Systèmes réactifs logiciels
le design du langage synchrone Esterel v5

Gérard Berry

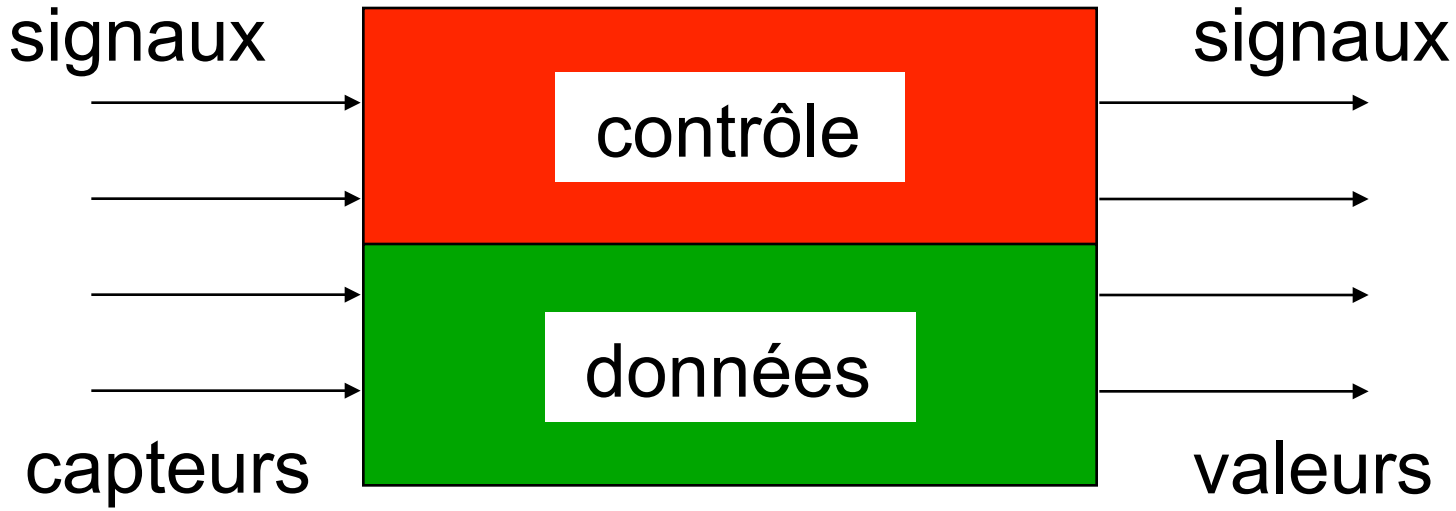
Chaire Algorithmes, machines et langages

Collège de France

Cours 3, 16 avril 2013

COLLÈGE
DE FRANCE
1530

Systèmes réactifs : contrôle vs. données (1982-2000)

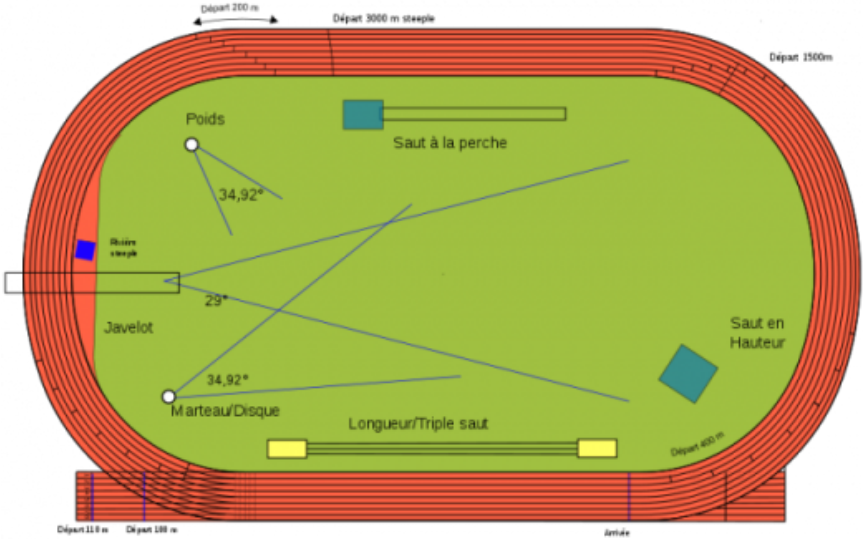


En liaison avec l'automatique

Esterel
*Statecharts
Argos
SyncCharts

Lustre
SCADE
Signal

Temps multiforme hiérarchique




```
module Training :
   input Second, Hour, Morning;
   relation Hour => Second, Morning => Second;
   input Meter, Lap, Step, HeartBeat;
   relation Lap => Meter;
   ... // behavioral code
end module
```

```
trap HeartAttack in
   every Morning do
      abort
         loop
            abort  run Slowly  when 100 Meter ;
            abort
               every Step do
                  run Jump || run Breathe || <CheckHeart>
               end every
            when 15 Second ;
            run FullSpeed
         each  Lap
      when  4 Lap
   end every
handle HeartAttack fo
   run RushToHospital
end trap
```

exécution d'un sous-module

préemption

parallélisme

séquence

exception

exit HeartAttack

Code de <CheckHeart>

```
loop
    await 3 Second ;
    exit HeartAttack
each HeartBeat
```

Chaque HeartBeat annule une attente de 3 Second

Idées fondatrices

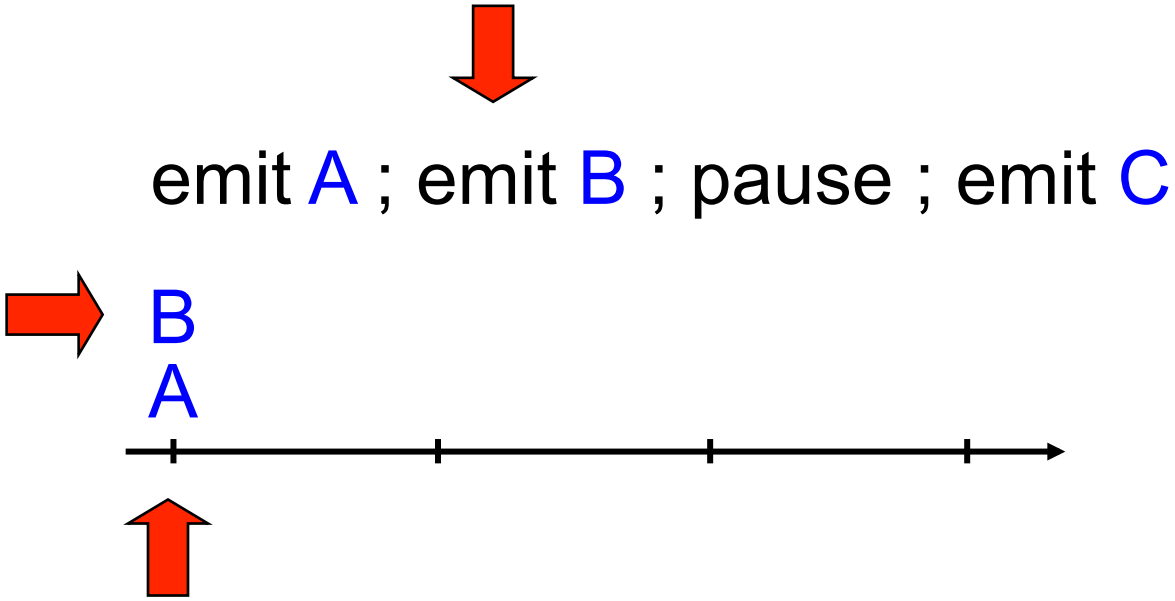
- Synchronisme parfait
  - administration et communication en temps 0
  - parallélisme déterministe et compréhensible
  - partage de comportement (WTO = Write Things Once)
- Sémantique mathématique
  - vraie définition du langage
  - compréhension fine des problèmes de causalité
  - sans restrictions artificielles « pour simplifier »
- Implémentation et utilisation professionnelle
  - compilation efficace et vérification formelle
  - WYPIWYE : What you prove is what you execute
- Utilisation professionnelle pour des logiciels critiques
  - montez-vous dans l'avion que vous avez programmé?

La séquence

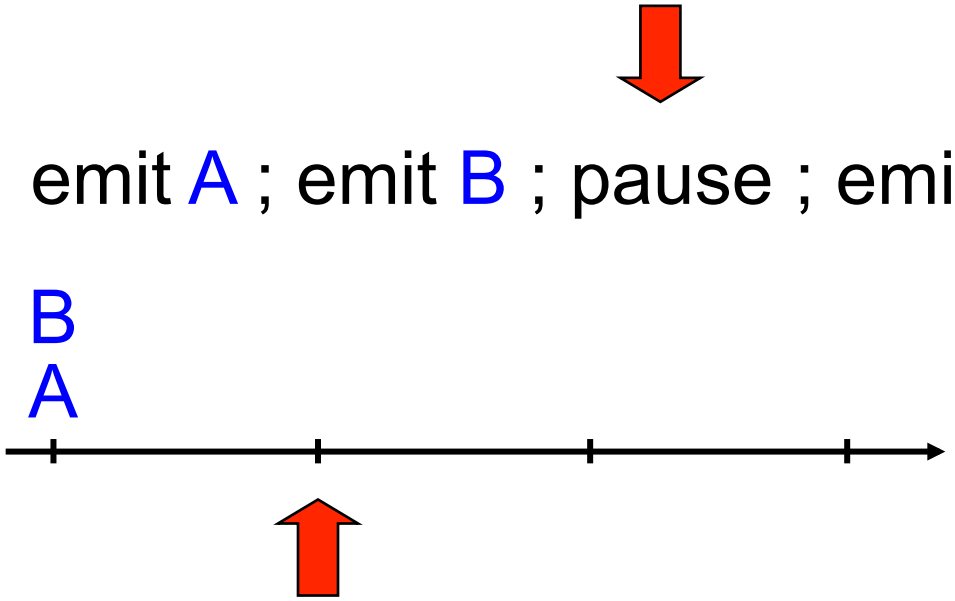
emit A ; emit B ; pause ; emit C

A

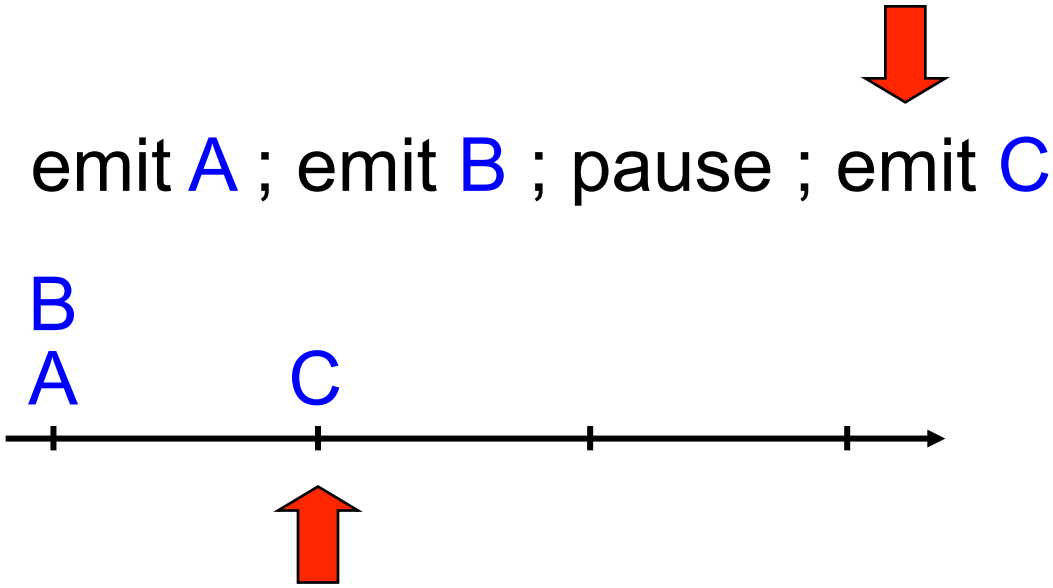
La séquence



La séquence

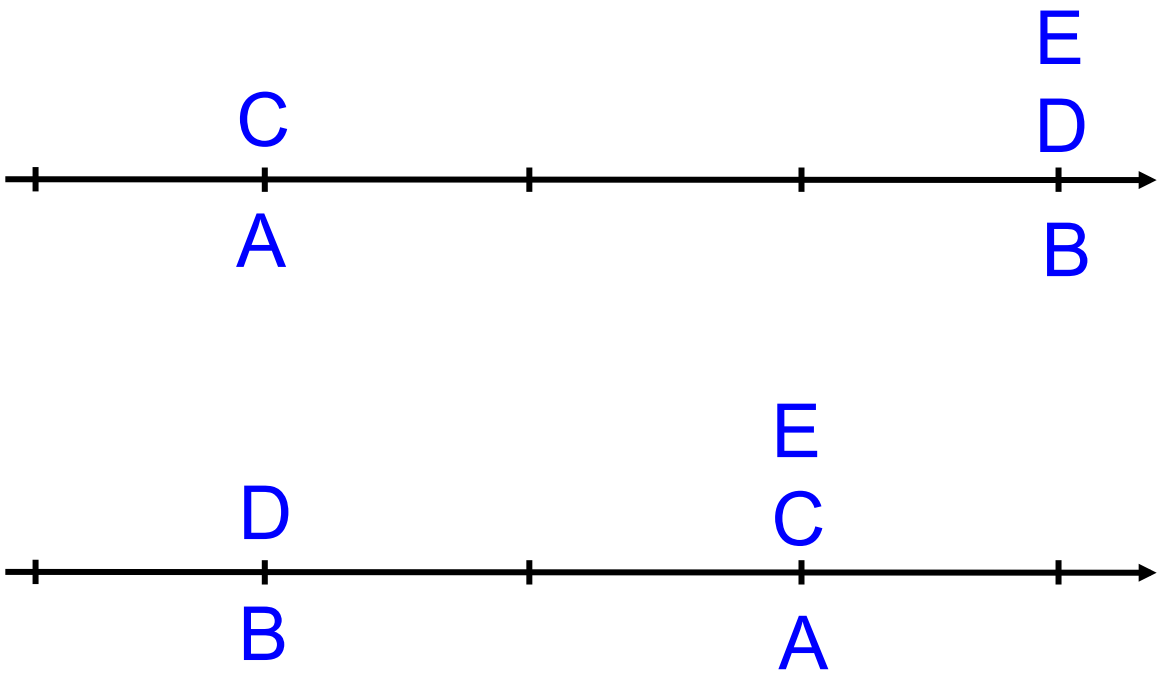


La séquence



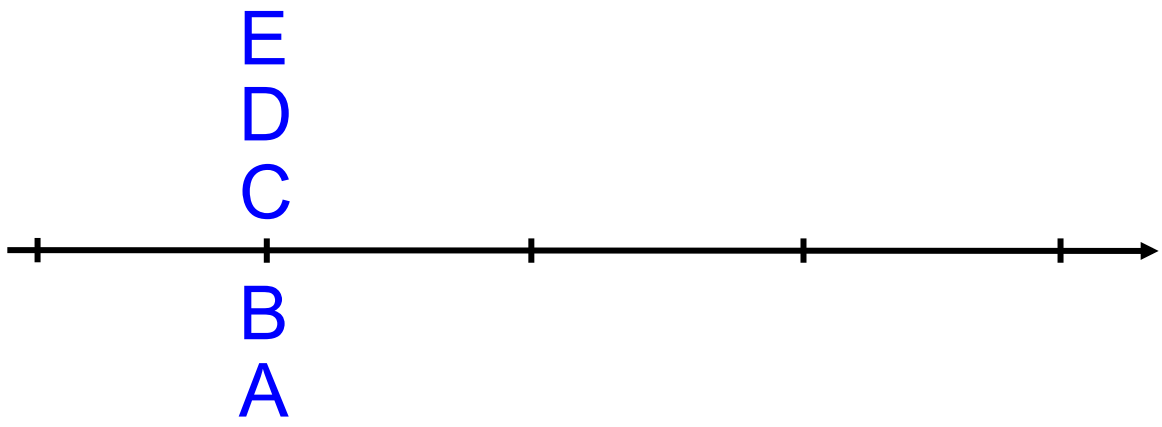
Le parallélisme

[await A ; emit C || await B ; emit D] ; emit E



Le parallélisme

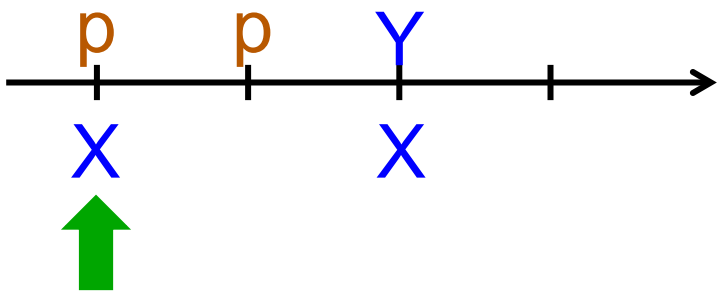
[await A ; emit C || await B ; emit D] ; emit E



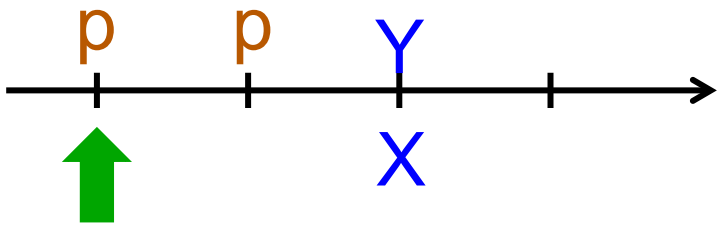
Préemption forte

await X; emit Y

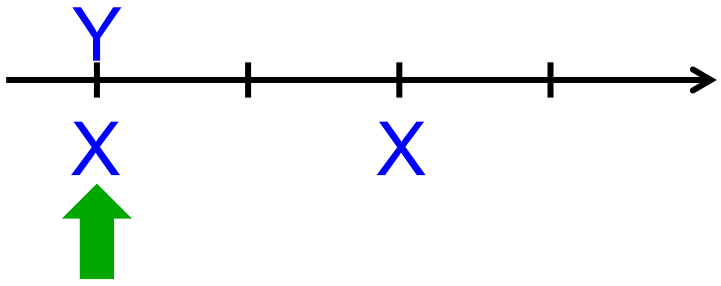
abort p when X;
emit Y



await immediate X;
emit Y

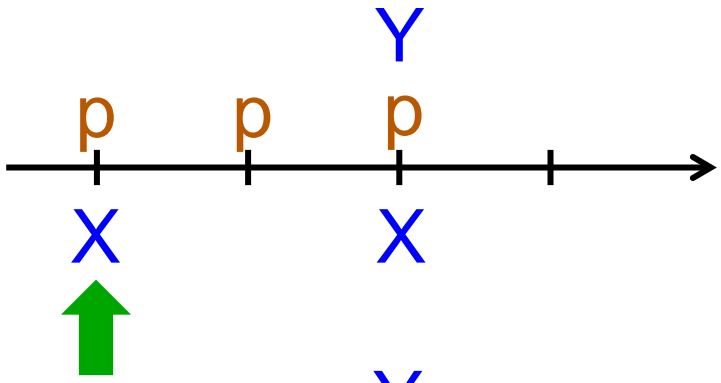


abort p when immediate X;
emit Y

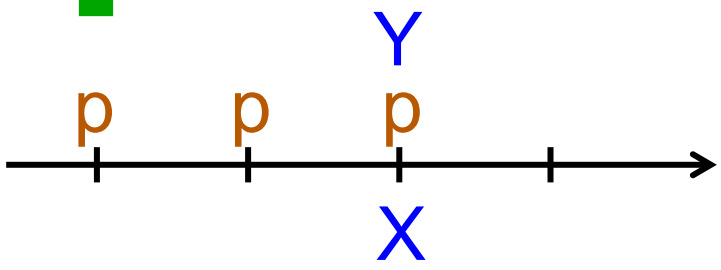


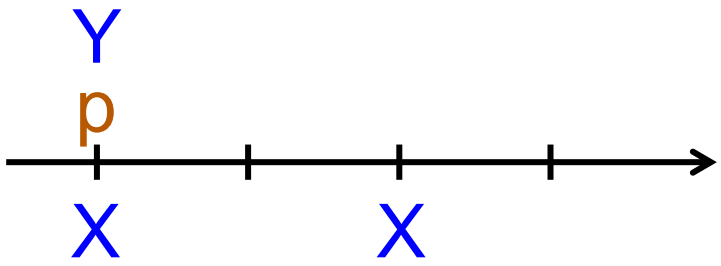
Préemption faible

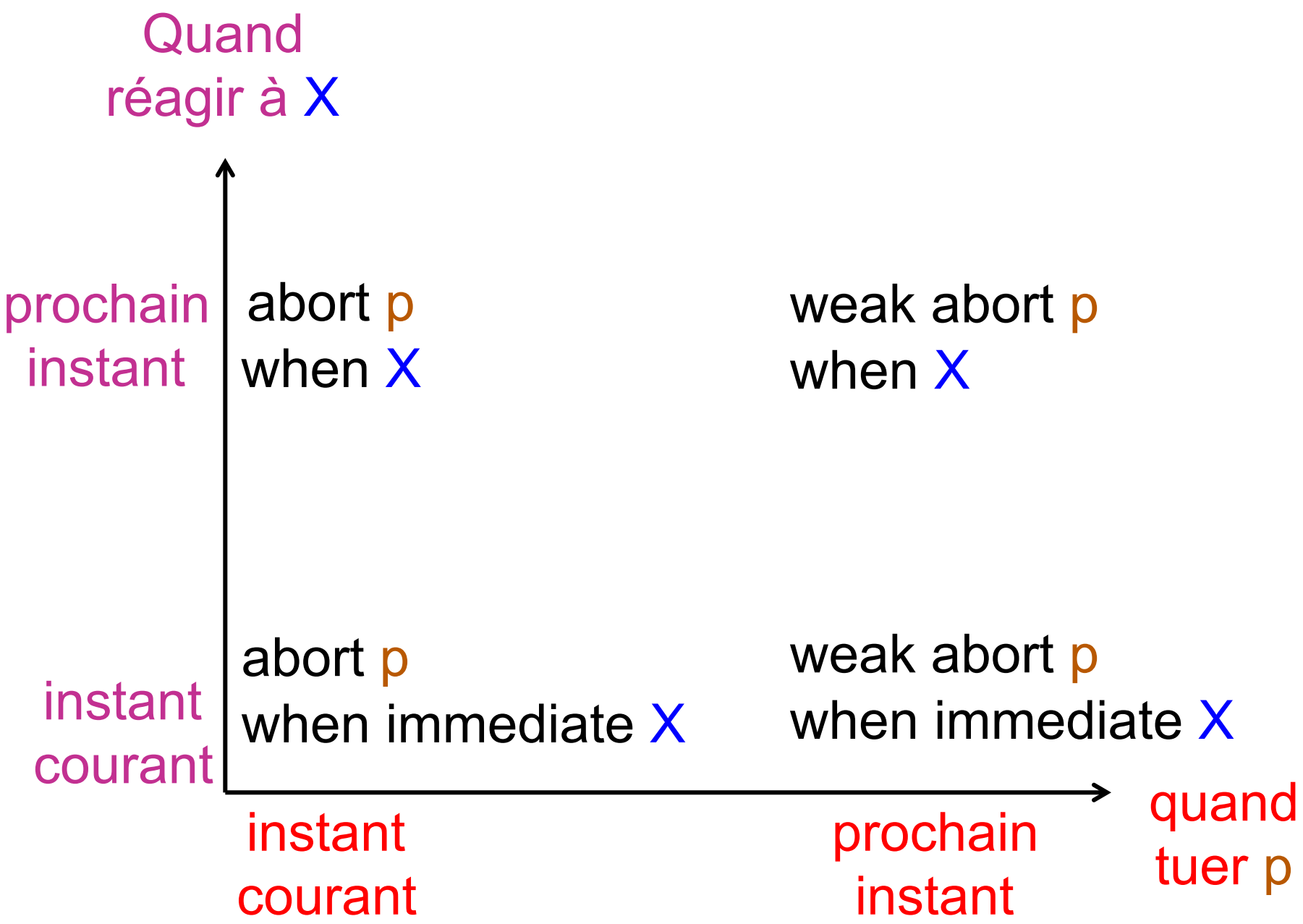
weak abort p when X;
emit Y



weak abort
p
when immediate X;
emit Y

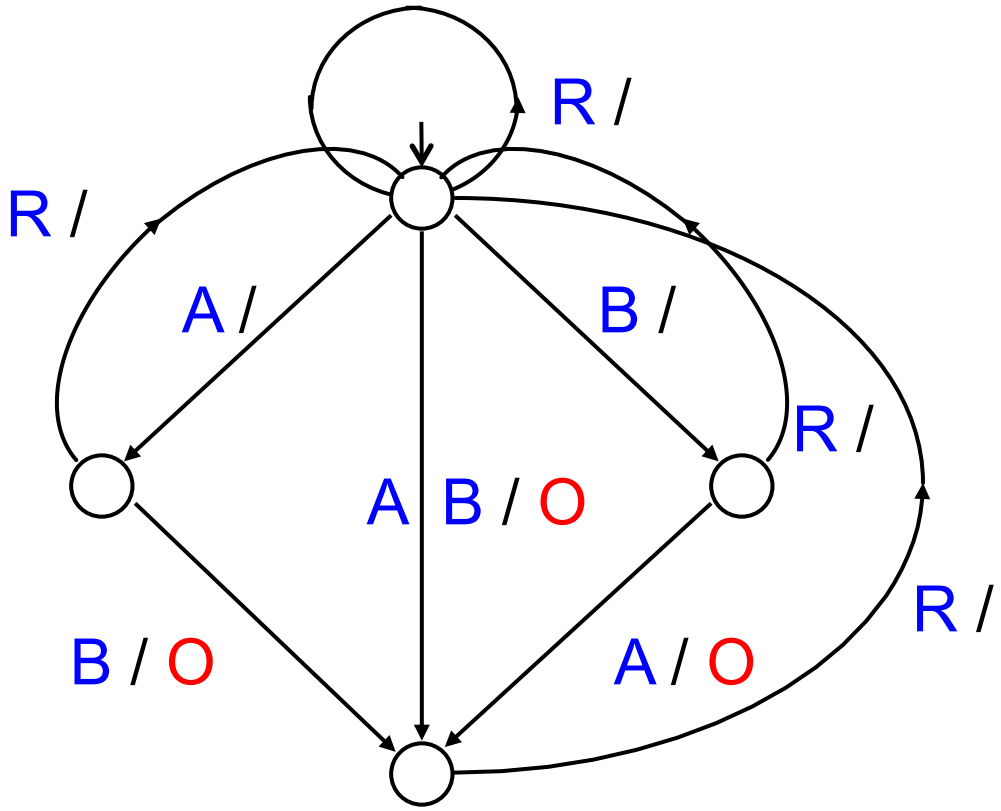





Quand
réagir à X
prochain
instant
abort p
when X
weak abort p
when X
instant
courant
abort p
when immediate X
weak abort p
when immediate X
instant
courant
prochain
instant
quand
tuer p

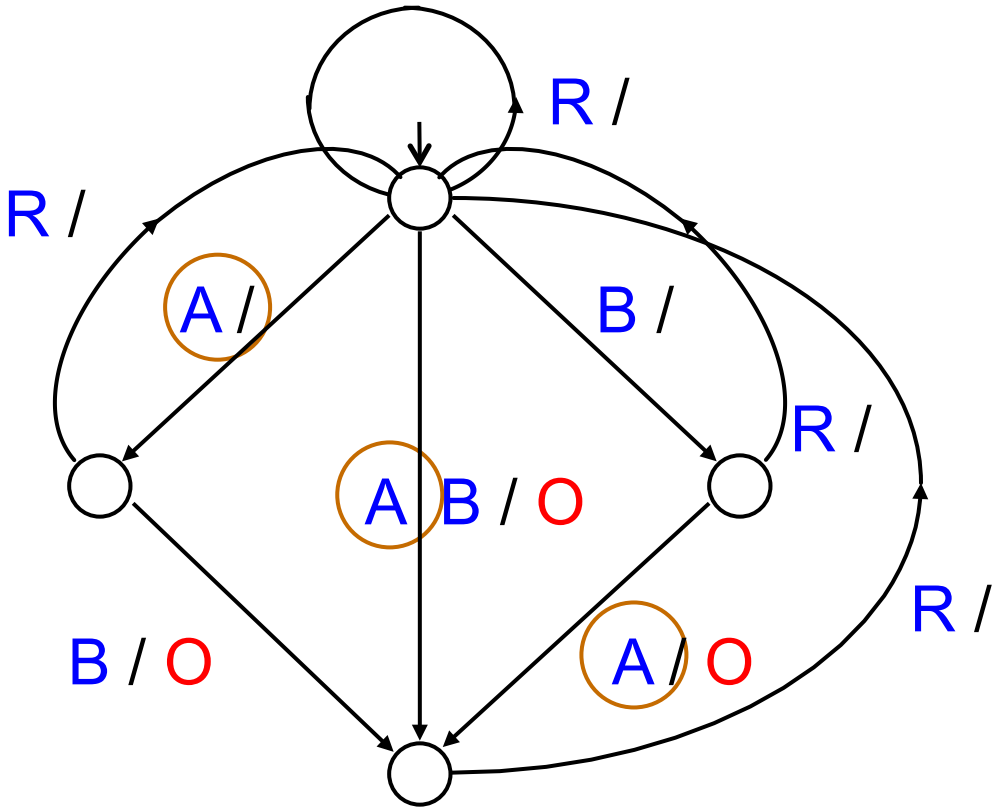
ABRO, le Fibonnacci d'Esterel

Emettre O dès que A et B sont arrivés
Réinitialiser le comportement à chaque R



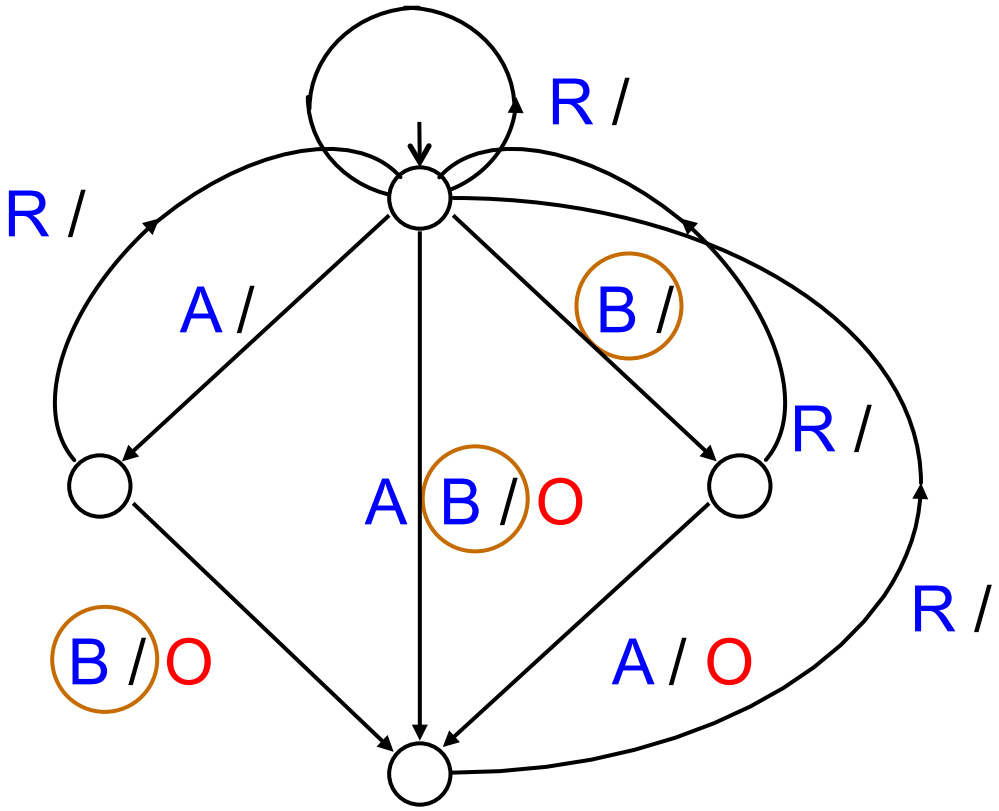
ABRO, le Fibonnacci d'Esterel

Emettre O dès que A et B sont arrivés
Réinitialiser le comportement à chaque R



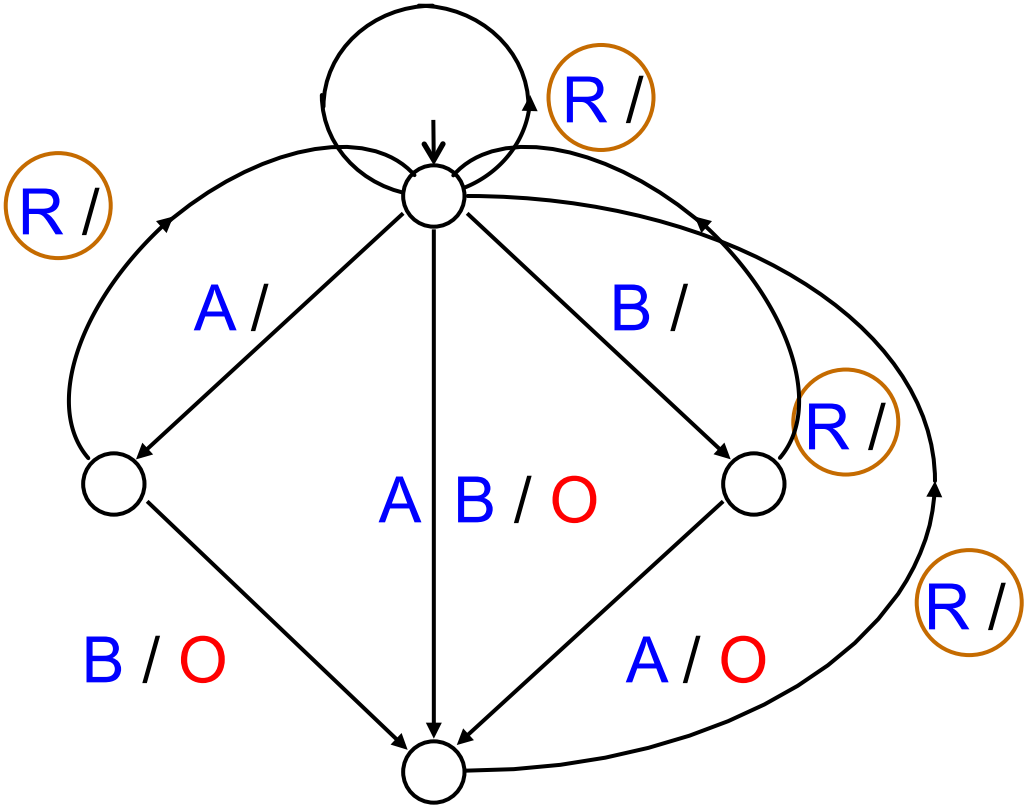
ABRO, le Fibonnacci d'Esterel

Emettre O dès que A et B sont arrivés
Réinitialiser le comportement à chaque R



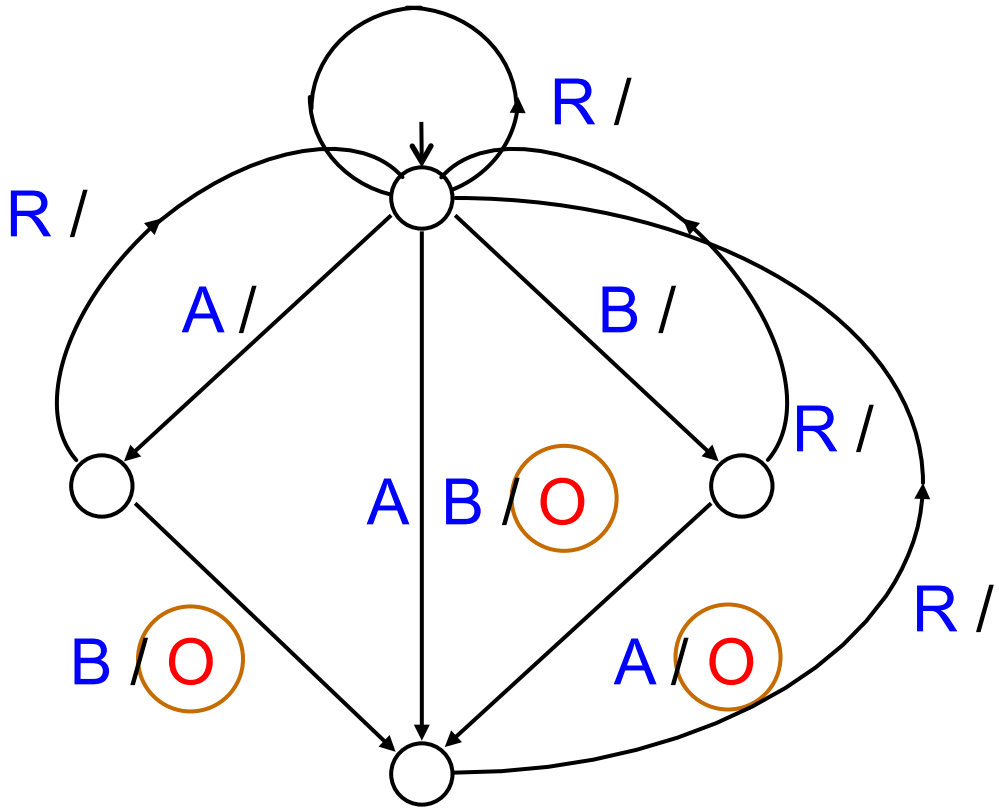
ABRO, le Fibonnacci d'Esterel

Emettre O dès que A et B sont arrivés
Réinitialiser le comportement à chaque R



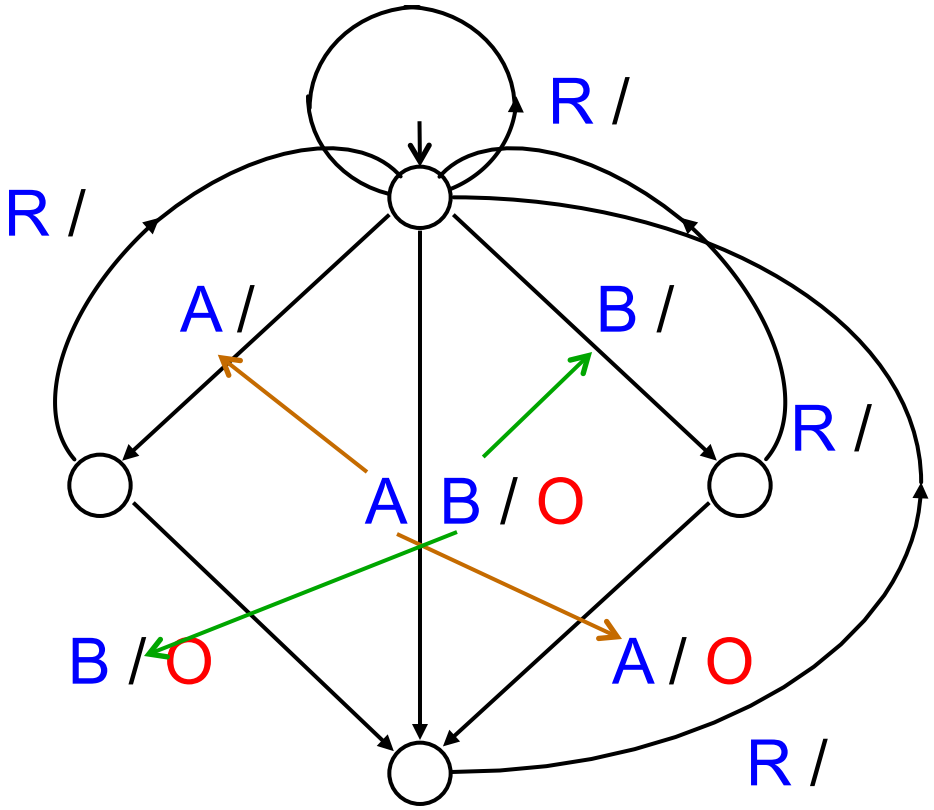
ABRO, le Fibonnacci d'Esterel

Emettre O dès que A et B sont arrivés
Réinitialiser le comportement à chaque R



quid si A, B, R
au même instant ?

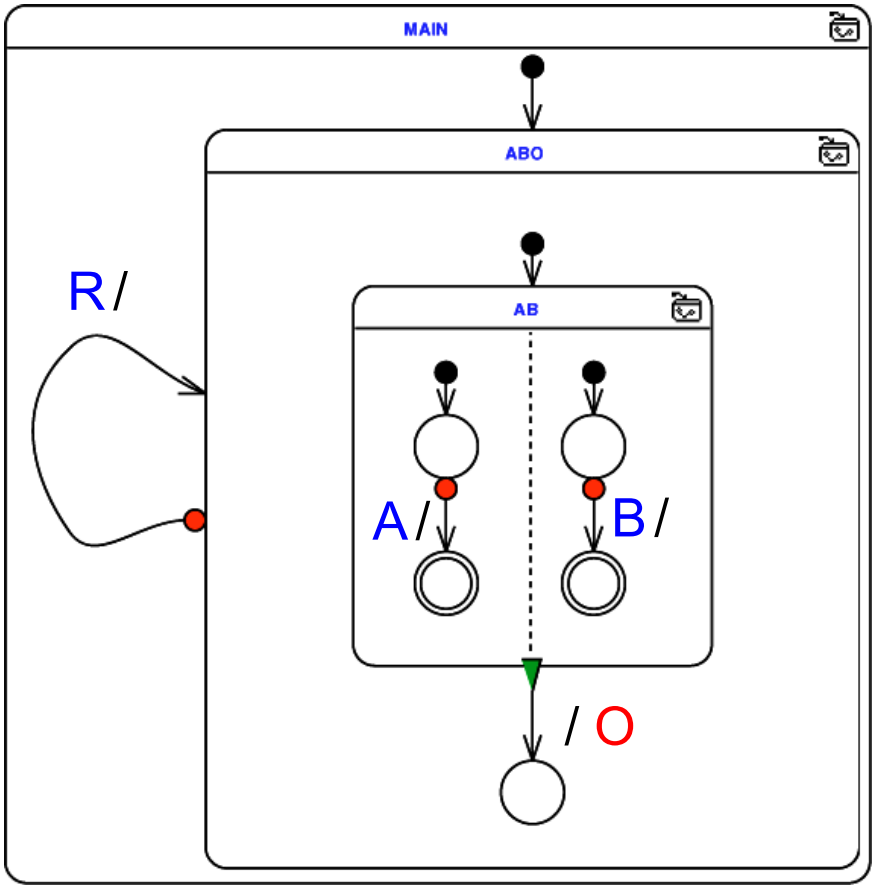
Esterel = spécification linéaire




```
loop
   abort
        { await A || await B };
        emit O;
        halt
   when R
end loop
```

copies = résidus !
Esterel = partage des résidus

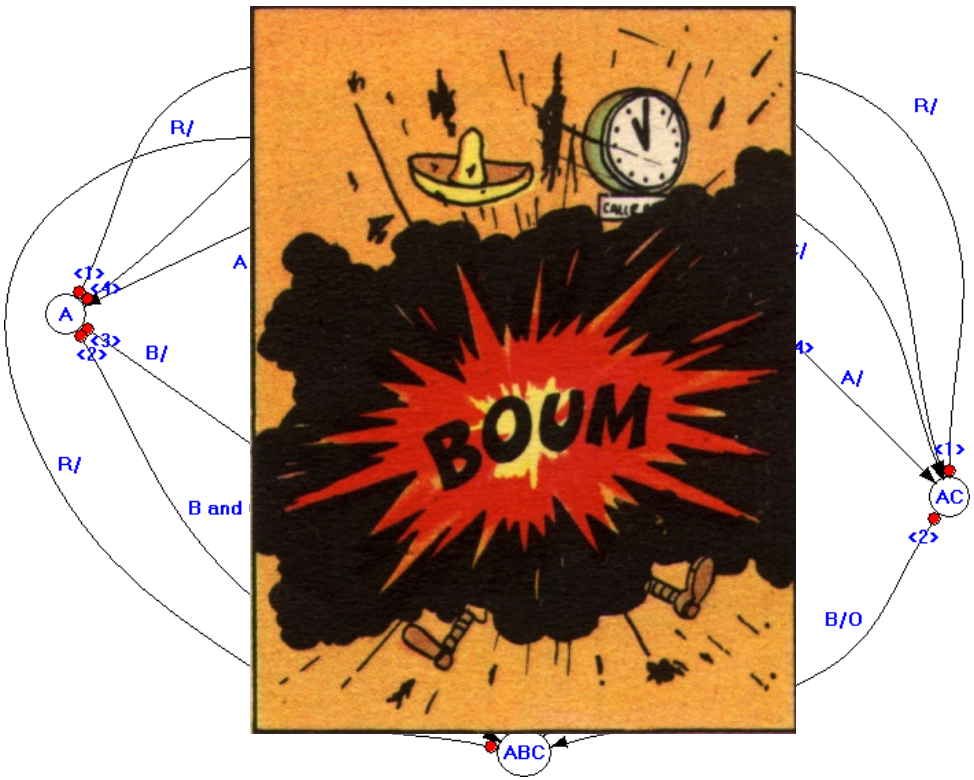
SyncCharts (C. André)



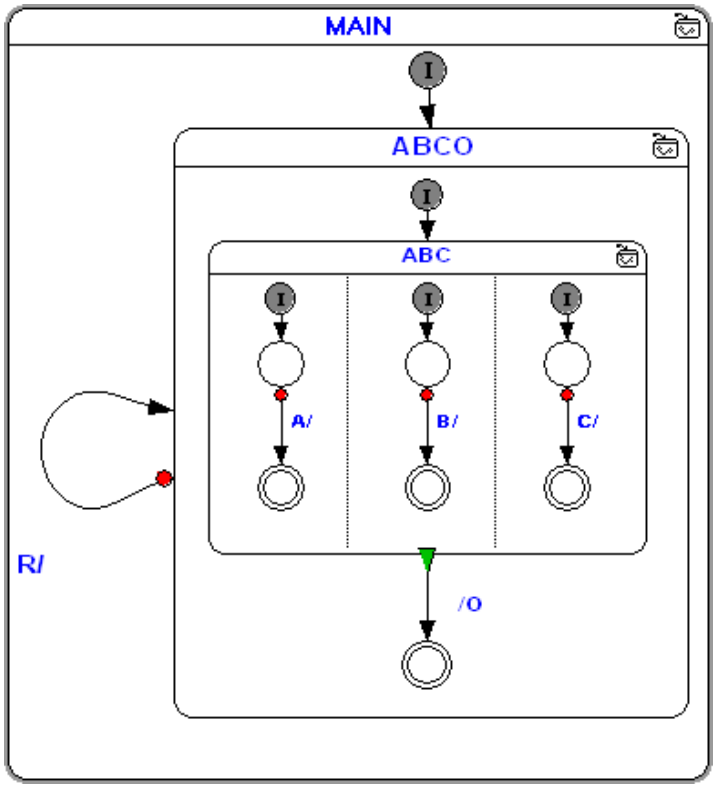

```
loop
   abort
        { await A || await B };
        emit O;
        halt
   when R
end loop
```

automates parallèles
hiérarchiques synchrones
(Statecharts synchrones)

Esterel / SyncCharts = linéaire



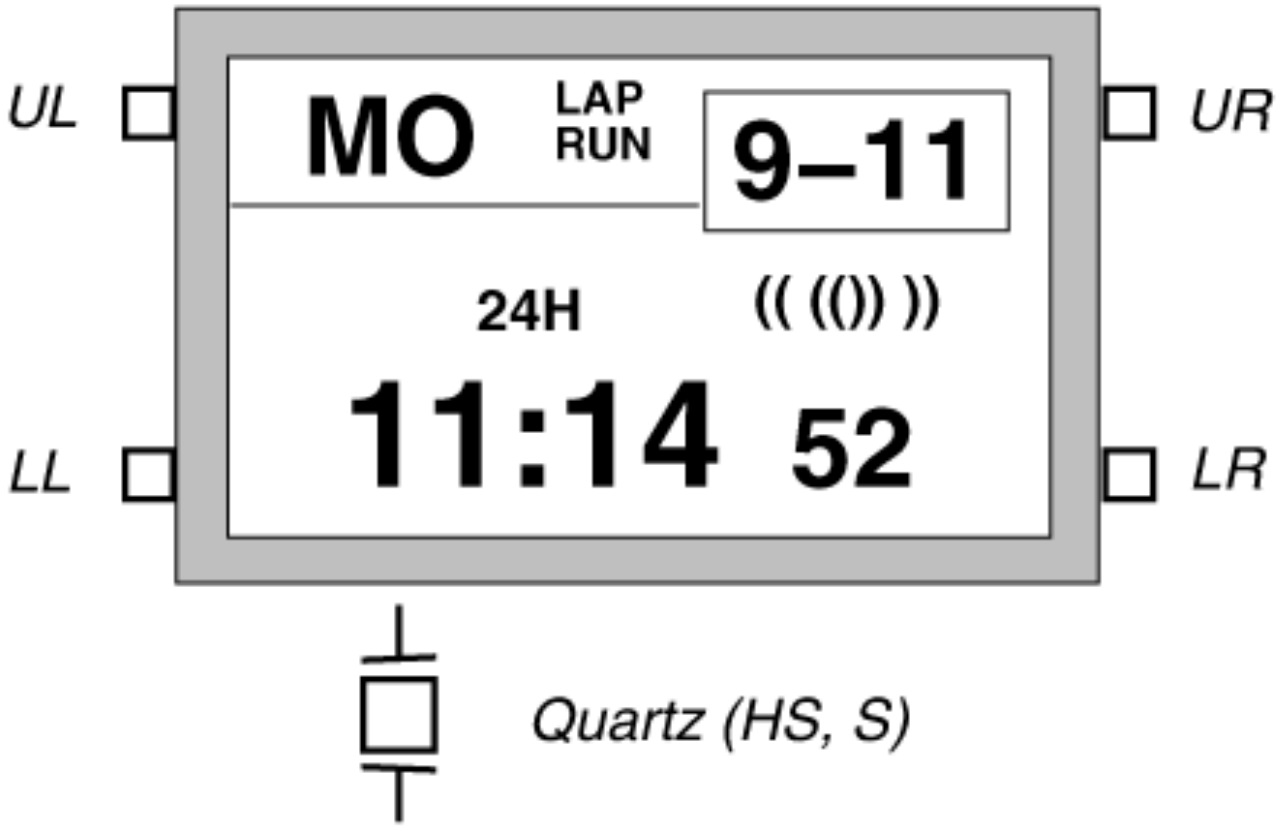
automate plat



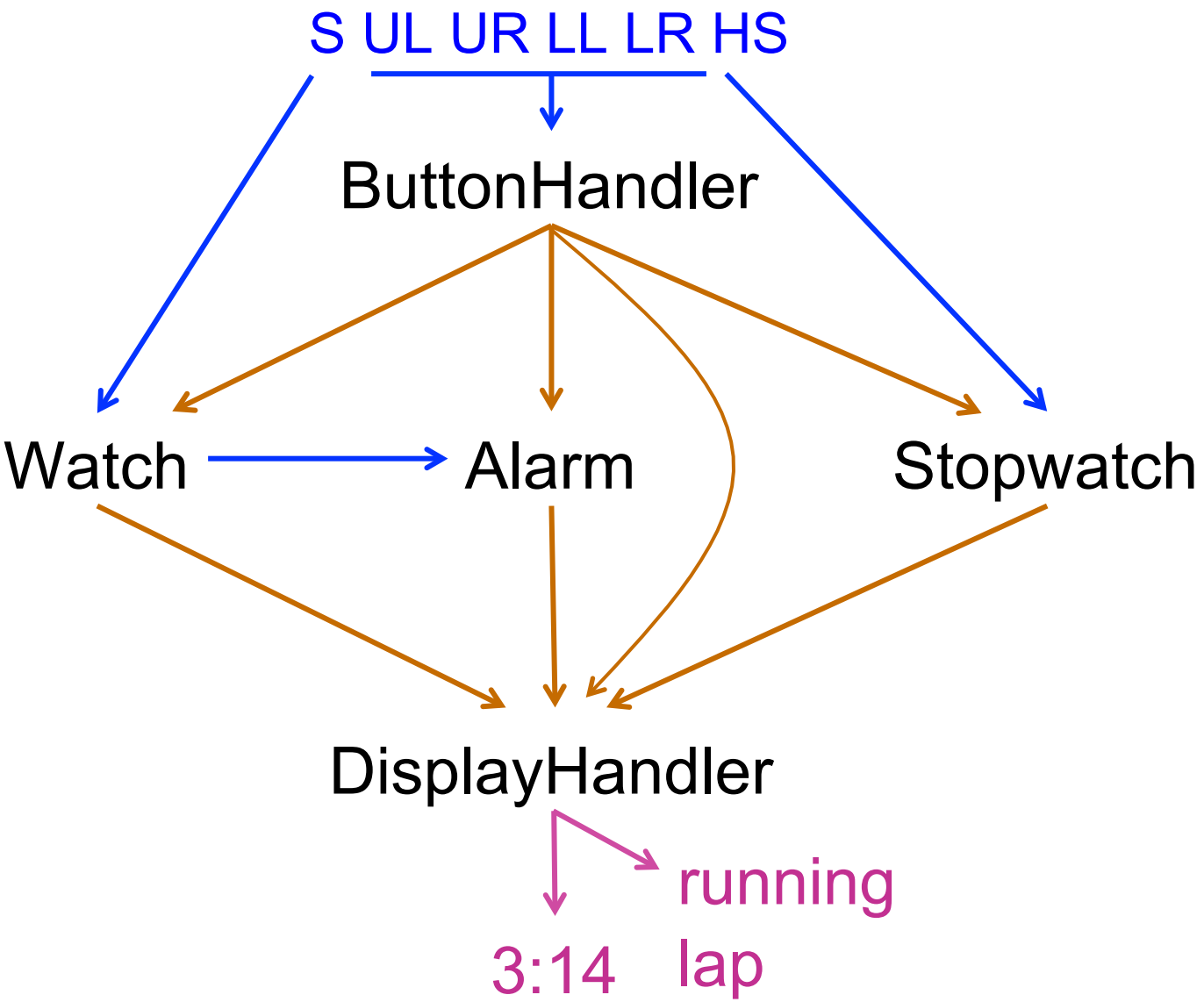
Esterel / SyncCharts
linéaire

source: l'oreille cassée, Hergé

La montre digitale



La montre digitale



ButtonHandler

```
output StartStopCommand, LapCommand ;
loop
   abort
      // watch mode ...
   when LL;
   abort
     // stopwatch mode
        every LR do emit StartStopCommand
     ||
        every UR do emit LapCommand
   when LL;
   abort
      // alarm mode ...
   when LL;
end loop
```

```
output DisplayedTime : Time;                    // Stopwatch
trap Reset in
   signal InternalTime : Time, Stopped in
      emit InternalTime(ZeroStopwatchTime)
      loop
         abort sustain Stopped when StartStopCommand ;
         abort
            every HS do
               emit InternalTime(Increment(pre(?InternalTime)))
            end
         when StartStopCommand
      end loop
   ||
      loop
         abort sustain DisplayedTime(?InternalTime) when LapCommand ;
         present Stopped then exit Reset end;
         await LapCommand
      end loop
   end signal
end trap
```

Un peu d'histoire

- 1982 : concours Microsystèmes : voitures autonomes
  - idées du temps multiforme et du synchronisme parfait
  - débuts du langage : J-P. Rigault, J-P. Marmorat
- 1983 : solidification du langage, essais sémantiques
  - Synchronous CCS (SCCS) de Milner → pas suffisant!
- 1984 : première sémantique, étude de la causalité,
  - L. Cosserat, compilateur Esterel v1, très expérimental
- 1985-86 : le vrai démarrage
  - compilateur complet, Esterel v2, GB et P. Couronné
  - résiduelle de Brzozowski pour Esterel (cf cours 5 de 2010)
  - mais lent et explosif !

- 1986-89 : Premiers utilisateurs expérimentaux
  - Dassault Aviation (E. Ledinot), Bertin, Renault, Thomson
- 1986-89 : étude profonde de la causalité
  - théorie des potentiels, compilation rapide (G. Gonthier)
  - Esterel v3 (équipe), toujours explosif
  - vérification par automates : Auto / Autograph
  - industrialisation : CISI Ingéniérie → ILOG
- 1990-92 : compilation par circuits, Digital Equipment
  - plus d'explosion → Esterel v4
  - vérification / optimisation formelle par BDDs : SIS, TiGeR, J.C. Madre, O. Coudert, H. Touati

- 1992-2000 :
  - → sémantique constructive
  - → Esterel v5
  - → Dassault, Thomson, Bell Labs, etc.
  - → circuits, codesign : Synopsys, Cadence, Intel
  - → Columbia Compiler, S. Edwards
  - → SynchCharts (C. André)

- 2000-2013 : Création d'Esterel Technologies
  - → Esterel v7
  - → Quartz, K. Schneider (Kaiserslautern)
  - → HipHop, GB, M. Serrano, C. Nicolas

A suivre !

Numérotation des traps

```
trap T in
   trap U in
      nothing0
   ||
      pause1
   ||
      exit U2
   ||
      exit T3
   end trap
||
   exit T2
end trap
```

Si deux traps sont levés en même temps, seul le plus extérieur compte

Code de retour du parallèle
= max des codes des branches
(codage de Gonthier)

Esterel noyau

nothing	0
emit s	!s
pause	1
present s then p else q	s ? p, q
suspend p when s	s ⊃ p
p;q	p;q
p || q	p|q
loop p end	p*
trap T in p end	{ p}
	↑p
exit T^k	k
signal s in p end	p \ s

Instructions dérivées

- if s then p end = if s then p else nothing end
  = s? p,0
- halt = loop pause end = 1*
- await s = trap T in
    loop pause ; if s then exit T end loop
  end
  = (1;s?2,0)* }
- await immediate s = if s then nothing
  else
    await s
  end if

- abort p when s = trap T in
  suspend p when s
  ||
  await s; exit T
  end trap

s? p ={ s⊃↑p [(1;s?2,0)*}}

- loop p each s = loop
  abort p ; halt when s
  end loop

- every s do p end = await s ; loop p each s

Les signaux en Esterel v5

s

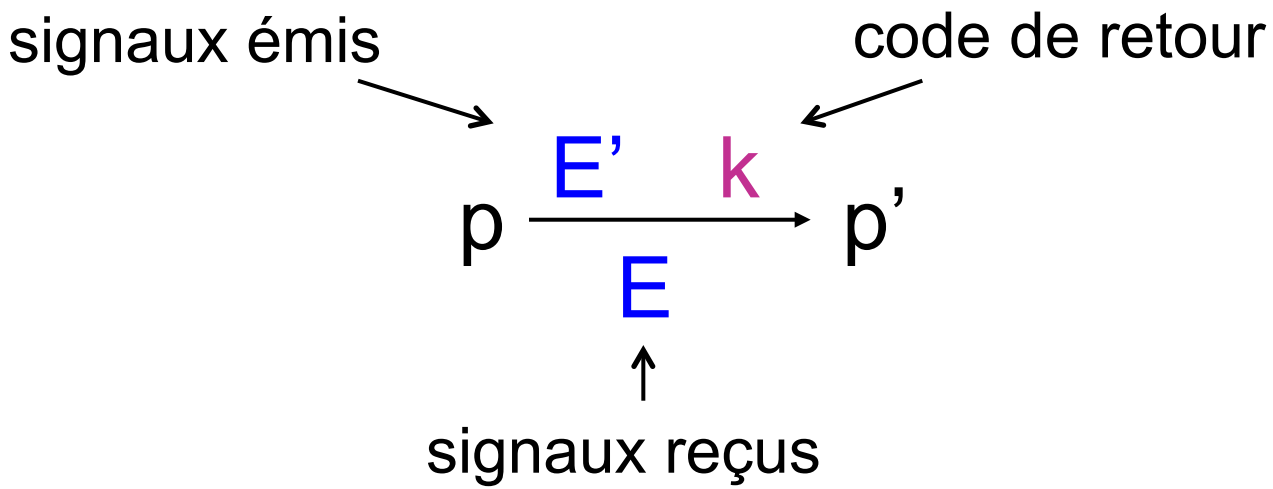
statut s : présent / absent, 0/1, + / -

optionnel : valeur ?s dans un type quelconque T
fonction de combinaison f : T → T
associative et commutative

- statut présent ssi s émis dans l'instant
- valeur ?s :
  combinaison par f de l'ensemble des valeurs émises

Dans la portée lexicale du signal s, à chaque instant
- toutes les instructions voient le même statut s
- toutes les instructions voient la même valeur ?s

La sémantique comportementale



Diffusion : E’ ⊆ E

k

- 0 : terminaison
- 1 : pause
- 2 : sortie d’un niveau de trap
- 3 : sortie de deux niveaux de trap

Sémantique comportementale logique (1)

$$k \xrightarrow[E]{\emptyset, k} 0 \qquad !s \xrightarrow[E]{\{s\}, 0} 0$$

$$\frac{s \in E \qquad p \xrightarrow[E]{E', k} p'}{s?\,p,q \xrightarrow[E]{E', k} p'} \qquad \frac{s \notin E \qquad q \xrightarrow[E]{F', l} q'}{s?\,p,q \xrightarrow[E]{F', l} s?\,q'}$$

$$\frac{p \xrightarrow[E]{E', k} p'}{s \supset p \xrightarrow[E]{E', k} s \multimap p'}$$

avec $s \multimap p' = \{s?1,2\}^*(s \supset p')$

Sémantique comportementale logique (2)

$$\frac{p \xrightarrow[E]{E',k} p' \quad k \neq 0}{p;q \xrightarrow[E]{E',k} p';q}$$

$$\frac{p \xrightarrow[E]{E',0} p' \quad q \xrightarrow[E]{F',l} q'}{p;q \xrightarrow[E]{E' \cup F',l} q'}$$

$$\frac{p \xrightarrow[E]{E',k} p' \quad k \neq 0}{p^* \xrightarrow[E]{E',k} p';p^*}$$

pas de règle pour $k = 0$!

$$\frac{p \xrightarrow[E]{E',k} p' \qquad q \xrightarrow[E]{F',l} q'}{p|q \xrightarrow[E]{E' \cup F',\max(k,l)} p'|q'}$$

Sémantique comportementale logique (3)

$$\frac{p \xrightarrow[E]{E',k} p' \quad k \in \{0,2\}}{\{p\} \xrightarrow[E]{E',0} 0}$$

$$\frac{p \xrightarrow[E]{E',k} p' \quad (k=1, \downarrow k=1) \text{ ou } (k>2, \downarrow k = k-1)}{\{p\} \xrightarrow[E]{E',\downarrow k} \{p'\}}$$

$$\frac{p \xrightarrow[E]{E',k} p' \quad (k \le 1, \uparrow k = k) \text{ ou } (k>1, \uparrow k = k+1)}{\uparrow p \xrightarrow[E]{E',\uparrow k} \uparrow p'}$$

Sémantique comportementale logique (4)

$$\frac{p \xrightarrow[E]{E', k} p' \quad s \notin E, s \notin E'}{p \backslash s \xrightarrow[E]{E', k} p' \backslash s}$$

absence

$$\frac{p \xrightarrow[E \cup \{s\}]{E' \cup \{s\}, k} p' \quad s \in E, s \in E'}{p \backslash s \xrightarrow[E]{E', k} p' \backslash s}$$

présence

si une seule règle s'applique => déterminisme
Mais si les deux s'appliquent, problème !

if s then emit s end ???

Cycles de causalité

- if s else emit s end s?0,!s

  contradiction pour la causalité

- if s then emit s end s?!s,0

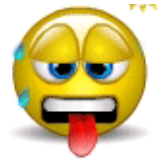
  pas contradictoire, mais deux choix possibles

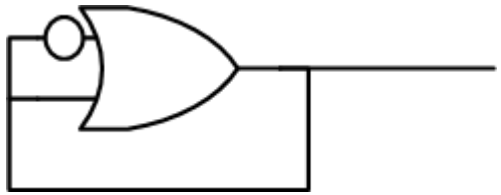
- if s then emit s else emit s end s?!s,!s

  pas contradictoire, un seul choix possible

  mais problématique car non constructif!

Hamlet : ToBe = ToBe or not ToBe





Propagation constructive

```
   emit x
|| if not x then emit y
|| if not y then emit z
```

must

```
   emit x
|| if not x then emit y
|| if not y then emit z
```

$\Rightarrow x = 1$

cannot

```
   emit x
|| if not x then emit y
|| if not y then emit z
```

$\Rightarrow y = 0$

car les émetteurs de y ont disparu

must

```
   emit x
|| if not x then emit y
|| if not y then emit z
```

$\Rightarrow z = 1$

Acceptation des bons cycles

```
if I then
   if X then emit Y
else
   if Y then emit X
end
```

X→Y xor Y→X
grâce au if

```
if X then emit Y;
pause;
if Y then emit X
```

X→Y xor Y→X
grâce au pause

Comme pour l'analyse constructive des circuits cycliques, cf. cours 1 !

Références

- *The Foundations of Esterel*
  G. Berry. In *Proof, Language and Interaction: Essays in Honour of Robin Milner*, MIT Press, Foundations of Computing Series, 2000.
- *The Esterel v5_91 Primer*
  G. Berry. Web book, www-sop.inria.fr/members/Gerard/Berry, 2001
- *Programmation Synchrone de Systèmes Réactifs avec Esterel et les SyncCharts*.
  Luigi Zaffalon. Presses Polytechniques Romandes, 2005.
- *The Constructive Semantics of Pure Esterel*
  G. Berry. web book, www-sop.inria.fr/members/Gerard/Berry, 2001
- *Compiling Esterel*
  D. Potop-Butucaru, S. Edwards et G. Berry, Springer, 2008
- *Synchronous Programming of Reactive Systems*.
  N. Halbwachs. Kluwer Academic Publishers, 1993.
- *Modeling Reactive Systems with Statecharts : the Statemate Approach*
  D. Harel et M. Politi. McGraw-Hill, 1998.