Attention, j'ai réordonner les slides.

Pb de décision :

Π = " Soit n un entier, est-ce que n est pair ? "

n un entier : entrée

est-ce que n est pair ? : question fermée

) Version informelle du pb

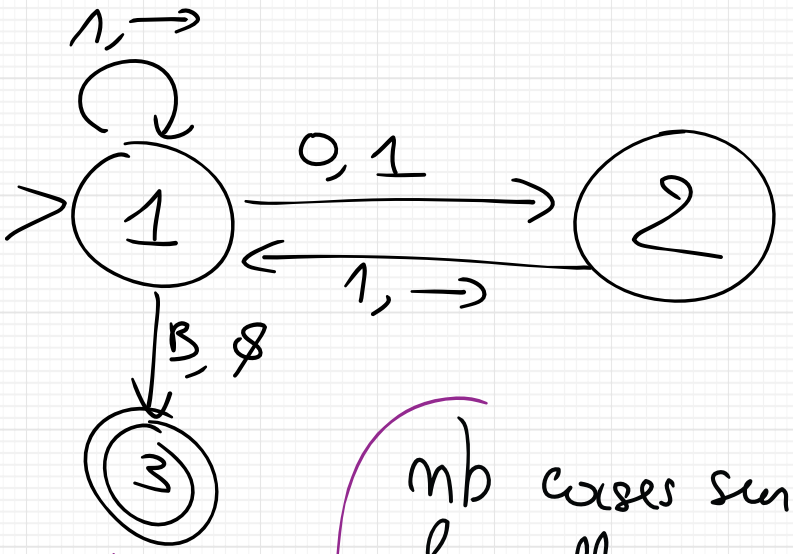
$\mathcal{L} = \{ \text{entiers} \} = \mathbb{N} = (1|0)^*$

$\mathcal{L}^V = \{ \text{entiers pairs} \} = (1|0)^* 0$

$\mathcal{L}^N = \{ \text{entiers impairs} \} = (1|0)^* 1$

} Version formelle

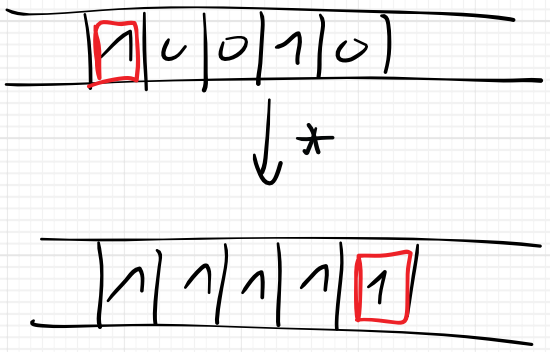
Complexité



Existé en espace $S(n, z)$

nb cases sur lesquelles on écrit

$= nb\ 0 = O(n)$



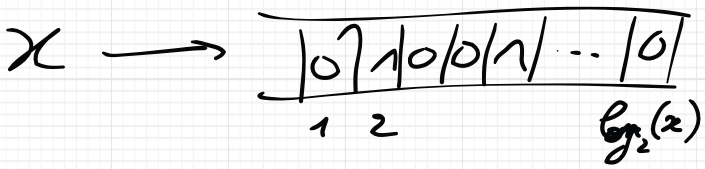
nombre de transitions :

$nb\ 0 \times 2 + nb\ 1 + 1$

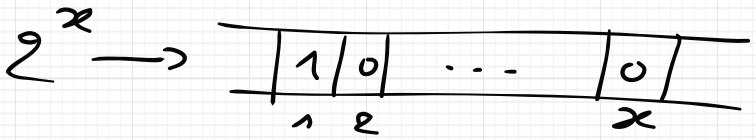
$= n + nb\ 0 + 1 = O(n)$

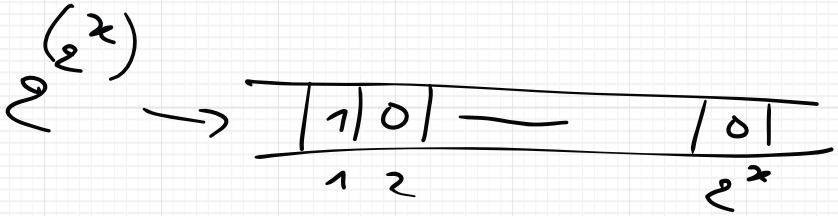
on touche à l'entrée

Existé en temps $t(n, z)$



Taille d'un entier !





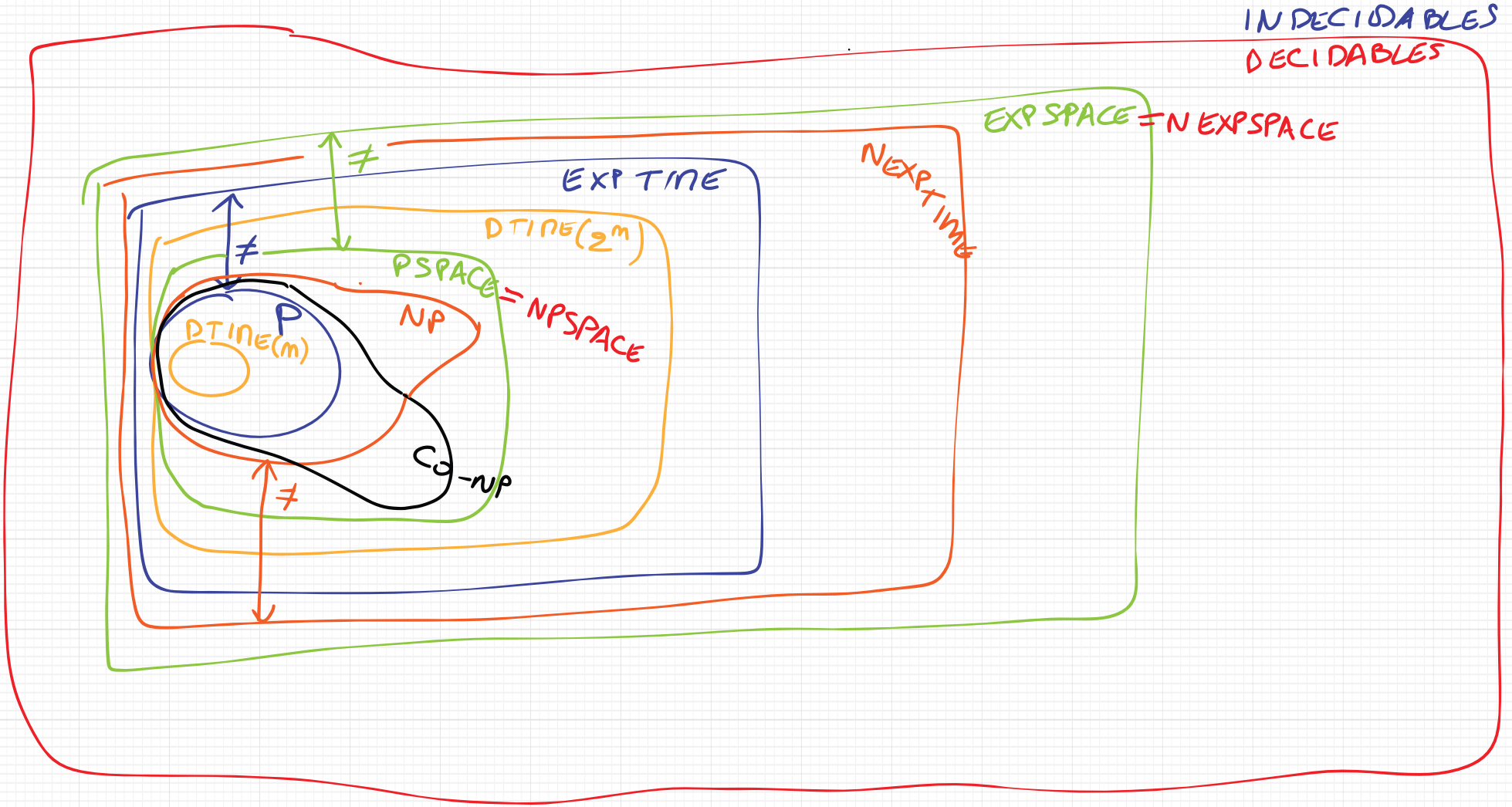
La taille d'un entier dépend (comme toutes les entrées) de la manière dont on encode l'entier. Classiquement, on différencie 2 types d'encodage:
- binaire (avec des 0 et des 1)
- unaire (avec juste un symbole baton " | " , par exemple 4 = ||||, 7 = |||||||)

L'avantage du binaire est qu'il est compact. La taille d'un entier x en binaire est log(x), et celle en binaire est x. C'est à dire que si on encode en binaire, l'entier prend exponentiellement plus de place.

Si j'ai une complexité f(x) qui dépend de la taille de x, passer en unaire va donc diminuer exponentiellement la complexité.

Par exemple si f(x) = O(x^2) ; alors en unaire la complexité est quadratique, mais en binaire elle est exponentielle car, en réécrivant cette complexité comme fonction de la taille de x (c'est à dire log(x)), on obtient O(2^(2 log(x))).

On peut se dire que c'est une vision de l'esprit puisque x = 2^log(x), le temps de calcul sur la machine est le même. Pourquoi s'intéresser à ça et ne pas tout encoder en unaire?
- l'espace occupé est moindre en binaire, on va donc préférer encoder en binaire quand c'est possible; et donc se référer au binaire. On peut considérer qu'être obligé de passer à l'encodage unaire pour améliorer artificiellement la complexité est un échec.
- Il existe des problèmes qui sont NP-Complets (cf la fin du cours) quand l'entrée est binaire ; et polynomiaux sinon.
Il existe également des problèmes qui sont NP-Complets tout le temps, quelque soit l'encodage. Il est clair que la seconde catégorie est plus difficile que la première.



Attention, contrairement à ce que le dessin peut laisser penser, il me semble qu'on ne sait pas où se situent NP et PSPACE par rapport à DTIME(2^n).

Il est possible que PSPACE = NP et aussi que PSPACE = EXPTIME.
Tout comme il est possible que PSPACE != EXPTIME mais que PSPACE et DTIME(2^n) ne se contiennent pas mutuellement

On va très souvent ignorer les paramètres logarithmiques.

Π_1 : "Soient $x_1 - x_n, B \in \mathbb{N}$ (taille $\sum \log(x_i) + \log(B)$ → précis mais pas souvent utile) en n → imprécis, est-ce que $\sum_{i=1}^{n} x_i = B$?" → $\mathcal{O}\left(n \left(\times \log\left(\max_{B} \sum x_i\right)\right)\right)$ → $\Pi_1 \in P$

Facultatif la plupart du temps

"Soient $x_1 - x_n, B \in \mathbb{N}$, est-ce qu'il existe $I \subset [\![1, n]\!]$ tq $\sum_{i \in I} x_i = B$?" Tout tester : $\mathcal{O}(2^n)$ → $\Pi_2 \in$ EXPTIME

(ici par exemple p, on les a ignoré)

Une machine ND choisit I (de manière ND) et on vérifie si $\sum_{i \in I} x_i = B$ → en temps $\mathcal{O}(n)$ → $\Pi_2 \in$ NP

Π_3 : "Soient $x_1 - x_n, B \in \mathbb{N}$, est-ce que, pour tout $I \subset [\![1, n]\!]$, $\sum_{i \in I} x_i \neq B$?"

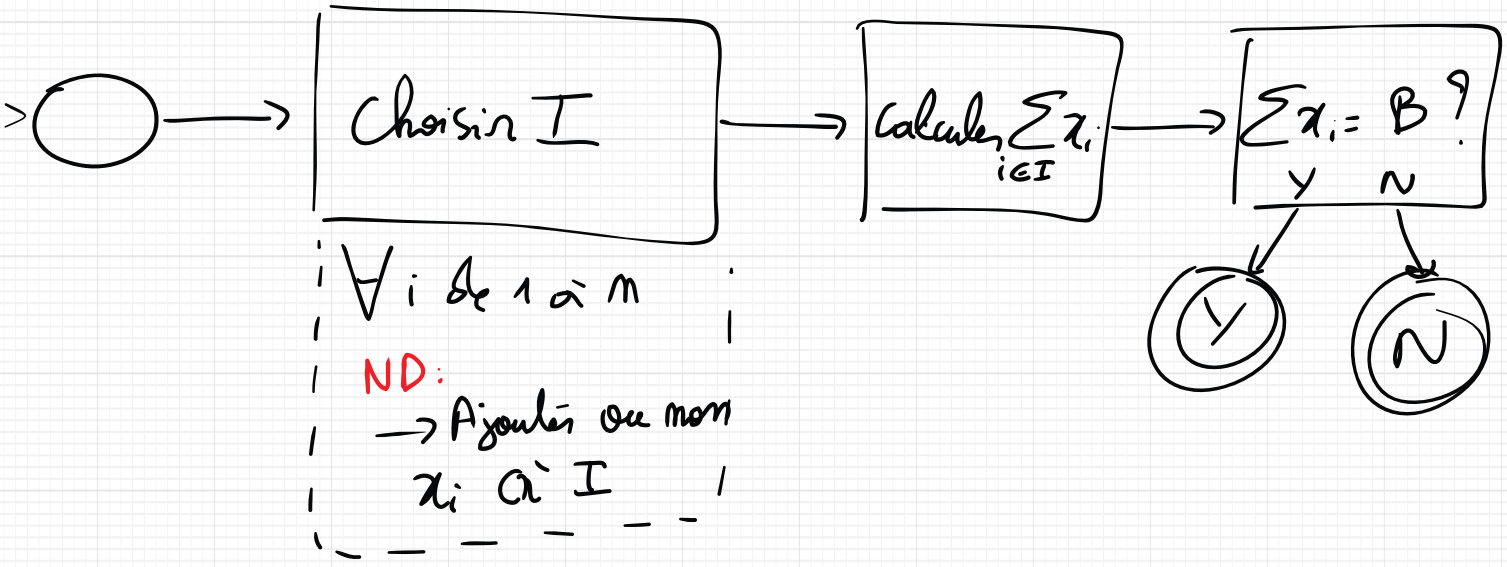
Tout tester → $\mathcal{O}(2^n)$ → $\Pi_3 \in$ EXPTIME

(là aussi)

$\Pi_3 = Co\text{-}\Pi_2$ (Inverse Yet N) → $\Pi_3 \in Co\text{-NP}$

Pi_3 est le problème Pi_2 où on a inversé les réponses OUI et NON. Puisque PI_2 appartient à NP, PI_3 appartient à Co-NP.

Machine non déterministe pour résoudre PI_2

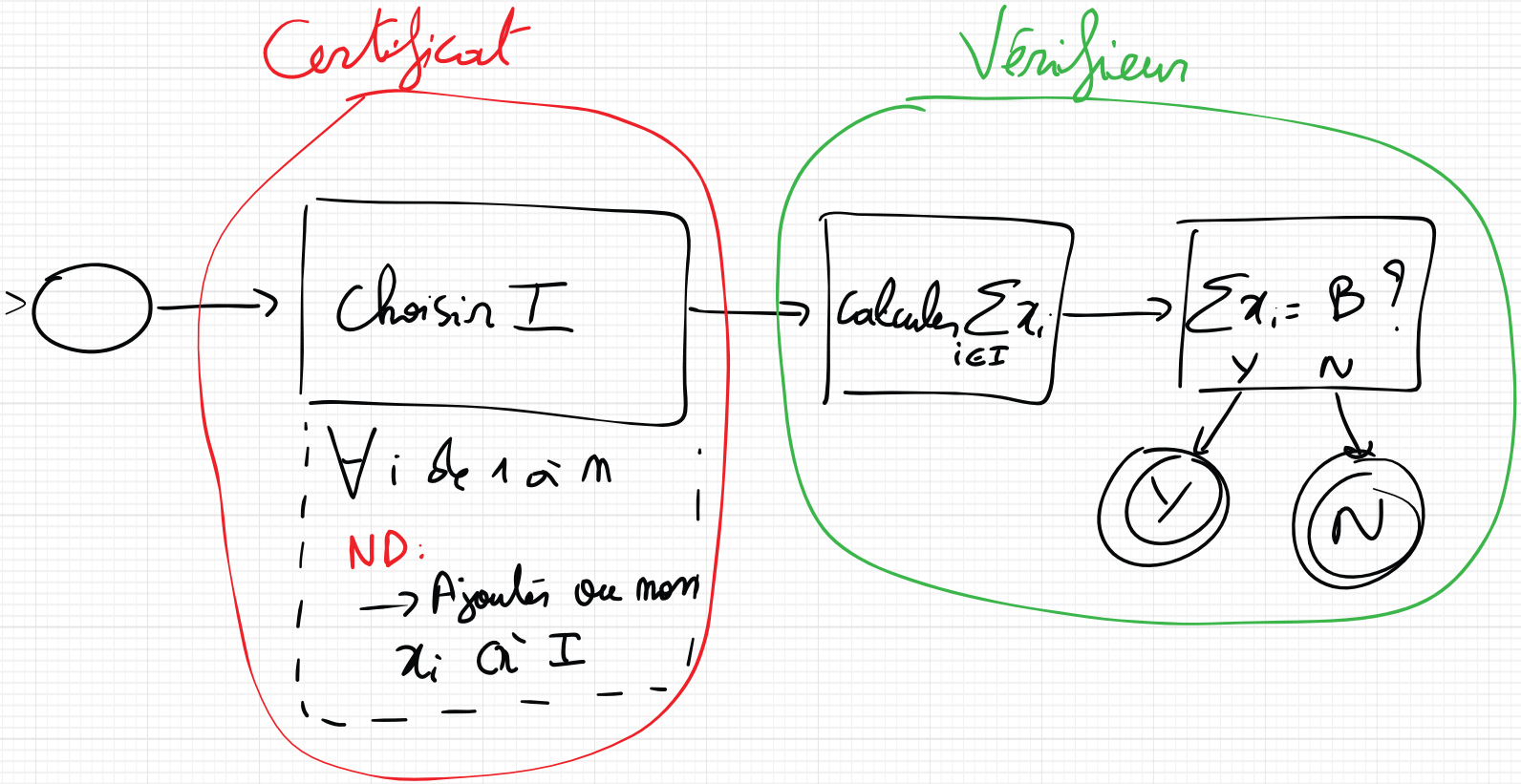


NP : définition bis

" On peut vérifier une solution en temps polynomial ") Informelle

$\Pi \in NP \Leftrightarrow$ " $\exists$ algo V qui prend en entrée
- une entrée x de Π
- un mot c appelé certificat

tq
- Si $x \in \mathcal{L}_Y$, $\exists$ certificat c de taille poly / $V(x,c) = 1$
- Si $x \in \mathcal{L}_N$, $\forall$ certificat c de taille poly / $V(x,c) = 0$
- V est polynomial. "



C'est la même machine que plus haut, où j'indique quelle partie correspond au certificat et quelle partie correspond au vérifieur dans la définition précédente.

Remarque : il n'est pas toujours possible de créer une machine aussi simple. On peut souvent le faire.

(3-SAT) " Soit φ une conjonction (et) de disjonction (ou) où chaque clause a 3 littéraux,

Clauses : C1, C2, ..., Cm

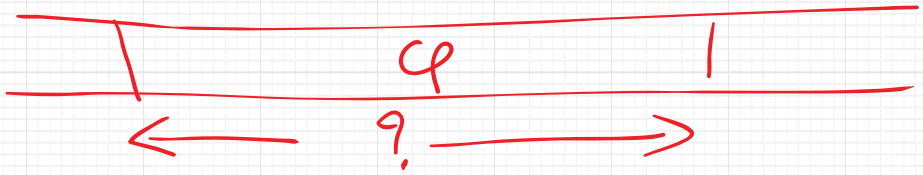
$$\varphi = C_1 \wedge C_2 \wedge \dots \wedge C_m \qquad C_i = (l_1^i \vee l_2^i \vee l_3^i)$$

$$l_i \in \{x_1 - x_n \; \overline{x_1} - \overline{x_n}\}$$

$\exists$? une affectation des variables $x_1 - x_n$ qui rende φ vraie ? "

↳ φ satisfiable ?

taille des entrées de (3-SAT)



Chaque clause a 3 littéraux. Une formule est donc 3m littéraux côte à côte.

$$|\varphi| = |(\quad)_1 \wedge (\quad)_2 \wedge \cdots \wedge (\quad)_m| = m \times 3 \times \boxed{|x_i| \text{ ou } |\overline{x_i}|}$$

Juste du "0-padding"
On complète avec des
0 à gauche pour que tous les
identifiants soient de même
taille.

$$\sim \log(m) + 1$$

$$0 \cdots 0 \log(i) + 1$$

$$\log(m) - \log(i)$$

Bit de "signe"
0 pour x_i
1 pour (x_i barre)

Autre méthode :
une matrice A de m lignes et n colonnes
A[i][j] = 1 si la clause Ci contient x_j
A[i][j] = 0 si la clause Ci contient (x_j barre)
A[i][j] = -1 si la clause Ci ne contient ni x_j ni (x_j barre)

Taille : m * n * 2 (il faut 2 bits pour représenter 1, 0 ou -1)

Exemple $\varphi_1 = (x \vee y \vee z) \wedge (\bar{x} \vee x \vee z) \wedge (y \vee \bar{z} \vee \bar{t})$

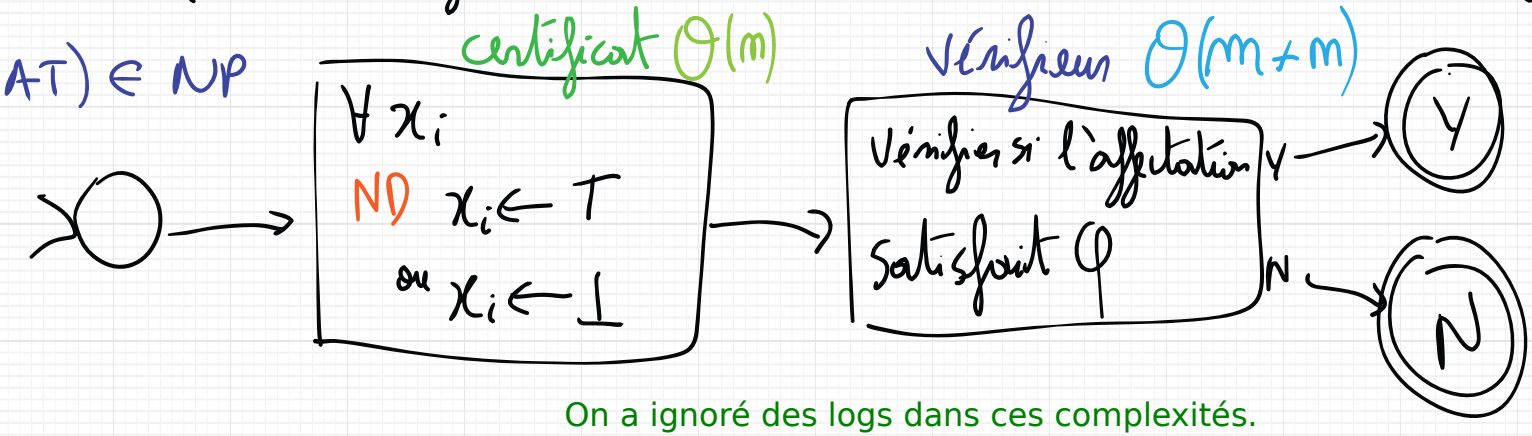
φ_1 est satisfiable — preuve : $x = \top$ (VRAI) $z = \bot$ (FAUX) $y = \top$ $t = \bot$ fonctionne

Se demander si, quand la réponse est OUI, on peut facilement s'en convaincre est un bon début pour comprendre pourquoi un problème est dans NP.

$\varphi_2 = (x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{x} \vee y) \wedge (\bar{x} \vee \bar{y})$

φ_2 non satisfiable — preuve : si on teste tout, on voit que ça ne fonctionne pas.

$(3SAT) \in NP$



On a ignoré des logs dans ces complexités.

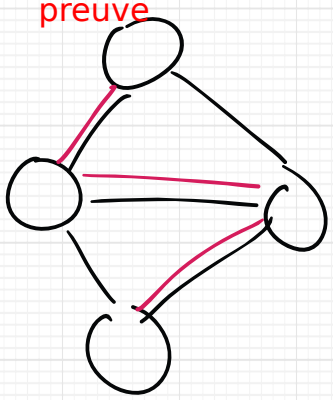
(HAM) "Soit G un graphe non orienté, est-ce que G est hamiltonien ?"

$G=(V,E)$ $|V|=n$ $|E|=m$

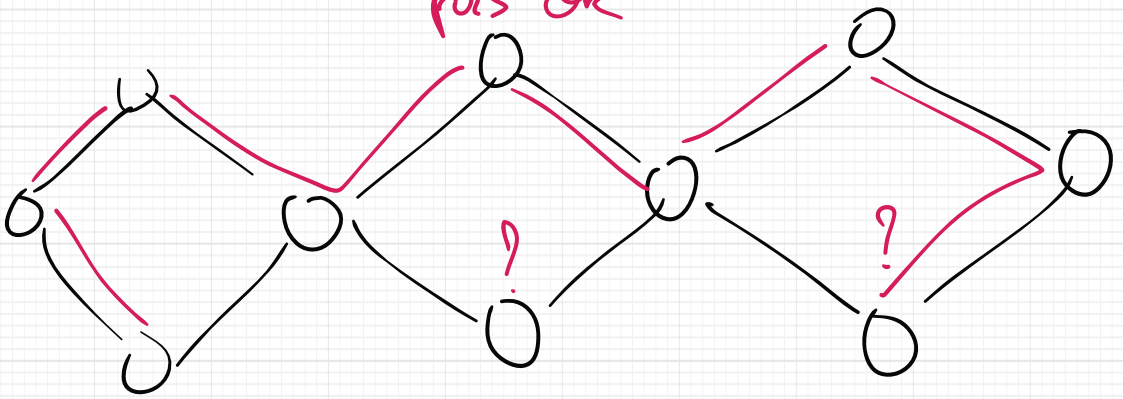
↳ ∃ une chaîne qui passe par tous les noeuds une fois !

ok

preuve

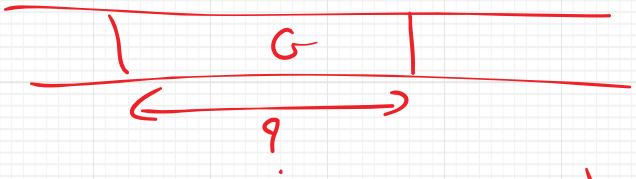


pas ok



preuve : faut tout tester

Taille des entrées de (HAM)



$E = \{(u,v)\} \longrightarrow m \times 2 \times \log_2(n) + \log_2(n)$

(nb arête ; identifiées par un couple de nœuds ; nb sommets)

ou

matrice d'adjacence $\longrightarrow n \times n \longrightarrow A = \begin{pmatrix} & u \\ \vdots & \\ \cdots & 1 \end{pmatrix}$ (u)

ou

matrice d'incidence $\longrightarrow n \times n \longrightarrow I = \begin{pmatrix} & u & v \\ \vdots & & \\ \cdots & 1 & \cdots 1 \cdots \end{pmatrix}$ ((u,v))

ou

listes d'adjacence $\longrightarrow \underbrace{\log(n)}_{\times m} + \underbrace{\log(n)}_{\times m}$

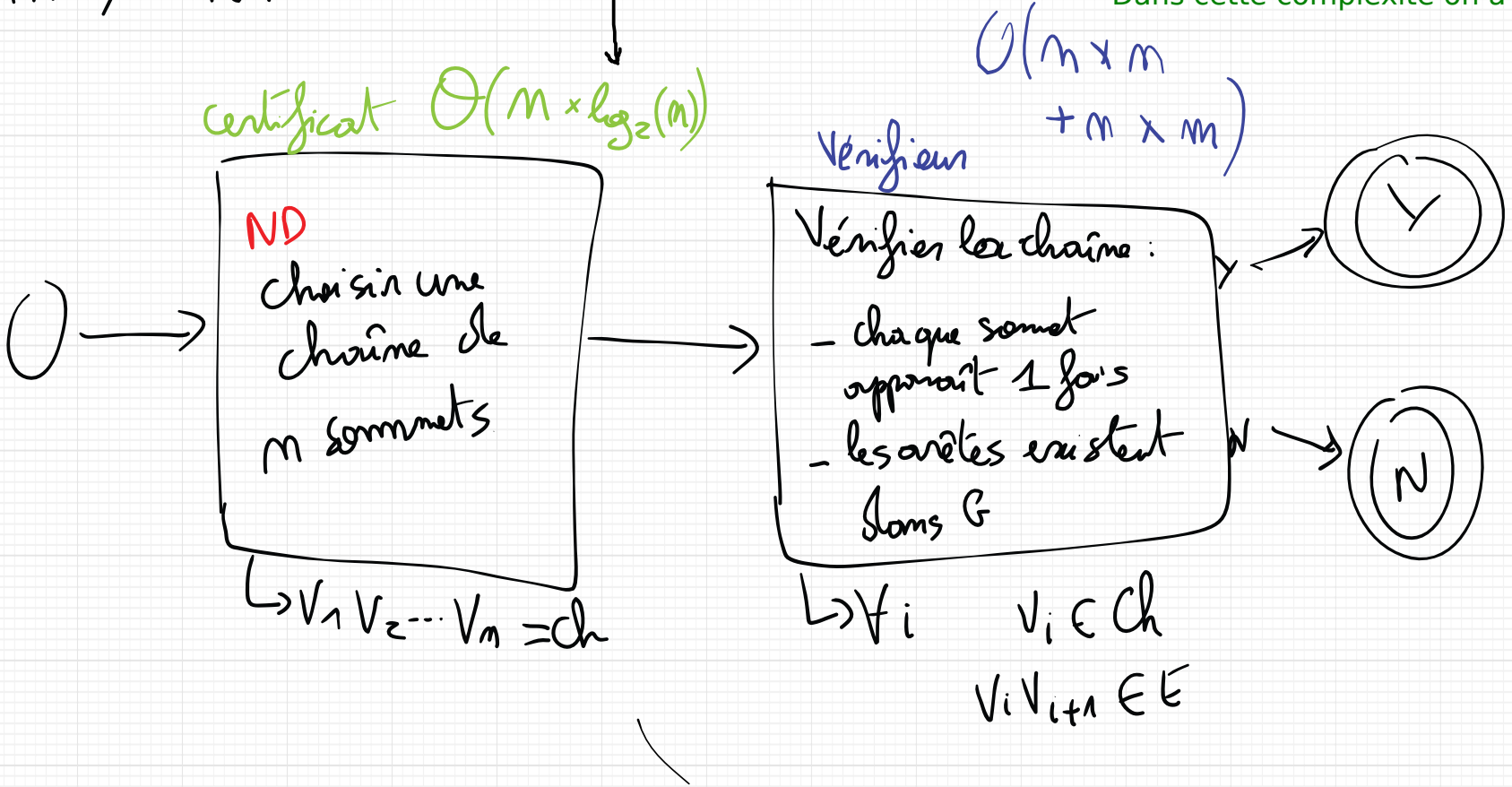
$\longrightarrow$ u : [Voisins de u]
v : [Voisins de v]
⋮

Toutes ces manières de représenter un graphe sont équivalentes dans le sens où quelle que soit celle qu'on utilise la complexité n'explose pas. (contrairement aux entiers quand on écrit en binaire ou en unaire, cf page 3

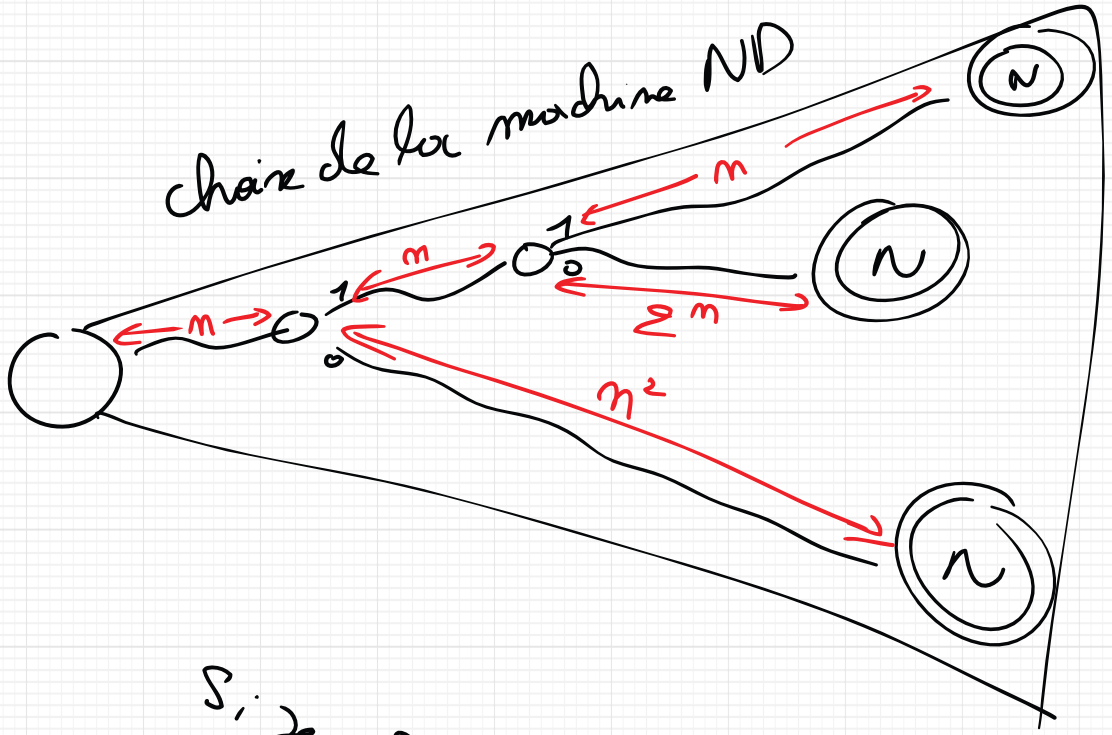
$(HAM) \in NP$

Facultatif :
Pour chaque noeud, on écrit son indice dans ch.

Dans cette complexité on a ignoré le log.



Ce slide rejoint mon explication de pourquoi la complexité d'une machine non déterministe devrait être 'max max' et non 'min min', cf la page 5 du cours



$C = 11 \rightarrow V(x, 11) = N$ → temps n

$C = 10 \rightarrow V(x, 10) = N$ → temps $\leq 2^n$

$C = 0 \rightarrow V(x, 0) = N$ → temps n^2

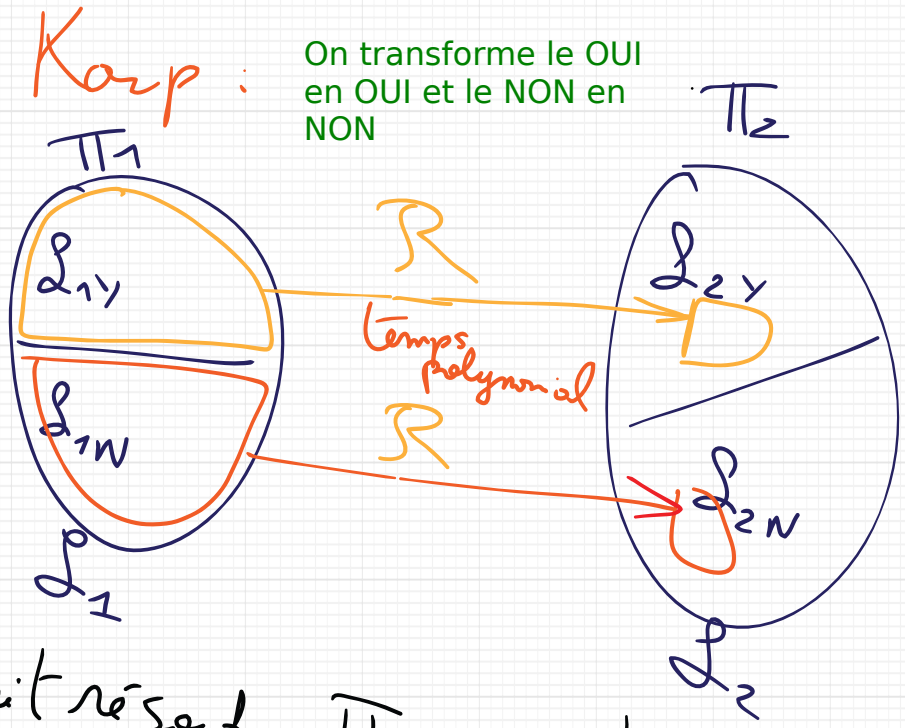
Si $x \in \int_N$ (si la répons est NON)

$\Pi_1 \leq \Pi_2$

(est plus facile)

Π_1 est dur $\Rightarrow$ Π_2 est dur.

Karp : On transforme le OUI en OUI et le NON en NON



Si on sait résoudre Π_2, on peut résoudre Π_1

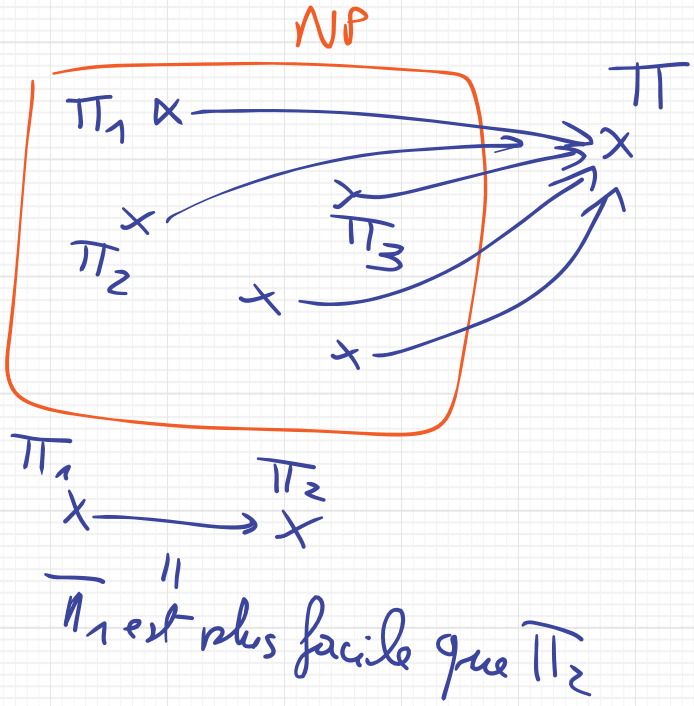
$x \in \mathcal{L}_{Y_1} \Leftrightarrow R(x) \in \mathcal{L}_{Y_2}$

Je pense qu'on appelle ça une "réduction" parce qu'on voit que les images de L1Y et L1N sont plus petites que L2Y et L2N.

En toute rigueur, c'est cette image de L1Y et L1N qui est plus dure que Pi1, car, juste avec ces instances, on peut résoudre PI1. On devrait donc dire que PI1 est plus facile qu'une partie de PI2, une partie "réduite".

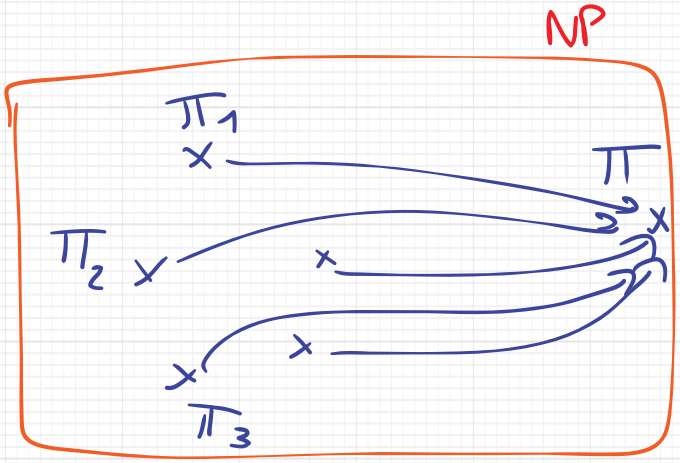
NP - Difficile

Un problème NP - difficile est plus dur que tous les problèmes de NP



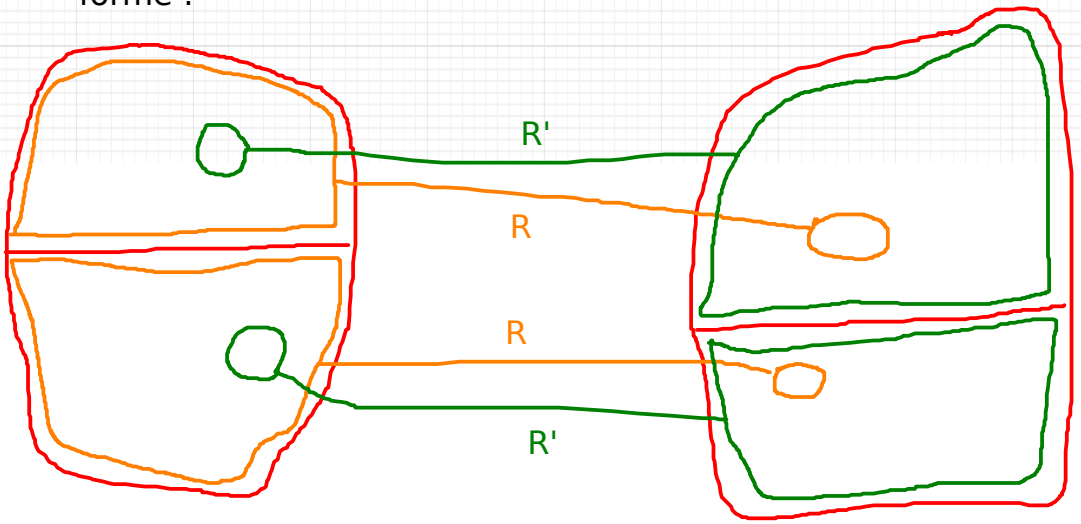
NP - Complet

= Difficile + appartient à NP



Un problème NP-Complet est aussi plus dur que tous les problèmes de NP donc il est "le problème le plus dur de NP".

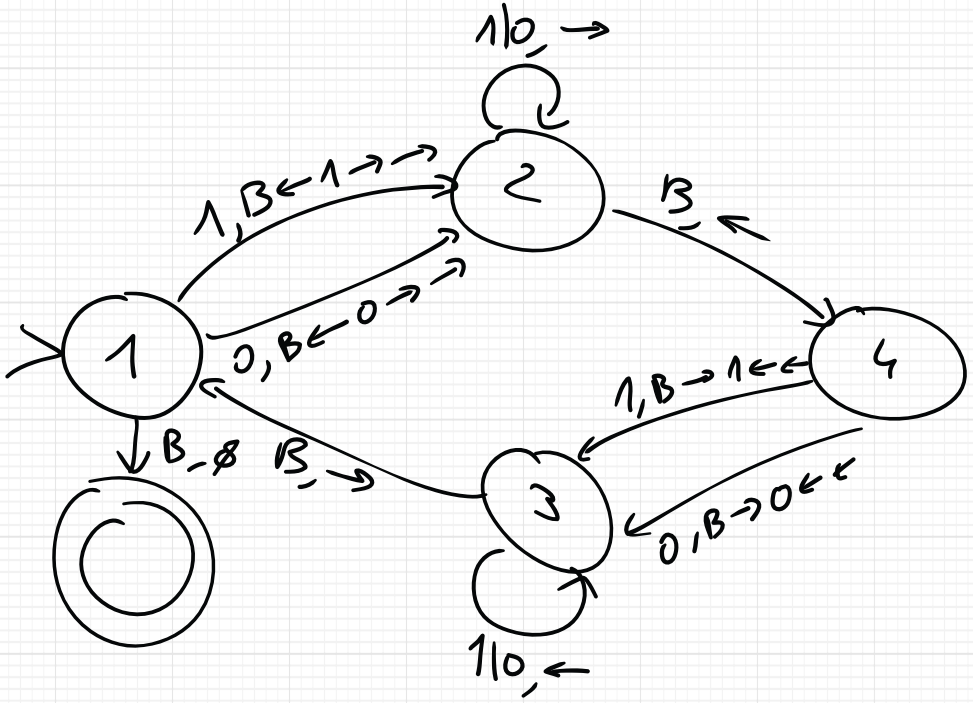
Il existe plus qu'un problème NP-Complet.
Par exemple, 3-SAT et HAM sont NP-Complet. Ca implique que 3-SAT est à la fois plus dur et plus facile que HAM. On a donc deux réductions R et R' sous la forme :



Et ça, ça veut dire qu'on peut réduire 3-SAT à lui même avec R(R') et ça veut dire qu'il existe un sous-ensemble des instances de 3-SAT qui sont plus dure que l'ensemble des instances de 3-SAT.

Et ça c'est fun.

Interro : solution :



4