Master Informatique FIIL — 2016–17
Preuves Interactives et Applications — 5 octobre 2016
http://www.lri.fr/~paulin/PreuvesInteractives

Examen - 16 novembre 2015

L'examen dure 3 heures. L'énoncé est composé de **??** pages.
Le barème est indicatif. Toutes les réponses devront être clairement justifiées.
Documents autorisés : copie des transparents de cours, sujets et corrigés du TD disponibles sur le site à l'exclusion de tout autre document.

Le tableau résumant les règles de la déduction naturelle est donné à la fin du sujet.

Inscrivez votre nom sur chaque copie et numérotez-la.
Cacheter toutes les copies, recopier le numéro d'anonymat sur les intercalaires !

Exercice 1 (Questions de cours, 4 points)

Dire si les affirmations suivantes sont vraies ou fausses dans chacun des systèmes Isabelle/-HOL et Coq. On attend donc deux réponses à chaque fois. Une bonne réponse compte 1 point, une mauvaise réponse fait perdre $\frac{1}{2}$ point sur cet exercice. On ne demande pas de justification.

1. Le système est basé sur le principe d'isomorphisme de Curry-Howard.
2. On peut formaliser (sans axiome) la fonction qui décide pour une codage d'une machine Turing si l'exécution s'arrête pour toutes les entrées.
3. On peut prouver dans ces systèmes tout ce qui est vrai, leur logique est complète.
4. Un type n'est jamais vide.

Exercice 2 (λ-calcul simplement typé, 2 points)

On rappelle que la constante $\forall$ a pour type $(\alpha \to \texttt{Prop}) \to \texttt{Prop}$, la constante $\_=\_$ a pour type $\alpha \to \alpha \to \texttt{Prop}$, la constante $\_+\_$ a pour type $\texttt{nat} \to \texttt{nat} \to \texttt{nat}$ et la constante 0 a pour type $\texttt{nat}$. On utilise une notation infixe $x + y$ (resp. $x = y$) pour le résultat de l'application de la constante $\_+\_$ (resp. $\_=\_$) aux termes x et y.

Dire si les expressions suivantes sont bien typées dans le λ-calcul simplement typé et si oui donner le type le plus général, si non expliquer pourquoi.

1. $\lambda f.\lambda x.f\ x$
2. $\lambda f.\lambda x.f\ (x\ x)$
3. $\lambda P.\forall(\lambda x.(P\ x\ x))$
4. $\lambda f.(f\ (\lambda x.(x + x)) = 0)$

Exercice 3 (Preuves, 4 points)

Trouver et corriger 5 coquilles ou erreurs dans les définitions et preuves du script suivant. Pour trouver les erreurs, il peut-être utile de dessiner les peuves à la main.

Cet exercice peut être réalisé au choix de l'étudiant en Coq ou en Isabelle. Par conséquent, il faut choisir **l'un des deux scripts** ci-dessous :

- **Script Isabelle**

```
lemma and_sym: "A ∧B ⟶B ∧A"
apply(rule impI)
apply(rule conjI)
apply(erule conjE)
apply(erule conjE)
apply(assumption)
done

lemma all_istr: "(∀x. A ⟶P(x)) = (A ⟶(∀x. P(x)))"
apply(rule iffI)
 apply(rule allI)
  apply(rule mp)
 apply(erule spec)
apply(assumption)
apply(rule allI)
apply(rule impI)
apply(rule_tac P="λ x. P x" in spec)
apply(drule mp)
apply(assumption)+
done

(* la declaration suivante est incorrecte, corriger *)
datatype 'a seq = Empty | Seq 'a 'a seq

fun conc :: "'a seq ⇒'a seq ⇒'a seq"
where
  "conc Empty ys = ys"
| "conc (Seq x xs) ys = Seq x (conc xs ys)"

lemma conc_empty:"conc xs Empty = xs"
  apply(induct xs)
  apply(simp)
  done

lemma conc_assoc: "conc (conc xs ys) zs = conc rs (conc ys zs)"
  by (induct xs) simp_all
```

Rappel : les règles `iffI`, `mp` et `spec` ont la forme :

```
"(P ⟹Q) ⟹(Q ⟹P) ⟹P = Q"
"(P ⟶Q) ⟹P ⟹Q"
"(∀x. P x) ⟹P x"
```

- **Script Coq** :

```
Lemma and_sym (A B : Prop) : A /\ B → B /\ A.
intro H.
split.
destruct H.
destruct H.
assumption.
Qed.

Lemma all_istr (A:Prop) X (P: X → Prop)
   : (∀ x, A → P x) ↔ (A → ∀ x, P x).
split; intro H.
+ intro x.
  apply H.
  assumption.
+ intros x HA.
  apply H.
  assumption.
Qed.

Set Implicit Arguments.
(* la declaration suivante est incorrecte, corriger *)
Inductive seq A := Empty : seq A | Seq : A → seq → seq A.

Fixpoint conc A (xs ys: seq A) : seq A :=
  match xs with Empty ⇒ ys
              | Seq x xs ⇒ Seq x (conc xs ys)
  end.

Lemma conc_empty A xs : conc xs (Empty A) = xs.
induction xs.
simpl; auto.
Qed.

Lemma conc_assoc A : ∀ xs ys zs : seq A,
        conc (conc xs ys) zs = conc rs (conc ys zs).
induction xs; simpl; auto.
Qed.
```

On rappelle que `A <-> B` est défini comme `A -> B /\ B -> A` et donc que la tactique `split` engendre deux sous-buts `A -> B` et `B -> A`.

Exercice 4 (Modélisation, 10 points)
Cet exercice peut être réalisé au choix en utilisant la syntaxe de Coq ou celle d'Isabelle.

Un arbre rouge-noir est un type particulier d'arbre binaire de recherche dans lequel chaque nœud a un attribut supplémentaire : sa couleur, qui est soit rouge soit noire. Il préserve des propriétés d'équilibre qui garantissent des temps d'accès, d'insertion ou de retrait logarithmiques.

Les données stockées dans l'arbre sont dans un type A qui est muni d'une opération de comparaison `compare` dont le résultat `compare` $x\,y$ appartient à l'ensemble $\{$`Lt`, `Eq`, `Gt`$\}$. Cet ensemble est défini comme un type inductif à trois valeurs constantes. La fonction `compare` vérifie les propriétés suivantes :

- (`compare` $x\,y$ = `Eq`) $\Leftrightarrow$ ($x = y$)
- (`compare` $x\,y$ = `Lt`) $\Leftrightarrow$ (`compare` $y\,x$ = `Gt`)
- (`compare` $x\,y$ = `Lt`) $\wedge$ (`compare` $y\,z$ = `Lt`) $\Rightarrow$ (`compare` $x\,z$ = `Lt`)

On dira que x est plus petit que y si `compare` $x\,y$ = `Lt`.

Le type des arbres (ordinaires) est donné par la déclaration suivante en Coq

```
Inductive tree A := leaf : tree A | node : A → tree A → tree A → tree A
```

ou en Isabelle/HOL

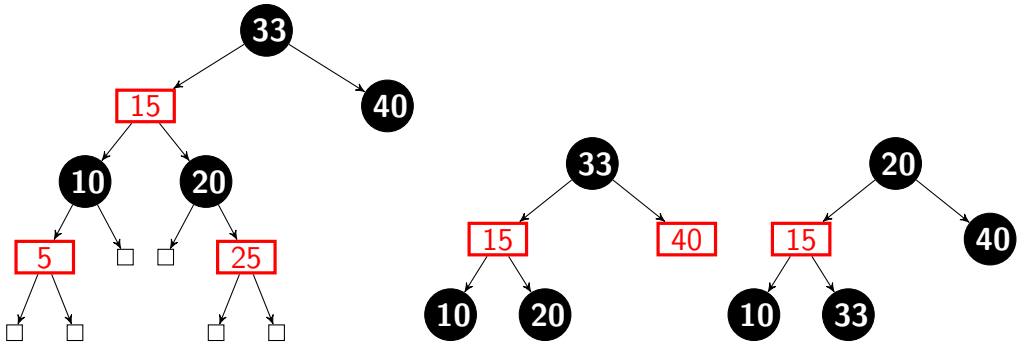
```
datatype 'a tree = leaf | node 'a "'a tree" "'a tree"
```

On rappelle qu'un arbre binaire de recherche a la propriété que pour chaque nœud interne (`node` $n\,l\,r$), les valeurs stockées dans le sous-arbre gauche l sont plus petites que n qui est lui-même plus petit que toutes les valeurs stockées dans le sous-arbre droit r.

Dans les arbres rouge-noir, on a de plus les propriétés suivantes

- La racine est noire ;
- Le parent d'un nœud rouge est noir ;
- Tous les chemins d'une feuille à la racine contiennent le même nombre de nœuds noirs.

1. Dire si les arbres suivants sont bien des arbres rouge-noir sur les entiers. Les nœuds noirs sont représentés par des ronds et les nœuds rouges par des carrés.



2. Définir un type de données `color` pour représenter les deux couleurs rouge et noir par les constantes `R` (rouge) et `B` (noir).
3. Definir le type `ctree` `A` des arbres colorés, tel que chaque nœud comporte à la fois une valeur de type A et une couleur. On pourra réutiliser le type `tree`.
4. Ecrire les expressions du type `ctree` A qui correspondent aux arbres de la question **??** bien formés (on pourra définir des abbréviations pour alléger l'écriture).

5. Ecrire une fonction `testcol` qui prend en argument une couleur c et un arbre t et renvoie un booléen qui est vrai exactement lorsque t est un arbre non réduit à une feuille dont la racine est de couleur c.
6. Définir une fonction `hb` qui compte le nombre de nœuds noirs sur les branches d'un arbre tout en vérifiant qu'il y a le même nombre de nœuds noirs sur chaque branche. La fonction calcule un objet de type `option nat`. Elle renvoie (`Some` n) si toutes les branches de l'arbre on exactement n nœuds noirs et `None` sinon.
7. Définir de manière récursive une fonction `isin` qui étant donnés une valeur a dans A et un arbre t de type `ctree` A qui vérifie de plus la propriété d'être un arbre binaire de recherche, renvoie vrai si et seulement si la valeur a apparaît dans un nœud de t.
8. Soit la définition inductive en syntaxe Coq

```
Inductive allt A (f : A → bool) : tree A → Prop :=
   allleaf : allt A f (leaf A)
|  allnode : ∀ (a:A) (l r : tree A), f a = true
             → allt A f l → allt A f r → allt A f (node a l r)
```

   ou bien en syntaxe Isabelle/HOL :

```
inductive_set allt :: "('a ⇒bool) ⇒'a tree set" for f::"'a ⇒bool"
where
  allleaf : "leaf ∈ allt f"
| allnode : "(f a) ⟹ l ∈ allt f ⟹ r ∈ allt f ⟹ (node a l r) ∈ allt f"
```

   - Donner une autre formulation (en langage naturel et non inductive) de ce que représente cette définition.
   - Définir la même notion de manière récursive sur la structure de l'arbre.
9. Définir par le moyen de votre choix un prédicat `RB` sur les arbres colorés qui spécifie que l'arbre vérifie bien les quatre invariants des arbres rouge-noir. On appellera arbre rouge-noir un arbre coloré qui vérifie la propriété `RB`.
10. On définit une fonction `insert` qui étant donnés une valeur a un arbre rouge-noir t renvoie un arbre rouge-noir qui contient tous les éléments de t plus la valeur a.
    - donner le type de la fonction `insert`
    - écrire une formule qui exprime le fait que le programme `insert` est correct par rapport à sa spécification (c'est-à-dire qu'il ajoute l'élément a à l'arbre en préservant les invariants).

Rappel des règles logiques

Hypothèse	si $A \in \Gamma$ alors $\Gamma \vdash A$	
Connecteur	Introduction	Élimination
$\top$	$\Gamma \vdash \top$	
$\bot$		$\dfrac{\Gamma, \neg C \vdash \bot}{\Gamma \vdash C}$
$\neg$	$\dfrac{\Gamma, A \vdash \bot}{\Gamma \vdash \neg A}$	$\dfrac{\Gamma \vdash \neg A \quad \Gamma \vdash A}{\Gamma \vdash C}$
$\wedge$	$\dfrac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}$	$\dfrac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \quad \dfrac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}$
$\vee$	$\dfrac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \quad \dfrac{\Gamma \vdash B}{\Gamma \vdash A \vee B}$	$\dfrac{\Gamma \vdash A \vee B \quad \Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma \vdash C}$
$\Rightarrow$	$\dfrac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B}$	$\dfrac{\Gamma \vdash A \Rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B}$
$\forall$	$\dfrac{\Gamma \vdash P \quad x \notin \mathsf{FV}(\Gamma)}{\Gamma \vdash \forall x, P}$	$\dfrac{\Gamma \vdash \forall x, P}{\Gamma \vdash P[x \leftarrow t]}$
$\exists$	$\dfrac{\Gamma \vdash P[x \leftarrow t]}{\Gamma \vdash \exists x, P}$	$\dfrac{\Gamma \vdash \exists x, P \quad \Gamma, P \vdash C \quad x \notin \mathsf{FV}(\Gamma, C)}{\Gamma \vdash C}$
$=$	$\Gamma \vdash \forall x, x = x$	$\dfrac{\Gamma \vdash t = u \quad \Gamma \vdash P[x \leftarrow t]}{\Gamma \vdash P[x \leftarrow u]}$