Cours2

Préliminaires: causalité, horloges et vecteurs d'horologe, liens de causalité, logique temporelle

Terminaison distribuée (début)

Causalité

E: ensemble des événements: événements locaux, et événements de communication (émission de m, réception de m). E_p est l'ensemble des événements de p et $E = \cup_{p \in \Pi} E_p$

Relation de causalité (ordre de Lamport)

$a \preceq b$ si et seulement si:

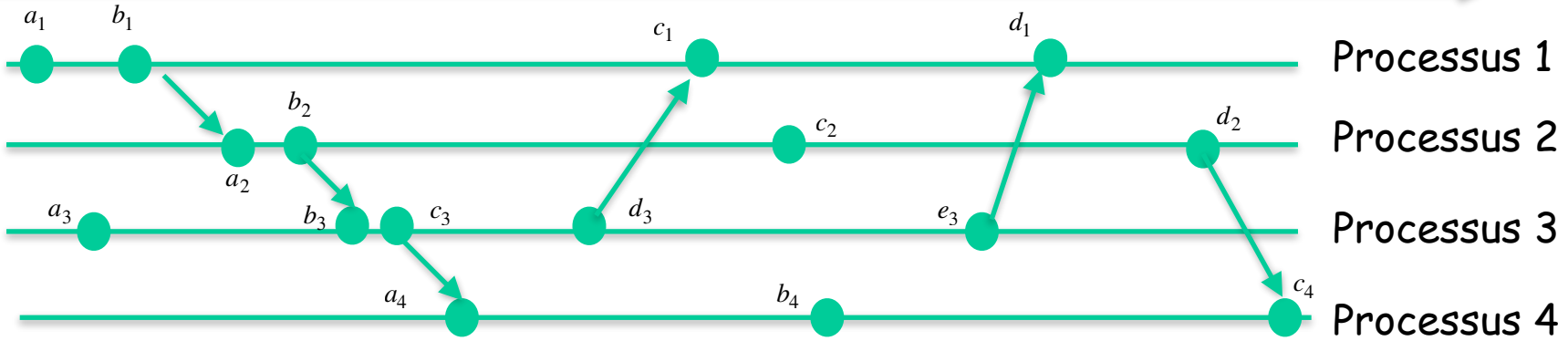
- $a = b$ (réflexivité)
- a et b sont sur le même processus a est séquentiellement avant b (séquentialité)
- a est l'émission du message m et b la réception de ce message (communications)
- Il existe c tel que $a \preceq c \wedge c \preceq b$ (fermeture transitive)

$\preceq$ est une relation d'ordre (partielle) (prouvez-le!)

Si a et b sont incomparables ($\neg(a \preceq b) \wedge \neg(b \preceq a)$ a et b sont concurrents

Si $a \prec b$ alors quelque soit l'exécution l'événement a se produit (pour le temps) avant l'évènement b: a peut être une cause de b mais b ne peut pas être une cause de a (on ne remonte pas le temps!)

Temps



$a_1 b_1 a_3 a_2 a_3 b_2 c_2 d_2 b_3 c_3 a_4 b_4 d_3 e_3 c_1 c_4 d_1$

$a_1 b_1 a_2 b_2 c_2 d_2 a_3 b_1 c_3 d_3 e_3 c_1 d_1 a_4 b_4 c_4$

...

$a_2 \preceq d_1$

a_4 et d_1 concurrents

Remarque: Si $a_1, \cdots, a_k$ une exécution et σ une permutation soit $a_{\sigma(1)}, \ldots a_{\sigma(k)}$ telle que $a_{\sigma(i)} \preceq a_{\sigma(j)} \Rightarrow i \leq j$ alors $a_{\sigma(1)}, \ldots a_{\sigma(k)}$ est aussi une exécution

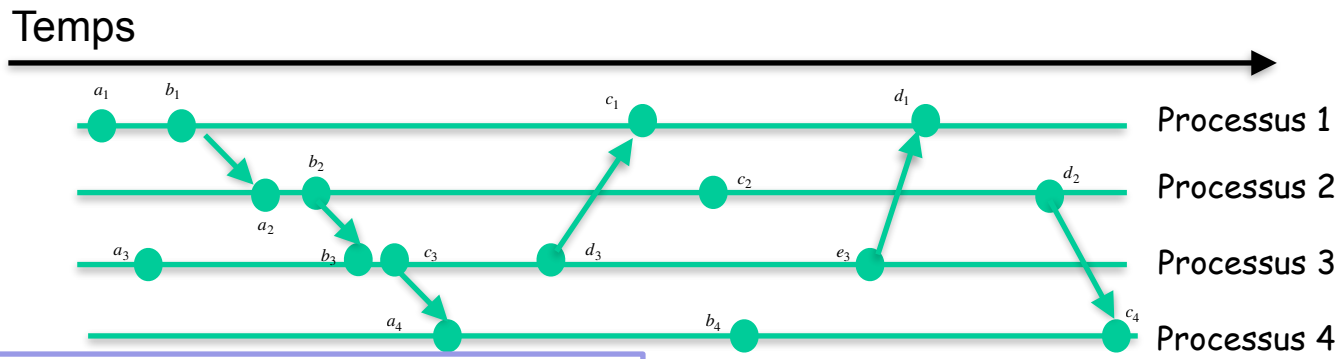
Causalité: horloge logique

Algorithme:

- Chaque processus p maintient un compteur c_p (initialisé à 0)
- À chaque action locale ou d'émission: $c_p := c_p + 1$
- Quand p envoie un message m il envoie $< m, c_p >$ (piggybacking)
- Quand p reçoit un message $< m, v >$ il met à jour son compteur: $c_p := \max(c_p, v) + 1$

$c(e)$ la valeur du compteur pour l'événement e:

$a \prec b \Rightarrow c(a) < c(b)$



$< c(e), p >$ pour l'ordre lexicographique:

$(c(e), p) < (c(e'), p' > \Leftrightarrow c(e) < c(e') \vee (c(e) = c(e') \wedge p < p')$

est une horloge logique les événements sont totalement ordonnés

Causalité

Peut-on avoir mieux?

$$c(a) \leq c(b) \Leftrightarrow a \preceq b$$

Horloge vectorielle

Mécanisme d'estampilles (comme pour les horloges de Lamport) à l'événement a on associe un vecteur est, tel que $est[p]$ est l'estimation par p de la valeur de l'horloge de Lamport

Règles de mises à jour:

- Événement interne pour p: $est[p] := est[p] + 1$
- Emission de v: $est[p] := est[p] + 1$ et envoi de $< v, est >$
- Réception de $< v, e >$:

$\forall q \neq p \in \Pi : est[q] := max(est[q], e[q])$ et $est[p] := est[p] + 1$

Horloges vectorielles

Comparaison

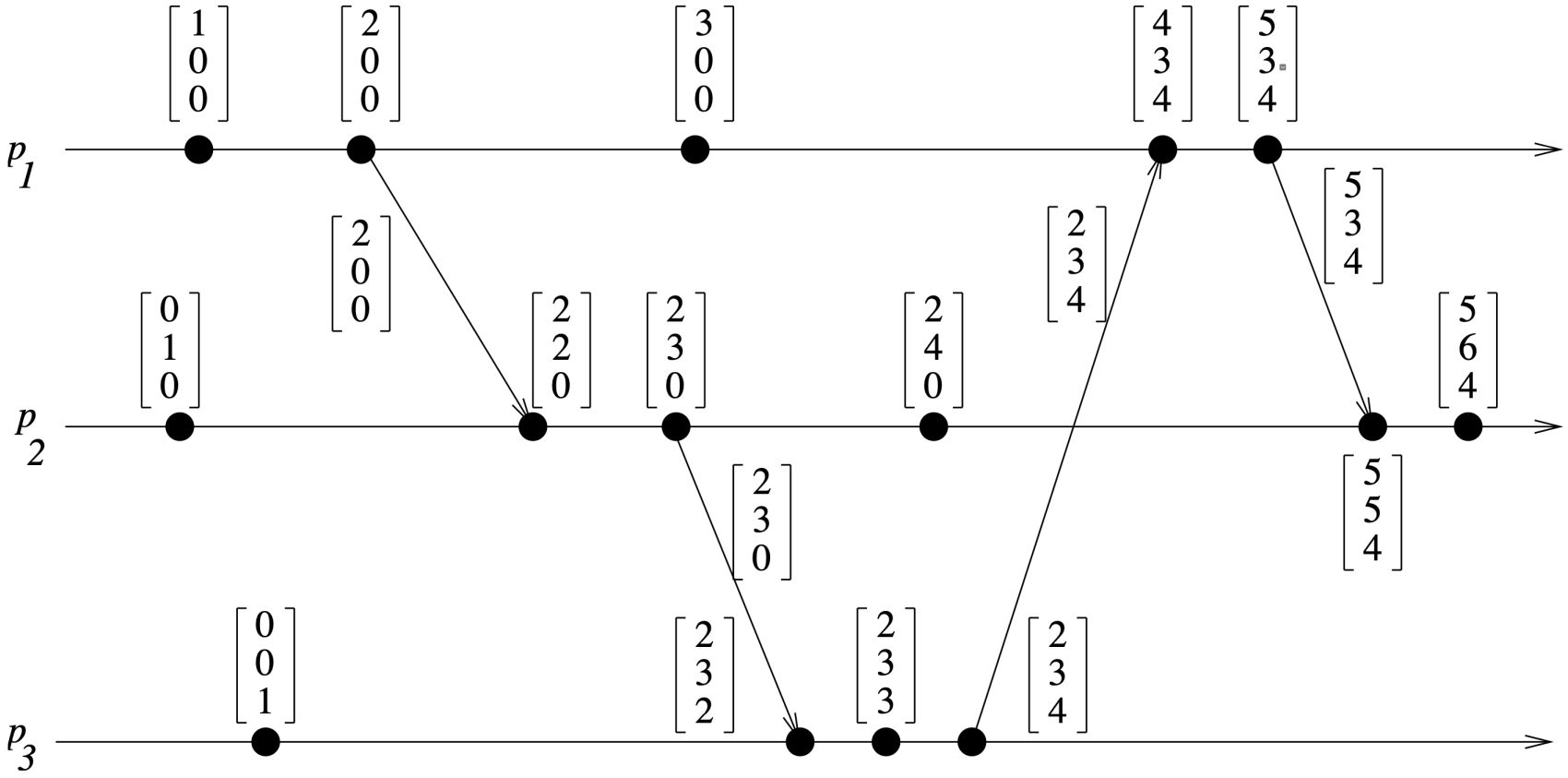
- $e < e'$ (ordre sur les vecteurs) si et seulement si
  - $\forall i : e[i] \leq e'[i] \wedge \exists j : e[j] < e'[j]$

Dans ces conditions:

Pour a et b deux événements (en notant $e(a)$ la valeur de est pour l'événement a) on a:

- $a \prec b \Leftrightarrow e(a) < e(b)$

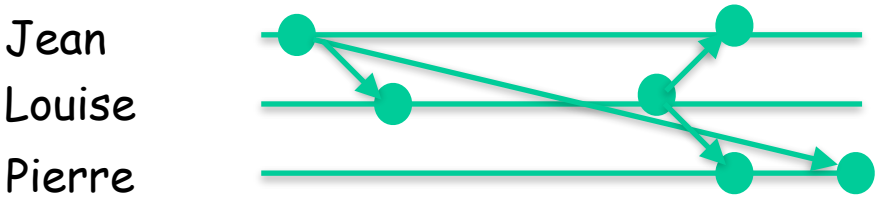
Exemple


p1
p2
p3
[1 0 0]
[2 0 0]
[3 0 0]
[4 3 4]
[5 3 4]
[2 0 0]
[5 3 4]
[0 1 0]
[2 2 0]
[2 3 0]
[2 4 0]
[2 3 4]
[5 6 4]
[5 5 4]
[2 3 0]
[0 0 1]
[2 3 2]
[2 3 3]
[2 3 4]

Causalité…

Ordre causal:

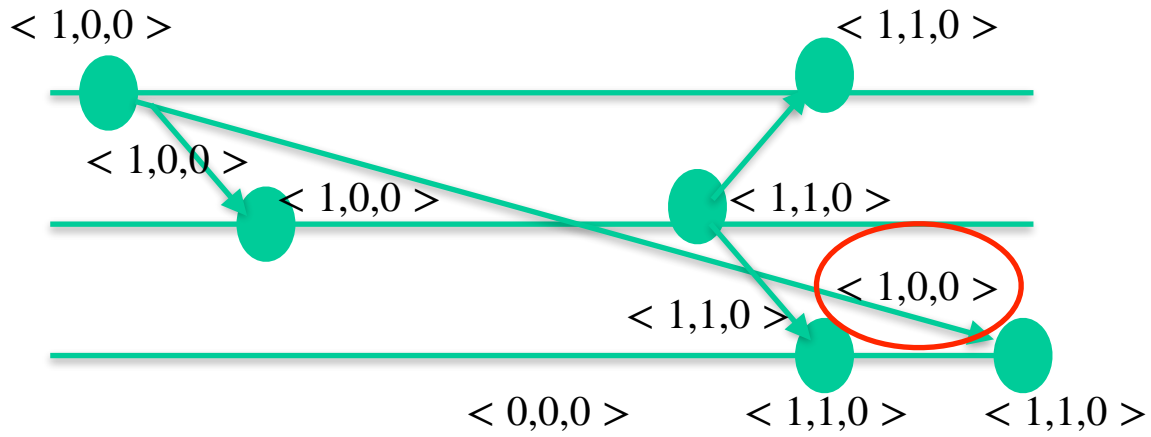
- Jean envoie un message à Pierre et à Louise, Louise répond à Jean et à Pierre.
- (Même si le canal est FIFO) Il est possible que Pierre reçoive la réponse de Louise avant d'avoir le message de Jean !
- Un canal et causal s'il délivre les messages suivant l'ordre causal: pour a (reps. b) émission et a' (reps. b') réception correspondante sur le même processus si $a \prec b$ alors $a' \prec b'$



Non causal

Diffusion causale

On peut utiliser les horloges vectorielles pour réaliser une diffusion causale (les messages sont délivrés suivant l'ordre causal):



Horloges matricielles

Les horloges vectorielles permettent de « dater » les événements suivant la relation $\preceq$. Elles ne permettent pas à un processus de savoir la « perception » des autres processus concernant le temps logique.

Exemple:

pour réaliser un ordre causal sur les réceptions, un processus doit « retarder » la délivrance du message jusqu'à être sûr que le passé causal est complet et qu'il a déjà délivré les messages qui sont dans le « passé » de ce message.

Algorithme:

- Une estampille est est une matrice $n \times n$, $est[p]$ est l'horloge vectorielle (classique) de $pest[q]$ est l'estimation de l'horloge vectorielle de q.
  - Événement local: $est[p,p] := est[p,p] + 1$
  - Chaque message contient est
  - Sur réception d'un message de q contenant l'estampille e,
    - $\forall i, j : est[i,j] := \max(est[i,j], e[i,j])$
    - $est[p,p] := est[p,p] + 1$

Exercice: émuler un canal causal en utilisant des horloges matricielles.

Propriétés des liens (canaux) de communication…

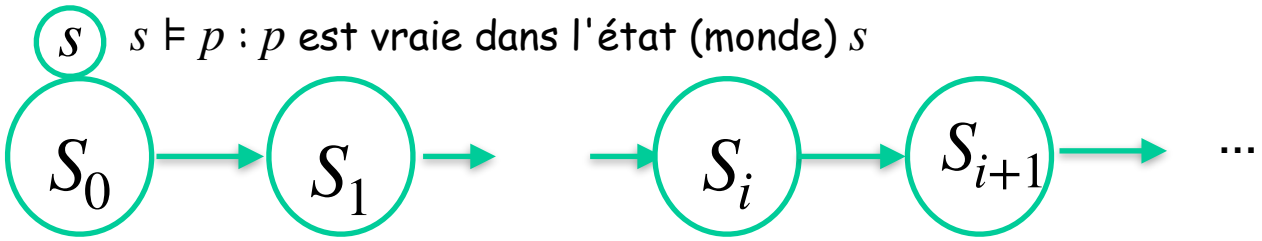
On suppose qu'une infinité de messages sont émis

- Lien fiable: sans pertes de messages (sur le lien de p à q)
- Lien finalement (eventually) mort: il existe un temps t à partir duquel tous les messages sont perdus.
- Lien infini : une infinité de messages est reçue
- Lien finalement fiable: il existe un temps t à partir duquel plus aucun message ne sera perdu.
- (Lien juste: si à partir du temps t, m est émis toujours, alors m sera reçu)
- Lien équitable (fair) : un message émis infiniment souvent sera reçu.

Exercices:

- Comparer ces diverses propriétés.
- Internet? UDP versus TCP
- Comment émuler un lien sans perte à partir d'un lien équitable?
  - Implémentation de TCP?
- Décrire ces différentes propriétés par des formules de logique temporelle

Logique temporelle (linéaire)



$w = s_0, s_1, s_2, s_3, \ldots$ et
$w^i = s_i, s_{i+1}, \ldots$

Formules à partir des variables propositionnelles) et de $next, U, \Box, \Diamond$

$w \vDash p \equiv s_0 \vDash p$

$w \vDash next(\phi) \equiv w^1 \vDash \phi$

$w \vDash \phi U \psi \equiv \exists i : w^i \vDash \psi$ et
$\forall 0 \leq k < i : w^k \vDash \phi$

$w \vDash \Box \phi \equiv \forall i : w^i \vDash \phi$

$w \vDash \Diamond \phi \equiv \exists i : w^i \vDash \phi$

LTL

Exemples:

$true U \phi \equiv \Diamond \phi$

$\neg \Diamond \neg \phi \equiv \Box \phi$

$\Box \Diamond p$: p est vraie infiniment souvent

$\Diamond \Box p$: p sera vrai un jour pour toujours

....

Lien (canal) de communication…

On suppose qu'une infinité de messages sont émis ($\Box \Diamond \exists m : sent(m)$)

- Lien fiable: sans pertes de messages (sur le lien de p à q)
- Lien finalement (eventually) mort: il existe un temps t à partir duquel tous les messages sont perdus.

- Lien infini : une infinité de messages est reçue:
- Lien finalement fiable: il existe un temps t à partir duquel aucun message ne sera perdu.

- (Lien juste: si à partir du temps t m est émis toujours, alors m sera reçu)

- Lien équitable (fair) : un message émis infiniment souvent sera reçu.

Exercices:

- Comparer ces diverses propriétés.
- Internet? UDP versus TCP
- Comment émuler un lien sans perte à partir d'un lien équitable?
  - Implémentation de TCP?
- Décrire ces différentes propriétés par des formules de logique temporelle

Partie I:

Construire un état global

Algorithmique distribuée

Déterminer un état global du système à partir des données locales:

- Exemple de la terminaison distribuée

Un exemple: la terminaison distribuée

Le problème:

- des employés avec téléphone (seul moyen de communiquer avec les autres)
- du travail à faire: chaque employé a initialement des dossiers à traiter, au cours du traitement il peut donner du travail aux autres (il peut aussi en recevoir)
- quand tout le travail est fini l'employé peut rentrer chez lui.

- Terminaison distribuée
  - un employé sait qu'il a terminé localement son travail (terminaison locale) mais cela peut être temporaire : il peut de nouveau avoir du travail donné par un de ses collègues
  - comment savoir que le travail est (de façon globale) terminé (terminaison distribuée)?

trouver un algorithme qui permet aux employés de tester si le travail est (globalement) terminé.

Détecter la terminaison globale à partir de la terminaison locale

Code

TL_p: p a localement terminé

Processus p:

(initialement $TL_p = false$)

pour toujours faire: (initialement $TL_p = false$)

$\vee(\ \neg TL_p \rightarrow$ travail local)

$\vee(\neg TL_p \rightarrow TL_p = true)$

$\vee(\neg TL_p \rightarrow$ donner travail à $q)$

$\vee($recevoir du travail de $q \rightarrow TL_p = False)$

Terminaison distribuée

TL_p est un prédicat local qui indique que le travail local de p est terminé

TG est un prédicat qui indique terminaison globale:

- on a $TG = \bigwedge_{p \in \Pi} TL_p$
- mais on a en plus:
  - <u>stabilité</u> de TG:
    - si à l'instant t, TG est vrai pour tout $t' \geq t$, TG est vrai

      $(TG^t \Rightarrow \forall t' \geq t : TG^{t'})$

<u>Notation</u>: En supposant un temps (fictif) et P un prédicat P^t est la valeur de ce prédicat à l'instant t

P est stable: $P^t \Rightarrow \forall t' \geq t : P^{t'}$

Détection P?

(Pour la terminaison distribuée $P = TG = \bigwedge_{p \in \Pi} TL_p$)

Algorithme de détection de P:

- ajouter du code (sans « modifier » le code initial) de façon à calculer un prédicat A tel que
  - sûreté (safety): $A^t \Rightarrow P^t$ (si on détecte P alors P est vrai)
  - vivacité (liveness): $(\exists t : P^t) \Rightarrow (\exists t' \geq t : A^{t'})$ (si un jour P est vrai alors un jour P sera détecté)

- (on suppose que P (et A) sont stables)

Exercice

- Un prédicat P est stable si:
  - $P^t \Rightarrow \forall t' \geq t : P^{t'}$
- Pour un programme P, une propriété P est un invariant (inductif) si et seulement si
  - initialement P est vraie et
  - Pour chaque pas de calcul si P est vraie P reste vraie après le pas de calcul.

  Dans ce cas, P est toujours vrai ($\Box P$)
- Exercice:
  - Comment montrer que TG est stable?
  - Quelle est la différence entre un invariant et une propriété stable.

Invariant…

Dans un programme séquentiel

- boucle: `while(B) Ins;`
  - Pour prouver que si P est vraie alors après l'exécution de la boucle Q est vraie
    - Inv invariant de la boucle
      - $P \Rightarrow Inv$ (Inv est au début de la boucle)
      - Si $Inv \wedge \neg B$ alors après exécution de Ins, Inv est encore vraie
    - $(Inv \wedge \neg B) \Rightarrow Q$ (si on sort de la boucle Q est vrai)
  - (Correction partielle: on ne prouve pas que la boucle termine (safety))
  - (Terminaison: si P est vraie alors il existe une itération de la boucle telle que $\neg B$ sera vraie (liveness))
  - (Correction totale= correction partielle + terminaison)
  - (Intuitivement $\Box P$ correspond à la safety (invariance P est vrai dans tous les états), $\Diamond P$ correspond à la liveness (il existe un état pour lequel P sera vrai))

Invariant…

Invariant:

- en général prouver qu'un programme vérifie une propriété P on vérifie que toutes les exécutions vérifient P
- Pour cela on montre en général que P est un invariant (inductif), c'est à dire que pour le programme on a $\Box P$:
  - dans tous les états initiaux P est vrai
  - Si P est vrai dans un état du système S alors P est vrai dans tout état suivant (état dans $next(S)$)
- Dans la pratique, on veut montrer Q une propriété qui n'est pas un invariant. Pour cela on trouvera un invariant INV
  - Dans tous les états initiaux INV est vrai
  - Si INV est vrai dans un état du système S alors INV est vrai dans tout état $next(S)$
  - $INV \Rightarrow Q$
- (On peut avoir une propriété P vraie dans tous les états du système sans être un invariant de cette sorte)

TLA+

```
THEOREM Spec=>[]Inv
<1>1. Init => Inv
<1>2. Inv /\ [Next]_vars => Inv'
<1>3. QED
   BY <1>1,<1>2
```

```
Correct = ...          \* The invariant you really
                       \*want to prove
Inv = ... /\ Correct \* the inductive invariant

THEOREM Spec=>[]Correct
 <1>1. Init => Inv
 <1>2. Inv /\ [Next]_vars => Inv'
 <1>3. Inv => Correct
 <1>4. QED
     BY <1>1, <1>2, <1>3
```

Invariant et induction…

Remarque: induction sur $\preceq$

- L'ordre de Lamport est bien fondé (il n'existe pas de séquence infinie (x_n) telle que $(\forall n : x_{n+1} \preceq x_n)$: l'induction s'applique
- Pour montrer P pour une exécution (= montrer que $\Box P$ = montrer que P est vrai dans tous les états)
  - On peut montrer:
    - P pour tous les éléments minimaux de l'exécution
    - et Si
      - P est vrai pour tout x tel que $x \prec a$ implique P est vrai en a
    - alors $\Box P$ pour l'exécution

Détection de la terminaison

Trouver un algorithme?