TD 5 : Preuves de terminaison en logique de Hoare

Matthieu Sozeau (matthieu.sozeau@inria.fr)

October 27, 2014

Ce TD porte sur la preuve de terminaison de programmes impératifs. 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

On rappelle que les règles d'inférence de la logique de Hoare *pour la correction totale* sont données par :

$$\{P\}\ \texttt{skip}\ \{P\} \qquad \{P[exp/x]\}\ x\ :=\ exp\ \{P\}$$

$$\frac{\{P\}\ C\ \{Q\} \qquad \{Q\}\ D\ \{R\}}{\{P\}\ C; D\ \{R\}}$$

$$\frac{\{E\ \wedge\ P\}\ C\ \{Q\} \qquad \{\neg E\ \wedge\ P\}\ D\ \{Q\}}{\{P\}\ \texttt{if}\ E\ \texttt{then}\ C\ \texttt{else}\ D\ \{Q\}}$$

$$\frac{\rho : X \to T \qquad (T, \prec) \text{ is a WFS} \qquad \{E\ \wedge\ I \wedge \rho = e\}\ C\ \{I \wedge \rho \prec e\}}{\{I\}\ \texttt{while}\ E\ \texttt{do}\ C\ \texttt{done}\ \{\neg E\ \wedge\ I\}}$$

$$\frac{P'\ \Rightarrow\ P \qquad \{P\}\ C\ \{Q\} \qquad Q\ \Rightarrow\ Q'}{\{P'\}\ C\ \{Q'\}}$$

(où C, D désignent des commandes, E, V des expressions sans effets de bord, ρ un fonction de mesure, $\prec$ un ordre bien fondé sur T).

Exercice 1. *Trouvez l'invariant et le variant permettant de montrer la terminaison de:*

{??} while (n $\geq$ 0) do n:=n-1 done {n = 0}

Solution:

```
e:nat;
{n >= 0}
while (n > 0) {n = e} do
  { n > 0 /\ n = e }
  { e > 0 /\ n - 1 = e - 1 }
  n:=n-1
  { e > 0 /\ n = e - 1 }
  { n < e }
done
{~ n > 0}
{n = 0}
```

Exercice 2. *Donner un ordre permettant de montrer la terminaison de la fonction suivante, donner sa speci cation et faire sa preuve de correction totale:*

```
log10(z : Int) : Nat :=
  l := 0;
  while z <> 0 do
    if -10 < z < 10 then
      z := 0
    else (
      l := l + 1;
      z := z / 10;
    )
  done;
  assert(?)
```

Solution:
On utilise l'order ¡ sur les entiers naturels et la fonction —x— comme mesure.

```
log10(z : Int) : Nat :=
  E : nat;
  l := 0;
  while z != 0 do
  { z != 0 /\ |z| = E }
  { E != 0 /\ |z| = E }
    { |z| <= l * 10 =< |z|+1 /\ E != 0 /\ |z| = E }
    if -10 < z < 10 then
     { |0| <= l * 10 =< |0|+1 /\
       -10 < 0 < 10 /\ E != 0 /\ 0 = 0 }
      z := 0
     { |z| <= l * 10 =< |z|+1 /\
       -10 < z < 10 /\ E != z /\ z = 0 }
     { |z| <= l * 10 =< |z|+1 /\
       -10 < z < 10 /\ |z| < E }
    else (
      { |z / 10| <= l * 10 + 10 =< |z / 10|+1 /\
```

```
      not (-10 < z / 10 < 10) /\ |z| = E /\ E != 0 }
     { |z / 10| <= (l + 1 * 10) =< |z / 10|+1 /\
      not (-10 < z / 10 < 10) /\ |z| = E /\ E != 0 }
     l := l + 1;
     { |z / 10| <= l * 10 =< |z / 10|+1 /\
      not (-10 < z / 10 < 10) /\ |z / 10| < E}
     z := z / 10;
     { |z| <= l * 10 =< |z|+1 /\
      not (-10 < z < 10) /\ |z| < E}
    )
    { |z| <= l * 10 =< |z|+1 /\ |z| < E}
  done;
  assert(|z| <= l * 10 =< |z|+1)
```

Exercice 3 (Division euclidienne)**.** *Trouvez le variant et annotez la boucle pour montrer la correction totale de la division euclidienne:*

```
assume(x >= 0 /\ y > 0);
a := 0;
b := x;
while b >= y do
  b := b - y;
  a := a + 1;
done;
assert(?)
```

Exercice 4 (Renversement itératif)**.** *Donnez le variant, l'ordre et l'annotation de la boucle pour le renversement iteratif d'une liste et montrez que le triplet correspondant au corps de la boucle.*

$\rho : List[\star]$;

irev $(\ell : List[\star]) =$
    assume($true$);
    $\ell' : List[\star]$;
    $\rho := []$;   *% ρ is the reverse of the treated pre x of ℓ*
    $\ell' := \ell$;   *% ℓ' is the non-treated su x of ℓ*
    while $\ell' \neq []$ do
        invariant?
        $\rho := head(\ell') \cdot \rho$;
        $\ell' := tail(\ell')$;
    assert($\rho = Rev(\ell)$)

Solution

$\rho : List[\star]$;

irev $(\ell : List[\star]) =$

assume($true$);

$\ell' : List[\star]$;

$\{\ell = \ell\}$

$\{\ell = rev([])@\ell\}$

$\rho := []$; % ρ is the reverse of the treated prefix of ℓ

$\{\ell = rev(\rho)@\ell\}$

$\ell' := \ell$; % ℓ' is the non-treated suffix of ℓ

$\{\ell = rev(\rho)@\ell'\}$

while $\ell' \neq []$ do

invariant $\ell = rev(\rho)@\ell'$

$\{\ell = rev(\rho)@\ell' \wedge \ell' \neq []\}$

$\{\ell = rev(\rho)@head(\ell') \cdot tail(\ell') \Leftrightarrow \ell = rev(head(\ell') \cdot \rho)@tail(\ell')\}$

$\rho := head(\ell') \cdot \rho$;

$\{\ell = rev(\rho)@tail(\ell')\}$

$\ell' := tail(\ell')$;

$\{\ell = rev(\rho)@\ell'\}$

$\{\ell = rev(\rho)@\ell' \wedge \ell' = []\}$

$\{\rho = rev(\ell))\}$

assert($\rho = rev(\ell)$)

Exercice 5 (Merge)**.** *Que calcule la fonction suivante, etant donnee deux listes ordonnees en entree?*

1. *Donnez sa spe cation.*
2. *Trouvez l'invariant et le variant pour la boucle.*
3. *Montrez la correction totale de l'algorithme.*

```
merge(l l' : List[Nat]) : List[Nat] :=
  r := [];

  while l != [] && l' != [] do
    if l = [] then { r := r @ l'; l' := [] }
    else if l' = [] then { r := r @ l; l := [] }
    else
    if head l < head l' then
      { r := r @ [head l]; l := tail l }
    else
      { r := r @ [head l']; l' := tail l' }
  done;
assert(?)
```