Modélisation du comportement des systèmes 4

Frédéric Boulanger

Novembre 2019

Rappels

Acteurs, modèles de calcul

- modélisation par acteurs : boîtes noires + ports + relations
- syntaxe abstraite uniforme
- modèle de calcul = règles d'interprétation d'un modèle

Modèles de calculs usuels

KPN réseaux de processus de Kahn

SDF flots de données synchrones

DDF flots de données dynamiques (`switch` et `select`)

DE événements discrets

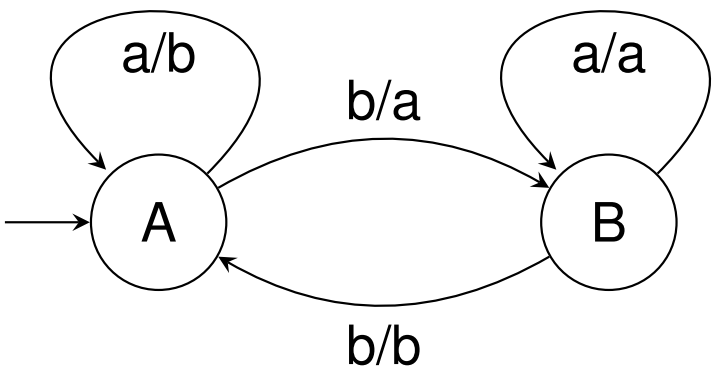
CT temps continu

Automates

Les bases

- modélisation explicite de l'état du système
- état initial
- transitions : événement (+ garde) / action
- états finals (ou acceptants)

Exemple



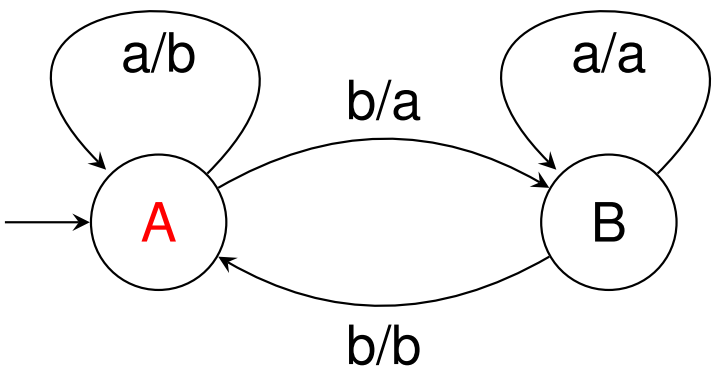
Modèle Ptolemy

Automates

Les bases

- modélisation explicite de l'état du système
- état initial
- transitions : événement (+ garde) / action
- états finals (ou acceptants)

Exemple



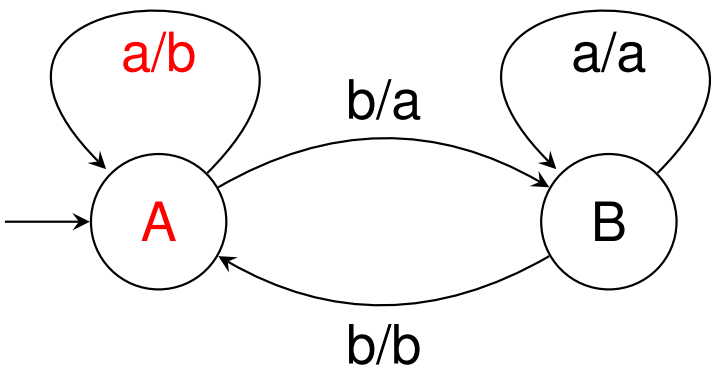
Modèle Ptolemy

Automates

Les bases

- modélisation explicite de l'état du système
- état initial
- transitions : événement (+ garde) / action
- états finals (ou acceptants)

Exemple



Modèle Ptolemy

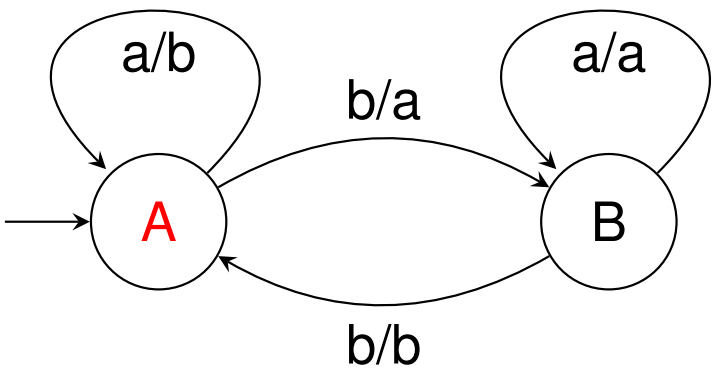
a

Automates

Les bases

- modélisation explicite de l'état du système
- état initial
- transitions : événement (+ garde) / action
- états finals (ou acceptants)

Exemple



Modèle Ptolemy

a

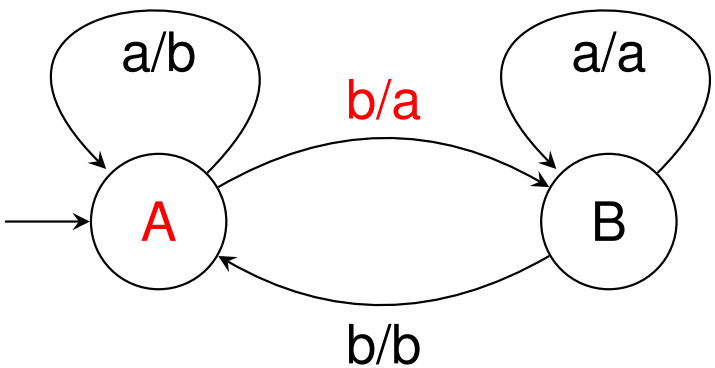
b

Automates

Les bases

- modélisation explicite de l'état du système
- état initial
- transitions : événement (+ garde) / action
- états finals (ou acceptants)

Exemple



Modèle Ptolemy

a b

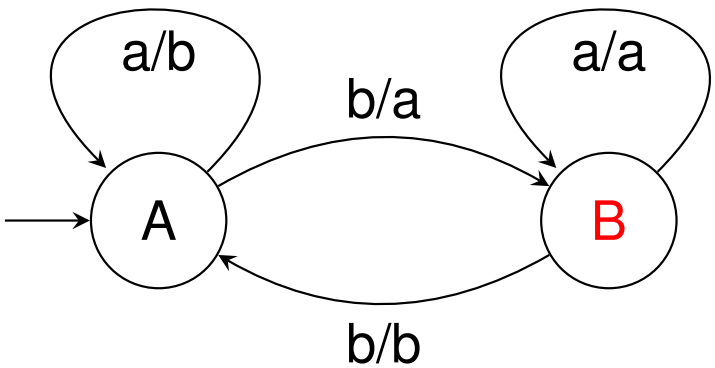
b

Automates

Les bases

- modélisation explicite de l'état du système
- état initial
- transitions : événement (+ garde) / action
- états finals (ou acceptants)

Exemple



Modèle Ptolemy

a b

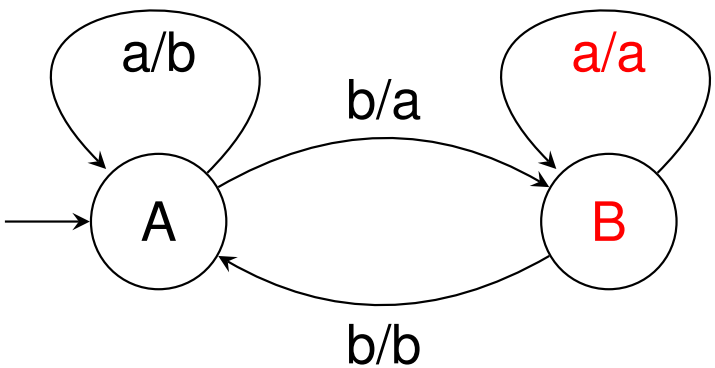
b a

Automates

Les bases

- modélisation explicite de l'état du système
- état initial
- transitions : événement (+ garde) / action
- états finals (ou acceptants)

Exemple



Modèle Ptolemy

a b a

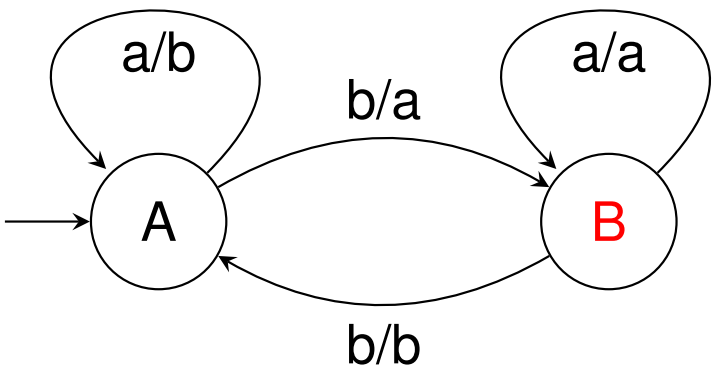
b a

Automates

Les bases

- modélisation explicite de l'état du système
- état initial
- transitions : événement (+ garde) / action
- états finals (ou acceptants)

Exemple



Modèle Ptolemy

a b a

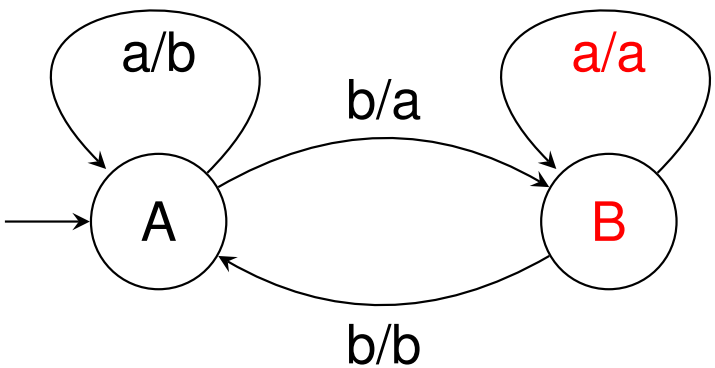
b a a

Automates

Les bases

- modélisation explicite de l'état du système
- état initial
- transitions : événement (+ garde) / action
- états finals (ou acceptants)

Exemple



Modèle Ptolemy

a b a a

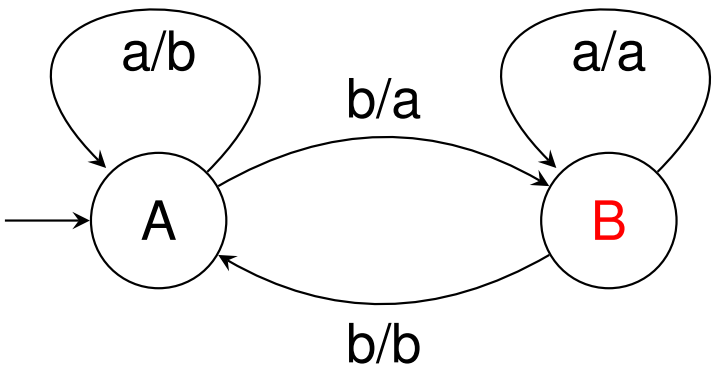
b a a

Automates

Les bases

- modélisation explicite de l'état du système
- état initial
- transitions : événement (+ garde) / action
- états finals (ou acceptants)

Exemple



Modèle Ptolemy

a b a a

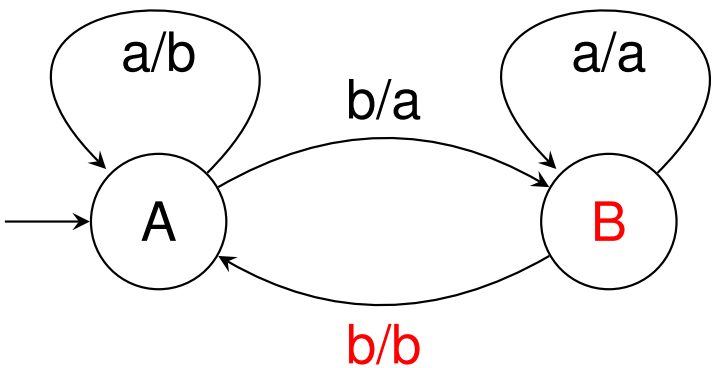
b a a a

Automates

Les bases

- modélisation explicite de l'état du système
- état initial
- transitions : événement (+ garde) / action
- états finals (ou acceptants)

Exemple



Modèle Ptolemy

a b a a b

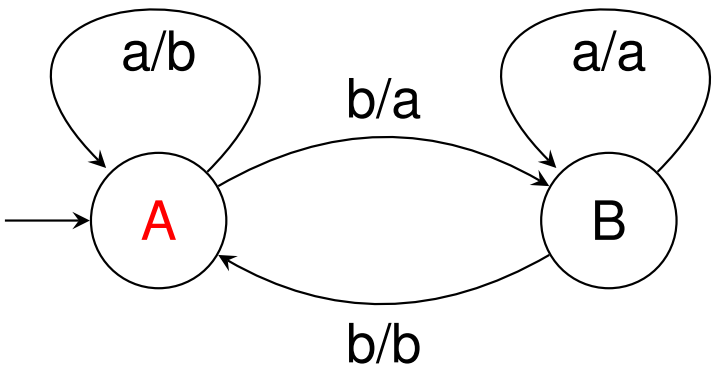
b a a a

Automates

Les bases

- modélisation explicite de l'état du système
- état initial
- transitions : événement (+ garde) / action
- états finals (ou acceptants)

Exemple



Modèle Ptolemy

a b a a b

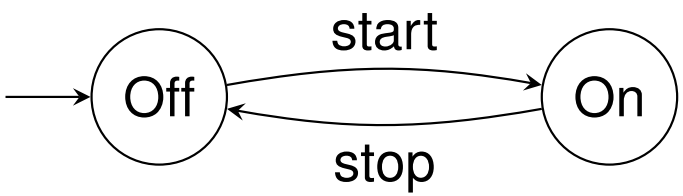
b a a a b

Automates hiérarchiques

Des états dans l'état...

- le comportement du système dans un état est décrit par un automate
- outil pour l'abstraction des comportements
- introduit quelques subtilités

Exemple

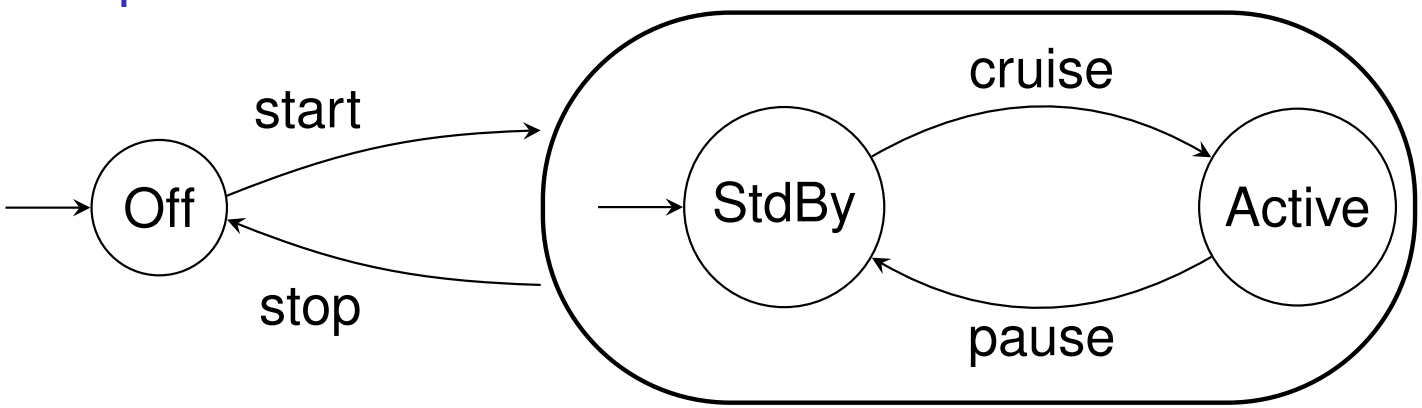


Automates hiérarchiques

Des états dans l'état...

- le comportement du système dans un état est décrit par un automate
- outil pour l'abstraction des comportements
- introduit quelques subtilités

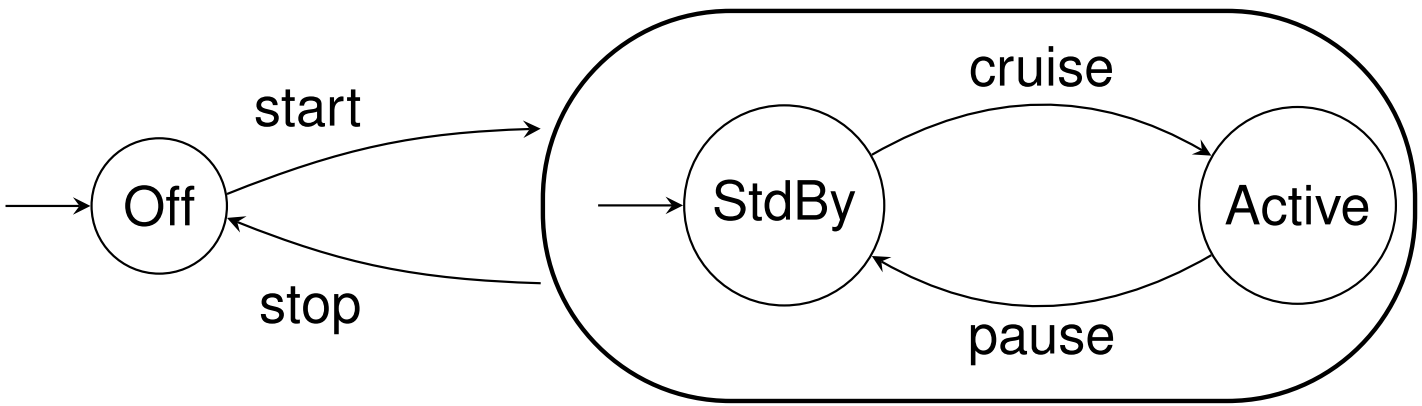
Exemple



Automates hiérarchiques

Subtilités

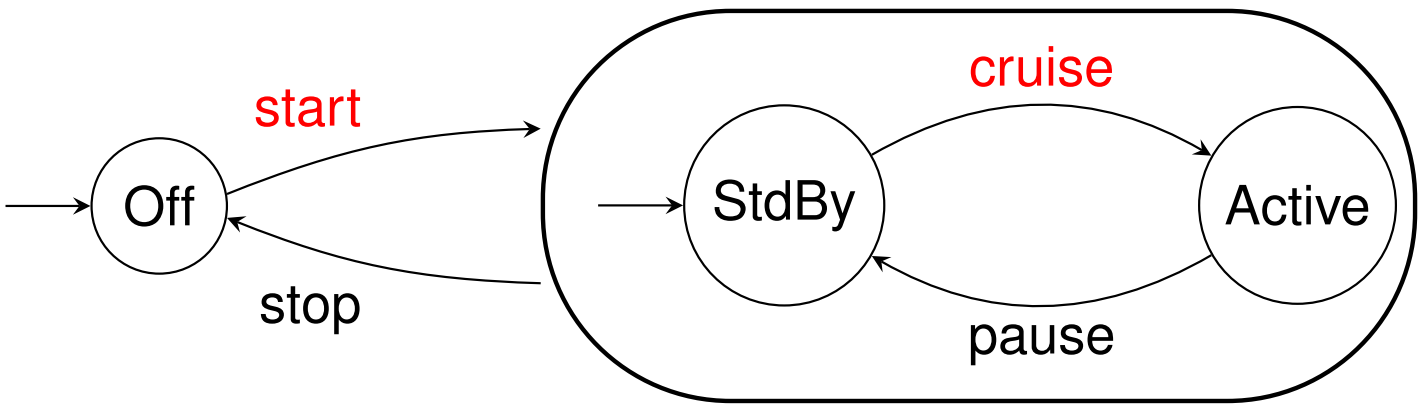
- Off : start et cruise en même temps ?
- On : pause et stop en même temps ?
- Off $\rightarrow$ On/StdBy $\rightarrow$ On/Active $\rightarrow$ Off $\rightarrow$ On/ ?



Automates hiérarchiques

Solutions

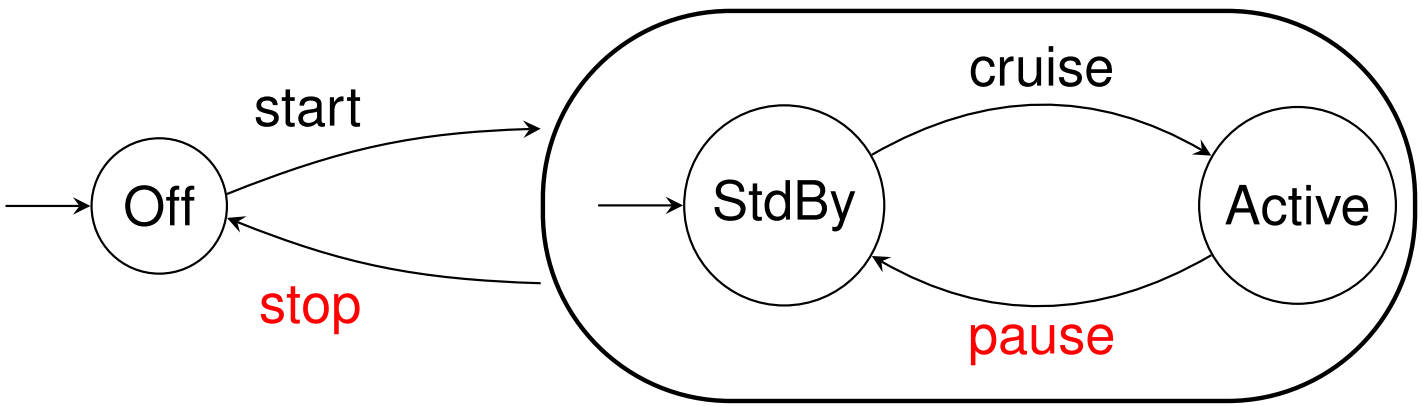
- prise en compte immediate ou non des événements



Automates hiérarchiques

Solutions

- prise en compte immediate ou non des événements
- préemption forte ou faible

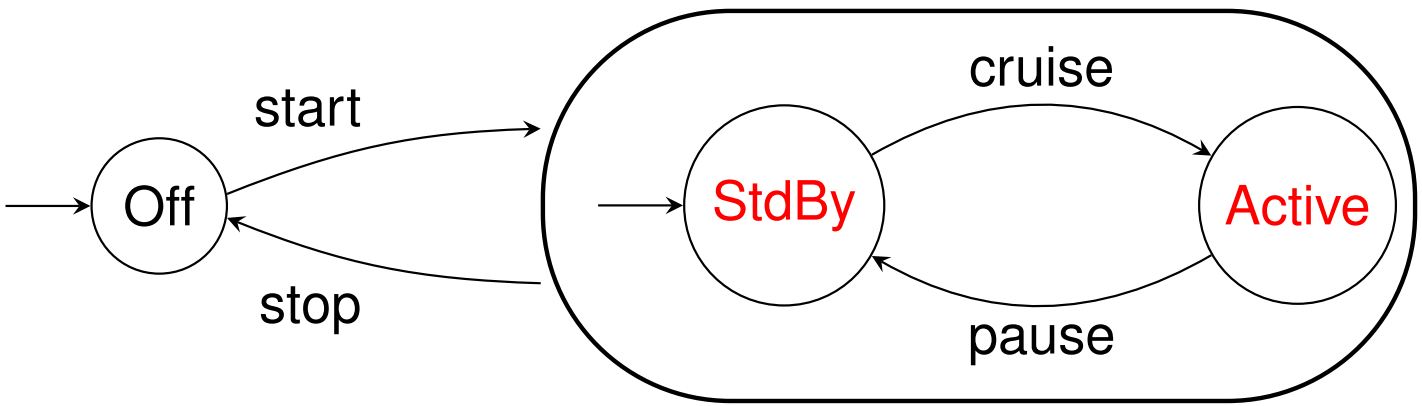


Modèle Ptolemy

Automates hiérarchiques

Solutions

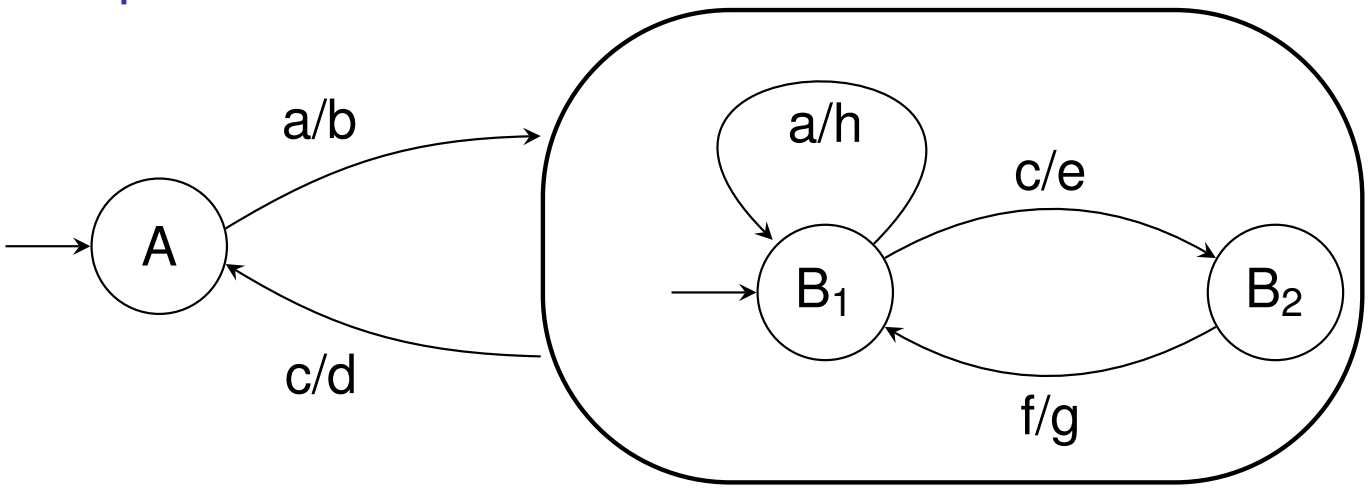
- prise en compte immediate ou non des événements
- préemption forte ou faible
- arrivée dans un état en mode initial ou historique



Modèle Ptolemy

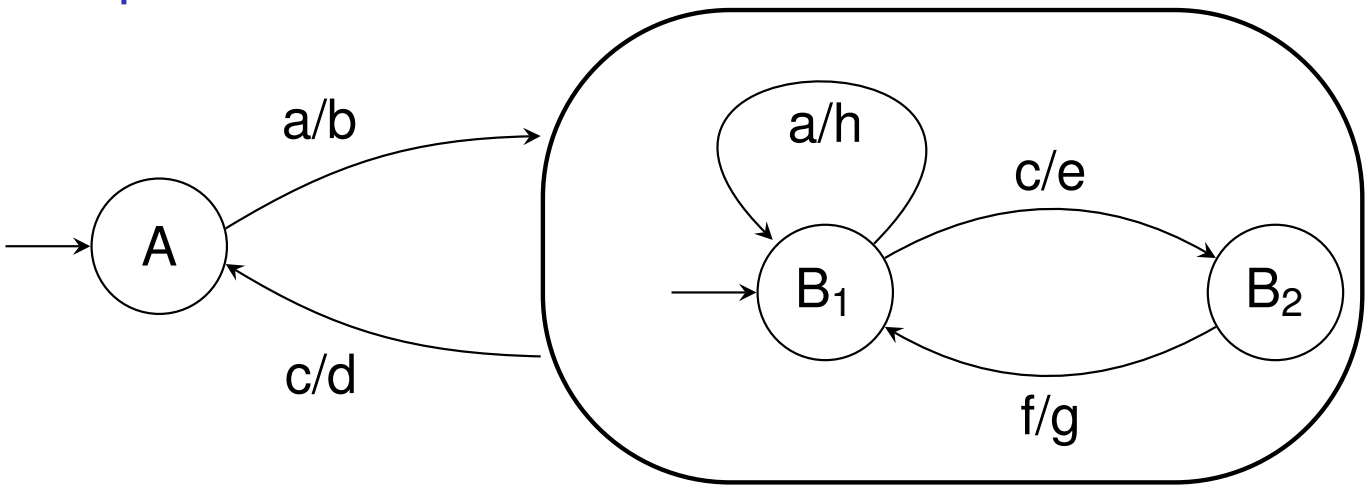
Automates hiérarchiques

Exemple



Automates hiérarchiques

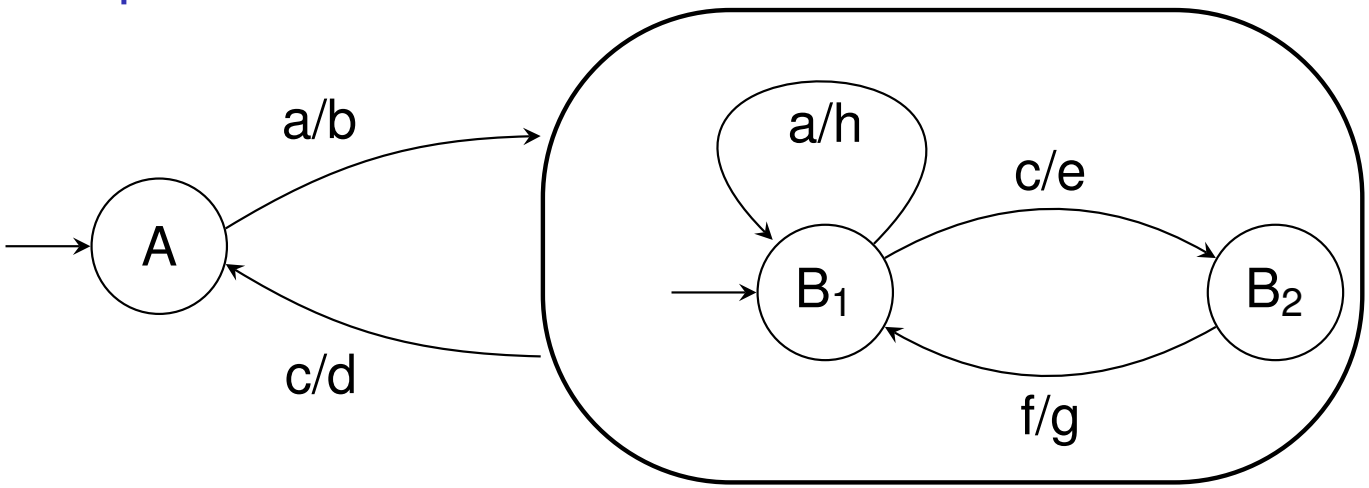
Exemple



entrée :	a	c	a	f	

Automates hiérarchiques

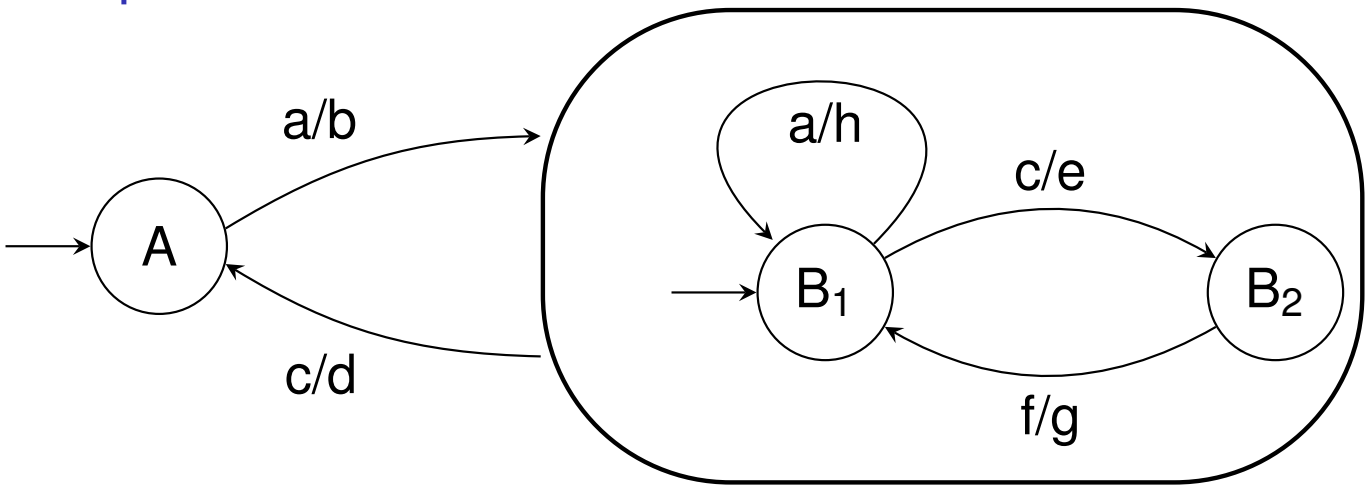
Exemple



entrée :	a	c	a	f	
	b	d	b	$\perp$	préemptif + reset

Automates hiérarchiques

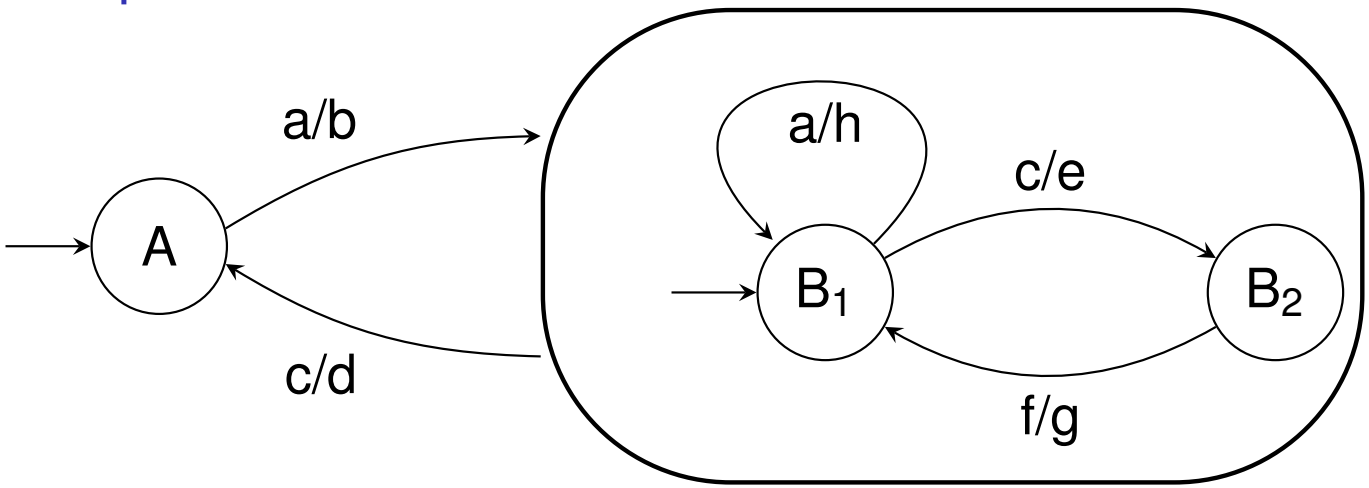
Exemple



entrée :	a	c	a	f	
	b	d	b	$\perp$	préemptif + reset
	b	d e	b	$\perp$	non préemptif + reset

Automates hiérarchiques

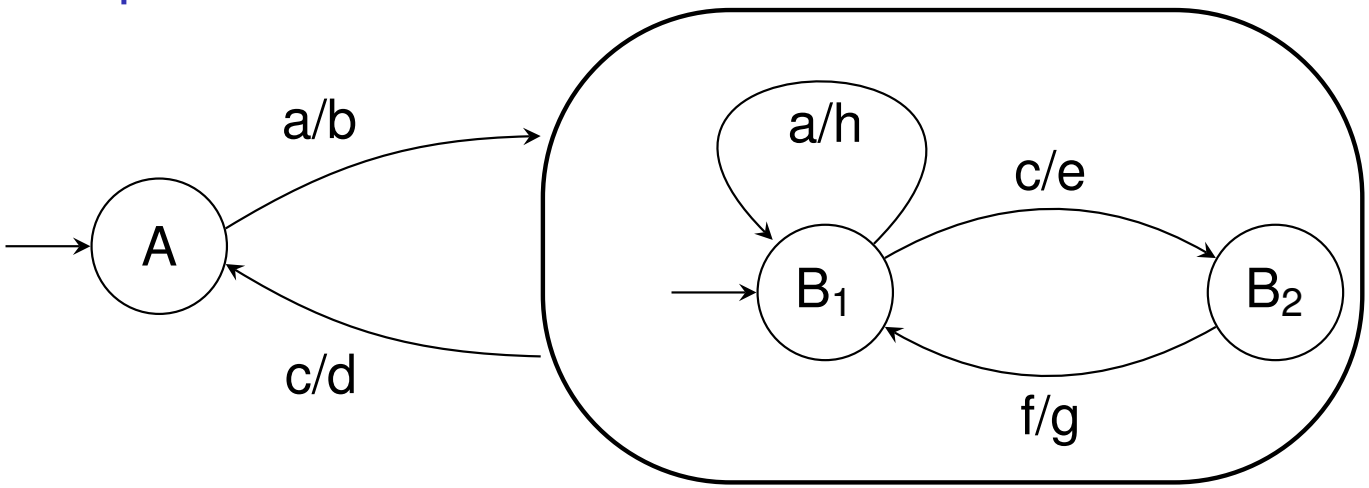
Exemple



entrée :	a	c	a	f	
	b	d	b	$\perp$	préemptif + reset
	b	d e	b	$\perp$	non préemptif + reset
	b	d e	b	g	non préemptif + historique

Automates hiérarchiques

Exemple



entrée :	a	c	a	f	
	b	d	b	$\perp$	préemptif + reset
	b	d e	b	$\perp$	non préemptif + reset
	b	d e	b	g	non préemptif + historique
	+h				dans le cas immédiat

Pourquoi le parallélisme

- le monde physique est intrinsèquement parallèle
  votre vie ne s'exécute pas en temps partagé sur un CPU universel
- c'est un outil de conception commode
  tâche de téléchargement + tâche d'affichage de sa progression

Quelques modèles

- threads et sémaphores
- algèbres de processus : CSP (Hoare), CCS et π-calcul (Milner)
- approche synchrone

Threads et sémaphores

Modèle

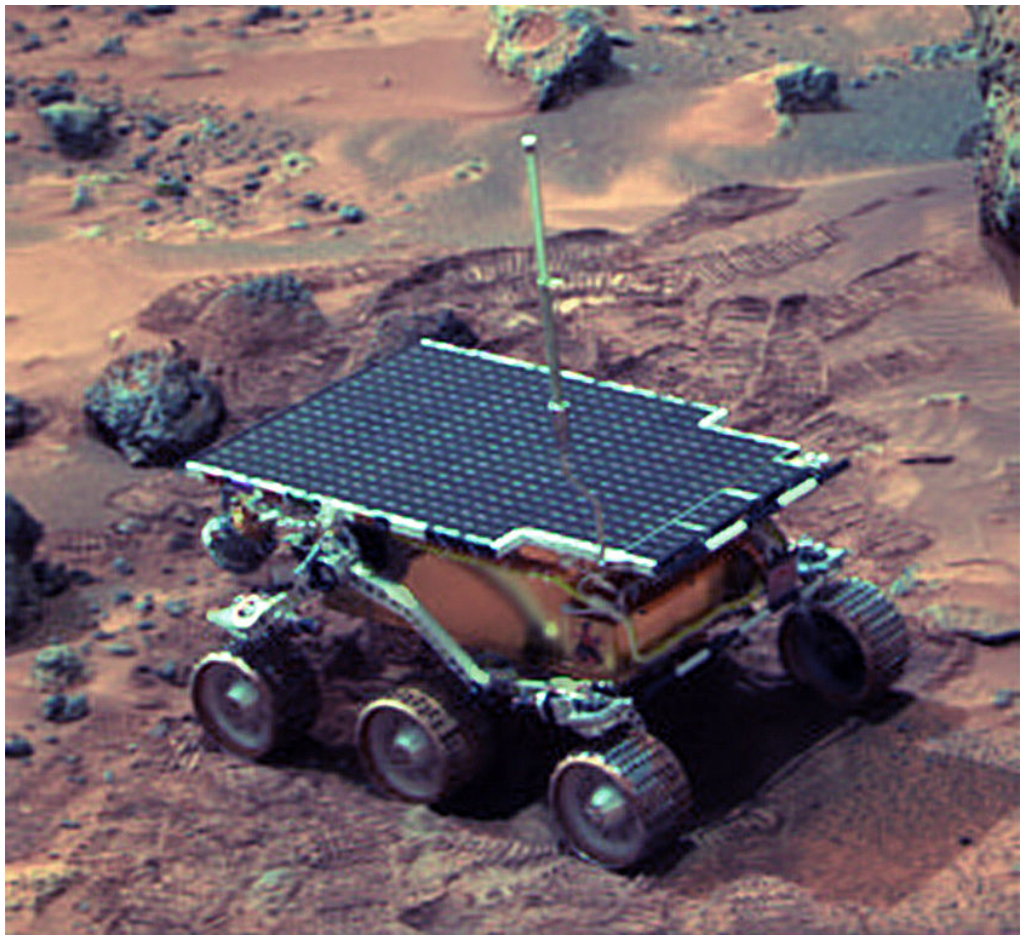
- Thread : flot séquentiel d'instructions
- Exécution asynchrone, priorités
- Synchronisation par sémaphores
  - acquire : acquisition de la ressource (bloquant)
  - release : production de la ressource

Problèmes

- Sémantique dépendante de l'ordonnancement
- Problèmes de synchronisation difficiles à analyser
- Pas d'algèbre : la composition de 2 threads n'est pas un thread

Exemple : Sojourner

Véhicule d'exploration (rover) de la mission Mars pathfinder (1997)



Exemple : Sojourner

Problème : le système redémarre régulièrement

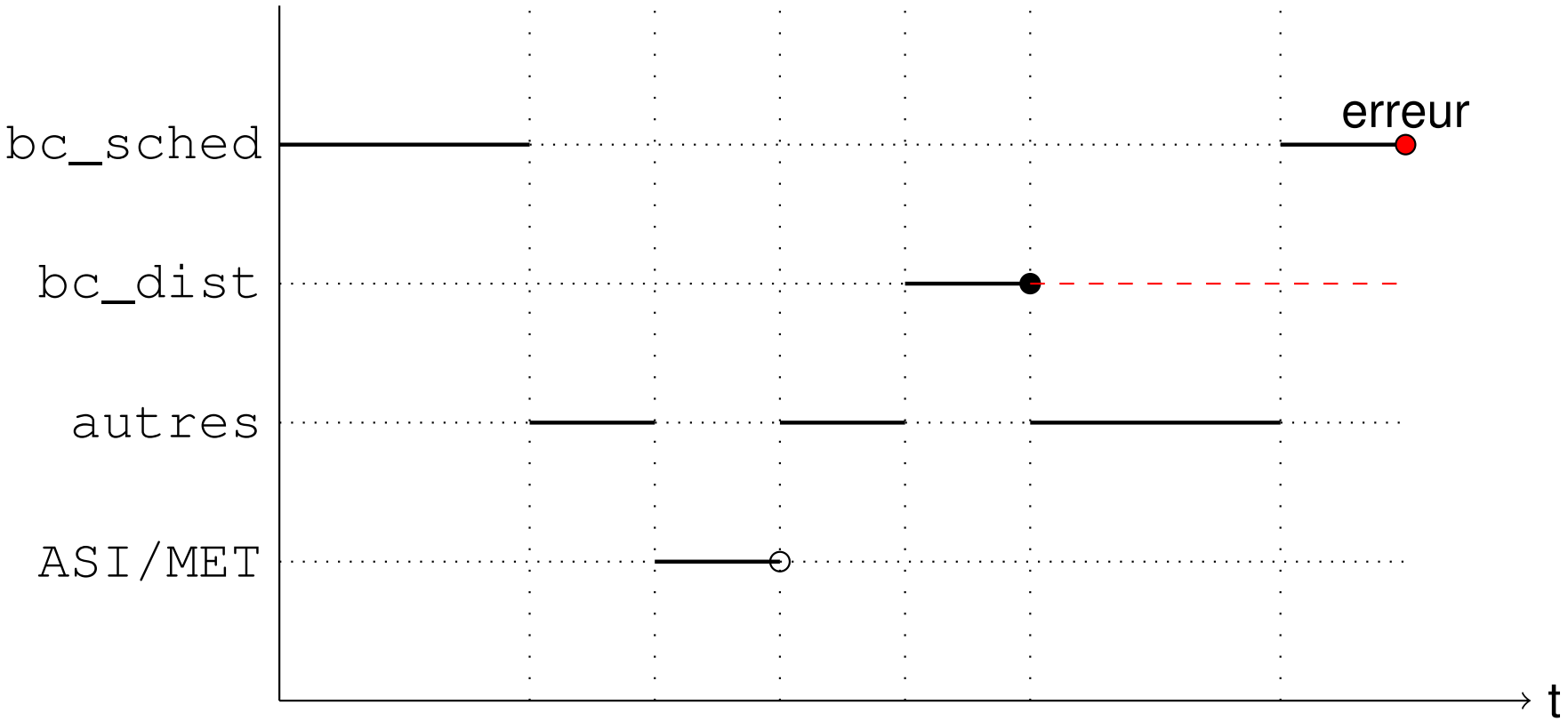
Éléments du problème

Programmation multi-threads sous VxWorks
Tâche `bc_sched`, priorité très élevée, ordonnance les transferts sur le bus
Tâche `bc_dist`, priorité élevée, distribue les données par `pipeWrite()`
Mesures météo ASI/MET, priorité faible, lit ses données par `select()`
`pipeWrite()` et `select()` sont synchronisées par un mutex.

Scénario

ASI/MET appelle `select()` qui acquiert le mutex, mais est préempté par des tâches de priorité moyenne avant de le rendre. Lorsque `bc_dist` appelle `pipeWrite()`, elle acquiert le mutex pour accéder à la liste des flux et est bloquée en attente. ASI/MET ne s'exécute pas car des tâches de priorité moyenne s'exécutent, et donc `bc_dist` reste bloquée. Lorsque c'est au tour de `bc_sched` de s'exécuter à nouveau, elle détecte que `bc_dist` n'est pas terminée, et déclare une erreur qui provoque le redémarrage du système.

Exemple : Sojourner



Inversion de priorité

Une tâche de priorité élevée (`bc_dist`) ne peut pas s'exécuter car elle est en attente d'une ressource (le mutex) qu'une tâche de priorité faible (`ASI/MET`) ne peut pas libérer car des tâches de priorité moyenne s'exécutent.

Exemple : Sojourner

Ce qui a sauvé la mission

- Les outils de debug ont été laissés dans la version opérationnelle (philosophie *Test what you fly and fly what you test*)
- Les ingénieurs du JPL ont pu récupérer les logs et trouver la source du problème
- La coopération avec Wind River leur a permis de trouver une solution (héritage de priorité) et d'évaluer son impact sur les performances.
- Ils ont pu tester la solution sur le prototype resté au sol
- Un mécanisme était prévu pour télécharger la nouvelle version sur le rover.

Exemple : Sojourner

Conclusion

- Même des ingénieurs ayant une grande expérience de la programmation multi-threads peuvent avoir du mal a maîtriser les interactions entre threads.
- Les threads et les sémaphores restent toutefois une des techniques les plus utilisées dans les systèmes embarqués temps-réel, ainsi que dans les systèmes d'information.

Algèbres de processus

Langage formel définissant des opérations de composition de processus.

π-calcul

- Création d'un canal de communication pour P : $(\nu c)P$
- Émission d'une donnée sur un canal : $c\langle x\rangle$ (ou $c!x$)
- Lecture d'une donnée depuis un canal : $c(x)$ (ou $c?x$)
- Exécution parallèle de 2 processus : $P|Q$
- Exécution en séquence de 2 processus : $P.Q$
- Exécution d'une infinité d'instances de P : $!P$

Exemple : gestionnaire d'impression

$$\begin{aligned}
\text{Spooler} &= (\nu spool)(spool\langle p_1\rangle | spool\langle p_2\rangle \\
&\quad | !spooler(job, cb).spool(p).(\nu w)(p\langle job, w\rangle | w().(cb\langle\rangle | spool\langle p\rangle))) \\
\text{Client} &= (\nu done)(spooler\langle doc, done\rangle | done()).Client \\
\text{Printer}_i &= p_i(job, ans).print_job.ans\langle\rangle.Printer_i
\end{aligned}$$

Évolution des processus

$$
\begin{aligned}
&\mathrm{Spooler} \mid \mathrm{Client} \mid \mathrm{Printer}_i = \\
&\quad (\nu\, spool)(\\
&\qquad spool\langle p_1 \rangle | spool\langle p_2 \rangle \\
&\qquad | !spooler(job, cb).spool(p).(\nu\, w)(p\langle job, w \rangle | w().(cb\langle\rangle | spool\langle p \rangle))) \\
&\quad | \ (\nu\, done)(spooler\langle doc, done \rangle | done()).Client \\
&\quad | \ p_i(job, ans).print\_job.ans\langle\rangle.Printer_i
\end{aligned}
$$

Évolution des processus

$$
\begin{aligned}
&\mathrm{Spooler} \mid \mathrm{Client} \mid \mathrm{Printer}_i = \\
&\quad (\nu\, spool)(\\
&\qquad spool\langle p_1 \rangle | spool\langle p_2 \rangle \\
&\qquad | !spooler(job, cb).spool(p).(\nu\, w)(p\langle job, w \rangle | w().(cb\langle\rangle | spool\langle p \rangle))) \\
&\quad | \ (\nu\, done)(spooler\langle doc, done \rangle | done()).Client \\
&\quad | \ p_i(job, ans).print\_job.ans\langle\rangle.Printer_i
\end{aligned}
$$

Évolution des processus

$$
\begin{aligned}
&\mathrm{Spooler} \mid \mathrm{Client} \mid \mathrm{Printer}_i = \\
&\quad (\nu\, spool)(\\
&\qquad spool\langle p_1 \rangle | spool\langle p_2 \rangle \\
&\qquad | !spooler(job, cb).spool(p).(\nu\, w)(p\langle job, w \rangle | w().(cb\langle\rangle | spool\langle p \rangle))) \\
&\quad | \ (\nu\, done)(spooler\langle doc, done \rangle | done()).Client \\
&\quad | \ p_i(job, ans).print\_job.ans\langle\rangle.Printer_i
\end{aligned}
$$

$$
\begin{aligned}
&\mathrm{Spooler} \mid \mathrm{Client} \mid \mathrm{Printer}_i = \\
&\quad (\nu\, spool)(\\
&\qquad spool\langle p_1 \rangle | spool\langle p_2 \rangle \\
&\qquad | spool(p).(\nu\, w)(p\langle doc, w \rangle | w().(done\langle\rangle | spool\langle p \rangle))) \\
&\quad | \ (\nu\, done)(done()).Client \\
&\quad | \ p_i(job, ans).print\_job.ans\langle\rangle.Printer_i
\end{aligned}
$$

Évolution des processus

$$
\begin{aligned}
&\text{Spooler} \mid \text{Client} \mid \text{Printer}_i = \\
&\quad (\nu\, spool)(\\
&\qquad spool\langle p_1 \rangle \mid spool\langle p_2 \rangle \\
&\qquad \mid !spooler(job, cb).spool(p).(\nu\, w)(p\langle job, w \rangle \mid w().(cb\langle\rangle \mid spool\langle p \rangle))) \\
&\quad \mid (\nu\, done)(spooler\langle doc, done \rangle \mid done()).Client \\
&\quad \mid p_i(job, ans).print\_job.ans\langle\rangle.Printer_i
\end{aligned}
$$

$$
\begin{aligned}
&\text{Spooler} \mid \text{Client} \mid \text{Printer}_i = \\
&\quad (\nu\, spool)(\\
&\qquad spool\langle p_1 \rangle \mid spool\langle p_2 \rangle \\
&\qquad \mid spool(p).(\nu\, w)(p\langle doc, w \rangle \mid w().(done\langle\rangle \mid spool\langle p \rangle))) \\
&\quad \mid (\nu\, done)(done()).Client \\
&\quad \mid p_i(job, ans).print\_job.ans\langle\rangle.Printer_i
\end{aligned}
$$

Évolution des processus

$$
\begin{aligned}
&\mathrm{Spooler} \mid \mathrm{Client} \mid \mathrm{Printer}_i = \\
&\quad (\nu\, spool)(\\
&\qquad spool\langle p_1 \rangle | spool\langle p_2 \rangle \\
&\qquad | !spooler(job, cb).spool(p).(\nu\, w)(p\langle job, w \rangle | w().(cb\langle\rangle | spool\langle p \rangle))) \\
&\quad | \ (\nu\, done)(spooler\langle doc, done \rangle | done()).Client \\
&\quad | \ p_i(job, ans).print\_job.ans\langle\rangle.\mathrm{Printer}_i
\end{aligned}
$$

$$
\begin{aligned}
&\mathrm{Spooler} \mid \mathrm{Client} \mid \mathrm{Printer}_i = \\
&\quad (\nu\, spool)(\\
&\qquad spool\langle p_2 \rangle \\
&\qquad | (\nu\, w)(p_1\langle doc, w \rangle | w().(done\langle\rangle | spool\langle p_1 \rangle))) \\
&\quad | \ (\nu\, done)(done()).Client \\
&\quad | \ p_i(job, ans).print\_job.ans\langle\rangle.\mathrm{Printer}_i
\end{aligned}
$$

Évolution des processus

$$
\begin{aligned}
&\mathrm{Spooler} \mid \mathrm{Client} \mid \mathrm{Printer}_i = \\
&\quad (\nu\, spool)(\\
&\qquad spool\langle p_1 \rangle | spool\langle p_2 \rangle \\
&\qquad | !spooler(job, cb).spool(p).(\nu\, w)(p\langle job, w \rangle | w().(cb\langle\rangle | spool\langle p \rangle))) \\
&\quad | \ (\nu\, done)(spooler\langle doc, done \rangle | done()).Client \\
&\quad | \ p_i(job, ans).print\_job.ans\langle\rangle.Printer_i
\end{aligned}
$$

$$
\begin{aligned}
&\mathrm{Spooler} \mid \mathrm{Client} \mid \mathrm{Printer}_i = \\
&\quad (\nu\, spool)(\\
&\qquad spool\langle p_2 \rangle \\
&\qquad | (\nu\, w)(p_1\langle doc, w \rangle | w().(done\langle\rangle | spool\langle p_1 \rangle))) \\
&\quad | \ (\nu\, done)(done()).Client \\
&\quad | \ p_i(job, ans).print\_job.ans\langle\rangle.Printer_i
\end{aligned}
$$

Évolution des processus

$$
\begin{aligned}
&\mathrm{Spooler} \mid \mathrm{Client} \mid \mathrm{Printer}_i = \\
&\quad (\nu\, spool)(\\
&\qquad spool\langle p_1 \rangle | spool\langle p_2 \rangle \\
&\qquad | !spooler(job, cb).spool(p).(\nu\, w)(p\langle job, w \rangle | w().(cb\langle\rangle | spool\langle p \rangle))) \\
&\quad | \ (\nu\, done)(spooler\langle doc, done \rangle | done()).Client \\
&\quad | \ p_i(job, ans).print\_job.ans\langle\rangle.Printer_i
\end{aligned}
$$

$$
\begin{aligned}
&\mathrm{Spooler} \mid \mathrm{Client} \mid \mathrm{Printer}_i = \\
&\quad (\nu\, spool)(\\
&\qquad spool\langle p_2 \rangle \\
&\qquad | (\nu\, w)(w().(done\langle\rangle | spool\langle p_1 \rangle))) \\
&\quad | \ (\nu\, done)(done()).Client \\
&\quad | \ print\_job.w\langle\rangle.Printer_i
\end{aligned}
$$

Évolution des processus

$$
\begin{aligned}
&\text{Spooler} \mid \text{Client} \mid \text{Printer}_i = \\
&\quad (\nu\, spool)(\\
&\qquad spool\langle p_1 \rangle | spool\langle p_2 \rangle \\
&\qquad | !spooler(job, cb).spool(p).(\nu\, w)(p\langle job, w \rangle | w().(cb\langle\rangle | spool\langle p \rangle))) \\
&\quad | \ (\nu\, done)(spooler\langle doc, done \rangle | done()).Client \\
&\quad | \ p_i(job, ans).print\_job.ans\langle\rangle.Printer_i
\end{aligned}
$$

$$
\begin{aligned}
&\text{Spooler} \mid \text{Client} \mid \text{Printer}_i = \\
&\quad (\nu\, spool)(\\
&\qquad spool\langle p_2 \rangle \\
&\qquad | (\nu\, w)(w().(done\langle\rangle | spool\langle p_1 \rangle))) \\
&\quad | \ (\nu\, done)(done()).Client \\
&\quad | \ print\_job.w\langle\rangle.Printer_i
\end{aligned}
$$

Évolution des processus

$$
\begin{aligned}
&\mathrm{Spooler} \mid \mathrm{Client} \mid \mathrm{Printer}_i = \\
&\quad (\nu\, spool)(\\
&\qquad spool\langle p_1 \rangle | spool\langle p_2 \rangle \\
&\qquad | !spooler(job, cb).spool(p).(\nu\, w)(p\langle job, w \rangle | w().(cb\langle\rangle | spool\langle p \rangle))) \\
&\quad | \ (\nu\, done)(spooler\langle doc, done \rangle | done()).Client \\
&\quad | \ p_i(job, ans).print\_job.ans\langle\rangle.Printer_i
\end{aligned}
$$

$$
\begin{aligned}
&\mathrm{Spooler} \mid \mathrm{Client} \mid \mathrm{Printer}_i = \\
&\quad (\nu\, spool)(\\
&\qquad spool\langle p_2 \rangle \\
&\qquad | (\nu\, w)(w().(done\langle\rangle | spool\langle p_1 \rangle))) \\
&\quad | \ (\nu\, done)(done()).Client \\
&\quad | \ w\langle\rangle.Printer_i
\end{aligned}
$$

Évolution des processus

$$
\begin{aligned}
&\mathrm{Spooler} \mid \mathrm{Client} \mid \mathrm{Printer}_i = \\
&\quad (\nu\, spool)(\\
&\qquad spool\langle p_1 \rangle | spool\langle p_2 \rangle \\
&\qquad | !spooler(job, cb).spool(p).(\nu\, w)(p\langle job, w \rangle | w().(cb\langle\rangle | spool\langle p \rangle))) \\
&\quad | \ (\nu\, done)(spooler\langle doc, done \rangle | done()).Client \\
&\quad | \ p_i(job, ans).print\_job.ans\langle\rangle.Printer_i
\end{aligned}
$$

$$
\begin{aligned}
&\mathrm{Spooler} \mid \mathrm{Client} \mid \mathrm{Printer}_i = \\
&\quad (\nu\, spool)(\\
&\qquad spool\langle p_2 \rangle \\
&\qquad | (\nu\, w)((done\langle\rangle | spool\langle p_1 \rangle))) \\
&\quad | \ (\nu\, done)(done()).Client \\
&\quad | \ Printer_i
\end{aligned}
$$

Évolution des processus

$$
\begin{aligned}
&\mathrm{Spooler} \mid \mathrm{Client} \mid \mathrm{Printer}_i = \\
&\quad (\nu\, spool)(\\
&\qquad spool\langle p_1 \rangle | spool\langle p_2 \rangle \\
&\qquad | !spooler(job, cb).spool(p).(\nu\, w)(p\langle job, w \rangle | w().(cb\langle\rangle | spool\langle p \rangle))) \\
&\quad | \ (\nu\, done)(spooler\langle doc, done \rangle | done()).Client \\
&\quad | \ p_i(job, ans).print\_job.ans\langle\rangle.Printer_i
\end{aligned}
$$

$$
\begin{aligned}
&\mathrm{Spooler} \mid \mathrm{Client} \mid \mathrm{Printer}_i = \\
&\quad (\nu\, spool)(\\
&\qquad spool\langle p_2 \rangle \\
&\qquad | (\nu\, w)((done\langle\rangle | spool\langle p_1 \rangle))) \\
&\quad | \ (\nu\, done)(done()).Client \\
&\quad | \ Printer_i
\end{aligned}
$$

Évolution des processus

$$
\begin{aligned}
&\text{Spooler} \mid \text{Client} \mid \text{Printer}_i = \\
&\quad (\nu\, spool)(\\
&\qquad spool\langle p_1 \rangle \mid spool\langle p_2 \rangle \\
&\qquad \mid !spooler(job, cb).spool(p).(\nu\, w)(p\langle job, w \rangle \mid w().(cb\langle\rangle \mid spool\langle p \rangle))) \\
&\quad \mid (\nu\, done)(spooler\langle doc, done \rangle \mid done()).Client \\
&\quad \mid p_i(job, ans).print\_job.ans\langle\rangle.Printer_i
\end{aligned}
$$

$$
\begin{aligned}
&\text{Spooler} \mid \text{Client} \mid \text{Printer}_i = \\
&\quad (\nu\, spool)(\\
&\qquad spool\langle p_2 \rangle \\
&\qquad \mid spool\langle p_1 \rangle \\
&\quad \mid Client \\
&\quad \mid Printer_i
\end{aligned}
$$

Évolution des processus

$$
\begin{aligned}
&\mathrm{Spooler} \mid \mathrm{Client} \mid \mathrm{Printer}_i = \\
&\quad (\nu\, spool)(\\
&\qquad spool\langle p_1 \rangle | spool\langle p_2 \rangle \\
&\qquad | !spooler(job, cb).spool(p).(\nu\, w)(p\langle job, w \rangle | w().(cb\langle\rangle | spool\langle p \rangle))) \\
&\quad | \ (\nu\, done)(spooler\langle doc, done \rangle | done()).Client \\
&\quad | \ p_i(job, ans).print\_job.ans\langle\rangle.Printer_i
\end{aligned}
$$

$$
\begin{aligned}
&\mathrm{Spooler} \mid \mathrm{Client} \mid \mathrm{Printer}_i = \\
&\quad (\nu\, spool)(\\
&\qquad spool\langle p_2 \rangle \\
&\qquad | spool\langle p_1 \rangle \\
&\qquad | !spooler(job, cb).spool(p).(\nu\, w)(p\langle job, w \rangle | w().(cb\langle\rangle | spool\langle p \rangle))) \\
&\quad | \ Client \\
&\quad | \ p_i(job, ans).print\_job.ans\langle\rangle.Printer_i
\end{aligned}
$$

L'approche réactive synchrone

France, années 1980

- Lustre (Paul Caspi, Nicolas Halbwachs)
- Esterel (Gérard Berry)
- Signal (Albert Benveniste, Paul Le Guernic)

Réactif synchrone

- À la frontière entre l'automatique et l'informatique
- Interaction permanente avec l'environnement
- Nécessité de modéliser explicitement le temps
- Comportement = suite de réactions instantanées

L'hypothèse synchrone

La réaction du système à son environnement est instantanée

En théorie

- Donne un sens précis à la composition parallèle de processus
- Permet d'allier parallélisme et déterminisme
- Compilation en circuits logiques, vérification formelle

En pratique

- Il suffit que la réaction du système soit suffisamment rapide
- Compilation du parallélisme en code séquentiel
- Vérification par model-checking et interprétation abstraite
- SCADE, Esterel Studio, Sigali

Exemple : Esterel

```
module SpeedCounter:
  input m, s;
  output speed : integer;
  loop
    var dist := 0 : integer in
      abort
        every m do
          dist := dist + 1
        end every
      when s;
      emit speed(dist)
    end var
  end loop
end module
```

Exemple : Esterel

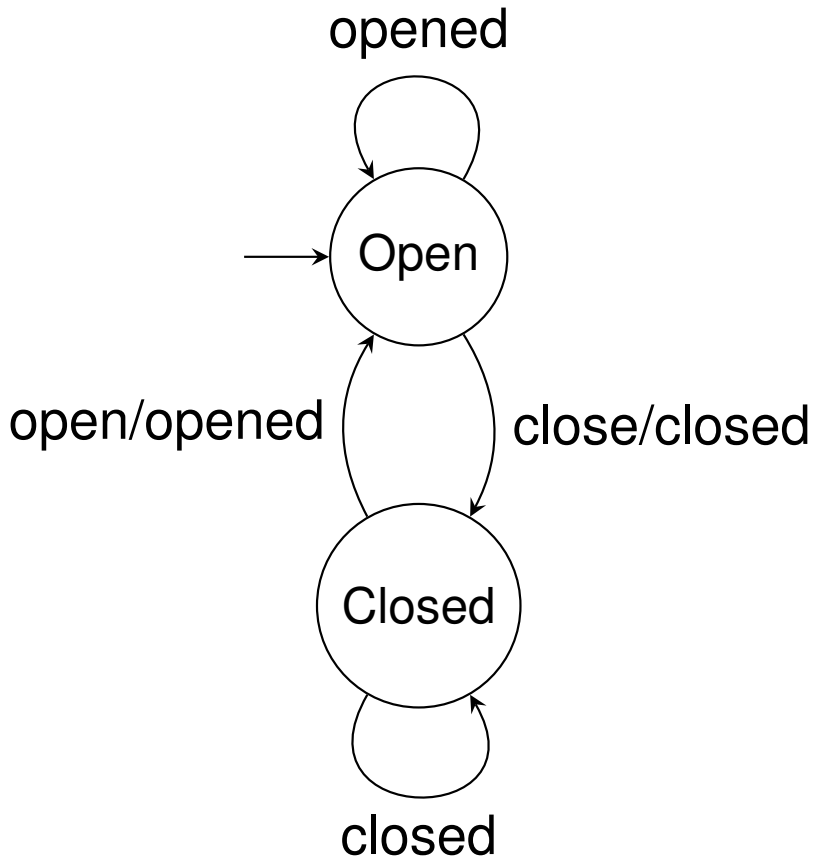
```
module SpeedCounter:
  input m, s;
  output speed : integer;
  loop
    var dist := 0 : integer in
      weak abort
        every m do
          dist := dist + 1
        end every
      when s;
      emit speed(dist)
    end var
  end loop
end module
```

Exemple : Esterel

```
module SpeedCounter:
  input m, s;
  output speed : integer;
  loop
    var dist := 0 : integer in
      weak abort
        every immediate m do
          dist := dist + 1
        end every
      when s;
      emit speed(dist)
    end var
  end loop
end module
```

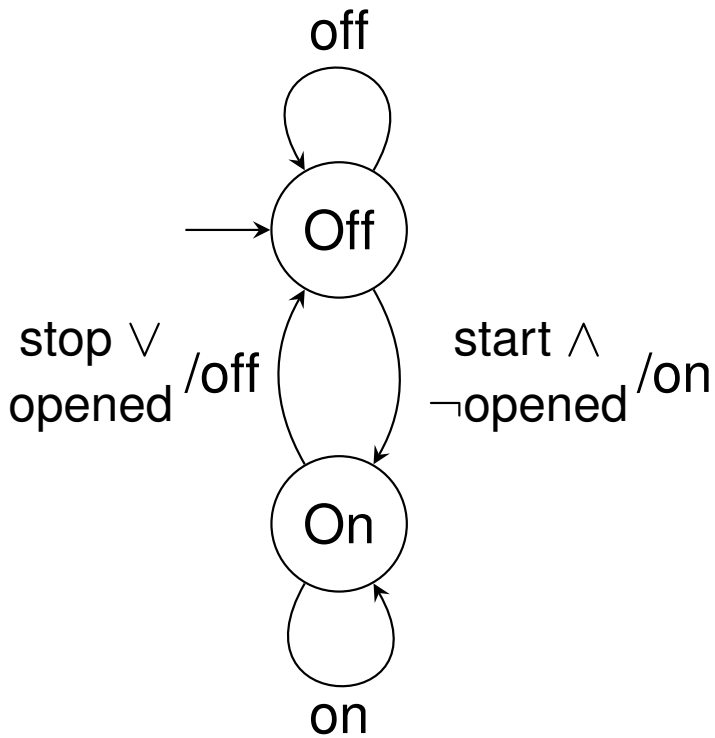
Exemple : four micro-ondes

```
module Doors:
  input open, close;
  output opened, closed;

  loop
    abort
      sustain opened
    when close;
    abort
      sustain closed
    when open
  end loop
end module
```



Exemple : four micro-ondes

```
module Power:
  input start, stop, opened;
  output on, off;

  loop
    abort
      sustain off
    when [start and not opened];
    abort
      sustain on
    when [stop or opened]
  end loop
end module
```



Exemple : four micro-ondes

```
module MicroWave:
  input start, stop, open, close;
  output on, off, opened, closed;

  run Doors
||
  run Power
end module
```

Exemple : vérification

```
module CheckOven:
  input start, stop, open, close;
  output failure;
  signal on, off, opened, closed in
    run MicroWave
  ||
  [
    await immediate [on and opened];
    emit failure
  ]
  end signal
end module
```

Sémantique constructive du synchrone

Principes

- Pas d'hypothèse sur l'état des signaux
- Propagation des informations connues
- Acteurs non-stricts (réactions partielles)
- Méthode non exhaustive, mais en temps polynomial

Exemple : Arbitre de bus

Ce qu'il faut retenir

- langage généraliste $\neq$ meilleur outil pour traiter un problème
- choix du meilleur paradigme au bon niveau d'abstraction

Les avantages d'un domaine sémantique limité

- moins de risque de faire des erreurs
- concepts du langage = concepts métier
- possibilité de vérification, de génération automatique
- outillage *grammarware* pour les DSLs

Mais...

- domaine sémantique restreint $\Rightarrow$ ne convient pas au système complet
- modèle global du système = somme de modèles hétérogènes
- comment faire ? ☞ suite du cours par Lina Ye

Modélisation multi-paradigme