Cours 7

Anonymat, randomisation (fin)

Attaque coordonnée (début)

Calcul sur l'anneau

Fonction cyclique:

- f de X^* dans A est cyclique si et seulement si $f(x) = f(y)$ pour y obtenu par permutations circulaires (cycles) de x
- La somme, $\vee$, $\wedge$, … sont cycliques.

Calcul de f : chaque processus p écrit le résultat dans res_p:

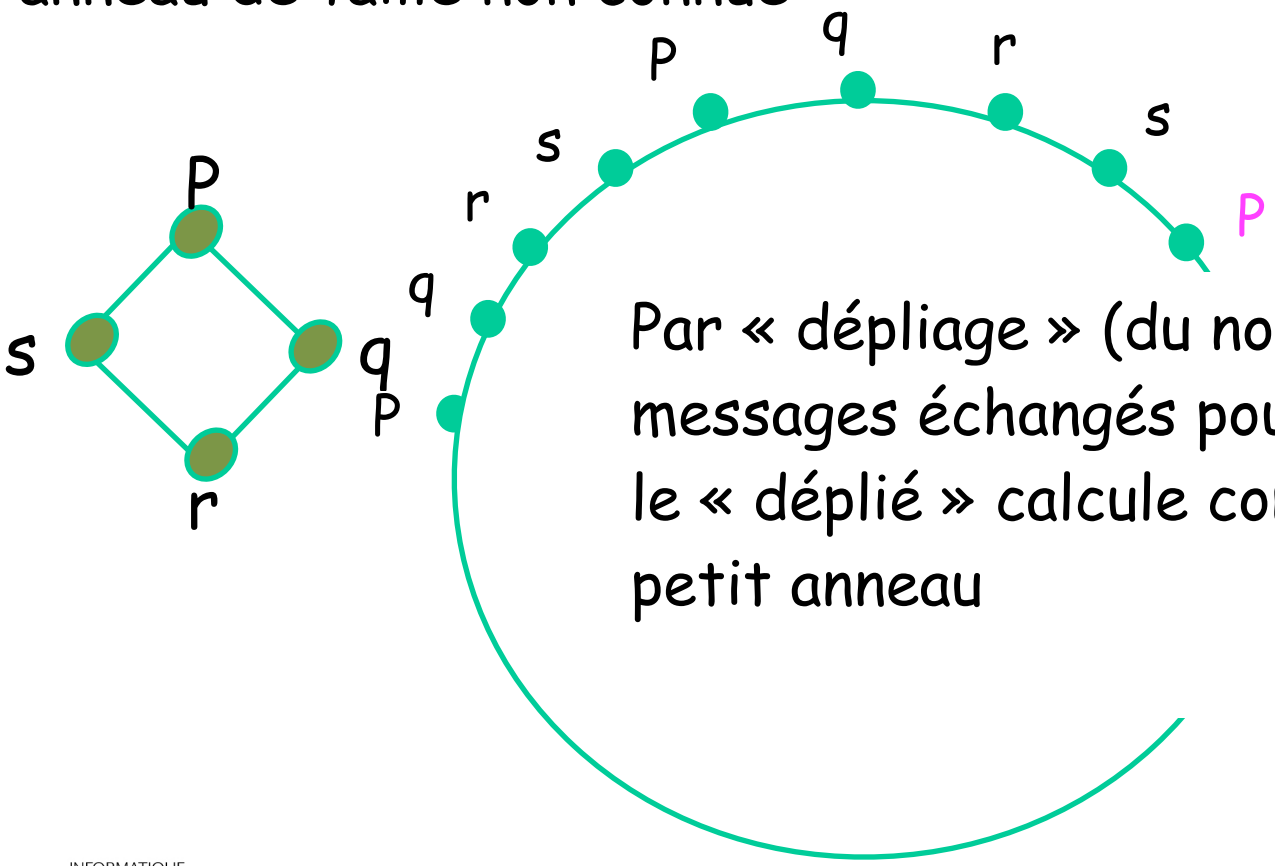
$$res_p = f(x_0, \cdots, x_{n-1})$$

Propriété: si la taille de l'anneau est connue toute fonction cyclique peut être calculée.

(On peut montrer que si la taille de l'anneau n'est pas connue il n'existe pas d'algorithme déterministe qui termine pour les processus pour calculer une fonction non constante)

Taille de l'anneau

Il n'y a pas d'algorithme déterministe qui termine pour les processus pour calculer une fonction non constante sur un anneau de taille non connue



Par « dépliage » (du nombre max de messages échangés pour terminer), le « déplié » calcule comme dans le petit anneau

Impossibilité…

Supposons qu'il existe un algorithme déterministe $\mathscr{A}$ pour calculer f non constante sur l'anneau:

il existe $x = (x_0, \cdots, x_k)$ et $y = (y_0, \cdots, y_k)$ tels que $f(x) \neq f(y)$

- Soit $\mathscr{A}$ un algorithme déterministe sur l'anneau qui calcule f.
- C_x le calcul de $\mathscr{A}$ pour x, C_y le calcul de $\mathscr{A}$ pour y. (C_x et C_y sont finis!)
- En « dépliant » l'anneau de façon à avoir un segment de longueur correspondant à C_x, on obtient un processus qui calculera $f(x)$ sur C_x. De même en « dépliant » l'anneau de façon à avoir un segment correspondant à C_y que l'on mettra après le segment précédent, on obtient un processus qui calculera $f(y)$.

- Dans l'anneau considéré on a deux processus qui obtiennent des résultats différents.

Remarque

On peut facilement calculer de façon déterministe le « ou » pour la « terminaison des messages »:

Pour chaque processus p: (x_p valeur initiale)

$res_p := x_p$ (valeur initiale)

Si x_p alors envoyer <> au suivant

Recevoir <>

Si $res_p = False$ alors $res_p := True$ send <> au suivant

Taille de l'anneau?

La connaissance de la taille de l'anneau permet l'élection (sous certaines conditions)

Elle permet aussi de calculer toute fonction symétrique.

On a:

Il n'existe pas d'algorithme déterministe (qui termine pour les processus) qui permette de calculer une fonction non constante si la taille de l'anneau n'est pas connue.

Il n'existe pas d'algorithme non-déterministe qui termine pour les processus pour calculer la taille d'un anneau qui soit correct avec une probabilité $r > 0$

Il n'existe pas d'algorithme non-déterministe qui termine pour les messages qui calcule la taille de l'anneau qui soit correct avec probabilité 1.

Terminaison pour les processus

Il n'existe pas d'algorithme non-déterministe qui termine pour les processus pour calculer la taille d'un anneau qui soit correct avec une probabilité $r > 0$

Supposons le contraire:

- $\mathscr{A}$ un algorithme, un anneau de taille n, un ρ-calcul C_n qui termine après une plus longue séquence de K messages et L pas de calculs avec $res_p = n$
- On « déplie » ce calcul en segment de longueur $K+1$, on peut obtenir une séquence de bits « correspondant » à ce ρ-calcul C_n et avoir un p_0 qui termine avec $res_p = n$: cette séquence a une probabilité de $2^{-(K+1)L}$
- En mettant bout à bout R telles séquences on a une probabilité de $(2^{-(K+1)L})^R$ que des processus terminent avec $res_p = n$ (qui n'est pas la taille de l'anneau!): $(2^{-(K+1)L})^R$peut être rendu aussi petit que l'on veut.

Terminaison pour les messages

Il n'existe pas d'algorithme qui termine pour les messages qui calcule la taille de l'anneau qui soit correct avec probabilité 1.

- $\mathcal{A}$ un algorithme, un anneau de taille n, un ρ-calcul C_n qui termine pour les messages et L le nombre de pas de calculs des processus
- Avec deux copies de l'anneau on construit un anneau de taille $2n$: chaque pas de C_n est fait concurremment par un processus du « premier » anneau et son correspondant dan le deuxième… Avec probabilité au moins $1 - 2^{-2nL}$, on n'a pas la bonne taille de l'anneau

Calcul de la taille de l'anneau?

D'après ce qui précède on peut au mieux avoir un algorithme non-déterministe qui termine pour les messages partiellement correct avec une probabilité inférieure à 1

Calcul de la taille de l'anneau

Monte-Carlo: le programme calcule la taille de l'anneau avec une certaine probabilité et termine pour les messages:

Principe:

- Estimation (croissante) locale cède la taille de l'anneau (est_p)
- Générer des jetons avec un label lab tiré dans un ensemble $\{1,\cdots,R\}$, avec une estimation de la taille de l'anneau initialisée à l'estimation locale est_p et un compteur du nombre de « hops »(initialisé à 1)

- Lancer un jeton
- pour toujours faire:
  - Si un jeton (l, est, h) arrive avec $est > est_p$ (estimation plus élevée que l'estimation locale)
    - changer l'estimation locale:
      - Si $h = est$ (le nombre de hops du jeton est atteint) alors $est_p := est + 1$ (la nouvelle estimation locale est l'estimation du jeton incrémenté)
        sinon $est_p := est$ (la nouvelle estimation locale est l'estimation du jeton)
    - relancer un nouveau jeton $((?\{1,\cdots,R\}, est_p, 1)$ (label tiré dans un ensemble $\{1,\cdots,R\}$ avec l'estimation locale et le nombre de hops à 1)
  - Si $est_p = est$ (c'est (c'est (peut-être) mon jeton qui revient)
    - Si $h < est$ le nombre de hop est inférieur à l'estimation du jeton, envoyer le jeton $(l, est, h+1)$
    - Sinon
      - Si $l = lab$ (c'est mon label) c'est probablement mon jeton et l'estimation locale est bonne
      - Si $l \neq lab$ (ce n'est pas mon label) mettre à jour mon estimation locale $est_p = est + 1$ et relancer un nouveau jeton
  - Jeter les jetons ayant des estimations inférieures

```
cons R      : integer     ;  (* Determines correctness probability *)

var  est_p  : integer ;
     lbl_p  : integer ;

begin est_p := 2 ; lbl_p := rand({1, ..., R}) ;
      send ⟨ test, est_p, lbl_p, 1 ⟩ to Next_p ;
      while true do
        begin receive ⟨ test, est, lbl, h ⟩ ;
              if est > est_p then     (* p's estimate must increase *)
                if est = h then     (* est is also too low *)
                  begin est_p := est + 1 ; lbl_p := rand({1, ..., R}) ;
                        send ⟨ test, est_p, lbl_p, 1 ⟩ to Next_p
                  end
                else     (* forward token and increase est_p *)
                  begin send ⟨ test, est, lbl, h + 1 ⟩ to Next_p ;
                        est_p := est ; lbl_p := rand({1, ..., R}) ;
                        send ⟨ test, est_p, lbl_p, 1 ⟩ to Next_p
                  end
              else if est = est_p then
                     if h < est then send ⟨ test, est, lbl, h + 1 ⟩ to Next_p
                     else (* This token has made est hops *)
                       if lbl ≠ lbl_p then
                         begin est_p := est + 1 ;
                               lbl_p := rand({1, ..., R}) ;
                               send ⟨ test, est_p, lbl_p, 1 ⟩ to Next_p
                         end
                       else skip     (* Possibly p's token returned *)
                   else (* est < est_p *) skip
        end
end
```

Calcul de la taille de l'anneau

- L'estimation de la taille est toujours inférieure ou égale à la taille.
- L'algorithme termine pour les messages: au plus $O(n^3)$ messages
- Quand l'algorithme termine (plus aucun message) tous les est_p sont égaux.

Résultat:

- Quand l'algorithme termine pour les messages: $est_p = n$ avec une probabilité au moins égale à $1 - (n-2)(1/R)^{n/2}$

Défaillances…

Défaillances

Défaillances de liens de communication:

- Duplications - pertes de messages
- Comment? Combien?
  - Pertes finies
  - Pertes équitables
  - …

Défaillances de processus

- Un processus
  - peut s'arrêter (crash)
  - Oublier d'envoyer/ recevoir des messages
  - Faire n'importe quoi
- (Nombre de processus corrects)
  - (Correct?)

- Dans ce qui suit on suppose que les processus sont tous corrects mais que des messages peuvent se perdre (sans restriction)

Attaque coordonnée

n processus avec communication avec pertes de messages

Spécification de l'attaque coordonnée:

- Chaque processus p a une valeur initiale $v_p \in \{0,1\}$,

  algorithme de décision: variable d_p initialement indéfinie qui ne peut être écrite qu'une seule fois (décision).
- On veut un algorithme de décision tel que:
  - Accord: si p et q décident alors il décident de la même valeur $d_p = d_q$
  - Terminaison; tous les processus doivent décider (un jour)
  - Validité:
    - Si tous le processus ont la valeur initiale 0, alors la seule décision possible est 0
    - Si tous les processus ont la valeur initiale 1 et si aucun message n'est perdu, la seule décision possible est 1
      - (Dans tous les autres cas la décision peut être quelconque 0 ou 1)

Attaque coordonnée: autre version

Pour deux:

- Deux armées A et B séparées par des ennemis ne peuvent communiquer que par envoi de messagers qui traversent les lignes ennemies et peuvent être capturés.
- L'armée A et l'armée B ont une idée initiale (v_p) sur le fait d'attaquer ou pas, mais pour que l'attaque réussisse il faut que les deux armées soient d'accord pour attaquer et décident toutes les deux d'attaquer. Les généraux de ces deux armées sont prudents et n'attaqueront que s'ils sont sûrs de réussir (et doivent attaquer dans ce cas): A ne peut attaquer que s'il sait que B a décidé d'attaquer idem pour B.
- Les deux généraux de ces armées sont de parfaits logiciens

Attaque coordonnée

- De la deuxième condition on déduit:
  - Supposons que A et B veulent attaquer
  - Pour être sûrs de gagner il faut que A et B sachent qu'ils veulent attaquer (prudence)
  - Mais cela ne suffit pas: il faut que A sache que B sait que A va attaquer (sinon B pourrait ne pas attaquer), idem pour B
  - Mais cela ne suffit pas: il faut que A sache que B sait que A sait que B sait que A veut attaquer (pourquoi?)
  - Etc… on peut continuer ainsi indéfiniment

  - (on a la même chose si A envoie un message à B et qu'on veut que A et B sachent que le message est arrivé et B et que A et B le sache: A envoie m à B, B envoie un ack, A envoie un ack de l'ack quand peut-on s'arrêter?)

Logique de la connaissance

P une propriété et S_A un état pour A, on dira que $(S_A \models P)$

Si P est vraie en S_A.

$K_A(P) \Rightarrow (S_A \models P)$,

On notera $K_{S_A}(P)$ (ou $K_A(P)$ si l'état est clair) - A « connaît » P- le fait que A sait (=peut déduire) que P est vrai dans cet état.

On suppose que A ne connaît que des choses vraies:

$K_A(P) \Rightarrow (S_A \models P)$.

(On pourra aussi supposer que A connaît tout de son état local)

(On peut définir ainsi une logique de la connaissance)

Par exemple de $K_A(P)$ et de $K_A(Q)$ on peut déduire $K_A(P \wedge Q)$

Augmenter la connaissance

$$K^1(P) = K(P) \equiv K_A(P) \wedge K_B(P)$$
$$K^{n+1}(P) = K(K^n(P))$$

- l'émission d'un message ne change pas la « connaissance »: (le message peut être perdu)
- La réception change la connaissance: si on avait pour X: $K_X(P)$, ont aura après réception du message de X en Ypour Y: $K_Y K_X(P)$
- Avec un « aller et retour » entre X et Y: on passe de $K_X(P)$ à $K_X(K(P))$ et après n allers et retours on aura $K_X K^n(P)$

Sur une échange de messages entre A et B

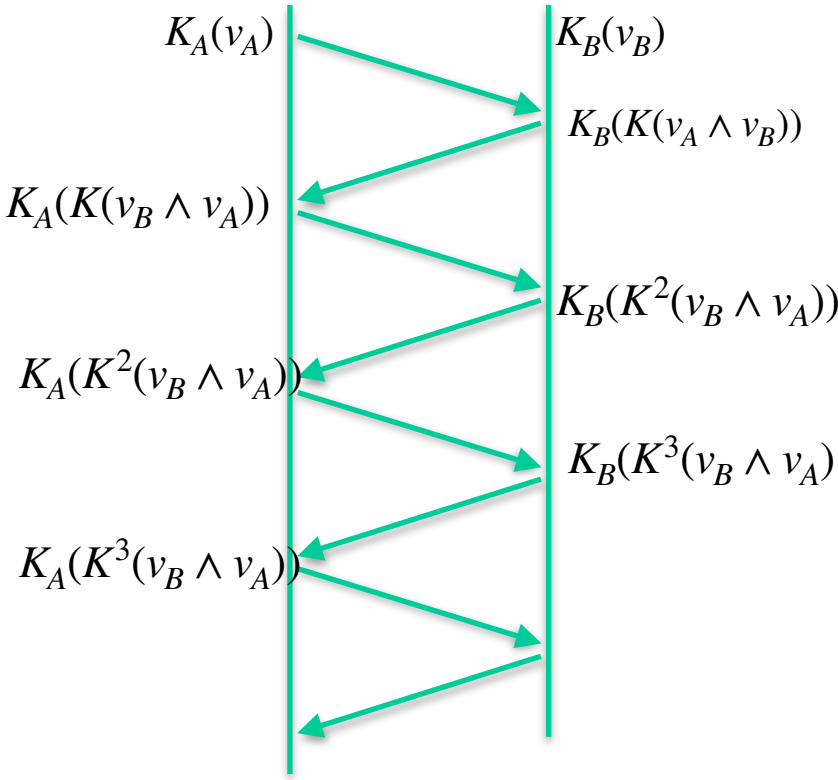
Initialement:

$$K_A(v_A) \text{ et } K_X(K(P))K_B(v_B)$$

Après réception du premier message de A vers B:

$$K_B(K_A(v_A)) \wedge K_B(v_B) = K_B(K(v_A \wedge v_B)$$

Plus généralement après réception du n-iéme message par B on aura $K_B(K^n(v_A \wedge v_B))$ (et par A $K_A(K^n(v_A \wedge v_B)))$



Notoriété

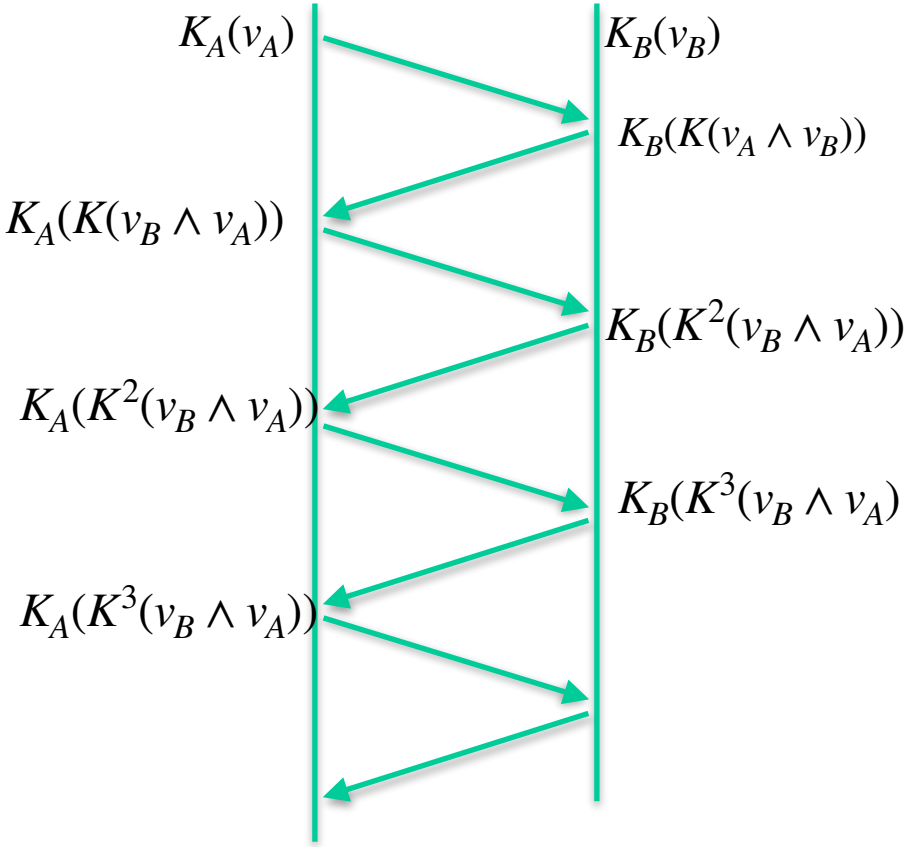
Pour l'attaque coordonnée:

Il faut que obtenir pour A et B que

$$\forall n : A \vDash K^n(P) \wedge B \vDash K^n(P)$$

(Notoriété de P)

Pourquoi?



Notoriété

Exemple du code de la route

Exemple de « Bagdad »

Exemple du condamné à mort