INF110 / CSC-3TC34-TP
Contrôle de connaissances

Logique et Fondements de l'Informatique

29 janvier 2025

Consignes.

Les exercices et le problème sont totalement indépendants les uns des autres. 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 et éclaircissements 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.

Dans les exercices portant sur Coq (exercices 1 à 4), les erreurs de syntaxe Coq ne seront pas pénalisées tant qu'on comprend clairement l'intention.

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

Cet énoncé comporte 8 pages (page de garde incluse).

Git : 18f8b91 Thu Jan 23 22:39:25 2025 +0100

Exercice 1.

Dans cet exercice, on considère des paires d'états d'une preuve en Coq avant et après l'application d'une tactique. On demande de retrouver quelle est la tactique ou la séquence de tactiques appliquée.

(1) On part de l'état suivant :

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

et on veut arriver à l'état suivant :

```
First subgoal:

  A, B, C : Prop
  ============================
  (A /\ B) /\ C -> A /\ B /\ C

Second subgoal:

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

(2) On part de l'état suivant :

```
  A, B, C : Prop
  H : (A /\ B) /\ C
  ============================
  A
```

et on veut arriver à l'état suivant :

```
  H1 : A
  H2 : B
  H3 : C
  ============================
  A
```

(3) On part de l'état suivant :

```
  A, B, C : Prop
  H : A
  ============================
  A \/ B
```

et on veut arriver à l'état suivant :

```
  No more goals.
```

(4) On part de l'état suivant :

```
  ============================
  (A -> B) -> ~ B -> ~ A
```

et on veut arriver à l'état suivant :

```
  H1 : A -> B
  H2 : ~ B
  H3 : A
  ============================
  False
```

(5) On part de l'état suivant :

```
  n, m : nat
  ============================
  n + m = m + n
```

et on veut arriver à l'état suivant :

```
First subgoal:

  m : nat
  ============================
  0 + m = m + 0

Second subgoal:

  n, m : nat
  IHn : n + m = m + n
  ============================
  S n + m = m + S n
```

(6) On part de l'état suivant :

```
  n, m : nat
  ============================
  S (n + m) = S (m + n)
```

et on veut arriver à l'état suivant :

```
  n, m : nat
  ============================
  n + m = m + n
```

(7) On part de l'état suivant :

```
n, m : nat
H : S n = S m
============================
n + 1 = m + 1
```

et on veut arriver à l'état suivant :

```
n, m : nat
H : n = m
============================
n + 1 = m + 1
```

Exercice 2.

Si l'on dispose du lemme suivant en Coq :

```
Lemma add_0_r : forall n : nat, n + 0 = n.
```

Parmi les buts suivants, quand peut-on utiliser ce lemme avec la tactique `rewrite` ? Quand peut-on utiliser ce lemme avec la tactique `apply` ? Justifier brièvement.

(1)

```
n : nat
============================
n + 0 = 0 + n
```

(2)

```
n, m : nat
============================
(n + m) + 0 = m + n
```

(3)

```
n, m : nat
============================
(m * n) + 0 = m * n
```

(4)

```
n, m : nat
============================
(n + 0) + m = n + m
```

Exercice 3.

(A) Dans cet exercice, on donne un terme de preuve Coq et on demande de retrouver le théorème du calcul propositionnel intuitionniste qu'il prouve.

(1) `fun (H : A) => H`

(2) `fun (H1 : A -> B) (H2 : B -> C) (H3 : A) => H2 (H1 H3)`

(3) `fun (H1 : A -> B) (H2 : ~ B) => (fun (H3 : A) => H2 (H1 H3))`

(B) Écrire la démonstration décrite par le terme (3) en déduction naturelle (on donnera l'arbre de preuve ou la présentation en style drapeau, comme on préfère).

Exercice 4.

(1) Définir en Coq un type pour représenter les jours de la semaine (7 valeurs possibles).

(2) Définir une fonction qui, étant donné un jour de la semaine, renvoie le jour suivant.

(3) Énoncer un lemme en Coq affirmant que le jour suivant du dimanche est le lundi.

(4) Avec quelle(s) tactique(s) peut-on prouver ce lemme ?

(5) Énoncer un lemme en Coq affirmant que le jour suivant du jour j n'est pas j.

(6) Expliquer dans les grandes lignes comment on pourrait prouver ce lemme en Coq (notamment quelles seraient les principales tactiques utilisées).

Exercice 5.

En utilisant une des sémantiques vues en cours pour le calcul propositionnel intuitionniste, montrer que la formule suivante n'est pas démontrable en calcul propositionnel intuitionniste :

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

(*Indications :* Si on souhaite utiliser la sémantique des ouverts, on pourra utiliser deux demi-droites de même origine dans la droite réelle. Si on souhaite utiliser la sémantique de Kripke, on pourra prendre le cadre $\{u, v, w\}$ avec l'ordre engendré par les seules relations $u \leq v$ et $u \leq w$, et considérer les affectations de vérité valant 1 uniquement dans le monde v respectivement w. Si on souhaite utiliser la sémantique de la réalisabilité propositionnelle, on remarquera qu'un programme e qui réalise cette formule pour $A = \mathbb{N}$ et $B = \varnothing$ doit renvoyer un élément de la forme $\langle 0, — \rangle$ si on l'applique à $\langle c, c \rangle$ avec c le code de Gödel d'une fonction constante quelconque, et que pour $A = \varnothing$ et $B = \mathbb{N}$ il doit renvoyer $\langle 1, — \rangle$.)

La formule en question est-elle démontrable en calcul propositionnel classique ? (On ne demande pas forcément d'en exhiber une démonstration.)

Problème 6.

Notations : Dans tout cet exercice, on notera comme d'habitude $\varphi_e \colon \mathbb{N} \dashrightarrow \mathbb{N}$ (pour $e \in \mathbb{N}$) la e-ième fonction partielle calculable (i.e. générale récursive). On notera

$$\mathsf{PartCalc} := \{\varphi_e \ : \ e \in \mathbb{N}\} = \{g \colon \mathbb{N} \dashrightarrow \mathbb{N} \ : \ \exists e \in \mathbb{N}.\, (g = \varphi_e)\}$$

l'ensemble des fonctions *partielles* calculables $\mathbb{N} \dashrightarrow \mathbb{N}$, ainsi que

$$\mathsf{TotCode} := \{e \in \mathbb{N} \ : \ \varphi_e \text{ est totale}\} = \{e \in \mathbb{N} \ : \ \varphi_e \in \mathsf{TotCalc}\}$$

l'ensemble des codes des programmes qui définissent une fonction *totale* $\mathbb{N} \to \mathbb{N}$ (i.e., terminent et renvoient un entier quel que soit l'entier qu'on leur fournit en entrée) [1], et

$$\mathsf{TotCalc} := \{\varphi_e \ : \ e \in \mathsf{TotCode}\} = \{g \colon \mathbb{N} \to \mathbb{N} \ : \ \exists e \in \mathbb{N}.\, (g = \varphi_e)\}$$

l'ensemble correspondant des fonctions *totales* calculables $\mathbb{N} \to \mathbb{N}$, i.e., celles calculées par les programmes qu'on vient de dire.

On va s'intéresser à la notion (qu'on va définir) de fonction calculable $\mathsf{PartCalc} \to \mathbb{N}$ d'une part, et $\mathsf{TotCalc} \to \mathbb{N}$ d'autre part. (On parle parfois de « fonctionnelles » ou de « fonction de second ordre » pour ces fonctions sur les fonctions. On souligne qu'on parle bien ici de fonction *totales* $\mathsf{PartCalc} \to \mathbb{N}$ et $\mathsf{TotCalc} \to \mathbb{N}$.)

Définition : (A) Une fonction (totale !) $F \colon \mathsf{PartCalc} \to \mathbb{N}$ est dite *calculable* lorsque la fonction $\hat{F} \colon \mathbb{N} \to \mathbb{N}$ définie par $\hat{F}(e) = F(\varphi_e)$ est calculable. (B) De même, une fonction (totale) $F \colon \mathsf{TotCalc} \to \mathbb{N}$ est dite calculable lorsqu'il existe une fonction $\hat{F} \colon \mathbb{N} \dashrightarrow \mathbb{N}$ partielle calculable telle que telle que $\hat{F}(e) = F(\varphi_e)$ pour tout $e \in \mathsf{TotCode}$.

En français : une fonctionnelle définie sur l'ensemble des fonctions calculables partielles ou totales est elle-même dite calculable lorsqu'il existe un programme qui calcule $F(g)$ à partir du code e d'un programme quelconque calculant g.

Dans le cas (A), la définition implique que la fonction $\hat{F}$ vérifie forcément $(\varphi_{e_1} = \varphi_{e_2}) \implies (\hat{F}(e_1) = \hat{F}(e_2))$; c'est-à-dire qu'elle est « extensionnelle » : elle doit renvoyer la même valeur sur deux programmes qui calculent la même fonction (= ont la même « extension »). D'ailleurs (on ne demande pas de justifier ce fait), se donner une fonction $F \colon \mathsf{PartCalc} \to \mathbb{N}$ revient exactement à se donner une fonction $\hat{F} \colon \mathbb{N} \to \mathbb{N}$ ayant cette propriété d'« extensionnalité ». (De même, dans le cas (B), se donner une fonction $F \colon \mathsf{TotCalc} \to \mathbb{N}$ revient exactement à se donner une fonction qui soit extensionnelle sur $\mathsf{TotCode}$.) La définition ci-dessus dit donc que F est calculable lorsque la $\hat{F}$ qui lui correspond est elle-même calculable dans le cas (A), ou, dans le cas (B) la restriction à $\mathsf{TotCode}$ d'une fonction calculable partielle sur $\mathbb{N}$ dont le domaine de définition contient au moins $\mathsf{TotCode}$.

(1) Montrer que toute fonction (totale !) calculable $F \colon \mathsf{PartCalc} \to \mathbb{N}$ est en fait constante.

Pour cela, on pourra supposer par l'absurde que F prend deux valeurs distinctes, disons v_1 et v_2, et considérer l'ensemble des e tels que $F(\varphi_e) = v_1$ (i.e., tels que $\hat{F}(e) = v_1$). (Pourquoi est-il décidable ? Et pourquoi est-ce une contradiction ?)

(2) Expliquer pourquoi il existe des fonctions calculables (totales) $F \colon \mathsf{TotCalc} \to \mathbb{N}$ qui ne sont pas constantes. Donner un exemple explicite.

Pour cela, on pourra penser évaluer en un point, c'est-à-dire exécuter, le programme dont on a reçu le code en argument (on rappellera pourquoi on peut faire cela). Par ailleurs, on fera clairement ressortir pourquoi le raisonnement tenu ici ne s'applique pas à la question (1).

1. Par souci de cohérence, on pourrait aussi vouloir définir $\mathsf{PartCode} = \mathbb{N}$ comme l'ensemble des codes des programmes définissant une fonction partielle : nous suivons la convention que toute erreur dans un programme (même « syntaxique ») conduit à une valeur non-définie.

Fixons maintenant une fonction calculable $F\colon \mathsf{TotCalc} \to \mathbb{N}$, ainsi que la fonction $\hat{F}\colon \mathbb{N} \dashrightarrow \mathbb{N}$ qui lui correspond d'après le (B) des définitions ci-dessus.

Dans les questions (3) à (8) qui suivent, on cherche à démontrer que F a la propriété [2] suivante (« théorème de Kreisel-Lacombe-Shoenfield ») : quelle que soit $g \in \mathsf{TotCalc}$, il existe n tel que pour toute fonction $g' \in \mathsf{TotCalc}$ qui coïncide avec g jusqu'au rang n (i.e. : $\forall i \leq n.\, (g'(i) = g(i))$) on ait $F(g') = F(g)$.

Notations : Soit $\mathsf{NulAPCR}$ l'ensemble des fonctions $h\colon \mathbb{N} \to \mathbb{N}$ qui sont nulles à partir d'un certain rang, c'est-à-dire telles que $\exists m. \forall i \geq m.\, (h(i) = 0)$.

(3) (a) Expliquer pourquoi $\mathsf{NulAPCR} \subseteq \mathsf{TotCalc}$, i.e., pourquoi toute fonction $\mathbb{N} \to \mathbb{N}$ nulle à partir d'un certain rang est automatiquement calculable. **(b)** Montrer, plus précisément, qu'il existe une fonction calculable $\Gamma\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ telle que les fonctions $h \in \mathsf{NulAPCR}$ soient exactement celles de la forme $\Gamma(j, —)$ (c'est-à-dire $i \mapsto \Gamma(j, i)$) pour un certain j. (*Indication :* on pourra utiliser un codage de Gödel des suites finies d'entiers naturels par des entiers naturels j ; on ne demande pas de justifier les détails de ce codage.) **(c)** En déduire qu'il existe $\gamma\colon \mathbb{N} \to \mathbb{N}$ calculable telle que les fonctions de $\mathsf{NulAPCR}$ soient exactement celles de la forme $\varphi_{\gamma(j)}$ pour un certain j. (*Indication :* on pourra utiliser le théorème s-m-n.)

Notations : Si $g \in \mathsf{TotCalc}$ et $n \in \mathbb{N}$, on appellera $\mathcal{B}(g, n) := \{g' \in \mathsf{TotCalc} : \forall i \leq n.\, (g'(i) = g(i))\}$ l'ensemble des g' qui coïncident avec g jusqu'au rang n, et $\mathcal{B}_0(g, n) := \mathcal{B}(g, n) \cap \mathsf{NulAPCR}$ celles qui, en outre, sont nulles à partir d'un certain rang (c'est-à-dire les fonctions prenant des valeurs de la forme $g(0), g(1), \ldots, g(n), ?, ?, \ldots, ?, 0, 0, 0, \ldots$).

(On rappelle que notre but est de montrer qu'il existe n tel que F soit constante sur $\mathcal{B}(g, n)$.)

(4) Soit $g \in \mathsf{TotCalc}$ et $n \in \mathbb{N}$. Expliquer *brièvement* pourquoi les conclusions de la question (3) valent encore en remplaçant $\mathsf{NulAPCR}$ par $\mathcal{B}_0(g, n)$ partout. (*Indication :* on peut par exemple fabriquer un élément de $\mathcal{B}_0(g, n)$ en insérant les valeurs $g(0), \ldots, g(n)$ avant un élément de $\mathsf{NulAPCR}$.) En particulier : on notera $\gamma(g, n, j)$ la conclusion de la dernière sous-question, c'est-à-dire que les fonctions $h \in \mathcal{B}_0(g, n)$ soient exactement celles de la forme $h = \varphi_{\gamma(g,n,j)}$ pour un certain j ; on expliquera *brièvement* pourquoi $\gamma(g, n, j)$ est calculable en fonction de j, de n et du code (dans $\mathsf{TotCode}$) d'un programme calculant g.

Si on n'a pas su traiter les questions (3) et/ou (4), pour la suite, **on retiendra juste ceci :** $\gamma(g, n, j)$ est calculable, et on a $\mathcal{B}_0(g, n) = \{\varphi_{\gamma(g,n,j)} : j \in \mathbb{N}\}$.

Algorithme A : Pour $g \in \mathsf{TotCalc}$, on considère maintenant le programme prenant deux entrées d et ℓ, décrit informellement ainsi :

— lancer l'exécution de $\varphi_d(0)$ pour au plus ℓ étapes [3] ;
— si l'exécution ne s'est pas terminée dans le temps imparti, terminer en renvoyant la valeur $g(\ell)$;
— sinon, disons si l'exécution s'est terminée en exactement $n \leq \ell$ étapes, lancer une boucle non bornée pour rechercher un $j \in \mathbb{N}$ tel que [4] $\hat{F}(\gamma(g, n, j)) \neq F(g)$:
  — si un tel j est trouvé, on renvoie $\varphi_{\gamma(g,n,j)}(\ell)$,

2. Si on sait ce que cela signifie, cette propriété peut s'exprimer ainsi : F est continue (ou, ce qui revient ici au même : localement constante) lorsque $\mathsf{TotCalc}$ est muni de la topologie [héritée de la topologie] produit sur $\mathbb{N}^{\mathbb{N}}$.

3. Si on préfère la notion d'arbre de calcul, remplacer par : « rechercher parmi les entiers $n \leq \ell$ le code d'un arbre de calcul de $\varphi_d(0)$ », et de même plus loin, remplacer « l'exécution s'est terminée en exactement $n \leq \ell$ étapes » par « n est le code d'un arbre de calcul de $\varphi_d(0)$ ». Cela ne changera rien au problème.

4. On rappelle à toutes fins utiles que $\hat{F}(\gamma(g, n, j)) = F(\varphi_{\gamma(g,n,j)})$ (c'est la définition de $\hat{F}$).

— sinon, bien sûr, l'algorithme ne termine pas.

On notera $g_d(\ell)$ la valeur calculée par cet algorithme A (s'il termine).

(5) (a) Justifier que les opérations présentées dans l'algorithme A ont bien un sens (i.e., que c'est bien un algorithme, qu'on a bien défini une fonction $\mathbb{N}\times\mathbb{N} \dashrightarrow \mathbb{N}$ partielle calculable $(d,\ell)\mapsto g_d(\ell)$). **(b)** En déduire qu'il existe une fonction calculable totale $e\mapsto e_d$ telle que $g_d = \varphi_{e_d}$ pour tout d.

(6) Soit $\mathscr{H} := \{d\in\mathbb{N} : \varphi_d(0)\downarrow\}$ l'ensemble des d tels que l'exécution de $\varphi_d(0)$ termine. **(a)** Lorsque $d\notin\mathscr{H}$, que vaut g_d ? et du coup, que vaut $\hat{F}(e_d)$? **(b)** Montrer que $\{d\in\mathbb{N} : \hat{F}(e_d) = F(g)\}$ est semi-décidable. **(c)** Le complémentaire $\complement\mathscr{H} = \{d\in\mathbb{N} : \varphi_d(0)\uparrow\}$ de $\mathscr{H}$ est-il semi-décidable ? **(d)** En déduire qu'il existe $d\in\mathscr{H}$ tel que $\hat{F}(e_d) = F(g)$.

(7) On a montré en (6) qu'il existe d tel que $\varphi_d(0)\downarrow$ et que $\hat{F}(e_d) = F(g)$. Soit n le nombre d'étapes d'exécution [5] de $\varphi_d(0)$. Montrer que pour tout $g'\in\mathcal{B}_0(g,n)$ on a $F(g') = F(g)$. (*Indication :* sinon, la recherche d'un j dans l'algorithme A aurait réussi à trouver un j : on pourra montrer qu'on aurait $g_d = \varphi_{\gamma(g,n,j)}$ donc $F(g_d)\neq F(g)$ dans ce cas.)

(8) On a montré en (7) que pour tout $g\in\mathsf{TotCalc}$ il existe n tel que pour tout $g'\in\mathcal{B}_0(g,n)$ on a $F(g') = F(g)$. On considère maintenant $g'\in\mathcal{B}(g,n)$ (qui n'est plus supposé nul à partir d'un certain rang) : d'après ce qu'on vient de dire, il existe n' tel que pour tout $g''\in\mathcal{B}_0(g',n')$ on a $F(g'') = F(g')$. En justifiant qu'on peut trouver $g''\in\mathcal{B}_0(g,n)\cap\mathcal{B}_0(g',n')$, montrer que $F(g') = F(g)$. Conclure.

(9) En observant que l'algorithme A peut s'écrire (à g variable) à partir du code d'un programme calculant g, justifier qu'un n dont on a montré l'existence en (8) peut être obtenu de façon calculable en fonction du code d'un programme calculant g. (On esquissera un algorithme B qui calcule un n en fonction d'un code r d'un programme calculant g.)

(10) On a montré en (3)–(8) le théorème de Kreisel-Lacombe-Shoenfield qui affirme que

$$\forall F : \mathsf{TotCalc}\to\mathbb{N}\text{ calculable. }\forall g\in\mathsf{TotCalc}.\ \exists n\in\mathbb{N}.\ \forall g'\in\mathcal{B}(g,n).\ (F(g') = F(g))$$

Montrer par un exemple simple que l'affirmation suivante [6]

$$\forall F : \mathsf{TotCalc}\to\mathbb{N}\text{ calculable. }\exists n\in\mathbb{N}.\ \forall g\in\mathsf{TotCalc}.\ \forall g'\in\mathcal{B}(g,n).\ (F(g') = F(g))$$

n'est pas valable. (*Indication :* proposer une fonction F qui évalue un nombre de valeurs $g(i)$ de g dépendant, disons, de $g(0)$.)

(11) On a prouvé en (9) qu'un n (tel que $\forall g'\in\mathcal{B}(g,n).\ (F(g') = F(g))$) peut être calculé algorithmiquement en fonction du code r d'un programme qui calcule g. En fait, on pourrait prouver un peu mieux (*ceci n'est pas demandé*) : il est possible de calculer un tel n par un algorithme qui a seulement le droit d'interroger les valeurs [7] de la fonction g. En *admettant* ce fait, expliquer pourquoi la fonction F elle-même peut être calculée par un tel algorithme.

(12) Quelle est la morale de l'ensemble de ce problème ? Autrement dit, expliquer *en français de façon informelle* le sens du théorème de Kreisel-Lacombe-Shoenfield : on cherchera à dire quelque chose de la forme « la seule manière de construire une fonction totale calculable sur l'ensemble des fonctions est la suivante ». On contrastera avec la situation des questions (1) et (2).

5. Ou si on préfère : le code de l'arbre de calcul.

6. Ce serait une affirmation de continuité *uniforme* de F.

7. C'est-à-dire : un algorithme qui a accès à un oracle calculant qui renvoie $g(i)$ quand on lui soumet la valeur i. L'algorithme a le droit d'appeler librement l'oracle un nombre fini quelconque de fois au cours de son exécution.