École polytechnique — X2010 — INF423 Fondements de l'informatique

Logique, modèles, calculs: PC notée

(corrigé)

L'énoncé comporte 4 parties indépendantes, qui pourront être traitées dans un ordre quelconque. En revanche, dans chaque partie, il peut être utile, dans la réponse à une question, d'utiliser les questions précédentes !

1 Machines rationnelles

Une machine rationnelle est définie comme une liste finie de fractions irréductibles $r_1, \ldots, r_n \in \mathbb{Q}$. Par exemple, la liste $(2/3, 7/5, 3/7)$ définit une machine M_1. Une machine rationnelle prend en entrée un entier naturel $k_0 \in \mathbb{N}$. Une configuration d'une machine rationnelle est simplement un entier $k \in \mathbb{N}$. Initialement, $k = k_0$. Une étape de calcul d'une machine rationnelle se déroule ainsi : la machine cherche dans la liste une fraction p/q telle que le dénominateur q divise k. S'il n'en existe pas, la machine s'arrête. Sinon, elle prend la première telle fraction apparaissant dans la liste, et remplace k par $k \times p/q$. L'entrée k_0 est acceptée par la machine si celle-ci s'arrête sur une valeur k divisible par 2.

Question 1.1. *Donner l'exécution de M_1 sur l'entrée 15.*

Solution : Les configurations successives sont : 15, $15 \times 2/3 = 10$, $10 \times 7/5 = 14$, $14 \times 3/7 = 6$, $6 \times 2/3 = 4$. □

Question 1.2. *Calculer l'ensemble des entiers naturels acceptés par M_1.*

Solution : Cet ensemble est l'ensemble des entiers divisibles par 2, 3, 5 ou 7. En effet, si p est un nombre premier différent de $3, 5, 7$ qui divise k_0, alors par récurrence sur le nombre d'étapes, on voit que p divise k à chaque étape, alors que les puissances de 7 sont remplacées par des puissances de 5, celles de 5 par des puissances de 3, et enfin celles de 3 par celles de 2. Autrement dit, si $k_0 = 2^a 3^b 5^c 7^d N$ où N n'est pas divisible par 2,3,5,7, alors la machine s'arrête avec la valeur $2^{a+b+c+d} N$ qui est paire si et seulement si $a + b + c + d \neq 0$. □

Question 1.3. [Cette question représente la moitié des points de cet exercice.] *Cette question vise à montrer que l'on peut simuler une machine de Turing par une machine rationnelle. Pour cela, on considère des machines à deux compteurs a et b. Initialement, $b = 0$ et $a \in \mathbb{N}$ est l'entrée. Une telle machine comporte un nombre fini L d'instructions. Pour $i \in \{1, \ldots, L\}$, l'instruction i est de l'une des formes*

1. $\mathsf{Incr}(c, j)$ *qui incrémente $c \in \{a, b\}$ puis va à l'instruction $j \neq i$;*
2. $\mathsf{Decr}(c, j)$ *qui décrémente $c \in \{a, b\}$ s'il n'est pas nul, puis va à l'instruction $j \neq i$;*
3. $\mathsf{IsZero}(c, j, k)$ *qui teste si $c \in \{a, b\}$ est nul, va à l'instruction $j \neq i$ si c'est le cas, et à l'instruction $k \neq i$ sinon ;*
4. Halt *qui arrête le calcul.*

Montrer comment coder une telle machine à deux compteurs dans une machine rationnelle.

Solution : Une solution consiste à coder les numéros des instructions par des nombres premiers. On numérote les nombres premiers à partir de 5 : $p_1 = 5, p_2 = 7, \ldots, p_L$. À tout moment la valeur stockée par la machine rationnelle est de la forme $2^a 3^b p_i$, où p_i va permettre d'aller à l'instruction i. Initialement, l'entier k_0 vaut $2^a p_1$. Le codage des transitions est alors réalisé par les ajouts suivants dans la liste de rationnels (le programme) :

1. $\mathsf{Incr}(c, j)$: $\phi(c)p_j/p_i$ (avec $\phi(a) = 2, \phi(b) = 3$) ;
2. $\mathsf{Decr}(c, j)$: $p_j/(\phi(c)p_i), p_j/p_i$ dans cet ordre, le premier permettant de tester si $c \neq 0$;
3. $\mathsf{IsZero}(c, j, k)$: $p'_k/(\phi(c)p_i), (\phi(c)p_k)/p'_k, p_j/p_i$, par la même idée que précédemment, où l'on utilise un premier supplémentaire p'_k au-delà de p_L pour réincrémenter c ;
4. Halt : $1/p_i$. Il ne reste plus alors que les contenus des deux compteurs sous la forme $2^a 3^b$.

□

Question 1.4. *Montrer que les machines rationnelles sont équivalentes aux machines de Turing.*

Solution : Il ne reste qu'à montrer que réciproquement, les machines de Turing peuvent simuler les machines rationnelles. On peut par exemple choisir de coder les entiers en base 2, et les rationnels comme des paires d'entiers. La multiplication, le test d'appartenance à une liste, le test de divisibilité sont calculables par machine de Turing. Tester si un nombre est une puissance de 2 aussi : il suffit de tester si nombre est de la forme $10\cdots0$. Pour chaque élément p/q de la liste on construit donc une sous-machine qui teste si le mot d'entrée est divisible par q et va en état de refus s'il ne l'est pas, sinon divise le mot par q et le multiplie par p et sort en acceptant. Ces machines sont alors enchaînées en envoyant la transition vers l'état de refus de l'une vers l'état initial de la suivante, et l'état d'acceptation vers une routine centrale qui teste si la valeur du mot est une puissance de 2 et accepte alors, ou relance une passe dans la liste. □

2 Gestion de la mémoire dans les langages de programmation modernes

Nous considérerons dans cet exercice que le langage Java constitue un système de programmation au sens de Turing, c'est-à-dire qu'il permet de représenter toutes les fonctions récursives, que le problème de l'arrêt y est indécidable, etc.

En langage Java, si la création d'un objet en mémoire se fait par un appel explicite à **new** ClasseDeLObjet(...), sa destruction se fait de façon automatique, par un *garbage collector*. Dans cette question, nous tenterons de préciser certaines limites théoriques de cette technologie.

Nous considérerons qu'à un instant de l'exécution d'un programme, un objet est devenu inutile s'il n'existe aucune prolongation possible de cette exécution où le programme se sert de cet objet. Par exemple, dans le programme suivant, l'objet de type **int[]** créé à la ligne 3 est inutile dès la fin de la ligne 4, vu que l'instruction `t[3] = 0` de la ligne 7 ne peut être exécutée en raison de conditions contradictoires sur `i` :

```
1  class Exemple {
2    void toto(int i) {
3      int[] t = new int[30];
4      t[1] = 4;
5      if (i < 10) {
6        if (i > 20) {
7          t[3] = 0;
8        }
9      }
10   }
11 }
```

Question 2.1. *Dans le programme suivant, le tableau pointé par* t *est-il utile à la fin de la ligne 3 ?*

```
1 class Boucle {
2     void toto() {
3         int[] t = new int[10];
4         int x = 0;
5         while(x == 0) {
6         }
7         t[0] = 1;
8     }
9 }
```

Solution : Non, car l'affectation `t[0] = 1;` n'est jamais exécutée, la boucle étant infinie. □

Considérons maintenant le programme suivant, où **void** Inconnu.f(**int** i) est une fonction définie ailleurs :

```
1 classe Exemple2 {
2   void toto2(int i) {
3     int[] t = new int[30];
4     Inconnu.f(i);
5     t[0] = 3;
6   }
7 }
```

Question 2.2. *L'objet tableau pointé par* t *peut-il être inutile dès la fin de la ligne 3 ? Si oui, à quelle condition nécessaire et suffisante ?*

Solution : Il est inutile si et seulement si Inconnu.f(i) ne termine pas. □

Question 2.3. *Est-il possible de faire un* garbage-collector *idéal, c'est-à-dire qui libère tous les objets inutiles et seulement ceux-ci ? Justifiez rapidement.*

Solution : Non, car l'exemple précédent montre qu'il faudrait pour cela résoudre le problème de l'arrêt. □

En raison de ce qui précède, les vrais *garbage collectors* n'essayent pas d'éliminer tous les objets inutiles, mais seulement ceux «prouvablement inutiles» car inaccessibles depuis les variables du programme. Un objet est réputé accessible s'il est pointé par une variable du programme, ou par un champ d'objet accessible. Un objet est donc inaccessible s'il n'existe aucune chaîne de «pointeurs» depuis les variables du programme qui aboutit à cet objet.

Ainsi, dans le programme de la classe Exemple, l'objet pointé par t est inutile dès la fin de la ligne 4, mais n'est pas «prouvablement inutile» car la variable t continue de pointer dessus. En revanche, dans le programme suivant, l'objet pointé par t devient prouvablement inutile dès la sortie de la fonction f car la variable t cesse alors d'exister.

```
class Exemple3 {
  void f() {
    int[] t = new int[30];
  }
}
```

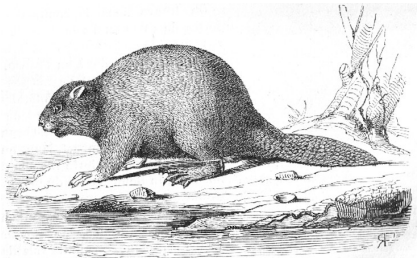
Question 2.4. *On prétend parfois que dans les langages comme Java ou Caml, il ne peut y avoir de fuite de mémoire, c'est-à-dire d'accumulation indéfinie de données inutiles en mémoire, bug assez courant pour les programmes écrits en C ou C++, où le programme doit explicitement désallouer sa mémoire. Montrez que c'est faux, au besoin en décrivant succinctement un exemple.*

Solution : Il suffit d'un programme qui stocke des références sur des objets dans un tableau et n'utilise jamais plus ces objets. Tant qu'une variable pointera sur ce tableau, la mémoire ne sera pas libérée. □

3 Le castor affairé

La fonction du « castor affairé » (*busy beaver*) a été introduite en 1962 par Tibor Radó.

On considère les machines de Turing déterministes dans les notations du poly, avec ensemble d'états $Q = \{q_1, \ldots, q_n\} \cup \{q_0, q_a, q_r\}$, un ruban infini dans les deux directions, l'alphabet de travail ne contenant que les symboles $0, 1$. Chaque machine est donc déterminée par une fonction de transition qui à chaque état de $\{q_0, q_1, \ldots, q_n\}$, et chaque valeur lue dans $\{0, 1\}$, associe une direction dans $\{\leftarrow, \rightarrow\}$, une valeur à écrire dans $\{0, 1\}$, et un état dans Q.



Le castor, par Alcide Raillet, 1895

Question 3.1. *Donnez le nombre $N(n)$ de machines de Turing du type ci-dessus. (Deux machines identiques à renommage d'états près sont considérées distinctes.)*

Solution : À chaque couple (état, valeur lue), dans $\{q_0, \ldots, q_n\} \times \{0, 1\}$ (ensemble à $2(n+1)$ éléments), on associe un triplet (nouvel état, direction, valeur écrite) dans $Q \times \{\leftarrow, \rightarrow\} \times \{0, 1\}$, ensemble à $4(n+3)$ éléments. On a donc

$$N(n) = (4(n+3))^{2n+2}\,.$$

□

Parmi ces machines, on nomme « castors à n états » les machines qui, sur une entrée constituée d'un ruban ne contenant que des 0, terminent dans l'état q_a d'acceptation. Le score d'un castor est le nombre de 1 écrits sur le ruban lorsque la machine s'arrête.

Question 3.2. *Montrez que l'ensemble des castors à n états est non vide.*

Solution : Prendre une machine triviale qui saute immédiatement à l'état d'acceptation. □

Question 3.3. *Montrer qu'il existe une fonction $\Sigma : \mathbb{N} \to \mathbb{N}$ tel que $\Sigma(n)$ est le score maximal d'un castor à n états.*

Solution : L'ensemble des castors à n états est non vide et fini. La fonction score est donc bornée et son maximum est atteint. □

On appelle Σ la *fonction du castor affairé*. On peut calculer la valeur de Σ pour des petites valeurs de n. Par exemple, $\Sigma(0) = 1, \Sigma(1) = 4, \Sigma(2) = 6, \Sigma(3) = 13$. Au-delà, les valeurs ne sont pas connues, le calcul devenant très rapidement extrêmement difficile. L'objectif de cet exercice est de montrer que cette fonction n'est pas calculable et d'en tirer quelques conséquences.

Question 3.4. *Montrer que $\Sigma(n) > n$.*

Solution : On construit explicitement un castor à n états de score $n+1$, avec comme transitions $(q_i, 0) \mapsto (\rightarrow, 1, q_{i+1})$ pour $0 \leq i < n$ et $(q_n, 0) \mapsto (\leftarrow, 1, q_a)$. □

Question 3.5. *Soit $f : \mathbb{N} \to \mathbb{N}$ une fonction calculable arbitraire. Montrer qu'il existe un entier N tel que pour tout $n \geq N$, $\Sigma(n) > f(n)$. (En particulier Σ n'est pas une fonction calculable). Pour cela :*

- *montrer que sans perte de généralité on peut considérer que f est croissante ;*
- *à partir de machines de Turing calculant $x \mapsto f(x)$, $x \mapsto x+1$ et $x \mapsto 2x$, construire un castor de score $f(2n)+1$ et en déduire une borne sur $f(2n)$ en fonction de $\Sigma(n+C)$ pour un C à préciser ;*
- *utiliser la croissance de f pour conclure.*

Solution : Si f n'est pas croissante, on peut considérer la fonction calculable g définie par $g(0) = f(0)$ et $g(n+1) = \max(f(n+1), g(n))$ qui est croissante. L'obtention du résultat sur g l'entraine alors sur f.

Soient a, b, c les nombres d'états des trois machines indiquées. On considère alors la machine M obtenue en composant dans cet ordre : le castor à $n-1$ états de la question 3.4 qui calcule n ; la machine qui calcule $x \mapsto 2x$; la machine qui calcule f ; la machine qui calcule $x \mapsto x+1$. Cette machine est un castor avec au plus $n+a+b+c$ états et de score $f(2n)+1$. Il s'ensuit que $f(2n) < \Sigma(n+a+b+c)$.

Pour $n \geq a+b+c$, on a $n+a+b+c \leq 2n$ et donc par croissance de f

$$f(n+a+b+c) \leq f(2n) < \Sigma(n+a+b+c)$$

pour n assez grand, comme souhaité. □

Question 3.6. *Soit $S(n)$ le nombre maximum de transitions d'un castor à n états. Montrer que la fonction $n \mapsto S(n)$ est bien définie, mais n'est pas calculable. Observer qu'on obtient ainsi une preuve alternative de l'indécidabilité du problème de l'arrêt.*

Solution : L'existence de la fonction découle du même argument que pour Σ. Par ailleurs, le nombre de transitions est au moins égal au score, puisque l'écriture de chaque 1 nécessite une transition. On a donc $S(n) \geq \Sigma(n)$ pour tout n et la question précédente montre alors que S n'est pas calculable.

Si le problème de l'arrêt était décidable, la fonction S serait calculable : il suffirait de boucler sur toutes les machines à n étapes, tester si elles s'arrêtent et si oui les simuler en comptant leur nombre de transitions, et en gardant le maximum. On a donc bien une preuve assez simple (sans codage de machines de Turing) de l'indécidabilité du problème de l'arrêt. □

4 Le corps des réels

On s'intéresse ici à la structure $\mathbb{R} = \langle \mathbb{R}, +, \cdot, <, 0, 1 \rangle$, le corps ordonné des réels. La construction habituelle de $\mathbb{R}$ réels passe par un certain nombre d'axiomes qui caractérisent $\mathbb{R}$ à isomorphisme près. L'un des axiomes de la construction indique le caractère complet pour la topologie et peut s'exprimer ainsi (les ensembles bornés supérieurement ont un supremum) :

$$\forall X \subset \mathbb{R}(X \neq \emptyset \wedge (\exists B \in \mathbb{R}, \forall x \in X, x \leq B) \Rightarrow \exists T \in \mathbb{R}((\exists b \in \mathbb{R}, \forall x \in X, x \leq b) \Rightarrow T \leq b)).$$

Question 4.1. *Cette formule est-elle du premier ordre ? Justifier.*

Solution : Elle ne l'est pas puisque la quantification porte sur un sous-ensemble et non pas un élément. □

L'objectif de cet exercice est de montrer le résultat suivant.

Théorème 1. *Il n'existe pas d'ensemble d'axiomes du premier ordre caractérisant $\mathbb{R}$ à isomorphisme près.*

Pour prouver ce résultat, on fera appel au théorème de Löwenheim-Skolem : *Si $\mathcal{T}$ une théorie sur une signature dénombrable possède un modèle, alors elle possède un modèle dont l'ensemble de base est dénombrable.*

Question 4.2. *Montrer le théorème de Löwenheim-Skolem en réutilisant la preuve du théorème de complétude.*

Solution : La preuve du théorème de complétude fonctionne en deux temps. Est d'abord traitée le cas d'une théorie avec témoins de Henkin. La signature étant dénombrable, alors le domaine considéré, à savoir l'ensemble des termes clos sur la signature, est également dénombrable. Si la théorie n'admet pas de témoins de Henkin, elle est d'abord étendue en une théorie $\mathcal{T}'$ qui en admet. Cette extension n'ajoute qu'une quantité dénombrable de nouvelles constantes, ce qui permet de conclure. □

Question 4.3. *Prouver le théorème 1 à l'aide du théorème de Löwenheim-Skolem.*

Solution : La signature est bien dénombrable, il existe donc un modèle dénombrable, qui ne peut donc être isomorphe à $\mathbb{R}$. □

Même avec une signature non-dénombrable, le résultat du théorème 1 persiste : on considère maintenant la structure $\langle \mathbb{R}, +, \cdot, <, r \rangle_{r \in \mathbb{R}}$ avec un symbole de constante r différent par nombre réel. Une des propriétés de $\mathbb{R}$ est son caractère *archimédien* :

$$\forall x \exists n, x \leq n,$$

où n est le terme $((1+1)+\cdots+1)$ n fois.

Question 4.4. *Justifier pourquoi cette formule n'est pas du premier ordre.*

Solution : Le quantificateur existentiel porte sur une variable n qui est contrainte à être un entier et non un élément quelconque du domaine $\mathbb{R}$. □

On introduit un nouveau symbole de constante c. Nous prendrons comme axiomes l'ensemble des formules du premier ordre vraies sur $\mathbb{R}$, et les formules $c > r$ pour tous les réels r.

Question 4.5. *Montrer que la théorie ainsi formée possède un modèle ${}^\star\mathbb{R}$.*

Solution : Il s'agit d'une application du théorème de compacité. Toute partie finie de cette théorie ne fait intervenir qu'un nombre fini de symboles de constantes r. Il suffit alors d'interpréter c comme 1 plus le plus grand de ces réels, et $\mathbb{R}$ fournit un modèle de cette partie. □

Question 4.6. *Conclure que ${}^\star\mathbb{R}$ est un modèle non-archimédien des formules du premier ordre vraies sur $\mathbb{R}$ qui n'est donc pas isomorphe à $\mathbb{R}$.*

Solution : Le raisonnement est le même que pour les entiers non-standards. Le modèle est bien un modèle des axiomes de corps ordonné, et il contient un élément qui est plus grand que tous les réels, donc que tout n. Il n'est donc pas archimédien et ne peut donc pas être isomorphe à $\mathbb{R}$. □