

**Exercice 1** (*dis-slidingwindow*)

**Question 1.1** Le protocole appelé *Sliding Window Protocol* est fondé sur une 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+.

**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.

— Initialisation du protocole

$$INITIALISATION \stackrel{def}{=} \left[ \begin{array}{l} OUT := \emptyset \\ i := 0 \\ chan := \emptyset \\ ack := \emptyset \\ got := \emptyset \end{array} \right]$$

— Phase d'envoi et de réception

$$send \stackrel{def}{=} \text{if } \left[ \begin{array}{l} j \in i..i+l \\ j \leq n \\ j \notin got \end{array} \right] \text{ then } [ chan(j) := IN(j) ] \text{ end}$$

$$receive \stackrel{def}{=} \text{if } \left[ \begin{array}{l} j \in dom(chan) \\ j \in i..i+l \end{array} \right] \text{ then } \left[ \begin{array}{l} OUT(j) := chan(j) \\ ack := ack \cup \{j\} \end{array} \right] \text{ end}$$

— Phase d'accusé de réception et de compl  tion

$$receiveack \stackrel{def}{=} \text{if } [ k \in ack ] \text{ then } \left[ \begin{array}{l} got := got \cup \{k\} \\ ack := ack \setminus \{k\} \end{array} \right] \text{ end}$$

$$completion \stackrel{def}{=} \text{if } [ i = n+1 \wedge got = \emptyset ] \text{ then } [ skip ] \text{ end}$$

— Gestion de la fen  tre

$$sliding \stackrel{def}{=} \text{if } \left[ \begin{array}{l} grd1 : got \neq \emptyset \\ grd3 : i \in got \\ grd4 : i+l < n \end{array} \right] \text{ then } \left[ \begin{array}{l} act1 : i := i+1 \\ act2 : got := got \setminus \{i\} \\ act3 : ack := ack \setminus \{i\} \end{array} \right] \text{ end}$$

La fen  tre ne glisse plus quand elle ne peut plus mais elle se vide et fonde en quelque sorte ( $i+l \geq n$ ).

$$\begin{aligned}
\text{emptywindow} &\stackrel{\text{def}}{=} \text{if} \left[ \begin{array}{l} \text{grd1} : \text{got} \neq \emptyset \\ \text{grd2} : i \in \text{got} \\ \text{grd3} : i+l \geq n \\ \text{grd4} : i \leq n \end{array} \right] \text{then} \left[ \begin{array}{l} \text{act1} : i := i+1 \\ \text{act2} : \text{got} := \text{got} \setminus \{i\} \\ \text{act3} : \text{ack} := \text{ack} \setminus \{i\} \end{array} \right] \text{end} \\
\text{loosingchan} &\stackrel{\text{def}}{=} \text{if} \left[ \begin{array}{l} \text{grd1} : j \in i..i+l \\ \text{grd2} : j \in \text{dom}(\text{chan}) \\ \text{grd3} : j \notin \text{got} \end{array} \right] \text{then} \left[ \text{act1} : \text{chan} := \{j\} \triangleleft \text{chan} \right] \text{end} \\
\text{loosingack} &\stackrel{\text{def}}{=} \text{if} \left[ \text{grd1} : k \in \text{ack} \right] \text{then} \left[ \text{act1} : \text{ack} := \text{ack} \setminus \{k\} \right] \text{end}
\end{aligned}$$

**Exercice 2** (*plusboulanger.tla*)

L'algorithme BAKERY résout le problème de l'exclusion mutuelle pour un système centralisé. Vous pouvez récupérer le fichier tla correspondant à cet exemple sur le site.

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

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

**Question 2.3** En bornant les valeurs de  $y_1$  et  $y_2$ , 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  $y_1$  et  $y_2$  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** Énoncer la propriété à vérifier.

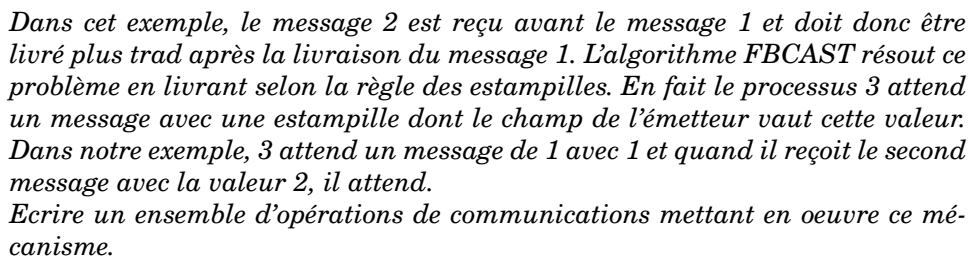
**Question 3.3** Poser une question montrant que toute demande de section critique par un processus  $P$  sera servie.

**Question 3.4** Modifier les actions pour supprimer le sémaphore  $ey$  et 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'algorithme de Carvalho et Roucairol.

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  $m_1$  avant un message  $m_2$ , aucun processus correct de  $g$  ne livrera le message  $m_2$  avant le message  $m_1$ .



*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  $m_1$  puis  $m_2$  à un processus  $Q$ , alors le protocole livrera d'abord  $m_1$  puis  $m_2$ .*

- Le vecteur d'horloges  $VC \in 1..n \rightarrow (1..n \rightarrow \mathbb{N})$  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  $TM \in MES \rightarrow \mathbb{N}$

- Quand un processus  $j$  reçoit un message estampillé  $m$ , il met à jour l'horloge de  $j$  comme suit :

$$\forall k \in 1..n : VC(j)[k] = \text{Max}(VC(j)[k], TM(m)[k])$$

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 la 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  $VC(i)$ .