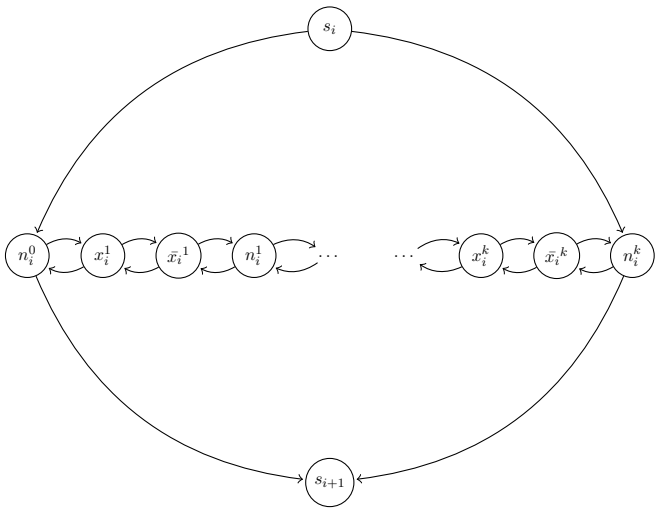
1 Réduction de CNF-SAT à chemin hamiltonien

Un chemin de s à t dans un graphe G est dit hamiltonien si le chemin passe exactement une fois dans chaque nœud du graphe. On définit alors le problème $HAMPATH$ de la façon suivante :

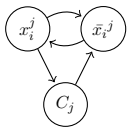
$$HAMPATH = \{(G, s, t) \mid \text{il existe un chemin hamiltonien de } s \text{ à } t \text{ dans le graphe orienté } G\}$$

Question 1. Montrer que $HAMPATH$ est $\mathcal{NP}$-complet.

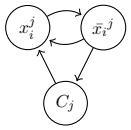
Suggestion : partir d'une formule CNF-SAT à k clauses $(C_1, \dots, C_k)$ et l variables $(x_1, \dots, x_l)$ et construire un graphe qui contient (pour i une variable et j une clause) : trois nœuds x_i^j, $\bar{x}_i^j$ et n_j^i ; deux nœuds s_i et n_i^0 ; un nœud C_j plus un nœud s_{l+1} et utilise les gadgets suivants :



(a) Gadget de variable



(b) Gadget pour x_i utilisé dans C_j



(c) Gadget pour $\bar{x}_i$ utilisé dans C_j

Question 2. Montrer que le problème du chemin hamiltonien reste $\mathcal{NP}$-complet pour les graphes non orientés.

2 Le théorème de hiérarchie temporelle

Pour simplifier la programmation des machines, nous n'utiliserons ici que des machines de Turing déterministes à 2 bandes. On utilise un encodage raisonnable des MT que l'on note $\langle M \rangle$ pour une machine M. Par exemple, $\langle M \rangle$ code les états par des mots de $\{0,1\}^*$, l'alphabet commun des rubans par des mots de $\{\alpha, \beta\}^*$, une direction de changement des têtes par $\{<, >, \_\}^2$. Une machine est alors représentée par une suite de transitions pour chaque état. Pour l'état e on représente ça de la façon $e|a_1, b_1, c_1, d_1, e_1, f_1, \dots a_k, b_k, c_k, d_k, e_k, f_k|$ où les $e, (e_i)_i$ sont des états, les $(a_i)_i, (b_i)_i, (c_i)_i, (d_i)_i$ sont des lettres du ruban, les $(f_i)_i$ des directions. a correspond à la lettre lue sur le premier ruban, b sur le second ruban, c à celle écrite sur le premier ruban et d à celle sur le second.

Définition 1. *L'ensemble $TIME(f(n))$ correspond à l'ensemble des problèmes pour lesquels il existe une machine de Turing déterministe qui décide en moins de $f(n)$ étapes de calcul les entrées de taille n.*

Attention, de la même manière qu'on dit que le tri peut se faire en $O(n \times ln(n))$ en omettant de dire "sur une entrée de taille n", on écrit toujours $TIME(f(n))$ mais il faut bien évidemment lire $TIME(n \to f(n))$ car le n représente la taille de l'entrée.
Dans la suite de ce TD, si besoin, vous pouvez vous reposer sur le théorème d'accélération linéaire qui énonce que pour tout g et tout $\epsilon > 0$ **fixés**, $TIME(g(n)) \subseteq TIME(n + 2 + \epsilon g(n))$.

Définition 2. *Une fonction f est dite* constructible en temps *s'il existe une machine M qui prend en entrée un mot 1^n et produit la représentation binaire de $f(n)$ en temps $f(n)$.*

Le théorème de hiérarchie temporelle dit que si f est une fonction constructible en temps alors :

$$TIME\left(o\left(\frac{f(n)}{logf(n)}\right)\right) \subsetneq TIME(f(n))$$

Nous allons ici montrer une version amoindrie, $TIME(f(n)) \subsetneq TIME(f(2n+1)^3)$ *avec* $n^3 \leq f(n)$.
Pour f constructible en temps, on définit le langage $Halt_f = \{\langle M\rangle \# x \mid M \text{ accepte } x \text{ en } f(|x|) \text{ étapes}\}$.

Question 3. Justifier que $Halt_f \in TIME(f(n)^3)$.

Supposons par l'absurde que $Halt_f \in TIME\left(\frac{f(\lfloor n/2 \rfloor)}{2}\right)$. Soit K décidant $Halt_f$ en temps $f(\lfloor n/2 \rfloor)/2$, on peut alors construire D_K qui prend en entrée $\langle M\rangle$ le code d'une machine de Turing et lance K sur $\langle M\rangle \# \langle M\rangle$. D_K accepte quand K refuse et refuse quand K accepte.

Question 4. Justifier que D_K peut être construit de façon à ce que $D_K \in TIME(f(n))$.

Question 5. Justifier que D_K ne peut pas exister par un argument de diagonalisation.

Question 6. En déduire que $TIME(f(\lfloor n/2 \rfloor)/2) \subsetneq TIME(f(n)^3)$.

Question 7. En déduire que $P \subsetneq EXPTIME \subsetneq 2 - EXPTIME$.
On rappelle : $P = \bigcup_{k\in\mathbb{N}} TIME(n^k)$, $EXPTIME = \bigcup_{k\in\mathbb{N}} TIME(2^{n^k})$, $2-EXPTIME = \bigcup_{k\in\mathbb{N}} TIME(2^{2^{n^k}})$

3 Un problème $EXPTIME$ complet

Question 8. Justifier que $f(x) = 2^x$ est constructible en temps. En déduire que $Halt_f \in EXPTIME$.

Question 9. Soit $L \in EXPTIME$, donner une réduction polynomiale de L à $Halt_f$. En déduire que $Halt_f$ est $EXPTIME$ complet.

4 Un problème $PSPACE$ complet

L'ensemble des formules booléennes quantifiées (QBF) est défini par induction :

- Une variable propositionnelle est une formule booléenne quantifiée ;
- Si ϕ est une formule booléenne quantifiée, alors $\neg\phi$ est une formule booléenne quantifiée ;
- Si ϕ et ψ sont deux formules booléennes quantifiées, alors $\phi \wedge \psi$ est une formule booléenne quantifiée ;
- Si ϕ est une formule booléenne quantifiée et p est une variable propositionnelle, alors $\forall p\phi$ et $\exists p\phi$ sont des formules booléennes quantifiées.

Les QBF sont dotées de leur valeurs de vérité usuelle. Le langage $TQBF$ est le langage des QBF valant vrai.

Question 10. Montrer que le problème $TBQF$ est $PSPACE$ complet.

Suggestion pour la partie $PSPACE$ difficile : partir d'un problème L qui est $PSPACE$. Montrer que L est décidé par une machine K pour laquelle il existe $P \in \mathbb{N}[X]$ tel que sur une entrée de taille n, K accepte ou refuse en temps $2^{P(n)}$ avec $P(n)$ de mémoire. Faire une réduction de L à $TQBF$ Encoder l'état de K et de la mémoire de K avec un $P(n)$ uplet de $\{0,1\}$ et donner la formule $\Phi(c_1, c_2, t)$ qui décrit si l'on peut passer de l'état c_1 à c_2 en temps 2^t (utiliser une exponentiation rapide).