# SYRF Case Studies
# The SAAB Climatic Chamber
# An Experience with Mode-Automata

Florence Maraninchi Yann Rémond

VERIMAG*– Centre Equation, 2 Av. de Vignate – F38610 GIERES
http://www-verimag.imag.fr/PEOPLE/Florence.Maraninchi
(Florence.Maraninchi|Yann.Remond)@imag.fr
Fax : (33) 4.76.63.48.50

**Abstract**

The SAAB case study is a small system devoted to temperature regulation in the cabin of an airplane. It is given as an ASA+ program, with informal documentation (see details below). The informal description talks about *modes* of the system: it is a typical case to which we can try to apply *mode-automata*. Mode-automata have been proposed in [MR98], as a means of introducing some imperative control structure in the dataflow language Lustre.

We aim at demonstrating that this new structure is useful in a dataflow language. This is not a case study for evaluating the efficiency of the compilation process. Indeed, we used the first implementation of mode-automata, i.e. a translation into pure Lustre.

## 1 The case-study

The SAAB case study is a small system devoted to temperature regulation in the cabin of an airplane.

### 1.1 System Interface and informal behavior

We restricted ourselves to the main controller. The complete system connects this main controller to a component that filters the inputs, and to a component that formats the outputs.

Figure 1 describes the system interface with the physical environment under control (a fan, a heater, a thermometer and flow measurement device) and with the human environment (lamps and buttons).

The behavior is a follows:

- The human user chooses a temperature and a ventilation time (the time that it takes to completely change the air in the cabin), using buttons of the interface.

*Verimag is a joint laboratory of Université Joseph Fourier (Grenoble I), CNRS and INPG

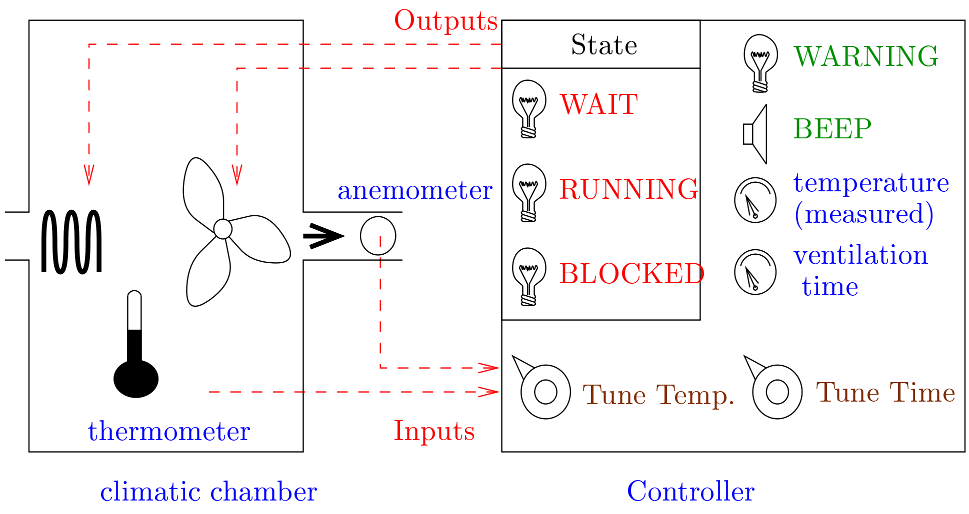


Figure 1: The system interface

- The regulation system continuously measures the flowing rate and the temperature (the inputs), and performs actions on the fan and the heater (the outputs) in order to make the physical parameters reach the values requested by the user, and then stay in a predefined range around these values.

Note that there is an implicit notion of running mode here, since the behavior of the regulation system is likely to be different when trying to reach a given value of the temperature, and when trying to maintain this temperature within a given range. The two behaviors are indeed identified as such in the program we study (see below).

### 1.2 Sources

The main source is an ASA+ program, with a two-pages documentation. ASA+ is a language for describing reactive systems. It is based upon the hierarchic dataflow structure of a program like in SADT, but the leaves of this structure are automata. All these automata are supposed to run in parallel in an *asynchronous* way. However, the particular program we got for the SAAB example is designed in such a way that the automata on the leaves of the program may take transitions only `if clock`, where `clock` is a global synchronization signal transmitted from the toplevel to all subprograms.

We also have a Lustre version, written by hand — but in a systematic way — from the ASA+ version. This is possible due to the quite synchronous behavior of the ASA+ program.

The Lustre program may be compiled and simulated, which is not the case for the ASA+ one.

# 2 Building a Mode-Automaton Program

## 2.1 Basic ideas

The main idea is the following: the informal documentation of the example explains the behavior of the system in terms of 5 main *modes*.

The ASA+ program has a variable `mode`, assigned to by one particular automaton, in which the mode-structure of the system is visible: modes are states, and each transition from state `A` to state `B` assigns the value `B` to the variable `mode`, in such a way that the variable reflects the current state. This is redundant, of course. However, copying the state into a variable allows the other components to read the current state; in ASA+, one automaton cannot access the state of another one.

Indeed, all the other behaviors of the leaves refer to this `mode` variable explicitly, like in `provided (mode=work or mode=wait...)`.

The mode-structure of the system is duplicated, and difficult to read.

## 2.2 The Approach

For obtaining a mode-automaton program of the SAAB case study, we took the following approach.

We extracted the mode-structure of the system, from the informal documentation, and from the ASA+ code (`modeselect.bdy`). Then we read all the other module bodies, gathering information about the activity of the system in each mode. It was a quite systematic task. In some sense, we *projected* each module body onto each mode (replacing the variable `mode` by one particular value), and obtained a simplified program.

## 2.3 The main mode-structure

Figure 2 gives the main mode-structure of the system, extracted from the informal documentation and one component of the ASA+ program.

## 2.4 Conditions for changing modes

Figure 3 decorates the mode-structure with some conditions for changing modes. These conditions were extracted from the ASA+ component that manages the variable `mode`. We performed a kind of "common sub-expressions" analysis in order to identify conditions. In particular, conditions of the form: $\mid a - b \mid > T$ were written in a number of different ways.

## 2.5 Programs of the modes

Finally, by gathering, for each mode `M`, all the program pieces computed `provided (mode = M)`, we obtain the programs attached to modes (Figure 4).

The Lustre equations use some auxiliary functional nodes like `calcfan`. These nodes are given in the annex.

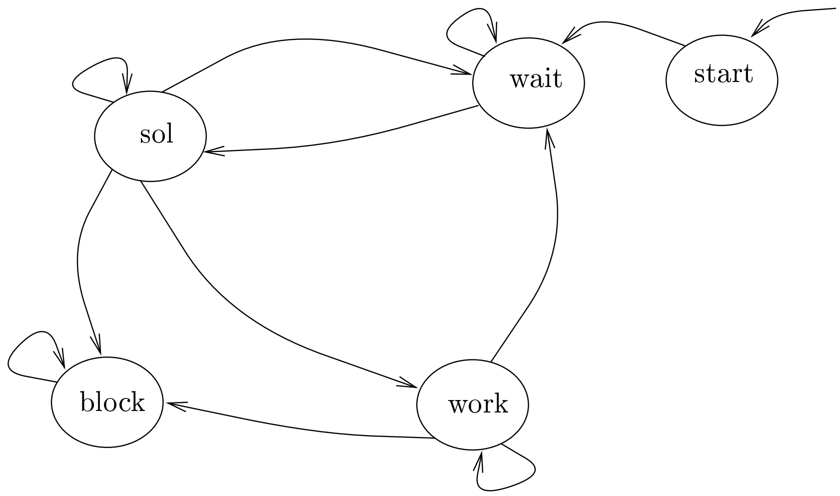


Figure 2: The main mode-structure

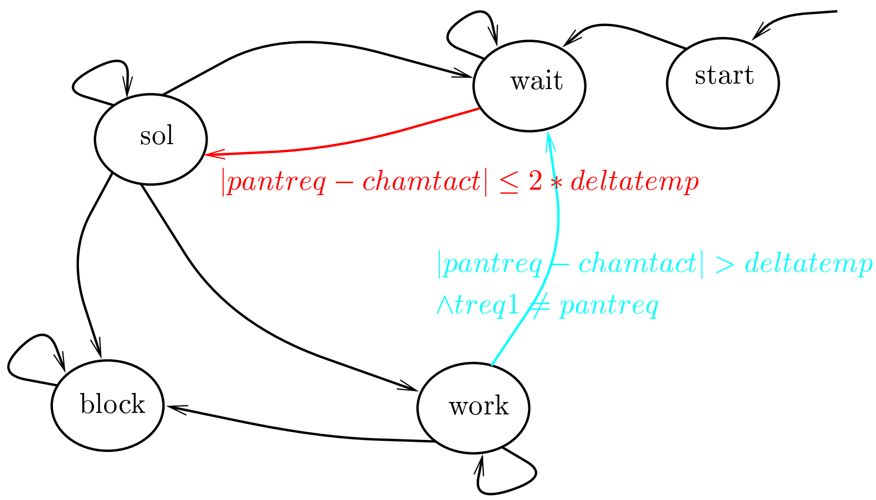


Figure 3: The main mode-structure with mode changes

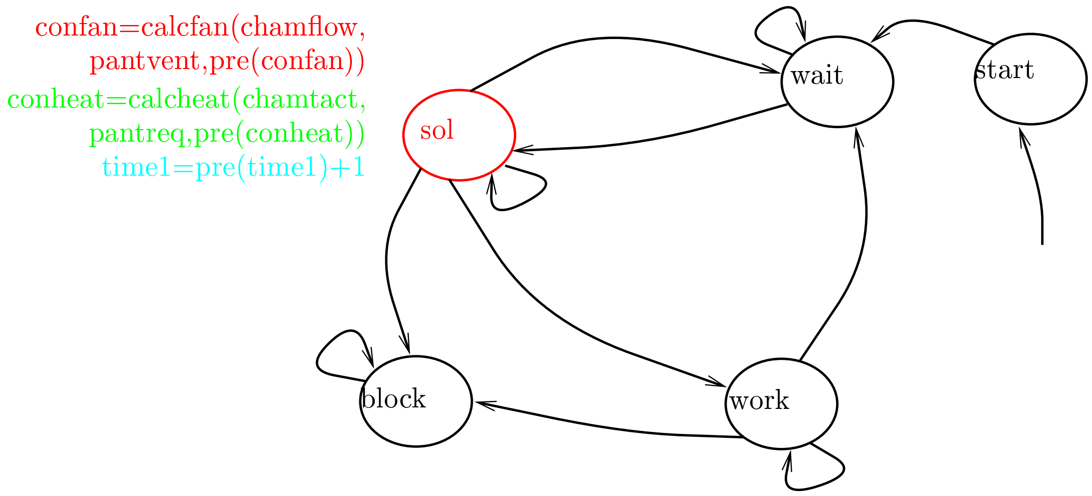


Figure 4: The main mode-structure with the programs attached to modes

## 2.6 Remarks on the structure

### 2.6.1 Variables local to a mode, and counters

In the simplified view of Figure 4, we observe the equation `time1=pre(time1)+1` attached to state `sol`. All the other states define `time1=0`. This is a typical case where the variable `time1` is used as a counter, for determining how much time has been spent in state `sol` (how many "ticks" of the basic clock, but the basic clock can be related to physical time if needed). The variable `time1` reaching a given value is a condition for leaving state `sol`. See for instance (in the program page 8) the condition `(time1>=(10*solution_time))` on a transition leaving state `sol`.

Writing the equation `time1=0` attached to all other modes is the only way to ensure that `time1` is re-initialized to 0 each time `sol` is entered, from another state. But it is likely to yield code in which the imperative assignment `v_time1 = 0` is performed at each instant of the basic clock (unless we can perform a complex static analysis to determine that this is not necessary). Moreover the equation `time1=0` has to be duplicated on all states, which is never a good thing.

Hence we should provide a *local variable* mechanism.

### 2.6.2 The conditions are combinational

When defining mode-automata, We decided not to write `pres` in the conditions of the transitions, mainly for readability. Putting `pres` in the conditions interferes in a quite unintuitive way with the equations attached to states.

It appears that we need not introduce such `pres` for the example.

## 3 Technical Stuff

We used the first implementation of mode-automata, i.e. the translation into pure Lustre (we know have a compiler from mode-automata to DC, see deliverable of task WP2.2).

Since we did not rewrite the whole program, but only the controller part, we had to mix original lustre code with the part produced from mode-automata. At the Lustre level, it is relatively simple.

## 4 Evaluation of the results

The main question now is: how can we compare the two (or even three) versions of the system? We got a mode-automaton version of the regulation system, but there is no guarantee that it indeed behaves *as expected*. Moreover, our notion of correctness is given by another program, not by a higher level specification. The ASA+ program may be a specific implementation of a more general specification.

Anyway, we have two Lustre programs. 1) The one written by hand, following the structure of the ASA+ program; this one is relatively relevant since the ASA+ program is indeed synchronous. 2) The one obtained by the compiler of mode-automata into pure Lustre, from our brand new description of the system.

We cannot *prove* formally that the two programs are equivalent, although Lustre has a formal semantics, because they involve numeric computations, and the equivalence is therefore undecidable.

We decided to test the two programs together, using a set of test sequences generated by Lurette [RWNH98] from a description of the physical environment given in WP7 (see deliverables ...). The observation is that the two systems *do not* behave the same.

However, they do not differ to much, and they behave *almost* the same. We tried to formalize this notion: it appears that the two behaviors sometimes differ in time. For instance, the emission of a command to the heater may last one an instant of the basic clock more (or start one instant later) in the mode-automaton version than in the Lustre program obtained from the ASA+ code. Sometimes it is the contrary.

Our opinion is that this difference is not relevant, and we can indeed consider that our program behaves as expected. It is based upon the following observation: the ASA+ code is a particular implementation of an unknown specification, which is likely to be far less precise about the exact timing of the outputs than the ASA+ code. This is because the system is a *regulation* system, not an event-driven system. The inputs are obtained by sampling continuous physical phenomena, and the tick is only the sampling rate. If the mode-automaton version of the program implies that the outputs corresponding to a given input are produced too late, it probably means that the basic clock is too low.

## 5 Conclusion of the case study

The aim of the case-study was to demonstrate the use of a new construct in Lustre. We think that the result is promising: for this kind of regulation system, where the mode-structure is part of the informal specification, the new construct is appropriate.

The program we obtain is far smaller than the ASA+ one, and far more readable then the Lustre version: dealing with modes is done in one single place, and the programs of the modes are simple. As a side-effect of the reengineering process we performed on the example, we identified some common expressions in the definitions of flows, which led to the auxiliary functional nodes written in Lustre. When we observe the mode-structure of the mode-automaton, we clearly see where the modes differ.

The case-study was successfull for evaluating the new construct, and we can now work on an efficient implementation, which implies a translation into Lustre with clocks, or directly into DC.

## References


[MR98] Maraninchi (F.) et Rémond (Y.). – Mode-automata: About modes and states for reactive systems. *In : European Symposium On Programming.* – Lisbon (Portugal), mars 1998.

[RWNH98] Raymond (P.), Weber (D.), Nicollin (X.) et Halbwachs (N.). – Automatic testing of reactive systems. *In : 19th IEEE Real-Time Systems Symposium.* – Madrid, Spain, décembre 1998.

# A The mode-automaton program

## A.1 The automaton part in targos

```
TARGOS

AUTOMATON        SAAB
STATES
    start    init    [ confan  = 100;
                       conheat = pre(conheat)+10;
                       treq1   = pantreq ; ]

    wait             [ confan  = calcfan(chamflow,pantvent,pre(confan));
                       conheat = calcheat(chamtact,pantreq,pre(conheat));
                       treq1   = pantreq;
                       time1   = 0; ]

    sol              [ confan  = calcfan(chamflow,pantvent,pre(confan));
                       conheat = calcheat(chamtact,pantreq,pre(conheat));
                       time1   = pre(time1+1); ]

    work             [ confan  = calcfan(chamflow,pantvent,pre(confan));
                       conheat = calcheat(chamtact,pantreq,pre(conheat));
                       time1   = 0; ]

    block            [ confan  = if btime>tblock then 0
                                 else calcfan(chamflow,pantvent,pre(confan));
                       conheat = 0;
                       time1   = 0;
                       starttime = pre(starttime)+1;
                       btime   = starttime div 60; ]

TRANS

    FROM start TO wait WITH  [1: true ]

    FROM wait  TO sol  WITH  [1: abs(pantreq-chamtact)<=(2*deltatemp) ]

    FROM sol   TO wait WITH  [1: (abs(pantreq-chamtact)>deltatemp) and
                                    (treq1<>pantreq) ]
    FROM sol   TO work WITH  [2: (abs(pantreq-chamtact)<=(2*deltatemp)) and
                                    ((tempd div 5)<tgrad) and
                                    (time1>=(10*solution_time)) ]
    FROM sol   TO block WITH [3:
               not (
                     ( (abs(pantreq-chamtact)>deltatemp)
                       and   (treq1<>pantreq))
               or    ( (abs(pantreq-chamtact)<=(2*deltatemp))
                       and ((tempd div 5)<tgrad)
                       and (time1>=(10*solution_time)))
               or    ( (abs(pantreq-chamtact)<=(2*deltatemp))
                       and ((tempd div 5)<tgrad)
```

```
                        and (time1<(10*solution_time)))
                )
                ]

    FROM work  TO wait WITH rien0 [1: (abs(pantreq-chamtact)>deltatemp) and
                                      (treq1<>pantreq) ]

    FROM work  TO block WITH rien0  [2:
                not (
                      ( (abs(pantreq-chamtact)>deltatemp)
                        and (treq1<>pantreq))
                or    ( (abs(pantreq-chamtact)<=(2*deltatemp))
                        and ((tempd div 5)<tgrad))
                )
                ]




PROCESS SAAB [ in (pantreq,          -- inputs
                   pantvent,
                   chamtact,
                   chamflow,
                   deltatemp,
                   tgrad,
                   solution_time,
                   tblock,
                   tempd : int),
               out (treq1=200,       -- outputs, with initial value
                    time1=0,
                    confan=0,
                    conheat=0,
                    starttime=0,
                    btime=0 : int),
               function (abs,calcfan,calcheat)   -- imported Lustre nodes
             ]
RAFF SAAB -- the program structure : a single mode-automaton.


[
node abs      (a :int) returns (b:int); imported;
node calcfan  (chamflow,pantvent,pconfan : int) returns (confan : int); imported;
node calcheat (chamtact,pantreq,pconheat : int) returns (conheat : int); imported
;
]
ENDTARGOS
```

## A.2 Additional functional nodes written in Lustre

```
-- absolute value
node abs (a :int) returns (b:int);
let   b=if a<0 then -a else a;    tel



node  calcfan  (chamflow,pantvent,pconfan : int)
      returns  (confan : int);
var flowtime,secflow,volumedm : int;
let
volumedm  = 10 -> volume*1000;
    secflow   = chamflow div 60;
    flowtime  = 0 -> volumedm div secflow;
    confan    = if flowtime<pantvent
                then pconfan-10
                else if flowtime>pantvent
                     then pconfan+10
                     else pconfan;
tel



node calcheat  (chamtact,pantreq,pconheat : int)
     returns   (conheat : int);
let
    conheat   = if chamtact<pantreq
                then pconheat+10
                else if chamtact>pantreq
                     then 0
                     else pconheat;
tel
```

## A.3 Integration in the global Lustre program

We simply replace the node `controller_u` of the Lustre program (see annex page 11) by the following, where `SAAB` is the Lustre node produced from the mode-automaton.

```
node controller_u (pancon: [int,int]; chamcon: [int,int]; cononoff: bool)
returns (outonoff: bool; concham: [int,int]; mode: MODE);
var pantreq : int; -- 0 .. 1000 (temp. requested by panel)
    pantvent : int; -- 0..32000 (time  requested by panel)
    chamtact: int;
    conheat:int;
    chamflow: int;
    confan: int;
start,wait,sol,work,block : bool;
marche: bool;
time : int;
treq1,time1,tempd :int;
starttime,btime : int;
```

```
let
   [pantreq, pantvent] = pancon;
   [chamtact,chamflow] = chamcon;
   concham = [conheat,confan];
   tempd = 0->(chamtact-pre(chamtact->
               pre(chamtact->pre(chamtact->
               pre(chamtact->pre(chamtact))))));

   (marche,treq1,time1,confan,conheat,starttime,
    btime,start,wait,sol,work,block)                     =

   SAAB (pantreq,pantvent,chamtact,chamflow,
         deltatemp,tgrad,solution_time,tblock,tempd);

   mode=if start then Start
else if wait then Wait
else if sol then Sol
  else if work then Work
   else Block;

   (outonoff,time) = clock(cononoff);
tel
```

# B The Lustre program obtained from ASA+

```
#line 46 "body.nw"
type MODE=int;
-- type MODE;
#line 51 "body.nw"
const
  Wait=0; Sol=1;  Work=2; Block=3;
  -- Wait, Sol, Work, Block: MODE;
#line 67 "body.nw"
  volume = 20 ;
#line 72 "body.nw"
  envtemp = 200 ;
#line 78 "body.nw"
  treqmax = 300 ;
  treqmin = 100 ;
#line 85 "body.nw"
  tgrad = 50 ;
#line 92 "body.nw"
  solution_time = 600 ;
  tventmax = 30;
  tventmin = 5;
  tblock = 600 ;
#line 101 "body.nw"
  deltatemp = 50 ;
  x = 10 ;
#line 111 "body.nw"
node controller (on_off: bool; opinp: [int,int]; chamcon: [int,int])
```

```
returns (concham: [int,int]; chambonoff: bool;
          panop: [bool,[bool,bool,bool,bool],[int,int]]);
var
  cononoff, outonoff: bool;
  pancon: [int,int];
  mode: MODE;
let
  (cononoff,pancon) = panelin(on_off,opinp);
  (outonoff,concham,mode) = controller_u(pancon,chamcon,cononoff);
  (chambonoff,panop) = panelout(mode,pancon,chamcon,outonoff);
tel -- controller
#line 18 "body.nw"

#line 129 "body.nw"
node panelin (on_off: bool; opinp: [int,int])
returns (cononoff: bool; pancon: [int,int]);
var panonoff: bool;
    pantreq : int; -- 0 .. 1000 (temp. requested by panel)
    optreq  : int; -- 0 .. 1000 (temp. requested by operator)
    pantvent : int; -- 0..32000 (time requested by panel)
    optvent : int; -- 0 ..50 (time requested by operator)
let
   panonoff = powerswitch(on_off);
        (cononoff, pantreq) =  treqknob(optreq, panonoff);
   pantvent = tventknob(optvent, panonoff);
   [optreq,optvent] = opinp;
   pancon = [pantreq, pantvent];
tel -- panelin
#line 18 "body.nw"

#line 150 "body.nw"
node powerswitch (on_off: bool)
returns (panonoff: bool);
let
   panonoff = on_off;
tel -- powerswitch
#line 160 "body.nw"
node treqknob (optreq: int; panonoff: bool)
returns (cononoff: bool; pantreq: int);
let
  pantreq = 200 ->
     if panonoff then
         if optreq>=treqmin  and optreq<=treqmax then optreq*10
         else if optreq<treqmin then treqmin*10
         else treqmax*10
     else pre(pantreq);
  cononoff = panonoff;
tel -- treqknob
#line 19 "body.nw"

#line 176 "body.nw"
node tventknob (optvent: int; panonoff: bool)
```

```
returns (pantvent: int);
let
   pantvent = 3000 ->
      if panonoff then
         if optvent>=tventmin and optvent<=tventmax then optvent*60
         else if optvent < tventmin then tventmin*60
         else tventmax*60
      else pre(pantvent);
tel -- tventknob
#line 19 "body.nw"

#line 191 "body.nw"
node controller_u (pancon: [int,int]; chamcon: [int,int]; cononoff: bool)
returns (outonoff: bool; concham: [int,int]; mode: MODE);
var pantreq : int; -- 0 .. 1000 (temp. requested by panel)
    pantvent : int; -- 0..32000 (time requested by panel)
    cononoff1 : bool;
    time : int;
    chamtact: int;
    cononoff2: bool;
    conheat:int;
    cononoff3: bool;
    --envtemp: int;
    chamflow: int;
    confan: int;
let
   [pantreq, pantvent] = pancon;
   [chamtact,chamflow] = chamcon;
   concham = [conheat,confan];
   (cononoff1,time) = clock (cononoff);
   (cononoff2,mode) = modeselect(time,pantreq,chamtact,solution_time,
                                 tgrad,deltatemp,cononoff1);
   (conheat,cononoff3) = tempreg(mode,pantreq,envtemp,
                                 chamtact,cononoff2);
   (confan,outonoff) = flowreg(mode,time,pantvent,chamflow,cononoff3);
tel -- controller_u
#line 222 "body.nw"
node clock (cononoff: bool)
returns (cononoff1: bool; time: int)
var n: int;
let
   n = 0 ->
      if cononoff then
         if pre(n)=10 then 0 else pre(n)+1
      else pre(n);
   time = 0 ->
      if cononoff then
         if pre(n)=10 then pre(time)+1 else pre(time)
      else pre(time);
   cononoff1 = false-> cononoff and pre(n)<>10;
tel -- clock
#line 20 "body.nw"
```

```
#line 242 "body.nw"
node modeselect (time: int; pantreq: int; chamtact: int;
  solution_time: int; tgrad: int; deltatemp: int; cononoff1: bool)
returns (cononoff2: bool; mode: MODE);
var
    temp : int^6;
    t_req1, time1, tempd : int;
    -- STATES
    st_start, st_wait, st_sol, st_work, st_block: bool;
    -- TRANSITIONS
    start_start, start_wait, wait_wait, wait_sol,
    sol_sol, sol_wait, sol_work, sol_block,
    work_work, work_wait, work_block, block_start: bool;
    trans: bool;
let
   mode = Wait ->
      if start_wait or wait_wait or sol_wait or work_wait  then Wait
      else if wait_sol or sol_sol then Sol
      else if sol_work or work_work then Work
      else if sol_block or work_block then Block
      else pre(mode);
   cononoff2 = false -> cononoff1;
   (temp[0..4],temp[5]) = (200^5,200) ->
      if start_wait then (chamtact^5,chamtact)
      else if  wait_wait or wait_sol or sol_wait or sol_sol or sol_work
             or work_wait or work_work
      then (pre(temp[1..5]) , chamtact)
      else (pre(temp[0..4]),pre(temp[5]));
   t_req1 = 200 ->
      if  start_wait or wait_wait or wait_sol or work_wait then pantreq
      else pre(t_req1);
   time1 = 1 -> if wait_wait or sol_wait or sol_work or work_wait then 1
                else if wait_sol or sol_sol then pre(time1) + 1
                else pre(time1);
   tempd =  0 -> if wait_sol or sol_sol or sol_work or work_work
                 then temp[5] - temp[0]
                 else pre(tempd);
  -- THE AUTOMATON
    st_start = true -> pre(start_start or block_start)
                       or pre(st_start and not trans);
    st_wait = false ->
             pre(start_wait or wait_wait or sol_wait or work_wait)
             or pre(st_wait and not trans);
    st_sol = false -> pre(wait_sol or sol_sol)
             or pre(st_sol and not trans);
    st_work = false -> pre(sol_work or work_work)
             or pre(st_work and not trans);
    st_block = false -> pre(sol_block or work_block)
             or pre(st_block and not trans);
    trans = start_start or start_wait or wait_wait or wait_sol or
           sol_wait or sol_sol or sol_work or sol_block or work_wait or
```

```
           work_work or work_block or block_start;
    start_start = cononoff1 and st_start and
                    (false->pre(EQ_MODE(mode,Block)));
    start_wait = cononoff1 and st_start and
                    (true->pre(not EQ_MODE(mode,Block)));
    wait_wait = cononoff1 and st_wait and
               ( (pantreq-chamtact>2*deltatemp)
                 or (chamtact-pantreq > 2*deltatemp));
    wait_sol = cononoff1 and st_wait and
               (pantreq-chamtact<=2*deltatemp) and
               (chamtact-pantreq<=2*deltatemp);
    sol_wait = cononoff1 and st_sol and (pre(t_req1)<>pantreq) and
         ((pantreq>chamtact+deltatemp) or (pantreq<chamtact-deltatemp));
    sol_sol = cononoff1 and st_sol and
           (pantreq-chamtact<=2*deltatemp) and
           (chamtact-pantreq<=2*deltatemp) and
           (pre(tempd) div 5 < tgrad) and
           (pre(time1) div 10 < solution_time) ;
    sol_work = cononoff1 and st_sol and
           (pantreq-chamtact<=2*deltatemp) and
           (chamtact-pantreq<=2*deltatemp) and
           (pre(tempd) div 5 < tgrad) and
           (pre(time1) div 10 >= solution_time) ;
    sol_block = cononoff1 and st_sol and
                 not(sol_wait or sol_sol or sol_work);
    work_wait = cononoff1 and st_work and (pre(t_req1)<>pantreq) and
          ((pantreq>chamtact+deltatemp) or (pantreq<chamtact-deltatemp));
    work_work = cononoff1 and st_work and
          (pantreq-chamtact<=2*deltatemp) and
          (chamtact-pantreq<=2*deltatemp) and
          (pre(tempd) div 5 < tgrad);
    work_block = cononoff1 and st_work and not(work_wait or work_work);
    block_start = cononoff1 and st_block ;
tel -- modeselect
#line 20 "body.nw"

#line 331 "body.nw"
node tempreg (
   mode: MODE; pantreq : int; envtemp: int; chamtact: int;
   cononoff2: bool)
returns (conheat:int ; cononoff3: bool);
var st_off, st_regulate, st_hold: bool;
    --TRANSITIONS
    off_off, off_reg, off_hold,
    reg_off1, reg_off2, reg_reg, reg_hold,
    hold_off1, hold_off_2, hold_reg, hold_hold : bool;
    trans: bool;
let
  cononoff3 = false -> cononoff2;
  conheat = 0 ->
       if off_reg or reg_reg or hold_reg then pre(conheat)+10
       else if off_off or reg_off1 or reg_off2 or
```

```
              hold_off1 or hold_off_2 then 0
       else pre(conheat);
  -- AUTOMATON
  st_off = true ->
      pre(off_off or reg_off1 or reg_off2 or hold_off1 or hold_off_2)
      or pre(st_off and not trans);
  st_regulate = false -> pre(off_reg or reg_reg or hold_reg)
              or pre(st_regulate and not trans);
  st_hold = false -> pre(off_hold or reg_hold or hold_hold)
              or pre(st_hold and not trans);
  off_reg = cononoff2 and st_off and
           (EQ_MODE(mode,Work) or EQ_MODE(mode,Wait))
           and chamtact<pantreq;
  off_hold = cononoff2 and st_off and
       (EQ_MODE(mode,Work) or EQ_MODE(mode,Wait) or EQ_MODE(mode,Sol))
       and (chamtact=pantreq);
  off_off = cononoff2 and st_off and
                        not(off_hold or off_reg);
  reg_off1 = cononoff2 and st_regulate and EQ_MODE(mode,Block);
  reg_reg = cononoff2 and st_regulate and
       (EQ_MODE(mode,Work) or EQ_MODE(mode,Wait) or EQ_MODE(mode,Sol))
       and chamtact<pantreq;
  reg_hold = cononoff2 and st_regulate and
       (EQ_MODE(mode,Work) or EQ_MODE(mode,Wait) or EQ_MODE(mode,Sol))
       and chamtact=pantreq;
  reg_off2 = cononoff2 and st_regulate and
       not (reg_off1 or reg_reg or reg_hold);
  hold_off1 = cononoff2 and st_hold and EQ_MODE(mode,Block);
  hold_reg = cononoff2 and st_hold and
       (EQ_MODE(mode,Work) or EQ_MODE(mode,Wait) or EQ_MODE(mode,Sol))
       and chamtact<pantreq;
  hold_hold = cononoff2 and st_hold and
       (EQ_MODE(mode,Work) or EQ_MODE(mode,Wait) or EQ_MODE(mode,Sol))
       and chamtact=pantreq;
  hold_off_2 = cononoff2 and st_hold and
        not (hold_off1 or hold_reg or hold_hold);

  trans = off_off or off_reg or off_hold or
    reg_off1 or reg_off2 or reg_reg or reg_hold or
    hold_off1 or hold_off_2 or hold_reg or hold_hold;
tel -- tempreg
#line 392 "body.nw"
node flowreg (mode: MODE; time: int; pantvent: int;
   chamflow: int; cononoff3: bool)
returns (confan: int; outonoff: bool);
var volumedm : int;
    btime : int;
    starttime : int;
    flowtime : int;
    secflow : int;
    --STATES
    st_off, st_regulate, st_hold: bool;
```

```
  --TRANSITIONS
  off_off, off_reg,
  reg_off, reg_reg1, reg_reg2, reg_hold1, reg_hold2,
  hold_off, hold_reg1, hold_reg2, hold_hold : bool;
  trans: bool;

let
  outonoff = cononoff3;
  confan = 0 -> if off_off or reg_off or hold_off then 0
                else if off_reg then 100
                else if reg_reg1 or reg_reg2 then
                  if pre(flowtime) > pantvent then pre(confan)+10
                  else if pre(flowtime) < pantvent then pre(confan)-10
                  else pre(confan)
                else pre(confan);
  volumedm = 10 -> volume * 1000;
  secflow = 0 -> if trans then chamflow div 60 else pre(secflow);
  flowtime = 0 -> if trans then
                  current((volumedm when trans) div (secflow when trans))
                  else pre(flowtime);
  --flowtime = 0 -> if trans then volumedm else pre(flowtime);
  btime = 0 -> if off_off or reg_off or hold_off then 0
               else if reg_reg1 or reg_hold1 or hold_reg1
               then starttime div 60
               else pre(btime);
  starttime = 0 -> if off_off or reg_off or hold_off then 0
                   else if reg_reg1 or reg_hold1 or hold_reg1
                   then pre(starttime)+1
                   else pre(starttime);
  -- AUTOMATON
  st_off = true ->
      pre(off_off or reg_off or hold_off)
      or pre(st_off and not trans);
  st_regulate = false -> pre(off_reg or reg_reg1 or reg_reg2
               or hold_reg1 or hold_reg2)
              or pre(st_regulate and not trans);
  st_hold = false -> pre(reg_hold1 or reg_hold2 or hold_hold)
              or pre(st_hold and not trans);
  off_off = cononoff3 and st_off and
       EQ_MODE(mode,Block);
  off_reg = cononoff3 and st_off and not off_off;
  reg_off = cononoff3 and st_regulate and
       EQ_MODE(mode,Block) and pre(btime) > tblock;
  reg_reg1 = cononoff3 and st_regulate and
       EQ_MODE(mode,Block) and pre(btime) <= tblock and
       pre(flowtime) <> pantvent;
  reg_hold1 = cononoff3 and st_regulate and
       EQ_MODE(mode,Block) and pre(btime) <= tblock and
       pre(flowtime) = pantvent;
  reg_reg2 = cononoff3 and st_regulate and
       (EQ_MODE(mode,Work) or EQ_MODE(mode,Wait) or EQ_MODE(mode,Sol))
       and pre(flowtime) <> pantvent;
```

```
  reg_hold2 = cononoff3 and st_regulate and
       (EQ_MODE(mode,Work) or EQ_MODE(mode,Wait) or EQ_MODE(mode,Sol))
       and pre(flowtime) = pantvent;
  hold_hold = cononoff3 and st_hold and
       (EQ_MODE(mode,Work) or EQ_MODE(mode,Wait) or EQ_MODE(mode,Sol))
       and pre(flowtime) = pantvent;
  hold_off = cononoff3 and st_hold and
       EQ_MODE(mode,Block) and pre(btime) > tblock;
  hold_reg1 = cononoff3 and st_hold and
       EQ_MODE(mode,Block) and pre(btime) <= tblock;
  hold_reg2 = cononoff3 and st_hold and
       (EQ_MODE(mode,Work) or EQ_MODE(mode,Wait) or EQ_MODE(mode,Sol))
       and pre(flowtime) <> pantvent;
  trans = off_off or off_reg or
    reg_off or reg_reg1 or reg_reg2 or reg_hold1 or reg_hold2 or
    hold_off or hold_reg1 or hold_reg2 or hold_hold;
tel -- flowreg
#line 21 "body.nw"

#line 476 "body.nw"
node panelout
  (mode: MODE; pancon: [int,int]; chamcon: [int,int]; outonoff: bool)
returns
  (chambonoff: bool; panop: [bool,[bool,bool,bool,bool],[int,int]]);
var
  pantreq: int;
  chamtact: int;
  panaudio: bool;
  panlamps: [bool,bool,bool,bool];
  pangauges: [int,int];
  outonoff1, outonoff2: bool;
let
  (panaudio,outonoff1) = audioprep(chamtact,mode,pantreq,outonoff);
  (panlamps,outonoff2) =
         lampprep(panaudio,chamcon,mode,pancon,outonoff1);
  (chambonoff,pangauges) = gaugeprep(chamcon,outonoff2);
  chamtact = chamcon[0];
  pantreq  = pancon[0];
  panop = [panaudio, panlamps, pangauges];
tel -- panelout

-- |------------------------|
-- | Nodes used in PANELOUT |
-- |------------------------|

node audioprep
  (chamtact: int; mode: MODE; pantreq: int; outonoff: bool)
returns (panaudio: bool; outonoff1: bool);
let
  outonoff1 = outonoff;
  panaudio = false ->
    if outonoff then
```

```
      ((chamtact-pantreq > deltatemp) or (pantreq-chamtact > deltatemp))
      and (mode=Work)
    else pre(panaudio);
tel -- audioprep

node lampprep
  (panaudio: bool; chamcon: [int,int]; mode: MODE;
   pancon: [int,int]; outonoff1: bool)
returns
  ( panlamps: [bool,bool,bool,bool]; outonoff2: bool);
var
   panwait: bool; panblock : bool; panoper: bool;
   panwarn: bool;
let
   (outonoff2,panwait,panblock,panoper) = modelamps(mode,outonoff1);
   panwarn = warninglamp(mode,chamcon,pancon,panaudio,outonoff1);
   panlamps = [panoper,panblock,panwait,panwarn];
tel -- lampprep

---------------------------
--  Nodes used in LAMPREP
---------------------------

  node modelamps
   (mode: MODE; outonoff1: bool)
  returns
    (outonoff2: bool; panwait: bool; panblock : bool; panoper: bool);
  let
    outonoff2 = outonoff1;
    (panwait, panoper, panblock)= (true, false, false) ->
       if outonoff1 then
          if mode=Wait then (true, false, false)
          else if mode=Work or mode=Sol then (false, true, false)
          else -- mode=Block
            (false,false,true)
       else pre(panwait, panoper, panblock);
  tel -- modelamps

  node warninglamp
    (mode: MODE; chamcon: [int,int] ; pancon: [int,int];
     panaudio: bool; outonoff1: bool)
  returns (panwarn: bool);
  var outonoff11, outonoff12: bool;
      chamtact: int;
      pantreq: int;
      warntemp: bool;
      chamflow: int;
      pantvent: int;
      warnvent: bool;
  let
    [chamtact,chamflow] = chamcon;
    [pantreq, pantvent] = pancon;
```

```
    (outonoff11,warntemp) =
        tempwarn(mode,chamtact,pantreq,deltatemp,outonoff1);
    (outonoff12,warnvent) = ventwarn(chamflow,pantvent,x,outonoff11);
     panwarn = execwarn(warnvent,warntemp,mode,panaudio,outonoff12);
  tel -- warninglamp

  -- / Nodes used in WARNINGLAMP

    node tempwarn
      (mode: MODE; chamtact: int; pantreq: int; deltatemp: int;
        outonoff1: bool)
    returns
      (outonoff11: bool; warntemp: bool);
    let
      outonoff11 = outonoff1;
      warntemp = false ->
        if outonoff1 then
          if (pantreq-chamtact>deltatemp or chamtact-pantreq>deltatemp)
             and EQ_MODE(mode,Work) then true
          else if (pantreq-chamtact>2*deltatemp or
             chamtact-pantreq>2*deltatemp) and
             EQ_MODE(mode,Sol) then true
          else if EQ_MODE(mode,Block) then true
          else false
        else pre(warntemp);
    tel -- tempwarn

    node ventwarn
      (chamflow: int; pantvent: int; x: int; outonoff11: bool)
    returns (outonoff12: bool; warnvent: bool);
    let
      outonoff12 = outonoff11;
      warnvent = false ->
          if outonoff11 then
             (((1000*volume) div chamflow) >
                 (((100+x)*(pantvent div 6)) div 100))
          else pre(warnvent);
    tel -- ventwarn

    node execwarn
      (warnvent: bool; warntemp: bool; mode: MODE;
       panaudio: bool; outonoff12: bool)
    returns
      (panwarn: bool);
    let
      panwarn = false ->
        if outonoff12 then
           (warnvent or warntemp or panaudio or (mode=Block))
        else pre(panwarn);
    tel -- execwarn

node gaugeprep (chamcon: [int,int]; outonoff2: bool)
```

```
returns (chambonoff: bool; pangauges: [int,int]);
var chamflow, chamtact: int;
    pantvent, pantact: int;
let
  [chamtact,chamflow] = chamcon;
  pangauges = [pantact,pantvent];
  pantact = tactprep(chamtact,outonoff2);
  (chambonoff,pantvent) = tventprep(chamflow,outonoff2);
tel -- gaugeprep

-- * Nodes used in GAUGEPREP

  node tventprep
    (chamflow: int; outonoff2: bool)
  returns (chambonoff: bool; pantvent: int);
  let
    pantvent = (volume+10) div (chamflow div 100);
    chambonoff = outonoff2;
  tel -- tventprep

  node tactprep (chamtact: int; outonoff2: bool)
  returns (pantact: int)
  let
    chamtact = pantact;
  tel -- tactprep

#line 21 "body.nw"




#line 57 "body.nw"
node EQ_MODE(m1,m2: MODE) returns (eq: bool);
let eq = (m1=m2) ; tel
-- function EQ_MODE(m1,m2: MODE) returns (eq: bool);
```