Annexe : codes SMV des exemples du rapport

1 Code de ex1.smv

```
#define MAX 3 /*nombre d'étages*/

module main()
{

/* déclaration des variables */

  b_appel : array 0 .. MAX of boolean;
  l_appel : array 0 .. MAX of boolean;
  cabine  : 0 .. MAX;
  dir : {haut,bas,arret};

/* initialisation des variables */

  init(l_appel) := [0 : i = 0 .. MAX];
  init(dir) := {haut,bas,arret};
  init(cabine) := 0;

/* code de l'ascenseur */

  for(i = 0; i <= MAX; i = i+1){
    next(l_appel[i]) := (l_appel[i]|next(b_appel[i])) & !(next(cabine)=i);
  }

  default {
        next(dir) := arret;
  } in default {
  chain(i = 0; i <= MAX; i = i+1)
    if((! dir=arret) & l_appel[i])
next(dir) := dir;
  } in default {
  chain(i = 0; i <= MAX; i = i+1)
    if(dir=arret & l_appel[i])
next(dir) := {bas,haut};
  } in default {
  chain(i = 0; i <= MAX; i = i+1)
    if(dir=bas & cabine=0 & l_appel[i])
next(dir) := haut;
  } in default {
  chain(i = 0; i <= MAX; i = i+1)
    if(dir=haut & cabine=MAX & l_appel[i])
next(dir) := bas;
  } in {
  chain(i = 0; i <= MAX; i = i+1)
    if(dir=arret & l_appel[i])
case{
   cabine>=i : next(dir) := bas;
   cabine<=i : next(dir) := haut;
        }
```

```
  }

  case{
dir = haut & cabine < MAX : next(cabine) := cabine + 1;
dir = bas  & cabine > 0   : next(cabine) := cabine - 1;
1                         : next(cabine) := cabine;
  }

/* propriétés à vérifier */

   toto :  assert G (X 1) ; /*non blocage */

   for(i = 0; i <= MAX; i = i+1){

/* quand la cabine est là, les voyants sont éteints */
     ok1[i] : assert G ( (cabine=i) -> (!l_appel[i]) );

/* quand un voyant est allumé, il le reste jusqu'à ce que la cabine soit là */
     ok2[i] : assert G ( l_appel[i] -> ( (G l_appel[i]) | (l_appel[i] U cabine=i) ) );

/* quand un voyant s'éteint, la cabine vient d'arriver */
     ok3[i] : assert G ( (l_appel[i] & X !l_appel[i]) -> (!cabine=i & X cabine=i) );

/* quand un voyant est allumé, il le reste jusqu'à ce que la cabine arrive finalement */
     str_ok[i] : assert G ( l_appel[i] -> (l_appel[i] U cabine=i) );

  }
}
```

2 Code de asc-essai-a.smv

```
/*cabine oscillante */

#define MAX 3

module main()
{

/* implementation M1 */

      cabine : 0..MAX ;
      boutons : array 0..MAX of boolean ;
      lumieres : array 0..MAX of boolean ;
      dir : {haut,bas,arret};

      init(cabine):= 0;
      init(lumieres):= [0 : i = 0 .. MAX];
      init(dir) := {haut,bas};


      for(i = 0; i <= MAX; i = i+1){
          next(lumieres[i]) := (lumieres[i]|next(boutons[i])) & !(next(cabine)=i);}
```

```
       case{
  dir = haut & cabine < MAX : next(cabine) := cabine + 1;
          dir = bas  & cabine > 0   : next(cabine) := cabine - 1;
  1                         : next(cabine) := cabine;
        }

       case{
cabine = MAX : next(dir) := bas;
cabine = 0   : next(dir) := haut;
1      : next(dir) := dir;
  }

/* le layer M2 */

       layer M2 : {

             cabine := {0..MAX} ;

             for (i=0 ; i<= MAX ; i=i+1)
                  case{
               cabine=i : lumieres[i] := 0;
               !(cabine=i) : lumieres[i] := {0,1};
                       }
        }
```

3 Code de asc-essai-b.smv

```
/*cabine oscillante */

#define MAX 3

module main()
{
      cabine : 0..MAX ;
      boutons : array 0..MAX of boolean ;
      lumieres : array 0..MAX of boolean ;
      dir : {haut,bas,arret};


      init(cabine):= 0;
      init(lumieres):= [0 : i = 0 .. MAX];
      init(dir) := {haut,bas,arret};


      for(i = 0; i <= MAX; i = i+1){
          next(lumieres[i]) := (lumieres[i]|next(boutons[i])) & !(next(cabine)=i);}


       case{
  dir = haut & cabine < MAX : next(cabine) := cabine + 1;
          dir = bas  & cabine > 0   : next(cabine) := cabine - 1;
  1                         : next(cabine) := cabine;
        }
```

```
       case{
cabine = MAX : next(dir) := bas;
cabine = 0   : next(dir) := haut;
1      : next(dir) := dir;
        }


       layer M2 : {

             cabine := {0..MAX} ;

             for (i=0 ; i<= MAX ; i=i+1)
                  case{
               cabine=i : lumieres[i] := 0;
               !(cabine=i) : lumieres[i] := {0,1};
                       }
        }

      /* proprietes a verifier */

      for(i = 0; i <= MAX; i = i+1){

          ok1[i] : assert G ( (cabine=i) -> (!lumieres[i]) );

          ok2[i] : assert G ( lumieres[i] -> ( (G lumieres[i]) | (lumieres[i] U cabine=i) ) );

          str_ok[i] : assert G ( lumieres[i] -> (lumieres[i] U cabine=i) );

}

}
```

4 Code de asc-essai-c.smv

```
/*cabine oscillante */

#define MAX 3

module main()
{
      cabine : 0..MAX ;
      boutons : array 0..MAX of boolean ;
      lumieres : array 0..MAX of boolean ;
      dir : {haut,bas,arret};


      init(cabine):= 0;
      init(lumieres):= [0 : i = 0 .. MAX];
      init(dir) := {haut,bas,arret};


      for(i = 0; i <= MAX; i = i+1){
```

```
          next(lumieres[i]) := (lumieres[i]|next(boutons[i])) & !(next(cabine)=i);}


       case{
  dir = haut & cabine < MAX : next(cabine) := cabine + 1;
          dir = bas  & cabine > 0   : next(cabine) := cabine - 1;
  1                         : next(cabine) := cabine;
        }

       case{
cabine = MAX : next(dir) := bas;
cabine = 0   : next(dir) := haut;
1      : next(dir) := dir;
       }

       layer M2a : {

                 case{
             cabine = MAX : next(dir) := bas;
             cabine = 0   : next(dir) := haut;
             1            : next(dir) := dir;
                     }

             for (i=0 ; i<= MAX ; i=i+1)
                 case{
              cabine=i : lumieres[i] := 0;
              !(cabine=i) : lumieres[i] := {0,1};
                      }
         }


       layer M2b : {

             cabine := {0..MAX} ;

             for (i=0 ; i<= MAX ; i=i+1)
                  case{
               cabine=i : lumieres[i] := 0;
               !(cabine=i) : lumieres[i] := {0,1};
                       }
        }

   M2a refines M2b ;

   /* proprietes a verifier */

   toto : assert G (cabine <= MAX) ;

   using lumieres//M2a prove ok1[3];

   for(i = 0; i <= MAX; i = i+1){

     ok1[i] : assert G ( (cabine=i) -> (!lumieres[i]) );
```

```
     ok2a[i] : assert G ( lumieres[i] -> ( (G lumieres[i]) | (lumieres[i] U cabine=i) ) );

     str_ok2a[i] : assert G ( lumieres[i] -> (lumieres[i] U cabine=i) );

}

}
```

5 Code de time1.smv

```
#define MAX 3

module main()
{

    l_appel, b_appel : array 0..MAX of boolean ;
    pos1, pos2 : 0 .. MAX;
    dir1, dir2 : {haut,bas,arret} ;

   init(l_appel):= [0 : i = 0 .. MAX] ;
   init(dir1) := {haut,bas,arret};
   init(pos1) := 0 ;
   init(dir2) := {haut,bas,arret};
   init(pos2) := 0 ;


    for(i = 0; i <= MAX; i = i+1){
          next(l_appel[i]) := (l_appel[i]|next(b_appel[i])) & !(next(pos1)=i) &!(next(pos2)=i) ;}


   default {
     next(dir1):= arret;
     next(dir2):= arret;
}
   in {
      chain(i = 0; i <= MAX; i = i+1)
      {

        aux1, aux2 : -MAX..MAX ;

        case{
           (dir1=arret) & (dir2=arret) & (l_appel[i]) :
{
        aux1 := pos1 - i ;
        aux2 := pos2 - i ;

  if  ((aux1 > 0) & (aux2 > 0) & (aux1<=aux2)) {next(dir1):=bas;}
  else if ((aux1 > 0) & (aux2 > 0) & (aux1>aux2)) {next(dir2):=bas;}
  else if ((aux1 > 0) & (aux2 < 0) & (aux1<=-aux2)) { next(dir1):=bas;}
  else if ((aux1 > 0) & (aux2 < 0) & (aux1>-aux2)) {next(dir2):=haut;}
  else if ((aux1 < 0) & (aux2 > 0) & (-aux1<=aux2)) {next(dir1):=haut;}
  else if ((aux1 < 0) & (aux2 > 0) & (-aux1>aux2)) {next(dir2):=bas;}
```

```
  else if ((aux1 < 0) & (aux2 < 0) & (-aux1<=-aux2)) {next(dir1):=haut;}
  else if ((aux1 < 0) & (aux2 < 0) & (-aux1>-aux2)) {next(dir2):=bas;}

}

           !(dir1=arret)   & (dir2=arret) & (l_appel[i]) :
{
        aux1 := pos1 - i ;
        aux2 := pos2 - i ;

  next(dir1):=dir1 ;
  if (aux2=0) {next(dir2):= arret ;}
  else if (aux2>0) {next(dir2):= bas ;}
  else {next(dir2):= haut ;}
}



           (dir1=arret) & !(dir2 = arret )  & (l_appel[i])  :
{
        aux1 := pos1 - i ;
        aux2 := pos2 - i ;

  next(dir2):=dir2 ;
  if (aux1=0) {next(dir1):= arret ;}
  else if (aux1>0) {next(dir1):= bas ;}
  else  {next(dir1):= haut ;}
}



                  l_appel[i]                        :
{
   if ((pos1 = MAX) & !(dir1=arret)) next(dir1):= bas;
   else if ((pos1 = 0) & !(dir1=arret)) next(dir1):= haut;
   if ((pos2 = MAX) & !(dir2=arret)) next(dir2):=bas ;
   else if ((pos2 = 0) & !(dir2=arret)) next(dir2):=haut ;

}

}
}
}

    case{
dir1 = haut & pos1 < MAX : next(pos1) := pos1 + 1;
dir1 = bas  & pos1 > 0   : next(pos1) := pos1 - 1;
1                        : next(pos1) := pos1;
    }
    case{
dir2 = haut & pos2 < MAX : next(pos2) := pos2 + 1;
dir2 = bas  & pos2 > 0   : next(pos2) := pos2 - 1;
1                        : next(pos2) := pos2;
```

```
    }


/*properties*/

    for(i = 0; i <= MAX; i = i+1){

     ok1[i] : assert G ( (pos1=i) -> (!l_appel[i]) );
     ok2[i] : assert G ( (pos2=i) -> (!l_appel[i]) );

     using l_appel//spec_cab1 prove ok1;
     using l_appel//spec_cab2 prove ok2;


     ok3[i] : assert G ( l_appel[i] -> ( (G l_appel[i]) | (l_appel[i] U ((pos1=i) | (pos2=i)))));
     ok5a[i] : assert G ((l_appel[i] & X !l_appel[i]) -> (!pos1=i & X pos1=i)|(!pos2=i & X pos2=i));

      for(j = 0; j <= MAX; j = j+1){
        hyp_cont1[i][j] :
  assert G( (pos1 = i & X pos1 = j) -> ((i - j < 2) & (j - i < 2)));
    }

      hyp_fair[i] : assert G (l_appel[i] -> F ((pos2=i)|(pos1=i))) ;


      mouv[0]   : assert G ((pos1=0   & X !dir1=arret) -> X X ! pos1=0 );
      mouv[MAX] : assert G ((pos1=MAX & X !dir1=arret) -> X X ! pos1=MAX);
}

    for (i = 1; i < MAX; i = i+1){
      mouv[i] : assert G ((pos1=i & !dir1=arret) -> X ! pos1=i); ;
}

    for (i = 0; i <= MAX; i = i+1){
        hyp_final_no_req[i] : assert F G (!l_appel[i]);
        assume hyp_final_no_req[i];
   }
    hyp_final_stop : assert F G (dir1=arret & dir2=arret);

    using hyp_final_no_req prove hyp_final_stop;

  dir_ok1 : assert G( (dir1=arret & pos1=1  & !l_appel[0]) -> ! X dir1=bas  );
  dir_ok2 : assert G( (dir1=arret & pos1=MAX-1 & !l_appel[MAX]) -> ! X dir1=haut );

  toto : assert G ((&(!(l_appel)))-> X dir1=arret);

}
```

6 Code de time2.smv

```
#define MAX 3

module main()
```

```
{

    l_appel, b_appel : array 0..MAX of boolean ;
    pos1, pos2 : 0 .. MAX;
    dir1, dir2 : {haut,bas,arret} ;

   init(l_appel):= [0 : i = 0 .. MAX] ;
   init(dir1) := {haut,bas,arret};
   init(pos1) := 0 ;
   init(dir2) := {haut,bas,arret};
   init(pos2) := 0 ;


    for(i = 0; i <= MAX; i = i+1){
          next(l_appel[i]) := (l_appel[i]|next(b_appel[i])) & !(next(pos1)=i) &!(next(pos2)=i) ;}


   default {
     next(dir1):= arret;
     next(dir2):= arret;
}
   in {
      chain(i = 0; i <= MAX; i = i+1)
      {

        aux1, aux2 : -MAX..MAX ;

        case{
           (dir1=arret) & (dir2=arret) & (l_appel[i]) :
{
        aux1 := pos1 - i ;
        aux2 := pos2 - i ;

  if  ((aux1 > 0) & (aux2 > 0) & (aux1<=aux2)) {next(dir1):=bas;}
  else if ((aux1 > 0) & (aux2 > 0) & (aux1>aux2)) {next(dir2):=bas;}
  else if ((aux1 > 0) & (aux2 < 0) & (aux1<=-aux2)) { next(dir1):=bas;}
  else if ((aux1 > 0) & (aux2 < 0) & (aux1>-aux2)) {next(dir2):=haut;}
  else if ((aux1 < 0) & (aux2 > 0) & (-aux1<=aux2)) {next(dir1):=haut;}
  else if ((aux1 < 0) & (aux2 > 0) & (-aux1>aux2)) {next(dir2):=bas;}
  else if ((aux1 < 0) & (aux2 < 0) & (-aux1<=-aux2)) {next(dir1):=haut;}
  else if ((aux1 < 0) & (aux2 < 0) & (-aux1>-aux2)) {next(dir2):=bas;}

}

           !(dir1=arret)   & (dir2=arret) & (l_appel[i]) :
{
        aux1 := pos1 - i ;
        aux2 := pos2 - i ;

  next(dir1):=dir1 ;
  if (aux2=0) {next(dir2):= arret ;}
  else if (aux2>0) {next(dir2):= bas ;}
  else {next(dir2):= haut ;}
```

```
}


           (dir1=arret) & !(dir2 = arret )  & (l_appel[i])  :
{
        aux1 := pos1 - i ;
        aux2 := pos2 - i ;

  next(dir2):=dir2 ;
  if (aux1=0) {next(dir1):= arret ;}
  else if (aux1>0) {next(dir1):= bas ;}
  else  {next(dir1):= haut ;}
}


                  l_appel[i]                        :
{
   if ((pos1 = MAX) & !(dir1=arret)) next(dir1):= bas;
   else if ((pos1 = 0) & !(dir1=arret)) next(dir1):= haut;
   if ((pos2 = MAX) & !(dir2=arret)) next(dir2):=bas ;
   else if ((pos2 = 0) & !(dir2=arret)) next(dir2):=haut ;

}

}
}
}

    case{
dir1 = haut & pos1 < MAX : next(pos1) := pos1 + 1;
dir1 = bas  & pos1 > 0   : next(pos1) := pos1 - 1;
1                        : next(pos1) := pos1;
    }
    case{
dir2 = haut & pos2 < MAX : next(pos2) := pos2 + 1;
dir2 = bas  & pos2 > 0   : next(pos2) := pos2 - 1;
1                        : next(pos2) := pos2;
    }


/* layer */
     layer spec_cab1 :
         {
              init(pos1) := 0;

              next(pos1) :=
                   case{
                       pos1 = 0   : {0,1} ;
                       pos1 = MAX : {MAX,MAX-1} ;
                       dir1 = haut : pos1 + 1;
                       dir1 = bas : pos1 - 1;
                       1 : pos1 ;
              } ;
```

```
              for (i=0 ; i<= MAX ; i=i+1) {
                   case{
                 pos1=i : l_appel[i] := 0;
                 !(pos1=i) : l_appel[i] := {0,1};
                        }
               }
         }

/*properties*/

    for(i = 0; i <= MAX; i = i+1){

     ok1[i] : assert G ( (pos1=i) -> (!l_appel[i]) );
     ok2[i] : assert G ( (pos2=i) -> (!l_appel[i]) );

     using l_appel//spec_cab1 prove ok1;
     using l_appel//spec_cab2 prove ok2;


     ok3[i] : assert G ( l_appel[i] -> ( (G l_appel[i]) | (l_appel[i] U ((pos1=i) | (pos2=i)))));
     ok5a[i] : assert G ((l_appel[i] & X !l_appel[i]) -> (!pos1=i & X pos1=i)|(!pos2=i & X pos2=i));

      for(j = 0; j <= MAX; j = j+1){
        hyp_cont1[i][j] :
  assert G( (pos1 = i & X pos1 = j) -> ((i - j < 2) & (j - i < 2)));
    }

      hyp_fair[i] : assert G (l_appel[i] -> F ((pos2=i)|(pos1=i))) ;

      mouv[0]   : assert G ((pos1=0   & X !dir1=arret) -> X X ! pos1=0 );
      mouv[MAX] : assert G ((pos1=MAX & X !dir1=arret) -> X X ! pos1=MAX);
}

    for (i = 1; i < MAX; i = i+1){
      mouv[i] : assert G ((pos1=i & !dir1=arret) -> X ! pos1=i); ;
}

    for (i = 0; i <= MAX; i = i+1){
        hyp_final_no_req[i] : assert F G (!l_appel[i]);
        assume hyp_final_no_req[i];
   }
    hyp_final_stop : assert F G (dir1=arret & dir2=arret);

    using hyp_final_no_req prove hyp_final_stop;

  dir_ok1 : assert G( (dir1=arret & pos1=1  & !l_appel[0]) -> ! X dir1=bas  );
  dir_ok2 : assert G( (dir1=arret & pos1=MAX-1 & !l_appel[MAX]) -> ! X dir1=haut );

  toto : assert G ((&(!(l_appel)))-> X dir1=arret);

}
```

7 Code de big-lift.smv

```
#define NB_LIFT 2
#define TOP 3
#define GROUND 0
#define up 1
#define dn 0


typedef FLOOR GROUND..TOP;
typedef DIRECTION {up,dn} ;


module main()
{
        moving : array 1..NB_LIFT of boolean; /* is a lift moving ? */
        floors : array 1..NB_LIFT of FLOOR ; /* at what floor is the lift ? */
        dirs : array 1..NB_LIFT of DIRECTION ; /* what's the direction of the lift ? */

        /* boolean tools */
        attracted_up, attracted_dn : array 1..NB_LIFT of boolean;
        can_continue_up, can_continue_dn : array 1..NB_LIFT of boolean;

        /* requirements of users */
        tab_in : array FLOOR of array 0..1 of boolean ; /* in_floor requirements */
        tab_out : array 1..NB_LIFT of array FLOOR of boolean ; /* in_lift requirements */



/* evolution of boolean tools */
       for(i=1 ; i<=NB_LIFT ; i=i+1)
          default {if ((floors[i]=GROUND)) {next(attracted_up[i]):=1;} else {next(attracted_up[i]):=0;}}
          in chain (j=GROUND ; j<=TOP ; j=j+1)
                 case{
                   (j>floors[i])& (tab_out[i][j]): next(attracted_up[i]):=1;
                   (j>floors[i])& (tab_in[j][1]): next(attracted_up[i]):=1;
                   (j>floors[i])& (tab_in[j][0]): next(attracted_up[i]):=1;
                 }
```

```
        for(i=1 ; i<=NB_LIFT ; i=i+1)
           default {if ((floors[i]=TOP)) {next(attracted_dn[i]):=1;} else {next(attracted_dn[i]):=0;}}
           in chain (j=GROUND ; j<=TOP ; j=j+1)
                  case{
                    (j<floors[i])& (tab_out[i][j]): next(attracted_dn[i]):=1;
                    (j<floors[i])& (tab_in[j][0]): next(attracted_dn[i]):=1;
                    (j<floors[i])& (tab_in[j][1]): next(attracted_dn[i]):=1;
                  }

        for(i=1 ; i<=NB_LIFT ; i=i+1)
        case{
        (!tab_out[i][floors[i]])&(!tab_in[floors[i]][dirs[i]])&(attracted_up[i])&(floors[i]<TOP) : next(can_continue_up[i]):=1 ;
        1 : next(can_continue_up[i]):=0 ;
        }


        for(i=1 ; i<=NB_LIFT ; i=i+1)
        case{
        (!tab_out[i][floors[i]])&(!tab_in[floors[i]][dirs[i]])&(attracted_dn[i])&(floors[i]>GROUND) : next(can_continue_dn[i]):=1 ;
        1 : next(can_continue_dn[i]):=0 ;
        }


/* lifts movements */
        chain(i=1 ; i<=NB_LIFT ; i=i+1)
           case{
              ((moving[i])&(dirs[i]=up)&(can_continue_up[i]))  :   if (!(floors[i]=TOP)) next(floors[i]):=floors[i]+1 ;
              ((moving[i])&(dirs[i]=dn)&(can_continue_dn[i]))  :   if (!(floors[i]=GROUND)) next(floors[i]):=floors[i]-1 ;


              ((moving[i])&(dirs[i]=up)&(!can_continue_up[i])) : { next(moving[i]):=0 ;
                                                                   next(tab_out[i][floors[i]]):=0 ;
                                                                   next(tab_in[floors[i]][dirs[i]]):= 0;}
              ((moving[i])&(dirs[i]=dn)&(!can_continue_dn[i])) : { next(moving[i]):=0 ;
                                                                   next(tab_out[i][floors[i]]):=0 ;
                                                                   next(tab_in[floors[i]][dirs[i]]):= 0;}
```

```
              ((!moving[i])&(dirs[i]=up)&(attracted_up[i]))    : { next(moving[i]):=1 ;
                                                                   if (!(floors[i]=TOP)) {next(floors[i]):=floors[i]+1 ;}}
              ((!moving[i])&(dirs[i]=dn)&(attracted_dn[i]))    : { next(moving[i]):=1 ;
                                                                   if (!(floors[i]=GROUND)) {next(floors[i]):=floors[i]-1 ;}}

/* changes of direction */

              ((!moving[i])&(dirs[i]=up)&(!attracted_up[i])&(attracted_dn[i])) : {next(tab_in[floors[i]][0]):=0 ; next(dirs[i]):=dn;}
              ((!moving[i])&(dirs[i]=dn)&(!attracted_dn[i])&(attracted_up[i])) : {next(tab_in[floors[i]][1]):=0 ; next(dirs[i]):=up;}

            }


/* assumptions*/


        floor_ground : assert G (tab_in[GROUND][0]=0) ;
        floor_top : assert G (tab_in[TOP][1]=0);
        assume floor_top, floor_ground;


        for(i=GROUND ; i<=TOP ;i=i+1)
          for(j=1 ; j<=NB_LIFT ; j=j+1){
              well_served1[i][j] : assert G (((floors[j]=i)&(!moving[j])&(dirs[j]=up)) -> tab_in[i][1] = 0);
              well_served2[i][j] : assert G (((floors[j]=i)&(!moving[j])&(dirs[j]=dn)) -> tab_in[i][0] = 0);
              assume well_served1[i][j];
              assume well_served2[i][j];
              }

        for(i=1 ; i<=NB_LIFT ; i=i+1)
          for(j=GROUND ; j<=TOP ; j=j+1){
              well_served3[i][j] : assert (G ((!moving[i]) & (floors[i]=j)) -> (tab_out[i][j])= 0);
              assume well_served3[i][j];
          }

/* properties */


        /* all requests from a floor are satisfied */
```

```
        for(i=GROUND ; i<=TOP ; i=i+1){
             reqf_ok [i] :assert G ((tab_in[i][0]) | (tab_in[i][1]))-> F (floors[1]=i |floors[2]=i) ;
         }

         using  floor_ground, floor_top, well_served1, well_served2 , well_served3 prove reqf_ok;

        /* all requests from a lift are satisfied */
        for(i=1 ; i<=NB_LIFT ; i=i+1){
           for (j=GROUND ; j<=TOP ; j=j+1)
              reql_ok[i][j] : assert G (tab_out[i][j] -> F (floors[i]=j));
        }

         using  floor_ground, floor_top, well_served1, well_served2 , well_served3 prove reql_ok;

        /* if there is no request, all lifts stop */
        for(i=1 ; i<=NB_LIFT ; i=i+1){
           for(j=GROUND ; j<=TOP ; j=j+1){
              hyp_no_reqf[i][j] : assert F G (tab_out[i][j]=0) ;
           }
        }
        for(i=1 ; i<=NB_LIFT ; i=i+1)
           hyp_no_reqf[i] : assert G ((tab_in[i][0]=0) & (tab_in[i][1]=0)) ;
        for(i=1 ; i<=NB_LIFT ; i=i+1) hyp_final_stop[i] : assert G F (!moving[i]) ;

        for(i=1 ; i<=NB_LIFT ; i=i+1) using hyp_final_no_req prove hyp_final_stop[i];


        layer req_int : {
            chain(i=1 ; i<=NB_LIFT ; i=i+1)
               next(tab_out[i][floors[i]]):={0,1} ;
        }

}
```