Examen - Ingénierie de la preuve

Durée : 3h00

Les notes de cours, de travaux dirigés et de travaux pratiques sont autorisées. Le sujet comporte 2 pages et les trois parties sont complètement indépendantes. Les règles logiques utiles pour ce contrôle sont redonnées à la fin du sujet. Le barème est donné à titre indicatif.

On s'attachera à soigner la présentation, en particulier lors des démonstrations par récurrence et on sera vigilant quant à la syntaxe lors de l'écriture de fragment de code devant être accepté par `Coq`*.*

1 Questions de logique

Question 1 Rappeler la différence entre logique intuitioniste et logique classique.
Question 2 Pour chacune des formules suivantes, proposer une démonstration sous forme d'arbre en déduction naturelle :

$$(A \to B) \to (\neg B \to \neg A) \qquad (\neg B \to \neg A) \to (A \to B)$$

On travaillera *a priori* en logique intuitioniste. On pourra toutefois passer en logique classique si cela s'avère nécessaire, sous réserve de le justifier clairement.
Question 3 Ecrire la preuve de $(A \to B) \to (\neg B \to \neg A)$, en utilisant les tactiques de Coq afin que cette démonstration soit acceptée par le système.
Question 4 Construire des termes de preuve pour chacune des démonstrations de la question 2.

2 Définitions inductives

2.1 Représentation d'expressions arithmétiques sous forme d'arbres

On considère le type inductif suivant :

```
Inductive arbre : Set :=
| feuille : nat -> arbre
| plus    : arbre -> arbre -> arbre
| moins   : arbre -> arbre -> arbre
| fois    : arbre -> arbre -> arbre.
```

Question 5 Comment représenter l'expression (1+2)*3 avec ce type de données ?
Question 6 L'ajout de cette définition dans `Coq` conduit à la construction automatique d'un principe d'induction associé (à savoir `arbre_ind`). Donner l'énoncé de ce principe d'induction.
Question 7 Ecrire une fonction permettant de tester si un objet de type `arbre` est une feuille ou non.
Question 8 Ecrire une fonction d'évaluation `eval : arbre -> nat` qui calcule la valeur de l'expression arithmétique représentée par l'arbre ?

2.2 Représentation d'expressions arithmétiques sous forme de listes

On considère maintenant la représentation des expressions arithmétiques sous forme de listes en forme polonaise inverse. L'expression (1+2)*3 sera représentée par la liste [1, 2, +, 3, *] . On utilisera pour cela les définitions inductives suivantes :

```
Inductive operateur : Set :=
  oplus : operateur | omoins : operateur | ofois : operateur.
```

```
Inductive liste : Set :=
  | vide : liste
  | cons_operateur : operateur -> liste -> liste
  | cons_valeur : nat -> liste -> liste
```

Question 9 Ecrire un terme de type `liste` représentant l'expression $(1 + 7) * (9 - 2) + 3$.
Question 10 Préciser le principe d'induction associé à la définition du type inductif `liste`.
Question 11 Définir, en `Coq`, une fonction `append` de concaténation de deux listes.
Question 12 Ecrire une fonction de traduction `trad : arbre -> liste` d'une expression arithmétique représentée sous forme d'arbre vers une expression en forme polonaise inverse représentée sous forme d'une liste.

On reprend la représentation usuelle des piles :

```
Inductive pile : Set := pvide : pile | empiler : nat -> pile -> pile.
```

Question 13 Programmer les fonctions `depiler` et `sommet` pour ce type de données. On supposera, par convention, que le sommet de la pile vide renvoie 0 et que dépiler une pile vide retourne une pile vide.
Question 14 A l'aide des piles, programmer en Coq une fonction d'évaluation (`eval_l : liste -> nat`) d'une expression en forme polonaise inverse bien formée.
Question 15 Enoncer dans Coq la propriété suivante : l'évaluation d'un arbre et l'évaluation de sa traduction en forme polonaise inverse donne bien toujours le même résultat. Comment peut-on procéder pour démontrer ce résultat dans `Coq` ? Donner les grandes étapes de cette démonstration.
Question 16 Ecrire un prédicat `bien_formee : liste -> Prop` permettant de tester si une liste est bien formée (c'est-à-dire si elle représente bien une expression en forme polonaise inverse). Par exemple la liste [1, -, 2, +, -] n'est pas bien formée. Quelle propriété de la traduction peut-on énoncer à la suite de cette définition ?

A Rappel des règles logiques

On rappelle que $\bot \equiv$ `False` et $\neg A = A \to \bot$.

	règle d'élimination	règle(s) d'introduction
$\to \forall$	$\dfrac{\Gamma \vdash A \quad \Gamma \vdash A \to B}{\Gamma \vdash B}$	$\dfrac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}$
$\wedge$	$\dfrac{\Gamma, A, B \vdash P \quad \Gamma \vdash A \wedge B}{\Gamma \vdash P}$	$\dfrac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}$
$\vee$	$\dfrac{\Gamma, A \vdash P \quad \Gamma, B \vdash P \quad \Gamma \vdash A \vee B}{\Gamma \vdash P}$	$\dfrac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \quad \dfrac{\Gamma \vdash B}{\Gamma \vdash A \vee B}$
$\bot$	$\dfrac{\Gamma \vdash \bot}{\Gamma \vdash A}$	
$\exists$	$\dfrac{\Gamma, x : A, P\ x \vdash Q \quad \Gamma \vdash \exists x : A, P\ x}{\Gamma \vdash Q}$	$\dfrac{\Gamma, v : A \vdash P\ v}{\Gamma \vdash \exists x : A, P\ x}$