Examen cours 2-7-2 Assistants de preuve

Mardi 9 février 2010

L'énoncé est composé de 4 pages. L'examen dure 2 heures. Les notes de cours manuscrites ainsi que les supports de cours distribués cette année sont les seuls documents autorisés. Les exercices sont indépendants. Les exercices 1 et 2 nécessitent d'écrire des termes et des preuves Coq ; la notation de l'examen sera indulgente vis-à-vis de la syntaxe employée à condition qu'il n'y ait pas d'ambiguïté sur leur signification.

1 Définitions inductives : lambda-termes (6 points)

Le type L défini ci-dessous permet de représenter les lambda-termes :

```
Inductive option (A : Type) : Type := Some : A → option A | None : option A
Inductive L (A:Type) : Type :=
  | Var : A → L A
  | App : L A → L A → L A
  | Lam : L (option A) → L A.
```

Le paramètre A sert à représenter les variables libres du lambda-terme. Dans la suite ce paramètre sera toujours traité comme implicite.

Les variables liées utilisent une notation à la de Bruijn mais à l'aide du type option plutôt que les entiers. Dans une notation à la de Bruijn, une variable liée est représentée par un entier n qui correspond au nombre d'abstractions λ qu'il faut traverser jusqu'au λ lieur. La même variable est donc représentée par des entiers différents suivant sa position dans le terme.

Dans notre langage, l'entier de de Bruijn n est représenté par le terme Some^n None. Par exemple le constructeur None représente une variable liée par le premier Lam englobant. Une variable libre a sera représentée par le terme $\mathsf{Some}^n\ a$ si elle se trouve sous n constructeurs Lam.

Ainsi pour représenter le terme $\lambda x.((\lambda y.(y\,x))\,(z\,x))$ qui contient l'unique variable libre z et dont la représentation en de Bruijn est $\lambda\,((\lambda\,(0\,1))\,(z\,0))$, on utilisera le terme :

```
Lam (App (Lam (App (Var None) (Var (Some None)))) (App (Var (Some z)) (Var None))
```

Ce terme a pour type L A si z a pour type A. On remarquera que la même variable x est représentée une première fois par Var (Some None) sous le λy et une deuxième fois par Var None en dehors de cette abstraction.

1. Le principe d'induction L_ind engendré par Coq pour L sert à prouver :
   $\forall(P : \forall A,\ \mathsf{L}\,A \to \mathsf{Prop})(A : \mathsf{Type})(x : \mathsf{L}\,A),\ P\,A\,x$.
   Donner le type complet de ce principe d'induction.
2. Écrire en Coq une fonction map qui étant donnés deux types A et B et une fonction f de type $A \to B$ et un terme t de type L A applique la fonction f à toutes les variables libres de t pour le transformer en un terme de type L B.

3. Écrire en Coq une opération subst de substitution parallèle qui prend en argument un terme t de type $\mathsf{L}\,A$ et une substitution s de type $A \to \mathsf{L}\,B$ et renvoie un terme de type $\mathsf{L}\,B$. La substitution s associe à chaque variable a de type A un lambda-terme sur un ensemble de variables B. Le terme $\mathsf{subst}\,t\,s$ est obtenu en remplaçant toutes les variables libres a de t par $s\,a$.
Indication : dans le cas de la lambda-abstraction $\mathsf{Lam}\,t'$, il faut appliquer à t' une substitution de type $\mathsf{option}\,A \to \mathsf{L}\,(\mathsf{option}\,B)$. Cette substitution ne change pas la variable None (qui est liée). Par contre, pour une variable de la forme $\mathsf{Some}\,a$, on applique la substitution s à a, ce qui donne un terme de type $\mathsf{L}\,B$ auquel il faut ensuite appliquer un changement de variable en renommant chaque variable b en $\mathsf{Some}\,b$ afin de prendre en compte le passage du lieur.

2 Égalité de preuves d'égalité (7 points)

En règle générale et sans axiome, il n'est pas possible de prouver que deux preuves d'un même théorème sont égales dans le formalisme de Coq. C'est cependant le cas pour les preuves de propriété d'égalité sur un type où l'égalité est décidable. L'exercice suivant s'intéresse plus particulièrement aux propriétés de la forme : $\mathsf{true} = b$.

On rappelle que l'égalité de Leibniz $t = u$ est définie comme un type inductif dont l'unique constructeur est $\mathsf{refl\_equal}\,t$ de type $t = t$:

```
Inductive eq (A : Type) (x : A) : A → Prop := refl_equal : eq A x x.
```

et que le principe d'élimination dépendante est :

$$\forall t\,(P : \forall y,\ t = y \to \mathsf{Prop}),\ P\,t\,(\mathsf{refl\_equal}\,t) \to \forall y\,(e : t = y),\ P\,y\,e$$

1. Le principe d'élimination dépendante suggère que toute preuve d'égalité e est une preuve de réflexivité. Pourquoi n'est-il pas possible d'énoncer et à plus forte raison de prouver : $\forall t\,y\,(e : t = y),\ e = \mathsf{refl\_equal}\,t$?
On cherche maintenant à montrer la propriété $\forall t\,(e : t = t),\ e = \mathsf{refl\_equal}\,t$ en utilisant une élimination dépendante sur e. Quel problème cela pose-t-il ?
2. Définir en Coq une fonction $\mathsf{eqbool\_dep}$ de type

$$\forall (P : \mathsf{bool} \to \mathsf{Prop}),\ P\,\mathsf{true} \to \forall b,\ P\,b \to \mathsf{Prop}$$

telle que

$$\mathsf{eqbool\_dep}\,P\,h_1\,b\,h_2 = \begin{cases} h_1 = h_2 & \text{si } b = \mathsf{true}, \\ \mathsf{False} & \text{sinon.} \end{cases}$$

Rappel : la syntaxe du match dépendant est

```
match x return T with ... end
```

où le terme T (dépendant de x) est le type des valeurs retournées par chacune des branches du match.
3. Dans le lemme $\mathsf{eqbool\_refl} : \forall b\,(h : \mathsf{true} = b),\ \mathsf{eqbool\_dep}\,P\,(\mathsf{refl\_equal}\,\mathsf{true})\,b\,h$, comment faut-il instancier la propriété P pour que le lemme soit bien formé ? Prouver ce lemme. (On donnera la suite de tactiques Coq à utiliser.)

4. Quel est le type réduit du terme eqbool_refl true ?
5. En déduire une preuve du lemme eqbool_irrel : $\forall b\,(h_1\,h_2 : \mathsf{true} = b),\ h_1 = h_2$.
6. Étant donné un type A et une fonction $f : A \to \mathsf{bool}$, prouver le théorème

$$\forall (u\,v : \{x : A \mid \mathsf{true} = f\,x\}),\ \mathsf{projT1}\,u = \mathsf{projT1}\,v \to u = v.$$

Remarque : u et v sont des paires de la forme (x, h) avec x de type A et h une preuve de $\mathsf{true} = f\,x$. Le terme $\mathsf{projT1}\,u$ désigne la première projection de u. Le théorème énonce donc que de telles paires sont égales à partir du moment où leurs premières composantes le sont.

3 Logique de Hoare et exceptions (7 points)

Dans un langage de programmation avec exceptions, il y a deux façons possibles de sortir d'un bloc de code : de façon normale ou en lançant une exception. La logique de Hoare peut s'adapter à un tel langage en autorisant plusieurs postconditions : une pour les comportements normaux et une autre pour les comportements exceptionnels.

Supposons un langage impératif simple avec les structures de contrôle suivantes :

```
block ; block
throw value
try block catch binder → block
try block finally block
```

La sémantique opérationnelle de ces constructions décrit comment l'environnement S du programme évolue :

$$\frac{S_1 \vdash \mathsf{u} \mapsto S_2 + \mathrm{ok} \quad S_2 \vdash \mathsf{v} \mapsto S_3 + E}{S_1 \vdash \mathsf{u}\ ;\ \mathsf{v} \mapsto S_3 + E} \qquad \frac{S_1 \vdash \mathsf{u} \mapsto S_2 + \mathrm{exn}(e)}{S_1 \vdash \mathsf{u}\ ;\ \mathsf{v} \mapsto S_2 + \mathrm{exn}(e)}$$

$$\frac{}{S \vdash \mathbf{throw}\ \mathsf{e} \mapsto S + \mathrm{exn}(e)} \qquad \frac{S_1 \vdash \mathsf{u} \mapsto S_2 + \mathrm{ok}}{S_1 \vdash \mathbf{try}\ \mathsf{u}\ \mathbf{catch}\ \mathsf{x} \to \mathsf{v} \mapsto S_2 + \mathrm{ok}}$$

$$\frac{S_1 \vdash \mathsf{u} \mapsto S_2 + \mathrm{exn}(e) \quad S_2[x \leftarrow e] \vdash \mathsf{v} \mapsto S_3 + E}{S_1 \vdash \mathbf{try}\ \mathsf{u}\ \mathbf{catch}\ \mathsf{x} \to \mathsf{v} \mapsto S_3 + E}$$

$$\frac{S_1 \vdash \mathsf{u} \mapsto S_2 + \mathrm{ok} \quad S_2 \vdash \mathsf{v} \mapsto S_3 + E}{S_1 \vdash \mathbf{try}\ \mathsf{u}\ \mathbf{finally}\ \mathsf{v} \mapsto S_3 + E}$$

$$\frac{S_1 \vdash \mathsf{u} \mapsto S_2 + \mathrm{exn}(e) \quad S_2 \vdash \mathsf{v} \mapsto S_3 + \mathrm{ok}}{S_1 \vdash \mathbf{try}\ \mathsf{u}\ \mathbf{finally}\ \mathsf{v} \mapsto S_3 + \mathrm{exn}(e)}$$

Les triplets de Hoare deviennent alors $\{P\}\mathsf{block}\{Q; Q_e\}$, avec Q_e la postcondition décrivant l'état du programme quand une exception s'échappe de block. La propriété Q_e pourra utiliser error pour désigner la valeur transportée par l'exception, c'est-à-dire celle passée à throw puis plus tard liée à binder.

1. Comment spécifier un triplet pour s'assurer que block se termine toujours de façon normale ?
2. Donner les règles de déduction en logique de Hoare correspondant aux quatre structures de contrôles ci-dessus. Des règles simples et peu nombreuses sont préférables.

3. Soit $wp(\mathsf{statement}, Q)$ la fonction calculant la plus faible précondition menant à une postcondition Q pour une instruction isolée (une assignation par exemple). L'étendre en une fonction $wp(\mathsf{block}, Q, Q_e)$ et donner les règles de calcul correspondant aux structures de contrôle ci-dessus. Un calcul qui ne fait par exploser la taille des formules logiques est préférable.

Remarque : si la sémantique opérationnelle ne décrit pas une certaine situation, un programme qui se trouve dans cette situation a un comportement non défini. Les règles de déduction et le calcul de plus faible précondition ne doivent pas permettre de prouver la correction d'un tel programme.