Fondements de l'informatique. Examen
Durée: 3h

Sujet proposé par Olivier Bournez
Version 9
(corrigé)

Les 4 parties sont indépendantes, et peuvent être traitées dans un ordre quelconque. On pourra admettre le résultat d'une question pour passer aux questions suivantes. On pourra utiliser tous les résultats et les théorèmes démontrés ou énoncés en cours ou en petite classe ou dans le polycopié sans chercher à les redémontrer.

Il est possible d'avoir la note maximale sans répondre à toutes les questions. La difficulté des questions n'est pas une fonction linéaire ni croissante de leur numérotation.

La ***qualité et clarté de votre argumentation et de votre rédaction*** *sera une partie importante de votre évaluation.*

1 A propos des gestes barrières

Question 1. *Parmi les questions suivantes, lesquelles sont décidables ? Récursivement énumérable ? dans P ? Justifier votre réponse.*

1. *Déterminer si une machine de Turing*[1] *M est telle qu'elle accepte un langage qui contient le mot* appliquonslesgestesbarrieres.
2. *Déterminer si un programme* Python *contient une variable dont le nom est* appliquonslesgestesbarrieres *et l'instruction "*appliquonslesgestesbarrieres*=1"*
3. *Déterminer si un programme* Python *possède une variable dont le nom est* appliquonslesgestesbarrieres*, qui sera modifiée à un moment lorsqu'il sera exécuté.*

Solution : Le premier problèmes est indécidable. C'est une application directe du Théorème de Rice : il y a une machine qui reconnaît le langage réduit à ce mot, et une machine de Turing qui accepte le mot vide, et donc la propriété est bien non triviale. Il est récursivement énumérable, car il suffit de simuler la machine M sur la chaîne de caractère *appliquonslesgestesbarrieres* et d'accepter si et seulement si la simulation accepte pour le reconnaître. Il n'est pas dans P car les langages de P sont décidables.

Le second problème est décidable (et donc récursivement énumérable) et est dans P : il suffit de parcourir le programme à la recherche de ce mot.

Le troisième problèmes est indécidable : Le problème universel se réduit à ce problème. Étant donnée une instance (M, w) du problème universel, on peut construire un programme $P_{M,w}$ en *Python*, qui possède une variable dont le nom est *appliquonslesgestesbarrieres*, qui simule la machine de Turing M sur w, et si cette simulation termine avec M qui accepte, ajoute 1 à la variable *appliquonslesgestesbarrieres* : la fonction qui à M et w associe $P_{M,w}$ est bien facilement calculable.

Il est récursivement énumérable, car il suffit de simuler le programme pour déterminer si la propriété est vérifiée. Il n'est pas dans P car les langages de P sont décidables.

□

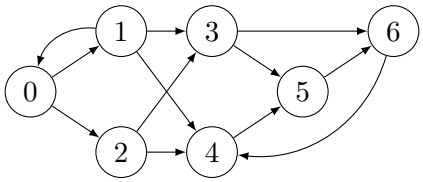
1. Qui travaille sur l'alphabet des symboles de l'alphabet Latin.

2 NP-complétude de Coupe-cycles

A propos des notations : On note $G = (V, E)$ pour un graphe dont l'ensemble des sommets est V, et $E \subseteq V \times V$ ses arêtes (dans le cas non-orienté) ou ses arcs (dans le cas orienté). On notera (u, v) pour l'arête de u vers v dans le cas non-orienté, et $(u \to v)$ pour l'arc de u vers v dans le cas orienté. Rappel : un cycle (orienté) de longueur n est une suite d'arcs du type $(u_1 \to u_2)$, $(u_2 \to u_3)$, $\ldots(u_{n-1} \to u_n)$ et $(u_n \to u_1)$. Une boucle est un cycle de longueur 1, c'est-à-dire un arc $(u \to u)$.

Soit $G = (V, E)$ un graphe orienté sans boucle. Un **coupe-cycles** est un ensemble d'arcs du graphe, tel que la suppression de ces arcs rend le graphe sans cycle.

Par exemple l'ensemble des arcs $\{(1 \to 0), (6 \to 4)\}$ est un coupe-cycles du graphe orienté suivant :



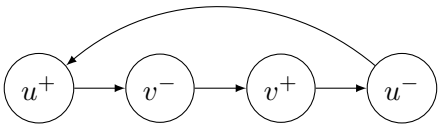
On rappelle qu'une *couverture de sommets* d'un graphe $G = (V, E)$ est un ensemble $S \subseteq V$ de sommets tel que toutes les arêtes de G ont au moins une extrémité dans S.

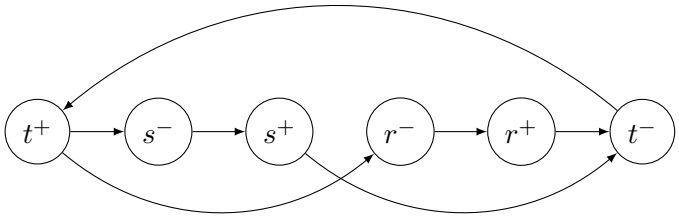
On rappelle/admettra que le problème COUVERTURE-DE-SOMMETS *suivant est NP-complet :*
***Données** : un graphe non-orienté G sans boucle et un entier k.*
***Question** : G contient-il une couverture de sommets avec au plus k sommets.*

On introduit la transformation $\mathcal{F}$ qui à un graphe $G = (V, E)$ non-orienté associé le graphe orienté $\mathcal{F}(G) = (V', E')$ défini par :
— $V' = \{u^-, u^+ : u \in V\}$;
— $E' = \{(u^- \to u^+) : u \in V\} \cup \{(u^+ \to v^-), (v^+ \to u^-) : (u, v) \in E\}$.

Question 2.
— *Représentez graphiquement $\mathcal{F}(G)$ et décrire un coupe-cycles de $\mathcal{F}(G)$ de la taille la plus petite possible pour $G = (V, E)$ avec $V = \{u, v\}$ et E=$\{(u, v)\}$.*
— *Même question pour $G = (V, E)$ avec $V = \{s, t, r\}$ et E=$\{(s, t), (t, r)\}$.*

Solution :

$\mathcal{F}(G_1)$ est donné par  et admet $\{(u^- \to u^+)\}$ comme coupe-cycles



$\mathcal{F}(G_2)$ est donné par et admet $\{(t^- \to t^+)\}$ comme coupe-cycles.

□

On note $|S|$ pour le nombre d'éléments de l'ensemble S.

Question 3. *Montrer que si C est un coupe-cycles du graphe $\mathcal{F}(G)$, alors il existe un ensemble S avec $|S| \leq |C|$ qui est une couverture de sommets du graphe G.*

Solution :

On considère S tel que

— $u \in S$ si $(u^- \to u^+) \in C$ ou si $(u^+ \to v^-) \in C$.

Puisque cela garantit $|S| \leq |C|$, il suffit de prouver que S est une couverture sommets de G.

Soit $e = (u, v)$ une arête de G. Observons que $u^-u^+v^-v^+u^-$ est un cycle de $\mathcal{F}(G)$. Cela signifie qu'au moins un des ces 4 arcs doit être dans C.

— Si $(u^- \to u^+) \in C$ alors cela signifie que u est dans S.

— Si $(v^- \to v^+) \in C$ alors cela signifie que v est dans S.

— Si $(v^+ \to v^-) \in C$, alors cela signifie que v est dans S.

— Si $(u^+ \to u^-) \in C$, alors cela signifie que u est dans S.

Donc pour les 4 cas, S couvre bien l'arête e. Donc S est bien une couverture de sommets. □

Question 4. *Montrer que si S est une couverture de sommets du graphe non-orienté G, alors l'ensemble $\{(u^- \to u^+) : u \in S\}$ est un coupe-cycles du graphe $\mathcal{F}(G)$.*

Solution : [Sur le vocabulaire utilisé dans ce qui suit : lorsqu'on a un arc $(u \to v)$, on dit que cet arc est un arc sortant en u, et entrant en v.]

Pour tout sommet u, le seul arc sortant en u^- va vers u^+. Les arcs entrants sont tous vers des v^-, pour $v \in V$. Par conséquent, un cycle doit donc alterner les $-$ et les $+$.

Soit C un cycle de $\mathcal{F}(G)$. Le cycle C est de longueur paire par la remarque précédente. Il ne peut pas être de longueur 2 car il n'y a pas de boucle dans $\mathcal{F}(G)$. Il est donc de la forme $u^-u^+v^-v^+ \dots u^-$ pour un certain u et un certain v, et une arête (u, v) de G. Puisque S est une couverture de sommets, soit u soit v est dans S, et donc l'un des arcs $(u^- \to u^+)$ ou $(v^- \to v^+)$ coupe le cycle et appartient à l'ensemble $\{(u^- \to u^+) : u \in S\}$.

□

Question 5. *Démontrer que le problème de décision suivant est NP-complet :*
***Données** : un graphe orienté G sans boucle et un entier k.*
***Question** : G contient-il un coupe-cycles avec au plus k arcs ?*

Solution : Le problème est dans NP, car la donnée d'une couple-cycles est un certificat vérifiable en temps polynomial : il suffit de vérifier qu'il ne possède pas plus de k arcs, et que c'est bien un coupe-cycles du graphe (ce qui peut se faire en vérifiant que le graphe obtenu en enlevant ces arcs est sans cycles). Tout cela peut bien se faire en temps polynomial.

Les questions précédentes donnent une réduction via la fonction $\mathcal{F}$ entre le problème de couverture de sommets et ce problème. Il ne reste plus qu'à observer que $\mathcal{F}$ se calcule clairement en temps polynomial pour conclure. □

3 Un peu de définissabilité

Question 6. *Soit T une théorie sur une signature du calcul des prédicats qui contient le symbole $=$.*

Montrer que si T possède des modèles égalitaires dont l'ensemble de base est de taille arbitrairement grande, alors T possède un modèle égalitaire dont l'ensemble de base est infini.

Solution : Par hypothèse, toute partie finie de $T \cup \{F_n : n \in \mathbb{N}\}$ possède un modèle, où $F_n = \exists v_1 \exists v_2 \dots \exists v_n \bigwedge_{1 \leq i < j \leq n} \neg v_i = v_j$ exprime qu'il y a au moins n éléments.

Par le théorème de compacité, $T \cup \{F_n : n \in \mathbb{N}\}$ possède un modèle. Ce modèle a son ensemble de base infini, puisque il possède au moins n éléments pour tout n. □

Question 7. *Supposons que T est une théorie sur une signature du premier ordre avec un nombre fini de formules.*

Supposons que l'on ait deux théories Σ et Λ avec la propriété que pour tout modèle $\mathcal{A}$ de T, $\mathcal{A}$ est un modèle de Σ si et seulement si $\mathcal{A}$ n'est pas un modèle de Λ.

Montrer que les théories $T \cup \Sigma$ et $T \cup \Lambda$ sont finiment axiomatisables : on peut construire pour chacune une théorie avec un nombre fini de formules qui leur est équivalente.

Solution : Par hypothèse, $T \cup \Sigma \cup \Lambda$ ne possède aucun modèle. Par le théorème de compacité, il y a un sous-ensemble fini de celle-ci qui ne possède aucun modèle : donc un sous-ensemble T de T, Σ' de Σ, et Λ' de Λ tel que $T' \cup \Sigma' \cup \Lambda'$, et donc à fortiori $T \cup \Sigma' \cup \Lambda'$ ne possède aucun modèle.

Un modèle de $T \cup \Sigma'$ est donc un modèle de T et de la formule Φ qui est la conjonction de la négation de toutes les formules de Λ'. La propriété est vraie à fortiori pour les modèles de $T \cup \Sigma$. Réciproquement, un modèle de Φ et de T doit être un modèle de $T \cup \Sigma$, par la propriété de l'énoncé. Autrement dit, $T \cup \Sigma$ s'axiomatise par $T \cup \{\Phi\}$.

Symétriquement, $T \cup \Lambda$ s'axiomatise par $T \cup \{\Psi\}$, où Ψ est la conjonction de la négation de toutes les formules de Σ'. □

4 Programmer avec des additions de vecteurs ordonnés

On considère un langage de programmation inspiré par une variation du modèle des *VECTOR ADDITION SYSTEMS* et de considérations sur un autre modèle dû à John Conway : un programme $VASS_{ord}$ est une liste V ordonnée finie de vecteurs de $\mathbb{Z}^m$ pour un certain entier m.

On exécute le programme V sur une entrée $\vec{n} \in \mathbb{N}^m$ de la façon suivante :

1. pour le premier vecteur $\vec{v}$ dans la liste V pour lequel $\vec{n} + \vec{v}$ est un élément de $\mathbb{N}^m$, on remplace $\vec{n}$ par $\vec{n} + \vec{v}$.
2. on répète la règle 1. tant qu'on peut appliquer cette règle, c'est-à-dire, jusqu'à ce qu'il n'y ait plus de vecteur $\vec{v}$ avec $\vec{n} + \vec{v}$ qui reste dans $\mathbb{N}^m$. On dit alors que le résultat du calcul est ce (dernier) vecteur $\vec{n}$.

(si la règle 1. s'applique pour toujours, on dit que V ne termine pas sur l'entrée $\vec{n}$).

Par exemple, le programme $VASS_{ord}$ donné par les vecteurs

$$V = (-1, -1, 0), (0, -1, 1)$$

appliqué à l'entrée $(2, 3, 0)$ va terminer avec $(0, 0, 1)$: en effet, il va transformer $\vec{n} = (2, 3, 0)$ en $(1, 2, 0)$ puisque $(2, 3, 0) + (-1, -1, 0) = (1, 2, 0)$, puis en $(0, 1, 0)$ puisque $(1, 2, 0) + (-1, -1, 0) = (0, 1, 0)$, puis en $(0, 0, 1)$ puisque $(0, 1, 0) + (-1, -1, 0)$ n'est pas un élément de $\mathbb{N}^3$, mais que $(0, 1, 0) + (0, -1, 1) = (0, 0, 1)$.

Question 8. *Que retourne ce programme sur $n = (a, b, 0)$ selon la valeur des entiers naturels a et b ?*

Solution : Le premier vecteur décremente a et b de 1 tant que cela est possible. On va donc appliquer ce vecteur jusqu'à obtenir $(a-b, 0, 0)$ si $a > b$ ou $(0, b-a, 0)$ si $b > a$, et $(0, 0, 0)$ si $a = b$. Le second vecteur va alors pouvoir s'appliquer que si $b - a > 0$, et tant que la seconde composante n'est pas nulle. Chaque application de ce vecteur décrémente de 1 la seconde composante, et augmente de 1 la troisième composante. En répétant, on va donc se retrouver avec $(0, 0, b - a)$, où cette fois plus aucun vecteur ne s'applique.

Le résultat est donc $(a - b, 0, 0)$ si $a \geq b$ et $(0, 0, b - a)$ sinon. □

*On rappelle que les machines à compteurs sont des machines qui n'ont que des instructions du type incrémenter un compteur (*Inc*), décrémenter un compteur (*Decr*), tester si un compteur est nul (*IsZero*) et arrêt (*Halt*) (voir par exemple les transparents du cours 6).*

On note $p_1 = 2$, $p_2 = 3$, $p_3 = 5$, ..., et ainsi de suite, les nombres premiers.

On rappelle que $f : \mathbb{N}^m \to \mathbb{N}$ est une fonction partielle calculable s'il existe une machine de Turing M, avec m rubans, telle si l'on place initialement le codage de n_i en binaire sur le ruban i, alors si $(n_1, \ldots, n_m)$ est dans le domaine de f, alors M s'arrête avec $f(n_1, \ldots, n_m)$ sur son premier ruban et tous les autres rubans vides, et M ne s'arrête pas si $(n_1, \ldots, n_m)$ n'est pas dans le domaine de f.

Question 9. *Soit $f : \mathbb{N}^m \to \mathbb{N}$ une fonction partielle calculable.*

Justifier pourquoi on peut construire une machine à 2 compteurs telle que si on l'exécute sur l'entier $n = 2^{n_1}3^{n_2}5^{n_3}7^{n_4}\ldots p_m^{n_m}$ dans son premier compteur, elle s'arrêtera avec l'entier $2^{f(n_1,n_2,\ldots,n_m)}$ dans son premier compteur lorsque $(n_1, \ldots, n_m)$ est dans le domaine de f, et ne s'arrêtera pas sinon.

Solution : Le cours explique comment simuler une machine de Turing par une machine à 2-compteurs (en passant par les machines à 2 piles). La machine obtenue fait essentiellement ce travail quand elle simule la machine de Turing qui calcule f. On peut aussi passer via des machines à $2m$ piles si on préfère, pour gérer les m arguments/rubans.

Mais en réalité, ce n'est pas tout à fait cela car la machine M travaille sur des codages en binaire dans la définition plus haut, et non pas en unaire : le codage de n en binaire ne va pas correspondre à k^n dans la simulation du cours pour k un nombre premier.

Mais dans la définition plus haut de fonction partielle calculable, on pourrait remplacer "binaire" par "unaire", quitte à convertir les représentations au début et à la fin du calcul quand on s'arrête. Cette fois, la simulation du cours de cette dernière machine par une machine à compteur fait le travail voulu. □

Question 10. *Supposons que l'on change le type des instructions autorisées dans les machines à compteur. Au lieu d'avoir des instructions du type* $\mathrm{IsZero}(c, j, k)$*, on autorise des instructions du type* $\mathrm{IsZero}'(c, j, k)$ *dont l'effet est le suivant : cela teste si le compteur c est nul, et on va à l'instruction j si c'est le cas, et sinon on diminue de 1 le compteur de c et on va à l'instruction k.*

Supposons par ailleurs que le programme n'est autorisé à ne contenir qu'une unique instruction Halt*.*

Justifier pourquoi cela ne change rien à la puissance du modèle obtenu des machines à compteur, en termes de ce qu'il est capable de calculer.

Solution : Il suffit de remplacer chaque instruction $\mathrm{IsZero}(c, j, k)$ par $\mathrm{IsZero}'(c, j, k')$ où k' est une nouvelle instruction $\mathrm{Inc}(c, j, k)$, et le nouveau programme simulera l'ancien.

De façon duale, un programme avec cette instruction peut être simulé par les machines du cours, en remplaçant chaque instruction $\mathrm{IsZero}'(c, j, k)$ par $\mathrm{IsZero}(c, j, k')$ où k' est une nouvelle instruction $\mathrm{Decr}(c, j, k)$.

On peut aussi supposer qu'il n'y a qu'une unique instruction Halt, en remplaçant chaque instruction Halt par $\mathrm{Inc}(1, n)$ où n et $n + 1$ sont les instructions $\mathrm{Decr}(1, n + 1)$ et Halt.

[On peut aussi simplement remplacer les adresses de saut vers Halt dans les instructions qui en ont, si vous préférez.] □

Question 11. *Soit M une machine à 2-compteurs. Expliquer comment on peut construire un programme* VASS_{ord} *qui simule sur $\vec{n} = (n, 0, \ldots, 0)$ l'évolution de M sur l'entrée n.*

[Indication : On pourra d'abord chercher à simuler sur $\vec{n} = (n, 0, 1, 0, \ldots, 0)$ l'évolution de M sur l'entrée n, en travaillant dans $\mathbb{N}^m$ avec m qui est le nombre d'instructions de M plus 2.]

Solution : L'idée est de simuler le fait que la valeur des 2 compteurs de M valent a, b et que la machine est dans l'instruction q par le fait que $\vec{n}$ vaut $(a, b, 0, \ldots, 0, 1, 0, \ldots, 0)$, où le 1 est en

position q. On écrira un tel vecteur (abusivement) $(a, b, \vec{e}_q)$, où $\vec{e}_q$ est le vecteur qui ne contient que des 0, sauf pour la composante q qui vaut 1.

En utilisant la question précédente, on peut supposer que les instructions sont avec l'opérateur $\text{IsZero}'(c, j, k)$.

Pour simuler $\text{Inc}(a, j)$ à l'instruction i, on introduit le vecteur $(1, 0, \vec{e}_j - \vec{e}_i)$.

Pour simuler $\text{Decr}(a, j)$ à l'instruction i, on introduit le vecteur $(-1, 0, \vec{e}_j - \vec{e}_i)$.

Pour simuler $\text{IsZero}'(a, j, \ell)$ à l'instruction i, on introduit les vecteurs $(-1, 0, \vec{e}_\ell - \vec{e}_i)$ et $(0, 0, \vec{e}_j - \vec{e}_i)$ dans cet ordre.

Pour les instructions où le premier argument vaut b, on intervertit la première et deuxième coordonnées dans ces vecteurs.

Cela résout le problème suggéré par l'indication.

Il reste à faire démarrer la simulation : donc à remplacer $\vec{n} = (n, 0, \dots, 0)$ par $\vec{n} = (n, 0, 1, 0, \dots, 0)$: il suffit d'ajouter le vecteur $(0, 0, 1, 0, \dots, 0)$ tout à la fin de la liste de vecteurs. On utilise le fait qu'il ne pourra s'appliquer que tout au début, car dans tout autre cas, un autre vecteur de ceux construit plus haut s'appliquera.

[On peut faire observer que cette façon de faire construit un programme $VASS_{ord}$ qui simule M jusqu'à ce qu'il termine, mais lorsque M termine ne s'arrête pas lui, puisque le vecteur $(0, 0, 1, 0, \dots, 0)$ à la fin s'applique toujours : il boucle quand M s'arrête. Si l'on veut éviter cela, on peut, au lieu d'ajouter $(0, 0, 1, 0, \dots, 0)$ à la fin, plutôt ajouter un programme au début qui transforme $(n, 0, \dots, 0)$ en $(0, 0, n, 1, 0, \dots)$ quitte à travailler dans $\mathbb{N}^{m+2}$, et ajouter deux composantes nulles à chaque vecteur de la simulation précédente. En supposant sans perte de généralité que $n \neq 0$, une façon de faire cela est de mettre les vecteurs

$$(-1, 1, 1, -1, 0, \dots, 0), (0, -1, 0, 1, \dots, 0,), (-1, 0, 1, 1, 0, \dots, 0)$$

au début du programme $VASS_{ord}$.]

□

John Conway a décrit un programme qui génère tous les nombres premiers : si on l'exécute sur $(2, 0, \dots, 0)$, il ne terminera pas, mais à certains instants $\vec{n}$ sera de nouveau de la forme $(2^p, 0, \dots, 0)$, et cela se produira exactement pour p un entier premier. Et par ailleurs, le programme génère les entiers p dans ce sens dans l'ordre croissant.

Question 12. *Expliquer pourquoi il existe un tel programme. On ne demande pas de produire explicitement ce programme* VASS_{ord}*, mais d'expliquer comment il peut être obtenu.*

Note culturelle : John Conway a décrit explicitement un tel programme constitué de 14 vecteurs de $\mathbb{Z}^{10}$, qu'il a appelé les 14 vecteurs[2] *fantastiques.*

Solution : On peut construire une machine de Turing, telle que si on lui donne $n = p_i$ le $i$ème nombre premier, elle renvoie p_{i+1} : par exemple, de façon brutale, en tentant de diviser $n+1$ par tous les entiers plus petit que lui, et en passant à $n+2$ si l'on en trouve un, puis à $n+3$ si l'on en trouve un autre, et ainsi de suite, jusqu'à trouver un nombre premier.

En utilisant la question 9, cela peut aussi se faire avec une machine à compteur. Cette dernière peut se simuler par un programme $VASS_{ord}$ par ce qui précède. En ajoutant le vecteur $(0, 0, \vec{e}_1 - \vec{e}_{halt})$ de façon à ce que l'(unique) instruction numéro *halt* qui contient Halt fasse reboucler sur la première instruction, on obtiendra exactement ce qui est désiré. □

En réalité, il n'y a pas que les nombres premiers que l'on peut programmer en $VASS_{ord}$. On peut programmer en $VASS_{ord}$ toutes les fonctions calculables (et exactement elles) :

Question 13. *Démontrer qu'il existe un programme* VASS_{ord} *universel : il existe une liste de vecteurs V, telle que pour toute fonction $f : \mathbb{N} \to \mathbb{N}$ partielle calculable, il existe un entier C tel que, pour tout entier n, le programme F appliqué sur le nombre $(C, 2^{2^n}, 0, \dots, 0)$ retourne le résultat $(2^{2^{f(n)}}, 0, \dots, 0)$ si et seulement si f est définie en n.*

2. En réalité, le modèle de Conway parle de fractions, et il parle des 14 fractions fantastiques.

Solution : On a démontré en cours qu'il existait une machine de Turing universelle M. Dans le cours on utilise un codage par les mots, mais quitte à voir les entiers comme leur codage en binaire, et un mot sur l'alphabet $\{0,1\}$ comme l'écriture d'un entier, on peut aussi considérer que cette machine universelle fonctionne sur des entiers plutôt que sur des mots : on fixe un codage des machines de Turing par les entiers, et cette machine de Turing universelle M est telle que si on lui donne c et n, elle simule la machine de Turing qui correspond à c sur n [Ce paragraphe est un peu une remarque pour les plus puristes, car on a fait cela en PCs parfois implicitement].

On en déduit que la fonction f qui à 2^n et c associe 2^r où r est le résultat du calcul de la machine de Turing qui correspond à c sur l'entrée n est partielle calculable : il suffit de vérifier que l'entrée est de la forme 2^n puis d'utiliser cette machine de Turing universelle pour simuler la machine de Turing qui correspond à c sur n, et si cette simulation termine avec le résultat r, retourner 2^r.

Le programme $VASS_{ord}$ qui lui correspond par ce qui précède (questions 9, 11) est exactement ce que l'on cherche : l'entier C correspond à 3^c, où c est le codage de la machine de Turing qui calcule f. □

Question 14. *Peut-on énumérer (par un algorithme) les entiers C pour lesquels le programme* VASS_{ord} *universel précédent termine pour tout n ?*

Solution : Cela n'est pas possible. En effet, par contradiction, si cela était possible, étant donné un entier n, on pourrait considérer le $n$ème entier C_n, et considérer le résultat r_n du programme $VASS_{ord}$ sur n.

La fonction qui à n associe $r_n + 1$ serait donc une fonction totale calculable. Elle devrait donc correspondre à un entier C_p pour un entier p. Mais par construction, elle ne vaut pas r_p pour l'argument p (car elle vaut $r_p + 1$). Contradiction. □

La conjecture de Goldbach dit que tout entier pair strictement plus grand que 2 est la somme de deux entiers premiers. C'est l'un des exemples de conjecture parmi les plus célèbres pour laquelle on ne connaît aucune preuve (ni aucun contre-exemple).

On note $\mathcal{V}^+$ pour l'ensemble des programmes $VASS_{ord}$ V, dont les coefficients des vecteurs sont soit $-1, 0$ ou 1, et qui terminent en partant de $(1, 0, \ldots, 0)$.

Question 15. *Etant donné un entier m, on note $T(m)$ pour le plus grand nombre d'étapes qu'une liste $V \in \mathcal{V}^+$ de au plus m vecteurs*[3] *de $\mathbb{Z}^m$ effectue avant de terminer en partant de $(1, 0, \ldots, 0)$.*

Expliquer pourquoi $T(m)$ est bien définie pour toute valeur m.

Expliquer pourquoi vous savez explicitement produire un entier m_0 pour lequel déterminer la valeur $T(m_0)$ est au moins aussi difficile que de déterminer si la conjecture de Goldbach est vraie.

Solution : Quand m est fixé, il est facile de construire au moins un tel programme $V \in \mathcal{V}^+$ qui termine en partant de $(1, 0, \ldots, 0)$: choisir par exemple la liste réduite au vecteur $(0, -1, 0, \ldots, 0)$. Par ailleurs, il y a un nombre fini de vecteurs à coefficients dans $\{-1, 0, 1\}$. Par conséquent, $\Sigma(m)$ est donc le maximum d'un ensemble fini et non vide de valeurs, et donc est parfaitement défini pour tout entier m.

On peut construire une machine de Turing M qui termine si et seulement si la conjecture de Goldbach est fausse : construire une machine qui teste pour tous les entiers $n \geq 4$ si l'on peut trouver deux entiers inférieurs premiers tels que n est leur somme.

Elle possède n_0 états pour un certain entier n_0. Cette machine peut se simuler par certain programme $VASS_{ord}$ V qui terminera si et seulement si la machine de Turing termine. Les simulations précédentes n'utilisent que des vecteurs avec des coefficients dans $\{-1, 0, 1\}$.

Si on note m_0 le maximum de son nombre de vecteurs (que l'on peut calculer à partir de n_0), et de leur dimension, cela signifie que connaitre $T(m_0)$ est au moins aussi difficile que de résoudre la conjecture de Goldbach.

3. On peut donc aussi dire : de $\{-1, 0, 1\}^m$ au lieu de $\mathbb{Z}^m$.

En effet, si on le connaissait, disons que l'on saurait qu'il vaut n_0, pour décider si V termine, il suffirait de simuler V pendant n_0 transitions, et d'accepter si et seulement si V a accepté en moins de n_0 étapes.

[Note culturelle : En fait, plus généralement, la connaissance de $T(m)$ permettrait de connaître la réponse à de nombreuses conjectures mathématiques, et pas seulement de la conjecture de Goldbach. Et cet argument n'est pas spécifique aux programmes $VASS_{ord}$ (mais est lié à ce qu'on appelle les fonctions de castor-affairé] □