

Master
systèmes embarqués et mobiles sûrs



UE : SMB204
Systèmes Embarqués et Enfouis

Le langage Esterel

Jean-Ferdinand Susini

Maître de Conférences, CNAM

département Informatique - Laboratoire CEDRIC

le cnam

Paris, 30 oct. 2012



Master
systèmes embarqués et mobiles sûrs



UE : SMB204
Systèmes Embarqués et Enfouis

Présentation

*d'après les supports de Gérard BERRY, Frédéric Boussinot,
Marc Pouzet...*

le cnam

Paris, 30 oct. 2012

Esterel

- “Le plus ancien des langages synchrones” (J.-P. Marmorat et J.-P. Rigault en 82)
- Crée à l’École des Mines de Paris et l’INRIA Sophia-Antipolis (projet commun : MEIJE ; G. GONTIER, G. BERRY)
- ~~Distribution commerciale : Esterel Technologies avec le produit Esterel Studio (SyncCharts I3S, C. ANDRE)~~
- Orientation : spécification et vérification de circuits, SoC, intégration d’IPs,...
- génération automatique de VHDL, C, System C

Caractéristiques

- instructions d'ordonnancement ; style impératif : séquence, boucle, composition parallèle, pause, ...
- communication par diffusion de signaux purs (présent/absent) ou valués (données numériques : seuils événements ponctuels, capteurs lisibles continu,...)
- instructions événementielles : émission, attente de signaux, test instantané, préemption, ...
- Programmation modulaire (modularité descendante)
- déclaration des signaux (portée locale, entrée, sortie ou les deux)
- Gestion de tâches asynchrones

Caractéristiques

- fonction de combinaison des valeurs de signaux
- Configuration de signaux (simultanéité, absence, multiplicité)
- Gestion de variables
- Modèle synchrone :
 - Contrôle exécuté en temps nul
 - Diffusion des signaux en temps nul

Exemple

loop

await 60 SECOND;

emit MINUTE

end loop

- émet le signal MINUTE toutes les 60 occurrences du signal SECOND ; l'émission de MINUTE est simultanée à la 60-ème occurrence de SECOND
- ***Un événement*** : un ensemble d'occurrences simultanées de signaux
 - e.x : {SECOND}, {SECOND, MINUTE},...
 - Un signal (pur) particulier *tick* est présent à chaque instant (dans chaque événement) : **tick définit l'horloge de base**

Exemple 2

- Un pilote de bouton souris

```
module souris: % <- nom du module
    input click,top; % <- déclarations E/S
    output double,simple;

    loop % <- un programme réactif est généralement
        % une boucle infinie
        await click; % <- attente d'un signal
        abort
            abort % <- préemption
            halt
        when 4 top do
            emit simple % <- émission d'un signal
        end abort
        when click do
            emit double
        end abort
    end loop
end module
```

Exemple 3

- Un robot

```
module robot:  
    input seconde,devant;  
    output avancer,reculer,droite;  
  
    loop  
        abort  
            sustain avancer % <- maintenir le signal  
        when devant do  
            abort  
                sustain reculer  
            when 5 seconde;  
            emit droite;  
            pause  
        end abort  
    end loop  
end module
```

Exemple 4

- Signaux valued, approximation de la vitesse instantanée

```
module vitesse:  
    constant circonference: integer;  
    input seconde, tourderoue;  
    output vitesseinstantanee: integer;  
  
loop  
    var nb:=0 : integer in % <- variable  
    abort  
        every tourderoue do  
            nb:=nb+1;  
        end every  
    when seconde do  
        emit vitesseinstantanee(nb*circonference)  
            % ^ signal valued  
    end abort  
    end var  
end loop  
end module
```

Entraînement matinal

```
module runner:  
    input seconde, meter, lap, morning, step;  
    ...  
    every morning do  
        abort  
        loop  
            abort RunSlowly when 15 seconde;  
            abort  
                every step do  
                    Jump || Breathe  
                end every  
                when 100 metre;  
                FullSpeed  
                each lap  
                when 2 lap  
            end every  
    end module
```

Entraînement matinal sécurisé

```
every morning do
    trap HeartAttack in
        abort
        loop
            abort RunSlowly when 15 seconde;
            abort
                every step do
                    Jump || Breathe || CheckHeart
                end every
                when 100 metre;
                FullSpeed
                    each lap
                    when 2 lap
    handle HeartAttack do
        GoToHospital
    end trap
end every
```

Éléments de syntaxe

- nop : **nothing**
- Séquence : $P ; Q$
- Parallélisme : $P \parallel Q$
- fin d'instant explicite : **pause** (**await tick**), **halt**
- boucles : **loop** P **end loop**, **repeat** exp **times** P **end repeat**
- affectation $X := 1$
- appel de procédure : **call** $P (X_1, X_2) (R_1, R_2)$
- émission : **emit** S
- attente : **await** S , **await immediate** S
- test : **present** S **then** P **else** Q **end present**
- déclaration locale : **signal** S **in** P **end signal**
- préemption : **abort** P **when** S , **weak abort** P **when** S
- modules : **module** M : ... **end module** et **run** M [formel/effectif]

Ordonnancement

- Séquence :
 - **p ; q**
 - l'exécution de q commence dès que p termine (en temps nul : lien de dépendance causal)
 - Exemple :
x:=1 ; emit S(x)
 - ⇒ émission de S avec la valeur 1

Ordonnancement

- Parallélisme :
 - **p || q**
 - p et q s'exécutent en parallèle. La construction parallèle termine lorsque p et q ont tous les deux terminé.
 - Exemple :
[await A || await B] ; emit S
 - émission de S lorsque A et B ont été reçus (pas forcément simultanément)

Ordonnancement

- Boucle indéfinie :
 - **loop p end loop**
 - réitère l'exécution de p dès que celui-ci termine (ré-exécution au même instant que sa terminaison)
 - sémantique classique : **loop p end loop = p;**
loop p end loop
 - danger des boucles instantanées !
 - Exemple : **loop await A ; emit S end loop**
~~**loop await immediate A ; emit S end loop**~~

Ordonnancement

- Boucle finie :
 - **repeat n times p end repeat**
 - réitère n fois l'exécution de p dès que celui-ci termine (ré-exécution au même instant que sa terminaison)
 - n est un entier positif ou nul. n peut être obtenue par une expression qui est alors évaluée au début de l'exécution
 - Exemple :
repeat 5 times await A ; emit S end repeat

Attente et émission de signaux

- **await S** : attente de la prochaine occurrence du signal S dans le futur strict (premier instant exclu)
- **await immediate S** : termine immédiatement si S est présent au premier instant
- **emit S** : émission instantanée du signal S
- **sustain S** : émission continue (d'instant en instant) du signal S
- cas des signaux valués : **emit S(e)**
- Exemple : **await A; emit B**

Entrées					
Sorties					

Attente et émission de signaux

- **await S** : attente de la prochaine occurrence du signal S dans le futur strict (premier instant exclu)
- **await immediate S** : termine immédiatement si S est présent au premier instant
- **emit S** : émission instantanée du signal S
- **sustain S** : émission continue (d'instant en instant) du signal S
- cas des signaux valués : **emit S(e)**
- Exemple : **await A; emit B**

	tick				
Entrées	A				
Sorties	-				

Attente et émission de signaux

- **await S** : attente de la prochaine occurrence du signal S dans le futur strict (premier instant exclu)
- **await immediate S** : termine immédiatement si S est présent au premier instant
- **emit S** : émission instantanée du signal S
- **sustain S** : émission continue (d'instant en instant) du signal S
- cas des signaux valués : **emit S(e)**
- Exemple : **await A; emit B**

	tick	tick			
Entrées	A	-			
Sorties	-	-			

Attente et émission de signaux

- **await S** : attente de la prochaine occurrence du signal S dans le futur strict (premier instant exclu)
- **await immediate S** : termine immédiatement si S est présent au premier instant
- **emit S** : émission instantanée du signal S
- **sustain S** : émission continue (d'instant en instant) du signal S
- cas des signaux valués : **emit S(e)**
- Exemple : **await A; emit B**

	tick	tick	tick		
Entrées	A	-	-		
Sorties	-	-	-		

Attente et émission de signaux

- **await S** : attente de la prochaine occurrence du signal S dans le futur strict (premier instant exclu)
- **await immediate S** : termine immédiatement si S est présent au premier instant
- **emit S** : émission instantanée du signal S
- **sustain S** : émission continue (d'instant en instant) du signal S
- cas des signaux valués : **emit S(e)**
- Exemple : **await A; emit B**

	tick	tick	tick	tick	
Entrées	A	-	-	A	
Sorties	-	-	-	B	

Attente et émission de signaux

- **await S** : attente de la prochaine occurrence du signal S dans le futur strict (premier instant exclu)
- **await immediate S** : termine immédiatement si S est présent au premier instant
- **emit S** : émission instantanée du signal S
- **sustain S** : émission continue (d'instant en instant) du signal S
- cas des signaux valués : **emit S(e)**
- Exemple : **await A; emit B**

	tick	tick	tick	tick	tick...
Entrées	A	-	-	A	-...
Sorties	-	-	-	B	-...

Attente et émission de signaux

- **await S** : attente de la prochaine occurrence du signal S dans le futur strict (premier instant exclu)
- **await immediate S** : termine immédiatement si S est présent au premier instant
- **emit S** : émission instantanée du signal S
- **sustain S** : émission continue (d'instant en instant) du signal S
- cas des signaux valués : **emit S(e)**
- Exemple : **await A; emit B**

Entrées					
Sorties					

Attente et émission de signaux

- **await S** : attente de la prochaine occurrence du signal S dans le futur strict (premier instant exclu)
- **await immediate S** : termine immédiatement si S est présent au premier instant
- **emit S** : émission instantanée du signal S
- **sustain S** : émission continue (d'instant en instant) du signal S
- cas des signaux valués : **emit S(e)**
- Exemple : **await A; emit B**

	tick				
Entrées	-				
Sorties	-				

Attente et émission de signaux

- **await S** : attente de la prochaine occurrence du signal S dans le futur strict (premier instant exclu)
- **await immediate S** : termine immédiatement si S est présent au premier instant
- **emit S** : émission instantanée du signal S
- **sustain S** : émission continue (d'instant en instant) du signal S
- cas des signaux valués : **emit S(e)**
- Exemple : **await A; emit B**

	tick	tick			
Entrées	-	-			
Sorties	-	-			

Attente et émission de signaux

- **await S** : attente de la prochaine occurrence du signal S dans le futur strict (premier instant exclu)
- **await immediate S** : termine immédiatement si S est présent au premier instant
- **emit S** : émission instantanée du signal S
- **sustain S** : émission continue (d'instant en instant) du signal S
- cas des signaux valués : **emit S(e)**
- Exemple : **await A; emit B**

	tick	tick	tick		
Entrées	-	-	A		
Sorties	-	-	B		

Attente et émission de signaux

- **await S** : attente de la prochaine occurrence du signal S dans le futur strict (premier instant exclu)
- **await immediate S** : termine immédiatement si S est présent au premier instant
- **emit S** : émission instantanée du signal S
- **sustain S** : émission continue (d'instant en instant) du signal S
- cas des signaux valués : **emit S(e)**
- Exemple : **await A; emit B**

	tick	tick	tick	tick	
Entrées	-	-	A	-	
Sorties	-	-	B	-	

Attente et émission de signaux

- **await S** : attente de la prochaine occurrence du signal S dans le futur strict (premier instant exclu)
- **await immediate S** : termine immédiatement si S est présent au premier instant
- **emit S** : émission instantanée du signal S
- **sustain S** : émission continue (d'instant en instant) du signal S
- cas des signaux valués : **emit S(e)**
- Exemple : **await A; emit B**

	tick	tick	tick	tick	tick...
Entrées	-	-	A	-	-...
Sorties	-	-	B	-	-...

Attente et émission de signaux

- **await S** : attente de la prochaine occurrence du signal S dans le futur strict (premier instant exclu)
- **await immediate S** : termine immédiatement si S est présent au premier instant
- **emit S** : émission instantanée du signal S
- **sustain S** : émission continue (d'instant en instant) du signal S
- cas des signaux valués : **emit S(e)**
- Exemple : **await immediate A; emit B**

Entrées					
Sorties					

Attente et émission de signaux

- **await S** : attente de la prochaine occurrence du signal S dans le futur strict (premier instant exclu)
- **await immediate S** : termine immédiatement si S est présent au premier instant
- **emit S** : émission instantanée du signal S
- **sustain S** : émission continue (d'instant en instant) du signal S
- cas des signaux valués : **emit S(e)**
- Exemple : **await immediate A; emit B**

	tick				
Entrées	A				
Sorties	B				

Attente et émission de signaux

- **await S** : attente de la prochaine occurrence du signal S dans le futur strict (premier instant exclu)
- **await immediate S** : termine immédiatement si S est présent au premier instant
- **emit S** : émission instantanée du signal S
- **sustain S** : émission continue (d'instant en instant) du signal S
- cas des signaux valués : **emit S(e)**
- Exemple : **await immediate A; emit B**

	tick	tick			
Entrées	A	-			
Sorties	B	-			

Attente et émission de signaux

- **await S** : attente de la prochaine occurrence du signal S dans le futur strict (premier instant exclu)
- **await immediate S** : termine immédiatement si S est présent au premier instant
- **emit S** : émission instantanée du signal S
- **sustain S** : émission continue (d'instant en instant) du signal S
- cas des signaux valués : **emit S(e)**
- Exemple : **await immediate A; emit B**

	tick	tick	tick		
Entrées	A	-	-		
Sorties	B	-	-		

Attente et émission de signaux

- **await S** : attente de la prochaine occurrence du signal S dans le futur strict (premier instant exclu)
- **await immediate S** : termine immédiatement si S est présent au premier instant
- **emit S** : émission instantanée du signal S
- **sustain S** : émission continue (d'instant en instant) du signal S
- cas des signaux valués : **emit S(e)**
- Exemple : **await immediate A; emit B**

	tick	tick	tick	tick	
Entrées	A	-	-	A	
Sorties	B	-	-	-	

Attente et émission de signaux

- **await S** : attente de la prochaine occurrence du signal S dans le futur strict (premier instant exclu)
- **await immediate S** : termine immédiatement si S est présent au premier instant
- **emit S** : émission instantanée du signal S
- **sustain S** : émission continue (d'instant en instant) du signal S
- cas des signaux valués : **emit S(e)**
- Exemple : **await immediate A; emit B**

	tick	tick	tick	tick	tick...
Entrées	A	-	-	A	-...
Sorties	B	-	-	-	-...

Attente et test de signaux

- Forme particulières

```
await
  case S1 do P
  case S2 do Q
  ...
end await
```

- Test présence/absence à un instant donné :
 - **present S then p else q end present**
 - Deux formes simplifiées existent :
 - **present S then p end present**
 - **present S else q end present**

Préemption

- Terminer l'exécution d'un programme sur la réception d'un signal
 - équivalent au Ctrl-C d'un shell UNIX (hypothèse synchrone en plus)
- **Préemption forte**

```
abort
  p
when S do
  q
end abort
```

- exécution de p interrompue dès que S est là (avant la réaction de p). En cas de préemption on exécute q (sauf premier instant)
- Si p termine, le bloc termine

Préemption

- **Préemption faible**

weak abort

```
p
when S do
  q
end abort
```

- exécution de *p* interrompue lorsque *S* est là (mais après la réaction de *p*). En cas de préemption on exécute *q* (sauf premier instant)
 - Si *p* termine, le bloc termine
- Pour la préemption faible ou forte, en cas d’imbrication, la priorité est donnée au plus englobant

Exemples

abort

```
  await immediate E;  
  emit C
```

```
when immediate S do emit K end;  
emit D
```

Exemples

abort

await immediate E;

emit C

when immediate S do emit K end;

emit D

Exemples

abort

```
  await immediate E;  
  emit C  
when immediate S do emit K end;  
emit D
```

tick				
E,S				
K,D				

Exemples

abort

```
  await immediate E;  
  emit C  
when immediate S do emit K end;  
emit D
```

tick	tick			
E,S	-			
K,D	-			

Exemples

abort

```
  await immediate E;  
  emit C  
when immediate S do emit K end;  
emit D
```

tick	tick	tick		
E,S	-	-		
K,D	-	-		

Exemples

abort

```
  await immediate E;  
  emit C  
when immediate S do emit K end;  
emit D
```

tick	tick	tick	tick	
E,S	-	-	-	
K,D	-	-	-	

Exemples

abort

```
  await immediate E;  
  emit C  
when immediate S do emit K end;  
emit D
```

tick	tick	tick	tick	tick
E,S	-	-	-	-
K,D	-	-	-	-

Exemples

abort

```
  await immediate E;  
  emit C  
when immediate S do emit K end;  
emit D
```

tick	tick	tick	tick	tick	...
E,S	-	-	-	-	...
K,D	-	-	-	-	...

Exemples

abort

```
  await immediate E;  
  emit C
```

```
when immediate S do emit K end;  
emit D
```

Exemples

abort

 await immediate E;

 emit C

when immediate S do emit K end;

emit D

Exemples

abort

```
  await immediate E;  
  emit C  
when immediate S do emit K end;  
emit D
```

tick				
E				
C,D				

Exemples

abort

```
await immediate E;  
emit C  
when immediate S do emit K end;  
emit D
```

tick	tick			
E	-			
C,D	-			

Exemples

abort

```
  await immediate E;  
  emit C  
when immediate S do emit K end;  
emit D
```

tick	tick	tick		
E	-	S		
C,D	-	-		

Exemples

abort

```
await immediate E;  
emit C  
when immediate S do emit K end;  
emit D
```

tick	tick	tick	tick	
E	-	S	-	
C,D	-	-	-	

Exemples

abort

```
await immediate E;  
emit C  
when immediate S do emit K end;  
emit D
```

tick	tick	tick	tick	tick
E	-	S	-	-
C,D	-	-	-	-

Exemples

abort

```
await immediate E;  
emit C  
when immediate S do emit K end;  
emit D
```

tick	tick	tick	tick	tick	...
E	-	S	-	-	...
C,D	-	-	-	-	...

Exemples

abort

```
  await immediate E;  
  emit C  
when S do emit K end;  
emit D
```

Exemples

abort

```
  await immediate E;  
  emit C  
when S do emit K end;  
emit D
```


Exemples

abort

```
  await immediate E;  
  emit C  
when S do emit K end;  
emit D
```

tick				
-				
-				

Exemples

abort

```
  await immediate E;  
  emit C  
when S do emit K end;  
emit D
```

tick	tick			
-	-			
-	-			

Exemples

abort

```
  await immediate E;  
  emit C  
when S do emit K end;  
emit D
```

tick	tick	tick		
-	-	S		
-	-	K, D		

Exemples

abort

```
await immediate E;  
emit C  
when S do emit K end;  
emit D
```

tick	tick	tick	tick	
-	-	S	E	
-	-	K, D	-	

Exemples

abort

```
await immediate E;  
emit C  
when S do emit K end;  
emit D
```

tick	tick	tick	tick	tick
-	-	S	E	-
-	-	K, D	-	-

Exemples

abort

```
  await immediate E;  
  emit C  
when S do emit K end;  
emit D
```

tick	tick	tick	tick	tick	...
-	-	S	E	-	...
-	-	K, D	-	-	...

Exemples

abort

```
  await immediate E;  
  emit C  
when S do emit K end;  
emit D
```

Exemples

abort

```
  await immediate E;  
  emit C  
when S do emit K end;  
emit D
```


Exemples

abort

```
  await immediate E;  
  emit C  
when S do emit K end;  
emit D
```

tick				
-				
-				

Exemples

abort

```
  await immediate E;  
  emit C  
when S do emit K end;  
emit D
```

tick	tick			
-	-			
-	-			

Exemples

abort

```
await immediate E;  
emit C  
when S do emit K end;  
emit D
```

tick	tick	tick		
-	-	E, S		
-	-	K, D		

Exemples

abort

```
await immediate E;  
emit C  
when S do emit K end;  
emit D
```

tick	tick	tick	tick	
-	-	E, S	-	
-	-	K, D	-	

Exemples

abort

```
  await immediate E;  
  emit C  
when S do emit K end;  
emit D
```

tick	tick	tick	tick	tick
-	-	E, S	-	-
-	-	K, D	-	-

Exemples

abort

```
  await immediate E;  
  emit C  
when S do emit K end;  
emit D
```

tick	tick	tick	tick	tick	...
-	-	E, S	-	-	...
-	-	K, D	-	-	...

Exemples

```
weak abort
  await immediate E;
  emit C
when S do emit K end;
emit D
```

Exemples

weak abort

await immediate E;

emit C

when S do emit K end;

emit D

Exemples

weak abort

```
await immediate E;  
emit C
```

```
when S do emit K end;  
emit D
```

tick				
-				
-				

Exemples

weak abort

```
await immediate E;  
emit C
```

```
when S do emit K end;  
emit D
```

tick	tick			
-	-			
-	-			

Exemples

weak abort

```
await immediate E;  
emit C  
when S do emit K end;  
emit D
```

tick	tick	tick		
-	-	S		
-	-	K, D		

Exemples

weak abort

```
await immediate E;  
emit C  
when S do emit K end;  
emit D
```

tick	tick	tick	tick	
-	-	S	E	
-	-	K, D	-	

Exemples

weak abort

```
await immediate E;  
emit C  
when S do emit K end;  
emit D
```

tick	tick	tick	tick	tick
-	-	S	E	-
-	-	K, D	-	-

Exemples

weak abort

```
await immediate E;  
emit C  
when S do emit K end;  
emit D
```

tick	tick	tick	tick	tick	...
-	-	S	E	-	...
-	-	K, D	-	-	...

Exemples

```
weak abort
  await immediate E;
  emit C
when S do emit K end;
emit D
```

Exemples

weak abort

await immediate E;

emit C

when S do emit K end;

emit D

Exemples

```
weak abort
  await immediate E;
  emit C
when S do emit K end;
emit D
```

tick				
-				
-				

Exemples

```
weak abort
  await immediate E;
  emit C
when S do emit K end;
emit D
```

tick	tick			
-	-			
-	-			

Exemples

weak abort

```
await immediate E;  
emit C
```

```
when S do emit K end;  
emit D
```

tick	tick	tick		
-	-	E, S		
-	-	C, D		

Exemples

weak abort

```
await immediate E;  
emit C  
when S do emit K end;  
emit D
```

tick	tick	tick	tick	
-	-	E, S	-	
-	-	C, D	-	

Exemples

weak abort

```
await immediate E;  
emit C  
when S do emit K end;  
emit D
```

tick	tick	tick	tick	tick
-	-	E, S	-	-
-	-	C, D	-	-

Exemples

weak abort

```
await immediate E;  
emit C  
when S do emit K end;  
emit D
```

tick	tick	tick	tick	tick	...
-	-	E, S	-	-	...
-	-	C, D	-	-	...

Exemple

- Imbrication

```
[  
    abort  
        abort  
            halt  
            when A do  
                emit D  
            end abort  
            when B do  
                emit E  
            end abort  
    ]  
|| [  
    || [  
        emit A  
    || [  
        emit B  
    ]]
```

Exemple

- Imbrication

```
[  
    abort  
        abort  
            halt  
            when A do  
                emit D  
            end abort  
            when B do  
                emit E  
            end abort  
    ]  
|| [  
    || [  
        emit A  
    ||  
        emit B  
    ]  
||  
emit A  
||  
emit B  
||  
emit E
```

↔

Exemple

- Imbrication

```
[  
    abort  
        abort  
            halt  
            when A do  
                emit D  
            end abort  
            when B do  
                emit E  
            end abort  
    ]  
|| [  
    || [  
        emit A  
    ||  
        emit B  
    ]  
|| ?  
    ⇔|| emit A  
    || emit B  
    || emit E
```

Exemple

- Imbrication

```
[  
    abort  
        abort  
            halt  
            when immediate A do  
                emit D  
            end abort  
            when immediate B do  
                emit E  
            end abort  
    ]  
|| [  
    || [  
        emit A  
    ||  
        emit B  
    ]  
||  
emit A  
||  
emit B  
||  
emit E
```

↔

Horloge de base

- Le signal *tick* définit l’horloge de base du système et est présent à chaque instant
- **pause** : fin explicite d’un instant dans une séquence (équivalent à **await tick** : ancienne syntaxe)
- **halt** : suspension indéfinie de l’exécution (équivalent à **loop pause end loop**)
- **sustain S** : maintient la présence d’un signal en le ré-émettant à plusieurs instants consécutifs (équivalent à **loop emit S; pause end loop**)
- **await S** équivaut à **abort halt when S**

Suspension

- Suspendre l'exécution d'un programme si un signal est présent
 - correspond plus ou moins au ctrl-Z d'un shell UNIX
- Forme :

suspend

p

when S

- Si S est présent, p reste dans son état
- Si S est absent, p est exécuté pour l'instant courant

Exemple

```
module threadSynchrone:  
    input start, stop, susp, res;  
    await immediate start;  
    abort  
        signal dodo in [  
            loop  
                await susp;  
                abort  
                    sustain dodo  
                    when res  
            end loop  
        ]  
        suspend  
            run PROG()  
            when dodo  
        ] end signal  
    when stop;  
end module
```

Réinitialisation (boucles temporelles)

- Syntaxe : **loop p each d**
- d représente un délai (signal d'un temporisateur)
- Équivalent à **loop abort p; halt when d end loop**
- Autre forme : **every d do p end every**
- Équivalent à **await d; loop p each d**
- autre préemption : **do p upto s** équivalent à **abort p;halt when s**

Exceptions

- Syntaxe :

```
trap S in
    p
end trap
```

- La levée d'une exception : **exit S**
- p est exécuté jusqu'à terminaison ou à la levée d'une exception
- correspond à la préemption faible du **weak abort**

weak abort
p
when S



trap Exit in
p; **exit Exit**
||
 await S; exit Exit
end trap

Exceptions

- Syntaxe générale :

```
trap T1, ..., Tn in
    p
handle T1 do q1
...
handle Tn do qn
end trap
```

- Plusieurs handlers peuvent être sélectionnés par des exit en parallèle

Modules

- Syntaxe :

```
module nom :  
    % déclaration des paramètres (liste  
    % d'E/S), constantes...
```

```
    %corps du module  
end module
```

- Instanciation (copie module) :

```
run nom[param effectif/ param formel, ...]
```

- Exemple : run souris [button1 / click,
millisec / top]

Signaux locaux

- Syntaxe :

```
signal S in
  p
end signal
```

- Notation complète

```
signal S1, ... , Sn in
  p
end signal
```



```
signal S1 in
...
signal Sn in
  p
end signal
...
end signal
```

Combinaison de signaux

- Expressions booléennes sur l’absence ou la présence de signaux ; décision immédiate
- permet de tester la simultanéité (et logique), l’absence (non logique) ou l’exclusivité (ou exclusif)
- extension à la notion de délais (multiplicateurs entiers)
- Exemple :

await A and B

present A or not B then ...

await 5 second

present A and B then p end \Leftrightarrow

present A then present B then p end end

Variables

- Déclaration :

```
var x1:type1, ... , xn:typen in
    p
end var
```

- Affectation :

```
x:=1;
```

- Séquence et parallélisme :

- **x:=0 ; x:=x+1 => ok dans le même instant !**
- **x:=0 || x:=x+1 => interdit**
- test : **if c then p else q end if**

Signaux valués

- Déclaration :

```
signal S:combine type with op in
  p
end signal
```
- Valeur de S : `emit S(exp) , ?S, present S(x) then`
`p else q end`
- Valeur conservée entre les instants (même si le signal est absent)
- Les valeurs portées par les signaux peuvent venir de capteurs toujours disponibles

Problème de causalité

- Programmes incohérents (pas de solution)

```
emit S(?S + 1) %dépendance cyclique instantanée  
  
loop  
    x := 1  
end loop % boucle instantanée  
  
signal S in  
    present S else emit S end;  
end
```

La réaction instantanée à l'absence du signal S provoque son émission immédiate. Un signal est présent ou absent dans un instant

Problème de causalité

- Programmes non déterministes (plusieurs solutions)

```
emit S(?S) % une infinité de solutions pour la valeur
           % de S
```

```
signal S in
    present S then emit S end;
end signal
```

```
signal S1, S2 in
    present S1 then emit S2 end present
    ||
    present S2 then emit S1 end present
end signal
```

➡ Obstacle à la modularité ascendante

Problème de causalité

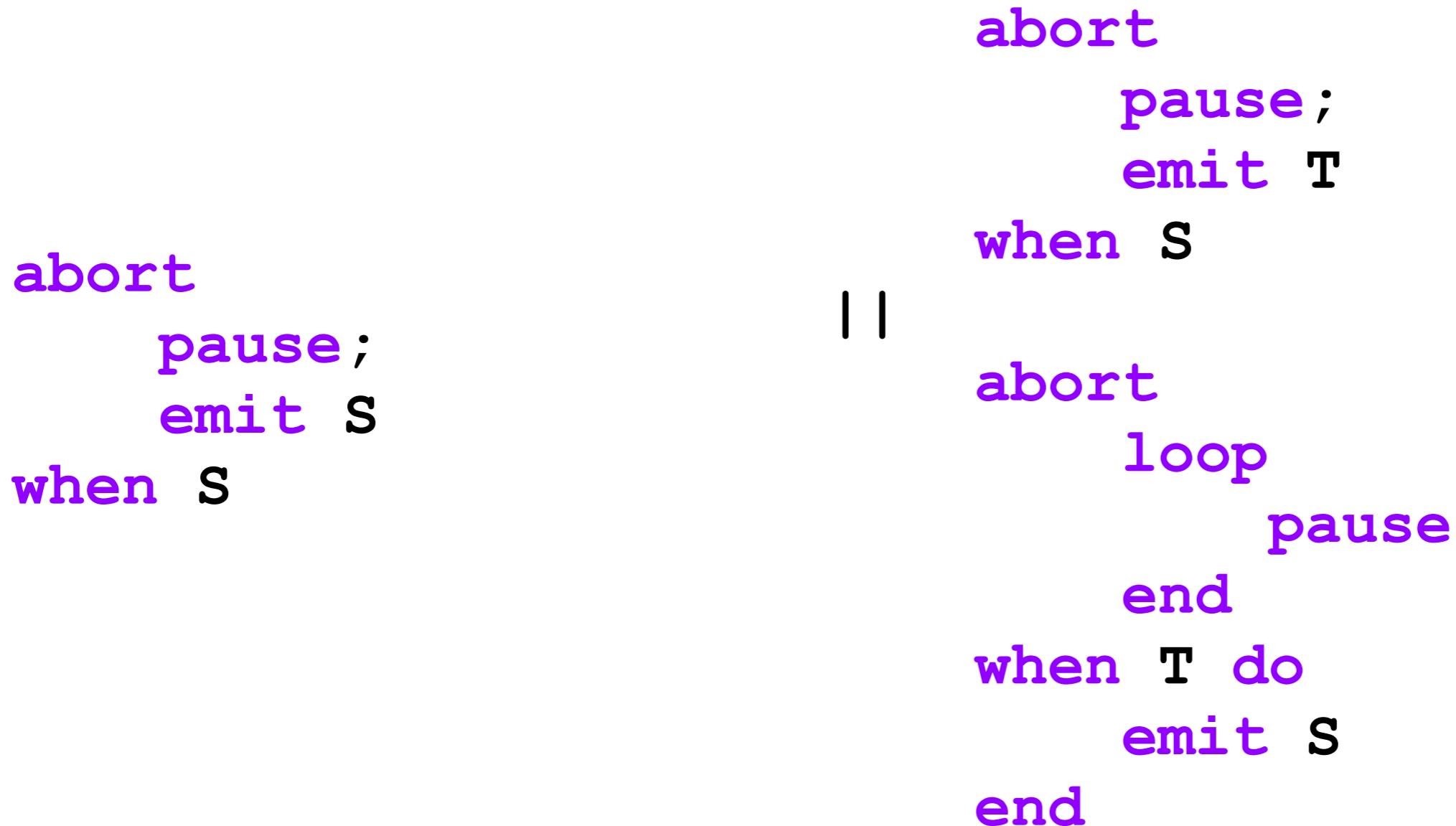
- Programmes non causaux : l’unicité de la solution n’est pas une condition suffisante

```
signal S in
    present S then emit T end
  ||
    emit S
end
```

comparé à

```
signal S in
    present S then emit T end;
    emit S
end
```

Causalité et préemption



- Pas de problème avec la préemption faible
- Problème := réaction instantanée à l'absence

Causalité et préemption

```
weak abort
pause;
emit T
when S
||

weak abort
loop
pause
end
when T do
emit S
end
```

- Pas de problème avec la préemption faible
- Problème := réaction instantanée à l'absence

Problème de causalité

- Trouver une solution => complexité en 2^N
- Comment éviter d'essayer toutes les solutions ?
- Travail du compilateur : approximation basée sur des heuristiques particulières et justifiables :
 - Émission potentielles (v3) : calcul d'un potentiel d'émission approximé.
 - Analyse des dépendances cycliques dans le graphe d'un programme (v4)
 - Causalité constructive (v5)

Différences d'analyse

```
signal S1, S2 in
```

```
    emit S1;
```

rejeté par la v3

```
    present S2 then
```

```
        present S1 else emit S2 end
```

```
    end
```

```
end signal
```

```
signal S1, S2 in
```

```
    present S1 then emit S2 end;
```

```
    pause;
```

```
    present S2 then emit S1 end
```

```
end signal
```

rejeté par la v4

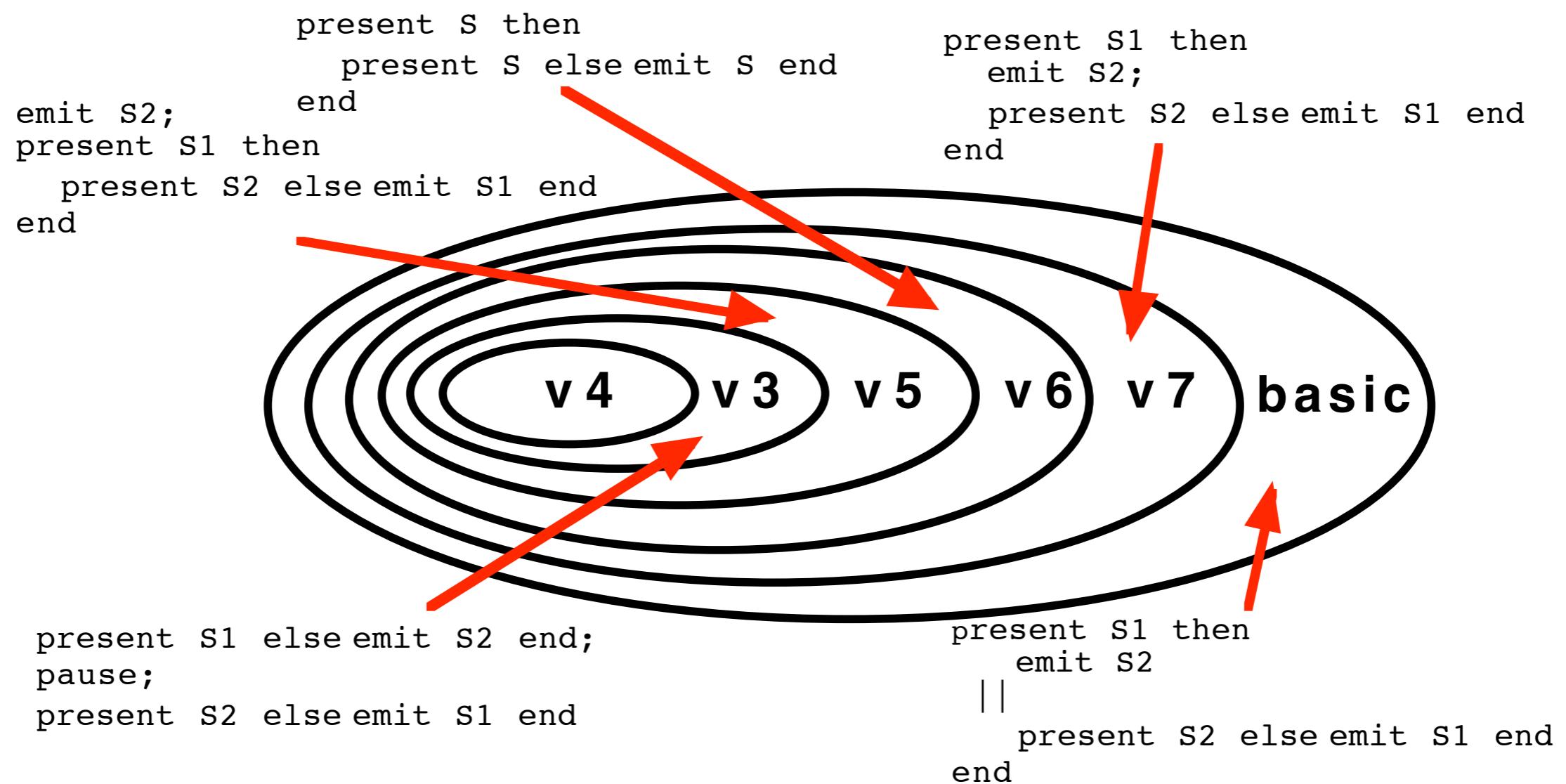
```
signal S in
```

```
    present S else emit S end;
```

```
end signal
```

rejeté par la v5

Fonction potentiel d'émission





Master
systèmes embarqués et mobiles sûrs



UE : SMB204
Systèmes Embarqués et Enfouis

Sémantique d'Esterel

d'après les supports de F. Boussinot

le cnam

Paris, 30 oct. 2012

Règles SOS

- Règles de réécriture exprimées dans un formalisme “à la” Plotkin
 - t instruction au début de l'instant
 - t' instruction à la fin de l'instant
 - P signaux présents dans l'événement courant
 - E signaux émis (remarque : $E \subseteq P$)
 - b booléen de terminaison (vrai = terminé)

Règles conditionnelles :

$$\frac{\text{prédicats}}{t, P \xrightarrow{b} t', E}$$

Instructions de base

- nothing

$$\text{nothing}, P \xrightarrow{\textit{true}} \text{nothing}, \{\}$$

- pause

$$\text{pause}, P \xrightarrow{\textit{false}} \text{nothing}, \{\}$$

- émission

$$\text{emit } S, P \xrightarrow{\textit{true}} \text{nothing}, \{S\}$$

Séquence binaire

- Le terme de gauche ne se termine pas

$$\frac{t, P \xrightarrow{\textit{false}} t', E}{t ; u, P \xrightarrow{\textit{false}} t' ; u, E}$$

- Le terme de gauche se termine

$$\frac{t, P \xrightarrow{\textit{true}} t', E \quad u, P \xrightarrow{\alpha} u', E'}{t ; u, P \xrightarrow{\alpha} u', E \cup E'}$$

Parallélisme binaire

- terminaison globale lorsque les 2 branches terminent

$$\frac{t, P \xrightarrow{b_1} t', E \quad u, P \xrightarrow{b_2} u', E'}{t \parallel u, P \xrightarrow{b_1 \wedge b_2} t' \parallel u', E \cup E'}$$

Boucles

- Boucle infinie

$$\frac{t, P \xrightarrow{\text{false}} t', E}{\text{loop } t \text{ end}, P \xrightarrow{\text{false}} t' ; \text{ loop } t \text{ end}, E}$$

→ boucle instantanée => pas de réécriture

- Boucle finie

$$\frac{n > 0 \quad t, P \xrightarrow{\text{true}} t', E \quad \text{repeat } n - 1 \text{ times } t \text{ end}, P \xrightarrow{b} u, E'}{\text{repeat } n \text{ times } t \text{ end}, P \xrightarrow{b} u, E \cup E'}$$

$$\frac{n > 0 \quad t, P \xrightarrow{\text{false}} t', E}{\text{repeat } n \text{ times } t \text{ end}, P \xrightarrow{b} t' ; \text{ repeat } n - 1 \text{ times } t \text{ end}, E}$$

$$\frac{\text{repeat } n \text{ times } t \text{ end}, P \xrightarrow{b} t' ; \text{ repeat } n - 1 \text{ times } t \text{ end}, E \quad \text{repeat } 0 \text{ times } t \text{ end}, P \xrightarrow{\text{true}} \text{nothing}, \{\}}{\text{repeat } 0 \text{ times } t \text{ end}, P \xrightarrow{\text{true}} \text{nothing}, \{\}}$$

Signaux locaux

- Hypothèse de présence

$$\frac{t, P + S \xrightarrow{b} t', E \quad S \in E}{\text{signal } S \text{ in } t \text{ end, } P \xrightarrow{b} \text{signal } S \text{ in } t' \text{ end, } E'} \\ E' = E - \{S\}$$

- Hypothèse d'absence

$$\frac{t, P - \{S\} \xrightarrow{b} t', E \quad S \notin E}{\text{signal } S \text{ in } t \text{ end, } P \xrightarrow{b} \text{signal } S \text{ in } t' \text{ end, } E}$$

E' semblable à E sauf pour S qui ne doit pas être produit
portée locale

Présence/Absence

- présence

$$S \in P \quad t, P \xrightarrow{b} t', E$$

present S then t else u end, $P \xrightarrow{b} t', E$

- absence

$$S \notin P \quad u, P \xrightarrow{b} u', E$$

present S then t else u end, $P \xrightarrow{b} u', E$

Préemption

- Premier instant

$$t, P \xrightarrow{b} t', E$$

abort t when $S, P \xrightarrow{b}$ abort t' when immediate S, E

- Instants suivants

present S else abort t when S end, $P \xrightarrow{b} u', E$

abort t when immediate $S, P \xrightarrow{b} u', E$

Arbre de preuve

- programme :

```
signal S in
    present S then emit T end present
    ||
        emit S
end signal
```

$$\frac{\text{emit } T, \{S, T\} \xrightarrow{vrai} \text{nothing}, \{T\} \quad \text{emit } S, \{S, T\} \xrightarrow{vrai} \text{nothing}, \{S\}}{\text{present } S \text{ then emit } T \text{ end}, \{S, T\} \xrightarrow{vrai} \text{nothing}, \{T\}} \quad \frac{}{\text{present } S \text{ then emit } T \text{ end} \parallel \text{emit } S, \{S, T\} \xrightarrow{vrai} \text{nothing}, \{S, T\}}$$

$$\text{signal } S \text{ in present } S \text{ then emit } T \text{ end} \parallel \text{emit } S \text{ end, } \{T\} \xrightarrow{vrai} \text{nothing, } \{T\}$$

Non déterminisme

- programme :

```
signal S in
    present S then emit S; emit T end
end signal
```

$$\frac{\begin{array}{c} \text{emit } T, \{T, S\} \xrightarrow{\text{vrai}} \text{nothing}, \{T\} \quad \text{emit } S, \{S, T\} \xrightarrow{\text{vrai}} \text{nothing}, \{S\} \\ \hline \text{present } S \text{ then emit } S; \text{emit } T \text{ end}, \{T, S\} \xrightarrow{\text{vrai}} \text{nothing}, \{S, T\} \end{array}}{\text{signal } S \text{ in present } S \text{ then emit } S; \text{emit } T \text{ end end}, \{T\} \xrightarrow{\text{vrai}} \text{nothing}, \{T\}}$$

$$\frac{\begin{array}{c} \text{nothing}, \{ \} \xrightarrow{\text{vrai}} \text{nothing}, \{ \} \\ \hline \text{present } S \text{ then emit } S; \text{emit } T \text{ end}, \{ \} \xrightarrow{\text{vrai}} \text{nothing}, \{ \} \end{array}}{\text{signal } S \text{ in present } S \text{ then emit } S; \text{emit } T \text{ end end}, \{ \} \xrightarrow{\text{vrai}} \text{nothing}, \{ \}}$$

Absence de solution

- programme :

```
signal S in
    present S else emit S; emit T end
end
```

$$\frac{\begin{array}{c} \text{emit } T, \{T\} \xrightarrow{\text{vrai}} \text{nothing}, \{T\} \quad \text{emit } S, \{T\} \xrightarrow{\text{vrai}} \text{nothing}, \{S\} \\ \hline \text{present } S \text{ else emit } S; \text{emit } T \text{ end}, \{T\} \xrightarrow{?} ? \end{array}}{\text{signal } S \text{ in present } S \text{ else emit } S; \text{emit } T \text{ end end}, \{} \xrightarrow{?} ?$$

$$\frac{\begin{array}{c} \text{nothing}, \{S\} \xrightarrow{\text{vrai}} \text{nothing}, \{} \\ \hline \text{present } S \text{ else emit } S; \text{emit } T \text{ end}, \{S\} \xrightarrow{\text{vrai}} \text{nothing}, \{} \\ \hline \text{signal } S \text{ in present } S \text{ else emit } S; \text{emit } T \text{ end end}, \{} \xrightarrow{\text{vrai}} ? \end{array}}$$

Solution non causale

- programme :

```
signal S in
    present S then emit T end; emit S
end
```

$$\frac{\begin{array}{c} \text{emit } T, \{S, T\} \xrightarrow{\text{vrai}} \text{nothing}, \{T\} \\ \hline \text{present } S \text{ then emit } T \text{ end}, \{S, T\} \xrightarrow{\text{vrai}} \text{nothing}, \{T\} \end{array}}{\text{present } S \text{ then emit } T \text{ end}; \text{emit } S, \{S, T\} \xrightarrow{\text{vrai}} \text{nothing}, \{S, T\}}$$
$$\frac{\text{present } S \text{ then emit } T \text{ end}; \text{emit } S, \{S, T\} \xrightarrow{\text{vrai}} \text{nothing}, \{S, T\}}{\text{signal } S \text{ in present } S \text{ then emit } T \text{ end}; \text{emit } S \text{ end}, \{} \xrightarrow{\text{vrai}} \text{nothing}, \{T\}$$

Le programme est accepté alors que la solution est non causale

Conclusion

- Sémantique opérationnelle “comportementale”
définition macro étape : une réécriture = un instant
 - simple : structurelle => pas de surcharge de l’env
 - n’exprime pas comment trouver la solution
 - accepte des programmes non causaux ou non déterministes (vérification d’unicité très coûteuse)
→ implantation directe inefficace
- Privilégier les solutions où le signal est absent pour éliminer les solutions non causales
- sémantique basée sur la notion de calcul de potentiel (classe des programmes ?)

Évolution d'Esterel

- v3 : Sémantique comportementale avec utilisation du calcul des potentiels : *la sémantique exécutoire*
 - Sur un test de signal, on explore l'impact d'une hypothèse (calcul de potentiel d'émission). On privilégie l'hypothèse d'absence et on montre que pour tout événement d'entrée, le signal testé ne peut plus être émis
 - ➡ Ne retient que des programmes cohérents et déterministes de la sémantique comportementale

Évolution d'Esterel

- Exemple :

```
signal S, T in
    present S else emit T; emit U end
    ||
    present T else emit S; emit V end
end signal
```

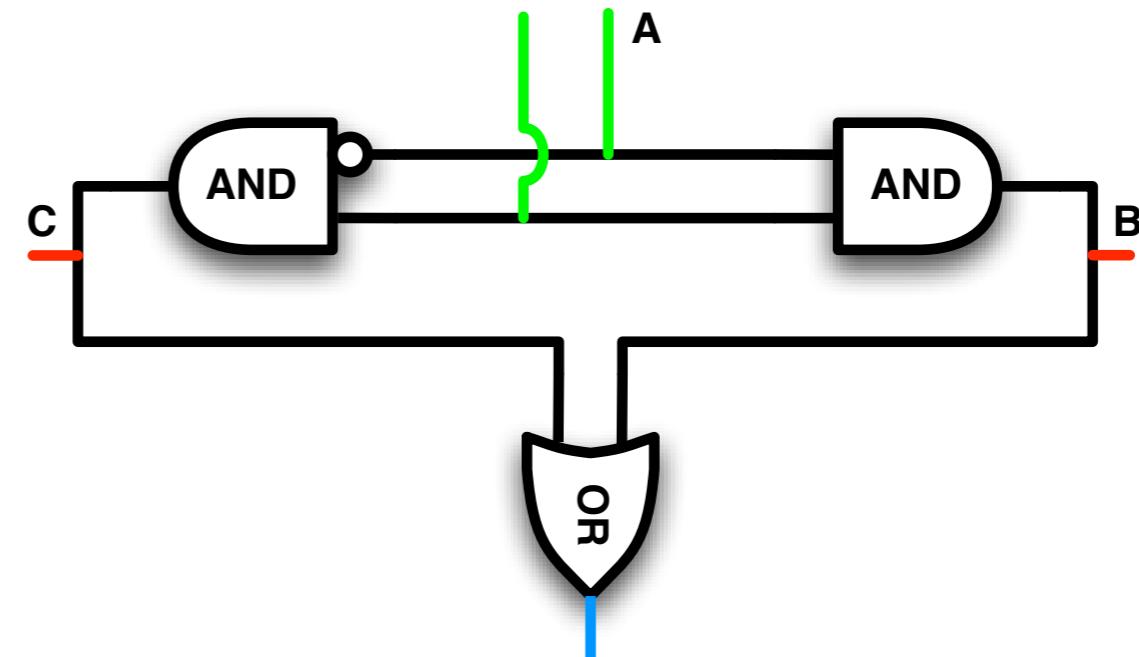
- Analyse : bloque sur les instructions **present** en parallèle
 - ➡ T, S sont des signaux potentiellement émis par les branches **else** => rejet du programme

Évolution d'Esterel

- si un programme est accepté => génération d'un automate explicite
- v4 : sémantique des circuits séquentiel acycliques ;
notion de *cycles de causalité*
 - Un signal S_1 dépend causalement d'un signal S_2 si l'émission de S_1 dépend d'un test de S_2
 - De même, un signal S_1 dépend causalement d'un signal S_2 si un test sur le signal S_1 dépend de l'émission de S_2
 - On établit des graphes de dépendances entre les signaux et on rejette les graphes cycliques

Évolution d'Esterel

```
present A then
    emit B
else
    emit C
end ;
```



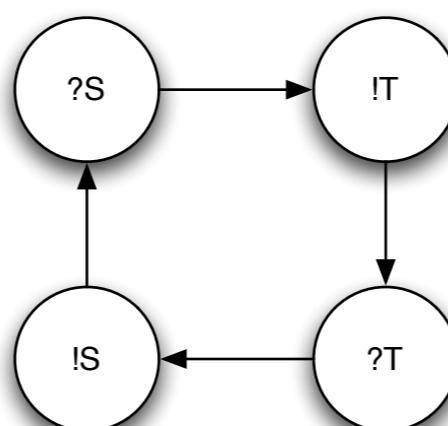
- Circuit séquentiel acyclique logique : peut être décrit par un ensemble d'équations booléennes sans cycle de dépendance => on peut trouver un ordre où toute variable ne dépend que des variables précédemment définies

Évolution d'Esterel

- Programmes sélectionnés cohérents et déterministes
- Exemple :

```
signal S, T in
    present S else emit T end
    ||
    present T else emit S end
end signal
```

- Analyse : on a une dépendance cyclique entre S et T



Évolution d'Esterel

- Rejette beaucoup trop de programmes
- Exemple :

```
signal S, T, U in
    present U then
        present S then emit T end
    else
        present T then emit S end
    end
end signal

signal S in
    emit S; present S else emit S end
end signal
```

- Plus restrictive que la v3 : ne tient pas compte des instants

Évolution d'Esterel

- v5 : la sémantique constructive (toujours définie par rapport aux circuits)
 - utilisation de deux fonctions qui sont construites sur la connaissance de l'environnement
 - *cannot* : calcul les signaux ne pouvant pas être émis
 - *must* : calcul les signaux nécessairement émis
 - L'environnement des signaux produits E est construit par un calcul de point fixe utilisant *cannot* et *must*
 - Si tous les signaux sont déterminés, le programme est constructif et la réécriture de ce dernier est calculée en utilisant la sémantique comportemental avec P

Évolution d'Esterel

```
module P2:  
    output O;  
    signal S in  
        emit S;  
        present O then  
            present S then  
                pause  
            end;  
            emit O  
        end  
    end signal  
end module
```

- Départ : S(\perp) et O(\perp)
- On détermine d'abord ce qu'on doit faire : emit S \Rightarrow S(+)
- Sur **present** on explore ce qu'on ne peut pas faire

Évolution d'Esterel

- Similaire à une fonction de potentiel qui explore le programme au niveau d'un test en tenant compte des événements présents avant le test, mais ne collecte pas d'information dans l'environnement au cours d'une exploration

```
present s then
    present s else emit s end
end present
```

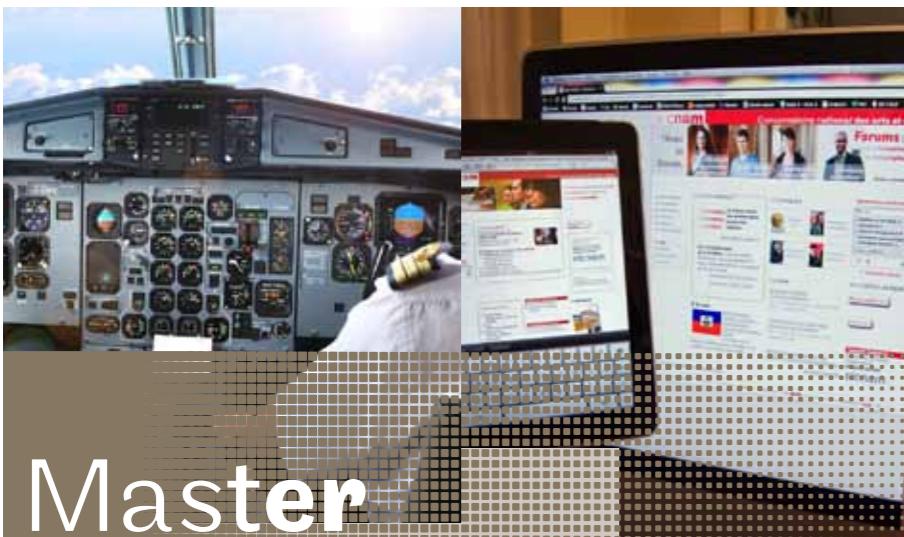
Notion de temps multiforme

- Objectif des langages synchrones : Produire des comportements déterministes (fonctionnellement et temporellement)
- Solution : Le temps physique est vu à travers une notion d'ordre grâce à laquelle on définit un ordre des événements et la simultanéité ou l'absence de signaux
- Conséquence : Contrairement aux approches temps réel classique, le temps physique n'a plus un rôle particulier (c'est un ensemble de signaux de l'environnement).

On parle de temps multiforme

Notion de temps multiforme

- Exemple : pour un train se déplaçant à une certaine vitesse, le contraintes suivantes :
 - Le train doit s’arrêter dans 10 secondes
 - Le train doit s’arrêter dans 100 mètres
- sont vues de la même manière contrairement aux langages qui traitent le temps physique de façon particulière :
 - Le train doit s’arrêter dans 10 (resp. 100) occurrences du signal seconde (resp. mètre)



Master
systèmes embarqués et mobiles sûrs



UE : SMB204
Systèmes Embarqués et Enfouis

Prélude à la génération de code

d'après les supports de G. Berry

le cnam

Paris, 30 oct. 2012

Chimie, Newton et Vibration

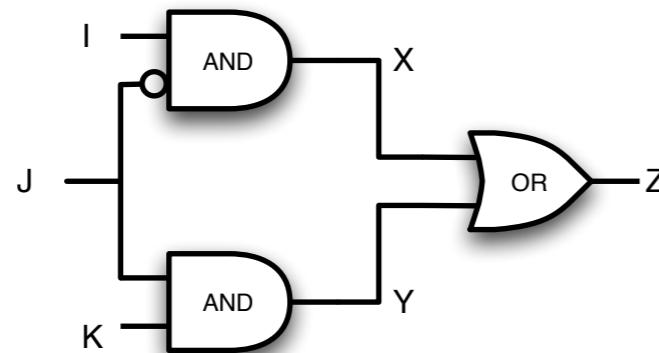
- Modèle de la concurrence en Esterel : G.Berry a proposé 3 modèles de la concurrence basé sur des analogies avec des phénomènes physiques élémentaires :
 - La machine chimique : calcul de processus ou les réductions opèrent lorsque 2 programmes compatibles communiquent (réaction entre 2 molécules) => destruction et production de nouvelles entités.

Chimie, Newton et Vibration

- Modèle de composition des systèmes
 - Modèle newtonien : analogie avec la mécanique classique ; notion de temps absolu (le même pour tous), communication instantanée.
 - Modèle “vibrationnel” : analogie avec la propagation des ondes ; communication effectué en un temps déterminé.
- Ces deux derniers modèles sont à la base des langages synchrones : la sémantique utilise le modèle newtonien, l’implantation un modèle vibrationnel (introduction d’un délai d : temps de stabilisation du système).

Chimie, Newton et Vibration

- Un circuit acyclique combinatoire

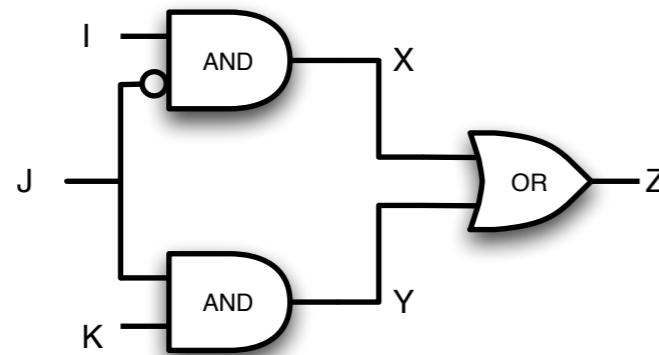


$$\begin{aligned}X &= I \text{ and not } J \\Y &= J \text{ and } K \\Z &= X \text{ or } Y\end{aligned}$$

- Point de vue Newtonien (zéro délai) : circuit vue comme un ensemble d'équation booléennes, acyclique (ordre) ; les sorties sont définies par combinaison des entrées booléennes
- Point de vue du design logique : on se concentre sur les propriétés booléennes du circuit et non sur le comment sont produites les représentations booléennes

Chimie, Newton et Vibration

- Un circuit acyclique combinatoire



$$\begin{aligned}X &= I \text{ and not } J \\Y &= J \text{ and } K \\Z &= X \text{ or } Y\end{aligned}$$

- Point de vue vibrationnel : circuit vu comme un ensemble de portes et de fils avec un délai de stabilisation/propagation fixe (borné) en fonction des tensions d'entrée.
- Point de vue de l'électronique : on se concentre sur le délai de stabilisation (à minimiser) et non sur ce que réalise le circuit



Master
systèmes embarqués et mobiles sûrs



UE : SMB204
Systèmes Embarqués et Enfouis

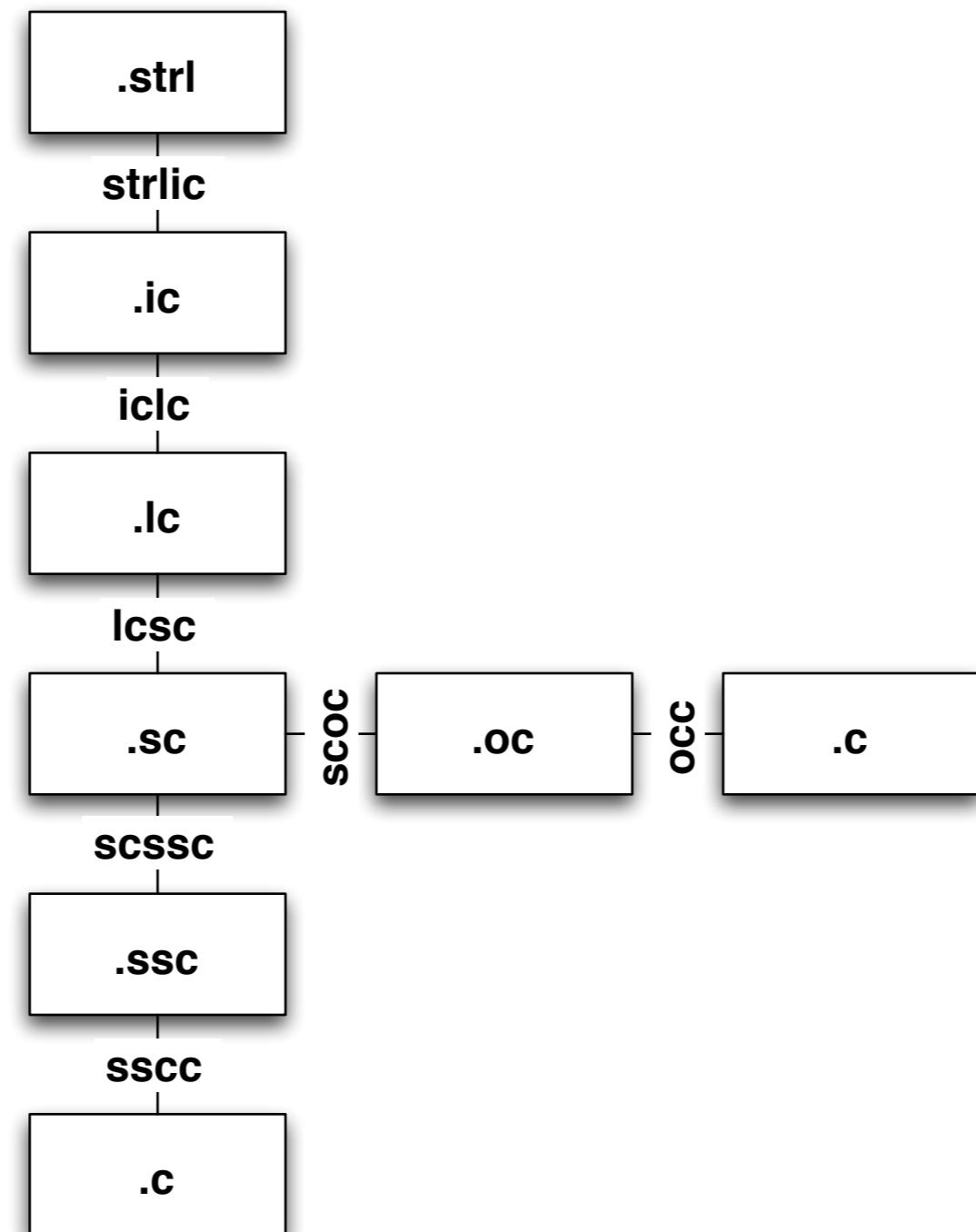
Compilation

considérations sur la version académique

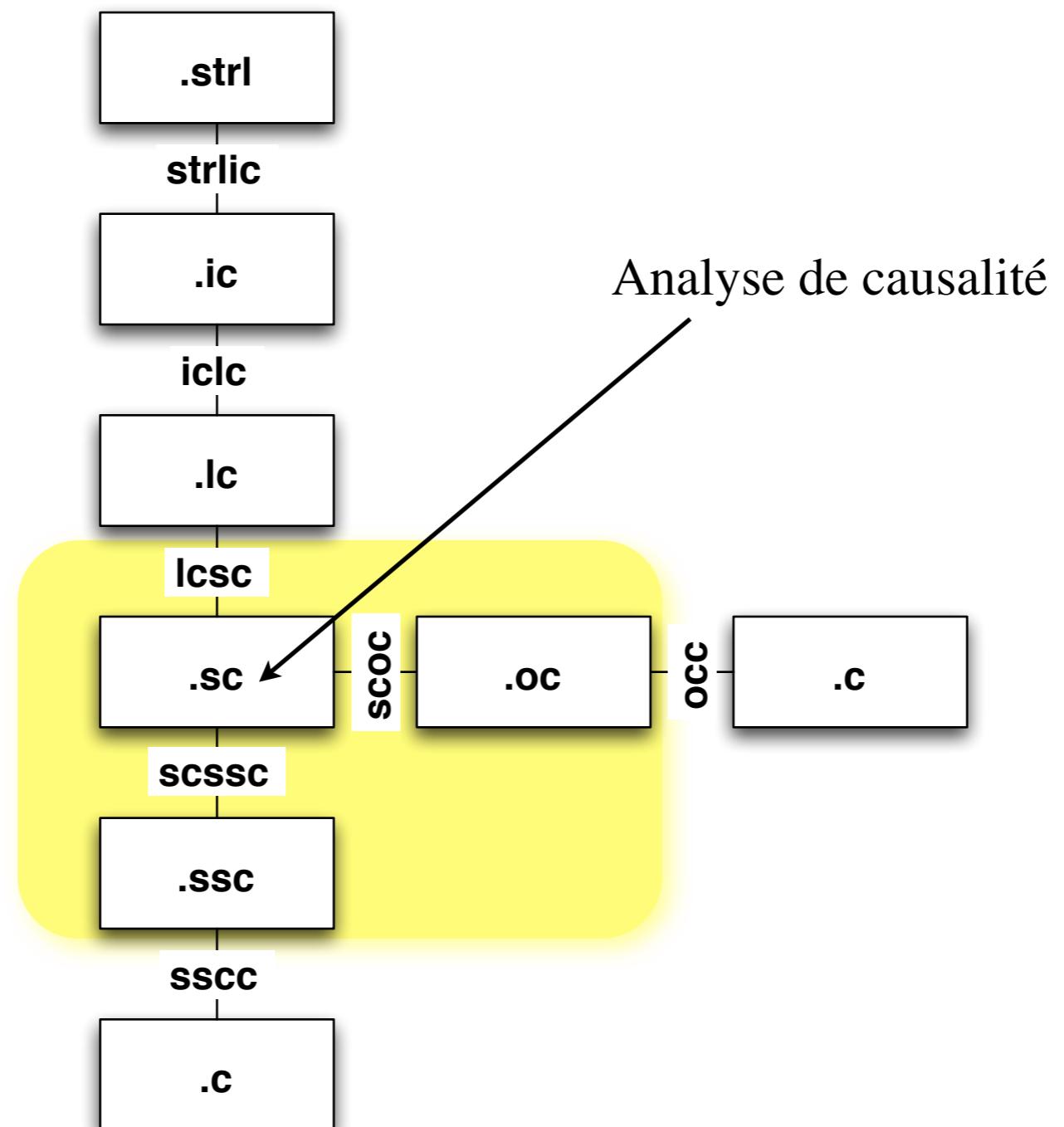
le cnam

Paris, 30 oct. 2012

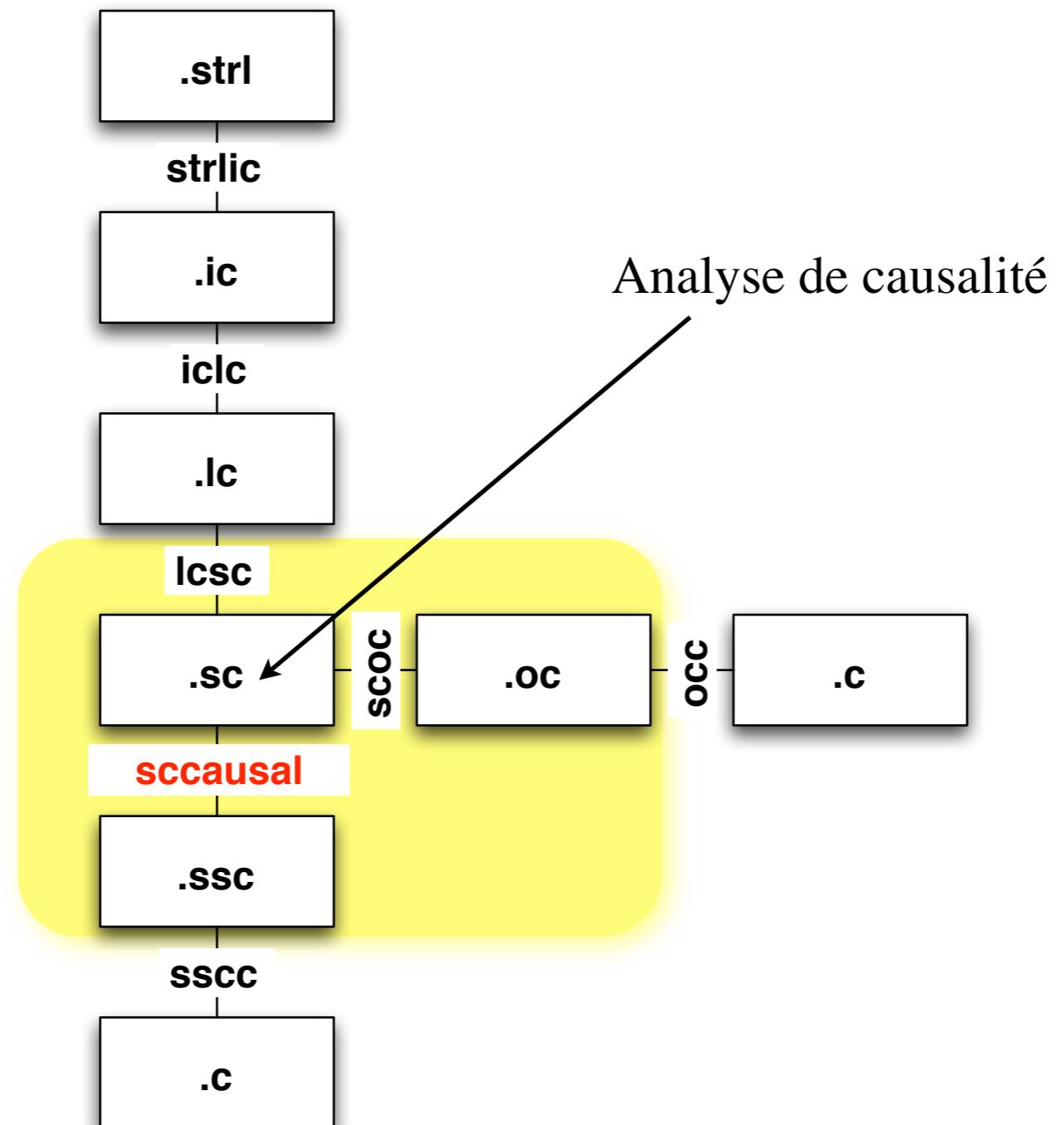
Le compilateur



Le compilateur



Le compilateur



La compilation d'un module Esterel

- On part d'un module EX défini dans un fichier EX.strl
- La chaîne de compilation standard (v5_92) produit un code C destiné à être utilisé dans un exécutif (machine d'exécution fournissant les signaux d'entrée et les actions en sortie de réaction) :
 - EX_reset() : (ré-)initialisation du module (comp)
 - EX_I_SIG1() : émission de l'entrée SIG1 (comp)
 - EX() : réaction du module (comp)
 - EX_O_SIG2() : action sur un signal de sortie (util)

Machine d'exécution

- Implantation typique : une boucle qui à chaque itération produit une réaction
 - (ré-)initialisation
 - détermination des entrées (liens avec l'environnement physique : capteurs, timer, ...)
 - réaction : Appel de EX
 - production des sorties
 - test terminaison

Autres éléments d'interfaces

- Esterel s'appuie sur un langage hôte : généralement C
 - Les types :
 - primitifs : boolean, integer, float, double, string
 - utilisateurs : on ne déclare que leurs noms :
`type Toto[, Autres_types];`
ils sont définis dans le langage hôte et sont traités de façon opaque par Esterel (opérateurs de comparaison simple et affectation := et =, ◊)
 - les constantes : noms dont la valeur est définie dans le langage hôte
`constant Toto = val : type[, Autre_const];`

Autres éléments d'interfaces

- Les fonctions :
 - définies dans le langage hôte
 - pas d'effet de bord (non vérifiable par le compilateur)
 - une liste de paramètres (non altérés)
 - une valeur de retour
 - appel dans les expressions sur les données

```
function Toto([par1[, par2...]]) : type_retour;
```

Autres éléments d'interfaces

- Les tâches
 - définies dans le langage hôte
 - deux listes de paramètres (passage par référence et passage par valeur)
 - exécution non atomique (appel avec `exec Toto ()()` `return FIN`) concurrente avec le programme esterel (prise en charge par la machine d'exécution)
`task Toto([ref1[, ref2...]]) ([val1[, val2...]]);`

Autres éléments d'interfaces

- Les procédures
 - définies dans le langage hôte
 - pas d'effet de bord (hormis appel par référence)
 - deux listes de paramètres (passage par référence et passage par valeur)
 - exécution atomique (appel avec **call**)

```
procédure Toto([ref1[, ref2...]]) ([val1[,  
val2...]]);
```

Analyse de causalité

- Le compilateur doit vérifier qu'il n'y a pas de dépendance instantanée entre les signaux
 - Évaluation tri-valuées des signaux : présent, absent, inconnu
- Utilisant la sémantique constructive (cannot, must). Première passe est une approximation. Erreur => analyse constructive complète (risque de non convergence)

Compilation en automates

- Le compilateur évalue l'évolution des instructions de contrôle en fonction des environnements de signaux possibles
 - Les instructions : if, present, trap, exit, await, ...
 - L'ensemble des événements courants : occurrence des signaux d'entrée et locaux, exécution des instructions exit, ...
- L'ensemble des instructions de contrôle est fini
→ automate d'états fini déterministe

Automates à états finis déterministes

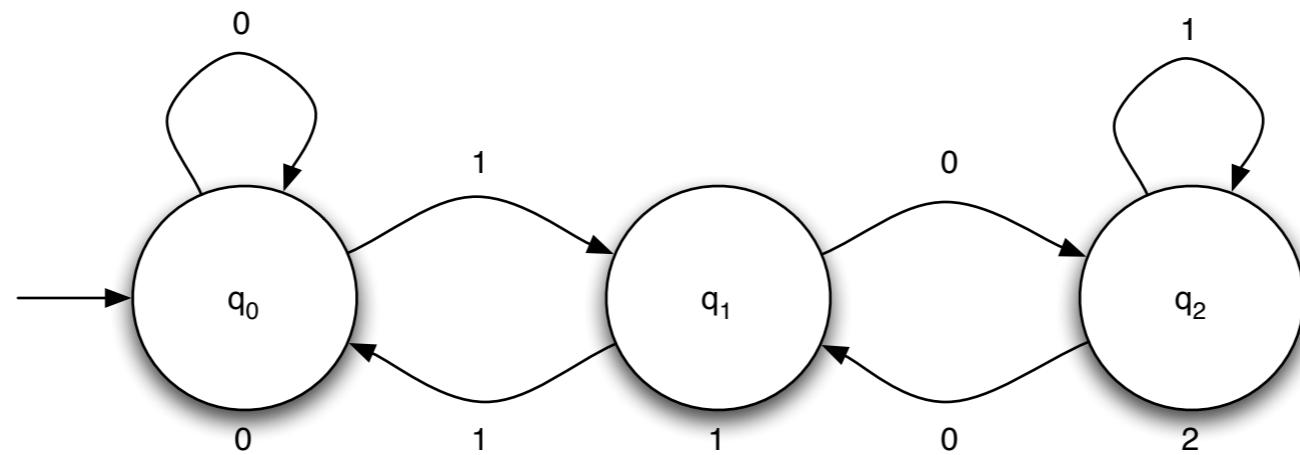
- Deux modèles d’automates (explicites) :
 - Machine de Moore : la sortie est associée à l’état
 - Machine de Mealy : la sortie est associée à la transition

Automates de Moore

- Définit par : $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$
 - Q ensemble fini d'états et q_0 est l'état initial
 - Σ alphabet des entrées
 - Δ alphabet des sorties
 - δ est une application de $Q \times \Sigma$ dans Q
 - λ est une application de Q dans Δ , donnant la sortie associée à chaque état
- La sortie en réponse à une série d'entrée (a_0, \dots, a_n) $n \geq 0$ est $\lambda(q_0) \lambda(q_1) \dots \lambda(q_n)$ ou $q_0 \dots q_n$ est la séquence d'états telle que $\delta(q_{i-1}, a_i) = q_i$ pour $1 \leq i \leq n$

Exemple

- Compter modulo 3 à partir d'un nombre écrit en binaire



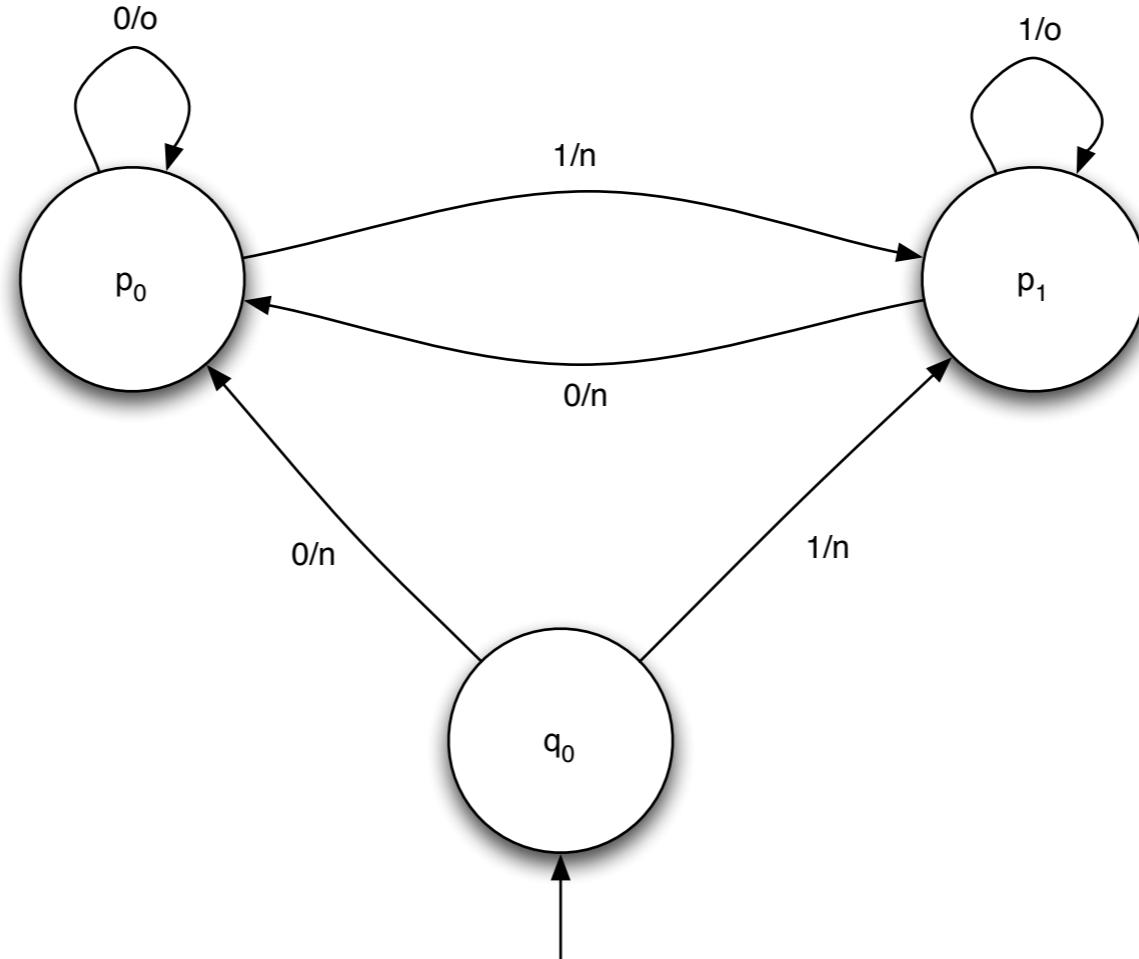
- Sur l'entrée 1010 la séquence d'état est q_0, q_1, q_2, q_2, q_1 et la sortie associée est 01221
 - $\epsilon \rightarrow 0 ; 1 \rightarrow 1 ; 10 \rightarrow 2 ; 101 \rightarrow 2 ; 1010 \rightarrow 1$

Automates de Mealy

- Définit par : $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$
 - Q ensemble fini d'états et q_0 est l'état initial
 - Σ alphabet des entrées
 - Δ alphabet des sorties
 - δ est une application de $Q \times \Sigma$ dans Q
 - λ est une application de $Q \times \Sigma$ dans Δ
- $\lambda(q, a)$ donne la sortie associée à une transition d'un état q sur l'entrée a . La sortie en réponse à une série d'entrées (a_0, \dots, a_n) $n \geq 0$ est $\lambda(q_{n-1}, a_n)$ et $\delta(q_{i-1}, a_i) = q_i$ pour $1 \leq i \leq n$

Exemple

- reconnaissance de mots binaires se terminant par 00 ou 11.



- Un automate de Mealy répond o (pour oui), n pour non
 - Ex : 01100 donne nnono

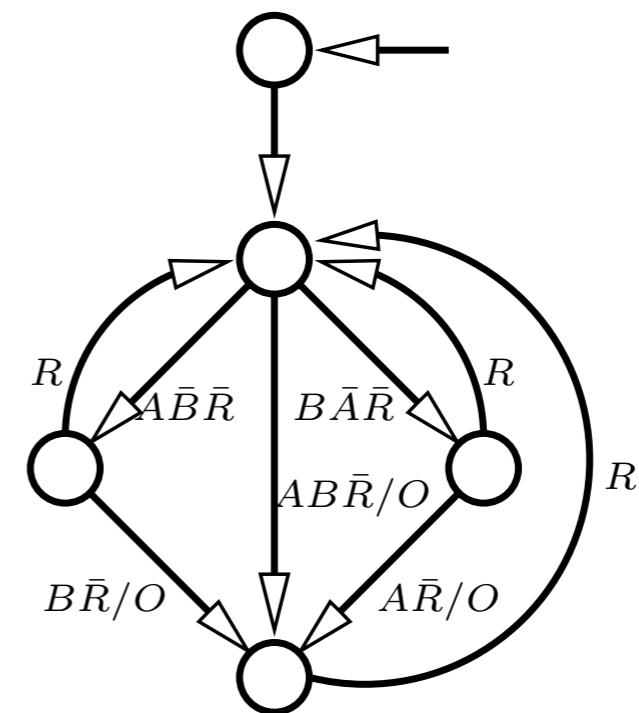
Exemple ABRO

- On émet O dès que A et B sont reçus et on recommence dès que R est reçu

```
module ABRO
  input A,B,R;
  output O;

loop
  [await A || await B];
  emit O;
each R

end module
```



- Compilation en automate explicite (format OC)

Le format OC

- C'est le format de description des automates
- Module OC = automate d'états fini déterministe étendu avec une table d'actions
 - ➡ Afin de manipuler les objets non discrets : variables entières et réelles ...
- Chaque état contient un graphe d'actions orienté acyclique (DAG)
 - Les nœuds unaires sont des actions
 - Les nœuds binaires sont des tests d'input ou de variable
 - Les feuilles sont des gotos
- Tous les objets internes sont tabulés pour une représentation compacte et efficace : types, variables, signaux, états,...

Exemple OC

```
module: ABRO
types: 0
end:

constants: 0
end:

functions: 0
end:

procedures: 0
end:

signals: 4
0: input: A 1 pure: bool: 0 %name: A% %previous: first:% %lc: 3 7 0%
1: input: B 3 pure: bool: 1 %name: B% %previous: 0% %lc: 3 10 0%
2: input: R 5 pure: bool: 2 %name: R% %previous: 1% %lc: 3 13 0%
3: output: O 6 pure: %name: O% %previous: 2% %lc: 5 8 0%
end:
```

Exemple OC

```
variables: 3
0: $0 %sigbool: 0% %lc: 3 7 0%
1: $0 %sigbool: 1% %lc: 3 10 0%
2: $0 %sigbool: 2% %lc: 3 13 0%
end:

actions: 7
0: call: $0 (0) (@$0)
1: present: 0 %lc: 3 7 0%
2: call: $0 (1) (@$0)
3: present: 1 %lc: 3 10 0%
4: call: $0 (2) (@$0)
5: present: 2 %lc: 3 13 0%
6: output: 3 %lc: 5 8 0%
end:

states: 6
startpoint: 1
sink: 0
calls: 27
```

Exemple OC

```
0: <0>

1: <2>

2: 5 (<2>) () 3 (1 (6 <3>) () <4>) () 1 (<5>) () <2>
%awaited: 0 1 2%

3: 5(<2>) () <3>
%awaited: 2%

4: 5 (<2>) () 1 (6 <3>) () <4>
%awaited: 0 2%

5: 5 (<2>) () 3 (6 <3>) () <5>
%awaited: 1 2%

end:

endmodule:
```

Automate implicite

- Un automate peut-être interprété de façon implicite par un ensemble d'équations booléennes (efficacement représenté par des BDDs)
- Ce qui revient à considérer un circuit constitué de portes logiques (partie combinatoire) et de registres (états du système) ; les registres codent les états
- Automate à N états = circuit séquentiel à $\log_2(N)$ registres (évite l'explosion combinatoire)
- Le compilateur génère des portes logiques et des fils pour les instructions ; seul le pause génère un registre

Le format SC

- format de description des circuits séquentiels (net list)
- Module SC = circuit séquentiel étendu avec une table d'actions
- La partie combinatoire contient des fils spéciaux qui, quand le front montant arrive, déclenchent une action de la table
- Là encore tout est tabulé et les tables sont communes avec les autres formats

Compilation hybride

- OC : très efficace mais très gros
- SC : très compacte mais exécution lente (sauf en circuit)
- Identification des points d'arrêt du programme
- Ordonnancement de ces points d'arrêt
- Génération de C efficace et compact
- utilisation d'un graphe de flot de contrôle (GFC,
compilateur SAXO, CEC, ...)

Vérification du synchronisme

- Connaissance des fréquences maximales des entrées et des contraintes temporelles
- L'automate fini (explicite ou implicite) permet de connaître exactement le temps de réaction du programme
- À comparer avec les contraintes temporelles
- Vérification de la mise en œuvre d'un module avec la notion de relations :
 - incompatibilité : **relation s1 # s2**
 - implication : **relation s1 => s2**

L'Approche Réactive/Synchrone

- Références:
 - *The Esterel Synchronous Language: Design, Semantics, Implementation*, G. Berry, G. Gonthier, Science of Computer Programming, 19(2), 1992
 - *The Esterel v5 Primer*, Gérard Berry, <http://www-sop.inria.fr/esterel.org/>
 - SugarCubes Implementation of Causality, F. Boussinot , INRIA Research Report, RR-3487, Sept., 1998

L'Approche Réactive/Synchrone

- Références:
 - *SAXO-RT: Interpreting Esterel Semantic on a Sequential Execution Structure*, E. Closse ; M. Poize ; J. Pulou ; P. Venier ; D. Weil, France Telecom R&D, SLAP'02
 - <http://www1.cs.columbia.edu/~sedwards/cec/>