Cours Algorithmique des systèmes parallèles et distribués
Exercices
Série 3 : Exclusion Mutuelle - Dates - Estampilles
par Alessio Coltellacci et Dominique Méry
3 mars 2025

Exercice 1 *(dis-slidingwindow)*

Question 1.1 *Le protocole appelé Sliding Window Protocol est fondé sur un fenêtre qui glisse pour valider progressivement les envois reçus.*
Le protocole est donné sous la forme d'invariant avec des événements. Proposer un schéma de traduction pour cet algorithme réparti en un module TLA+.

EVENT e
ANY
x
WHERE
$G(x,u)$
THEN
$u := f(x,u)$
END

$\rightsquigarrow$

$$BA(e) \stackrel{def}{=} \exists x. \left(\begin{array}{l} \wedge\ G(x,u) \\ \wedge\ u' = f(x,u) \end{array} \right)$$

Question 1.2 *Proposer un schéma de traduction pour un algorithme réparti en TLA$^+$.*

Question 1.3 *Reprendre la solution précédente pour modéliser chan comme un buffer de taille la taille de la fenêtre.*

AXIOMS
$axm1 : n \in \mathbb{N}_1$
$axm2 : IN \in \mathbb{N} \pfun D$
$axm3 : dom(IN) = 0 \upto n$
$axm4 : l \in \mathbb{N}$
$axm5 : l \leq n$

VARIABLES OUT, i, ack, got, b

INVARIANTS

$inv1 : OUT \subseteq IN$

$inv2 : 0 \mathinner{..} i{-}1 \lhd OUT = 0 \mathinner{..} i{-}1 \lhd IN$

$inv3 : i \in 0 \mathinner{..} n{+}1$

$inv4 : ack \cup got \subseteq i \mathinner{..} i{+}l \cap 0 \mathinner{..} n$

$inv5 : ack \subseteq dom(OUT)$

$inv1 : OUT \in \mathbb{N} \pfun D$

$inv2 : i \in 0 \mathinner{..} n{+}1$

$inv3 : 0 \mathinner{..} i{-}1 \subseteq dom(OUT) \land dom(OUT) \subseteq 0 \mathinner{..} n$

$inv8 : ack \subseteq \mathbb{N}$

$inv10 : got \subseteq \mathbb{N}$

$inv13 : got \subseteq dom(OUT)$

$inv14 : ack \subseteq dom(OUT)$

$inv16 : 0 \mathinner{..} i{-}1 \lhd OUT = 0 \mathinner{..} i{-}1 \lhd IN$

EVENT INITIALISATION

BEGIN

$act1 : OUT := \varnothing$

$act2 : i := 0$

$act5 : ack := \varnothing$

$act6 : got := \varnothing$

$act8 : b := \varnothing$

END

EVENT send

ANY

j

WHERE

$grd1 : j \in i \mathinner{..} i{+}l$

$grd2 : j \leq n$

$grd3 : j \notin got$

$grd4 : j{-}i \in 0 \mathinner{..} l$

THEN

$act3 : b(j{-}i) := IN(j)$

END

EVENT receive

ANY

j

WHERE

$grd2 : j \in i \mathinner{..} i{+}l$

$grd3 : j{-}i \in dom(b)$

THEN

$act2 : ack := ack \cup \{j\}$

$act3 : OUT(j) := b(j{-}i)$

END

EVENT receiveack
ANY
k
WHERE
$grd1 : k \in ack$
THEN
$act1 : got := got \cup \{k\}$
$act2 : ack := ack \setminus \{k\}$
END

EVENT sliding
ANY
c
WHERE
$grd1 : got \neq \varnothing$
$grd3 : i \in got$
$grd4 : i{+}l < n$
$$grd5 : \left(\begin{array}{l} c \in 0 \mathinner{..} l \pfun D \\ \land dom(c) = \{u \mid u \in 0 \mathinner{..} l{-}1 \land u{+}1 \in dom(b)\} \\ \land (\forall o \cdot o \in dom(b) \land o \neq 0 \Rightarrow o{-}1 \in dom(c) \land c(o{-}1) = b(o)) \end{array} \right)$$
THEN
$act1 : i := i{+}1$
$act2 : got := got \setminus \{i\}$
$act3 : ack := ack \setminus \{i\}$
$act5 : b := c$
END

EVENT emptywindow
ANY
c
WHERE
$grd1 : got \neq \varnothing$
$grd2 : i \in got$
$grd3 : i{+}l \geq n$
$grd4 : i \leq n$
$$grd5 : \left(\begin{array}{l} c \in 0 \mathinner{..} l \pfun D \\ \land dom(c) = \{u | u \in 0 \mathinner{..} l{-}1 \land u{+}1 \in dom(b)\}) \\ \land (\forall o \cdot o \in dom(b) \land o \neq 0 \Rightarrow o{-}1 \in dom(c) \land c(o{-}1) = b(o)) \end{array} \right)$$
THEN
$act1 : i := i{+}1$
$act2 : got := got \setminus \{i\}$
$act3 : ack := ack \setminus \{i\}$
$act5 : b := c$
END

EVENT completion
WHEN
$grd1 : i = n{+}1 \land got = \varnothing$
THEN
skip
END

EVENT loosingchan
ANY
j
WHERE
$grd1 : j \in i \mathinner{..} i{+}l$
$grd3 : j \notin got$
$grd4 : j{-}i \in dom(b)$
THEN
$act3 : b := \{j{-}i\} \domsub b$
END

EVENT loosingack
ANY
k
WHERE
$grd1 : k \in ack$
THEN
$act1 : ack := ack \setminus \{k\}$
END

Exercice 2 *(plusboulanger.tla)*
L'algorithme BAKERY *résout le problème de l'xclusion mutuelle pour un système centralisé. Vous pouvez récupérer le fichier tla crrespondant à cet exemple sur le*

site.

Question 2.1 *Poser une question sur l'accessibilité du processsus 1 en section critique.*

Question 2.2 *Poser une question sur l'accessibilité du processsus 2 en section critique.*

Question 2.3 *En bornant les valeurs de $y1$ et $y2$, montrer que la solution retenue satisfait la propriété d'exclusion mutuelle que vous énoncerez.*

Question 2.4 *Expliquez et justifiez expérimentalement que les valeurs de $y1$ et $y2$ croissent.*

Exercice 3 *dis-ricartagrawalav0*
Le fichier de l'algorithme de Ricart et Agrawala est sur le site.

Question 3.1 *Modéliser l'algorithme de Ricart et Agrawala en TLA$^+$.*

Question 3.2 *Enoncer la propriété à vérifier.*

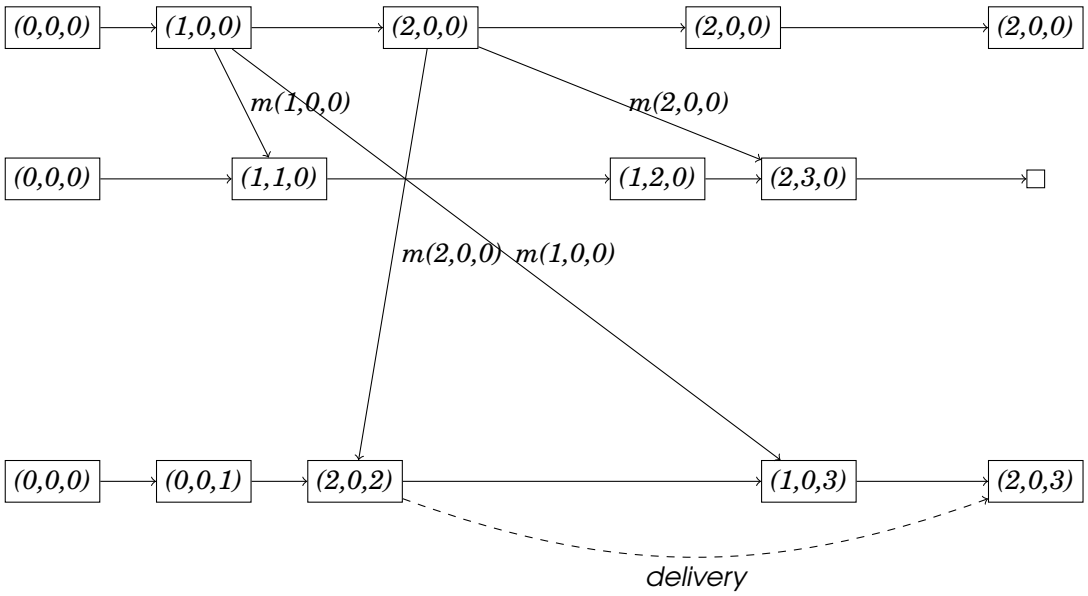
Question 3.3 *Poser une question montrant que toute demande de section critique par un processus P sera servie.*

Question 3.4 *Modifier les actions pour supprimer le sémaphore ey montrer qu'il y a un interblocage.*

Question 3.5 *En vous aidant de la version algorithmique de Ricart et Agrawala, écrire un algorithme PlusCal.*

Question 3.6 *En reprenant l'algorithme de Ricart et Agrawala, on peut le simplifier pour construire l'algorithme de Carvalho et Roucairol. Modéliser l'algoritme de Carvalho et Roucairol.*

Exercice 4 *(vecteurs d'horloge) (pluscal_vc.tla, vector_clock.tla)*
Soit un ensemble de processus P communiquant par messages en groupe. Cela signifie que les processus d'un groupe de P peuvent envoyer des messages à un groupe. On s'intéresse à l'ordre FIFO c'est-à-dire la propriété suivante : si un processus p d'un groupe g de P envoie un message m1 avant un message m2, aucun processus correct de g ne livrera le message m2 avant le message m1.



Dans cet exemple, le message 2 est reçu avant le message 1 et doit donc être livré plus trad après la livraison du message 1. L'algorithme FBCAST résout ce problème en livrant selon la règle des estampilles. En fait le processus 3 attend un message avec une estampille dont le champ de l'émetteur vaut cette valeur. Dans notre exemple, 3 attend un message de 1 avec 1 et quand il reçoit le second message avec la valeur 2, il attend.
Ecrire un ensemble d'opérations de communications mettant en oeuvre ce mécanisme.

Exercice 5 *(Protocole CBCAST)*
Le protocole CBCAST utilise les vecteurs d'horloges pour livrer les messages en respectant l'ordre FIFO des messages envoyés : si un processus P envoie un message m1 puis m2 à un processus Q, alors le protocole livrera d'abord m1 puis m2.
Le principe général est le suivant :

- *Le vecteur d'horloges* $VC \in 1..n \longrightarrow (1..n \longrightarrow \mathbb{N}$ *est initialisé à 0 pour toutes composantes.*
- *Si un processus i envoie un message m,* $VC(i)[i]$ *est incrémenté de 1.*
- *Tout message envoyé m est estampillé par* $VC(i)$ *:* $TM(m) = VC(i)$ *où* $TM \in MES \nrightarrow \mathbb{N}$
- *Quand un processus j reçoit un message estampillé* m*, il met à jour l'horloge de j comme suit :*
  $\forall k \in 1..n : VC(j)[k] = Max(VC(j)[k], TM(m)[k]$

Pour le protocole CBCAST, on a des restrictions sur la livraison effective des messages :

— *Si le processus i reçoit un message m, il le,place en file d'attente* ***CB-QUEUE*** *en attendant que le condition suivante soit vraie :*
$\forall k \in 1..n : \begin{cases} TM(m)[i] = VC(i)[k]+1 \\ TM(m)[k] \leq VC(i)[k] \end{cases}$

— *La livraison du message met à jour* $VC(i)$.