

Preuves assistées par ordinateur
Examen final – 27 mai 2005 – durée : 3h

Documents autorisés : notes de Cours/TD/TP

*Le barème ci-dessous est indicatif et est susceptible d'être modifié.
 Il est recommandé de lire le sujet.*

Exercice 1 (≈ 6 points) – Dédution naturelle et calcul des séquents

On se place en déduction naturelle intuitionniste, dans une signature Σ où sont déclarés un symbole de constante 0, un symbole de fonction s d'arité 1 et un symbole de prédicat p d'arité 1. Dans cet exercice, on autorise l'utilisation de la règle (admissible) d'affaiblissement.

Pour tout terme t formé sur la signature Σ et pour tout $n \in \mathbb{N}$, on note $s^n(t) \equiv \underbrace{s(\dots s(t)\dots)}_{n \text{ fois}}$.

1. Montrer que pour tout entier $n \in \mathbb{N}$ le séquent $p(0), \forall x (p(x) \Rightarrow p(s(x))) \vdash p(s^n(0))$ est dérivable. Quelle est la hauteur de la dérivation ainsi construite ?
 (On convient qu'une preuve réduite à un axiome est de hauteur 1, et on admet que l'utilisation de la règle admissible d'affaiblissement ne modifie pas la hauteur de la dérivation.)
2. Donner une dérivation du séquent $\forall x (p(x) \Rightarrow p(s(x))) \vdash \forall z (p(z) \Rightarrow p(s(s(z))))$.
3. Montrer que pour tout $n \in \mathbb{N}$ le séquent $\forall x (p(x) \Rightarrow p(s(x))) \vdash \forall z (p(z) \Rightarrow p(s^{2^n}(z)))$ est dérivable. Quelle est la hauteur de la dérivation ainsi construite ?
4. Déduire de ce qui précède une dérivation du séquent $p(0), \forall x (p(x) \Rightarrow p(s(x))) \vdash p(s^{1024}(0))$ dont la hauteur est inférieure à 100.
5. Montrer que le séquent $p(0), \forall x (p(x) \Rightarrow p(s(x))) \vdash \forall z p(z)$ n'est pas dérivable.
Indication : On pourra raisonner à l'aide d'un contre-modèle.
6. Dériver le séquent de la question 2 en calcul des séquents classique.

Exercice 2 (≈ 3 points) – Codages dans le système T

Dans cet exercice, on se place dans le système T restreint aux constructions flèche et produit, dont les types et les termes sont donnés par

Types $\tau, \sigma ::= \text{Nat} \mid \tau \rightarrow \sigma \mid \tau \times \sigma$
Termes $M, N ::= x \mid \lambda x : \tau. M \mid MN \mid \langle M, N \rangle \mid \pi_1(M) \mid \pi_2(M) \mid 0 \mid S \mid \text{rec}^\tau$

et dont les règles de réduction sont les règles habituelles restreintes aux constructions ci-dessus.

1. Écrire dans ce formalisme une implémentation $\text{pred} : \text{Nat} \rightarrow \text{Nat}$ de la fonction prédécesseur tronquée, qui à tout $n \in \mathbb{N}$ associe $\max(0, n - 1)$. En déduire un terme $\text{minus} : \text{Nat} \rightarrow \text{Nat} \rightarrow \text{Nat}$ qui implémente la soustraction tronquée $n \dot{-} m = \max(0, n - m)$.
2. Écrire une fonction $\text{fibo} : \text{Nat} \rightarrow \text{Nat}$ qui calcule les termes de la suite de Fibonacci $(u_n)_{n \in \mathbb{N}}$ définie par $u_0 = u_1 = 1$ et $u_n = u_{n-1} + u_{n-2}$ pour tout $n \geq 2$. On pourra commencer par écrire une fonction $\text{fibo}_2 : \text{Nat} \rightarrow \text{Nat} \times \text{Nat}$ qui calcule pour tout $n : \text{Nat}$ le couple $\langle u_n, u_{n+1} \rangle$.
3. Implémenter des fonctions $\text{min} : \text{Nat} \rightarrow \text{Nat} \rightarrow \text{Nat}$ et $\text{max} : \text{Nat} \rightarrow \text{Nat} \rightarrow \text{Nat}$ qui calculent respectivement le minimum et le maximum de leurs deux arguments.

Exercice 3 (\approx 5 points) – Théorie des types simples

Dans cet exercice, on se place en théorie des types simples (sans entiers naturels), en utilisant les codages au second ordre des unités \top et \perp , des connecteurs \wedge et \vee , du quantificateur existentiel \exists^τ et de l'égalité de Leibniz $=_\tau$ (sur tous les types simples τ).

Dans les dérivations on autorise toutes les règles admissibles liées aux constructions \top , \perp , \wedge , \vee , \exists^τ et à l'égalité $=_\tau$ (cf cours) ainsi que la règle d'affaiblissement. On écrira systématiquement toutes les prémisses de typage et de bonne formation de contexte qui apparaissent dans les règles (qu'elles soient primitives ou admissibles), mais sans faire figurer la dérivation qui les accompagne.

1. Étant donné un type τ quelconque, donner une dérivation de $\langle \emptyset \rangle \emptyset \vdash \exists f : (\tau \rightarrow \tau). \forall x : \tau. fx =_o x$. (Où \emptyset désigne indifféremment la signature vide ou le contexte vide.)
2. Dériver $\langle x : o \rangle x =_o \neg x \vdash x \Rightarrow \neg x$.
3. En déduire une dérivation de $\langle \emptyset \rangle \emptyset \vdash \forall x : o. \neg(x =_o \neg x)$.

Étant donné un type simple τ , on appelle *combineur de point fixe sur τ* tout objet Y de type $(\tau \rightarrow \tau) \rightarrow \tau$ satisfaisant la proposition $\text{Fix}^\tau(Y)$ définie en théorie des types simples par :

$$\text{Fix}^\tau(Y) \equiv \forall f : (\tau \rightarrow \tau). f(Y f) =_\tau Y f$$

4. Donner une dérivation de $\langle y : (o \rightarrow o) \rightarrow o \rangle \text{Fix}^o(y) \vdash \exists x : o. x =_o \neg x$.
5. En déduire une dérivation de $\langle \emptyset \rangle \emptyset \vdash \neg \exists y : ((o \rightarrow o) \rightarrow o). \text{Fix}^o(y)$

Exercice 4 (\approx 6 points) – Arbres binaires de recherche en Coq

Dans cet exercice, on considère un développement en Coq paramétré par un type de données A muni d'une relation d'ordre total `less`, lesquels sont introduits à l'aide des déclarations suivantes :

```
Parameter A : Set.
Parameter less : A -> A -> Prop.

Axiom less_refl : forall (x : A),      less x x.
Axiom less_trans : forall (x y z : A), less x y -> less y z -> less x z.
Axiom less_asym : forall (x y : A),   less x y -> less y x -> x = y.
Axiom less_total : forall (x y : A),  less x y \/ less y x.
```

A – Arbres binaires Le type `tree` des arbres binaires dont les nœuds sont étiquetés par des éléments de type A est défini en Coq par :

```
Inductive tree : Set :=
| Leaf : tree
| Node : A -> tree -> tree -> tree.
```

1. Définir en Coq des fonctions `height` et `size` de type `tree` \rightarrow `nat` qui calculent respectivement la hauteur et le nombre de nœuds d'un arbre (en supposant que `height Leaf = size Leaf = 0`). Pour écrire la fonction `height`, on supposera l'existence d'une fonction `max : nat` \rightarrow `nat` \rightarrow `nat` calculant le maximum de ses deux arguments.
2. On cherche à établir que pour tout arbre $t : \text{tree}$, on a `height t` \leq `size t`. Donner les grandes lignes de la preuve *informelle*. (On ne demande pas les détails et encore moins le script!) Sur quel lemme purement arithmétique repose l'argument central de cette preuve?
3. Comment définit-on en Coq la fonction `max : nat` \rightarrow `nat` \rightarrow `nat` utilisée par la fonction `height`?
4. Donner une définition inductive du prédicat `mem : A` \rightarrow `tree` \rightarrow `Prop` qui exprime que son premier argument appartient à l'arbre donné en second argument.

B – Arbres binaires de recherche On dit qu'un arbre $t : \text{tree}$ est un *arbre binaire de recherche* (a.b.r.) si dans tout sous-arbre de t de la forme `Node x t1 t2`, tous les éléments de $t1$ sont inférieurs ou égaux à x tandis que tous les éléments de $t2$ sont supérieurs ou égaux à x .

5. Définir en Coq des relations `tree_le` et `tree_ge` de type `tree → A → Prop` qui expriment que tous les éléments de l'arbre donné en premier argument sont inférieurs ou égaux (resp. supérieurs ou égaux) à la valeur donnée en second argument.
6. Donner une définition inductive en Coq du prédicat `abr : tree → Prop` qui exprime que son argument est un a.b.r.

On suppose que la comparaison entre éléments de type A est effectuée au moyen d'une fonction `cmp` de type `A → A → bool` déclarée en Coq par

```
Parameter cmp : A -> A -> bool.
```

et munie de l'axiome suivant :

```
Axiom cmp_less : forall (x y : A), cmp x y = true <-> less x y.
```

7. À l'aide de la fonction de comparaison introduite ci-dessus, définir en Coq une fonction d'insertion `insert : A → tree → tree` qui insère un élément $x : A$ dans un arbre $t : \text{tree}$ tout en préservant la structure d'a.b.r. lorsque t est déjà un a.b.r.
8. Énoncer en Coq deux lemmes `correct_A` et `correct_B` qui expriment respectivement la correction de la fonction d'insertion
 - A. par rapport à la relation d'appartenance `mem : A → tree → Prop` et
 - B. par rapport à la condition `abr : tree → Prop`.

Par quel type d'induction se fait la preuve de `correct_A`? la preuve de `correct_B`?

9. L'hypothèse `less_total` (qui exprime que la relation d'ordre `less` est totale) est-elle nécessaire pour garantir l'invariant `correct_A`? pour garantir l'invariant `correct_B`? Si oui, à quel(s) endroit(s) précis cette hypothèse intervient-elle dans la preuve du ou des lemmes correspondants? On illustrera chaque problème recensé par un contre-exemple bien choisi.