Cours 3

Terminaison distribuée (fin)

Vagues (début)

Quelques idées

Un chef: répartit le travail et/ou donne les autorisations de travail

Critique?

Quand on a fini on « consulte » tout le monde?

Consultation?

- pourquoi une simple consultation générale ne fonctionne pas?

Détection de la terminaison…

Consultation:

consulter tout le monde (une « vague »)

- un (ou plusieurs) initiateur i
- consultation de tout le monde après l'initiateur (= causalement après l'ordre de Lamport$\preceq$))
- une « décision » d est un événement qui « récupère » le résultat de la consultation: il est causalement après l'arrivée de la consultation sur chaque processus. Le décideur peut être ou non l'initiateur.

$$\boxed{\forall p : \exists a \in E_p : i \preceq a \preceq d}$$

Notations

- On ne s'intéresse qu'au schéma de communication nécessaire pas au contenu des messages (qu'on notera <>)- ils peuvent contenir toutes les infos

  (Dans une consultation on pose une question et on attend une réponse, cette question et les réponses correspondront à la valeur de <>)
- On considère les événements (émission et réception) de la vague
- On notera (en général) e_p le premier événement de la vague sur p
  - Si cet événement est une émission d'un message, p est un initiateur
  - Sinon cet événement est une réception d'un message de la vague
- Il peut y avoir un ou plusieurs initiateurs
- Il peut y avoir un ou plusieurs décideurs (mais tous les décideurs doivent vérifier $\forall p : \exists a \in E_p : i \preceq a \preceq d$
- (On peut facilement diffuser à tout le monde la décision et assurer ainsi que tout le monde soit décideur)

La consultation?

Consultation: on verra comment faire des consultations (vagues-traversées) ensuite.

La consultation est essentiellement un parcours distribué de graphe.

Exemple: un jeton qui passe par tous les processus sur un anneau

- Le jeton passe par tous les sites et retourne à son point de départ:
  - Chaque site maintient un booléen b_p: le jeton est un booléen B,
  - Quand le jeton arrive sur p: le jeton passe au site suivant avec la valeur $B := B \wedge b_p$.
  - Après un tour $B = \bigwedge_{p \in \Pi} b_p$, (en fait $B = \bigwedge_{p \in \Pi} b_p^{t_p}$ où $b_p^{t_p}$ est la valeur de b_p au moment où le jeton passe en p).

Simple consultation?

- Pourquoi une simple consultation: « avez-vous terminé localement? » n'est-elle pas suffisante?

Détection de la terminaison

Il ne suffit pas que la consultation ait eu des réponses TL_p pour tous les p (pourquoi?):

- il faudrait avoir $\exists t : \forall p \in \Pi : TL_p^t$ (pour le même t!)

Dans une consultation simple positive on a obtenu pour chaque $p \in \Pi$, que- <u>au moment t_p de la réponse de p</u>- $TL_p^{t_p}$ était vrai…

Proposez des solutions?

Geler?

Pour résoudre ce problème de la consultation on peut « geler » le système:

Un initiateur qui a localement terminé lance une détection de terminaison

- Une première vague « gèle » le travail (arrêt provisoire sur le processus)
- Quand un décideur sait que le travail est *gelé* chez tout le monde:
- Un deuxième vague interroge sur l'état au moment du « *gel* » (oui si terminaison locale)
  - Si le décideur de cette deuxième vague Si tout le monde répond « oui »: un décideur de la vague sait qu'il y a terminaison
  - (Sinon dans tous les cas il une diffusion « *dégeler* » les calculs)

Critique?

Détection de la terminaison

Comment faire?

- On fait des consultations successives (le début de la suivante ayant lieu (causalement) après la précédente)
- TG est stable il suffit donc d'assurer qu'on a eu un instant t_0 tel que TG^{t_0} était vrai:
  - On change la question:
    - est-ce que depuis la dernière consultation TL_p est restée vraie?Si tous répondent oui, la dernière consultation ayant eu lieu avant le début de celle qui a obtenu ce « oui »

soit t_p^0 l'instant de la consultation précédente sur p

Soit t_p^1 l'instant de la consultation sur p

On a la garantie que $\forall t(t_p^0 \leq t \leq t_p^1) : TL_p^t$

L'instant t_i du début de la consultation est tel que $\forall p : t_p^0 \leq t_i \leq t_p^1$ et donc au temps t_i $TL_p^{t_i}$ soit $\bigwedge_{p \in \Pi} TL_p^{t_i}$ et donc TG^{t_i}.

Détection de la terminaison

- Avec un jeton passant par tous les sites
  - Si le site qui reçoit le jeton a été actif depuis le dernier passage: le booléen est mis à « faux », Sinon on transmet le jeton sans changement
  - L'initiateur (inactif depuis le dernier passage) met le jeton à « vrai »)
  - Si le jeton arrive à l'initiateur avec « vrai » la terminaison est détectée

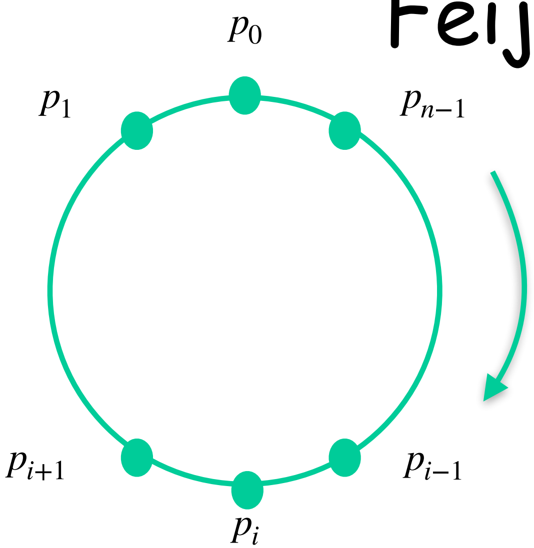
Remarque: on peut faire de nombreuses variantes de l'algorithme

Détection de la terminaison

Remarques:

- Ecrire un algorithme plus précis pour la détection de terminaison.
- Prouver les propriétés de vivacité et de sûreté.
- La forme de la consultation peut être quelconque (pourvue qu'elle vérifie les propriétés précédentes). Un jeton circulant par tous les sites peut faire l'affaire.
- Si au lieu d'avoir un téléphone (ou une communication instantanée) on avait une communication asynchrone (comme le mail) comment pourrait-on faire?

Une variante: algorithme de Dijkstra, Feijen, Van Gasteren



anneau construit sur

$\Pi = \{p_0, \cdots, p_{n-1}\}$, dans l'anneau

le jeton passe de p_i à $p_{(i-1) \mod n}$

p_0 est l'initiateur

Une variante: algorithme de Dijkstra, Feijen, Van Gasteren

$$Term \equiv \forall p : state_p = passive$$

(Ici la terminaison locale est la valeur de la variable state: $TL_p \equiv (state_p = passive)$)

Construction par invariant:

- On cherche une propriété P qui est initialement vraie et qui reste vraie à chaque « step » telle que quand le parcours a réussi (retour du jeton): $P \Rightarrow Term$
- On va procéder par étapes par des « affaiblissements » successifs pour obtenir l'invariant

Une variante: algorithme de Dijkstra, Feijen, Van Gasteren

Soit t l'identité du processus (p_t) qui a le jeton

- $P_0 \equiv \forall i (N > i > t) : state_{p_i} = passive$
  - P_0 est vrai initialement ($t = N - 1$)
  - Quand le jeton revient en p_0 et $state_{p_0} = passive$, $Term$ est vrai:
    - $(t = 0 \wedge state_{p_0} = passive \wedge P_0) \Rightarrow Term$
  - On en déduit:
  - <u>Règle 1:</u> un processus ne transmet le jeton que s'il est passif
    - Mais P_0 n'est pas un invariant: il peut être invalidé par un processus p_i avec $i \leq t$ qui « réveille » un processus p_j ($j > t$) (le jeton n'est pas encore passé par p_i, mais p_i a réveillé p_j qui a déjà eu le jeton)

Une variante: algorithme de Dijkstra, Feijen, Van Gasteren

Affaiblir P_0:

- un processus a une couleur black (white sinon) s'il a pu réveiller un autre processus (envoi d'un message): soit $P_1 \equiv \exists j(t \geq j \geq 0) : colors_{p_j} = black$
- $P_0 \vee P_1$ est vraie initialement et:

$$(P_0 \vee P_1) \\ \wedge color_{p_0} = white \wedge t = 0 \wedge state_{p_0} = passive \\ \Rightarrow Term$$

(Mais $(P_0 \vee P_1)$ n'est pas -encore- un invariant)

<u>Règle 2:</u> un processus qui envoie un message devient black

Une variante: algorithme de Dijkstra, Feijen, Van Gasteren

Il faut aussi assurer que la couleur est « enregistrée »: si un processus est black:

<u>Règle 3:</u> Le jeton devient black s'il rencontre un site black (sauf p_0)

$P_2 \equiv$ le jeton est black

$P_0 \vee P_1 \vee P_2$

- Si pour $t = 0$ le jeton est white P_2 est faux et donc en $t = 0$ si le jeton est white et $color_{t_0}$ est white et $state_{p_0}$ est passive, alors $Term$ est vrai.
- $P_0 \vee P_1 \vee P_2$ est bien un invariant !

Une variante: algorithme de Dijkstra, Feijen, Van Gasteren

- $P_0 \vee P_1 \vee P_2$ est bien un invariant, mais… un jeton black empêche toute détection de terminaison.
- <u>Règle 4:</u> quand p_0 reçoit un jeton black, il initie un nouveau parcours de jeton avec un jeton white
- <u>Règle 5:</u> un processus devient white dès qu'il a transmis le jeton

Résumé

R1: ne faire quelque chose que si on est passive

R2: si on envoie un message on devient black

R3: un processus black ne transmet que des jetons black (sauf p_0)

R4: quand la vague se termine p_0 commence une autre vague

R5: après envoi du jeton le processus devient white

Une variante: algorithme de Dijkstra, Feijen, Van Gasteren

var $state_p$: $(active, passive)$;
$\quad color_p$: $(white, black)$;

$\mathbf{C}_{pq}$: $\{\ state_p = active\ \}$
begin (* p sends a basic message, which is received by q *)
$\quad color_p := black$; (* Rule 2 *)
$\quad state_q := active$
end

$\mathbf{I}_p$: $\{\ state_p = active\ \}$
begin $state_p := passive$ **end**

Start the detection, executed once by p_0:
begin send $\langle \mathbf{tok}, white \rangle$ to p_{N-1} **end**

$\mathbf{T}_p$: (* Process p handles the token $\langle \mathbf{tok}, c \rangle$ *)
$\{\ state_p = passive\ \}$ (* Rule 1 *)
begin if $p = p_0$
$\quad$ **then if** $(c = white \wedge color_p = white)$
$\quad\quad$ **then** *Announce*
$\quad\quad$ **else** send $\langle \mathbf{tok}, white \rangle$ to p_{N-1} (* Rule 4 *)
$\quad$ **else if** $(color_p = white)$ (* Rule 3 *)
$\quad\quad$ **then** send $\langle \mathbf{tok}, c \rangle$ to $Next_p$
$\quad\quad$ **else** send $\langle \mathbf{tok}, black \rangle$ to $Next_p$;
$\quad color_p := white$ (* Rule 5 *)
end

Extrait de:
G. Tel *Introduction to distributed algorithms*

Exemple en TLA+

Pour ceux qui aiment la vérification, TLA+ (temporal logic of actions) :

(site de TLA+)

- langage de spécification formel développé par L. Lamport
- langage de programmation « pluscal » et traducteur
- modèle (spécification)
- TLC model checker
- TLAPS proof system

EWD840: l'algorithme de Dijkstra, Fijne, van Gasteren)

MODULE $EWD840$

TLA+ specification of an algorithm for distributed termination detection on a ring, due to *Dijkstra*, published as *EWD* 840: Derivation of a termination detection algorithm for distributed computations (with *W.H.J.Feijen* and *A.J.M.* van *Gasteren*).

EXTENDS $Naturals$

CONSTANT N

ASSUME $NAssumption \triangleq N \in Nat \setminus \{0\}$

VARIABLES $active, color, tpos, tcolor$

$Nodes \triangleq 0 \,..\, N-1$

$Color \triangleq \{\text{"white"}, \text{"black"}\}$

$TypeOK \triangleq$

$\land\ active \in [Nodes \rightarrow \text{BOOLEAN}]$	status of nodes (active or passive)
$\land\ color \in [Nodes \rightarrow Color]$	color of nodes
$\land\ tpos \in Nodes$	token position
$\land\ tcolor \in Color$	token color

Initially the token is black. The other variables may take any "type-correct" values.

$Init \triangleq$

$\land\ active \in [Nodes \rightarrow \text{BOOLEAN}]$
$\land\ color \in [Nodes \rightarrow Color]$
$\land\ tpos \in Nodes$
$\land\ tcolor = \text{"black"}$

Node 0 may initiate a probe when it has the token and when either it is black or the token is black. It passes a white token to node $N-1$ and paints itself white.

$InitiateProbe \triangleq$

$\land\ tpos = 0$
$\land\ tcolor = \text{"black"} \lor color[0] = \text{"black"}$
$\land\ tpos' = N - 1$
$\land\ tcolor' = \text{"white"}$
$\land\ active' = active$
$\land\ color' = [color \text{ EXCEPT } ![0] = \text{"white"}]$

A node i different from 0 that possesses the token may pass it to node $i-1$ under the following circumstances :

- node i is inactive or
- node i is colored black or
- the token is black.

Note that the last two conditions will result in an inconclusive round, since the token will be black. The token will be stained if node i is black, otherwise its color is unchanged. Node i will be made white.

$$
\begin{aligned}
&PassToken(i) \triangleq \\
&\quad \land\ tpos = i \\
&\quad \land\ \neg active[i] \lor color[i] = \text{"black"} \lor tcolor = \text{"black"} \\
&\quad \land\ tpos' = i - 1 \\
&\quad \land\ tcolor' = \text{IF } color[i] = \text{"black" THEN "black" ELSE } tcolor \\
&\quad \land\ active' = active \\
&\quad \land\ color' = [color \text{ EXCEPT } ![i] = \text{"white"}]
\end{aligned}
$$

token passing actions controlled by the termination detection algorithm

$$System \triangleq InitiateProbe \lor \exists\, i \in Nodes \setminus \{0\} : PassToken(i)$$

An active node i may activate another node j by sending it a message. If $j > i$ (hence activation goes against the direction of the token being passed), then node i becomes black.

$$
\begin{aligned}
&SendMsg(i) \triangleq \\
&\quad \land\ active[i] \\
&\quad \land\ \exists\, j \in Nodes \setminus \{i\} : \\
&\qquad \land\ active' = [active \text{ EXCEPT } ![j] = \text{TRUE}] \\
&\qquad \land\ color' = [color \text{ EXCEPT } ![i] = \text{IF } j > i \text{ THEN "black" ELSE } @] \\
&\quad \land\ \text{UNCHANGED } \langle tpos, tcolor \rangle
\end{aligned}
$$

Any active node may become inactive at any moment.

$$Deactivate(i) \triangleq$$
$$\land\ active[i]$$
$$\land\ active' = [active \text{ EXCEPT } ![i] = \text{FALSE}]$$
$$\land\ \text{UNCHANGED } \langle color, tpos, tcolor \rangle$$

actions performed by the underlying algorithm

$$Environment \triangleq \exists i \in Nodes : SendMsg(i) \lor Deactivate(i)$$

next-state relation: disjunction of above actions

$$Next \triangleq System \lor Environment$$

$$vars \triangleq \langle active, color, tpos, tcolor \rangle$$

$$Spec \triangleq Init \land \Box[Next]_{vars} \land \mathrm{WF}_{vars}(System)$$

Non-properties, useful for validating the specification with *TLC*.

$$TokenAlwaysBlack \triangleq tcolor = \text{"black"}$$

$$NeverChangeColor \triangleq \Box[\text{UNCHANGED } color]_{vars}$$

Main safety property: if there is a white token at node 0 then every node is inactive.

$$terminationDetected \triangleq$$
$$\land\ tpos = 0 \land tcolor = \text{"white"}$$
$$\land\ color[0] = \text{"white"} \land \neg active[0]$$

$$TerminationDetection \triangleq$$
$$terminationDetected \Rightarrow \forall i \in Nodes : \neg active[i]$$

Liveness property: termination is eventually detected.

$Liveness \triangleq$
$(\forall i \in Nodes : \neg active[i]) \leadsto terminationDetected$

The following property asserts that when every process always eventually terminates then eventually termination will be detected. It does not hold since processes can wake up each other.

$FalseLiveness \triangleq$
$(\forall i \in Nodes : \Box\Diamond\neg active[i]) \leadsto terminationDetected$

The following property says that eventually all nodes will terminate assuming that from some point onwards no messages are sent. It is not supposed to hold: any node may indefinitely perform local computations. However, this property is verified if the fairness condition WF_$vars$($Next$) is used instead of only WF_$vars$($System$) that requires fairness of the actions controlled by termination detection.

$SpecWFNext \triangleq Init \land \Box[Next]_{vars} \land \mathrm{WF}_{vars}(Next)$
$AllNodesTerminateIfNoMessages \triangleq$
$\Diamond\Box[\forall i \in Nodes : \neg SendMsg(i)]_{vars} \Rightarrow \Diamond(\forall i \in Nodes : \neg active[i])$

Dijkstra's inductive invariant

$Inv \triangleq$
$\lor\ P0 :: \forall i \in Nodes : tpos < i \Rightarrow \neg active[i]$
$\lor\ P1 :: \exists j \in 0 \,..\, tpos : color[j] = \text{"black"}$
$\lor\ P2 :: tcolor = \text{"black"}$

Use the following specification to let TLC check that the predicate $TypeOK \land Inv$ is inductive for EWD 840: verify that it is an (ordinary) invariant of a specification obtained by replacing the initial condition by that conjunction.

$CheckInductiveSpec \triangleq TypeOK \land Inv \land \Box[Next]_{vars}$

---

* Modification History
* Last modified *Tue Jun* 28 18:17:45 *CEST* 2016 by *merz*
* Created *Mon Sep* 09 11:33:10 *CEST* 2013 by *merz*

Algorithme de Safra

Et si la communication est asynchrone?

(Non-instantanée: des messages peuvent être en « transit »).

- Pourquoi cela ne marche pas?
- Que faire?

La propriété stable devient (B est le nombre de messages en « transit » (=envoyés et non reçus))

$$\forall p : state_p = passive \wedge B = 0$$

- Rendre la communication « synchrone »: à chaque message on attend un accusé de réception:
  - Il n'y a plus de messages (de l'algorithme de détection) dans les canaux
- Compter (estimer) le nombre de messages en transit
  - B est le nombre de message en transit : algorithme de Safra

Algorithme de Safra

Chaque processus maintient un compteur de messages mc_p

Le jeton a une couleur (black ou white) et le nombre estimé des messages en transit

Règles:

<u>Règle M</u>: quand p envoie un message il incrémente mc_p, quand il reçoit un message il décrémente mc_p

<u>Règle 1</u>: le jeton contient le nombre estimé de messages en transit: un processus p transmet le jeton que s'il est passive et ajoute mc_p à la valeur du jeton

<u>Règle 2</u>: un processus qui reçoit un message devient black

<u>Règle 3</u>: le jeton a aussi une couleur, quand un processus black transmet le jeton, le jeton devient black

<u>Règle 4:</u> La vague échoue si elle revient en p_0 avec un jeton black ou si $mc_{p_0} + q \neq 0$ (q est la valeur du jeton). Si la vague échoue p_0 initie une nouvelle vague

<u>Règle 5:</u> un processus devient white après l'envoi du jeton

Algorithme de Safra

```
var state_p   : (active, passive) ;
    color_p   : (white, black) ;
    mc_p      : integer init 0 ;

S_p: { state_p = active }
     begin send ⟨mes⟩ ;
           mc_p := mc_p + 1     (* Rule M *)
     end

R_p: { A message ⟨mes⟩ has arrived at p }
     begin receive ⟨mes⟩ ; state_p := active ;
           mc_p := mc_p − 1 ;     (* Rule M *)
           color_p := black     (* Rule 2 *)
     end

I_p: { state_p = active }
     begin state_p := passive end

Start the detection, executed once by p_0:
     begin send ⟨tok, white, 0⟩ to p_{N−1} end

T_p: (* Process p handles the token ⟨tok, c, q⟩ *)
     { state_p = passive }     (* Rule 1 *)
     begin if p = p_0
             then if (c = white) ∧ (color_p = white) ∧ (mc_p + q = 0)
                     then Announce
                     else send ⟨tok, white, 0⟩ to p_{N−1}     (* Rule 4 *)
             else if (color_p = white)     (* Rules 1 and 3 *)
                    then send ⟨tok, c, q + mc_p⟩ to Next_p
                    else send ⟨tok, black, q + mc_p⟩ to Next_p ;
           color_p := white     (* Rule 5 *)
     end
```

Extrait de:
G. Tel *Introduction to distributed algorithms*

Quelques remarques

De la communication instantanée à une communication asynchrone (du téléphone au courrier)?

- attendre de recevoir un accusé de réception
- (Méthodes similaires à l'algorihtme de Safra)
- (Avoir du broadcast (non-instantanné) ne change pas grand chose)

Centralisé- décentralisé:

- On peut aussi envisager des solutions avec un chef qui répartit le travail

Consultation

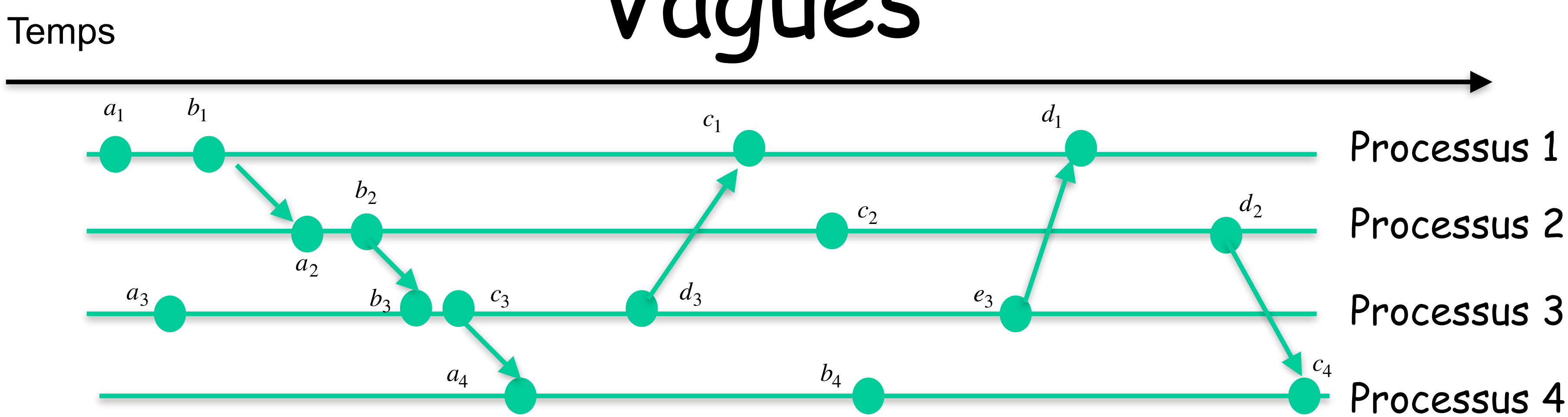
Vagues...

Algorithmes de Vagues

« Consultation » de tous les processus:

- Un initiateur (ou plusieurs) lance la consultation
- Un décideur (ou plusieurs) récupère le résultat de la consultation. Tous les processus on été consultés.
  - Une consultation est après le début de la consultation:
    - si c_p est un événement de la consultation sur p, et si i est un événement du démarrage de la consultation alors $i \preceq a_p$ (causalité)
  - Tous le monde a été consulté:
    - Si d_p est l'événement de décision (=fin de la consultation) sur le décideur p alors pour tout q il existe un événement de consultation $c_q$sur q tel que $c_q \preceq d_p$
      (pour tout q il existe un chemin de communication de l'initiateur i à la décision sur p passant par q)

Vagues



a_1 initiateur c_1 décision : n'est pas une vague

a_1 initiateur c_4 décision : est une vague

Vague: il existe i événement initiateur et d événement de décision tel que

$$\forall p \in \Pi \, \exists c_p \in E_p : i \leq c_p \leq d$$

des vagues…

Propagation d'information with Feedback (PIF):

- Les initiateurs ont un message M
- Diffuser ce message et attendre que le message soit diffusé (notification)

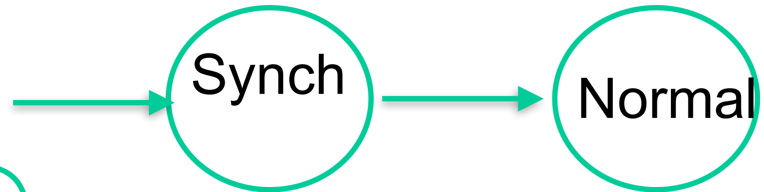
On peut vérifier:

- Tout algorithme de vague permet PIF
- Tout algorithme de PIF est un algorithme de vague
  - Soit une exécution de vague:
    - initiateurs de la vague= initiateurs (ceux qui ont initialement le message M)
    - M est ajouté aux messages de communication de la vague, quand M est reçu par p il est considéré comme diffusé en p
    - Une décision = notification
  - On a bien un algorithme de PIF

des vagues…

Resynch;

- Tous les processus avant d'aller dans l'état normal doivent être simultanément dans l'état synch
- Un algorithme de vague permet Resynch
- Un algorithme de Resynch est un algorithme de vague
  - Décision = état normal



des vagues…

Calcul du minimum

- $(X, \leq)$un ensemble ordonné
- Chaque processus p a une valeur $r_p \in X$
- Calculer $J = \min\{r_p \mid p \in \Pi\}$

- Tout algorithme de calcul du minimum est un algorithme de vagues
- Tout algorithme de vague permet le calcul du minimum

des vagues…

Chaque processus maintient une variable i_p :

- initialement $i_p = r_p$
- Quand p envoie un message, p ajoute au message sa variable i_p
- Quand p reçoit un message avec la valeur v, $i_p := \min(i_p, v)$
  - On notant $i^{(x)}$ la valeur de i pour l'événement x on a:
    $a \preceq b \Rightarrow i^{(a)} \geq i^{(b)}$
  - D'où:
    - $\forall p : i^{(d)} \leq i^{(e_p)} \leq r_p$ (d événement de décision et e_p premier événement de la vague sur p) et donc $i^{(d)} \leq \min\{r_p \mid p \in \Pi\}$ comme $i^{(d)}$ est un des r_p on déduit $i^{(d)} = \min\{r_p \mid p \in \Pi\}$

des vagues

Election:

- choix d'un leader= décider de l'identité d'un processus unique le leader.
- algorithme d'élection de leader:
  - en général algorithme symétrique: tous les processus exécutent le même code (mais ils ont de identités distinctes qu'ils connaissent)
  - en général décentralisé: tout processus peut prendre l'initiative
- Election : Calcul du minimum des identités

des vagues…

Topologie: déterminer la topologie du réseau

Tables de routages

Connectivité

Parcours d'un jeton (traversée : vague particulière sans concurrence des messages)

Exercices…

on suppose que tous les processus ont des identités. On veut un algorithme distribué qui calcule sur un noeud (tous les noeuds) la liste de toutes les identités. (décision: cette liste est calculée)

(1) montrer qu'un tel algorithme est nécessairement une vague

(2) partant d'un algorithme de vague donner un algorithme permettant de calculer la liste des identités