mathcal{L} \in \mathbf{P} \iff \mathbf{P} = \mathbf{NP}$

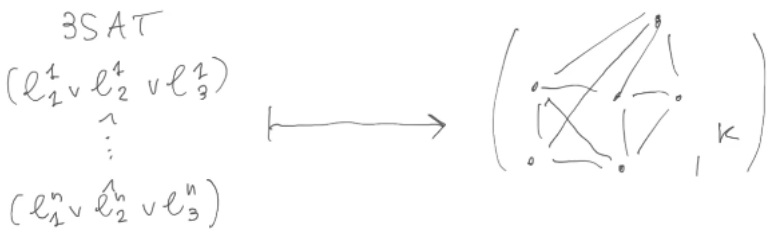
Sketch:

1. Reflexive $\equiv \mathcal{L} \leq_p \mathcal{L}$. This is of course true because we can reduce $\mathcal{L}$ to itself through the identity function $\mathrm{id} : \{0,1\}^* \rightarrow \{0,1\}^*$ such that $\mathrm{id}(x) = x$
   Transitivity $\equiv \mathcal{L} \leq_p \mathcal{H}, \mathcal{H} \leq_p \mathcal{K}$, then $\mathcal{L} \leq_p \mathcal{K}$. Since we know that $\mathcal{L} \leq_p \mathcal{H}$ and $\mathcal{H} \leq_p \mathcal{K}$, we have two reductions f, g, the former witnessing $\mathcal{L} \leq_p \mathcal{H}$, the latter witnessing $\mathcal{H} \leq_p \mathcal{K}$. But then the function $g \circ f$ such that x is mapped to $g(f(x))$ is a reduction of $\mathcal{L}$ to $\mathcal{K}$.
2. Suppose that $\mathcal{L}$ is then $\mathbf{NP}$-hard and in $\mathbf{P}$. Our task is to prove that $\mathbf{P} = \mathbf{NP}$, actually that $\mathbf{NP} \subseteq \mathbf{P}$. We start from any $\mathcal{H} \in \mathbf{NP}$ and we have to prove that $\mathcal{H} \in \mathbf{P}$. Since $\mathcal{L}$ is $\mathbf{NP}$-hard, we can certainly write $\mathcal{H} \leq_p \mathcal{L}$. $\mathcal{H}$ is then at most as difficult as $\mathcal{L}$ which however is in $\mathbf{P}$!! As a consequence, $\mathcal{H}$ is in $\mathbf{P}$ itself.
3. Suppose $\mathcal{L}$ is any $\mathbf{NP}$-complete problem. We want to prove that $\mathcal{L} \in \mathbf{P} \iff \mathbf{P} = \mathbf{NP}$.
   a. $\Rightarrow$) If $\mathcal{L} \in \mathbf{P}$ it means that any language $\mathcal{H} \in \mathbf{NP}$ is such that $\mathcal{H} \leq_p \mathcal{L}$. Since $\mathcal{L} \in \mathbf{P}$, then $\mathcal{H}$ is in $\mathbf{P}$ too. In other words $\mathbf{NP} \subseteq \mathbf{P}$
   b. $\Leftarrow$) Let us assume that $\mathbf{P} = \mathbf{NP}$. We want to prove that $\mathcal{L} \in \mathbf{P}$. However, we know that $\mathcal{L}$ is $\mathbf{NP}$-Complete by hypothesis, meaning that $\mathcal{L} \in \mathbf{NP} = \mathbf{P}$

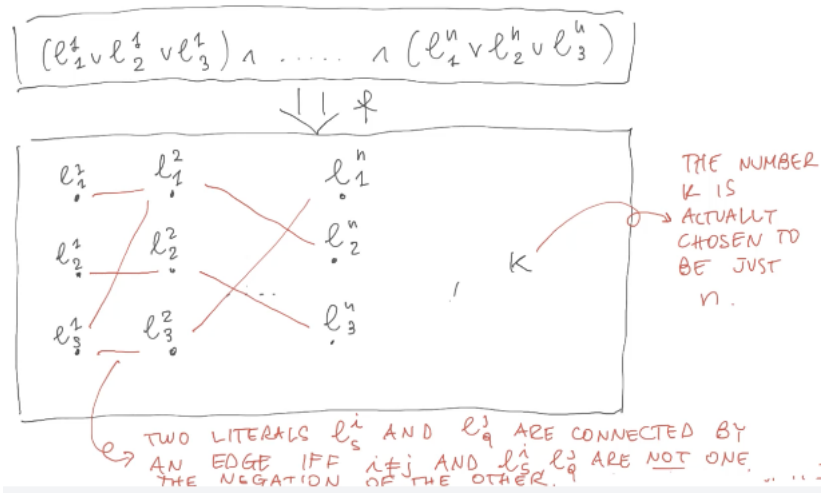
□

▼ *Prove that* `3SAT` $\leq_{\mathrm{p}}$ `CLIQUE`*, namely that* `CLIQUE` *is* $\mathbf{NP}$*-hard*

We have two problems with a distinct nature and we want to reduce the former to the latter.



The translation is defined as follows.



The only thing which is missing is that $x \in \texttt{3SAT} \iff f(x) \in \texttt{CLIQUE}$, where f is the function defined before, which by the way is poly-time computable.

$\Rightarrow$) Suppose that F is a satisfiable 3CNF. Then there is an assignment to the variable in F such that F is true in this assignment i.e. all the clauses of F are satisfied, i.e. for each clause in F there is one literal l_s^i which is satisfied in the assignment. This choice of literals forms a clique of size n in $f(F)$. This is because all of the literals involved must be logically coherent with all the other ones, namely all the corresponding nodes in $f(F)$ must be linked by an edge

$\Leftarrow$) Suppose that the graph $f(F)$ has a clique of size n. This means that all "columns" of $f(F)$ must be involved in the clique. There is thus a way of choosing, for each clause of F a literal which is part of the clique $f(F)$ in such a way that logical consistency is preserved, i.e. we do not have both X and $\neg X$ in the clique. We can then find an assignment of truth values to the variables such that all the literals in $f(F)$ are satisfied, which means that F is satisfiable.

@May 2, 2023

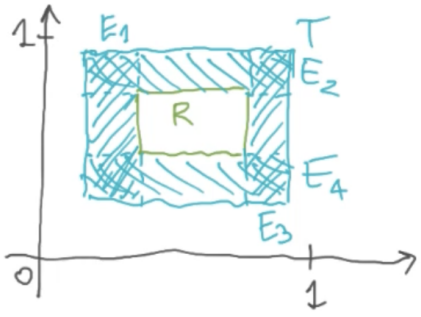
Theorem to Prove
For every distribution $\mathcal{D}$, for every $0 < \varepsilon < 1/2$ and for every $0 < \delta < 1/2$ if $m \geq 5/\varepsilon \ln(4/\delta)$, then:

$$\Pr_{D \sim \mathcal{D}^m}[\text{err}_{\mathcal{D},T(D)}(\mathcal{A}_{BFP}(T(D))) < \varepsilon] > 1 - \delta$$

Let us try to evaluate the following expression: $\text{err}_{\mathcal{D},T}(R) = \Pr_{x \in \mathcal{D}}[x \in (R - T) \cup (T - R)]$. Observe that this probability being below or above a certain threshold is a probabilistic event itself. We are interested in the case in which R is obtained through $\mathcal{A}_{BFT}$. Since $\mathcal{A}_{BFT}$ never produces any errors on negative data, we can write that:

$$\text{error}_{\mathcal{D},T}(R) = \Pr_{x \in \mathcal{D}}[x \in T - R]$$

Another observation has to do with how $T - R$ is structured.



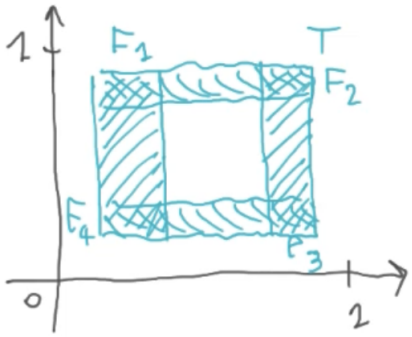
The probabilistic event $x \in E_i$ holds $\iff$ none of the training data ends up being in E_i.

Before starting to play with the probabilities, we might look at other stripes.

We build the F_i in such a way that:

$$\Pr[x \in F_i] = \frac{\varepsilon}{4}$$

Of course we are assuming $\Pr[x \in F_i] > \varepsilon$



We will prove the theorem backward.

$$\begin{gathered}\text{error}_{\mathcal{D},T}(R) < \varepsilon \iff \Pr_{x \in \mathcal{D}}(x \in T - R) < \varepsilon \\ \Leftarrow \Pr_{x \in \mathcal{D}}(x \in E_i) < \frac{\varepsilon}{4} \; \forall i\end{gathered}$$

This first implied ($\Leftarrow$) is obtained by the Union Bound: $P(A \cup B) \leq P(A) + P(B)$

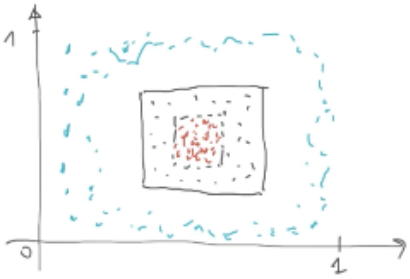
Let us consider each of the four stripes separately.

$$\begin{gathered}\Pr_{x \sim \mathcal{D}}(x \in E_i) < \frac{\varepsilon}{4} \iff \Pr_{x \sim \mathcal{D}}(x \in E_i) \leq \Pr_{x \sim \mathcal{D}}(x \in F_i) \\ \Leftarrow E_i \subseteq F_i \\ \Leftarrow \text{"Some point in the training data occurs in } F_i\text{"}\end{gathered}$$

We can finally look at the statement of the theorem itself:

$$\begin{gathered}\Pr_{D \sim \mathcal{D}^m}[\text{err}_{\mathcal{D},T(D)}(R) < \varepsilon] > 1 - \delta \Leftarrow \Pr_{D \sim \mathcal{D}^m}(\forall i. \text{"some points in the training data occur in } F_i\text{"}) > 1 - \delta \iff \\ \Leftarrow 4 \cdot (1 - \frac{\varepsilon}{5})^m \leq \delta \\ \Leftarrow 4 \cdot (e^{-\frac{\varepsilon}{5}})^m \leq \delta \Leftarrow \ln(4 \cdot (e^{-\frac{\varepsilon}{5}})^m \leq \ln \delta \\ \ln 4 + \ln(e^{-\frac{\varepsilon}{5}})^m \leq \ln \delta \\ \ln e^{\frac{\varepsilon m}{5}} \geq \ln \frac{4}{\delta} \\ \frac{\varepsilon m}{5} \geq \frac{4}{\delta} \\ \iff m \geq (\ln \frac{4}{\delta}) \cdot \frac{5}{\varepsilon}\end{gathered}$$

And we re-obtained the hypothesis. The chain of left implications



Structure of the proof on the negative result on 3DNFs

Theorem
If $\mathbf{NP} \neq \mathbf{RP}$, then the representation class of 3-Term DNFs is not efficiently PAC-learnable

This is a negative result, and we do not have to come up with an algorithm, but rather with a… reduction. The reduction we need to come up with is the following:

$$\alpha \in \{0,1\}^* \xmapsto{F} S_\alpha$$

An instance of a $\mathbf{NP}$-complete problem A set of training data for 3-Term DNFs

We have to make sure, when building, that an hypothetical algorithm for 3-DNF, when applied to S_α, outputs a concept from which an answer to $\alpha \overset{?}{\in} A$ can be answered. The kind of NP-Complete problem we need is 3-colorability problem for graphs.

@May 8, 2023

Complete Exam

▼ Multiple Choices questions

- Let f, g be the functions defined as $f(n) = 1n\log(n)$ and $g(n) = \frac{1}{n*\log(n)}$. Possible answers:

  A) $f \in \Theta(g)$
  B) $f \in \Omega(g)$
  C) $f \in O(g)$

  Compute $\frac{f(n)}{g(n)}$ = n^2. The limit for $n \to \infty$ is ∞. We then can exclude A and C. Answer is B

- Turing Machines. Possible answers

  A) Are such that certain problems that can be computed in time n^2 cannot be computed in time n.
  B) All work in polynomial time
  C) Can contain an arbitrary number of working tapes
  D) All work in exponential time

  A) is True because for example sorting can be done in n^2 but not in n. B) is False as there's no limit on the time of a Turing Machine. C) is True. D) is False

- The Halting problem is:

  A) Undecidable, but NP-Hard
  B) Nothing about its computational complexity is known
  C) Decidable in Polynomial time
  D) NP-Complete

  The Halting Problem is known to be undecidable, so this means that there is no Turing Machine that can solve this problem. The question hides a bit of a trap, but the answer is easily A) (All undecidable problems are more difficult than NP → so NP-Hard). The Halting Problem is not in NP, nor in P.

- Suppose that a language $\mathcal{L}$ is both in P and NP-Complete. Then:

  A) The classes EXP and P are different
  B) The classes EXP and NP are equal
  C) The classes P and NP are different
  D) The classes P and NP are equal

  If $\mathcal{L}$ ∈ P and $\mathcal{L}$ is NP-hard, it means NP⊆P. In this case, P=NP. We should know that P≠EXP, always. So A) is true. And also, NP ≠EXP.

- The notion of PAC-Learnable Concept Class:

  A) Does not make any reference to the complexity time of the learning algorithm
  B) Needs to hold for every distribution D on the instance class
  C) Cannot be reached when the underlying concept class is the conjunction of literals
  D) Requires the output concept to have probability of error ε in all cases

  Instance space X, Concept class $\mathcal{C} \subseteq P(X)$. I give in input to an algorithm $\mathcal{A}$ an oracle $EX(C, D)$ which gives out data from a real distribution, ε and δ and I get out a hypothesis $h \in \mathcal{C}$. PAC learnable $\iff \exists \mathcal{A}. \forall c \in \mathcal{C}, \forall 0 < \varepsilon < \frac{1}{2}, \forall 0 < \delta < \frac{1}{2}$ we have $\Pr[\text{error}_{C,D}\mathcal{A}(EX(C, D)\epsilon, \delta) < \epsilon] > 1 - \delta$

  We can see directly that A) is true and B) is true

▼ Problems

- Construct a Turing Machine, of the kind you prefer, which decides the following language: $\mathcal{L} = \{W \in \{0,1\}^* | \text{Every } 1 \text{ occurring in } W \text{ is immediately followed by a } 0\}$. Study of the the complexity of the TM you defined.

I work with a TM with only the input tape and two halting states, q_{accept} and q_{reject} for accepting or rejecting a word.

$\Gamma = \{\triangleleft, \square, 0, 1\}, Q = \{q_{rej}, q_{acc}, q_r, q_0\}$. The initial state is q_r.

$\delta(q_r, \triangleleft) = (q_r, R)$ we start reading the input - $\delta(q_r, 0) = (q_r, R)$ we continue - $\delta(q_r, 1) = (q_0, R)$ we need to check that the following letter is a 0, so we go to q_0 - $\delta(q_r, \square) = q_{acc}$ we successfully read all the input

$\delta(q_0, 0) = (q_r, R)$ we saw a 0, we can continue - $\delta(q_0, 1)/\delta(q_0, \square) = q_{rej}$ we reject because there is a 1 not followed by a 0 (also the blank is rejected).

The complexity of this TM is $O(n)$ - linear.

- You are required to prove that the following problem is in FP. To do that, you can give a TM or define some pseudo-code. The function is the one that, given two lists, $L = L_1, ..., L_n$ and $P = P_1, ..., P_n$ computes the scalar product $L_1 \times P_1 + ... + L_n \times P_n$.

```
R <- 0
for i = 1 to n do:
  hd1 <- L.pop()
  hd2 <- P.pop()
  R<- R + hd1xhd2
  i++
return R
```

This is in polynomial time. There are $O(n)$ instructions = $4n + 2$ (exactly). $|i| \leq |n|$ and $|hd1 \times hd2| \leq |hd1| + |hd2|$. $|R| \leq O(|L_1| + |P_1| + ...)$. So $|R|$ is polynomial. The instructions I use (pop, sum and multiplications) are poly-time computable.

- We studied **SAT** and **3SAT**. You are required to classify the problem **1SAT** containing the satisfiable 1CNFs. To which complexity class does **1SAT** belong? Prove your answer

A formula in 1CNF has the shape $l_1 \wedge l_2 \wedge ... \wedge l_n$ (because each disjunctive clause has at most 1 literal). Such a formula is satisfiable if and only if there is no contradiction $x_i \wedge \neg x_i$. We give a polynomial time algorithm. Input: list $l_1, ..., l_n$ of literals. Output: return 1 iff there is no contradiction in the list of literals.

```
X <- array(n, undefined) # an array of n undefined values
for i = 1 to n do:
  if l[i] = Xi then:
    if X[i] = False then return 0
    else X[i] <- true
  if l[i] = !Xi then:
    if X[i] = True then return 0
    else X[i] <- False
return 1
```

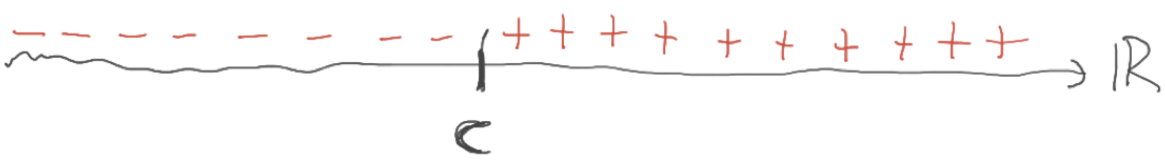
Linear number of instructions with $|X| \leq n$ by construction. All instructions are polynomial-time computable.

To conclude, **1SAT** is in $\mathbf{P}$.

Exercise on PAC-Learnability

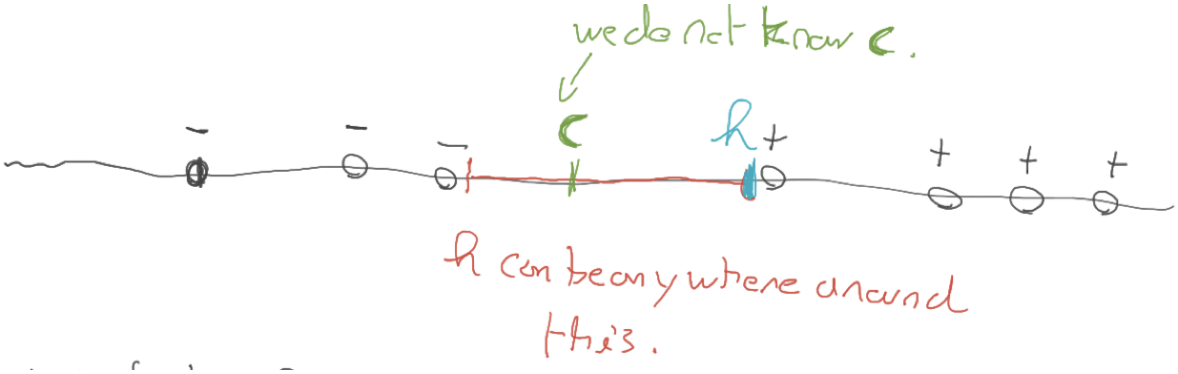
A concept class $\mathcal{C}$ is Efficiently PAC-Learnable if $\mathcal{A}$ is polynomial in $\frac{1}{\varepsilon}$, $\frac{1}{\delta}$, size(c) and calls to $EX(c, D)$

Instance Space $X = \mathbb{R}$. Concept class: Positive half lines



We need to define an algorithm $\mathcal{A}$ that takes in input $EX(c, D), \varepsilon, \delta$ and returns the hypothesis.

If we are given some data:



We take h the greatest possible sensible hypothesis.

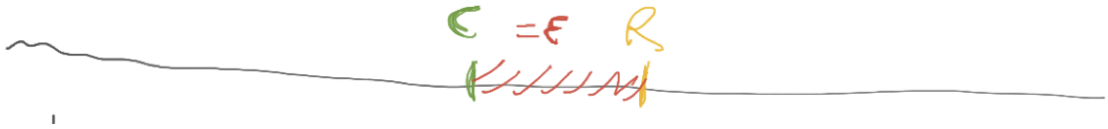
Let us look at the probability of error!

Here, the hypothesis h makes error with regard to c.

So $\Pr[\text{error}_{c,D}(\mathcal{A})] = \Pr_D[[c,h]]$

1. $\Pr_D[[c,h]] < \varepsilon$, then we successfully learned the concept class c.
2. $\Pr_D[[c,h]] \geq \varepsilon$.

   We define a real number R such that:



   We define R such that $\Pr_D[[c,R]] = \varepsilon$. If a training data point would have fallen into $[c,R]$, then we would have $\Pr[\text{error}_{c,D}(h)] < \varepsilon$.

   $\Pr[\text{err}_{c,D}(h) \geq \varepsilon] \leq \Pr[\text{no training data falls into } [c,R] \leq \Pr[d_1 \notin [c,R] \wedge ... \wedge d_m \notin [c,R] \Pr[\text{err}_{c,D}(h) \geq \varepsilon] \leq \Pr[\text{no training data falls into } [c,R] \leq \Pr[d_1 \in [c,R] \wedge ... \wedge d_m \in [c,R] \text{ (i.i.d)} \leq \Pr[d_1 \notin [c,R]] \times ... \times \Pr[d_m \notin [c,R]] \leq (1-\varepsilon)^m$

   You have always $1 + x \leq e^x$ for all x. So $(1-\varepsilon)^m \leq e^{-m\varepsilon}$

   $\Pr[\text{err}_{c,D}(h) \geq \varepsilon] \leq e^{-m\varepsilon}$. We want $\Pr[\text{err}_{c,D}(h) \geq \varepsilon] \leq \delta$. It is sufficient to have $e^{-m\varepsilon} \leq \delta$. $-m < \frac{1}{\varepsilon} \log \delta$. $m \geq \frac{1}{\varepsilon} \log \frac{1}{\delta}$.

We just proved that we need more than $\frac{1}{\varepsilon} \log \frac{1}{\delta}$ data points for training. Thus, $\mathcal{A}$ is polynomial in $\frac{1}{\varepsilon}, \frac{1}{\delta}$

We can conclude that positive half lines is efficiently PAC-Learnable

For learning rectangles in $\mathbb{R}^2$, the number of data points needed was $\frac{4}{\varepsilon} \ln \frac{4}{\delta}$