CENTRE NATIONAL
DE LA RECHERCHE
SCIENTIFIQUE

UNIVERSITÉ
NICE
SOPHIA
ANTIPOLIS

Programmation synchrone : propriétés dans l'instant

C. ANDRÉ

I3S

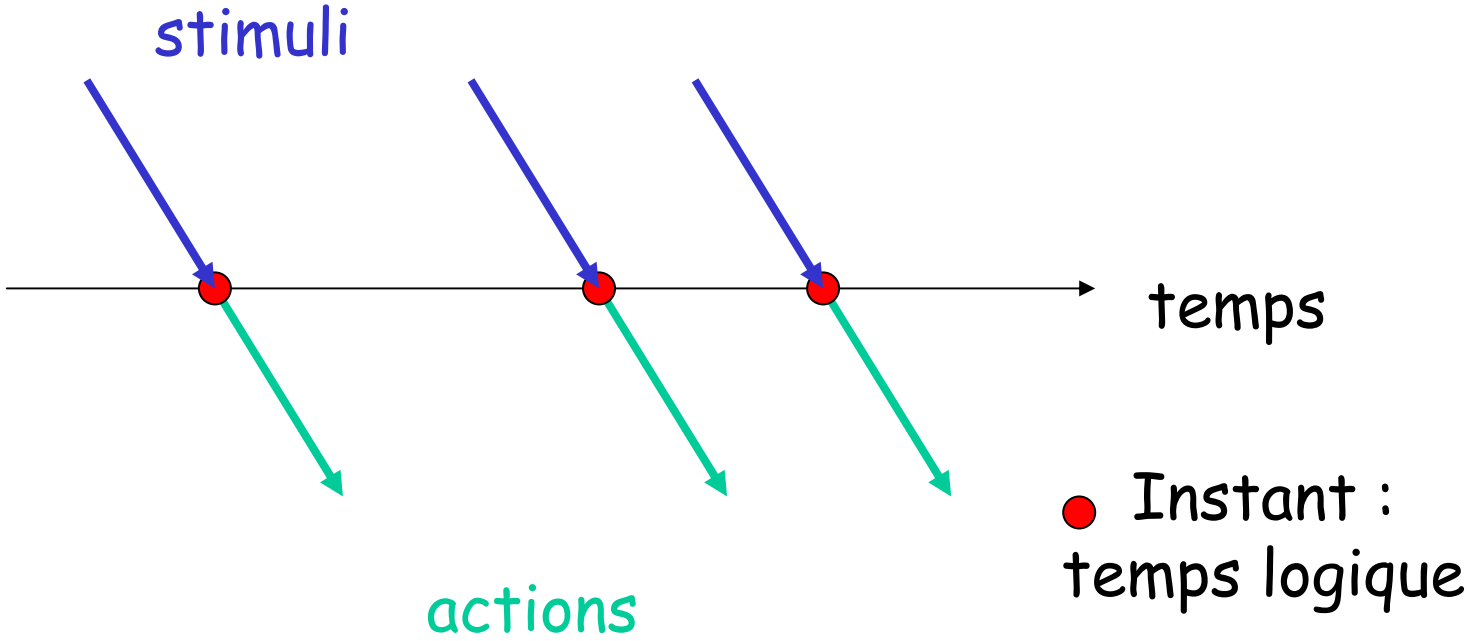
R. DE SIMONE

INRIA

Plan

- Comportement inter/intra instants
- Sémantique constructive
- Réfutation de causalité
- Exemple zFIFO
- Conclusion

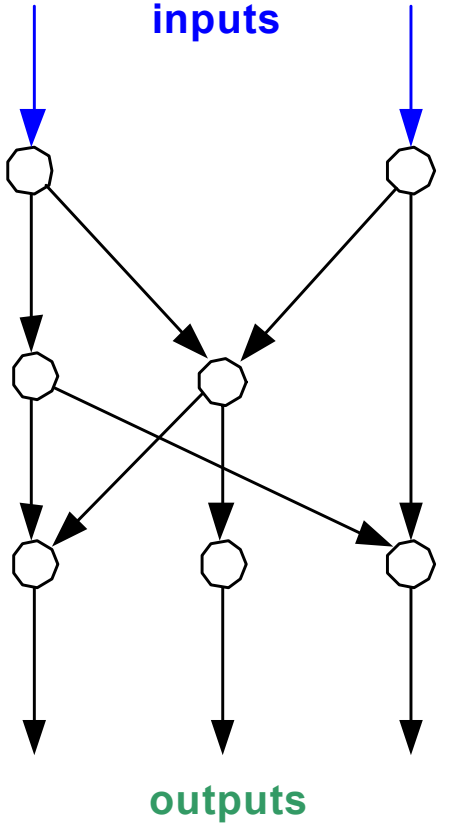
Comportement en temps logique



Histoire d'entrée → Histoire de sortie

Propriétés temporelles inter-instants

Comportement dans l'instant

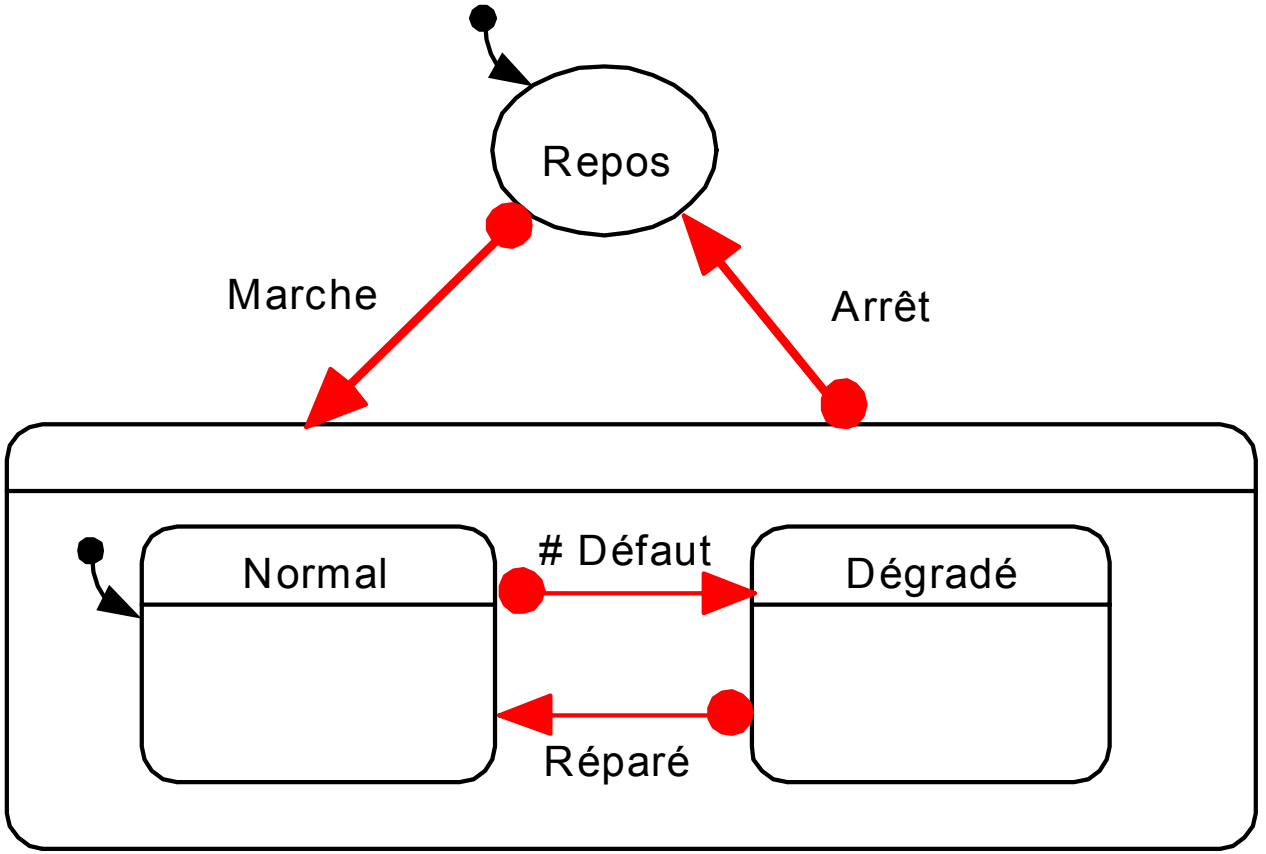


Dans l'instant :

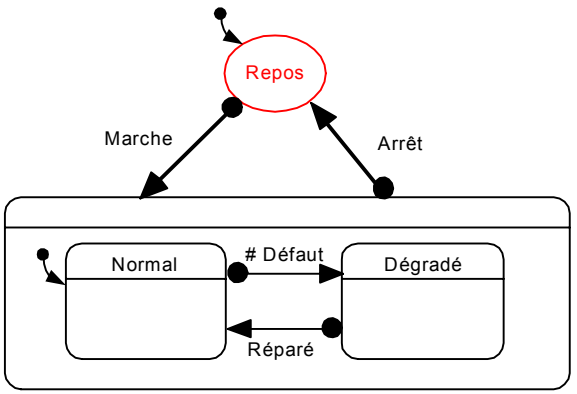
Ordre partiel des actions

Respect des causalités

Comportements réactifs complexes

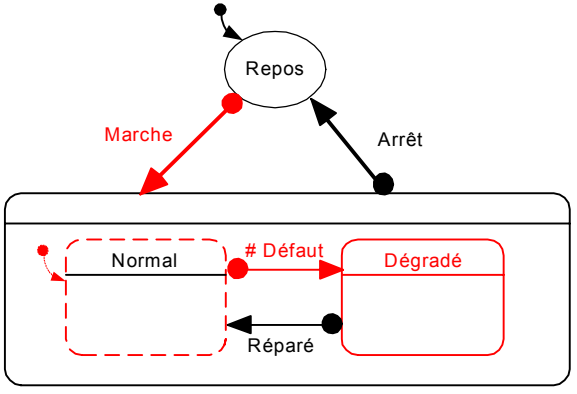


Squelette du prog. Esterel



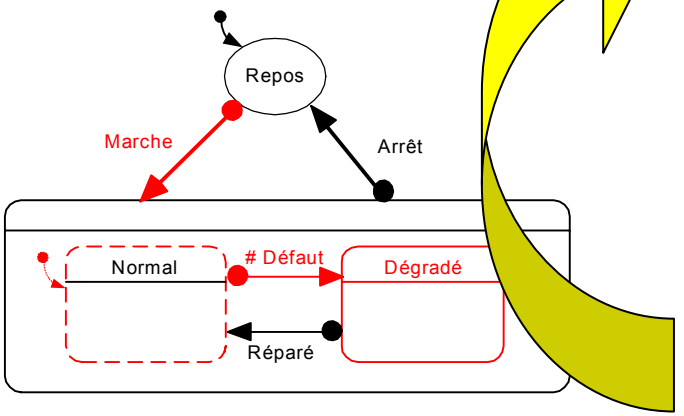

```
signal goto1, goto2, goto3 in
   emit goto1; %état initial
   loop % faire sans fin
      present goto1 then
        await Marche;emit goto2
      end present
     ||
      present goto2 then

      end present                ↵
     ||
      present goto3 then

      end present                ↵
     ||
      pause                      ↵
   end loop
end signal
```

Calcul de l'état suivant (1)



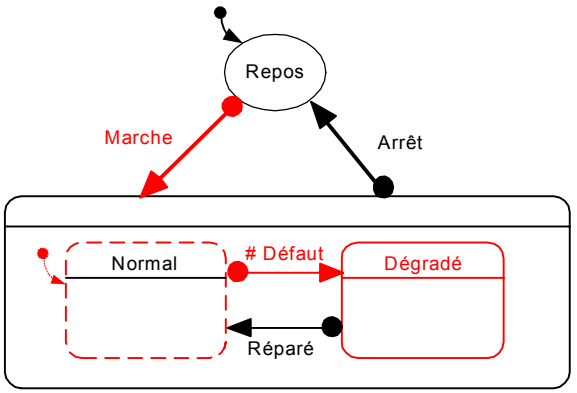

```
signal goto1, goto2, goto3 in
   emit goto1; %état initial
   loop % faire sans fin
      await Marche;emit goto2
   ||
   ||
   ||
   end loop
end signal
```

Calcul de l'état suivant (1)



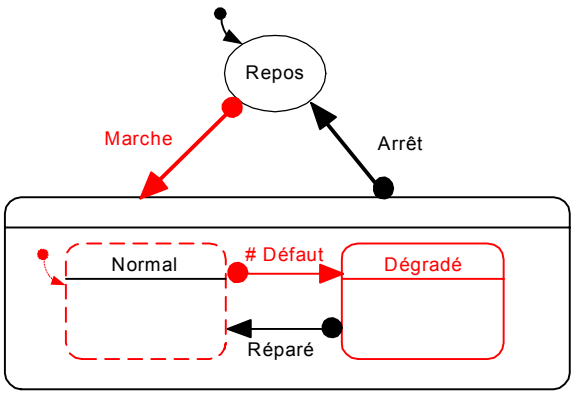

```
signal goto1, goto2, goto3 in
   emit goto1; %état initial
   loop % faire sans fin

      await Marche;emit goto2

   ||

   ||

   ||

   end loop
end signal
```

Calcul de l'état suivant (1)



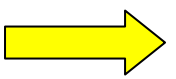

```
signal goto1, goto2, goto3 in
   emit goto1; %état initial
   loop % faire sans fin
      →
       await Marche;emit goto2

      ||
      →
      ||
      →
      ||
      →
   end loop
end signal
```

Calcul de l'état suivant (2)

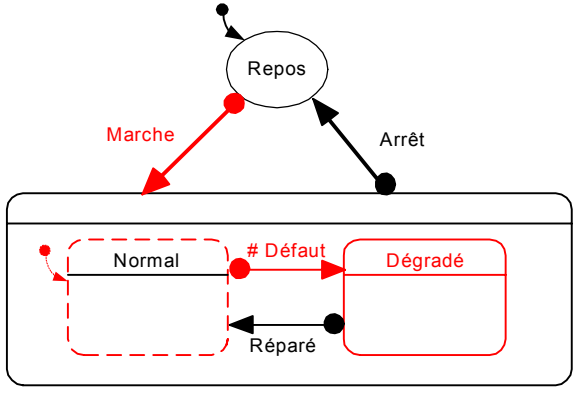



```
signal goto1, goto2, goto3 in
   emit goto1; %état initial
   loop % faire sans fin
      →
   ||
   present goto2 then
```

```
      end present
      ||
      →
      ||
       pause
   end loop
end signal
```

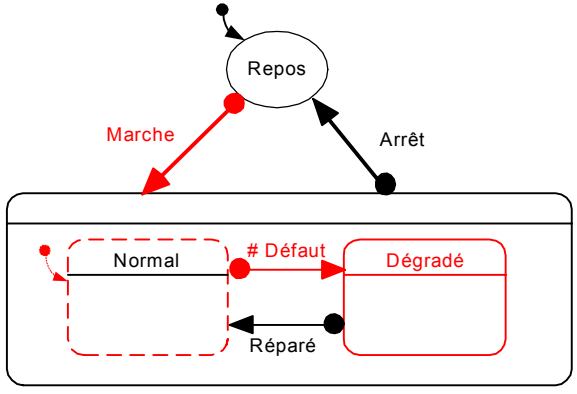


Calcul de l'état suivant (2)



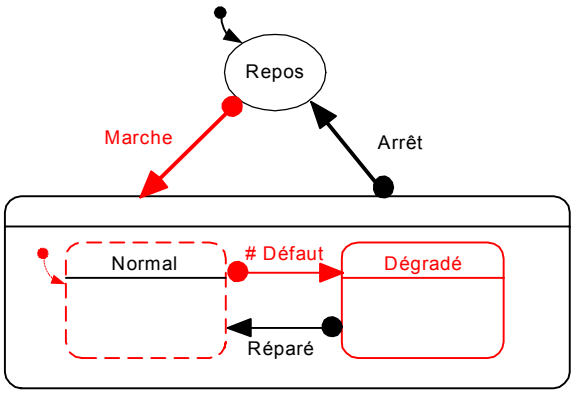

```
signal goto1, goto2, goto3 in
   emit goto1; %état initial
   loop % faire sans fin
      →
   ||
      present goto2 then
        abort
          run Normal
        when
          case Arrêt do emit goto1
          case immediate Défaut do emit goto3
        end abort
      end present
   ||
      →
   ||
      pause
   end loop
end signal
```

Calcul de l'état suivant (2)



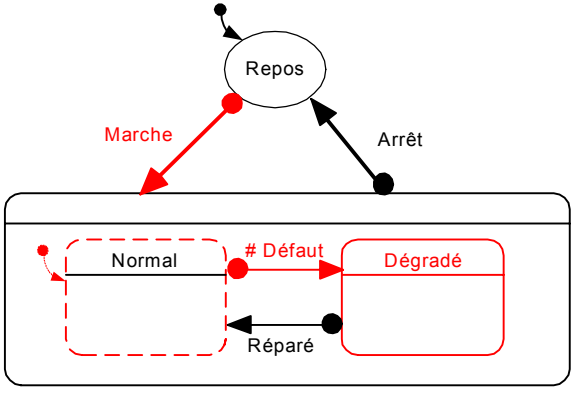

```
signal goto1, goto2, goto3 in
   emit goto1; %état initial
   loop % faire sans fin
      →
   ||
      present goto2 then
        abort
          run Normal
        when
          case Arrêt do emit goto1
          case immediate Défaut do emit goto3
        end abort
      end present                              ↵
      ||
       →
      ||
       pause
   end loop
end signal
```

Calcul de l'état suivant (3)



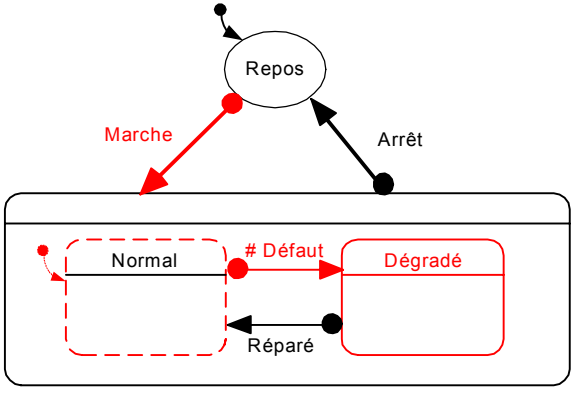

```
signal goto1, goto2, goto3 in
   emit goto1; %état initial
   loop % faire sans fin
      →
     ||
                                    ↵
     ||
      present goto3 then
           abort
                 run Dégradé
           when
              case Arrêt do emit goto1
              case Réparé do emit goto2
           end abort
        end present
     ||
      pause
   end loop
end signal
```

Calcul de l'état suivant (3)



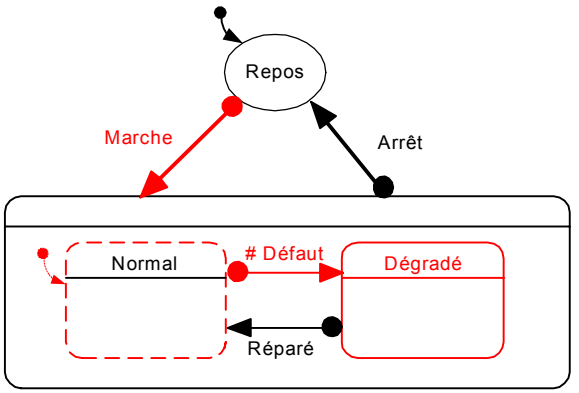

```
signal goto1, goto2, goto3 in
   emit goto1; %état initial
   loop % faire sans fin
      →
     ||
                                    ↵
     ||
      present goto3 then
           abort
                 run Dégradé
           when
              case Arrêt do emit goto1
              case Réparé do emit goto2
           end abort
        end present
     ||
      pause
   end loop
end signal
```

Calcul de l'état suivant (4)



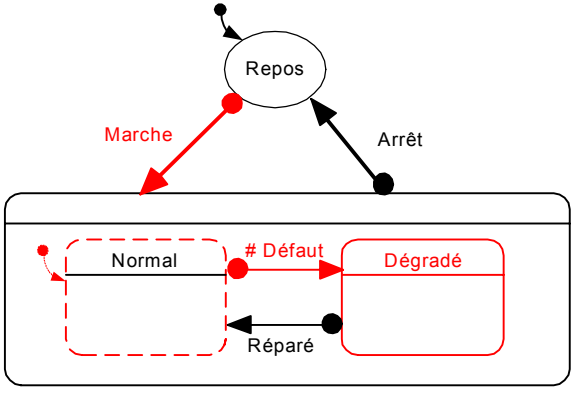

```
signal goto1, goto2, goto3 in
   emit goto1; %état initial
   loop % faire s      fin
      present got     th
           await Marche  emit goto2
       end present
     ||

     ||
       present goto3 then
           abort
                 run Dégradé
           when
              case Arrêt do emit goto1
              case Réparé do emit goto2
           end abort
        end present
     ||
       pause
   end loop
end signal
```

Calcul de l'état suivant (4)




```
signal goto1, goto2, goto3 in
   emit goto1; %état initial
   loop % faire sans fin
      present goto1 then
           await Marche; emit goto2
        end present
     ||
     ||
      present goto3 then
           abort
                 run Dégradé
           when
              case Arrêt do emit goto1
              case Réparé do emit goto2
           end abort
        end present
     ||
      pause
   end loop
end signal
```

Calcul de l'état suivant (5)




```
signal goto1, goto2, goto3 in
   emit goto1; %état initial
   loop % faire sans fin
                                                  ↵
      ||
                                                  ↵
      ||
       present goto3 then
            abort
                  run Dégradé
            when
               case Arrêt do emit goto1
               case Réparé do emit goto2
            end abort
         end present
      ||
                                                  ↵
   end loop
end signal
```

Sémantique constructive

- Pour trouver l'état (présent ou absent) et éventuellement la valeur de chaque signal de sortie ou local, on ne propage que des faits.
- Si on arrive à construire une solution, pour tout état et tout événement d'entrée, le programme est dit correct pour cette sémantique, sinon il est rejeté.

Problème

- Peut-on savoir si un ordre donné entre actions est respecté dans les instants?
- Bien-sûr, nous ne voulons pas regarder le code généré!
- Notre réponse :
  - Utiliser le compilateur lui-même
  - En instrumentant le programme
  - Pour mettre en œuvre une technique de réfutation de causalité

Réfutation de causalité (1)

- Soient a et b deux signaux dans un programme constructif P
- Soit $\sqsubset$ une relation de dépendance dans l'instant.
- Soit P' le programme obtenu à partir de P et instrumenté de façon à imposer $a \sqsubset b$.

Refutation de causalité (2)

- L'instrumentation consiste à garder les émissions de b par un test passant si l'état de présence de a est connu lors de l'émission de b.
- Une solution : remplacer
  - emit b
- par
  - present a then emit b else emit b end

Réfutation de causalité (3)

- Si P' n'est pas constructif, alors que P l'était c'est qu'il existe au moins une réaction contenant a et b dans une dépendance inverse de celle imposée.
- Si P' reste constructif, c'est que dans P, pour toute transition impliquant la partie de code modifiée
  - a et b ne sont pas simultanément présents
  - a et b sont simultanément présents et $a \sqsubset b$
  - a et b sont simultanément présents et indépendants

Application :zFIFO

- Une zFIFO = (0 fall-through time FIFO queue) est un tableau de cellules mémoire avec possibilités de décalage.
- On peut effectuer Put et Get simultanément. Ceci permet des dépôts et retraits simultanés même sur une zFIFO vide ou pleine.
- Implémentée en SyncCharts et Esterel (cf. le papier dans les actes).

zFIFO (2)

- La solution est composée de modules communicants.
- Ces modules expriment le comportement d'une cellule.
- Chaque cellule ne connaît que ses proches voisines.
- Est-ce que cette composition respecte bien la discipline FIFO?
- Cette question revient à monter que les lectures/écritures dans un même instant, sont correctement ordonnées.

zFIFO (3)

Principe de la preuve

1. Établir des propriétés de sûreté (vraies à chaque instant). Ceci est classique et utilise Xeve.
   Ex : $r_j \sqsubset w_j$ si écriture/lecture simultanées
2. Établir des propriétés par analyse causale (dans l'instant)

On veut montrer que $r_{j+1} \sqsubset w_j$

zFIFO (4)

- En fait, on établit une propriété plus forte $r_{j+1} \sqsubset r_j$

1. Notre solution, à vérifier, est constructive.
2. On instrumente notre programme en imposant $r_{j+1} \sqsubset r_j$. Ce programme programme est constructif.
3. On instrumente notre programme en imposant $r_j \sqsubset r_{j+1}$. Ce programme programme n'est pas constructif.
4. On en déduit que $r_{j+1} \sqsubset r_j$ est vrai pour notre programme.

Conclusion

- Introduction à la sémantique constructive,
- Analyse de propriétés DANS un instant,
- Réfutation de causalité
- Étude de la zFIFO
- Moyen de sérialiser des actions?
- Propriétés autres qu'ordre partiel?