Dérivation d'un multi-programme

H4311

18 avril 2016

1 Spécification formelle du programme

Dériver un multi-programme 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 $P.i$ affecte une valeur à cette variable. Le problème est de synchroniser les processus pour qu'à leur terminaison la post-condition suivante soit établie :

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

2 Dérivation

L'expression de R qui nous est donnée dans la spécification formelle du programme est difficilement exploitable telle quelle. On peut la reformuler sous la forme suivante :

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

La partie gauche de cette conjonction signifie qu'il existe au moins un j tel que $y.j$ soit vrai, alors que la partie droite siginifie qu'il en existe au plus un. Cette expression est donc cohérente avec ce que que l'on cherche, c'est à dire obtenir un unique j pour lequel $y.j = 1$.
On sait qu'à un certain moment du programme, on devra affecter une certaine valeur à $y.i$. On peut donc donner une première forme du programme :

Algorithm 1 Programme $P.i$

procedure P.I
$y.i \leftarrow B.i$
$\{?y.i \equiv B.i\}$

Nous considérons ici un multi-programme. Il est donc possible qu'après l'execution de $y.i := B.i$ un autre programme vienne changer la valeur de B.i, l'assertion $y.i \equiv B.i$ serait falsifiée. Cette assertion, de part l'affectation $y.i := B.i$ est localement vraie, mais pas forcèment globalement vraie.
Dans un premier temps, supposons que $y.i \equiv B.i$ soit globalement vraie dans le programme. C'est pour l'instant une hypothèse, dont on discutera de la validité par la suite.
On a alors :

$$wlp.(y.i := B.i).(R)$$

$$= \{definitionde :=\}$$

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

On veut obtenir cette post-condition à la fin de l'execution de l'ensemble des programmes, on veut donc que cette conjonction soit vraie. Si l'on étudie la deuxième partie de cette conjonction, c'est à dire $(\forall i, j :: B.i \wedge B.j \Rightarrow i = j)$, on aurait plusieurs possibilités pour la rendre vraie :

— On pourrait essayer de chercher $B.i$ et $B.j$ tels que $B.i \wedge B.j$ soit tout le temps faux (car, trivialement, $\neg(B.i \wedge B.j) \Rightarrow (B.i \wedge B.j \Rightarrow i = j)$). Cependant, nous n'avons ici aucune information sur les B.i.
— On peut se servir de la propriété de transitivité de l'égalité, c'est à dire $(a = b) \wedge (b = c) \Rightarrow (a = c)$. En posant $B.i$ de la forme $i = \alpha$, on aurait alors $(\forall i, j :: ((i = \alpha) \wedge (j = \alpha)) \Rightarrow (i = j))$ qui est toujours vrai.

Le programme aurait alors la forme suivante :

Algorithm 2 Programme $P.i$

procedure P.I
$y.i \leftarrow (i = \alpha)$
$\{?y.i \equiv (i = \alpha)\}$

α serait donc une variable partagée par l'ensemble des programmes. Pour atteindre la post-condition R, on veut rendre $y.i$ vrai, pour cela on peut simplement faire l'affectation $\alpha := i$ en début du programme $P.i$

Algorithm 3 Programme $P.i$

procedure P.I
$\alpha \leftarrow i$
$y.i \leftarrow (i = \alpha)$
$\{?y.i \equiv (i = \alpha)\}$

On a précédemment fais l'hypothèse que $y.i \equiv B.i$ était vrai globalement après avoir fait l'affectation de $y.i$ dans le programme $P.i$. On doit maintenant prouver que cette hypothèse est vérifiée. Lorsqu'on regarde la forme des programmes, on observe que seule l'instruction $\alpha := j$, réalisée par un autre programme $P.j$ différent de $P.i$, peut rendre $y.i \equiv B.i$ faux, car la valeur de la variable partagée α est changée.

On peut alors rechercher la précondition (libérale) la plus faible wlp qui, au travers de l'instruction $\alpha := j$, établira la post-condition $y.i \equiv B.i$. En effet si l'on arrive par la suite à vérifier une contrainte P plus forte que cette précondition la plus faible, alors on pourra affimer que $y.i \equiv (i = \alpha)$ est vraie globalement.

Soit $P \equiv (y.i \equiv (i = \alpha))$. On veut $P \Rightarrow wlp.(\alpha := j).P$. On a alors :

$$wlp.(\alpha := j).P$$

$$= \{definition\ de :=\}$$

$$y.i \equiv (i = j)$$
$$= (y.i \equiv Faux)$$
$$= \neg y.i$$

En réécrivant l'implication $P \Rightarrow wp.(\alpha := j).P$:

$$P \Rightarrow \neg y.i$$
$$= (y.i \equiv (i = \alpha)) \Rightarrow \neg y.i$$
$$= \neg(y.i \equiv (i = \alpha)) \vee \neg y.i$$
$$= \{\neg(a \equiv b) \equiv (\neg a \equiv b)\}$$
$$\neg y.i \equiv (i = \alpha) \vee \neg y.i$$

On obtient l'expression $\neg y.i \equiv (i = \alpha) \vee \neg y.i$. Étudions la table de vérité d'une expression de la forme $a \equiv b \vee a$:

a	b	$a \vee b$	$a \equiv b \vee a$	$b \Rightarrow a$
0	0	0	1	1
0	1	1	0	0
1	0	1	1	1
1	1	1	1	1

D'après la table de vérité on peut déduire que :

$$(\neg y.i \equiv (i = \alpha) \vee \neg y.i) = ((i \neq \alpha) \vee \neg y.i)$$

Cette contrainte $((i \neq \alpha) \vee \neg y.i)$ permet bien d'assurer que $y.i \equiv (i = \alpha)$ est vraie globalement. Cependant elle implique aussi que $y.i$ ne sera jamais vrai. En effet si $i \neq \alpha$ est vrai, alors l'affectation $y.i = (i = \alpha)$ rend $y.i$ faux. Et si $\neg y.i$ est vérifié alors trivialement $y.i$ est faux. Si $y.i$ n'est jamais vrai, alors la post condition du multi-programme ne sera jamais atteinte.

NB : On trouve ce résultat car on ne s'occupe que de la partie droite de R, celle qui définit qu'il y a au plus un i pour lequel $y.i$ est $Vrai$, sans se préoccuper de la partie gauche (qui elle fait en sorte qu'il y ait au moins un $y.i$ vrai).

Nous alons donc devoir contraindre davantage le système. Posons C.i une contrainte vérifiée à l'endroit du programme ou l'on veut s'assurer que $y.i \equiv (i = \alpha)$ soit vraie globalement et D.j la précondition pour qu'un autre programme, différent de $P.i$, réalise l'affectation $\alpha := j$. Les programmes $P.i$ et $P.j$ ont alors la forme suivante :

Algorithm 4 Programme $P.i$

procedure P.I
$\alpha \leftarrow i$
$y.i \leftarrow (i = \alpha)$
$\{C.i\}\{?y.i \equiv (i = \alpha)\}$

Algorithm 5 Programme $P.j$

procedure P.J
$\{D.j\}\alpha \leftarrow j$
$y.j \leftarrow (j = \alpha)$
$\{?y.j \equiv (j = \alpha)\}$

NB : Les programmes sont tous symétriques, c'est à dire que le programme $P.i$ contiendra aussi un $D.i$ et le programme $P.j$ contiendra un $C.j$. Afin de rendre plus compréhensible la réflexion, on ne notera que $C.i$ et $D.j$. On a alors :

$$\forall j : j \neq i : P \wedge C.i \wedge D.j \Rightarrow wlp.(\alpha := j).P$$
$$= \{(a \wedge b \Rightarrow c) \equiv (a \Rightarrow (b \Rightarrow c))\}$$
$$\forall j : j \neq i : C.i \wedge D.j \Rightarrow P \wedge wlp.(\alpha := j).P$$
$$= \{calcul\ precedent\}$$
$$= \forall j : j \neq i : C.i \wedge D.j \Rightarrow ((i \neq \alpha) \vee \neg y.i)$$
$$= \{(a \wedge b \Rightarrow c) \equiv (a \Rightarrow (b \Rightarrow c))\}$$
$$C.i \Rightarrow \forall j : j \neq i : \neg D.j \vee i \neq \alpha \vee \neg y.i$$

On peut enlever $\neg y.i$ dans cette expression, car celui-ci est redondant avec $i \neq \alpha$. En effet, suite à l'affectation $y := (i = \alpha)$, si $i \neq \alpha$, on a $\neg y.i$. On essaie donc avec :

$$C.i \equiv \forall j : j \neq i : \neg D.j \vee i \neq \alpha$$

Le programme aurait alors à présent la forme suivante :

Algorithm 6 Programme $P.i$

procedure P.I
$\{D.i\}\alpha \leftarrow i$
if $\forall j : j \neq i : \neg D.j \vee i \neq \alpha$ **then**
skip
$y.i \leftarrow (i = \alpha)$
$\{?y.i \equiv (i = \alpha)\}$

On doit bien faire la vérification sur C.i avant de faire l'affectation $y.i \leftarrow (i = \alpha)$ car cette affectation est de toute manière la dernière instruction du programme : on est pas interéssé par ce qui ce passe après avoir affecté la valeur à $y.i$.
On a donc ici un if bloquant. La vérification de C.i n'est clairement pas une opération atomique, et c'est potentiellement un problème.
Avant de le traiter, interessons nous aux $D.j$. La condition du if est $\forall j : j \neq i : \neg D.j \vee i \neq \alpha$. Si le if est débloqué car le membre de droite de la disjonction est vrai, c'est à dire que $i \neq \alpha$, alors lors de l'affectation de $y.i$, la valeur affectée sera $Faux$. Rendre vrai le membre de gauche de la disjonction, c'est à dire rendre $\neg D.j$ vrai pour chaque $P.j$ différent de $P.i$, est donc le seul moyen d'affecter la valeur $Vrai$ à $y.i$. On rentrera dans ce cas si on a pas $\alpha \neq i$, c'est-à-dire si tous les autres programmes ont fait leur affectation de α. On comprend bien ici que $D.j$ doit devenir une variable, qui doit être vraie pour chaque programme $P.j$ tant que ce $P.j$ n'a pas

réalisé l'affectation de α.

Algorithm 7 Programme $P.i$

procedure P.I
$\alpha \leftarrow i$
$D.i \leftarrow False$
if $\forall j : j \neq i : \neg D.j \vee (i \neq \alpha)$ **then**
skip
$y.j \leftarrow (i = \alpha)$

On supposera dans cet algorithme que $D.i$ a été préalablement initialisé avec la valeur $Vrai$. On peut maintenant discuter de l'atomicité de la vérification de la condition du if, qui est $\forall j : j \neq i : \neg D.j \vee (i \neq \alpha)$. Cette opération de vérification n'est clairement pas atomique, et il paraît compliqué de faire en sorte qu'elle le soit. Considérons le raisonnement suivant :

— Si on a évalué $i \neq \alpha$ Vrai et que le programme perd la main : cette assertion restera toujours vraie car le programme $P.i$ que l'on considère ne peut plus changer la valeur de α afin de faire $\alpha := i$, et c'est le seul programme qui aurait pu le faire car tout les indices de programme sont différents.
— Si l'on a évalué l'un des $D.j$ à $Faux$ et que le programme perd la main : une fois que la valeur d'un $D.j$ a changé de $Vrai$ à $Faux$, elle ne peut plus être changée. Donc le $D.j$ évalué $Faux$ restera $Faux$, et donc $\neg D.j$ restera $Vrai$.

La non-atomicité de la condition du if n'est donc pas un problème en réalité, car les différentes propositions qui la composent ne peuvent changer de valeur qu'à un seul moment de l'ensemble du multi-programme, et ne peuvent plus changer de valeur par la suite.

NB : On peut remarquer que le programme qui va mettre $y.i$ à $Vrai$ est celui qui s'exécutera en dernier.

La condition $\forall j : j \neq i : \neg D.j \vee (i \neq \alpha)$ correspond en réalité à une suite de if bloquant. Si on devait implémenter ce code, il serait certainement plus élégant de remplacer cette suite de vérifications par une seule variable, un compteur qui compte le nombre d'affectations de α. Notons qu'en faisant cela, on perd le fait de savoir quel programme a effectué l'affectation de α, ce qui n'est pas un problème car cette information nous est inutile ici. Appelons c le compteur et n le nombre de programmes tournant en parallèle, avec en précondition supplémentaire de notre multiprogramme :

$$c = 0$$

Le programme devient alors :

Algorithm 8 Programme $P.i$

procedure P.I
$\alpha \leftarrow i$
$c \leftarrow c + 1$
if $c = n \vee i \neq \alpha$ **then**
skip
$y.j \leftarrow (i = \alpha)$