Complexité algorithmique

par Natacha Portier

Contents

Cours

Partie 1.1

Les machines de Turing

Définition classique : avec 1 ruban infini d'un seul coté.

Définition : Une machine a k rubans et un triplet (Γ, Q, δ) ou Γ est l'alphabet, Q l'ensemble des etats et δ la fonction de transition.

Γ contient 0, 1, $\triangleright$ (symbol de début de ruban) et le blanc.

$$\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{-1; 0; 1\}^k$$

Le 1er ruban est le ruban d'entrée et on n'écrit pas dessus, le $k$eme ruban est le ruban de sortie. Q contient un état q_{start} et un état q_{halt}

Définition : Si $f : \{0;1\}^* \to \{0;1\}^*$ et $T : \mathbb{N} \to \mathbb{N}$ et M une machine de Turing, on dit que M calcul f si pour tout $x \in \{0;1\}^*$, en partant d'une configuration ou x est sur le 1er ruban (veut dire $\triangleright x$) et ou on est dans l'état q_{start}, le calcul se termine avec $f(x)$ sur le ruban de sortie. On dit que M calcul calcul en temps $T(x)$ si pour tout x le calcul de M sur x se termine en au plus $T(|x|)$

Définition : $T : \mathbb{N} \to \mathbb{N}$ est constructible en temps s'il existe une machine M qui calcul l'écriture en binaire de $T(n)$ en temps $\mathcal{O}(T(n))$

Définition : Machine à compteur A partir d'une machine M et d'une fonction "compteur" $T : \mathbb{N} \to \mathbb{N}$ on peut fabriquer une machine M' constructible en temps qui calcul la même chose que M (si M s'arrête) ou qui s'arrête (si M ne s'arrête pas) en temps $\mathcal{O}(T(n))$.

Codage des machines de Turing A Chaque machine on attribue un mot de $0; 1^*$ de manière a ce que :

- tous les mots sont des codes de machines
- chaque machine est représenté par une infinité de mots.

1.1.1

Machine universelle efficace

Théorème : Il existe une machine U qui sur l'entrée $< \alpha, x >$ calcul $M_\alpha(x)$. De plus si le calcul $M_\alpha(x)$ s'arrête en T pas de calcul, alors le calcul de $U(< \alpha, x >)$ s'arrête en $CT \log T$ pas de calcul ou C ne dépend que de la taille de l'alphabet de M_α, du nombre d'états de M_α et du nombre de ruban de M_α

De plus, on peut choisir U avec 2 ruban et oublieux, i.e. le déplacement de la tête de lecture ne dépend pas de l'entrée.

Preuve : en TD

Théorème : le problème de l'arrêt est indécidable

Définition : Soit $T : \mathbb{N} \to \mathbb{N}$ une fonction. La classe $DTIME(T(n))$ est l'ensemble des fonctions de $\{0;1\}^*$ dans $\{0;1\}$ (i.e. des langages, des problèmes de décision) calculable en temps $\mathcal{O}(T(n))$

$$P = \bigcup_{c>0} DTIME(n^c)$$

Pourquoi P ? Pourquoi les machines de Turing ?

1.1.2

La classe NP

Définition : Une machine non-déterministe M est un triplet (Q, Γ, δ) ou

$$Q = \{q_{\text{start}}, q_{\text{accept}}, \ldots\}$$

$$\Gamma = \{0, 1, \ldots\}$$

$$\delta = (\delta_0, \delta_1) \in (\delta_0 : Q \times \Gamma^k \to Q \times P^{k-1} \times \{-1; 0; 1\}^k)^2$$

Le calcul de M sur l'entrée de x est un calcul qui à chaque pas peut utiliser soit δ_0 soit δ_1. Le langage reconnu pas M est l'ensemble des mots tq il existe un calcul de M sur x qui termine sur a_{accept}.

Définition: NP Un langage $\mathcal{L}$ est dans NP s'il existe un polynôme p est une machine M travaillant en temps polynomial telle que $\forall x \in \{0;1\}^*$

$$x \in \mathcal{L} \Leftrightarrow \exists u \in \{0;1\}^{p(|x|)}, M(u, x) = 1$$

u s'appelle un certificat.

Théorème : NP est l'ensemble des problèmes décidables par des machines de Turing non déterministe en temps polynomial

Définition : Une machine de Turing non déterministe M calcul en temps $T(x)$ si tous les calculs de M sur x terminent en temps au plus $T(n)$

$NTIME(f(n))$ est l'ensemble des problèmes décidables en temps $\mathcal{O}(f(n))$ par une machine de Turing non déterministe.

$$NP = \bigcup_{x \geq 1} NTIME(n^c)$$

Preuve :

- $\mathcal{L} \in NP \Rightarrow \mathcal{L}$ calculable par une machine de Turing non déterministe en temps polynomiale

  Il existe p et M tq .... On construit M' non déterministe qui :

  - devine u
  - simule $M(u, x)$ et ecrit 1 si accepte

- Réciproque

  On a une machine de Turing non déterministe M qui décide $\mathcal{L}$ en temps $p(n)$. Considérons la machine M' déterministe qui à i de longueur $p(|x|)$ et x calcule : *WTF ...*

  M' a un ruban de plus que M (le 2eme)

  u est écrit sur le 2eme ruban et δ est defini par $\delta(q, a0 \sim)$ fait comme $\delta_0(a, a \sim)$ et en plus se decale d'une case vers la droite sur le 2nd ruban. On définit de meme $\delta(q, a1 \sim)$

Définition :

$$EXP = \bigcup_{c \geq 1} DTIME(2^{n^c})$$

Théorème :

$$P \subseteq NP \subseteq EXP$$

Preuve : $P \subseteq NP$

- Une machine déterministe est non-déterministe
- Ou avec l'autre définition, $p = 0$

$NP \subseteq EXP$ On part de la definition avec certificat. On définit M' qui décide $\mathcal{L}$ en temps exponentiel.
M' calcul $M(\bar{0}, x)$ puis $M(0\_01, x)$ puis tous les $M(u, x)$ pour $u \in \{0;1\}^{p(|x|)}$.
On accepte si un des calculs a accepté. Le temps de calcul est $\mathcal{O}(T2^p)$ ou M calcule en temps T.

Définition : Réduction (Karp, many-one)
Une langage $\mathcal{L}$ se reduit en temps polynomial à un langage $\mathcal{L}'$, et on écrit

$$\mathcal{L} \leq_p \mathcal{L}$$

s'il existe une fonction

$$f : \{0;1\}^* \to \{0;1\}^*$$

calculable en temps polynomial (qui s'appelle la réduction) tq $x \in \mathcal{L} \Leftrightarrow f(x) \in \mathcal{L}'$

Définition : $\mathcal{L}$ est NP-dur (ou NP-difficile) si pour tout $\mathcal{L}' \in NP$ on a $L \leq_p L'$. $\mathcal{L}$ est NP-complet si $\mathcal{L} \in NP$ et $\mathcal{L}$ est NP-dur.

Théorème :

- $\mathcal{L} \leq_p \mathcal{L}'$ et $\mathcal{L}' \leq_p \mathcal{L}''$ implique $\mathcal{L} \leq_p \mathcal{L}''$
- si $\mathcal{L}$ est NP-complet, alors $\mathcal{L} \in P \Leftrightarrow P = NP$
- si $\mathcal{L}$ est NP-dur alors ...

Théorème : Si $P = NP$ alors pour tout langage $\mathcal{L} \in NP$ il existe une machine M qui trouve un certificat pour $\mathcal{L}$ en temps polynomial.

Preuve : pour SAT
Si $\varphi(x_1, \ldots, x_n)$ est une CNF, alors $\varphi(x_1 = 1 \text{ ou } 0, x_2, \ldots, x_n)$ est une CNF
$\varphi \in SAT$ ssi $\varphi(x_1 = 0, \ldots)$ ou $\varphi(x_1 = 1, \ldots)$ est dans SAT s'appelle etre auto-réductible
Si $SAT \in P$ alors il existe M travaillant en temps $p(n)$ qui resout SAT
Pour trouver une solution on fait l'algo
Sur l'entrée φ :

- Calculer $M(\varphi(x_1 = 0, \ldots))$
  - si 1 alors on recommence avec $\varphi'(x_2, \ldots, x_n) = \varphi(x_1 = 0, \ldots)$
  - si 0 alors on recommence avec $\varphi'(x_2, \ldots, x_n) = \varphi(x_1 = 1, \ldots)$
- A la fin on renvoie les valeurs de $x_1, \ldots, x_n$

finalement, si $\varphi \in SAT$, l'algo renvoie une valuation qui satisfait φ.
Pour $\mathcal{L} \in NP$
Il existe M déterministe et p polynôme tq M calcul en temps polynomial q et

$$\bar{x}\mathcal{L} \text{ ssi } \exists \bar{y} \in \{0;1\}^p(|\bar{x}|)M(\bar{y}, \bar{x}) = 1$$

Il existe une réduction en temps polynomial f de $\mathcal{L}$ à SAT tq une solution pour $f(x)$ nous donne une solution pour x.

Définition : $\mathcal{L} \in$ co-NP ssi $\bar{\mathcal{L}} \in NP$

Autre définition : $\mathcal{L} \in$ co-NP s'il existe des polynômes p et q et une machine déterministe M travaillant en temps q tq $x \in L$ ssi

$$\forall \bar{y} \in \{0;1\}^p(|x|)M(\bar{y}, \bar{x}) = 1$$

Définition : $\mathcal{L}$ est co-NP-complet si $\mathcal{L}$ est co-NP et pour tout $\mathcal{L}' \in$ co-$NP, \mathcal{L}' \leq_p \mathcal{L}$

Exemple : Tautologie est l'ensemble des formules du calcul propositionnel qui sont des tautologies. C'est bien co-NP car on prend $M(\text{valuation des variables}, \varphi) = \text{valuation de } \varphi$

Si $\mathcal{L}' \in$ co-NP alors $\bar{\mathcal{L}} \in NP$ dont il existe une réduction polynomiale f tq $x \in \mathcal{L}'$ ssi $x \in \bar{\mathcal{L}}'$ ssi $f(x) \in SAT$ ssi $\neg f(x) \in$ Tautologie

donc $x \in \mathcal{L}'$ ssi $\neg f(x) \in$ `Tautologie`. La réduction cherché est $f'(x) = \neg f(x)$

Quel est le paysage ? Si $\mathcal{L} \in P$ alors $\bar{\mathcal{L}} \in P$ donc si $P = NP$ alors $P = NP =$ co-NP

Théorème de Ladnen si $P \neq NP$ alors il existe des problèmes NP ni P ni NP-complet et même on a une hiérarchie infinie.

$P = NP$ **?** exemple de problème NP dont on ne sait pas s'il est P ou NP-complet

- Isomorphisme de graphe
- (N, m, p) tq N a un facteur entre m et p

Définition :

$$NEXP = \cup_{c \geq 1} NTIME(2^{n^c})$$

Théorème : si $EXP \neq NEXP$ alors $P \neq NP$

preuve par rembourrage (padding)

Preuve : On vas montrer $P = NP \Rightarrow EXP = NEXP$

Soit $\mathcal{L} \in NEXP$, il existe c et M non déterministe qui calcul en temps 2^{n^c} et qui décide L.

On définit :

$$\mathcal{L}_{pad} = \{(x, 1^{|x|^c}) | x \in \mathcal{L}\}$$

donc $\mathcal{L}_{pad} \in NP$ donc $\mathcal{L}_{pad} \in P$

donc il existe une machine M travaillant en temps polynomiale $p(n)$ qui décide $\mathcal{L}_{pad}$

On considère la machine déterministe M' qui sur l'entrée x écrit $x, 1^{|x|^c}$ et simule M. Son temps de calcul est polynomial en $1^{|x|^c}$ donc $\mathcal{L} \in EXP$

Partie 1.2

Trigonalisations

- $\mathbb{R}$ n'est pas dénombrable
- Indécidabilité du problème de l'arret
- Gödel

1.2.1

Théorème de hiérarchie déterministe en temps

Si g est constructible en temps et

$$f(n)\log f(n) = \wr(g(n))$$

alors

$$DTIME(f(n)) \subsetneq DTIME(g(x))$$

Preuve : en particulier $g(n) \geq n$

Il existe une machine universelle U tel que pour toute machine M et entrée x, $U(M,x) = M(x)$ et M calcul en temps $T(n)$ et U calcul en temps $cT(n)\log T(n)$ ou c dépend de $M(|Q|, |\Gamma|$ et $k)$ mais pas de x

Soit D la machine qui sur l'entrée x simule $U(Mx,x)$ pendant $g(|x|)$ étapes et répond le contraire, et 0 si le calcul n'est pas fini

$$\mathcal{L}(D) \in DTIME(g(n))$$

Si $\mathcal{L}(D) \in DTIME(f(n))$ alors il existe M décidant $\mathcal{L}(D)$ en temps $\mathcal{O}(f(n))$

Il existe c et U simule M en temps $cf(n)\log f(n)$. Il existe x suffisamment grand pour que $cf(|x|)\log f(|x|) \leq g(|x|)$ et $M = Mx$. $U(Mx,x)$ nécessite $cf(|x|)\log f(|x|)$ pas de calcul pour terminer donc moins de $g(|x|)$ étapes

$$x \in \mathcal{L}(D) \text{ ssi } U(Mx,x) = 0 \text{ ssi } Mx(x) = 0 \text{ ssi } x \notin \mathcal{L}(D)$$

Lemme :

α Il existe une machine don déterministe NU telle que pour toute machine non déterministe Mx sur toute entrée y, le temps de calcul de $NU(x,y)$ est $cT(n)\log T(n)$ ou $T(n)$ est le temps de calcul de Mx et c une constante qui ne dépend que de Mx et $NU(x,y) = Mx(y)$

β ... est $cT(n)\ldots$

Théorème (Cook 1972) de hiérarchisation en temps non déterministe.

Si $f(n+1) = o(g(n))$ alors

$$NTIME(f(n)) \subsetneq NTIME(g(n))$$

1er preuve (classique sans tous les détails)

On a une énumération (Mi) des machines non-déterministe. On définit la machine D de la manière suivante. Sur l'entrée x, D simule tous les calculs de $NU(Mx,x)$ pendant $g(|x|)$ pas de calcul et si le calcul finit on répond le contraire et sinon on répond par exemple 0. D accepte ssi tous les calculs refusent.

Le temps de calcul de D est $\mathcal{O}(2^{g(n)})$. Donc $L(D) \in NTIME(g(n))$

Supposons $L(D) \in NTIME(f(n))$. On a une machine non déterministe M qui travaille en temps $\mathcal{O}(f(n))$ et telle que $L(M) = L(D)$

On prend x assez grand pour que le temps de calcul de M soit $< 2^{g(|x|)}$ et x est un numéro de M, donc $M = Mx$

$$x \in L(D) \text{ ssi } NU(Mx,x) = 0$$
$$\text{ssi } x \notin L(M)$$

(... même la prof ne sait plus ...)

Diagonalisation paresseuse L'idée est de prendre des intervalles "grands"

$$]h(i); h(i+1)]$$

et on va se débrouiller pour que dans cet intervalle

$$1^{h(i)+1} \in L \Leftrightarrow \ldots \Leftrightarrow 1^{h(i+1)} \in L \Leftrightarrow 1^{h(i)+1} \notin L$$

On définit D sur l'entrée x de taille n

1. Si $x \notin 1^*$ on rejette (sinon $x = 1^n$)
2. Trouver i tq $h(i) < n \leq h(i+1)$
3. Si $n < h(i+1)$ simuler $M_{1^n}(1^{n+1})$ pendant $g(n)$ étapes et si le calcul se termine on prend la même décision (et sinon $\ldots$)
4. Si $n = h(i+1)$ simuler tous les calculs de $M_{1^n}(1^{h(i)+1})$ pendant $g(h(i)+1)$ étapes et accepter ssi tous les calculs refusent.

Temps de calcul :

1. $\mathcal{O}(n)$
2. pas grand $\rightarrow$
3. $\mathcal{O}(g(n))$
4. $\mathcal{O}(2^{g(h(i)+1)})$

On prend h suffisamment grand pour que

$$2^{g(h(i)+1} < g(\underbrace{h(i+1)}_{n})$$

On a $L(D) \in NTIME(g(n))$. Supposons (...)
On prend x suffisamment grand pour que $M = Mx$ et $f(|x|+1)g(|x|)$ (avec peut etre des constantes t $|x| = h(i)+1$

$$\begin{aligned} 1^{h(i)+1} \vdash L(D) &\text{ ssi } 1^{h(i)+2} \vdash L(M) \\ &\text{ssi } 1^{h(i)+2} \vdash L(D) \\ &\text{ssi } 1^{h(i)+3} \vdash L(M) \\ &\ldots \\ &\text{ssi } 1^{h(i+1)} \vdash L(D) \\ &\text{ssi } 1^{h(i)+1} \nvdash L(M) \end{aligned}$$

Ou $1^{h(i)+1} \vdash L(D)$ veut dire qu'un calcul de $L(D)$ accepte $1^{h(i)+1}$ et $1^{h(i)+1}$ $\not L(M)$ indique que tous les calculs de $L(D)$ refusent $1^{h(i)+1}$

Preuve du blog de Fortnow (5 april 2011)
On définit D qui sur l'entrée 1^i01^m0y fait la chose suivante.

- Si $|y| < f(i+m+2)$ simuler NU sur (i, x_0) et (i, x_1) pendant $g(|x|)$ pas de calcul et accepter ssi ca finit et accepte dans les deux cas.
- Si $|y| = f(i+m+2)$ simuler M_i sur l'entrée 1^i01^m avec les choix de y et prendre la décision opposé
- Si $|y| > \ldots$

temps de calcul $\mathcal{O}(g)$. on suppose que $\ldots i$ et m assez grand M_i

$$\begin{aligned} 1^i01^m0 \in L &\text{ ssi } \forall |y| = 1 \text{ un calcul de } M_i \text{ sur } 1^i01^m0y\text{accepte} \\ &\text{ssi } \forall |y| = 2 \text{ un calcul de } M_i \text{ sur } 1^i01^m0y\text{accepte} \\ &\text{ssi } \forall |y| = f(i+m+2) \text{ un calcul de } M_i \text{ sur } 1^i01^m0y\text{accepte} \\ &\text{ssi } \forall |y| = f(i+m+2) \text{ le calcul de } M_i \text{ sur } 1^i01^m0y\text{avec les choix } y \text{ refuse} \\ &\text{ssi tous les calculs de } M_i \text{ sur } 1^i01^m0y\text{refusent} \end{aligned}$$

Les limites de la diagonalisation : les oracles On a une diagonalisation si

1. Il existe une énumération des machines
2. Il existe une machine universelle efficace

pour les machines à oracles c'est le cas mais on peut avoir $P = NP$ ou $P \neq NP$
Donc on ne peut prouver par diagonalisation que $P = NP$

Définition : Une machine de Turing déterministe à oracle est $(Q, k+2, \Gamma, \delta)$ avec ...
Un ruban appelé ruban de l'oracle et trois états $q_?, q_y, q_n$.
Le calcul se fait de la manière suivante comme d'habitude sauf quand on est dans l'état $q_?$ alors on interroge un oracle θ et après 1 pas de calcul on est dans l'état q_y si le mot sur le ruban de l'oracle $\in \theta$ et q_n sinon.
Pour définir l'oracle, il faut une machine à oracle M, un langage θ et une entrée x : $M^\theta(x)$
L'oracle n'est pas dans la description de M
On définit de même les machines non déterministe à Oracle.
Une machine de Turing est une machine à Oracle (dans laquelle on n'utilise pas l'oracle) mais pas l'inverse !

Définition : Pour un langage θ

$$P^\theta \text{ et } NP^\theta$$

ex :

- Si $\theta \in P$, alors $P^\theta = P$
- Si $S\bar{A}T$ est l'ensemble des formules CNI non satisfiables, alors $S\bar{A}T \in P^{SAT}$?

Théorème : Il existe des langages A et B tels que

$$P^A = NP^A$$

$$P^B \neq NP^B$$

Preuve : Soit $A = EXPCOM = \{(M, x, 1) | \text{le calcul de } M \text{ sur } x \text{ se termine en au plus } 2^n \text{ pas de calcul et répond oui}\}$. On mq $EXP = P^A = NP^A$ (on a $P^A \subseteq NP^A$)

1. $NP^A \subseteq EXP$:

   $L \in NP^A$ il existe une machine non déterministe à oracle calculant en temps n^c pour un certain c et reconnaissant L

   Sur une entrée x

   Pour chaque $y \in \{0;1\}^{n^c}$, faire le calcul de $M(x)$ avec les choix non déterministes y et quand on pose une question $z \in A$?

   z est de taille au plus n^c, $z = (M', x', 1^n)$ donc $n' \leq n^c$

   On calcul $U(M', x')$ sur $2^{n'}$ pas de calcul et on en déduit la réponse.

   $$\underbrace{2^{n^c}}_{|y|} \times \underbrace{2^{n^c}}_{\text{remplacement de l'oracle}} \rightarrow \in EXP$$

2. $EXP \subseteq P^A$

   Soit $L \in EXP$ résolu par une machine M en temps 2^{n^c}

   Soit M' la machine déterministe à oracle suivante qui résout L en temps polynomiale : sur l'entrée x.

   - Écrire sur le ruban de l'oracle $(M, x, 1^{|h|^c})$
   - Faire appel à l'oracle, répondre pareil.

On construit B tq $P^B \neq NP^B$. Soit $U_B = \{1^n | \text{il y a un mot de longueur } n \text{ dans } B\}$
On a $U_B \in NP^B$ (sur l'entrée x, écrire un mot de longueur $|x|$ sur le ruban de l'oracle, poser la question et conclure.
On construit B tq $U_B \notin P^B$ en mettant des mots dans B

$$B_0 = \emptyset, B_{i-1} \subset B_i \text{ et } B = \lim B_i = \cup B_i$$

On énumère les machines à oracle M_i

Hypothèse de récurrence (i) : $\cup B_i$ n'est pas décidé par M_j avec $j < i$ en temps inférieur à $\frac{2^{n_i}}{10}$

étape i : Soit $\begin{cases} n_i > \max\{|x||x \in B_{i-1}\} \\ n_i > n_{i-1} \end{cases}$

Considérons le calcul de M_i sur l'entrée 1^{n_i} pendant $\frac{2^{n_i}}{10}$ pas de calcul.

On regarde les questions posées à l'oracle de longueur n_i. $|E| < \frac{2^{n_i}}{10}$

Si le calcul finit et la réponse est oui alors on pose $B_i = B_{i-1}$ et sinon on prend $x \in \{O;1\}^{n_i} \setminus E$ et on pose $B_i = B_{i-1} \cup \{x\}$

Fin de la construction.

Si $U_B \in P^B$ alors il existe une fonction $f = o(2^n)$ et $U_B \in DTIME^B(f)$ donc il existe une machine M et une constante c tq U_B est décidé par M^B en temps $cf(n)$

Soit (ik) tq $M = M_{i_1} = M_{i_2} = \ldots = M_{i_k}$ (suite croissante)

D'après la construction M n'est pas décodé par M_{i_k} en temps $< \frac{2^n}{10}$ donc pour une infinité de n, $f(n) \geq \frac{2^n}{10}$

Impossible pour un poly f

Partie 1.3

Complexité en espace

Définition On considere des machines avec un ruban d'entrée et un ruban de sortie et des rubans de travail. Un ruban de sortie est un ruban sur lequel on ne peut qu'écrire et se déplacer vers la droite.

Soit $S : \mathbb{N} \mapsto \mathbb{N}$. Un langage L est dans $SPACE(S(n))$ s'il est calculable par une machine qui utilise (ie. visite) au plus $S(n)$ cases de ses rubans de travail.

Une fonction S est constructible en espace s'il existe une machine calculant $S(n)$ en espace $\mathcal{O}(S(n))$.

Théorème

$$DTIME(S(n)) \subseteq SPACE(S(n)) \subseteq NSPACE(S(n)) \subseteq DTIME(2^{\mathcal{O}(S(n))})$$

et donc

$$SPACE(S(n)) \subseteq DTIME(2^{\mathcal{O}(S(n))})$$

pour $S(n) \geq \log(n)$

Une machine travaillant en espace $S(n)$ avec k rubans a au plus, sur l'entrée x, $nS^{k-1}(n)|\Gamma|^{kS(n)}|Q| = 2^{\mathcal{O}(S(n))}$

Comme le calcul termine, il ne boucle pas, donc on ne peut pas passer deux fois par la même configuration. Donc le temps est au plus $2^{\mathcal{O}(S(n))}$

Définition de graphe de configuration

Soit M une machine de Turing (déterministe ou non). Une configuration finie de la machine est l'état + le contenu des rubans (mot fini) + les positions des têtes sur les rubans.

On la code par :

- si sur le ruban j on a : $B, a_1, a_2, \ldots, \overset{\downarrow}{a_i}, \ldots, a_n, B$
  codé par $(j, a_1 \ldots a_{i-1} q a_i \ldots a_n)$ on a $Q \cup \Gamma = \emptyset$

Le code de la configuration à k ruban est la suite des codes des rubans.

Par exemple la configuration initiale C_0 sur l'entrée x a le code

$$(1, q_0, x)(2, q_0) \ldots (k, q_0)$$

On considère des machines qui sur l'entrée $x$ont une seul configuration qui accepte, avec rien sur les rubans de travail (et de sortie) et la tête de lecture sur la première case du ruban d'entrée (on fait le ménage).

Code de la configuration qui accepte x

$$C_{accept} : (1, q_{acc}, x)(2, q_{acc}) \ldots (k, q_{acc})$$

Soit $G_{M,x}$ le graphe de configuration de M sur l'entrée x, avec M travaille en espace $S(n)$

Les sommets sont les configurations sur l'entrée x ou au plus $S(|x|)$ cases ont été visitées. $| \to 2^{\mathcal{O}(S(n))}$ sommets

(C, C') est une arête si on passe de C à C' avec 1 pas de calcul de M. Il existe une formule CNF $\varphi_{M,x}$ tq (C, C') est une arête ssi $\varphi_{M,x}(C, C') = 1$

$\varphi_{M,x}$ est de taille $\mathcal{O}(S(n))$

Lemme 1 : si M travaille en espace $\mathcal{O}(S(n))$ on peut construire $G_{M,x}$ en espace $2^{\mathcal{O}(S(n))}$

Lemme 2 : $PATH = \{< G, s, t > \mid$ il existe un chemin orienté de s à t dans $G\}$ se résout en temps linéaire

(preuve: parcours en largeur)

$$\textbf{Lemme 1} + \textbf{Lemme 2} \Rightarrow NSPACE(S(n)) \subseteq DTIME(2^{\mathcal{O}(S(n))})$$

Définition :

$$PSPACE$$

$$NPSPACE$$

$$L = SPACE(\log n)$$

$$NL = NSPACE(\log n)$$

Théorème :

$$L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE$$

Que pensez vous de :

$L = NP$ ouvert

$L = PSPACE$ voir plus tard

$L = NL$ ouvert

$P = PSPACE$ ouvert

Théorème : $PATH$ est NL-complet
La restriction de $PATH$ aux graphes non orientés est L

Lemme : $PATH$ est NL
On devine un chemin de longueur au plus n de s a t, un sommet à la fois. On suppose que le graphe est donné par la suite de ses arêtes et a n sommets.
On prend 2 rubans de travail

R_1 avec un compteur pour la taille du chemin

R_2 avec le numero du sommet courant

On commence avec $R_1 : 0, R_2 :$ numéro(S)
On lit l'entrée $b = s$.

- Si on trouve une arête (b, a) on le choisit ou on continue (choix non déterministe). Si on la choisit on écrit a sur R_2 et on incrémente R_1.
  - Si $a = t$ on accepte
  - Si $R_1 = n$ on refuse
- Sinon on recommence avec $b_i = a$

Théorème de hiérarchie en espace Si f et g sont constructibles en espace $f = o(g)$ alors $SPACE(f(n)) \subsetneq SPACE(g(n))$

Preuve : exercice

Définition : un langage L est $PSPACE$ dur si pour tout $L' \in PSPACE$, $\_ \leq_p \_$. L est $PSPACE$-complet s'il est $PSPACE$ et $PSPACE$-dur.

Remarque : Si un langage $PSPACE$-dur est dans P alors $\_$.

Définition : une formule QBF est une formule du calcul propositionnel sous forme prenexe, sans variable libre ie. sous la forme $Q_1 x_1 \dots Q_n x_n \underbrace{\varphi(x_1 \dots x_n)}_{\text{sans quantificateur}}$, avec $Q_i \in \{\forall, \exists\}$
ex : $\exists x \forall y \forall z \exists t((t \vee x) \wedge \neg y)$
Les formules QBF sont vraies ou fausses

Définition : $TQBF$ est l'ensemble des formules QBF vraies.

Remarque : $TQBF$ est NP-dur.
Si $\varphi(x_1 \dots x_n)$ est une formule CNF alors $\exists x_1 \dots \exists x_n \varphi$ est une QBF et $\varphi \in SAT$ ssi $\exists x_1 \dots \exists x_n \varphi \in TQBF$
$TQBF$ est co-NP-dur

Théorème : $TQBF$ est $PSPACE$-complet

Lemme : $TQBF$ est $PSPACE$
$Q_1x_1 \dots Q_nx_n\varphi(x_1 \dots x_n)$ de taille m. Espace $\mathcal{O}(n+m)$
On vas donner $\mathcal{O}(nm)$ un algo récursif
Soit L l'ensemble des formules de la forme $Q_1x_1 \dots Q_nx_n\varphi(x_1 \dots x_n, 0 \dots 1)$ avec φ sans quantificateurs et $Q_i \in \{\forall, \exists\}$
On donne un algo pour L comme $TQBF \subseteq L$ ca viendra
Algo $A(Q_1x_1 \dots \varphi(x1 \dots x_n, a_1 \dots a_p))$ et $a_i \in \{0;1\}$.

- Calculer $A(Q_2x_2 \dots \varphi(0, x2 \dots a_p))$
  Retenir un bit de résultat b_0
- Calculer $A(Q_2x_2 \dots \varphi(1, x2 \dots a_p))$
  Retenir un bit de résultat b_1

Déduire le résultat ie si Q_i est $\forall$ (resp $\exists$), le résultat est $b_0 \wedge b_1$ (resp $b_0 \vee b_1$)
Si pas de variable, évaluer $\varphi(a_1 \dots a_p)$
Espace de calcul de A sur une entrée de taille m avec n variables $S(m,n)$

$$S(m,n) = S(m,n-1) + cm$$

$$S(m,0) = cm$$

donc

$$S(m,n) = cmn = \mathcal{O}(mn)$$

$TQBF$ **est** $PSPACE$**-dur** Soit L un langage $PSPACE$ reconnu par la machine M en espace $S(n)$. On considère son graphe de config $G_{M,x}$ sur une entrée x.
$x \in L$ ssi il existe un chemin de C_0 à C_{acc} dans $G_{M,x}$

$$\exists C_1 \dots C_{2^{S(n)}} (W_{j=0}^{2^{S(n)}-1} \varphi_{M,x}(C_j, C_{j+1}))$$

tros gros.
Il existe un chemin de longueur 2^i entre C et $C' \leftarrow \varphi_i(C, C')$
$\Leftrightarrow \exists C''$ et des chemins de longueur 2^{i-1} entre C et C'' et C'' et $C' \leftarrow \exists C'' \varphi_{i-1}(C, C'') \wedge \varphi_{i-1}(C'', C')$
$\varphi_{2^{S(n)}}$ de bonne taille ?
On prend $\varphi_i(C, C') = \exists C'' \forall D \forall D' ((D = C \wedge D' = C'') \vee (D = C'' \wedge D' = C') \Rightarrow \varphi_{i+1}(D, D'))$

Théorème de Savitch Pour toute fonction constructible en espace $S : \mathcal{N} \mapsto \mathcal{N}$ avec $S(n) \geq \log n$ on a

$$NSPACE(S(n)) \subseteq SPACE(S^2(n))$$

1.3.1

NL-Complétude

Définition : Equivalente pour calcul en espace logarithmique.
Un fonction $f : \{0;1\}* \to \{0;1\}*$ est implicitement calculable en espace logarithmique si f est bornée polynomialement (ie $\exists c, \forall x |f(x)| \leq |x|^c$) et les langages suivants sont dans L :

$$L_f = \{< x; i > | f(x) = 1\}$$

$$L'_f = \{< x; i > | i \leq |f(x)|\}$$

On note $A \leq_p B$ si A se réduit avec une fonction calculable en espace logarithmique à B. Il existe f calculable en espace logarithmique tq $\forall x \in \{0;1\}^*, x \in Assif(x) \in B$
On dit que C est NL-complet si il est dans NL et que tout problème NL sont réductible en espace logarithmique en ce problème.

Lemme :

1. si $B \leq_p C$ et $C \leq_p D$ alors $B \leq_p D$
2. si $B \leq_p C$ et $C \in L$ alors $B \in L$

preuve Soit f (resp g) la réduction de B à C (resp C à D), alors $h = g \circ f$ est calculable en espace logarithmique :

$$M_f \text{ la machine } < x; i > \rightarrow f(x)$$

$$M_g \text{ la machine } < x; i > \rightarrow g(x)$$

Soit M la machine qui sur l'entrée $< x; i >$ calcule $g \circ f$ de la manière suivante (cf. DM)

Théorème : Path est NL-complet $\{< G, s, t >$ tq il existe une chemin de s à t dans le graphe orienté $G\}$

Preuve : Soit A un problème NL avec la machine non déterministe M calculant en espace logarithmique.
On considere le graphe de conf de M sur l'entrée x : $G_{M,x}$ a au plus $2^{\mathcal{O}(\log |x|)}$ sommets. $x \in A$ ssi il existe un chemin de C_init à C_accept dans $G_{M,x}$.
$G_{M,x}$ est donné par sa matrice d'adjacence entre sommets on peut calculer un bit en espace logarithmique ?
On peut décider si deux configurations sont adjacentes en espace logarithmique : les écrire et décider.

Théorème : $L = NL$ ssi $PATH \in L$

Propriété de NL avec les certificats.
$A \in NL$ ssi il existe une machine M, déterministe et qui calcul en espace log, et un polynome p tq

$$x \in A \text{ ssi } \exists y \in \{0;1\}^{p(|x|)} M(x, y) = 1$$

La machine doit avoir un ruban spécial sur lequel est écrit y et "read once" ie on ne peut qu'aller vers la droite

Preuve de la propriété :
S'il existe M tq ... et $x \in A$ ssi $\exists y \in \{0;1\}^{p(|x|)} M(x, y) = 1$ alors $A \in NL$ avec la machine non déterministe suivante M' :
Elle fait comme M et quand M lit un nouveau bit sur le ruban de certificat, elle fait un choix non déterministe.

Réciproque : Si $A \in NL$ avec la machine non deterministe M alors on considère la machine M' et le polynôme $p(n)$ egal au nombre max de config de M sur une entrée de taille n
Fonctionnement de M' : quand M doit faire un choix non deterministe δ_0 ou δ_1, M' lit un bit de y et dait δ_0 si c'est 0 et δ_1 sinon.

Théorème : $P\bar{A}TH$ est co-NL-complet

Théorème : $P\bar{A}TH \in NL$

Corollaire : $NL =$ co-NL

Preuve : La donnée G de taille $n + 1$, $s = v_0$, $t = v_n$
On pose C_i l'ensemble des sommets accessibles de s avec un chemin de longueur au plus i.

$$C_0 = \{s\}$$

$$C_1 = \{s\} \cup \{\text{voisins de } s\}$$

$(G, s, t) \in PATH$ ssi $t \in C_n$.

Échauffement : on peut donner un certificat de $v \in C_i$: une liste de i sommets dont le dernier est v.
Taille de la donnée v et i : $2 \log n$
Longueur du certificat : $i \log n \leq n \log n$

Algorithme A_1 Rubans :

- *Entrée* G, s, t
- E_1 v, i
- *Travail* 1 initialise avec 1
  2 initialise avec le numéro de v
  3 initialise avec rien

1. Recopier le numéro du sommet sur le ruban 3 (lu sur le certificat).
2. Vérifier qu'il existe bien une arête dans G entre les sommets des rubans 2 et 3, sinon on arrête et on refuse.
3. Regarder si le sommet du ruban 3 est v. Si oui on accepte et on arrête.
4. Incrémenter de 1 le ruban 1. Si supérieur à i, on arrête et on refuse.
5. Mettre en 2 ce qui est en 3 et effacer 3
6. Recommencer à l'étape 1

Idée : certificat de $v \notin C_{i+1}$ sachant $|C_i| = k$
$v \notin C_{i+1} ssi \forall w \in C_i, (w, v) \notin E$
Donnée : G, s, t et $i, k, v \rightarrow$ taille $3 \log n$
Taille du certificat (un certificat d'appartenance pour chaque $w \in C_i$)

$$k \times i \log n \leq n^2 \log n$$

Algorithme A_2 Rubans:

- *Entrée* G, s, t
- E_2 i, k, v
- *Travail* $E_1, 1, 2, 3, 4, 5$

1. Mettre 1 sur le ruban 4 (compteur pour k)
2. Mettre 0 sur le ruban 5 (0 : premier sommet s)
3. Si v n'est pas voisin de s, on arrête et on refuse.
4. Ajouter 1 au compteur du ruban 4. Si supérieure à k, on arrête et on accepte.
5. Lire dans le certificat un sommet x et le copier sur 5 (à la suite de celui qui y est). S'il y a un numéro plus petit (au sens large), refuser. Sinon, ne laisser que celui la sur 5
6. Sur E_1 recopier ce sommet et i.
7. Avec A_1, lire un certificat pour $w \in C_i$ etverifier.
8. Si v est le voisin de w, refuser.
9. Boucler à l'étape 4

Remarque : Espace de travail : ruban $5 \leq \log n$, ruban $4 \leq \log n$

Algorithme A_3 : Certificat de $|C_{i+1}| = k'$ connaissant $|C_i|$
Certificat : k' puis chaque sommet v de G donner 0 suivi d'un certificat de $v \notin C_{i+1}$ ou 1 suivi d'un certificat de $v \in C_{i+1}$
Longueur du certificat : $\log n + n \times (1 + n^2 \log n) \to \mathcal{O}(n^3 \log n)$
Rubans :

- *Entrée* G, s, t
- E_3 i, k (pour $|C_i|$)
- $E_1, E_2, 1, 2, 3, 4, 5$
- *Travail* 6 initialisé a k (compteur pour $|C_{i+1}|$)
  7 initialisé a 1 (sommet en cours)
  8 initialisé a k (pour se rappeler de $|C_{i+1}|$)

1. Lire un bit du certificat
2. Si c'est 0 faite A_1
3. Si c'est 1 faite A_2 et décrémenter 6. Si on ne peut pas décrémenter on refuse.
4. Si 7 contient $t = v_n$ et 6 contient 0 on accepte.
5. Si 7 contient $t = v_n$ et 6 ne contient pas 0 on refuse.
6. Sinon on ajoute 1 a 7 (sommet suivant) et on boucle à l'étape 1

Algorithme final Rubans :

- "read once" pour le certificat
- *Entrée* G, s, t
- $E_1, E_2, E_3, 1, 2, \ldots, 8$
- 9 : Compteur pour i

1. Initialiser 9 à 1 et sur E_3 $1, 1$
2. Faire A_3 de $i = 1$ à $i = n - 1$
   Puis A_2 pour $t = v_n$ et $|C_{n-1}|$

Longueur du certificat :

$$n \times n^3 \log n + n^2 \log n \to \mathcal{O}(n^4 \log n)$$

1.3.2 Récapitulatif

$$L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE = NSPACE \subseteq EXP \subseteq NEXP$$

$$NL \neq PSPACE$$

$$P \neq EXP$$

$$NP \neq NEXP$$

Partie 1.4

La hiérarchisation polynomiale

- Une généralisation de P, NP, co-NP
- Au moins 3 définitions
- Pb-complets ?

1.4.1

NP

On demande l'existence d'un objet qui vérifie une propriété dans P et si la propriété est plus compliquée à verifier ?
Egalite de f et $g : \forall x, f(x) = g(x)$
Et si on veut le plus petit objet vérifiant la propriété ?
$min_e q_d nf = \{< \varphi, k >$ tq il existe une formule sous forme DNF ψ de taille au plus k équivalente à $\varphi\}$ (cf. citation de Claude Shannon)
$indset = \{< G, k |$le graphe G a un ensemble indépendant de taille au moins $k\} \to NP$-complet.
$exact_i ndset = \{< G, k > |$ le plus grand ensemble indépendant de G est de taille $k\}$
$\exists S$ de taille k indépendant, $\forall S'$ de taille $> k, S'$ non indépendant

Premier niveau de hiérarchie NP, co-NP

Second niveau de hiérarchie

Définition : La classe Σ_2^p est l'ensemble des langages L tq il existe un polynome q est machine M travaillant en temps polynomial tq

$$x \in L \text{ ssi } \exists u_1 \in \{0;1\}^{(|x|)} \forall u_2 \in \{0;1\}^{(|x|)}, M(x, u_1, u_2) = 1$$

Exemple $min_e q_d if$ est Σ_2^p-complet
$exact_i ndset$ est Σ_2^p

$$P \subseteq \Sigma_2^p$$
$$NP \subseteq \Sigma_2^p$$
$$\text{co-}NP \subseteq \Sigma_2^p$$

On pense que les inclusions sont strictes

Définition $\Pi_2^p \dots \forall u_1, \dots \exists u_2$, autrement dit $L \in \Pi_2^p$ ssi $\bar{L} \in \Sigma_2^p$

Définition Pour tout i on définit Σ_i^p par un langage L est dans Σ_i^p s'il existe un polynome q et une machine M travaillant en temps polynomial tq :

$$\forall x, x \in L \text{ ssi } \exists u_i \in \{0;1\}^{(|x|)} \forall u_2 \in \{0;1\}^{(|x|)} \dots Q_i u_i \in \{0;1\}^{(|x|)}, M(x, u_1, \dots u_i) = 1$$

avec $Q_i = \exists$ si i est impaire et $\forall$ si i est pair.

$$\Pi_i^p \dots \forall \dots \exists \dots Q_i \dots$$

avec $Q_i = \forall$ si i est impaire et $\exists$ si i est pair.

Exemple

- $\Sigma_0^p = \Pi_0^p = P$
- $\Sigma_1^p = NP$
- $\Pi_1^p = NP$

Définition :

$$PH = \bigcup_{i \in \mathbb{N}} \Sigma_i^p$$

Propriété :

$$\Pi_i^p = \{\bar{L} | L \in \Sigma_i^p\}$$

Conjecture : On pense que tout est différent

Propriété : $PH = \bigcup_i \Pi_i^p$

Théorème

1. Pour $i \geq 1$ fixé, si on a $\Sigma_i^p = \Pi_i^p$ alors $PH = \Sigma_i^p$ et on dit que la hiérarchie s'effondre au rang i
2. Si $P = NP$ alors $PH = P$ ie la hiérarchie s'effondre sur P

$1 \Rightarrow 2$ 1 laissé en exercice

Preuve de 2 On montre par recurence sur i que $\Sigma_i^p = P$. On aura alors $PH = \bigcup \Sigma_i^p = P$

- $i = 0$: $P = P$
- $i = 1$: $\Sigma_i^p = NP = P$
- si pour $i \geq 1$ on a $\Sigma_i^p = P$ alors montrons que $\Sigma_{i+1}^p = P$

Soit $L \in \Sigma_{i+1}^p$, il existe un polynome q et une machine M travaillant en temps polynomiale tq

$$\forall x, x \in L \text{ ssi } \exists u_1 \in \{0;1\}^{(|x|)} \dots Q_{i+1} u_{i+1} \in \{0;1\}^{(|x|)}, M(x, u_1, \dots u_{i+1}) = 1$$

$$L' = \{(x, u_1) | \forall u_2 \in \dots Q_{i+1} u_{i+1} \in \dots M(x, u_1, \dots u_{i+1}) = 1\}$$
$$L' \in \Pi_i^p$$

Donc $\bar{L'} \in \Sigma_i^p$ donc $\bar{L'} \in P$ donc $L' \in P$
donc $L \in NP$
donc $L \in P$
donc $\Sigma_{i+1}^p = P$

Définition : Un probleme L est Σ_i^p-complet s'il est dans Σ_i^p et pour tout $L' \in \Sigma_i^p$ on a $L' \leq_p L$

Théorème :

1. $PH \subseteq PSPACE$
2. S'il existe un problème PH-complet alors la hiérarchie s'effondre à un certain niveau
3. $PH = PSPACE \Rightarrow$ la hiérarchie s'effondre

Preuve :

1. Par recurrence
   - $i = 0, P \subseteq PSPACE$
   - $i = 1, NP \subseteq PSPACE$
   - si $\forall i \geq 1$ on a $\Sigma_i^p \subseteq PSPACE$ alors
     Soit $L \in \Sigma_{i+1}^p$: il existe un polynome q et une machine M travaillant en temps polynomiale tq

     $$\forall x, x \in L, \exists u_i \in \dots \forall u_2 \dots Q_{i+1} u_{i+1} \dots M(x, u_1, \dots, u_{i+1}) = 1$$

     On définit $L' = \{(x, u_1) | \dots\}$
     $L' \in Sigma_i^p$ donc $L' \in PSPACE$
     donc $L \in NPSPACE = PSPACE$
     donc $\Sigma_{i+1}^p \subseteq PSPACE$
2. Soit $L \in PH$-complet, donc il existe i tq $L \in \Sigma_i^p$ et $PH = \Sigma_i^p$
3. Si $PH = PSPACE$, $TQBF$ est $PSPACE$-complet donc PH-complet donc on applique 2

Exemples : $\Sigma_i SAT = \{\exists \bar{u_1}, \forall \bar{u_2} \dots Q_i \bar{u_i}, \varphi(\bar{u_1}, \dots, \bar{u_i} \text{ vrai} | \varphi \text{ est une formule booléenne sans qualificateurs}\}$

$succint - set - cover$: on se donne une ensemble S de formules sous forme $3 - DNF\{\varphi_1 \dots \varphi_m\}$ à n variables, et un entier k. On doit determiner s'il existe $S' \subseteq \{1 \dots m\}, |S'| \geq k$ tq $S' \vee \varphi_i$ est une tautologie est Σ_2^p-complet.

Théorème : $\Sigma_i SAT$ est Σ_i^p-complet

Preuve :

- $\Sigma_i SAT \in \Sigma_i^p$
- Soit $L \in \Sigma_i^p$ donc il existe q et $M \dots$

  $x \in L$ ssi $\exists u_1 \in \dots Q_i u_i \dots M(x \dots u_i) = 1$

  Il existe φ de taille $\mathcal{O}(p^2)$ tq $\varphi(x \dots u_i) = 1$ ssi $M(x \dots u_i) = 1$

Un problème Π_i^p-complet

Deuxième définition : Les machines alternantes, une généralisation des machines non-déterministes.

1. Syntaxe : une machine alternante est une machine non-déterministe dont les états non-finaux sont étiquetés par $\forall$ ou $\exists$
2. La dynamique du calcul est la même que pour une machine non déterministe (on ne tient pas compte des étiquettes), idem pour le temps de calcul.

   On ne considère que les machines dont tous les calculs terminent
3. Sémantique : On considère $G_{m,x}$ et on va mettre des étiquettes A ou R sur les configurations.

   Une configuration finale qui accepte (resp. rejette) est étiqueté A (resp. R)

   Une configuration dont l'état est $\exists$ et dont une configuration suivante est A est A, sinon elle est R

   Une configuration dont l'état est $\forall$ et que toutes les configurations suivantes sont A, alors elle est A, sinon elle est R.

   La machine M accepte x si la configuration initiale de M sur x a l'étiquette A

$\Sigma_i TIME(T(n))$ est l'ensemble des langages reconnus par une machine alternante calculant en temps T tq q_0 est étiqueté par $\exists$ et sur tout chemin de calcul l'étiquette de l'état change au plus $i-1$ fois.

$$\Pi_i TIME(T(n)) = \dots \forall \dots$$

$$\Sigma_i P = \bigcup_c \Sigma_i TIME(n^c)$$

$$\Pi_i P = \bigcup_c \Pi_i TIME(n^c)$$

Théorème : $\forall i, \Sigma_i^p = \Sigma_i P$ et $\Pi_i^p = \Pi_i P$

Preuve :

1. $\Sigma_i^p \subseteq \Sigma_i^p$

   $L \in \Sigma_i^p$ il existe q et $M \dots$

   On définit N alternante :

   q_0 est étiqueté $\forall$

   N écrit $u_1 \in \{0;1\}^{q(|x|)}$ de manière non-deterministe avec des états $\exists$

   N écrit $u_2 \in \{0;1\}^{q(|x|)}$ de manière non-deterministe avec des états $\forall$

   $\dots$

   N s'écrit $u_i \in \{0;1\}^{q(|x|)}$ de manière non-deterministe avec des états Q_i
2. Réciproque

   Soit N une machine alternante pour $L \in \Sigma_i P$ (calcul en temps T)

   On définit la machine déterministe M sur l'entrée $x, u_i \in \{0;1\}^{T(|x|)} \dots u_i \in \{0;1\}^{T(|x|)}$

   M lit le premier bit de u_i et en déduit quel calcul de N faire

   et si l'état de N change d'étiquette (devient $\forall$) M passe au bloc suivant, ie va lire les bits de $u_2 \dots$

1.4.2

Compromis espace-temps pour SAT

Question ouvertes ?

- $SAT \in TIME(n)$?
- $SAT \in SPACE(\log^k n)$?

Définition : $TISP(f(n), g(n))$ est l'ensemble des problèmes solubles par une machine de Turing en temps $\mathcal{O}(f(n))$ et espace $\mathcal{O}(g(n))$

Théorème :

1. $SAT \notin TISP(n, \log^k n)$
2. $SAT \notin TISP(n^{1,1}, n^{0,1})$
3. $SAT \notin TISP(n^c, n^{\mathcal{O}(1)})$ avec $c < 2\cos(\frac{\pi}{7})$

On a $2 \Rightarrow 1$

lemme 1 $NTIME(n) \subseteq DTIME(n^{1,2})$
$\Rightarrow \Sigma_2 TIME(n^8) \subseteq NTIME(n^{9,6})$

lemme 2 $TISP(n^{12}, n^2) \subseteq \Sigma_2 TIME(n^8)$

lemme 3 $NTIME(n^{10}) \subseteq NTIME(n^{12})$

lemme 4 $NTIME(n) \nsubseteq TISP(n^{1,2}, n^{0,2})$

lemme 5 $SAT \in TISP(n^{1,1}, n^{0,1})$
$\Rightarrow NTIME(n) \subseteq TISP(n^{1,2}, n^{0,1})$

TD

Partie 2.1

Sceance 1 - 18 sept 2012

2.1.1

Le Théorème de Cook

Définition : Un littéral est une variable ou une négation de variable. Une clause est une disjonction finie de littéraux (ex. $x_1 \wedge \neg x_2$). Une formule sous forme normale conjonctive (CNF) est une conjonction finie de clauses

$$\bigwedge_i (\bigvee_j l_{ij})$$

Une formule sur les variables $i \in I$ est satisfiable s'il existe une valuation qui lui donne la valeur 1. Une valuation sur l'ensemble des variables

$$X = \{x_i | i \in I\}$$

est une fonction

$$v : X \to \{0;1\}^{|I|}$$

. Une valuation s'étend de manière canonique aux formules.

SAT est l'ensemble des formules CNF satisfiables.

$SAT3$ est l'ensemble des formules CNF dont chaque clause a au plus 3 littéraux, satisfiable.

Théorème : Toute fonction $f : \{0;1\}^k \to \{0;1\}$ peut s'exprimer comme une CNF avec au plus $k2^k$ littéraux

Preuve au tableau

Remarque : Il existe $\varphi(x_1, \ldots, x_k)$ tq $\forall \bar{x} \in \{0;1\}^k = f(\bar{x})$

Lemme : SAT est NP Machine qui sur l'entrée φ formule, v valuation (suite de 0 et de 1) calcule la valuation de φ

Théroème de Cook-Levin : SAT est $NP - dur$ Soit $L \in NP$, il existe une machine oublieuse à 2 rubans M travaillant en temps $T(n)$ et un polynome p tq

$$\forall \bar{x} \in L \text{ ssi } \exists \bar{y} \in \{0;1\}^{p(|x|)} M((\bar{y}, \bar{x})) \text{ accepte}$$

On vas construire $\varphi(\bar{x}, \bar{y}, \bar{z}) : CNF$ tq $\forall \bar{x} \varphi(\bar{x}, \bar{y}, \bar{z})$ satisfiable ssi $\bar{x} \in L$, ie ssi $\exists \bar{y} \in \{0;1\}^{p(|x|)} M(\bar{y}, \bar{x})$ accepte

Dit autrement, on cherche une rédaction de L à SAT. On cherche une fonction f calculable en temps polynomiale tq

$$\bar{x} \in L \text{ ssi } f(\bar{x}) \in SAT$$

en particulier $f(\bar{x})$ est une formule CNF

si $x \in L$ on prend $\varphi(\bar{x}, \bar{y}, \bar{z}) = 1$ sinon 0. marche si $L \in P$ donc

$$L \in P \Rightarrow L \leq_P SAT$$

On considère un calcul de M sur $\bar{y}, \bar{x}$. On appel un instantané à l'instant t le triplet (q, a, b) ou q est l'etat au temps de calcul t, a la lettre lue sur le ruban d'entrée et b la lettre lue sur le ruban de travail. Se code sur $3c$ bits avec c tel que $\max(|Q|, |P|) \leq 2^c - 1$.

A l'instant $t+1$ on a $(q_{t+1}, a_{t+1}, b_{t+1})$. Vrai ou faux ?

(...)

Partie 2.2

Séance 3 - 9 sept 2012

2.2.1

Exercice 1

1

Fait la semaine dernière

2

Fait la semaine dernière

3

Fait la semaine dernière

4

Pour $n' = 1$ a n, (calculer $H(j)$)
Pour tout $x \in \{0;1\}^*$ avec $|x| < \log n'$
Pour tout $i < \log\log n'$, calculer $M_i(x)$ (sur au plus $i|x|^i$ étapes)
Vérifier si $M_i(c) = SAT_H(x)$
$SAT_H(x)$ se calcule en vérifiant que $x = \varphi 01^{n'^{H(n')}}$ ($\varphi \in SAT > 2^{\log n}, m < \log n'$ donc $H(m')$ déjà calculé)
Stocker H_j

Complexité :

$$\mathcal{O}(n \times n \times \log\log(n) \times (\overbrace{\log(n)^{\log\log(n)}}^{=2^{\log^2\log(n)} \leq 2^{\log(n)}} \times \overbrace{\log\log(n)}^{<n} \overbrace{\log(\overbrace{\log\log(n)}^{<n} \times \overbrace{\log(n)^{\log\log(n)}}^{<n})}^{<\log(n)} + 2^{\log(n)} + \log(n) \times \log\log(n))$$

Complexité :

$$\mathcal{O}(n^4 \log(\log(n)) \log(n))$$

2.2.2

5

S tq $SAT_H \notin P$
Supposons $SAT_H \in P \Rightarrow H(n)$ borné
$\exists c, H(n) \leq c$
$\forall y \in \{0;1\}*, y \in SAT \Leftrightarrow y01^{|y|^{H(|y|)}} \in SAT_H$ or $|y|^{H(|y|)}$ se calcule en temps polynomial
Donc $SAT(y)$ est en temps polynomiale
$\Rightarrow SAT \in P$
$\Rightarrow P = NP$
or $P \neq NP$
$\Rightarrow SAT_H \notin P$
Supposons SAT_H NP-complet
Comme $\lim_{n\to\infty} H(n) = \infty$ (par 3), on peut réduire SAT à SAT_H en temps polynomial in^i
Alors, à x instance de SAT on associe $f(x)$ instance de SAT_M, avec $|f(x)| \leq i|x|^i$ telle que $x \in SAT \Leftrightarrow f(x) \in SAT_H$
Soit $\varphi \in SAT$, $f(\varphi) = \psi 01^{n^{F(n)}}$ ou $n = |\psi|$ et $\psi \in SAT$
(...)

Idée de correction : k constante
Stocker $SAT(x) pour |x| \leq k^2$
Algo pour $SAT(x)$:

- Si $|x| \leq k^2$, regarder dans la memoire
- Si $|x| > k^2$, alors $x \to^{SAT \to SAT_H} y$

1. Si $y \neq \psi 01^{|\psi|^{H(|\psi|)}}$ alors $y \notin SAT_H$ répondre non
2. Dans ce cas $|\psi| \leq \sqrt{|x|}$
3. Algo pour $SAT(\psi)$ par récurrence.

Pourquoi (2) ? $x \rightarrow^{SAT \rightarrow SAT_H} y$ se fait en temps $i|x|^i$. Or $y = \psi 01^{|\psi|^{H(|\psi|)}}$
Donc $|\psi|^{H(|\psi|)} \leq |y| \leq i|x|^i$ ie $H(|\psi|)\log(|\psi|) \leq i^2 \log(|x|)$

2 cas :

- Soit $|\psi| \leq k$, comme $|x| \geq k^2$ on a $|\psi| \leq \sqrt{|x|}$
- Soit $|\psi| > k$, alors $H(\psi) > 2i^2$ or $\log(|\psi|) \leq \frac{1}{2i^2} i^2 \log(|x|) = \log(\sqrt{|x|})$

7

$SAT_H \in NP$
Si $P \neq NP$ alors $SAT_H \notin P$ et SAT_H n'est pas NP-complet

Théorème de Ladner : Si $P \neq NP$, alors il existe des problèmes intermédiaires (ie des problèmes dans NP qui ne sont ni dans P ni NP-complet)