Un exemple de langage synchrone : le langage Esterel

Frank Singhoff

Bureau C-207

Université de Brest, France

LISyC/EA 3883

singhoff@univ-brest.fr

Sommaire

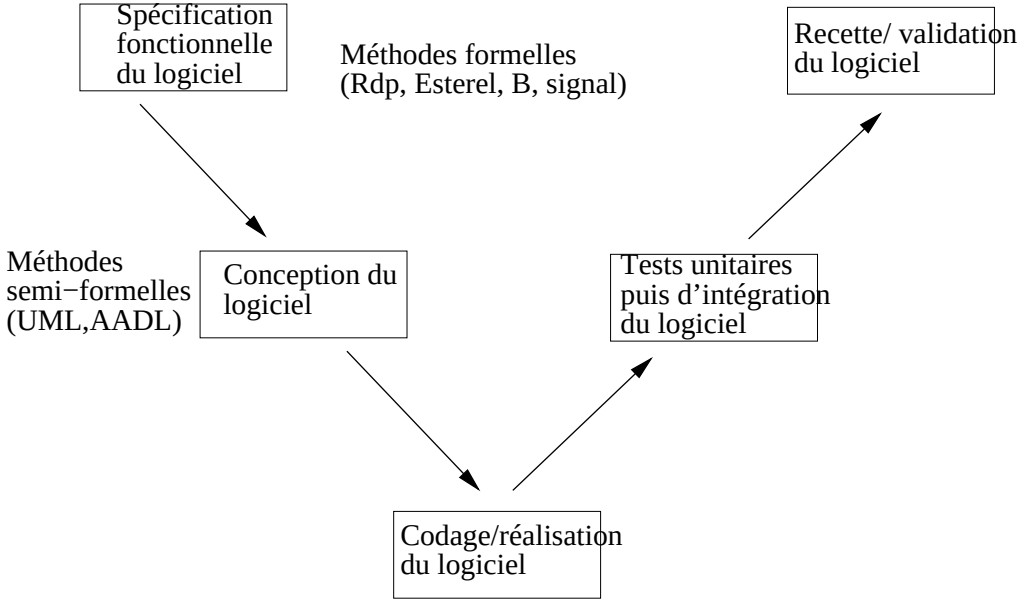
Génie logiciel et temps réel (1)

- Génie logiciel = méthodes, modèles et ateliers préconisés pour maîtriser la qualité des produits, leur coût et le respect des délais.

- **Spécificités des applications temps réel :**
  - Conséquences tragiques (vies humaines, faillites économiques).
  - Coût de développement trés lourd (validation temporelle et logique, applications peu flexible).
  - Maintenance souvent impossible (téléphone mobile, sonde spatiale) $\Longrightarrow$ erreur souvent irréversible.
  - Aspect statique et dynamique des applications (contre-ex : traitement batch). Synchronisation. Modélisation du temps.

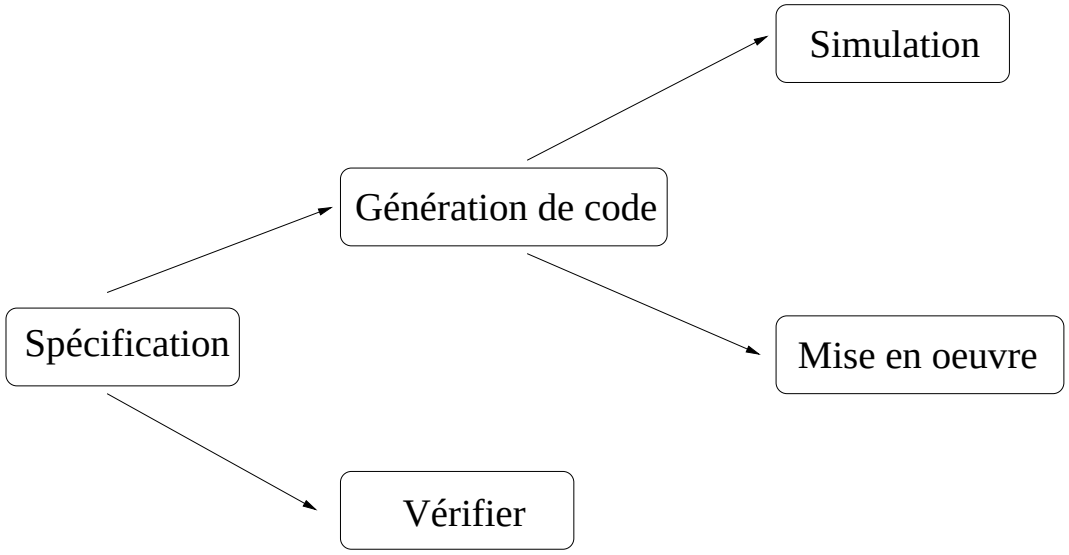
$\Longrightarrow$ Utilisation de méthodes, outils afin d'éviter au mieux tous ces pièges.

Génie logiciel et temps réel (2)



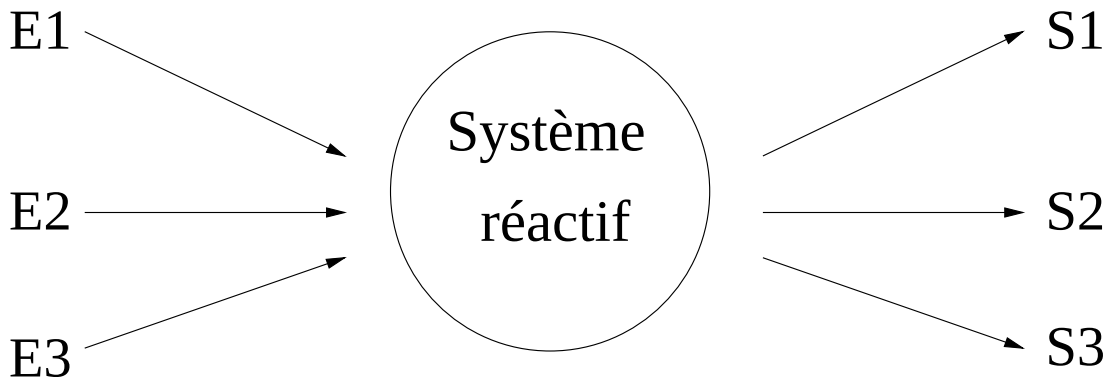
- Le cycle en V. Spécification = quoi faire ? Conception = comment faire ?
- Pourquoi spécifier :
  1. Contrat clair entre les différents intervenants d'un projet.
  2. Recherche de propriétés.
- Utilisable pendant les différentes étapes du cycle en V.

Les approches formelles



- **Objectifs :**
  - Spécifier et/ou concevoir formellement un système.
  - Vérifier des propriétés (ex : temporelles).
  - Génération de code pour effectuer des simulations.
  - Etre capable d'embarquer le logiciel.

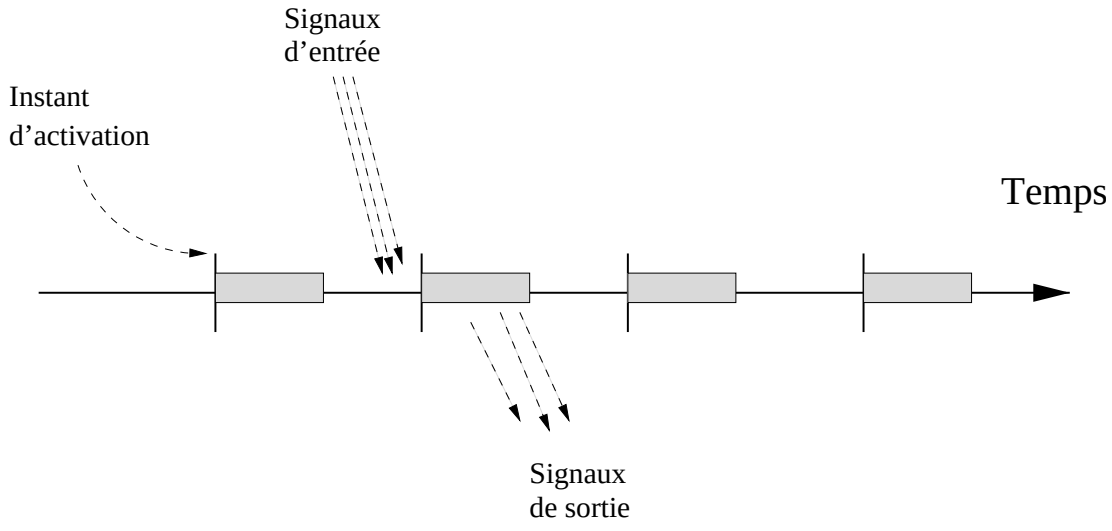
L’approche synchrone (1)



- **Système transformationnel :** système qui prend des données en entrée, et produit à son rythme des résultats. Pas d’interaction avec son environnement.
- **Systèmes réactifs[HAR 85] :** réagissent **immédiatement** à leur environnement. Activé par les entrées (événements). Fonctionne en mode "réflexe".

• Transformationnel = calcul "numérique" ; réactif = comportemental/actions.

L’approche synchrone (2)



- **Système synchrone :**
  - Modèle d’implantation des systèmes réactifs.
  - Système réactif dont le fonctionnement est régi par une horloge.
  - Temps discrétisé = facilite sa manipulation.
  - Temps d’exécution nul = temps de réaction plus court que le délai inter-arrivée des événements.
  - Exemples de langage synchrone : Lustre[HAL 91a], Signal[HAL 91b], Esterel[BER 92], StateChart[HAR 87], etc

Sommaire

1. Le modèle synchrone.
2. Présentation du langage Esterel
3. Que faire d'une spécification ?
4. Résumé.
5. Références.

Le langage Esterel

- Langage impératif et modulaire [BER 98b] (module, interface, composition).
- Notion de tâche mais parallélisme d'expression.
- Signaux et diffusion instantanée.
- Manipulation aisée du temps (temps multiforme).
- Intégration et génération de code en C/Ada $\Longrightarrow$ Esterel = contrôle uniquement.
- Déterminisme logique et temporel du code généré (automates, circuits booléens).
- Simulation/validation.
- Editeur : Esterel technologies (www.esterel-technologies.com). Exemples d'applications : Dassault, Audi, Airbus, Thalès, TI ...

Le langage Esterel

1. Généralités.
2. Instructions logiques.
3. Instructions temporelles.
4. A propos de l’expression du parallélisme.

Généralités

- Une application est un ensemble de modules.
- Un module est constitué d'une interface (signaux d'entrée et de sortie) et d'un corps (suite d'instructions).
- Types de données : $integer$, $boolean$, $double$, $float$, $string$ et types utilisateur.
- Deux types d'instruction Esterel :
  - Instructions logiques (temps d'exécution nul).
  - Instructions temporelles (temps d'exécution parfois non nul).
- Codes "transformationnels" (hors Esterel, langage hôte).

Instructions logiques

- Déclaration/affection d'une variable :

```
var i := 10 : integer in
  ...
  i:=i-1;
  ...
end var
```

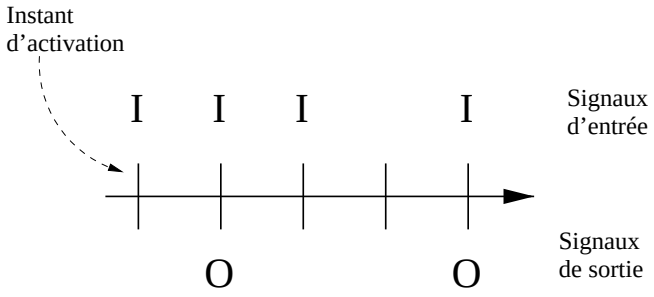
- La boucle :

```
loop
  ...
end loop
```

- La conditionnelle :

```
if cond
  then i1;
  else i2;
end if
```

Instructions temporelles (1)

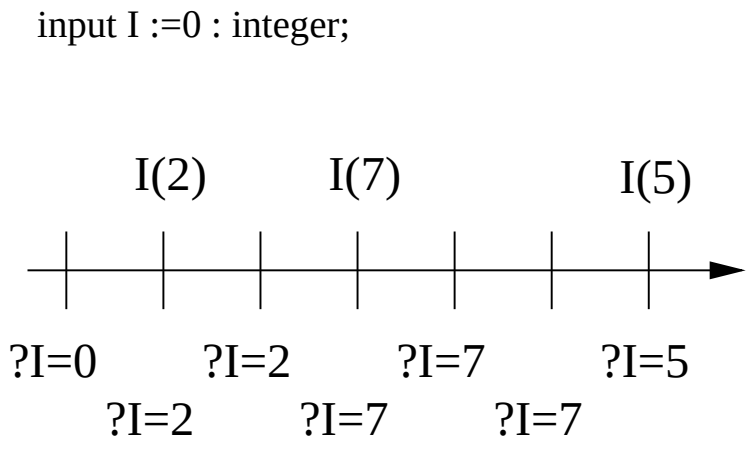



```
module premier:
   % Interface du module
   input I;
   output O;

   % Corps du module
   loop
       await 2 I;
       emit O;
   end loop
end module
```

- Modéliser le temps, c'est manipuler des signaux.
- $emit$ = diffusion instantanée d'un signal. Temps d'exécution nul.
- $await$ = attente et réception d'un signal. Temps d'exécution d'au moins un $tick$.

Instructions temporelles (2)

- Signaux **valués** vs signaux **purs** :




```
input I :=0 : integer;
output O : integer;

loop
   await I;
   emit O(?I);
end loop
```

- Valeur initiale.
- L'opérateur ? permet d'obtenir la **dernière valeur** du signal.
- Présence $\neq$ valeur.

Instructions temporelles (3)

- **Exemple :** signaux $\neq$ variables

```
module pair:

        input VAL : integer;
        output NB_PAIR : integer;
        output NB_IMPAIR : integer;

        var nb_nombre_pair :=0 : integer,
              nb_nombre_impair :=0 : integer in
           loop
                await VAL;
                if ?VAL mod 2=0
                  then nb_nombre_pair:=nb_nombre_pair+1;
                      emit NB_PAIR(nb_nombre_pair);
                  else nb_nombre_impair:=nb_nombre_impair+1;
                      emit NB_IMPAIR(nb_nombre_impair);
                end if;
           end loop
        end var
end module
```

Instructions temporelles (4)

- **Tout programme doit "prendre un temps d'exécution fini" :**

```
module pas_bon:
        loop
                x:=x+1;
        end loop

end module
```

$\Longrightarrow$ Dans une boucle, si l'on n'utilise pas d'instuction qui prend du temps ... le temps d'exécution est infini .....

$\Longrightarrow$ **Sémantique du programme**.

Instructions temporelles (5)

- **Autres exemples d'instructions :**

```
module troisieme :
        input I;
        output LA;
        output PAS_LA;

        loop
                present I
                        then emit LA;
                        else emit PAS_LA;
                end present;
                pause;
        end loop
end module
```

- $present$ = teste la présence d'un signal. Ne prend pas de temps.
- $pause$ = prend du temps ... et c'est tout.

Instructions temporelles (6)

- Le compteur d'événements :

```
input I;
output O;
every 2 I do
       emit O;
end every
```

- L'émission "permanente" :

```
output I;
sustain I;
```

- La préemption :

```
input I;
abort
       instructions;
when I
```

Instructions temporelles (7)

- **Exercice :**

Ecrire un programme Esterel qui analyse les données émises par un thermomètre. Le programme Esterel contient deux signaux :

- Un signal d'entrée valué par un entier qui fournit une nouvelle mesure de la température.
- Un signal de sortie valué par une chaîne de caractères qui émet un message indiquant si la température monte, descend ou reste stable vis-à-vis de la mesure précédente. Le message indiquant la tendance sur l'évolution de la température doit être émis à chaque fois qu'une nouvelle mesure est transmise par le thermomètre.

Branches parallèles (1)

- Parallélisme d'expression : le code Esterel est strictement séquentiel.
- Contraintes sur le partage des données.

- Instructions :
  - Séquence : `;`
  - Branches parallèles : `||`. Fonctionnement : une branche parallèle est terminée lorsque toutes ses composantes le sont.
  - Opérateur de priorité : `[p;q]`.

- Attention : la priorité de `;` et plus forte que celle de `||`. Ainsi `p;q||r` signifit en fait : `[p;q]||r`

Branches parallèles (2)

```
module ual_par :

        input OPERANDE1 : integer;
        input OPERANDE2 : integer;
        output RESULTAT : integer;

        loop
                [
                await OPERANDE1;
                ||
                await OPERANDE2;
                ];
                emit RESULTAT(?OPERANDE1+?OPERANDE2);
        end loop
end module
```

- L'émission de $RESULTAT$ est effectuée **seulement quand** $OPERANDE1$ **et** $OPERANDE2$ sont délivrés au programme.

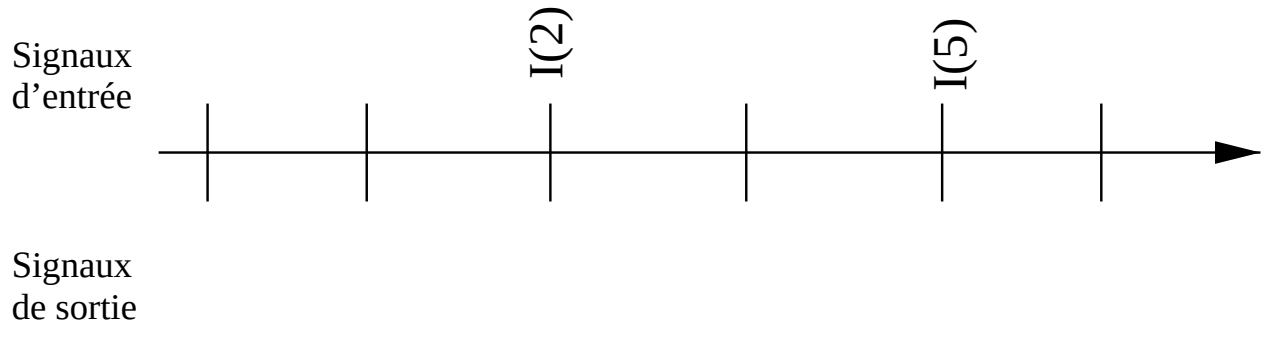
Branches parallèles (3)

- **Exercice : soit le programme Esterel suivant :**

```
module toto:

        input I :=0 : integer;
        output O : integer;

        loop
           await I;
          ||
        [
           emit O(?I);
           pause;
        ]
        end loop

end module
```

Complétez le chronogramme du transparent suivant, en ajoutant aux instants adéquats, les émissions des signaux de sortie.

Branches parallèles (4)



Branches parallèles (5)

```
input SORTIE;
input VALEUR : integer;
output SOMME : integer;

var total :=0 : integer in
        loop
                await SORTIE;
                emit SOMME(total);
        end loop
        ||
        loop
                await VALEUR;
                total:=total+?VALEUR;
        end loop
end var
```

- Variables partagées interdites... Pourquoi ?
- Propriété de causalité sur les événements observables dans le système.

Branches parallèles (6)

```
input VALEUR : integer;
output SOMME : integer;
input SORTIE;
signal LOCAL :=0 : integer in
        loop
                await [SORTIE and LOCAL];
                emit SOMME(?LOCAL);
        ||
        var mon_total :=0 : integer in
        loop
                present VALEUR then
                        mon_total:=mon_total+?VALEUR;
                end present;
                emit LOCAL(mon_total);
                pause;
 end signal
```

- Utilisation d'un signal local diffusé instantanément à toutes les branches parallèles.
- Propriété de causalité sur les occurrences de signaux.

Sommaire

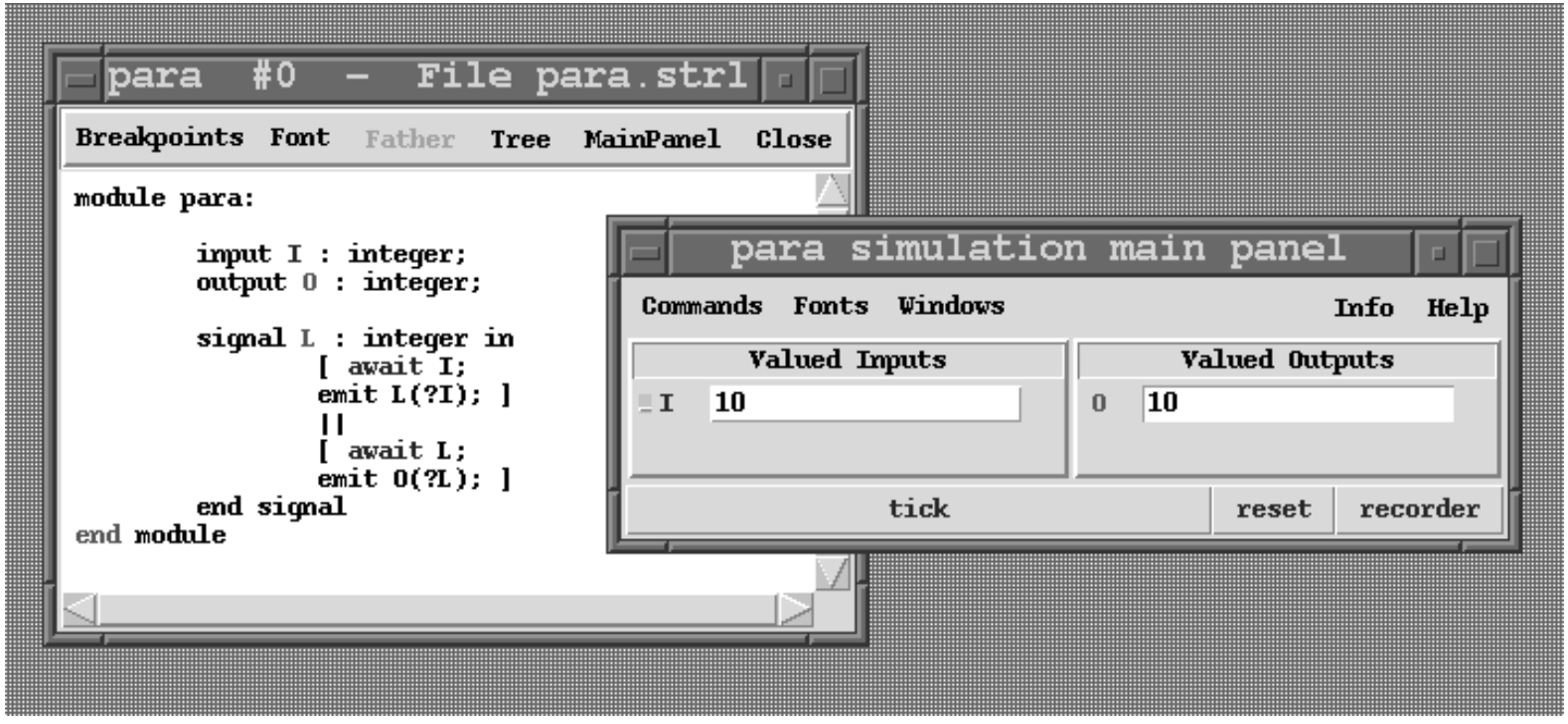
1. Le modèle synchrone.
2. Présentation du langage Esterel
3. Que faire d’une spécification ?
4. Résumé.
5. Références.

Que faire d’une spécification ?

1. Simuler.
2. Exécuter, voire embarquer.
3. Vérifier.

Simuler (1)

- **Simuler, c'est mettre au point le modèle/programme.**



- **Avec** Xes **:**

```
esterel -simul para.strl
gcc -c para.c
xes para.o
```

Simuler (2)

- **Fenêtre "interface module"** :
  - Tick/Reset : activation du module.
  - Signaux : rouge = présent (affichage valeur). bleu = absent.
  - Bouton gauche = active le signal.

- **Fenêtre "programme"** :
  - Instruction : rouge = en cours d'exécution.
  - Signaux : rouge = présent (affichage valeur). bleu = absent.
  - Variables/signaux : affichage valeur par click bouton gauche.

Exécuter, voire embarquer (1)

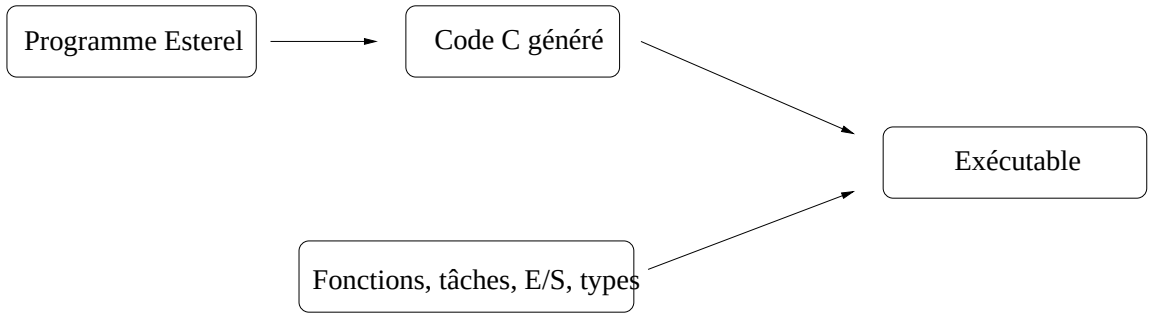
- Exécuter, voire embarquer : obtenir une spécification exécutable.
- Supprimer les potentielles erreurs de mise en oeuvre.

- **Difficultées :**
  - Disposer d'un environnment d'exécution garantissant les hypothèses du modèle synchrone.
  - Coopération avec l'environnement (qui est asynchrone).
  - Validité du générateur de code et de l'environnement d'exécution.

- Rappels : le code généré par le compilateur Esterel est déterministe du point de vue logique, mais aussi temporel.

Exécuter, voire embarquer (2)

- Exemple du générateur C du compilateur Esterel.



- **Ce qui est généré :**
  - Fonctions pour les signaux d'entrée.
  - Fonction d'initialisation du module.
  - Fonction d'activation du module.

- **Ce qui est donné par le programmeur :**
  - Fonctions pour les signaux de sortie.
  - Types de données.
  - Code transformationnel (tâches, procédures, fonctions).

Exécuter, voire embarquer (3)

```
module simple :

        input ENTREE : integer;
        output SORTIE : integer;

        loop
                await ENTREE;
                emit SORTIE(?ENTREE);
        end loop

end module
```

- **Compilation :**

```
esterel simple.strl
gcc -c simple.c
gcc -c simple_main.c
gcc simple.o simple_main.o -o simple
```

Exécuter, voire embarquer (4)

- **Programme *simple_main.c* :**

```
/* Fonctions générées par le compilateur esterel */
void simple(void);
void simple_reset(void);
void simple_I_ENTREE(int val);

/* Fonction à définir dans le programme */
void simple_O_SORTIE(int val) {
    printf("Emission du signal SORTIE : %d\n",val);
}

int main(int argc, char* argv[]) {
    simple_reset();   /* Initialisation du module  */
    simple();
    for(i=0; i<10; i++) {   /* Activation du module */
         simple_I_ENTREE(10);
         printf("Appel du programme Esterel\n");
         simple();
    ...
```

Interaction avec le transformationnel (1)

- Esterel est uniquement utilisé pour la partie "contrôle", "comportementale" d’un système.

- La partie "calcul", "transformationnel" du système est implantée par le biais d’un langage "hôte" :

1. Soit par appel de procédures, fonctions ou définition de types utilisateurs (interaction synchrone). Temps d’exécution nul.
2. Soit par utilisation de tâches Esterel (interaction asynchrone). Terminaison par signal Esterel et temps d’exécution éventuellement non nul.

Interaction avec le transformationnel (2)

```
module tampon:

        constant TAILLE_TAMPON : integer;

        type ELEMENT;
        type TAMPON;

        output TAMPON_FAMINE;
        output TAMPON_DEBORDEMENT;
        input  REQUETE_ECRITURE : ELEMENT;
        input  REQUETE_LECTURE;
        output REPONSE_LECTURE : ELEMENT;

        procedure consommeElement(TAMPON,ELEMENT)();
        procedure produitElement(TAMPON)(ELEMENT);
        function nbElements(TAMPON) : integer;
        function initTampon() : TAMPON;
```

Interaction avec le transformationnel (3)

```
var monTampon : TAMPON in monTampon:=initTampon();
    loop
       await
           case REQUETE_LECTURE do
                var donnee : ELEMENT in
                    if (nbElements(monTampon) > 0)
                        then    call consommeElement(monTampon,
                                                     donnee)();
                                emit REPONSE_LECTURE(donnee);
                        else    emit TAMPON_FAMINE;
                    end if
                end var
           case REQUETE_ECRITURE do
                if ( (nbElements(monTampon)+1) < TAILLE_TAMPON)
                    then    call produitElement(monTampon)
                                                (?REQUETE_ECRITURE);
                    else    emit TAMPON_DEBORDEMENT;
                end if
        end await
     end loop
end var
```

Vérifier (1)

- Vérifier : quelles sont les propriétés de la spécification ?

- **Méthodes :**

1. Automates (machine de Mealy[BER 98a]). Etat des signaux.
2. Circuit booléen (méthode plus récente et moins sujette à explosion combinatoire).

- Que dire du code transformationnel (tâches, procédures, fonctions) ??

- Exemple d'outils : Xeve/Fctools[BOU 97], TempEst[OLN 95], etc

Vérifier (2)

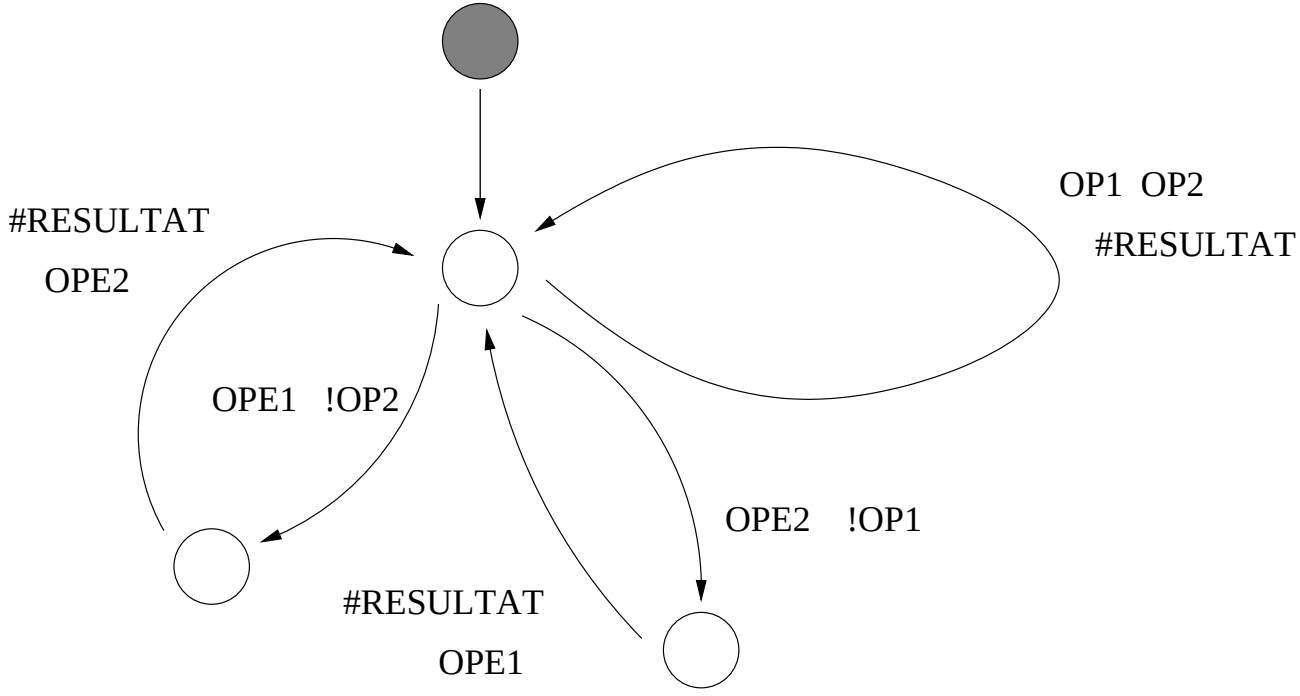
- **Exemple 1 :**

```
module ual_par :
        input OPE1 : integer;
        input OPE2 : integer;
        output RESULTAT : integer;

        loop
                [
                await OPE1;
                ||
                await OPE2;
                ];
                emit RESULTAT(?OPE1+?OPE2);
        end loop
end module
```

Vérifier (3)

- **Exemple 1 :**

X : X présent
!X : X absent
#X : X émis



Vérifier (4)

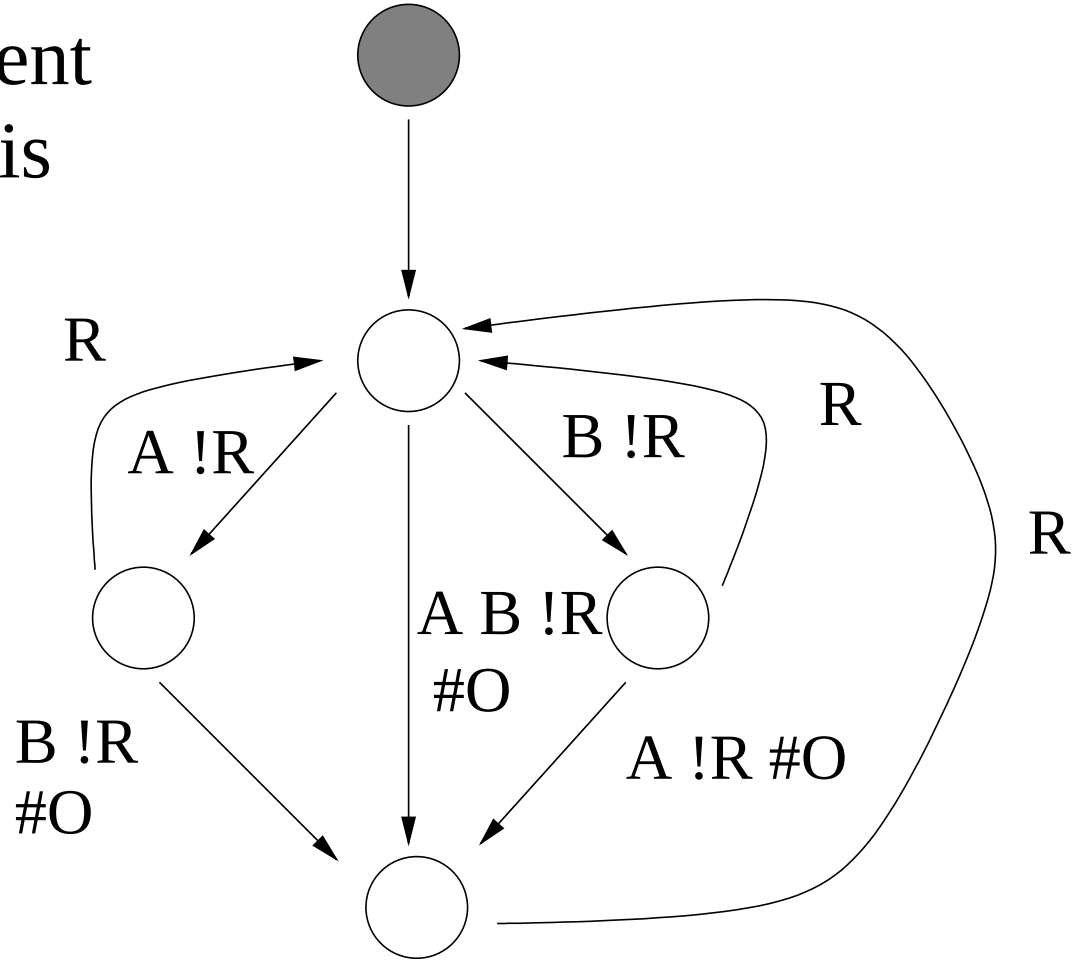
- **Exemple 2 :**

```
module abro:
        input A, B, R;
        output O;

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

Vérifier (5)

- **Exemple 2 :**

X : X présent
!X : X absent
#X : X émis



Sommaire

1. Le modèle synchrone.
2. Présentation du langage Esterel
3. Que faire d’une spécification ?
4. Résumé.
5. Références.

Résumé

- **Ce qu'il faut retenir :**
  - Méthodes formelles : spécifier, prouver, embarquer. **Ne concerne qu'une petite partie du code**.
  - Définition du modèle synchrone.
  - A propos du langage Esterel :
    1. Langage impératif.
    2. Déterminisme temporel et logique.
    3. Manipulation du temps.
    4. Expression du parallélisme, ordonnancement statique.
    5. Partie "comportementale" d'un système réactif $\Longrightarrow$ interaction avec son environnement, le code transformationnel.

Sommaire

1. Le modèle synchrone.
2. Présentation du langage Esterel
3. Que faire d’une spécification ?
4. Résumé.
5. Références.

Références (1)

[BER 92] G. Berry and G. Gonthier. « The Esterel Synchronous Programming Language: Design, Semantics, Implementation ». *Science Of Computer Programming*, 19(2):87–152, 1992.

[BER 98a] G. Berry. « The Esterel Language Primer V5.10 ». Technical Report, Ecole des Mines de Sophia Antipolis, CMA, March 1998.

[BER 98b] G. Berry. « The Foundations of Esterel ». in Proof, Language and Interaction: Essays in Honour of Robin Milner, G. Plotkin, C. Stirling and M. Tofte, editors, MIT Press, 1998.

[BOU 97] A. Bouali. « Xeve : An Esterel Verification Environnement (version v1.3) ». Technical Report, INRIA Technical Report RT-214, December 1997.

[HAL 91a] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. « Programmation et vérification des systèmes réactifs : le langage Lustre ». *Technique et Science Informatiques*, 10(2):139–157, 1991.

Références (2)

[HAL 91b] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. « The synchronous dataflow programming language Lustre ». *Proceedings of the IEEE*, 79(9):1305–1320, September 1991.

[HAR 85] D. Harel and A. Pnueli. « On the Developement of Reactive Systems ». pages 477–498. In Logic and Models of Concurrent Systems. Proc NATO Advanced Study Institute on Logics and Models for Verifications and Specification of Concurrent Systems. New York, 1985.

[HAR 87] D. Harel. « Statecharts: A visual formalism for complex systems ». *Science of Computer Programming*, 8(3):231–274, June 1987.

[OLN 95] L. Jategaonkar Jagadeesan C. Puchol J.E. Von Olnhausen. « Safety Property Verification of Esterel Programs and applications to telecommunications software ». Proc. of the Seventh Conference on Computer Aided Verification, July 1995.