INF110
Contrôle de connaissances

Logique et Fondements de l'Informatique

26 janvier 2024

Consignes.

Les exercices et le problème sont totalement indépendants. Ils pourront être traités dans un ordre quelconque, mais on demande de faire apparaître de façon très visible dans les copies où commence chaque exercice (tirez au moins un trait sur toute la largeur de la feuille entre deux exercices).

Les questions du problème dépendent les unes des autres, mais ont été rédigées de manière à ce que chacune donne toutes les informations nécessaires pour passer à la suite. Mais comme elles (les questions du problème) présentent une gradation approximative de difficulté, il est recommandé de les traiter dans l'ordre.

La longueur du sujet ne doit pas effrayer : l'énoncé du problème est très long parce que des rappels ont été faits et que les questions ont été rédigées de façon aussi précise que possible. Par ailleurs, il ne sera pas nécessaire de tout traiter pour avoir le maximum des points.

L'usage de tous les documents écrits (notes de cours manuscrites ou imprimées, feuilles d'exercices, livres) est autorisé.

L'usage des appareils électroniques est interdit.

Durée : 3h

Barème *approximatif* et *indicatif* (sur 20) : entre 2 et 3 points par exercice et environ 0.75 points par question du problème.

Cet énoncé comporte 7 pages (page de garde incluse).

Git : 334c718 Wed Jan 24 23:11:29 2024 +0100

Exercice 1.

On considère un extrait d'une démonstration interactive en Coq du théorème `forall (A:Prop) (B:Prop) (C:Prop), (A->B)->(B->C)->(A->C)` où on a recopié ci-dessous seulement la sortie de Coq. On demande de retrouver les deux commandes qui ont été utilisées, et de proposer l'étape suivante.

```
1 subgoal

A, B, C : Prop
============================
(A -> B) -> (B -> C) -> A -> C
```

`thm <` (commande à trouver)

```
1 subgoal

A, B, C : Prop
x : A -> B
y : B -> C
z : A
============================
C
```

`thm <` (commande à trouver)

```
1 subgoal

A, B, C : Prop
x : A -> B
y : B -> C
z : A
============================
B
```

`thm <` (commande à proposer)

(À défaut de connaître le nom précis de la tactique, on pourra expliquer ce qu'elle fait de façon générale, ou à quelle règle de logique elle correspond.)

Exercice 2.

Donner une démonstration en calcul propositionnel intuitionniste de la formule suivante :

$$A \Rightarrow (A \Rightarrow B) \Rightarrow B$$

On donnera la démonstration sous la forme qu'on préfère (arbre de preuve ou style drapeau), puis on donnera aussi un λ-terme de preuve. On décrira ensuite brièvement ce que fait ce terme vu en tant que programme (i.e., le comportement du programme associé à la preuve par la correspondance de Curry-Howard).

Exercice 3.

(Dans cet exercice, on ne demande pas de justifier les réponses.)

(1) En calcul propositionnel intuitionniste, quel est l'énoncé prouvé par le λ-terme de preuve suivant ? (Ou si on préfère : quel est le type de ce terme du λ-calcul simplement typé enrichi de types produits, qu'on a écrit en notations logiques ?)

$$\lambda(p : C \wedge (A \Rightarrow B)).\, \lambda(h : A).\, \langle (\pi_1 p),\, (\pi_2 p) h \rangle$$

(Ici, A, B, C sont des variables propositionnelles.)

(2) En logique du premier ordre intuitionniste, quel est l'énoncé prouvé par le λ-terme de preuve suivant ?

$$\lambda(p : C \wedge \forall x. B(x)).\, \lambda(x : I).\, \langle (\pi_1 p),\, (\pi_2 p) x \rangle$$

(Ici, C est une variable de relation nullaire, c'est-à-dire une variable propositionnelle, B est une variable de relation unaire, et I est le symbole du type des individus.)

(3) En logique du premier ordre intuitionniste, quel est l'énoncé prouvé par le λ-terme de preuve suivant ?

$$\lambda(p : \exists x.(A \Rightarrow B(x))).\, \lambda(h : A).\, (\texttt{match}\ p\ \texttt{with}\ \langle x, j \rangle \mapsto \langle x, jh \rangle)$$

(Ici, A est une variable de relation nullaire, c'est-à-dire une variable propositionnelle, et B est une variable de relation unaire.)

Exercice 4.

En appliquant l'algorithme de Hindley-Milner, trouver une annotation de type (dans le λ-calcul simplement typé) au terme suivant du λ-calcul non typé :

$$\lambda f.\, \lambda z.\, f z (\lambda g. g z)$$

(autrement dit, `fun f -> fun z -> f z (fun g -> g z)` en syntaxe OCaml). Quel théorème du calcul propositionnel intuitionniste obtient-on de ceci par Curry-Howard ?

(Une réponse qui ne suit pas l'algorithme de Hindley-Milner mais qui est néanmoins correcte vaudra une partie des points.)

Exercice 5.

En utilisant l'une quelconque des sémantiques vues en cours pour le calcul propositionnel intuitionniste (au choix : toutes marcheront), montrer que la formule suivante n'est pas démontrable en calcul propositionnel intuitionniste :

$$\big(\neg(A \wedge B) \wedge \neg(B \wedge C) \wedge \neg(C \wedge A)\big) \;\Rightarrow\; \big(\neg A \vee \neg B \vee \neg C\big)$$

(*Indications.* Si on préfère la sémantique de Kripke, considérer un monde permettant d'accéder à trois autres mondes, tous les trois inaccessibles depuis les uns les autres. Si on préfère la sémantique des ouverts, considérer trois secteurs angulaires ouverts dans le plan, ayant un même sommet et qui se jouxtent sans s'intersecter. Si on préfère la sémantique de la réalisabilité ou des problèmes finis, il s'agit d'énoncer le fait qu'il est impossible de déterminer une partie vide parmi trois, sans avoir accès à ces parties, même si on a la promesse qu'au moins deux d'entre elles sont vides : pour la réalisabilité, on pourra considérer une des parties valant $\mathbb{N}$ et les deux autres valant $\varnothing$ et constater qu'on ne peut pas réaliser les trois affectations à la fois, tandis que pour la sémantique des problèmes finis, on pourra considérer trois problèmes ayant un seul candidat chacun mais un seul ayant une solution.)

Exercice 6.

(On ne demande pas ici de réponses compliquées !)

(1) Expliquer rapidement pourquoi, si $\mathscr{T}$ est un ensemble de formules logiques (par exemple de logique de premier ordre, mais peu importent les détails), on peut démontrer $P \Rightarrow Q$ à partir de $\mathscr{T}$ si et seulement si on peut démontrer Q à partir de $\mathscr{T} \cup \{P\}$.

(2) Déduire de (1) et du théorème de Gödel que la théorie PA^* formée en ajoutant l'axiome $\neg\mathsf{Consis}(\mathsf{PA})$ à l'arithmétique de Peano (PA), ne prouve pas $\bot$ (c'est-à-dire qu'elle n'est pas contradictoire). Ici, $\mathsf{Consis}(\mathsf{PA})$ est l'énoncé qui affirme que PA n'est pas contradictoire, et on rappelle que cet énoncé est démontrable dans les mathématiques usuelles (ZFC) où on travaille.

(On rappelle aussi que PA travaille en logique classique.)

(3) Expliquer pourquoi $\mathsf{Consis}(\mathsf{PA}^*)$ est plus fort que (i.e., implique) $\mathsf{Consis}(\mathsf{PA})$, et en déduire que PA^* démontre $\neg\mathsf{Consis}(\mathsf{PA}^*)$.

(4) On a vu en (2) que PA^* n'est pas contradictoire, et en (3) que PA^* démontre que PA^* est contradictoire. Ceci peut sembler paradoxal. Qu'en pensez-vous ?

Problème 7.

Motivations : On s'intéresse dans cet exercice à la notion de *réel calculable*, qu'on va définir : c'est une formalisation possible d'un nombre réel exact pouvant être manipulé par un ordinateur. L'idée la plus évidente pour définir les réels calculables est sans doute de considérer ceux dont une écriture binaire est une fonction calculable (et de les représenter par cette fonction) : on va voir plus loin que cette définition « naïve » souffre de graves problèmes. À la place, on va définir un réel calculable comme un réel dont on peut calculer par un programme une approximation rationnelle à ε près pour n'importe quel $\varepsilon > 0$ donné en entrée (et représenter le réel par une fonction qui prend ε et renvoie l'approximation). Pour rendre cette définition plus commode à manier, on va se limiter à ε de la forme $\frac{1}{2^n}$ et renvoyer une approximation elle-même de la forme $\frac{k}{2^n}$ (cela ne change rien d'essentiel).

On introduit donc les définitions suivantes :

— Un **dyadique** est un rationnel de la forme $\frac{k}{2^n}$ avec $k \in \mathbb{Z}$ et $n \in \mathbb{N}$ (i.e., son dénominateur est une puissance de 2). Pour être plus précis, on peut appeler **dyadique de niveau** n un rationnel de la forme $\frac{k}{2^n}$ avec $k \in \mathbb{Z}$ (pour ce n-là). Noter qu'on ne demande pas que $\frac{k}{2^n}$ soit irréductible, par exemple $42 = \frac{84}{2} = \frac{168}{2^2} = \cdots$ peut être vu comme un dyadique de n'importe quel niveau.

— Si $x \in \mathbb{R}$, un **numérateur d'approximation dyadique de niveau** n de x est un entier k tel que $|k - 2^n\, x| \leq 1$. L'approximation dyadique elle-même est alors par définition $\frac{k}{2^n}$, et vérifie donc $|\frac{k}{2^n} - x| \leq \frac{1}{2^n}$.

— Un réel $x \in \mathbb{R}$ est dit **calculable** lorsqu'il y a une fonction *calculable* (totale) $\mathbb{N} \to \mathbb{Z}, n \mapsto k_n$ telle que k_n soit un numérateur d'approximation dyadique de niveau n de x. Autrement dit, il existe un programme qui, prenant n en entrée, renvoie une approximation dyadique $\frac{k_n}{2^n}$ de niveau n de x (représentée par le seul numérateur, puisque le dénominateur est par définition 2^n). Une telle fonction $n \mapsto k_n$, ou un programme qui la calcule, est dit **représenter** le réel x.

À titre d'exemple, l'entier 42 est un réel calculable puisque la fonction $n \mapsto 42 \times 2^n$ est évidemment calculable.

On prendra garde aux faits suivants :

— Un même réel x a plusieurs approximations dyadiques de niveau n (il en a même toujours 2 ou 3, puisque leurs numérateurs sont les entiers contenus dans l'intervalle $[2^n\, x - 1\,;\, 2^n\, x + 1]$ de longueur 2). À titre d'exemple, le réel 0 peut être représenté par n'importe quelle fonction calculable $n \mapsto k_n$ renvoyant un k_n dans $\{-1, 0, 1\}$.

— Même une fois fixée la fonction mathématique $n \mapsto k_n$ représentant un réel calculable x, il y a (toujours) plusieurs programmes (= fonctions informatiques) qui la calculent (« intentions »). C'est par cette dernière donnée qu'on va représenter le réel x.

On ne demande pas de justifier le fait évident qu'un programme informatique peut manipuler des données dans $\mathbb{Z}$ (entiers relatifs arbitraires), notamment faire de l'arithmétique basique dessus (sommes, différences, produits, exponentiation, division euclidienne, etc.).

Lorsqu'il est demandé d'écrire un programme, on l'écrira dans un langage de programmation raisonnable quelconque (capable de manipuler des entiers arbitrairement grands), ou sous forme de pseudocode. Par exemple, s'il est demandé d'écrire un programme représentant l'entier 42 on pourra écrire « `lambda n: 42 * 2**n` » ou « `fun n -> 42 * (pow 2 n)` » ou n'importe quoi d'aisément compréhensible de la sorte, mais en précisant ce qui n'est pas évident (par exemple que « `pow` » est une fonction calculant l'exponentiation entière). On pourra aussi librement décider si ce langage manipule des fonctions calculables sous forme de codes de Gödel de programmes les calculant, ou sous forme de fonctions informatiques (langage fonctionnel), ou même mélanger les deux au besoin (à condition de le préciser).

*

(1) Expliquer pourquoi, donnés $p \in \mathbb{Z}$ et $q \in \mathbb{N}\setminus\{0\}$ et $n \in \mathbb{N}$, on peut calculer algorithmiquement un numérateur d'approximation dyadique de niveau n de $\frac{p}{q}$. En déduire que tout rationnel est un réel calculable, et, plus précisément, écrire un programme qui, donnés p et q, renvoie une fonction représentant le rationnel $\frac{p}{q}$ comme réel calculable.

(2) Montrer que si x est un réel calculable alors $-x$ est calculable, et, plus précisément, écrire un programme qui, donnée une fonction qui représente x, en renvoie une qui représente $-x$. Faire de même pour $|x|$. (On rappelle à toutes fins utiles que $\Big||u| - |v|\Big| \leq |u - v|$.)

(3) Expliquer pourquoi, si k est un numérateur d'approximation dyadique de niveau $n + r$ de $x \in \mathbb{R}$ (avec $n, r \in \mathbb{N}$), alors k est aussi un numérateur d'approximation dyadique de niveau n de $2^r x$. En déduire que si x est un réel calculable et $r \in \mathbb{N}$ alors $2^r x$ est calculable, et, plus précisément, écrire un programme qui, donnés r et une fonction qui représente x, renvoie une fonction qui représente $2^r x$.

(4) Expliquer comment, donné $\ell \in \mathbb{Z}$, on peut trouver algorithmiquement un entier $k \in \mathbb{Z}$ tel que $[\frac{\ell}{4} - \frac{1}{2}\,;\, \frac{\ell}{4} + \frac{1}{2}] \subseteq [k - 1\,;\, k + 1]$ (on pourra faire un dessin de ces intervalles). En tirer une façon de trouver algorithmiquement un numérateur d'approximation dyadique de niveau n de $x + y$ à partir de numérateurs d'approximations dyadiques de niveau $n + 2$ de x et y respectivement.

En déduire que si x et y sont calculables alors $x + y$ est calculable, et, plus précisément, écrire un programme qui, données des fonctions qui représentent x et y, en renvoie une qui représente $x + y$.

(5) Montrer qu'un réel calculable z, représenté par une fonction $n \mapsto k_n$, vérifie $z \leq 0$ si et seulement on a $k_n \leq 1$ pour tout n. En déduire comment, donnée une fonction $n \mapsto k_n$ représentant z calculable, et en *supposant* $z > 0$, on peut calculer algorithmiquement un $n \in \mathbb{N}$ tel que $z \geq \frac{1}{2^n}$.

De façon analogue, montrer que, donnée une fonction $n \mapsto k_n$ représentant z calculable, et en supposant seulement $z \neq 0$, on peut décider algorithmiquement si $z < 0$ ou $z > 0$.

(6) (Cette question est indépendante de toutes les questions qui précèdent ou qui suivent.) Montrer qu'il n'existe pas de programme qui, donné le code d'un autre programme e, décide si ce dernier représente un réel calculable. Autrement dit, la fonction qui à e associe 1 si e calcule une fonction $n \mapsto k_n$ représentant un certain réel x, et 0 sinon, n'est pas calculable.

(7) Soit e un programme (vu comme l'indice d'une fonction générale récursive φ_e). Soit $\eta(e)$ le réel défini de la manière suivante :

$$\eta(e) = \begin{cases} \frac{1}{2^m} & \text{si le calcul de } \varphi_e(0) \text{ termine en exactement } m \text{ étapes} \\ 0 & \text{si } \varphi_e(0)\uparrow \end{cases}$$

(Plus exactement, il faudrait plutôt écrire « $\eta(e) = \frac{1}{2^m}$ si m est le code d'un arbre de calcul pour $\varphi_e(0)$ » à la première ligne, mais on peut se permettre de confondre ces deux notions.)

Expliquer comment, donnés e et $n \in \mathbb{N}$, calculer algorithmiquement un numérateur d'approximation dyadique de niveau n de $\eta(e)$. En déduire qu'on peut écrire un programme qui, donné e, renvoie une fonction représentant le réel $\eta(e)$ (en tant réel calculable).

(8) Expliquer l'erreur dans la réponse suivante à la question (7) : « On lance le calcul de $\varphi_e(0)$: s'il termine en (exactement) m étapes, on renvoie (une fonction représentant) $\frac{1}{2^m}$, qui est calculable d'après la question (1), sinon on renvoie la fonction constamment nulle. » On expliquera aussi en quoi on n'a pas commis cette erreur en répondant à ladite question.

(9) Déduire de la question (8) qu'il n'y a pas d'algorithme qui, donnée une fonction représentant un réel calculable x, teste si $x \neq 0$ ou $x = 0$ (i.e., renvoie « vrai » si $x \neq 0$ et « faux » si $x = 0$).

Pour la question suivante, on *admettra* le résultat suivant (plus ou moins vu en TD) : il n'existe pas d'algorithme qui, donné le code d'un programme e, termine toujours en temps fini et renvoie « vrai » si $\varphi_e(0)\downarrow = 0$ et « faux » si $\varphi_e(0)\downarrow \neq 0$ (si $\varphi_e(0)\uparrow$ il aurait le droit de répondre n'importe quoi mais toujours avec l'obligation de terminer).

(10) En modifiant un peu la construction proposée en (7) (on définira $\eta'(e)$ valant $\frac{1}{2^m}$ ou $-\frac{1}{2^m}$ ou 0 selon des cas judicieusement choisis), montrer qu'il n'est même pas possible algorithmiquement de tester si un réel calculable est ≥ 0 ou ≤ 0, i.e., il n'existe pas d'algorithme qui, donnée une fonction représentant un réel calculable x, peut renvoyer « vrai » si $x \geq 0$ ou « faux » si $x \leq 0$ (les deux réponses étant acceptables si $x = 0$).

Si x est un réel entre 0 et 1 (pour s'éviter le tracas de l'écriture des nombres négatifs), on rappelle qu'une **écriture binaire** de x est une suite $(u_n)_{n\geq 1}$ à valeurs dans $\{0, 1\}$ telle que $x = \sum_{n=1}^{+\infty} u_n \cdot 2^{-n}$. Une telle suite existe toujours (par exemple, on peut prendre celle donnée $u_n = \lfloor 2^n x \rfloor - 2\lfloor 2^{n-1} x \rfloor$ où $\lfloor — \rfloor$ désigne la partie entière), mais elle n'est pas unique en général (par exemple $\frac{1}{2}$ admet les deux écritures binaires $0.100000\ldots$ et $0.011111\ldots$; en fait, ce sont exactement les dyadiques qui admettent plus d'une écriture binaire, et ils en admettent exactement deux).

On dira qu'un réel entre 0 et 1 est **naïvement calculable** lorsqu'il admet une écriture binaire $n \mapsto u_n$ calculable. On pourra aussi dire que cette fonction, ou un programme qui la calcule, le **représente naïvement**. Le but des questions suivantes est de comprendre pourquoi cette notion ne se comporte pas bien (et donc pourquoi on ne l'a pas prise comme définition principale).

(11) Montrer qu'un réel (entre 0 et 1) naïvement calculable est calculable, et plus précisément qu'il existe un algorithme qui, donnée une fonction $n \mapsto u_n$ qui calcule l'écriture binaire de x, en renvoie une représentation $n \mapsto k_n$ comme réel calculable (ainsi que définie au début de ce problème).

(12) Dans l'autre sens, en utilisant la question (10), montrer qu'il n'existe pas d'algorithme qui, donnée une fonction $n \mapsto k_n$ représentant un réel x calculable entre 0 et 1, renvoie une fonction calculable $n \mapsto u_n$ qui soit une écriture binaire de x. (*Indication :* considérer $\frac{1}{2}(\eta'(e) + 1)$.)

(13) En utilisant de nouveau la question (10), montrer qu'il n'existe pas d'algorithme qui, donnés deux réels calculables naïfs x, y (avec, disons, x entre $\frac{1}{2}$ et 1 et y entre 0 et $\frac{1}{2}$) sous forme d'une fonction calculant une écriture binaire, renvoie $x - y$ sous cette même forme.

(14) Montrer que, la question (12) nonobstant, tout réel calculable (entre 0 et 1) est naïvement calculable. (On pourra distinguer deux cas : soit le réel est dyadique, soit il ne l'est pas.) On expliquera pourquoi ce n'est pas une contradiction qu'on ait montré que les réels calculables et naïvement calculables coïncident, mais qu'on peut algorithmiquement soustraire les réels calculables mais pas les réels naïvement calculables.