Fondements de l'informatique. Examen
Durée: 3h

Sujet proposé par Olivier Bournez
Version 4

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 Décidabilité

On considère des machines de Turing qui travaillent sur un alphabet qui contient au moins les chiffres $0, 1, \ldots, 9$.

Indication : aucune propriété avancée sur les décimales de π n'est nécessaire pour répondre à toutes les questions qui suivent.

Question 1. *Peut-on décider si le langage accepté par une machine de Turing M est constitué du mot qui correspond aux 42 premières décimales de π.*

Question 2. *On considère le problème de décision suivant :*
— ***Donnée****: Un entier k*
— ***Réponse****: Décider s'il y a k zéros consécutifs dans l'écriture décimale de π.*
Est-il décidable ? Dans NP ? Dans P ?

On dira qu'une machine de Turing M accepte une entrée w en temps quadratique si elle termine en temps n^2 où n est la longueur de w.

Question 3. *Etant donnés un mot w, et une machine de Turing M, peut-on décider si M accepte w en temps quadratique ?*

Question 4. *Peut-on décider si une machine de Turing M accepte toutes ses entrées, et ce, en temps quadratique ?*

Question 5. *On se donne une machine de Turing M qui termine sur toutes ses entrées. Peut-on décider la machine de Turing M accepte toutes ses entrées en temps quadratique ?*

2 Sur la nécessité d'autoriser des boucles infinies

Un langage de programmation est dit **Turing complet** si pour tout langage L décidable par une machine de Turing, il y a un programme de ce langage de programmation qui décide L.

Motivation : Le but de cette section est de prouver qu'il faut nécessairement permettre la possibilité de faire des boucles infinies dans les langages de programmation, sinon on n'est pas Turing-complet. Les langages comme Java, C et OCaml sont Turing-complets.

Nous supposons qu'un langage de programmation vérifie au minimum les hypothèses suivantes :

1. le langage de programmation permet une notion de décision : il y a un critère [1] qui permet de définir "acccepte" (et donc son complémentaire "refuse") pour le résultat de l'exécution d'un programme *lorsqu'il termine* sur une entrée donnée w ;
2. une machine de Turing M_1 peut vérifier qu'un texte correspond bien à un programme écrit avec ce langage de programmation ;
3. une machine de Turing M_2 est capable de le simuler : si on a un texte qui correspond bien à un programme écrit avec ce langage de programmation, et une entrée w, la machine M_2 avec le texte de ce programme et w simule [2] l'exécution du programme sur l'entrée w.

Question 6. *Supposons qu'un certain langage de programmation ne permet pas de faire des boucles infinies : ses programmes terminent toujours sur toute entrée.*

Montrer qu'il est impossible que ce langage de programmation soit Turing complet.

Indication : On pourra raisonner par l'absurde, et construire un langage L décidable [3] *tel qu'aucun programme du langage de programmation ne peut le décider.*

3 Programmes booléens

Motivation : Les programmes booléens constituent un exemple de langage de programmation qui n'est pas Turing complet par la section précédente. Cependant, nous allons montrer qu'ils suffisent pour défnir ce qu'est un problème NP-complet, et même suffisent à donner une preuve alternative au théorème de Cook-Levin.

Un *programme Booléen* est un programme avec un nombre fini d'instructions $I_1; I_2; \ldots; I_m$ où chaque instruction I et expression E, et variable X est définie de façon inductive [4] par les règles suivantes :

$$\begin{array}{lcl} I & \rightarrow & X := E | I_1; I_2 | if\ E\ then\ I_1\ else\ I_2 \\ E & \rightarrow & X | true | false | E_1 \vee E_2 | E_1 \wedge E_2 | \neg E_1 \\ X & \rightarrow & X_0 | X_1 | \ldots | X_v \end{array}$$

pour un nombre fini v de variables [5] $X_1, \ldots, X_v$.

Les programmes Booléens travaillent sur des variables qui sont booléennes : elles ne peuvent prendre que la valeur 1 (vrai) ou 0 (faux). Ils fonctionnent avec la sémantique attendue : par exemple, $X := Y \vee Z$ met dans la variable X le résultat du *ou logique* sur la valeur de la variable Y et Z. L'instruction $I; I'$ signifie effectuer l'instruction I puis l'instruction I'.

Remarque : Les programmes Booléens ne permettent pas de faire des boucles : si l'on se donne un tel programme, il va effectuer un nombre fini m d'instructions et s'arrêter.

On s'intéresse au problème de décision suivant :
SAT-PROGRAMME-BOOL

1. Par exemple, le fait qu'une certaine variable vaille 1, qu'il affiche "oui", etc...
2. Et en particulier, quand le programme termine, M_2 est capable de détecter si le résultat de son exécution satisfait le critère de "accepte" ou de son contraire "refuse".
3. Par machine de Turing.
4. La première ligne veut dire par exemple, que l'ensemble des instructions est le plus petit ensemble de mots, tel que $X := E$ est une instruction, et si I_1 et I_2 sont des instructions, alors $I_1; I_2$ est une instruction, et si E est une expression, alors $if\ E\ then\ I_1\ else\ I_2$ est une instruction.
5. Comme dans l'exemple qui suit, et dans la suite, pour aider à la lisibilité, nous nous autoriserons à appeler ces variables par d'autres noms, comme $X, Y, Z, S, Go, etc.$.

— **Donnée**: Un programme Booléen
— **Réponse**: Déterminer s'il existe des valeurs pour chacune des variables $X_1, \ldots, X_m$ (dans $\{0,1\}$) telles que si l'on exécute le programme avec ces valeurs initiales, alors à la fin de l'exécution du programme la variable X_1 vaut 1.

Question 7. *Prouver que le problème* SAT-PROGRAMME-BOOL *est NP-complet. On pourra utiliser la NP-complétude du problème* 3-SAT.

Motivation : Nous allons chercher à obtenir dans le reste de cette section une preuve alternative du Théorème de Cook-Levin, et à montrer directement la NP-complétude du problème SAT-PROGRAMME-BOOL.

Question 8. *Considérons une machine de Turing V, sur l'alphabet $\Sigma = \{0,1\}$, qui fonctionne en temps polynomial : en temps $\pi(n)$ sur les entrées de taille n pour un polynôme π. Montrer que pour tout n, on peut construire un programme Booléen $\mathcal{P}_n$ tel que pour chaque entrée $w = w_1 \ldots w_n$, si l'on exécute ce programme $\mathcal{P}_n$ avec initialement $X_1 = w_1$, ..., $X_n = w_n$, alors ce programme se terminera avec $X_1 = 1$ si et seulement si V accepte le mot w.*

Indication : On fixera un codage des configurations de V par des variables booléennes que l'on précisera, et on écrira le programme Booléen $\mathcal{P}_n$ de la forme

$$Init; UneEtape; UneEtape; UneEtape; \ldots; UneEtape$$

où $Init$ est une ou plusieurs instructions que l'on précisera, et $UneEtape$ est une ou plusieurs instructions que l'on précisera, qui sera répété un nombre de fois que l'on précisera.

Montrer que l'on peut déterminer la description de ce programme Booléen $\mathcal{P}_n$ en temps polynomial en n.

Question 9. *Prouver que le problème* SAT-PROGRAMME-BOOL *est NP-complet, sans utiliser la NP-complétude d'un autre problème connu pour être NP-complet.*

On dira que deux instructions I et I' sont strictement équivalentes, si leur exécution modifie exactement les mêmes variables et de la même façon.

Question 10. *On considère l'instruction $\overline{I}$ définie par $X := (E \wedge Go) \vee (X \wedge \neg Go)$. Donner une instruction I strictement équivalente à $\overline{I}$ lorsque la variable booléenne Go est vraie. Même chose lorsque la variable booléenne Go est fausse.*

Plus généralement, considérons une instruction I, et une instruction J avec possiblement plus de variables. On dira que I est équivalente à J si leur exécution modifie exactement les mêmes variables communes et de la même façon, et ce quel que soit la valeur initiale des variables supplémentaires de J. Par exemple $X := 0$ est équivalente à $S := Go; X := 0; Go := S$.

Question 11. *Donner une instruction équivalente à l'instruction "$if\ U\ then\ J\ else\ K$" sans utiliser d'instruction "$if\ then\ else$", construite à partir de $\overline{J}$ et $\overline{K}$.*

Question 12. *Prouver que l'on peut transformer un programme Booléen en un programme équivalent (mais avec possiblement plus de variables) qui ne contient pas d'instruction du type "$if\ then\ else$". Montrer que cette transformation se calcule en temps polynomial.*

Question 13. *Prouver que l'on peut transformer un programme Booléen sans instruction du type "$if\ then\ else$" en un programme équivalent (mais avec possiblement plus de variables) qui ne contient pas d'instruction du type "$if\ then\ else$" et où chaque variable apparaît au plus une fois*[6] *à gauche du signe :=. Montrer que cette transformation se calcule en temps polynomial.*

6. Autrement dit, on a jamais $X := \ldots$ quelque part, et $X := \ldots$ autre part dans le programme, et ce pour toute variable X.

Question 14. *En déduire*[7] *la NP-complétude de* $\leq$ 3-SAT

— ***Donnée****: Une formule propositionnelle φ en forme normale conjonctive avec au plus 3 littéraux par clause*

— ***Réponse****: Déterminer si φ est satisfiable.*

Ainsi que de : 3-SAT

— ***Donnée****: Une formule propositionnelle φ en forme normale conjonctive avec exactement 3 littéraux par clause*

— ***Réponse****: Déterminer si φ est satisfiable.*

4 Axiomatisation et Axiomatisation finie

On fixe une signature.

On dit qu'une classe[8] de structures K est axiomatisable si l'on peut trouver une théorie T qui la caractérise : les structures de la classe sont exactement les modèles de T. On dit qu'une classe de structures K est finimement axiomatisable s'il on peut trouver une théorie T avec un nombre fini d'axiomes qui la caractérise.

Remarque : En particulier, si K est finiment axiomatisable, alors K est modèle d'une unique formule : la conjonction des formules de T.

Question 15. *Supposons que T axiomatise K. On suppose que K est finiment axiomatisable (possiblement par une autre théorie finie T'). Montrer que l'on peut trouver un nombre fini d'axiomes de T qui axiomatise aussi K.*
Indication : on pourra utiliser le théorème de compacité.

Question 16. *Montrer que si K est finiment axiomatisable alors K et son complémentaire sont axiomatisables.*

Question 17. *Montrer que la réciproque est vraie : si K et son complémentaire sont axiomatisables alors K est finiment axiomatisable.*
Indication : on pourra utiliser le théorème de compacité, et utiliser le fait que l'union des deux théories doit être contradictoire.

Question 18. *Nous avons vu dans le cours que la classe des corps de caractéristique 0 est axiomiatisable. Montrer que la classe des corps de caractéristique 0 n'est pas finiment axiomatisable.*

Question 19. *Montrer que la classe des corps de caractéristique $\neq 0$ n'est pas finiment axiomatisable.*

Question 20. *Nous avons vu dans le cours que la classe des corps algébriquement clos est axiomiatisable. Montrer que la classe des corps algébriquement clos n'est pas finiment axiomatisable.*

7. En utilisant uniquement les questions 8 à 13, c'est-à-dire sans utiliser le théorème de Cook-Levin.
8. On peut prendre le mot "classe" dans cet énoncé comme un synonyme de "ensemble".