INF110 / CSC-3TC34-TP
Contrôle de connaissances

Logique et Fondements de l'Informatique

26 janvier 2026

Consignes.

Les exercices 1 à 5 (qui portent sur la logique) et le problème final (qui porte sur la calculabilité) 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). **Le non-respect de cette consigne pourra être pénalisé.**

La longueur du sujet ne doit pas effrayer : les réponses attendues sont souvent plus courtes que les questions. Notamment, l'énoncé du problème est long parce que des rappels et éclaircissements ont été faits et que les questions ont été rédigées de façon aussi précise que possible.

Dans les exercices portant sur Rocq (exercices 1 à 4), les erreurs de syntaxe Rocq ne seront pas pénalisées tant qu'on comprend clairement l'intention. De même, quand on demande d'écrire un λ-terme, il n'est pas indispensable de suivre exactement les notations introduites en cours.

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) : 2+2+4+3+3+6

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

Git : 3dd2bb6 Sun Jan 25 20:23:39 2026 +0100

Exercice 1.

Dans cet exercice, on considère des paires d'états d'une preuve en Rocq 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
H : (A /\ B) /\ C
============================
A
```

et on veut arriver à l'état suivant :

```
A, B, C : Prop
H1 : A /\ B
H2 : C
============================
A
```

(2) On part de l'état suivant :

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

et on veut arriver à l'état suivant :

```
First subgoal:

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

Second subgoal:

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

(3) On part de l'état suivant :

```
A, B, C : Prop
H1 : B
H2 : C
============================
A \/ B
```

et on veut arriver à l'état suivant :

```
No more goals.
```

(4) On part de l'état suivant :

```
A, B : Prop
H1 : A -> B
H2 : ~ B
============================
~A
```

et on veut arriver à l'état suivant :

```
A, B : Prop
H1 : A -> B
H2 : ~ B
H3 : A
============================
B
```

(5) On part de l'état suivant :

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

et on veut arriver à l'état suivant :

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

(6) On part de l'état suivant :

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

et on veut arriver à l'état suivant :

```
First subgoal:

============================
0 + 0 = 0

Second subgoal:

n : nat
IHn : n + 0 = n
============================
S n + 0 = S n
```

(7) On part de l'état suivant :

```
n : nat
IHn : n + 0 = n
============================
S n + 0 = S n
```

et on veut arriver à l'état suivant :

```
n : nat
IHn : n + 0 = n
============================
S (n + 0) = S n
```

(8) On part de l'état suivant :

```
n : nat
IHn : n + 0 = n
============================
S (n + 0) = S n
```

et on veut arriver à l'état suivant :

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

Exercice 2.

Si l'on dispose du lemme suivant en Rocq :

```
Lemma mul_0_r : forall n : nat, n * 0 = 0.
```

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 + 0
```

(2)

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

(3)

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

Exercice 3.

(A) Pour chacun des termes de preuve Rocq suivants, retrouver le théorème du calcul propositionnel intuitionniste qu'il prouve. (Ici, `A`, `B`, `C` vivent dans `Prop`.)

(1) `fun (H1 : A) (H2 : B) => H2`

(2) `fun (H1 : A) (H2 : ~ A) => H2 H1`

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

(B) Pour chaque formule logique suivante, en donner une preuve (en calcul propositionnel intuitionniste pour (1)–(4)). La preuve sera exprimée de préférence sous forme d'un λ-terme, qui n'a pas à être justifié si on est sûr qu'il est correct et qu'on veut gagner du temps ; toutefois, si on ne sait pas écrire le λ-terme ou si on a un doute à son sujet, on pourra donner une preuve en déduction naturelle (présentée comme arbre de preuve ou sous forme drapeau), qui vaudra au moins une partie des points.

(1) $A \Rightarrow A$

(2) $A \Rightarrow (A \wedge A)$

(3) $(A \vee A) \Rightarrow A$

(4) $\neg(A \vee B) \Rightarrow \neg A$

(5) $((\forall x.\ P(x)) \vee (\forall x.\ Q(x))) \Rightarrow (\forall x.\ (P(x) \vee Q(x)))$ (en logique du premier ordre intuitionniste)

Exercice 4.

(1) Définir en Rocq un type inductif pour représenter les arbres binaires contenant des entiers.

(2) Définir une fonction `miroir` qui, étant donné un arbre binaire, renvoie son miroir (symétrie gauche-droite).

(3) Énoncer un lemme en Rocq affirmant que le miroir du miroir d'un arbre est l'arbre lui-même.

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

(5) Définir une fonction `taille` qui calcule le nombre de noeuds d'un arbre binaire.

(6) Expliquer succinctement comment prouver en Rocq que la taille d'un arbre et la taille de son miroir sont égales.

Exercice 5.

Dans cet exercice, on veut montrer que la « formule de Scott », à savoir la formule propositionnelle suivante :

$$((\neg\neg A \Rightarrow A) \ \Rightarrow\ (A \vee \neg A)) \ \Rightarrow\ (\neg\neg A \vee \neg A)$$

n'est pas un théorème du calcul propositionnel intuitionniste.

Pour cela, on introduit l'espace topologique $X = \mathbb{R}$ et l'ouvert suivant :

$$\begin{aligned} U \ &= \ \left\{x \in \mathbb{R} \ :\ x > 0 \ \text{ et } \ \forall n \in \mathbb{N}.(x \neq 2^{-n})\right\} \\ &= \ \mathbb{R}_{>0} \setminus \{1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \ldots\} \\ &= \ ]1; +\infty[\ \cup \ ]\frac{1}{2}; 1[\ \cup \ ]\frac{1}{4}; \frac{1}{2}[\ \cup \ ]\frac{1}{8}; \frac{1}{4}[\ \cup \ \cdots \end{aligned}$$

(On rappellera brièvement pourquoi U est bien un ouvert.)

Donner la valeur, pour la sémantique des ouverts de X, de chaque sous-formule de la formule de Scott dans laquelle A a été remplacé par U, et en déduire pourquoi la formule de Scott n'est pas démontrable. On représentera chaque ensemble graphiquement en plus d'expliciter sa valeur avec des symboles.

(Il est recommandé de faire particulièrement au point 0 et, pour éviter les erreurs, de bien s'assurer qu'on a affaire à un ouvert à chaque fois.)

Problème 6.

Rappels de quelques définitions et notations habituelles. On rappelle qu'un **mot binaire** est une suite finie (éventuellement vide, c'est-à-dire de longueur nulle) de 0 et de 1. On notera $\{0,1\}^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, \ldots\}$ (ici, ε désigne le mot vide) l'ensemble de tous les mots binaires. La **longueur** $|w|$ d'un mot binaire $w \in \{0,1\}^*$ est le nombre total de bits qu'il contient (p.ex., $|00| = 2$ et $|\varepsilon| = 0$), et nous suivrons la convention de numéroter les bits de la gauche vers la droite de 0 à $|w| - 1$ (par exemple, le bit numéroté 0 de 1010 vaut 1, tandis que son bit numéroté 3 vaut 0). On dit qu'un mot binaire u est un **préfixe** d'un mot binaire v lorsque v commence par les bits de u, ou, formellement : $|u| \leq |v|$ et pour chaque $0 \leq j < |u|$, le bit numéroté j de v est égal au bit numéroté j de u. (Par exemple, 1010 est un préfixe de 1010111, tout mot binaire est un préfixe de lui-même, et ε est un préfixe de n'importe quel mot binaire.)

On pourra utiliser sans justification et sans commentaire le fait que les mots binaires peuvent être manipulés algorithmiquement (via un codage de Gödel qu'on ne demande pas de préciser) : notamment, calculer la longueur d'un mot, ou renvoyer son bit numéroté i, sont des opération calculables.

On appelle **arbre de Kleene** l'ensemble $\mathscr{K} \subseteq \{0,1\}^*$ de mots binaires défini de la manière suivante : un mot binaire $w \in \{0,1\}^*$ de longueur $|w|$ appartient à $\mathscr{K}$ lorsque, pour chaque $0 \leq i < |w|$, le bit numéroté i de w vaut

— 0 s'il existe un arbre de calcul (codé par un entier) $< |w|$ attestant $\varphi_i(0) = 0$,
— 1 s'il existe un arbre de calcul (codé par un entier) $< |w|$ attestant $\varphi_i(0) = r$, où $r \neq 0$,
— et arbitraire sinon,

où φ_i désigne la i-ième fonction générale récursive (d'arité 1).

Si on préfère parler en termes de machines de Turing, on pourra changer la définition en :

— 0 si la i-ième machine de Turing termine [1] en $< |w|$ étapes et renvoie 0,
— 1 si la i-ième machine de Turing termine en $< |w|$ étapes et renvoie une valeur $r \neq 0$,
— et arbitraire sinon

(cela ne changera rien de substantiel aux raisonnements).

Informellement dit, l'appartenance d'un mot w à $\mathscr{K}$ est déterminée par le résultat de l'exécution des $|w|$ premiers programmes, chacun jusqu'à la borne $|w|$, et le bit numéroté i de w est contraint, si le programme i termine, par le résultat de celui-ci.

(On prendra note du fait que $\varepsilon \in \mathscr{K}$ car la contrainte « pour chaque $0 \leq i < |w|$ » est vide — donc trivialement vérifiée — vu que $|\varepsilon| = 0$.)

(1) Montrer que si $v \in \mathscr{K}$ et si u est un préfixe de v. alors on a aussi $u \in \mathscr{K}$.

1. Sous-entendu : à partir d'une bande vierge (ou d'une bande représentant le nombre 0, ou tout autre état initial fixé sans importance).

On dit que $\mathscr{K}$ est un **arbre** de mots binaires [2] pour exprimer la propriété démontrée par cette question (1). (Formellement, un arbre de mots binaires est une partie $\mathscr{T} \subseteq \{0,1\}^*$ telle que si $v \in \mathscr{T}$ et que u est un préfixe de v, alors $u \in \mathscr{T}$.)

(2) Montrer que $\mathscr{K}$ est une partie *décidable* (i.e., calculable) de $\{0,1\}^*$. Sa fonction indicatrice est-elle primitive récursive ?

(3) Montrer qu'il existe dans $\mathscr{K}$ des mots binaires de longueur arbitrairement grande (formellement : $\forall n.\ \exists w \in \mathscr{K}.\ (|w| \geq n)$). On pourra même expliquer comment en calculer algorithmiquement un de longueur quelconque.

On appelle **branche infinie** de $\mathscr{K}$ une suite infinie $(b_i)_{i\in\mathbb{N}}$ de bits (i.e., un élément de $\{0,1\}^{\mathbb{N}}$) dont tous les préfixes appartiennent à $\mathscr{K}$, c'est-à-dire : $b_0 \cdots b_{\ell-1} \in \mathscr{K}$ pour tout $\ell \in \mathbb{N}$.

(4) *Indépendamment de tout ce qui précède,* montrer qu'il n'existe pas d'algorithme qui prend en entrée un $e \in \mathbb{N}$, *termine toujours en temps fini*, et renvoie
— 0 si $\varphi_e(0)\!\downarrow = 0$,
— 1 si $\varphi_e(0)\!\downarrow = r$ où $r \neq 0$,
— et une valeur arbitraire sinon (i.e., si $\varphi_e(0)$ n'est pas définie).

Si on préfère parler en termes de machines de Turing, on pourra montrer qu'il n'existe pas d'algorithme qui prend en entrée le code e d'une machine de Turing, *termine toujours en temps fini*, et renvoie
— 0 si la e-ième machine de Turing termine et qu'elle renvoie 0,
— 1 si la e-ième machine de Turing termine et qu'elle renvoie une valeur $r \neq 0$,
— et une valeur arbitraire sinon.

Indication : utiliser l'astuce de Quine pour construire un programme qui fait le contraire de ce qu'on lui prédit.

Attention ! On demande dans cette question une démonstration précise : on ne se contentera pas d'un raisonnement informel du type « on ne peut pas savoir si $\varphi_e(0)$ terminera un jour, donc on ne peut pas calculer sa valeur ».

(5) Déduire de (4) qu'il n'existe aucune branche infinie calculable de $\mathscr{K}$.

Le **lemme de Kőnig** est l'affirmation suivante : « tout arbre de mots binaires contenant des mots de longueur arbitrairement grande a une branche infinie » (les termes « arbre de mots binaires », « contenant des mots de longueur arbitrairement grande » et « branche infinie » ont été définis précisément ci-dessus). On ne demande pas de démontrer cette affirmation : néanmoins, c'est un théorème des mathématiques classiques usuelles.

(6) Pour résumer, que peut-on conclure du lemme de Kőnig, et des questions précédentes, concernant l'arbre $\mathscr{K}$? Comment expliqueriez-vous informellement la situation ?

(7) Expliquer pourquoi la question (5) *suggère* que le lemme de Kőnig n'est pas démontrable en mathématiques constructives. (On ne demande pas ici un raisonnement formel précis, mais une idée.)

2. Si cette terminologie semble mystérieuse, l'explication est que le graphe (infini d'après la question (3)) dont les sommets sont les éléments de $\mathscr{K}$, avec une arête de u à v lorsque u est un préfixe de v, est un arbre (infini) au sens de la théorie des graphes. Cette remarque n'est pas utile pour le présent exercice.