Cours 8

Attaque coordonnée

Introduction à l'auto-stabilisation

- Exemple u protocole du bit alterné

Attaque coordonnée

Résultat: pour tout graphe de communication avec au moins deux processus il n'existe pas d'algorithme pour l'attaque coordonnée

- Pour prouver le résultat on peut se restreindre à un graphe complet avec deux processus
  - Pourquoi? (Exercice)

Parce que!

Soit supposons $\mathscr{A}$ algorithme d'attaque coordonnée sur un graphe avec n processus $(n > 1)$ $\{p_0, \cdots, p_{n-1}\}$, avec A et B et un graphe complet à 2, on peut simuler les exécutions de cet algorithme: A simule p_0, B simule les autres, avec la simulation de $\mathscr{A}$, on aurait un algorithme pour l'attaque coordonnée pour A et B: s'il n'existe pas d'algorithme pour A et B, il n'y en pas pour un graphe de communication à n

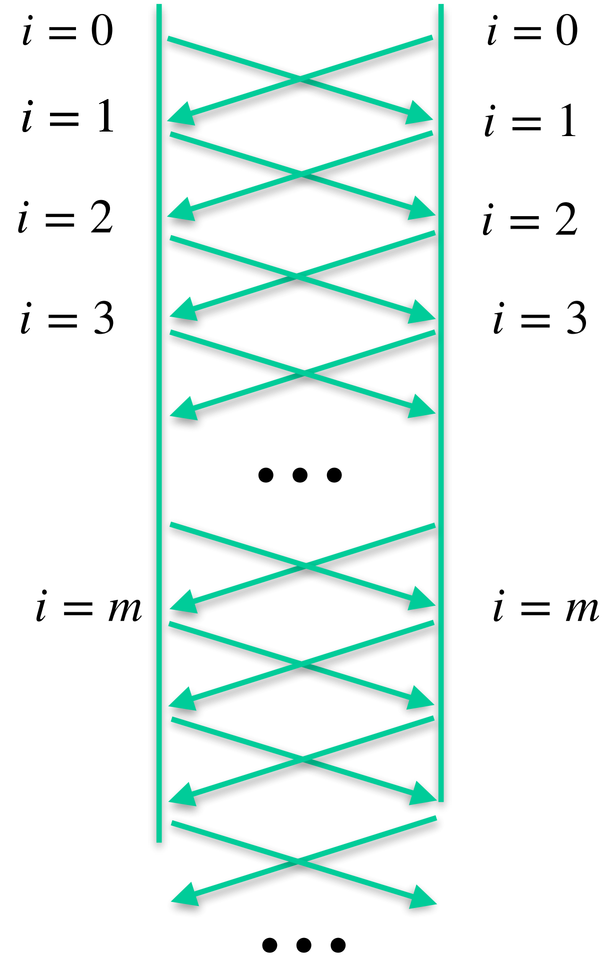
« Rondes »

On peut se restreindre à ne considérer que des rondes synchronisées:

Aux temps $i = 0, \cdots n, \cdots$:

- A envoie un message à B. Ce message est reçu par B au temps $i+1$(ou le message est perdu)

Et se restreindre à montrer qu'il n'existe pas d'algorithme pour deux processus en <u>rondes synchronisées</u>

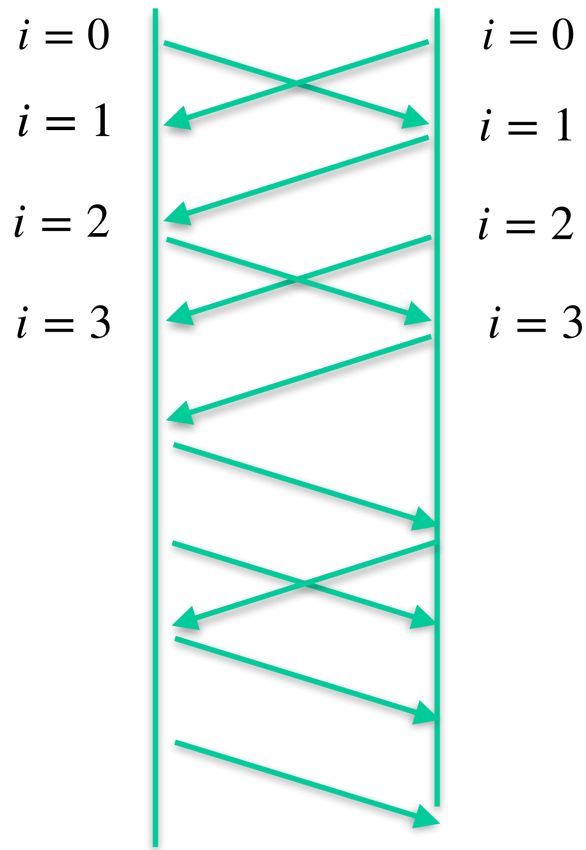


Pourquoi?

Schéma de communication

On suppose que les processus sont déterministes: une exécution est entièrement déterminée par

- L'état initial (v_A pour A et v_B pour B)
- Les messages qui sont perdus (schéma de communication)



S: schéma de communication:

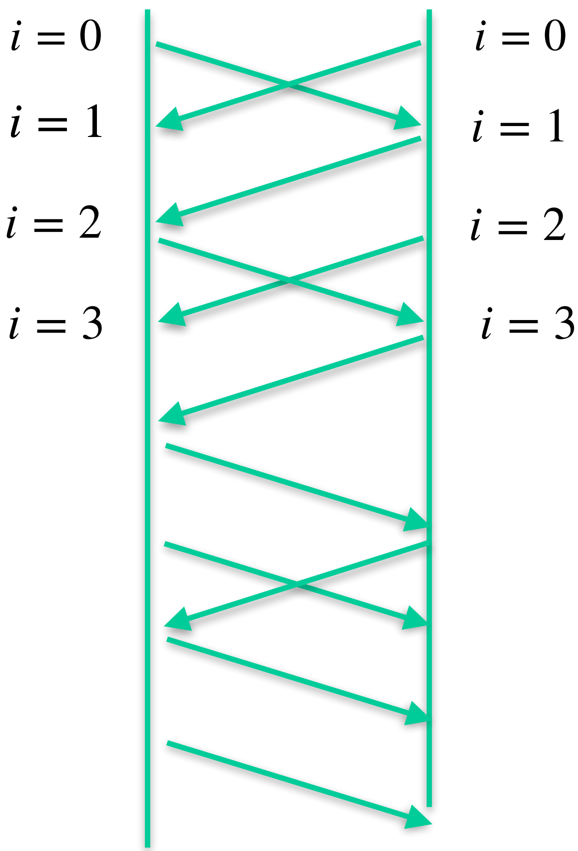
$(i, j, k) \in S$ si et seulement si

Le message envoyé par i dans la ronde k a été reçu par j

(schéma pour plusieurs processus, ici i et j sont A ou B)

Remarques...

- déterminisme : une exécution α est déterminée par les valeurs initiales (v_a, v_b) et un schéma de communication (C) : $\alpha = (C, v_a, v_b)$
- $\alpha[k]$ exécution jusqu'à la ronde k
- $\alpha[k] \sim_i \alpha'[k]$ indistinguable pour i
- $\alpha[k] \sim \alpha'[k]$ $\exists i$: $\alpha[k] \sim_i \alpha'[k]$indistinguable
- Remarques: (α exécution d'un algorithme d'AC)
  - terminaison et accord: pour tout α il y a une (unique) décision: $d(\alpha)$
  - validité:
    - si $\alpha = (C,0,0)$ alors $d(\alpha) = 0$ (C quelconque)
    - si α=($C_\varnothing$, 1,1) alors $d(\alpha) = 1$ ($C_\varnothing$ schéma sans perte)



preuve

Supposons qu'il existe un algorithme pour l'attaque coordonnée:

- $(C_{\emptyset},1,1)$ A (disons) décide en premier: il décide dans la ronde k
- C_k: tous les messages perdus après la ronde k: $(C_{\emptyset},1,1)[k] = (C_k,1,1) = \beta_k$ même décision
- β'_k: $(C'_k,1,1)$ où C'_k identique à C_k sauf que le dernier message de A vers B est perdu : $\beta_k \sim_A \beta'_k$: même décision 1
- β_{k-1}:$(C_{k-1},1,1)$ où C_{k-1} identique à C'_k sauf que le dernier message de B vers A est perdu: $\beta_{k-1} \sim_B \beta'_k$: même décision 1
- …
- β'_1: $(C'_1,1,1)$ où C'_1 identique à C_1 sauf que le dernier message de A vers B est perdu : $\beta_1 \sim_A \beta'_1$: même décision 1
- β_0:$(C_0,1,1)$ où dans C_0 tous les messages sont perdus $\beta_0 \sim_B \beta'_1$: même décision 1
- γ_0:$(C_0,1,0)$ on a $\gamma_0 \sim_A \beta_0$: même décision 1
- γ'_0:$(C_0,0,0)$ on a $\gamma'_0 \sim_B \gamma_0$: même décision 1

Mais par la validité γ_0 doit décider 0

Contradiction!

Résultat

Il n'existe pas d'algorithme déterministe pour l'attaque coordonnée

niveaux de connaissance

En reprenant les remarques sur la logique de la connaissance

En considérant 2 processus seulement:

- on peut définir le niveau de connaissance acquis à la ronde k: $l(k,p)$
  - en ronde 0:
    - $l(0,p) = 0$ au début on ne sait rien
  - en ronde k:
    - si p reçoit un message dans la ronde k, $l(k,p) = \max(l(k-1,p), l(k-1,q)+1)$ (q est l'autre processus)
- (On peut généraliser à un ensemble de processus)

niveaux…

On a:

- chaque processus peut calculer dynamiquement son niveau
- à la ronde k les niveaux de connaissance de A et B diffèrent d'au plus 1: $\forall k : |l(k,A) - l(k,B)| \leq 1$
- si tous les messages jusqu'à la ronde k arrivent, $l(k,A) = l(k,B) = k$
- Exercices:
  - définir un algorithme qui permet dynamiquement de calculer $l(k,p)$ pour
  - prouver que $\forall k : |l(k,A) - l(k,B)| \leq 1$

Un algorithme probabiliste (désaccord avec probabilité bornée par $1/r$)

A choisit aléatoirement k entre 1 et r

c_p initialement 0, niveau de connaissance de p ($p = A$ ou $p = B$)

$V = \bot$

chaque message de p contient c_p et v_p et k

à chaque ronde si p reçoit (c, v, x):

- $c_p := max(c_p, c + 1)$
- $V := v$
- $k = x$

après r rondes

- soit l $(l = c_p)$ le niveau de connaissance atteint
- si $l \geq k$ et si les deux valeurs initiales sont 1 ($V = 1 \wedge v_p = 1$) décider 1
  - sinon décider 0

Un algorithme probabiliste (Monte Carlo)

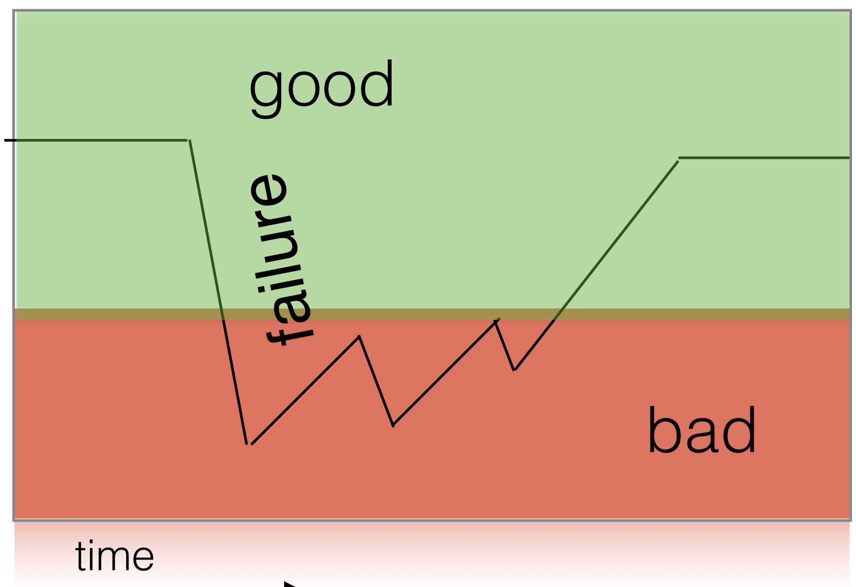
- terminaison: claire
- validité:
  - si $v_A = 1 \wedge v_B = 1$ et il n'y a pas de perte, à la ronde k, $l(k, A) = l(k, B) = k$ et donc A et B décident 1
  - si $v_A = 0 \wedge v_B = 0$, A et B décident 0
- accord probabiliste:
  - $v_A \neq 1 \vee v_B \neq 1$ il y a accord
  - si $v_A = 1 \wedge v_B = 1$ soit $l = \max(l(r, A), l(r, B))$
    - si $k < l$ accord et décision 1
    - si $k > l$ accord et décision 0
    - désaccord possible si $k = l$: désaccord si le choix de k dans $[1, \cdots, r]$ est égal à l: probabilité $1/r$
- Probabilité de désaccord est bornée par $1/r$

Auto-stabilisation
(Self-stabilization)

auto-stabilisation

Partant d'un état quelconque des variables, l'algorithme finit par satisfaire sa spécification:

- Un jour l'état du système est sûr (satisfait la spécification)
- Une fois dans un état sûr, l'algorithme reste dans un état sûr

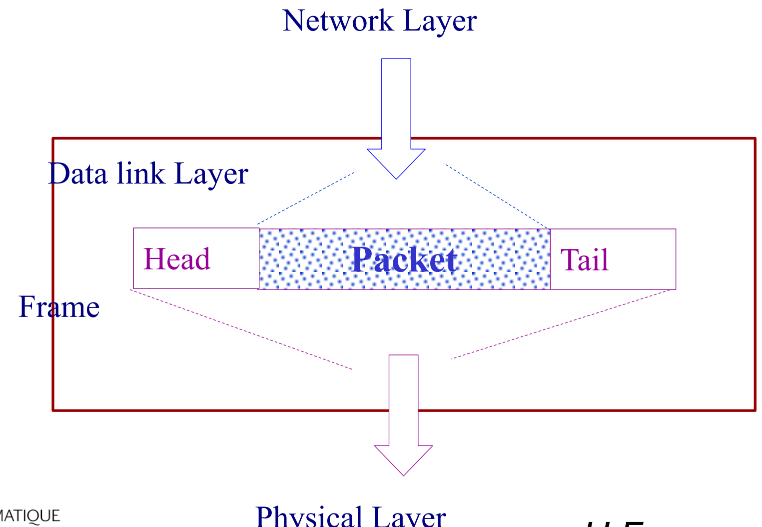


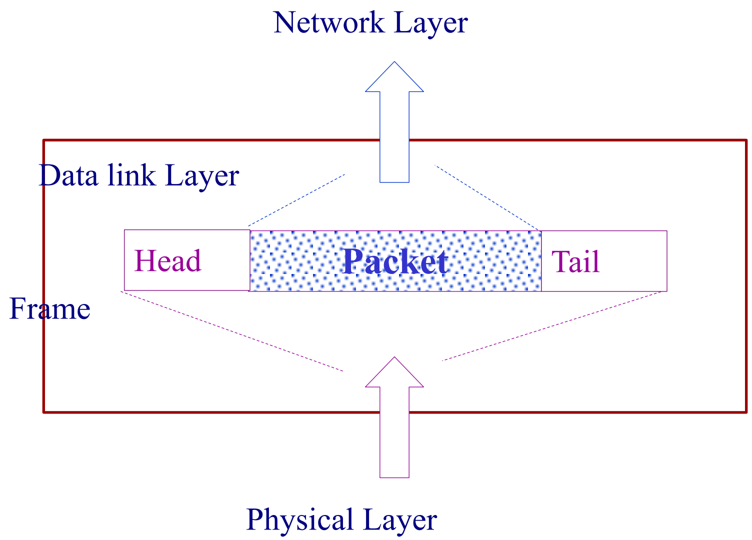
- Des défaillances transitoires (transient) comme corruption de la mémoire
- Auto-stabilisation: un jour tout redevient normal

exemple: protocole du bit alterné

Un algorithme (classique) de la couche liaison de données:

- Réaliser un transfert fiable de données sur un lien de communication non fiable:
  - Le lien de communication peut perdre des messages
  - Grâce à des retransmissions des messages l'alogithme permet d'obtenir un flux de données fiable



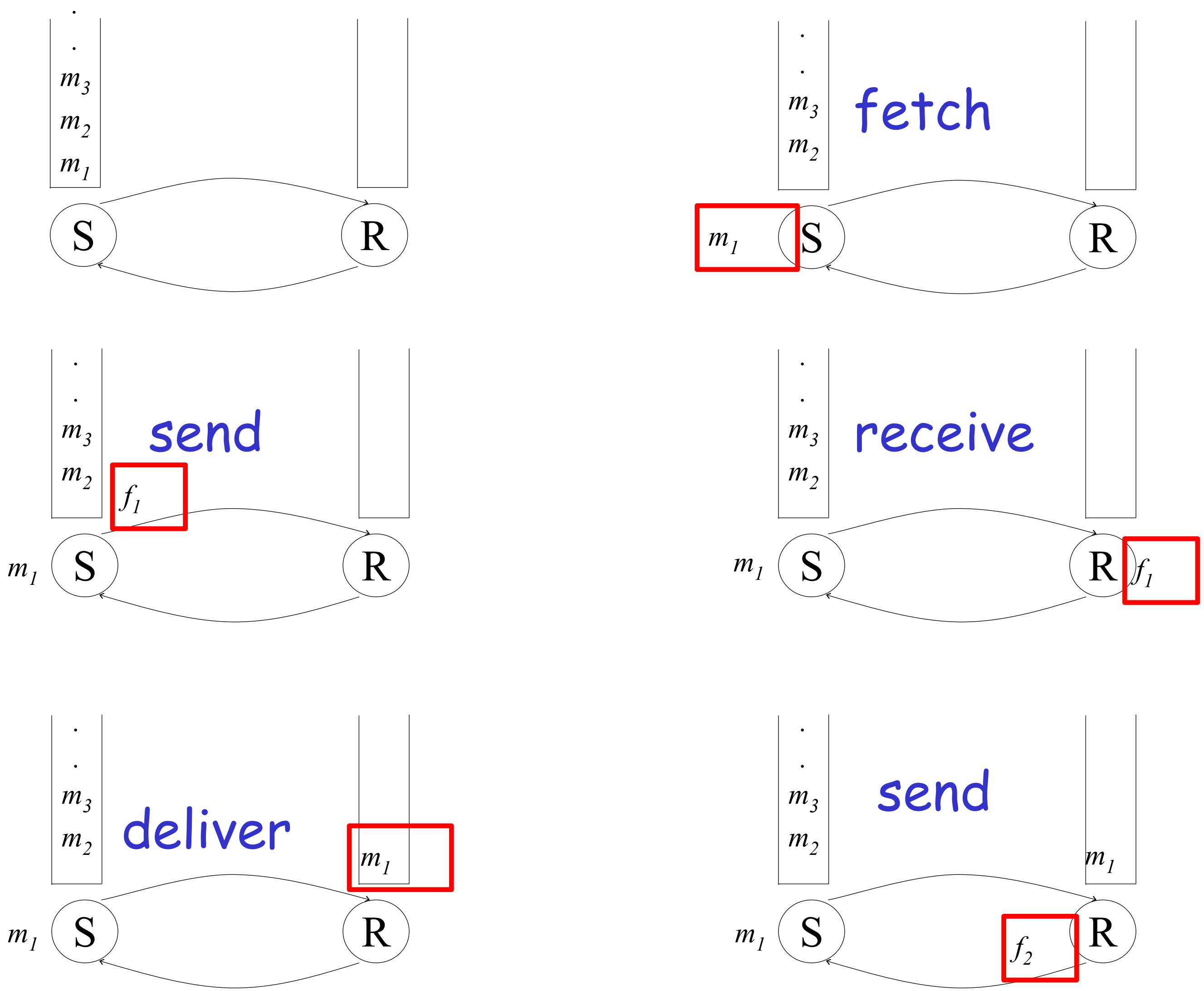


liaison de données

Flot des messages:

File d'entrée des messages
File de sortie des messages

fetch-deliver
send-receive



auto-stabilisation,
Protocole du bit alterné

L'algorithme:

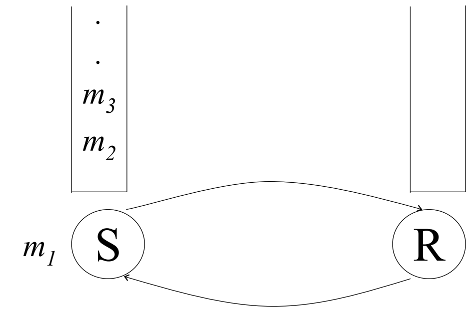
S l'émetteur a une file infinie de messages d'entrée (in[1],in[2],…) à transférer vers le récepteur dans le même ordre sans omissions ni duplications

R le récepteur a une file de messages de sortie (out[1],out[2],…). La séquence des messages de sortie dans la file doit toujours être un préfixe de la séquence des messages dans la file d'entrée.

Principes de l'algorithme:

- **Canaux**:
  - Un canal die l'émetteur vers le récepteur et un canal du récepteur vers le sender
  - Le canaux peuvent perdent de messages
- **Les processus:**
  - ajouter un bit à chaque message
  - alterner les bits: ajouter successivement 0 puis 1
  - Le récepteur accepte le message si le bit est le bit attendu (0 ou 1) et envoie un ack (0 ou 1)
  - Le récepteur envoie un nouveau message s'il a reçu le bit d'ack attendu
  - Timeout (sans réception de l'ack attendu) : l'émetteur envoie à nouveau le message

Protocole



Comportement du canal

LOSS Q_s**:**
$\{m \in Q_s\} \longrightarrow$
$\quad Q_s := Q_s - \{m\}$
LOSS Q_r**:**
$\{m \in Q_r\} \longrightarrow$
$\quad Q_r := Q_r - \{m\}$

Code pour l'émetteur s:

Initialization
$\quad i := 0$
$\quad bit_s := 0$
$\quad send(bit_s, in[i])$
$\quad reset(Timer)$
TIMEOUT:
{ **Timeout: Timer a expired** } $\longrightarrow$
$\quad send(bit_s, in[i])$
SEND:
{ **A message** (b) **in Queue** Q_s} $\longrightarrow$
$\quad receive(b)$
$\quad$ **if** $b = bit_s$ **then**
$\quad\quad bit_s := 1 - bit_s$
$\quad\quad i := i + 1$
$\quad send(bit_s, in[i])$
$\quad reset(Timer)$

Code du récepteur r:

Initialization
$\quad j := 0$
$\quad bit_r := 1$
RECIEVE:
{ **A message** $(b, data)$ **in Queue** Q_r} $\longrightarrow$
$\quad receive(b, data)$
$\quad$ **if** $b \neq bit_r$ **then**
$\quad\quad bit_r := 1 - bit_r$
$\quad\quad j := j + 1$
$\quad\quad out[j] := data /* Deliver */$
$\quad send(bit_r)$

Exercice:

Sûreté? Vivacité?

Etat sûr?

Si on est dans un état sûr on reste dans un état sûr.

On suppose que le canal assure que si une infinité de messages est émis une infinité de message est reçu.

Cette propriété assure la vivacité

Auto-stabilisation

- Partant de n'importe quel état on arrive à un état sûr
- Une fois dans un état sûr on reste dans des états sûrs.
- Ici on a une propriété plus faible (pseudo-stable): on peut garantir que partant de n'importe quel état du canal on arrivera on ne perdra qu'un nombre fini de messages.

Défaillances…

On considère ici des pannes « crashs »

- $\{p_1, \cdots, p_n\}$ n processus au plus t parmi eux peuvent tomber en panne (=arrêter d'exécuter leur code)
- Dans une exécution p est correct s'il fait une infinité de pas d'exécution sinon il est défaillant

Algorithme t résilient: vérifie sa spécification si au plus t parmi eux peuvent s'arrêter.

Consensus

On va considérer un problème très simple mais fondamental: le consensus

Spécification:

- algorithme de décision:
  - tous les processus p ont une valeur initiale v_p prise dans un ensemble V ($\forall p : v_p \in V$), chaque processus doit décider de façon irrévocable (écriture d'une variable d_p qui ne peut être écrite qu'une seule fois: écriture de d_p= décision de p)
  - qui vérifie
    - Accord: si p et q (corrects?) décident alors ils décident la même valeur
    - Validité: la valeur décidée est une des valeurs initiales
    - Terminaison: tout processus correct décide.

Rondes synchronisées

On considère ici un modèle synchrone pour les processus et les communication.

Rondes synchronisées

Suite de « rondes » r=1, …, m,…

- Dans la ronde r:
  - Chaque processus envoie à tous
  - Chaque processus reçoit de tous les messages de la ronde r
  - Chaque processus change son état (suivant ce qu'il a reçu)

Remarques

Exercices:

- montrer qu'on peut réaliser des rondes synchronisées si les processus et la communication sont synchrones.
- (En déduire que:
  - Il existe un algorithme pour P dans le modèle de rondes synchronisés si et seulement si il existe un algorithme pour P dans le modèle avec processus et communication synchrones)
- Que se passe-t-il en ce qui concerne les pannes des processus?

Exercice:

Essayer de trouver un algorithme de consensus:

- Dans un système synchrone sans pannes $(t = 0)$
- Dans un système asynchrone sans pannes $(t = 0)$
- Avec t pannes et des rondes synchronisées où si p tombe en panne dans la ronde r, soit aucun de ses messages n'arrive soit tous ses messages arrivent