Master Info - 2015-2016
MIF15 Complexité et Calculabilité

Examen Final
Corrigé rédigé par Paul Brunet et Laure Gonnord

Durée 1H30
Notes de cours et de TD autorisées. Livres et appareils électroniques interdits.
Le barème est donné à titre **indicatif** .

1 Machines de Turing

Question 1 (4 points)

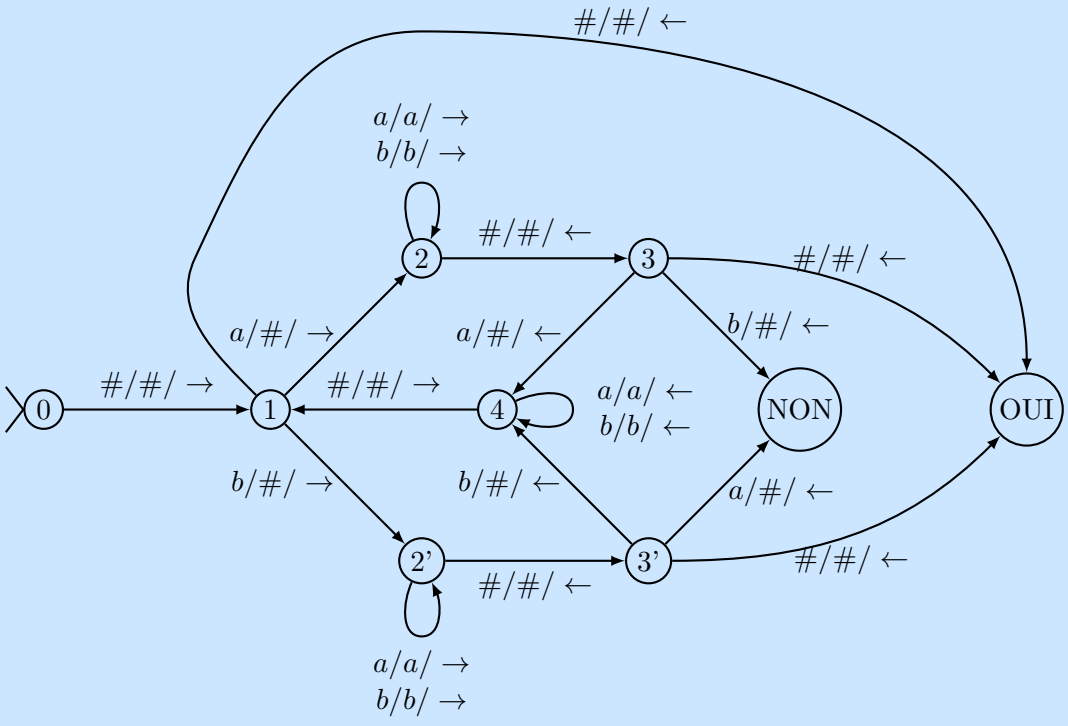
Construisez une machine de Turing déterministe acceptant le langage L des palindromes sur $\{a, b\}^*$, défini par $L = \{w \in \{a, b\}^* | w = w^R$, où w^R est le mot miroir de w).

Vous écrirez une machine de Turing :

- À un seul ruban infini à droite, le marqueur délimitant le mot d'entrée étant #, la tête de lecture initialement positionnée sur le premier blanc à gauche de w.
- Qui **décide** L (avec donc un état d'acceptation et un état de refus).

Votre machine de Turing viendra avec des **EXPLICATIONS** et un **EXEMPLE**. Soignez le dessin et la rédaction !

Solution:



2 Fonctions primitives récursives

Définition. Une fonction $f : \mathbb{N}^n \to \mathbb{N}$ (avec $n \geq 0$) est *récursive primitive* si elle est :

- Une des fonctions renvoyant 0 : $\mathsf{zero_n} : (x_1 \dots x_n) \mapsto 0$. La fonction constante $\mathsf{zero_0} : () \mapsto 0$ sera par abus de notation souvent notée 0.
- $\mathsf{succ} : x \mapsto x + 1$ la fonction successeur (alors $n = 1$) ;
- $\mathsf{id}_n^i : (x_1, \dots, x_n) \mapsto x_i$ les fonctions de projection, pour $1 \leq i \leq n$;

et stable par les opérations de base :

- $\mathsf{Comp}_n(g, h_1, \dots, h_m) : (x_1, \dots, x_n) \mapsto g(h_1(x_1, \dots, x_n), \dots, h_m(x_1, \dots, x_n))$ la composition. Notons que n désigne ici l'arité de la fonction obtenue (le nombre d'arguments, qui peut être 0), et m désigne l'arité de la fonction g. Le cas $m = 0$ permet de construire des fonctions avec g d'arité 0.
- $\mathsf{Rec}(g, h)$ la fonction définie par récurrence (à droite) comme

$$\begin{cases} f(x_1, \dots, x_{n-1}, 0) = g(x_1, \dots, x_{n-1}), \\ f(x_1, \dots, x_{n-1}, m+1) = h(x_1, \dots, x_{n-1}, m, f(x_1, \dots, x_{n-1}, m)), \end{cases}$$

Question 2 (3 points)

La fonction SommeDiviseurs est définie de la façon suivante : si $n \in \mathbb{N}$, alors SommeDiviseurs(n) renvoie la somme des diviseurs entiers positifs de n (n non compris).

Par exemple SommeDiviseurs$(6) = 1 + 2 + 3 = 6$.

Montrez que SommeDiviseurs est primitive récursive.

On pourra définir une fonction auxiliaire avec un argument de plus. On pourra également utiliser la fonction modulo$(n, m) \mapsto n[m]$ *(ou* $n \mod m$*) supposée primitive récursive, et une définition par cas* ite$(x, y, b) =$ *if* $(b > 0)$ *then* x *else* y *supposée aussi primitive récursive.*

Solution:

On peut définir $sommebis(n, i)$ qui fait la somme des diviseurs de n jusqu'à i. On a assez facilement :

- $sommebis(n, 0) = 0$
- $sommebis(n, i+1) = sommebis(n, i) + \begin{cases} i+1 & \text{si modulo}(n, i+1) > 0 \\ 0 & sinon \end{cases}$

Il resterait à recoller les morceaux.

Question 3 (3 points)

Soit g une fonction récursive primitive.
Soit f la fonction définie par $f(0, x) = g(x)$ et $f(n+1, x) = f(n, f(n, x))$.

Montrez que f est récursive primitive.

On commencera par prouver proprement que $f : (n, x) \mapsto g^{2^n}(x)$*, et on pourra utiliser sans preuve le fait que la fonction* 2_puis $: n \mapsto 2^n$*, est récursive primitive.*

Solution:

La première chose à faire est de prouver une spécification alternative de f :

$$\forall n \in \mathbb{N}, f(n, x) = g^{2^n}(x).$$

Ce résultat se prouve aisément par récurrence :

- $f(0, x) = g(x) = g^{2^0}(x)$;
- $f(n+1, x) = f(n, f(n, x))$ en appliquant deux fois l'hypothèse de récurrence, on obtient $f(n+1, x) = g^{2^n}(g^{2^n}(x)) = g^{2^{n+1}}(x)$.

On va donc commencer par définir par récurrence une fonction h telle que $h(x, n) = g^n(x)$.

- $h(x, 0) = x = \mathsf{id}_1^1(x)$;
- $h(x, n+1) = g(h(x, n)) = \mathsf{Comp}_3(g, \mathsf{id}_3^3)(x, n, h(x, n))$.

Donc on peut définir h par :

$$h := \mathsf{Rec}(\mathsf{id}_1^1, \mathsf{Comp}_3(g, \mathsf{id}_3^3)).$$

Avec h et 2_puis on peut finalement écrire $f(n, x) = h(x, \text{2\_puis}(n))$, soit :

$$f := \mathsf{Comp}_2(\mathsf{Rec}(\mathsf{id}_1^1, \mathsf{Comp}_3(g, \mathsf{id}_3^3)), \mathsf{id}_2^2, \mathsf{Comp}_2(\text{2\_puis}, \mathsf{id}_2^1))$$

3 Indécidabilité

Question 4 (2 points)
Montrez, par réduction à partir du problème de l'arrêt, que le problème suivant est indécidable :

entrée : une machine de Turing M qui s'arrête sur toutes les données.
question : $L(M)$ est-il infini ?

Solution:
Soit $< M, w >$ une instance du problème de l'arrêt. On construit M_0 comme suit :

- M_0 prend en entrée un compteur c.
- Si M s'arrête en moins de c étapes, M_0 accepte, sinon elle rejette.

M_0 accepte c ssi M s'arrête en moins de c étapes sur w. $L(M_0) = \{c \in \mathbb{N}$tels que $c \geq$ longueur du calcul de M sur $w\}$ est donc infini ssi M s'arrête sur w.

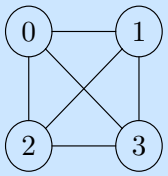
4 Problème : NP-complétude

Le problème Clique consiste à établir si un graphe G donné contient une clique de cardinal égal à un entier donné k. Un graphe G est défini par la donnée de son ensemble de sommets V et de l'ensemble de ses arêtes $E \subseteq V \times V$. Le cardinal d'un graphe est son nombre de sommets. Un ensemble $C \subseteq V$ de sommets de G est une *clique* si tous ces sommets sont reliés entre eux dans G, c'est à dire :

$$\forall u, v \in C, u \neq v \Rightarrow (u, v) \in E.$$

Question 5 (1 point)
Donnez un exemple de graphe contenant une clique de cardinal quatre.

Solution:
On pose $G = (V, E)$ avec $V = \{0, 1, 2, 3\}$ et $E = V \times V$. Il est trivial de vérifier que V lui-même est une clique de cardinal quatre dans G.



Le problème Clique s'énonce de la manière suivante :

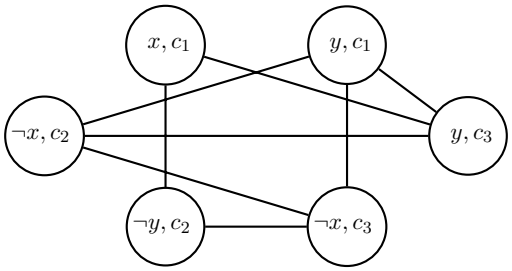
> Clique :
> **entrée :** un graphe $G = (V, E)$ et un entier k
> **question :** existe-t-il une clique C dans G de cardinal égal à k ?

Nous allons monter que ce problème est NP-complet, en le réduisant depuis Sat. Pour cela nous allons suivre une preuve proposée par Richard Karp en 1972.

Soit F une formule en forme normale conjonctive. On définit le graphe $G_F = (V_F, E_F)$ comme suit :

- les sommets de ce graphe sont les paires (ℓ, c), où c est une clause de F et ℓ est un littéral (c'est à dire une variable ou sa négation) qui apparaît dans c ;
- les sommets (ℓ, c) et (ℓ', c') sont reliés dans le graphe si et seulement si $c \neq c'$ et ℓ n'est pas la négation de ℓ'.

Par exemple si $F = c_1 \wedge c_2 \wedge c_3$ avec $c_1 = (x \vee y)$, $c_2 = (\neg x \vee \neg y)$ et $c_3 = (\neg x \vee y)$, G_F est le graphe suivant :



Question 6 (3 points)

Montrez que si F est constituée de n clauses, alors F est satisfiable si et seulement si G_F contient une clique de cardinal n. *On prouvera soigneusement les deux implications en construisant une clique dans un sens, et une interprétation des formules dans un autre.*

Solution:

Si F est satisfiable, soit I un modèle de F. Pour chaque clause c_k de F, il y a au moins un littéral $\ell(c_k)$ dans c_k tel que $[\ell(c_k)]_I = 1$. Considérons un tel ensemble $C_I = \{(\ell(c_k), c_k)\}$. F a n clauses, donc le cardinal de C_I vaut n. De plus, si (ℓ, c) et (ℓ', c') sont deux éléments différents de C_I, alors $c \neq c'$ et comme $[\ell]_I = [\ell']_I = 1$, ℓ et ℓ' ne sont pas la négation l'un de l'autre. Donc $((\ell, c), (\ell', c')) \in E_F$. C_I est donc une clique de cardinal n dans G_F.

Supposons au contraire qu'on a une clique $C \subseteq V_F$ de cardinal n. On peut définir une interprétation I_C de la manière suivante :

$$I_C(p) = \begin{cases} 1 & \text{si } (p, c) \in C, \text{ pour une certaine clause } c \\ 0 & \text{si } (\neg p, c) \in C, \text{ pour une certaine clause } c \\ 0 & \text{sinon} \end{cases}$$

Comme C est une clique, pour tous (ℓ, c) et (ℓ', c') différents dans C ces deux sommets sont reliés dans G_F, ce qui implique en particulier que c et c' sont des clauses différentes. Comme de plus le cardinal de C est égal au nombre de clauses de F, on en déduit que pour chaque clause c de F il y a un littéral ℓ tel que $(\ell, c) \in C$. Par définition de I_C, on a $[\ell]_{I_C} = 1$. Par conséquent comme I_C valide au moins un littéral dans chaque clause de F, $[F]_{I_C} = 1$, ce qui signifie que F est satisfiable.

Question 7 (2 points)

On définit la taille $|F|$ d'une formule F par le nombre de littéraux qui la constituent (dans l'exemple ci-dessus, $|F| = 6$), et la taille $|G|$ d'un graphe comme la somme de son cardinal (nombre de sommets) et de son nombre d'arêtes.
Bornez $|G_F|$ par un polynôme en $|F|$
(c'est-à-dire trouvez un polynôme p tel que $|G_F| \leqslant p(|F|)$).

Solution:
D'après la définition de G_F, on voit que chaque littéral de F donne lieu à au plus un sommet dans V_F. On en déduit que le cardinal de V_F est inférieur à $|F|$. De plus, le nombre d'arêtes étant borné par le carré du nombre de sommets, on a :

$$|G_F| = |V_F| + |E_F| \leqslant |V_F| + |V_F|^2 \leqslant |F| + |F|^2.$$

Question 8 (2 points)
En utilisant les résultats précédents, prouvez que Clique est NP-complet.

Solution:
On commence par montrer que Clique est NP-dur. Pour cela on donne une réduction polynomiale depuis Sat. Soit ρ la fonction qui à une formule F en forme normale conjonctive associe (G_F, n), avec n le nombre de clauses de F. D'après la question 7, on sait que G_F est de taille polynomiale par rapport à la taille de F, et il est évident à partir de la définition de G_F que la construction de ce graphe est linéaire en sa taille. Le calcul de n ne pose pas plus de problème. Donc ρ est une fonction P. D'après la question 6, on sait que :

$$F \in \text{Sat} \Leftrightarrow \rho(F) = (G_F, n) \in \text{Clique}.$$

Par conséquent, ρ est bien une réduction polynomiale de Sat à Clique, et comme Sat est NP-complet, Clique est NP-dur.
Il faut encore montrer que Clique appartient bien à la classe NP. Considérons l'algorithme non déterministe suivant :

input : Un graphe $G = (V, E)$ et un entier n
output: Un booléen, indiquant si G possède une clique de taille n.

$N \leftarrow 0$;
$C \leftarrow \emptyset$;
pour $v \in V$ **faire**
  choix entre
    $C \leftarrow C \cup \{v\}$;
    $N \leftarrow N + 1$;
  ou
    ne rien faire ;
  finchoix
finpour
si $N = n$ **alors**
  pour $u, v \in C$ **faire**
    si $u \neq v$ *et* $(u, v) \notin E$ **alors**
      `retourner (faux);`
    finsi
  finpour
  `retourner (vrai) ;`
sinon
  `retourner (faux);`
finsi

Cet algorithme calcule de manière non-déterministe ("devine") une clique , puis vérifie que c'est bien une clique de taille n. Il s'exécute en temps polynomial par rapport à $|G|$. Par conséquent on a établi que CLIQUE est bien dans la classe NP.