Esterel de A à Z

6. Exec, vérification formelle, HipHop.js

Gérard Berry

Collège de France
Chaire Algorithmes, machines et langages

Cours du 28 mars 2018

Précédé du séminaire de Lionel Rieg

COLLÈGE DE FRANCE
1530

gerard.berry@college-de-france.fr
http://www-sop.inria.fr/members/Gerard.Berry

Vérification par abstraction d'automates

- Travaille sur les automates explicites (option –A)
- Abstraction des données et de signaux, réduction par bisimulation observationnelle (cf cours du)
- Systèmes :
  - Auto (R. de Simone, D. Vergamini) : calcul efficace des réductions
  - Autograph (V. Roy) : dessin interactif des automates réduits (Auto et Autograph ont eu bien d'autres applications)

Très utile, mais limité par le risque d'explosion des automates
Plus récents : UPPAAL (c. 30/03/2016), CADP (c. 06/04/2016)

Exemple : robotique en ORRCAD

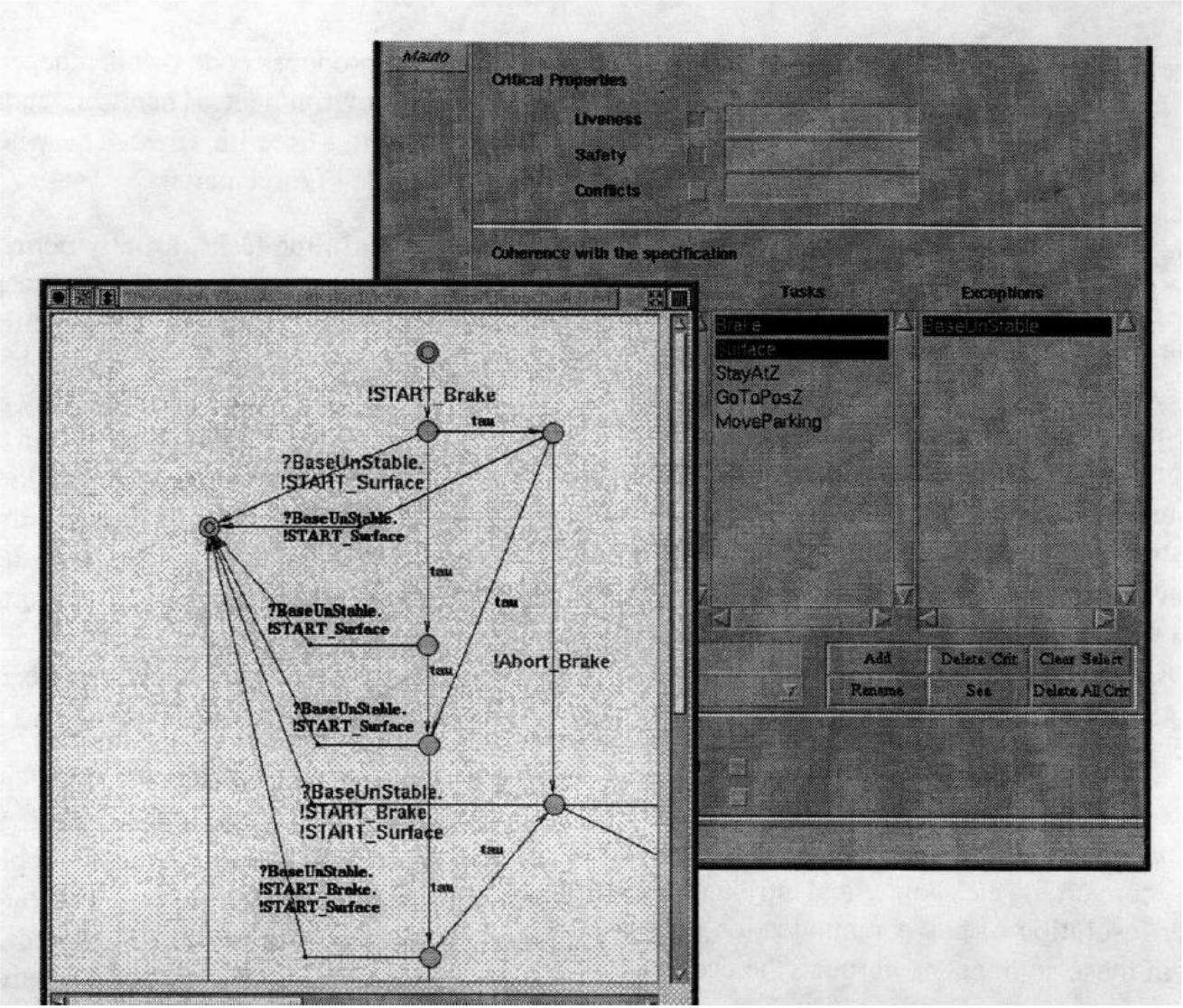


Fig. 12. The ORCCAD verification process.

Exemple : robotique en ORRCAD

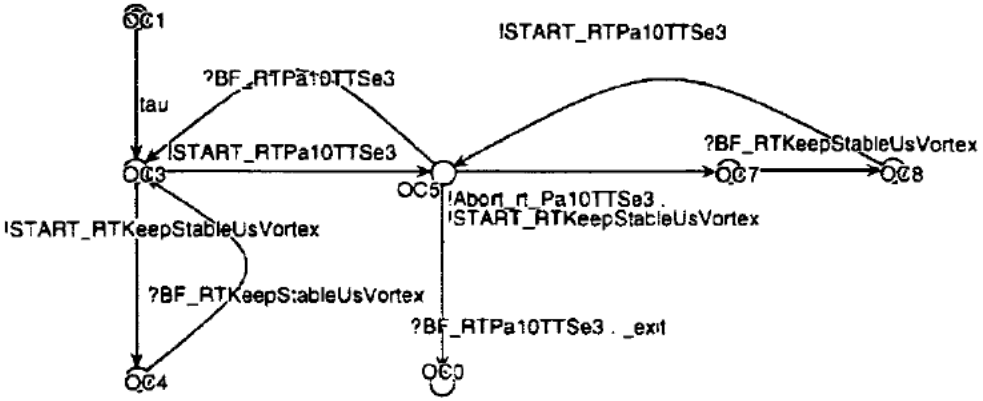


Fig. 14. Conformity with one of the specifications for the mission scenario.

The ORCCAD Team:
Jean-Jacques Borrelly
Eve Coste-Manière
Bernard Espiau
Konstantinos Kapellos
Roger Pissard-Gibollet
Daniel Simon
Nicolas Turro
INRIA Sophia Antipolis
INRIA Rhônes-Alpes

The International Journal of Robotics Research, April 1st, 1998

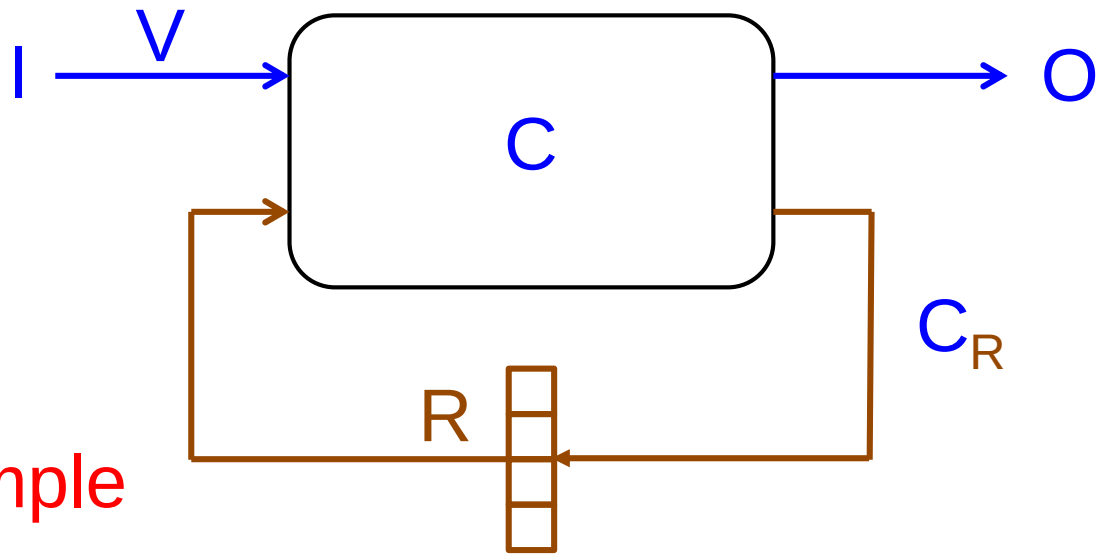
Vérification booléenne par BDDs

- Très utile, même si elle ne traite pas les données
- Utilisée en grand par Dassault Aviation et d'autres en logiciel, puis pour les circuits de contrôle (Intel, T.I., etc.)
- Fondée surtout sur le package de BDD TiGeR (O. Coudert, J-C. Madre et H. Touati) et développée par A. Bouali
- Principe : calcul des états atteignables, après abstraction éventuelles selon les propriétés à prouver
- Calcul de contre-exemples par exploration inverse de l'espace d'états

Utilité majeure aussi dans la génération de tests,
par contre-exemples à des propriétés évidemment fausses
Ascenseur :
Il est impossible de visiter tous les étages et d'y ouvrir la porte

2. Calcul des états accessibles (RSS)

(voir cours du 21 mars 2018)



contre-exemple

$R_0 = 0$

$R_1 = R_0 U C_R(R_0, V)$

...

$R_{i+1} = R_i U C_R(R_i, V)$

...

$R_{n+1} = R_n U C_R(R_n, V) = R_n$ $\Longrightarrow$ RSS=R_n

A chaque étape, utiliser R_i et V comme simplifieurs pour C_R

Bounded Model Checking en SAT / SMT

- Principe du Bounded Model Checking (*BMC*)
  - construire une formule SAT (contrôle) ou SMT (contrôle + données) par dépliement progressif de l'espace d'états

  → états accessibles en 1, 2, 3, … transitions
  - détecte toutes les erreurs, mais on ne sait pas quand…
  - des méthodes récentes incorporent la recherche automatique de propriétés inductives pour la preuve totale

Pour Esterel v7 : vérifieur SAT / SMT Prover SL de *Prover Technologies* (G. Stålmark)

L'ascenseur en Esterel v7

```
data SizeData :
    constant N : integer = 4;
end data

interface TimerIntf :
   output StartTimer ;
   input TimerExpired ;
end interface

interface ElevatorEngineIntf :
   output Start ;
   output Up;
   output Down ;
   output Stop ;
   input CabinStopped ;
end interface

// This interface comprises the sensors that tell the cabin is reaching a floor.
interface FloorSensorIntf :
   data SizeData ;
   input ReachingFloor [N];
end interface
```

```
interface CabinIntf :
   // user door control buttons
   input DoorOpen ;
   input DoorClose ;
   // the sensor unit that signals traversal of door ( light cell ) and shocks
   input DoorSensor ;
   // the sensors that signals proper door opening and closing
   input DoorIsOpen ;
   input DoorIsClosed ;
   // basic input relations
   input relation DoorIsClosed # DoorSensor ;
   input relation DoorIsClosed # DoorOpen ; // Alfred de Musset
   // commands to drive door opening and closing motors
   output OpenDoorMotorOn ;
   output OpenDoorMotorOff ;
   output CloseDoorMotorOn ;
   output CloseDoorMotorOff ;
end interface
```

```
interface ButtonsAndLightsIntf :
data SizeData ;
input CabinCall [N];              // numbered buttons in the cabin
input UpCall [N];                 // up buttons at floors (fake one at floor N -1)
input DownCall [N];               // down buttons at floors (fake one at floor 0)
output CurrentUp ;                // up arrows at floors and in cabin
output CurrentDown ;              // down arrows at floors and in cabin
output CurrentFloor [N];          // to light up current floor number (one-hot array )
output StoppedAtFloor [N];        // triggers a bell at floor
output PendingCabinCall [N];      // sent to the cabin floor button lights
output PendingUpCall [N];         // sent to floor up button lights
output PendingDownCall [N];       // sent to floor down button lights
output PendingCall [N];           // some pending call at floor N
end interface
```

```
module Cabin :
  extends TimerIntf ;
  extends CabinIntf ;
  input CabinStopped ; // the signal that tells the elevator is stopped
  loop
    trap Done in
      loop
        emit OpenDoorMotorOn ;
        await DoorIsOpen ;
        emit OpenDoorMotorOff ;
        abort
          finalize
            emit StartTimer ;
            await TimerExpired or DoorClose ;
            emit CloseDoorMotorOn ;
            await DoorIsClosed ;
            exit Done
          with
            emit CloseDoorMotorOff ;
          end finalize
        when DoorOpen or DoorSensor
      end loop
    end trap;
    await CabinStopped
  end loop
end module
```

```
module CallHandler :
   data SizeData ;
   extends ButtonsAndLightsIntf ;
   extends ElevatorEngineIntf ;
   extends FloorSensorIntf ;
input DoorIsClosed ; // from cabin
signal CallToGoUp , CallToGoDown in
   <compute current floor and decide whether to stop >
||
   <compute StoppedAtFloor [i] for each floor i>
||
   <compute pending calls >
||
   <compute calls to go up or down >
||
   <compute CurrentUp or CurrentDown directions >
||
   <start motor in appropriate direction >
||
   <internal consistency assertions >
end signal
end module
```

<compute current floor and decide whether to stop>

```
emit CurrentFloor [1];
always
  signal ReachingSomeFloor in
    for i < N dopar
      if ReachingFloor [i] then
        emit ReachingSomeFloor ;
        // set current floor i; automatically resets previous current floor
        emit CurrentFloor [i];
        // we have to stop if we are moving in a direction and
        // either there is a floor call for the same direction at this floor
        // or there is no more call at further floors in the same direction
        // warning : we have to take pre( CurrentUp ) because CurrentUp
        // is reset in the instant if there is no call up
        if ((pre(CurrentUp ) and (PendingUpCall [i] or not CallToGoUp))
            or (pre(CurrentDown ) and ( PendingDownCall [i] or not CallToGoDown)))
        then
          emit Stop
        end
      else
        // not reaching floor i , we simply need to cancel potential current
        // floor emission if some other floor has just been reached .
        emit CurrentFloor [i] <= pre( CurrentFloor [i]) and not ReachingSomeFloor
      end if
    end for
  end signal
end always
```

<compute StoppedAtFloor [i] for each floor i>

```
weak abort
   sustain StoppedAtFloor [1]
when Start ;
for i < N dopar
   loop
      await ReachingFloor [i] and Stop ;
      await CabinStopped ;
      weak abort
         sustain StoppedAtFloor [i]
      when Start
   end loop
end for
```

<compute pending calls >

```
for i < N dopar
   always
      if (not StoppedAtFloor [i]) then
         emit {
            PendingCabinCall [i] <= CabinCall [i] or pre(PendingCabinCall [i]),
            PendingUpCall [i] <= UpCall [i] or pre(PendingUpCall [i]),
            PendingDownCall [i] <= DownCall [i] or pre( PendingDownCall [i]),
            PendingCall [i] = PendingCabinCall [i]
                                 or PendingUpCall [i] or PendingDownCall [i],
         }
      end if
   end always
end for
```

<compute calls to go up or down >

```
signal AuxUpFloor [N], AuxCallToGoUp [N] in
  sustain {
    seq {
      for i < N -1 doup
        AuxUpFloor [i+1] <= CurrentFloor [i] or AuxUpFloor [i],
        AuxCallToGoUp [i+1] <= (PendingCall [i+1] and AuxUpFloor [i+1])
                                  or AuxCallToGoUp [i]
      end for
    },
    CallToGoUp <= AuxCallToGoUp [N -1]
  }
end signal
||
// computing whether there is a call for going down
// Warning : this code uses a combinational down carry chain
signal AuxDownFloor [N], AuxCallToGoDown [N] in
  sustain {
    seq {
      for i < N -1 dodown
        AuxDownFloor [i] <= CurrentFloor [i+1] or AuxDownFloor [i+1] ,
        AuxCallToGoDown [i] <= (PendingCall [i] and AuxDownFloor [i])
                                  or AuxCallToGoDown [i+1]
       end for
    },
    CallToGoDown <= AuxCallToGoDown [0]
  }
end signal
```

<compute CurrentUp or CurrentDown directions >

```
loop
  await
    case immediate CallToGoUp do
      abort
        sustain CurrentUp
      when not CallToGoUp
    case immediate CallToGoDown do
      abort
        sustain CurrentDown
      when not CallToGoUp
  end await
end loop
```

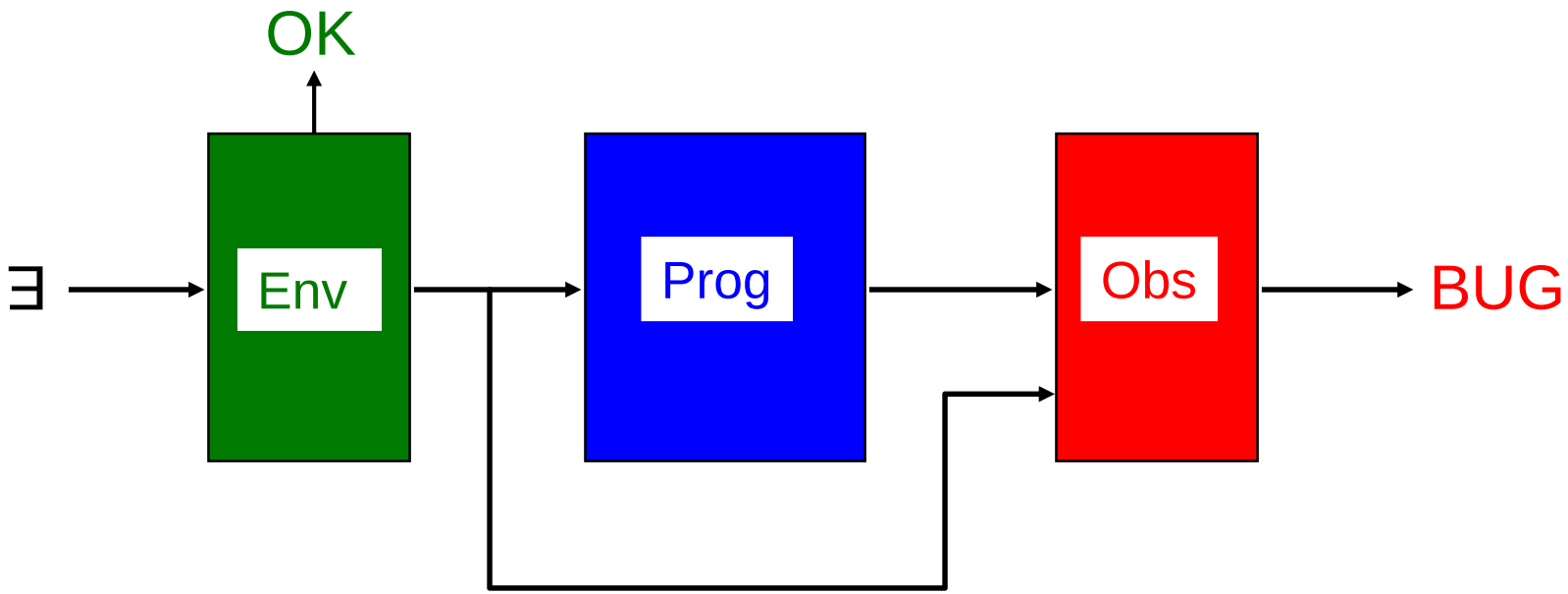
<start motor in appropriate direction >

```
loop
  await StartOK ;
  await immediate CallToGoUp or CallToGoDown ;
  emit Start ;
  if
    case pre(CurrentUp) and CallToGoUp do
      emit Up
    case pre(CurrentDown ) and CallToGoDown do
      emit Down
    case CallToGoUp do
      emit Up
    case CallToGoDown do
      emit Down
  end if
end loop
```

Assertions de cohérence interne

```
sustain {
   // there is always a direction given at start time
   assert SomeDirection = Start => (Up or Down),
   // always start up at bottom floor
   assert NoBottomCrash = (StoppedAtFloor [0] and Start ) => Up ,
   // always start down at top floor
   assert NoTopCrash = (StoppedAtFloor [N -1] and Start ) => Down ,
   // tell stopped at only one floor at a time (as is, not parametric in N)
   assert StopOK =   StoppedAtFloor [0] # StoppedAtFloor [1]
                          # StoppedAtFloor [2] # StoppedAtFloor [3]
}
```

Vérification par observateurs (N. Halbwachs)



Pour toutes les séquences d'entrée OK,
l'observateur n'émet jamais BUG

Assertion temporelle : l'ascenseur ne voyage pas la porte ouverte

```
signal Running in
   loop
      await Start ;
      abort
           sustain Running
      when CabinStopped
   end loop
||
   loop
      await OpenDoorMotorOn ;
      abort
          sustain assert DoorOpenedWhenMovingBug if Running
      when DoorIsClosed
   end loop
end signal
```

Définition de l'environnement

- Mais l'ascenseur n'évolue pas dans un environnement physique quelconque !

⟹ caractérisation de l'environnement, indispensable pour vérifier les propriétés et produire des contre-exemples pertinents en réduisant les entrées et l'espace d'états

La caractérisation de l'environnement est souvent longue et délicate !
… mais elle nous en apprend beaucoup

Contraintes d'environnement de l'ascenseur

```
trap Bad in
   always
      // no up call button at top floor
      if UpCall [N-1] then exit Bad end
   ||
      // no down call button at bottom floor
      if DownCall [0] then exit Bad end
   ||
      // elevator is at most at one floor at a time
      if not (    ReachingFloor [0]
                # ReachingFloor [1]
                # ReachingFloor [2]
                # ReachingFloor [3] ) then
         exit Bad
      end if
   ||
      // A door must be open or closed ( Musset )
      if not ( DoorIsOpen # DoorIsClosed ) then
         exit Bad
      end if
   end always
```

peut mieux faire !

Contraintes d'environnement de l'ascenseur

```
||
  // at most one TimerExpired after StartTimer
  abort
    every immediate TimerExpired do exit Bad end
  when StartTimer ;
  loop
    if TimerExpired then exit Bad end;
    await TimerExpired ;
    every TimerExpired do exit   end
  each StartTimer
||
  // at most one CabinStopped with one tick to react
  loop
    weak abort
      every immediate CabinStopped do exit Bad end
    when Stop ;
    await CabinStopped ;
    pause
  end loop
```

Contraintes d'environnement de l'ascenseur

```
||
  signal ReachingSomeFloor , ShiftedCurrentFloor [N+3] ,
         MovingUp , MovingDown in
    sustain {
      for i < N dopar
        ReachingSomeFloor <= ReachingFloor [i]
      end for
    }
  ||
    // cannot reach a floor between Stop and Start
    loop
      weak abort
        every ReachingSomeFloor do exit Bad end
      when Start ;
      await Stop
    end loop
```

Contraintes d'environnement de l'ascenseur

```
||
  signal ReachingSomeFloor , ShiftedCurrentFloor [N+3] ,
         MovingUp , MovingDown in
    …
  ||
    // when at floor i , the next floor is necessarily i+1 or i-1
    loop
      abort
        sustain {
          for i < N do
            ShiftedCurrentFloor [i+1] <= pre( ShiftedCurrentFloor [i+1])
          end for
        }
      when immediate ReachingSomeFloor ;
      emit {
        for i < N do
          ShiftedCurrentFloor [i +1] <= ReachingFloor [i]
        end for
      };
      pause
    end loop
```

Contraintes d'environnement de l'ascenseur

```
||
  signal ReachingSomeFloor , ShiftedCurrentFloor [N+3] ,
         MovingUp , MovingDown in
    …
  ||
    loop
      await Start ;
      weak abort
        if
          case Up do sustain MovingUp
          case Down do sustain MovingDown
          default do emit Ignore
        end if
      when Stop
    end loop
```

Contraintes d'environnement de l'ascenseur

```
    ||
      signal …
      ||
        emit ShiftedCurrentFloor [2] ;
        weak abort
          every immediate ReachingSomeFloor do exit Bad end
        when Start ;
        loop
          await ReachingSomeFloor ;
          for i<N dopar
            if ReachingFloor [i] then
              if MovingUp and not pre(ShiftedCurrentFloor [i]) then
                exit Bad
              end if ;
              if MovingDown and not pre(ShiftedCurrentFloor [i+2]) then
                exit Bad
              end if ;
            end if
          end for
        end loop
      end signal
    handle Bad do
      emit Ignore
    end trap
  end module
```

Agenda

1. La vérification formelle en Esterel
2. Interfacer le code généré et le simuler

Comment exécuter un programme Esterel

- En circuits : engendrer du Verilog, utiliser une CAO standard
- En logiciel : plusieurs manières équivalentes d'engendrer un code C (ou C++, etc.), et plusieurs façons de l'intégrer dans du code plus général
  - principe : définir des fonctions associées aux sorties, donner les entrées, appeler la fonction de réaction
  - pas de contrainte spécifique sur quand et comment
  - mais une contrainte absolue : l'exécution du code de la fonction de réaction doit être atomique.
  - en particulier, les valeurs des entrées doivent rester constantes pendant toute la réaction

Exemple pour ABRO

```
void ABRO_O_O () {
   printf(”O emitted\n”);
}

void main () {
   ABRO_reset ();  // ABRO initialization
   ABRO ();        // empty tick
   ABRO_I_A ();    // input A
   ABRO_I_B ();    // input B, same instant
   ABRO ();        // tick, emits O
   ABRO_I_R ();    // input R, reset
}
```

BLEU : à définir par l’utilisateur

MARRON : défini par le compilateur Esterel

Simuler un programme Esterel

- Seule chose à définir : les types et fonctions de données
- Trois types de simulation :
  - textuelle : entrée des signaux au terminal ou par fichier
    traçage optionnel des signaux, variables, etc.
  - graphique : xes, Esterel Studio
    entrées par clics, sorties visuelles
    animation colorée du code source
  - mixte : magnétophone dans les simulateurs graphiques
    lisant et écrivant les formats textuels

Usage : debugging, non-régression,
exécution des tests engendrés automatiquement,
visulalisation des contre-exemples de vérification

Agenda

1. La vérification formelle en Esterel
2. Interfacer le code généré et le simuler
3. Exec : contrôle d’activités asynchrones

L'instruction Exec

- Idée : appeler des fonctions (ou tâches) exécutées de façon asynchrone
- Attendre leur retour pour terminer au sens d'Esterel, lors d'un instant suivant
- Les suspendre ou les tuer si l'instruction exec est suspendue ou tuée par Esterel suspend, abort, exit, etc.

Usage : calculs longs, appels réseaux, mouvements de bras de robots, etc.

Exemple

```
input A, B, X;
return R;
var V, W : integer in
   loop
      suspend
         weak abort
            exec Task (V, W)(?X+1)
         when B
      when A
   end loop
end var
```

valeurs et retours par référence

valeurs

réincarnation ⟹
passer un unique id à chaque appel

Interface bas niveau d'Exec (Esterel v5)

```
typedef struct {
   unsigned int start : 1 ;
   unsigned int kill : 1 ;
   unsigned int active : 1 ;
   unsigned int suspended : 1 ;
   unsigned int prev_active : 1 ;
   unsigned int prev_suspended : 1 ; /* to generate suspend / resume */
   unsigned int exec_index ;          /* unique id in the program */
   unsigned int task_exec_index ;     /* unique run-time id*/
   void (*pStart)() ;                 /* takes a function as argument */
   int (*pRet)() ;                    /* may take a value as argument */
} __ExecStatus ;
```

Interface haut niveau d'Exec (Esterel v5)

```
#include "exec_status.h"
my_start () { ... }
my_kill () { ... }
my_suspend () { ... }
my_resume () { ... }

STD_EXEC (R1, REACT, my_start_1, my_kill_1, my_suspend_1, my_resume_1);

STD_EXEC (R2, REACT, my_start_2, my_kill_2, my_suspend_2, my_resume_2);

/*engendrer automatiquement un STD_EXEC pour chaque tâche */
STD_EXEC_FOR_TASK (TASK, REACT,
                   my_start, my_kill, my_suspend, my_resume);
```

Précis, mais assez lourd à l'usage
Très simplifié en HipHop.js (programmation fonctionnelle)

```
function execColor(langPair) {
                return MODULE (IN text, OUT color, OUT trans, OUT error) {
                  EMIT color("red");
                  AWAIT IMMEDIATE(NOW(text));
                  PROMISE trans, error translate(langPair, VAL(text));
                  EMIT color("green");
                }
              }

    window.onload = function() {
                hh = require("hiphop");
                m = new hh.ReactiveMachine(MODULE (IN text,
                                                   OUT transEn, OUT colorEn,
                                                   OUT transNe, OUT colorNe,
                                                   OUT transEs, OUT colorEs,
                                                   OUT transSe, OUT colorSe) {
                  LOOPEACH(NOW(text)) {
                            FORK {
                              RUN(execColor("fr|en"), color=colorEn, trans=transEn);
                            } PAR {
                              RUN(execColor("en|fr"), color=colorNe, text=transEn, trans=transNe);
                            } PAR {
                              RUN(execColor("fr|es"), color=colorEs, trans=transEs);
                            } PAR {
                              RUN(execColor("es|fr"), color=colorSe, text=transEs, trans=transSe);
                            }
                  }
                }, {debuggerName: "debug"});
              }
  }
```