Projet Qualité 2016

H 4115

Chef projet : Loïc CHAMPAVERE
Yassir BOUIRY
Cyril CANETE
Ayoub CHIOUKH
Romain DUTEIL
Daniela MITRICA
Victor NOUVELLET

I. Problème:

Dériver un multiprogramme qui permette à des drones voulant accéder à une même position au prochain pas de temps d'élire celui d'entre eux qui sera prioritaire. Le processus P.i (associé au drone i) possède la variable booléenne y.i qu'il est seul à pouvoir modifier. Chaque Pi affecte une valeur à cette variable. Le problème est de synchroniser les processus pour qu'à leur terminaison la post-condition suivante soit établie :

$$R: (\# j :: y.j) = 1$$

II. *Solution:*

On reformule la post condition précédente sous une forme plus exploitable. À la fin du multiprogramme il existe un processus élu et il n'y a on a pas 2 ou plus :

$$R : (\exists j :: y.j) \wedge (\forall i, j :: y.i \wedge y.j \Rightarrow i=j)$$

On admet que B.i est une fonctionne booléenne qui revoit le statut du drone tel que : ∀ i, B.i ≡ y.i. D'où la nouvelle post-condition :

$$R : (\exists j :: y.j) \wedge (\forall i, j :: B.i \wedge B.j \Rightarrow i=j)$$

Établissons B.i :

Par transitivité de l'égalité on a : $\forall i, j :: i=\alpha \wedge \alpha=j \Rightarrow i=j$. On pose donc : $B.i \triangleq i=\alpha$.
On vérifie que la post-condition élue est unique :

$$\begin{gathered} \exists j :: y.j \wedge \exists i :: i \neq j \text{ alors :} \\ (y.i \equiv (i=\alpha)) \Rightarrow \neg y.i \\ \equiv \\ (\neg y.i \equiv (i \neq \alpha)) \Rightarrow \neg y.i \\ \equiv \\ (y.i \equiv (i \neq \alpha)) \Rightarrow \neg y.i \\ \equiv \\ \neg y.i \vee (i \neq \alpha) \end{gathered}$$

On a le triplet de Hoare: { wlp } S { R }.
On reformule la post-condition: $R : (\exists j :: y.j) \wedge (\forall i, y.j \equiv (i=\alpha)$.
Pour satisfaire la première clause de la post-condition, on affecte i à α, d'où :

$$\begin{aligned} S:\ & \alpha := i \\ & ; y.i := (\alpha=i) \end{aligned}$$

Avec $y.i := (\alpha=i)$ on s'assure de la correction locale. Le programme n'est pas juste globalement. Lors de l'affectation de y.i on ne connaît plus la valeur de α et on est susceptible d'avoir plusieurs processus élus. Une manière de solutionner le problème c'est d'affecter y.i à la fin de toutes les modifications de α. On introduit un compteur (c) que chaque processus incrémente après l'affectation de α . D'où :

```
Initialisation :
        var c : int
        ; c=o
S :
        α := i
        ; c := c+ 1
          … // attente
```

On s'assure que tous les processus ont incrémenté le compteur:

if (c == nbProcessus) -> skip fi .

D'où :

```
S :
        α := i
        ; c := c+ 1
        if c == nbProcessus -> skip fi
{R1}:
        y.i := (α=i)

{R}
```

Le programme est juste mais il n'est pas optimal, chaque processus doit attendre que tous les autres processus ait incrémenté le compteur. Nous modifions le programme pour ne plus attendre à partir du moment où α a été modifié par un autre processus :

```
Initialisation :
        var c : int
        ; c=o
S :
        α := i
        ; c := c+ 1
        if ((c == nbProcessus) ⋁ ( i≠α )) -> skip fi
{R1}:
        y.i := (α=i)
{R}
```

La condition d'attente n'est plus atomique. Nous vérifions que le programme est correct globalement :