Examen du cours Calcul des Constructions Inductives

J.-C. Filliâtre, H. Herbelin, C. Marché et C. Paulin

22 mars 2004

L'examen dure deux heures, les notes de cours sont autorisées.
Rédiger chaque exercice sur une feuille séparée.

1 Preuve de programme

Dans cet exercice, on s'intéresse à la preuve de programmes contenant des boucles for « à la Caml », c.-à-d. de la forme `for` $i = e_1$ `to` e_2 `do` e_3 `done` avec la sémantique suivante :

- e_1 et e_2 sont évalués en v_1 et v_2 une fois pour toutes ;
- si $v_1 > v_2$ on ne fait rien ;
- sinon, on évalue e_3 successivement pour i prenant les valeurs v_1, $v_1 + 1$, …, v_2.

Remarquons que i n'est visible que dans e_3, dans lequel il n'est pas modifiable.

On rappelle qu'en Coq, `Z` est le type des entiers relatifs, et on dispose d'une fonction `Z_of_nat` d'injection de `nat` dans `Z`. On suppose également donnée une fonction inverse `nat_of_Z`, qui envoie tous les négatifs sur 0.

1. Définir une fonction Coq, de type `Z->Z->Z`, équivalente au programme Caml suivant :

   ```
   let f a b =
     let d = ref 1 in
     for i=a to b do d := 19 * !d + i done;
     !d
   ```

   On utilisera obligatoirement une fonction auxiliaire définie par récurrence structurelle sur un `nat`.

2. Compléter les règles de logique de Hoare suivantes, en justifiant brièvement pourquoi celles-ci sont correctes.

$$\frac{}{\{(a > \ldots) \wedge \ldots\}\ \texttt{for}\ i = a\ \texttt{to}\ b\ \texttt{do}\ s\ \texttt{done}\ \{Q\}}$$

$$\frac{\{(\ldots \leq i \leq \ldots) \wedge I(i)\}\ s\ \{I(\ldots)\}}{\{a \leq \ldots \wedge I(\ldots)\}\ \texttt{for}\ i = a\ \texttt{to}\ b\ \texttt{do}\ s\ \texttt{done}\ \{I(\ldots)\}}$$

   Attention : dans ces règles, a et b sont des constantes entières, et non des expressions.

3. Pour prouver en Coq des programmes avec boucles for, on a besoin d'un principe de récurrence `for_rec` sur un intervalle $[a, b]$ d'entiers, de type

   ```
   forall (a b:Z), a <= b+1 ->
     forall (P : Z -> Set),
       P a -> (forall i, a <= i <= b -> P i -> P (i+1))
       -> P (b+1).
   ```

   On demande de construire une définition de `for_rec`. On pourra utiliser une fonction auxiliaire travaillant sur des `nat`, de manière analogue à la première question (et dans ce cas on aura soin de bien préciser le type de cette fonction). On s'autorisera à ne pas donner de preuves pour les propriétés purement logiques (i.e. de sorte `Prop`), on utilisera dans de tels cas la notation $(? : P)$ où P est le type attendu.

4. À l'aide de `for_rec`, donner une définition Coq d'une fonction `sqr` de type

   ```
   forall z:Z, z>=0 -> { s:Z | s=z*z }
   ```

   correspondant au programme Caml

   ```
   let sqr z =
     let s = ref 0 in
     for i=0 to z-1 do s := !s + 2*i + 1 done;
     !s
   ```

2 Égalité

L'égalité polymorphe de Leibniz est définie de manière inductive par :

Inductive eq (U:Type) (x:U) : U → Prop := refl : (eq x x)

Cette égalité est utilisée dans l'énoncé pour différentes instances de U, néanmoins ce paramètre reste implicite dans les notations. On écrira simplement $(\texttt{eq}\ x\ y)$ ou bien $(\texttt{refl}\ x)$ au lieu de $(\texttt{eq}\ U\ x\ y)$ et $(\texttt{refl}\ U\ x)$ avec U le type de x.

1. Donner le type du principe d'élimination `eq_ind` non dépendante engendré par COQ pour cette définition. Quelle règle de réduction satisfait cet opérateur ?
2. Donner le type du principe d'élimination **dépendante** pour la définition `eq` que l'on notera `eq_ind_dep`.
3. On note `eq_unique_prop` la propriété d'unicité des preuves d'égalité sur un type A définie par $\forall(x : A)(e : \texttt{eq}\ x\ x), (\texttt{eq}\ e\ (\texttt{refl}\ x))$
   Construire une définition du principe d'élimination dépendante `eq_ind_dep` en utilisant uniquement `eq_ind` et une preuve `eq_unique` du principe `eq_unique_prop`. On pourra soit donner le terme de preuve, soit donner une suite de tactiques élémentaires (apply, intro, exact) en indiquant la forme du but après chaque tactique.
4. On se propose de montrer le principe `eq_unique_prop` dans le cas particulier où A est le type `nat` des entiers.
   (a) On suppose que pour tout P : `nat` → Prop on peut construire une égalité `EQ` de type $\forall n\ m : \texttt{nat}, (P\ n) \to (P\ m) \to$ Prop qui vérifie les deux propriétés suivantes :
   – `EQ_refl` de type $\forall(n : \texttt{nat})(p : P\ n), \texttt{EQ}\ n\ n\ p\ p$
   – `EQ_eq` de type $\forall(n : \texttt{nat})(p\ q : P\ n), \texttt{EQ}\ n\ n\ p\ q \to \texttt{eq}\ p\ q$.
   On fixe n et on prend pour prédicat P, la propriété $\texttt{fun}\ m\ \Rightarrow \texttt{eq}\ n\ m$.
   – En utilisant `eq_ind_dep`, donner une preuve de la propriété :
   $\forall(m : \texttt{nat})(e : \texttt{eq}\ n\ m), (\texttt{EQ}\ n\ m\ (\texttt{refl}\ n)\ e)$
   – En déduire une preuve de la propriété : $\forall(e : \texttt{eq}\ n\ n), (\texttt{eq}\ (\texttt{refl}\ n)\ e)$.
   (b) On cherche maintenant à construire un prédicat `EQ` aproprié.
      i. Donner le terme de preuve `eq_S` de type $\forall n\ m, \texttt{eq}\ n\ m \to \texttt{eq}\ (S\ n)\ (S\ m)$. Montrer que $(\texttt{eq\_S}\ n\ n\ (\texttt{refl}\ n))$ est une preuve de $(\texttt{eq}\ (S\ n)\ (S\ n))$ qui se réduit en $(\texttt{refl}\ (S\ n))$.
      ii. Construire une preuve `eqn_dec` de décidabilité de l'égalité sur `nat` (c'est-à-dire de type $\forall n\ m, \{\texttt{eq}\ n\ m\} + \{\neg\texttt{eq}\ n\ m\}$) telle que de plus la propriété :
      $\forall n, \texttt{eq}\ (\texttt{eqn\_dec}\ n\ n)\ (\texttt{left}\ (\texttt{refl}\ n))$ soit prouvable.
      On supposera qu'il existe des preuves des propriétés suivantes :
      – `neq_O_S` de type $\forall n, \neg(\texttt{eq}\ O\ (S\ n))$,
      – `neq_S_O` de type $\forall n, \neg(\texttt{eq}\ (S\ n)\ O)$,
      – `neq_S` de type $\forall n\ m, \neg(\texttt{eq}\ n\ m) \to \neg(\texttt{eq}\ (S\ n)\ (S\ m))$
      On rappelle que le type $\{A\} + \{B\}$ est un type inductif à deux constructeurs `left` de type $A \to \{A\} + \{B\}$ et `right` de type $B \to \{A\} + \{B\}$.
      iii. On définit alors la propriété $(\texttt{EQ}\ n\ m\ p\ q)$ en commençant par tester $(\texttt{eq}\ n\ m)$. Si $(\texttt{eq}\ n\ m)$ est vérifié, on se sert de cette preuve pour transformer p de type $(P\ n)$ en un terme p' de type $(P\ m)$ et $(\texttt{EQ}\ n\ m\ p\ q)$ est alors défini comme la propriété $(\texttt{eq}\ p'\ q)$. Si $(\texttt{eq}\ n\ m)$ n'est pas vérifié, alors $(\texttt{EQ}\ n\ m\ p\ q)$ est défini comme la propriété `False`.

– Donner la définition de `EQ` dans COQ.
– Montrer que les propriétés attendues de `EQ` (cf question 4a) sont bien vérifiées pour cette définition.