Projet Qualité 2016

### H 4115

Chef projet : Victor NOUVELLET

Yassir BOUIRY

Cyril CANETE

Loïc CHAMPAVERE

Ayoub CHIOUKH

Romain DUTEIL

Daniela MITRICA

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:*

Remarque : Afin de respecter l’équité entre les drones, le programme global est symétrique pour tous les processus (chaque processus a le même programme).

On reformule la post condition précédente sous une forme plus exploitable. À la fin du multiprogramme il existe un seul et unique processus élu :

R : ( ∋ j :: y.j ) ∧ ( ∀ i, j :: y.i ∧ y.j ⇒ i=j )

Dans un premier temps, on considère que { ?y.i ≡ B.i} comme vrai globalement. D’où la nouvelle post-condition :

R : ( ∋ j :: B.j ) ∧ ( ∀i, j :: B.i ∧ B.j ⇒ i=j )

Posons α une variable partagée entre tous les processus tel que S : α := i.

Or, par transitivité de l’égalité on a : ∀ i, j :: i=α ∧ α=j ⇒ i=j. Définissons alors : B.i ≙ (i=α).

Vérifions que y.i ≡ (i= α) ne sera pas rendu faux par au autre processus. Autrement dit, que ce soit vrai globalement) :

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

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

y.i := (α=i)

Avec y.i := ( α=i ) on s’assure de la correction locale.

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 d’apporter une solution au problème est d’affecter y.i après la modification de α par tous les processus en lice. Afin de gérer la synchronisation entre les processus, on introduit, avant l’affectation de y.i, un compteur (c) que chaque processus incrémente après l’affectation de α. D’où :

Initialisation (unique avant la création des processus) :

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 aient incrémenté le compteur (c). Affaiblissons la condition. Pour cela, nous pouvons alors modifier le programme pour ne plus attendre à partir du moment où α a été modifié par un autre processus :

Initialisation (unique avant la création des processus) :

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 pas atomique mais une falsification de nbProcessus dans la condition (c==nbProcessus) suite à une interruption n’empêchera pas le bon fonctionnement du programme. En effet, la condition ( i≠α ) sera alors vrai. Idem pour la falsification de α dans la condition (i≠α) qui rendra alors le processus non élu.

Nous avons donc un programme correct par construction.