TD 3 : Preuves de correction inductives

Matthieu Sozeau (matthieu.sozeau@inria.fr)

October 13, 2014

Ce TD porte sur la preuve de programmes spécifiés en logique du premier ordre. Les notes de cours et corrigés des TDs précédents sont disponibles sur la page du cours:

www.pps.univ-paris-diderot.fr/~sozeau/teaching/MVF-2014.fr.html

Des définition et propriétés utiles sont rappellées en annexe.

Ordres bien fondés

Exercice 1. *Donner une mesure sur les arbres binaires définis par les constructeurs:*

$$\begin{aligned} \mathit{Leaf} &: \mathit{Tree} \\ \mathit{Node} &: \mathit{Tree} \times \mathit{Nat} \times \mathit{Tree} \to \mathit{Tree} \end{aligned}$$

Exercice 2. *Donner un prédicat* NoDup *caractérisant une liste sans duplicats à base de l'opérateur At.*

Exercice 3. *Montrer* $\mathit{NoDup}(\grave{}) \Leftrightarrow \grave{} = [] \vee \exists a\ \grave{}', \grave{} = a \cdot \grave{}' \wedge \neg \mathit{Is\_in}(a; \grave{}') \wedge \mathit{NoDup}(\grave{}')$.

Dans la suite, on peut donc utiliser: $\mathit{NoDup}([])$ et $\mathit{NoDup}(a \cdot \grave{}') \Leftrightarrow \neg \mathit{Is\_in}(a; \grave{}') \wedge \mathit{NoDup}(\grave{}')$.

Exercice 4. *Définir une fonction* $\mathsf{remove\_onedup}$ *qui enlève d'une liste les éléments égaux à un élément donné.*

Exercice 5. *Montrer que la liste retournée par* $\mathsf{remove\_onedup}$ *est de taille inférieure ou égale à la taille de la liste en entrée.*

Exercice 6. *Donner une spécification pour* $\mathsf{remove\_onedup}$ *en termes du prédicat* $\mathit{Is\_in}$ *et montrez sa correction.*

Exercice 7. *Définir une fonction* $\mathsf{removedup}$ *qui prend une liste en entrée et renvoie cette liste sans duplicats. Justifiez sa terminaison.*

Exercice 8. *Donnez la spécification de* $\mathsf{removedup}$ *et prouvez sa correction.*

Une fonction de tri

On rappelle la spécification d'une fonction de tri:

$$Spec\_Sort(\grave{};\grave{}^{0}) = Ordered(\grave{}^{0}) \wedge Ms(\grave{}) = Ms(\grave{}^{0})$$

On veut faire la preuve de la fonction de tri suivante:

$$\begin{array}{rcl} MinSort([]) & = & [] \\ MinSort(a \cdot \grave{}) & = & \texttt{let}\ (m;\grave{}_m) = Extract\_min(a;\grave{})\ \texttt{in}\ m \cdot MinSort(\grave{}_m) \end{array}$$

La spécification d'$Extract\_min$ est:

$$\begin{array}{l} Spec\_Extract\_min(a;\grave{}_1;m;\grave{}_2) = \\ \quad Is\_in(m;a \cdot \grave{}_1)\ \wedge \\ \qquad\qquad \forall x \in ?:\ Is\_in(x;a \cdot \grave{}_1) \Rightarrow m \leq x\ \wedge \\ \qquad\qquad Ms(a \cdot \grave{}_1) = Sg(m) \uplus Ms(\grave{}_2) \end{array}$$

Exercice 9 (Minsort)**.** *Vérifiez la correction du programme de tri* $MinSort$ *en fonction de la spécification d'*$Extract\_min$*.*

Exercice 10 (Extractmin)**.** *Donnez une implémentation pour* $Extract\_min$ *et vérifiez sa correction.*

QuickSort – Devoir à rendre le 21 octobre

Pour le prochain cours, rédigez une preuve de la correction de la fonction $quicksort$ (preuve qu'elle respecte bien la spécification d'une fonction de tri):

$$\begin{array}{rcl} qsort([]) & = & [] \\ qsort(a \cdot \grave{}) & = & \texttt{let}\ (\grave{}_1;\grave{}_2) = \texttt{split}\ (a;\grave{})\ \texttt{in} \\ & & \qquad qsort(\grave{}_1)@(a \cdot qsort(\grave{}_2)) \end{array}$$

Vous aurez besoin de donner un type, une définition et une preuve de correction de la fonction $split$ qui prend un élément x, une liste l et renvoie deux listes contenant respectivement les éléments inférieur ou égaux à x et les éléments supérieurs à x.

Annexe: définitions

Ordres

La relation $\leq$ sur les entiers est réflexive et transitive. La relation $<$ sur les entiers irréflexive et transitive. Pour ces deux relations, on suppose les lemmes usuels en relation avec l'addition et la multiplication.

Prédicat *Ordered* : $List[*] \rightarrow prop$:

$$Ordered(\grave{}) = \forall i; j; \in Nat:\ (i < j < |\grave{}| \Rightarrow \grave{}[i] \leq \grave{}[j])$$

Multiensembles

- $\emptyset = \quad x \in ?: 0$
- $Sg(a) = \quad x \in ?: \texttt{if } x = a \texttt{ then } 1 \texttt{ else } 0$
- $M_1 \uplus M_2 = \quad x \in ?: M_1(x) + M_2(x)$

Propriétés:

- Neutral element: $\emptyset \uplus M = M \uplus \emptyset = M$
- Commutativity: $M_1 \uplus M_2 = M_2 \uplus M_1$
- Associativity: $M_1 \uplus (M_2 \uplus M_3) = (M_1 \uplus M_2) \uplus M_3$

Fonction Ms:

$$\begin{array}{rcl} Ms & : & List[*] \to Multiset[*] \\ Ms([]) & = & \emptyset \\ Ms(a \cdot `) & = & Sg(a) \uplus Ms(`) \end{array}$$

Propriétés:

- $Ms(`_1 @ `_2) = Ms(`_2 @ `_1) = Ms(`_1) \uplus Ms(`_2)$
- $Ms(Rev(`)) = Ms(`)$

Fonction Is_in:

$$\begin{array}{rcl} Is\_in & : & ? \times List[?] \to Bool \\ Is\_in(a; `) & = & Ms(`)(a) > 0 \end{array}$$

Propriétés:

- $\neg\ (Is\_in(x; []))$
- $\forall x\ l; Is\_in(x; a \cdot `) \Leftrightarrow (x = a \vee Is\_in(x; `))$
- $Is\_in(a; `) \Leftrightarrow \exists i < |`|; `[i] = a$
- $\forall l\ l'; Ms(l) = Ms(l') \Rightarrow \forall a; (Is\_in(a; l) \Leftrightarrow Is\_in(a; l'))$
- $\forall i < |`|; Is\_in(`[i]; `)$

Equivalences logiques

- $\wedge$ est associatif, commutatif, $\top$ est son élément neutre et $\bot$ absorbant (par exemple $p \wedge \bot = \bot$).
- $\vee$ est associatif, commutatif, $\bot$ est son élément neutre et $\top$ absorbant (par exemple $p \vee \top = \top$).
- $a \Rightarrow b$ est équivalent à $\neg a \vee b$.
- $a \Leftrightarrow b$ est équivalent à $(a \Rightarrow b) \wedge (b \Rightarrow a)$.
- $\neg p$ est équivalent à $p \Rightarrow \bot$.
- $\neg\top \Leftrightarrow \bot$ et $\neg\bot \Leftrightarrow \top$.
- On a les lois de De Morgan:
  - $\neg(a \wedge b) \Leftrightarrow \neg a \vee \neg b$.
  - $\neg(a \vee b) \Leftrightarrow \neg a \wedge \neg b$.
  - $\neg(\exists x; p) \Leftrightarrow \forall x; \neg p$.
  - $\neg(\forall x; p) \Leftrightarrow \exists x; \neg p$.