Corrigé

Master 1 Informatique, 2006–2007

MI5a : Algorithmes distribués

(A. Slissenko)

Examen. le 10 janvier 2007, 10h – 12h

1. [3 points] Donnez la définition de l'ordre causal. Donnez la définition d'un algorithme par vague. Prouvez la propriété suivante des algorithmes par vague : Si C est une exécution d'un algorithme par vague et $d_p \in C$ est un événement de décision du processus p, alors

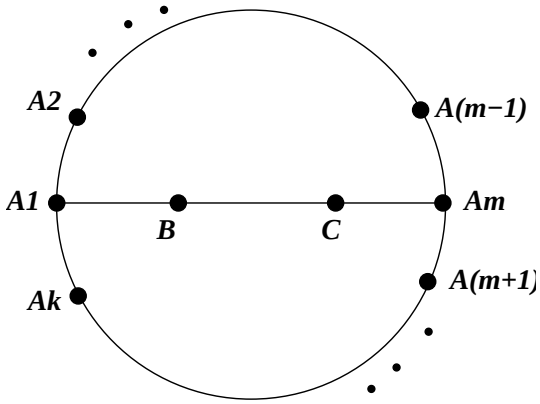
$$\forall\, q \neq p\, \exists\, f \in C_q \ \big(\ f \preceq d_p \ \wedge\ f \text{ est un envoi }\big),$$

où p et q sont des processus, f est un événement, $\preceq$ est l'ordre causal défini par l'exécution C, et C_q est l'exécution de q qui fait partie de C.

Réponse. Pour les définitions voir le cours.

Soit $q \neq p$. La dépendance dit qu'il y a un événement dans q qui précède d_p. Considérons tous les événement dans q qui précèdent d_p : $f_0 \prec f_1 \prec \cdots \prec f_k$ (les événements de C_q sont linéairement ordonnés par définition de l'ordre causal). Le dernier événement de cette suite, i. e., f_k est un envoi car l'ordre causal entre deux événements dans deux processus différents nécessite une chaîne d'envois/réceptions du premier au deuxième.

2. [5 points]



Graphe Φ.

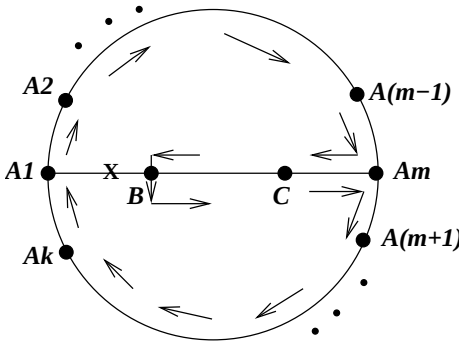
Vous disposez d'un algorithme d'élection local α pour les anneaux uni-directionnels. C'est une boîte noire dont l'interface suivante est disponible : on peut lancer l'algorithme comme initiateur ou non-initiateur, on peut lui envoyer un message, on peut recevoir son message, on sait s'il décide, on sait s'il s'arrête. Décrivez un algorithme d'élection pour le graphe Φ (voir le dessin, le graphe a $k+2$ sommets, $k > m$, dont k sommets sur le cercle) avec des canaux bi-directionnels utilisant l'algorithme d'élection α pour les anneaux uni-directionnels.

Il vous faut redéfinir les voisins et le fonctionnement de l'algorithme dans les sommets $A1$, B, C et Am (c'est très facile pour $A1$ et B, pour deux autres il faut appliquer la construction que vous connaissez pour ce genre de réduction). Rappelons que votre algorithme doit être valide pour un ensemble d'initiateurs arbitraire.

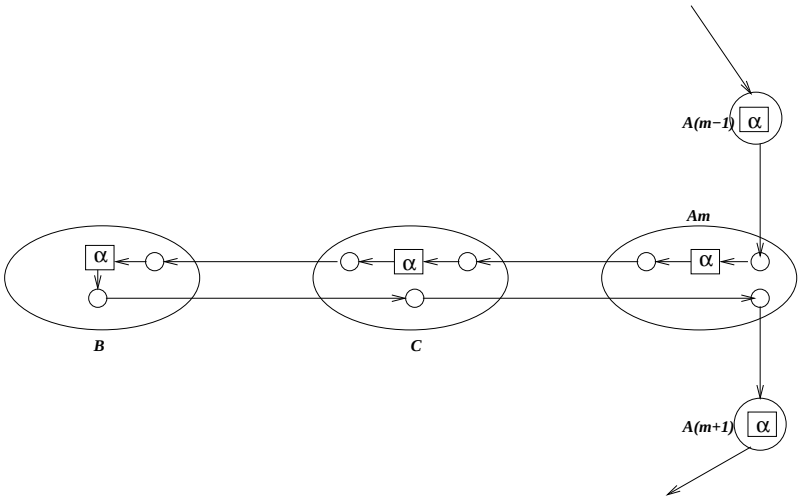
Réponse. Pour les définitions voir le cours.

Il y a plusieurs possibilités de transformer le graphe donné en anneau uni-directionnel. Choisissons

l'orientation suivante :



Graphe Φ avec une orientation choisie.



Les programmes dans les sommets B, C, Am.

Sur la figure droite la boîte noire est représentée par les carrés avec α, le petit cercle représente l'interface de notre programme, dénotons le par F, avec les canaux et la boîte noire.

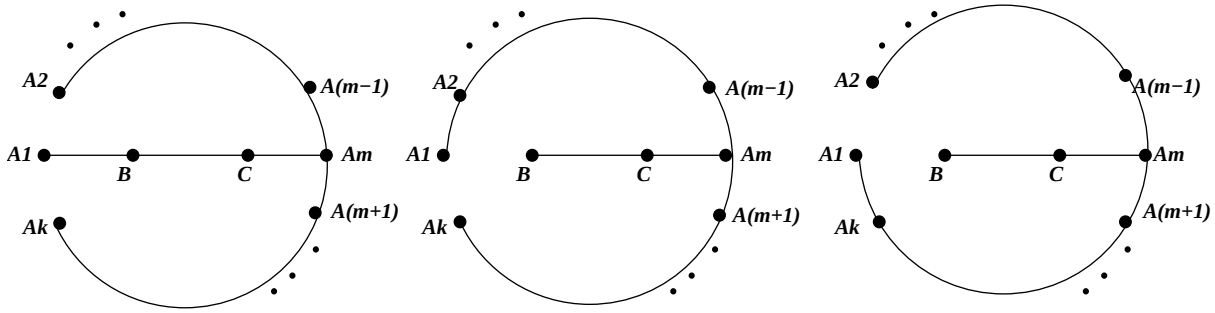
Dans les sommets $A1$ et B l'arête $\{A1, B\}$ est exclue de l'anneau. Dans les sommets $A1, \ldots, A(m-1)$, B, $A(m+1), \ldots, Ak$ l'interface de notre programme F avec la boîte noire est simple ; elle consiste en connecter la boîte noire avec les canaux qui sont définis par l'orientation choisie. Plus précisément, si le sommet est initiateur, alors F lance α comme initiateur, sinon comme non-initiateur. Ensuite F reçoit les messages du canal choisi comme le canal entrant et les envoie à α, et F reçoit les messages de α et les envoie au canal sortant.

Le fonctionnement de F dans les sommets C et Am est le suivant. Considérons son fonctionnement dans C, dans Am il est pareil. Si C est un initiateur, alors F lance α de C comme initiateur, sinon α est lancé comme non-initiateur. Si α envoie un message, alors F renvoie ce message selon la flèche, i. e. à B. Si un message arrive de Am, alors F le renvoie à α. Si α décide, alors F décide. Si un message arrive de B, alors F le renvoie à Am. Remarquons que si α de C s'arrête, alors aucun message ne peut passer à travers α. Dans le cas où α s'arrête, F continue à renvoyer les messages qui arrivent de B.

3. [5 points] Considérons les exécutions de l'algorithme de construction d'un arbre couvrant de la Fig. 1 au graphe Φ (voir le dessin ci-dessus), où $k = 2(m-1)$ (le nombre de sommets au-dessus du chemin $A1, B, C, Am$ est égal au nombre de sommets au-dessous de ce chemin). Prouvez que si $A1$ est l'initiateur, alors cet algorithme peut retourner seulement 3 arbres couvrants dont 2 sont isomorphes. Décrivez ou dessinez ces 3 arbres et justifiez brièvement votre réponse.

Réponse. L'algorithme a 3 choix à partir du sommet $A1$: aller soit à B soit à $A2$ soit à Ak. Chaque choix définit un arbre (chaque arbre peut être obtenu par 2 exécutions différentes selon le premier

choix de continuation à partir du sommet Am), voir la figure ci-dessous.



Trois arbres. Le 2ème et le 3ème sont isomorphes.

4. [4 points] Donnez la définition de la signature numérique. Donnez la définition du consensus byzantin.

Considérons la modification suivante de l'algorithme du cours de consensus byzantin avec signatures numériques. La modification concerne la notion de message valide :

On dit qu'un message $m = \langle \mathbf{val}, v\rangle : g : p_2 : \ldots : p_i$ reçu pendant l'impulsion i par le processus p est *valide* pour p si m a les signatures correctes de $(i-1)$ processus dont le premier est le général, p lui-même n'est pas parmi les processus qui ont signé le message et les signatures sont différentes (la définition correcte demande i signatures).

Modulo cette modification l'algorithme reste le même. On a N processus dont $t < N$ sont malveillants. A l'impulsion 1 le général diffuse le message $\langle \mathbf{val}, x_g\rangle : g$ qui contient son entrée x_g et qui est signé par lui-même. Pendant les impulsions $2, 3, ..., (t+1)$ chaque processus correct p signe et renvoie les 2 premiers messages valides (ayant des valeurs différentes), reçus précédemment et met les valeurs contenues dans ces messages dans son ensemble W_p qui est initialement vide. La décision est faite à la fin de l'impulsion (t+1) et est égale à v si $W_p = \{v\}$, et est égale à 0 si $|W_p| > 1$.

Pour cette modification de l'algorithme prouvez que quelque soit N il y a une exécution sans consensus.

Quelle est la résistance et la complexité en messages de l'algorithme correct de consensus avec 2 renvois de messages ?

Réponse. Pour les définitions, la complexité et la résistance voir le cours.

Remarquons que la définition de message valide suppose que la première signature est celle du général. Cela implique implicitement que $i \geq 2$, donc on ne peut pas envoyer un message valide pendant la première impulsion. Quand même si le général est correct l'algorithme assure l'accord (by défaut).

Supposons que $g, p_2, \ldots, p_t$ sont byzantins et que p et q, $p \neq q$ sont corrects. Jusqu'à l'impulsion $(t+1)$ le général g ne fait rien. Pendant l'impulsion $(t+1)$ le général (ou n'importe quel autre processus byzantins ou n'importe quels 2 ou plus processus byzantins qui font une coalition) envoie à p le message $\langle \mathbf{val}, u\rangle : g : p_2 : \ldots : p_t$ et à q le message $\langle \mathbf{val}, v\rangle : g : p_2 : \ldots : p_t$ avec $u \neq v$. Les deux messages sont valides du point de vue de la notion de validité modifiée. Donc à la fin de l'impulsion $(t+1)$ le processus p décide de u et le processus q décide de v. Il n'y a pas de l'accord.

5. [5 points] Donnez la définition de la notion "le système (algorithme) S se stabilise pour la propriété P". (Un ensemble de configurations légitimes joue un rôle central dans cette notion). L'algorithme Dijkstra's Token Ring est le suivant. On a $N > 2$ processus numérotés $0, \ldots, N-1$. Chaque processus i a une valeur $\sigma_i \in \{0, 1, \ldots, K-1\}$, où $K > N$. On dit que le processus $i \neq 0$ a le privilège si $\sigma_i \neq \sigma_{i-1}$, et que le processus 0 a le privilège si $\sigma_0 = \sigma_{N-1}$. Si un processus i a le privilège il peut changer la valeur de σ_i et donc changer son état : si $i \neq 0$ et $\sigma_i \neq \sigma_{i-1}$, alors $\sigma_i := \sigma_{i-1}$; si $\sigma_0 = \sigma_{N-1}$,

alors $\sigma_0 := \sigma_{N-1} + 1 \pmod K$.

Prouvez que si une configuration contient au plus un processus avec le privilège alors dans chaque exécution à partir de cette configuration chaque processus a le privilège infiniment souvent.

Prouvez qu'après chaque événement dans un processus différent du processus 0, la fonction $\sum_{i \in S} (N - i)$, où $S = \{i \geq 1 : \text{processus } i \text{ a le privilège}\}$, diminue.

Soit $N = 2m$. Considérons la configuration suivante : $\sigma_i = i \mod 2$. Trouvez une exécution qui converge vers une configuration légitime en temps linéaire et une exécution qui converge vers une configuration légitime en temps quadratique.

Réponse. Pour les définitions et les preuves voir le cours.

Une exécution qui converge en temps linéaire : $\sigma_1 := \sigma_0$ (= 0) (1 perd le privilège et 2 aussi car $\sigma_2 = 0$), $\sigma_3 := \sigma_2$ (= 0), ..., $\sigma_{2m-1} := \sigma_{2m}$ (= 0). Après ces transitions seulement 0 a le privilège. Le nombre de transition est $m = \frac{N}{2} = O(N)$.

Une exécution qui converge en temps quadratique : $\sigma_{2m-1} := \sigma_{2m-2}$ (= 0), $\sigma_{2m-2} := \sigma_{2m-3}$ (= 1),...,$\sigma_2 := \sigma_1$ (= 1), $\sigma_1 := \sigma_0$ (= 0), $\sigma_0 := \sigma_0 + 1$ (= 1). Après ces N transitions tous les processus gardent le privilège.

On répète les transitions avec les mêmes processus dans le même ordre et on arrive à $\sigma_0 = 2$, $\sigma_i = i \mod 2$. Cela donne N transitions de plus.

Maintenant on fait les mêmes transitions sauf la dernière car 0 n'a plus de privilège. Après ces $(N - 1)$ transitions on obtient : $\sigma_0 = \sigma_1 = 2$, $\sigma_i = (i + 1) \mod 2$ pour $2 \leq i \leq (N - 1)$. Donc 0 et 1 n'ont plus de privilège.

On continue de la même façon :

$\sigma_{N-1} := \sigma_{N-2}$, $\sigma_{N-2} := \sigma_{N-3}$,...,$\sigma_3 := \sigma_2$ (= 0), $\sigma_2 := \sigma_1$ (= 2). Cela donne $(n - 2)$ transitions, 0, 1, 2 n'ont plus de privilège.

$\sigma_{N-1} := \sigma_{N-2}$, $\sigma_{N-2} := \sigma_{N-3}$,...,$\sigma_3 := \sigma_2$ (= 2)

.....

$\sigma_{N-1} := \sigma_{N-2}$ (= 2). Maintenant le seul processus ayant le privilège est 0.

Le nombre total de transitions est $N + N + (N - 1) + (N - 2) + \ldots 1 = \frac{N(N+3)}{2} \sim \frac{N^2}{2}$.

6. [5 points] Donnez la définition du consensus asynchrone. Quelle est la résistance et la complexité en messages de l'algorithme du consensus asynchrone que vous connaissez (l'algorithme est sur la Figure 2)) ? Pour un nombre $(L + k)$ de processus corrects, où $L = \lceil \frac{N+1}{2} \rceil$, $k > 0$, et N est le nombre total de processus, trouvez une exécution avec le nœud le plus petit possible (en termes de nombre de sommets) et une exécution avec le nœud le plus grand possible ?

Réponse. Pour les définitions, la complexité et la résistance voir le cours.

Une exécution avec le nœud le plus petit possible : l'étoile d'un $p \in E =_{df} \{1, 2, \ldots, L\}$ consiste en tous $q \in E \setminus \{p\}$ plus p lui même au centre, bien sûr. Pour les $p \notin E$ les étoiles sont arbitraire, par exemple, l'étoile de p consiste, à part de p au centre, en $\{1, 2, \ldots, L-1\}$. Dans ce cas le nœud consiste en $\{1, 2, \ldots, L\}$.

Une exécution avec le nœud le plus grand possible : l'étoile de $p \in \mathbb{P} = \{1, 2, \ldots, L + k\}$ consiste en p au centre plus $\{p + 1, p + 2, \ldots, p + (L - 1) \mod (L + k)\}$ (ici $+ \mod (L + k)$ agit sur l'ensemble $\mathbb{P}$, donc $L + k + 1 = 1 \mod (L + k)$). Dans ce cas le nœud consiste en $\{1, 2, \ldots, (L + k)\}$ car pour chaque p il y a un arc de $(p + 1) \mod (L + k)$ à p.

var $used_p[q]$: *boolean* **init** $false$ $\forall\, q \in Neigh_p$;
(* Indique si p a déjà envoyé $\langle$**tok**$\rangle$ à q *)
$father_p$: process **init** undef ;

Seulement pour l'initiateur (qui est unique) exécuter une fois :
begin
$father_p := p$; **choose** $q \in Neigh_p$;
$used_p[q] := true$; **send** $\langle$**tok**$\rangle$ à q
end

Pour tout processus après une réception de $\langle$**tok**$\rangle$ de q_0 :
begin
if $father_p = undef$ **then** $father_p := q_0$;
if $\forall\, q \in Neigh_p .\, used_p[q]$
then *decide*
else
if $\exists\, q \in Neigh_p .(q \neq father_p \wedge \neg\, used_p[q])$
then
begin
if $father_p \neq q_0 \wedge \neg\, used_p[q_0]$
then $q := q_0$
else choose $q \in Neigh_p \setminus \{father_p\}$ tel que $\neg\, used_p[q]$;
$used_p[q] := true$; **send** $\langle$**tok**$\rangle$ à q
end
else
begin
$used_p[father_p] := true$;
send $\langle$**tok**$\rangle$ à $father_p$
end
end

FIG. 1 – Recherche en profondeur classique.

var $Corr_p$: ensemble de processus **init** $\emptyset$; -- correspondants de p ;
var G_p : graphe **init** $(\{p\}, \emptyset)$; -- graphe formé des étoiles des vivants
-- connus via $Corr_p$;
var $Reçus_p$: ensemble de processus **init** $\{p\}$; -- ensemble des processus d'où p a reçu un message
-- du type **étoile** plus p lui-même,
-- donc connus comme vivants
Notation : $L = \lceil \frac{N+1}{2} \rceil$

begin
1 : diffuser $\langle \mathbf{nom}, p \rangle$; -- on envoie aussi la valeur x_p d'entrée avec chaque nom ;
2 : **while** $\sharp Corr_p < L - 1$ **do**
begin
3 : recevoir $\langle \mathbf{nom}, q \rangle$; $Corr_p := Corr_p \cup \{q\}$;
end
4 : diffuser $\langle \mathbf{étoile}, p, Corr_p \rangle$;
-- l'étoile est le graphe dont les arcs sont (q, p) pour $q \in Corr_p$;
5 : $G_p := étoile(p, Corr_p)$;
6 : **while** $(sommets(G_p) \setminus Reçus_p) \neq \emptyset$ **do**
begin
7 : recevoir $\langle \mathbf{étoile}, q, Corr_q \rangle$ pour $q \in (sommets(G_p) \setminus Reçus_p)$;
8 : $G_p := G_p \cup étoile(q, Corr_q)$;
9 : $Reçus_p := Reçus_p \cup \{q\}$;
end
10 : calculer un nœud de G_p ; -- on peut définir un nœud soit comme une composante
-- fortement connexe sans arcs entrants
-- soit comme une la plus grande ;
11 : décider sur la base des valeurs d'entrée des sommets du nœud
end

FIG. 2 – Protocole de consensus asynchrone.