Projet Qualité 2016

### H 4115

Chef projet : Loïc CHAMPAVERE

Yassir BOUIRY

Cyril CANETE

Ayoub CHIOUKH

Romain DUTEIL

Daniela MITRICA

Victor NOUVELLET

1. 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

1. *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 : ( ∋ j :: y.j ) ∧ ( ∀ i, j :: y.i ∧ y.j ⇒ 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 : ( ∋ j :: y.j ) ∧ ( ∀ i, j :: B.i ∧ B.j ⇒ i=j )

Établissons B.i :

Par transitivité de l’égalité on a : ∀ i, j :: i=α ∧ α=j ⇒ i=j . On pose donc : B.i ≙ i=α .

On vérifie que la post-condition élue est unique :

∋ j :: y.j ∧ ∋ i :: i≠j alors :

( y.i ≡ (i=α)) ⇒ ­ ­┐y.i

≡

( ┐y.i ≡ ( i≠α )) ⇒ ­ ­┐y.i

≡

( y.i ≡ (i≠α)) ⇒ ­ ­┐y.i

≡

┐y.i ⋁ (i≠α)­ ­

On a le triplet de Hoare: { wlp } S { R }.

On reformule la post-condition: R : ( ∋ j :: y.j ) ∧ ( ∀ i, y.j ≡ (i=α).

Pour satisfaire la première clause de la post-condition, on affecte i à α, d’où :

S: α := i

; y.i := (α=i)

Avec y.i := ( α=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 :