Master 1

Conception Formelle

Alain Griffault

LaBRI

Année 2009-2010

Outline

1. The ARC tool
2. First manipulation : basics of AltaRica
3. Second manipulation : validation with a model checker
4. Formal design of a lift

Outline

1. The ARC tool
2. First manipulation : basics of AltaRica
3. Second manipulation : validation with a model checker
4. Formal design of a lift

The ARC tool

ARC : The AltaRica Checker

- Infos at `http ://altarica.labri.fr/wiki/tools :arc`
- `You have to modify your environment (.bashr_export).`
- `PATH=$PATH :/net/autre/LABRI/griffaul/AltaRica_Tools/bi`
- `$ arc is a command interpreter.`

The ARC tool

ARC : few commands

- `arc>help` : is a very usefull command.
- `arc>load` : to read a model or a specification.
- `arc>list` : to display objects known by ARC.
- `arc>flatten` : to compute a node's semantic as a leaf.
- `arc>run` : to simulate an AltaRica node.
- `arc>sequences` : to generate scenarii.
- `arc>exit` : to quit the AltaRica checker.

The ARC tool

ARC : classical usage

- You have to describe your model in a file (.alt) and load it.
- You have to describe requirements in a file (.spe) and load it.
- You have to understand results.

Outline

1. The ARC tool
2. First manipulation : basics of ALTARICA
3. Second manipulation : validation with a model checker
4. Formal design of a lift

First manipulation : basics of AltaRica

Models

Refer to lesson's slides for the syntax.

- Minimal and FIFO nodes
- Electrical circuit (V1, V2 and corrections).
- Scheduler with and without priority.
- Courses with and without broadcast.

Test various commands such as :

- `$ help, load, list, ....`
- `$ run` to simulate a node.

First manipulation : basics of AltaRica

Requirements : first example

```
with nodename [, nodename]* do
  quot() > '$NODENAME.dot';
  show(all) > '$NODENAME.res';
done
```

- `$ more nodename.dot`
- `$ dot -Tpdf nodename.dot > nodename.pdf`

Outline

1. The ARC tool
2. First manipulation : basics of ALTARICA
3. Second manipulation : validation with a model checker
4. Formal design of a lift

Second manipulation : validation with a model checker

The AltaRica checker ARC

- ARC is a very powerfull model-checker for AltaRica.
- Users can choose to encode models as graphs or as BDD. The fisrt one permits that all properties can be computed in a linear time in the size of the graph, and the second one permits to deal with very big systems.
- To prove that $M \models P$, you have to compute counter examples for P.
  - `notP := any - P ;`
  - `notP := formula-describing-P-counter-examples ;`

and you have to check the result with `test(notP, 0)`.

Second manipulation : validation with a model checker

To do with ARC

You must validate all AltaRica nodes for all examples.

- Compute `deadlock` and `notSCC` properties.
- Check for properties and output results in files.
- For each properties witch is not satisfy, compute a counter example and output it in dot format.
- If the number of configurations is not so big, output in dot format the reachability graph.
- Output in files property's cardinals.

You may also compute properties depending of the system's type.

- Electrical circuit : no loop of reactions.
- Scheduler : the priority between pools of jobs is respected.
- Courses : 3 students can't write at the same time.

Outline

1. The ARC tool
2. First manipulation : basics of ALTARICA
3. Second manipulation : validation with a model checker
4. Formal design of a lift

Specifications

Informal description

The lift must be use in any building. Its design must no be dependant on the number of floor.

- At each floor, you may call the lift with a button.
- In the lift, there are as many buttons than floors.
- A lighting button means that this request is not yet satisfy.
- When the lift stops, doors open automatically.

At each time, a software controller chooses the next thing to do between : open a door, close a door, go up, go down or nothing.

Specifications

The owner of the building wants that these requirements have been proved.

Requirements

1. When a button is push, it lights.
2. When the corresponding service is done, it lights off.
3. At each floor, the door is close if the lift is not here.
4. Each request must be honored a day.
5. The software opens the door at some floor only if there is some requests for that floor.
6. If there is no request, the lift must stay at the same floor.
7. When the lift moves, it must stop where there is a request.
8. When there are several requests, the software must (if necessary) continue in the same direction than its last move.

How to modelize ?

Remarks

- With finite model-checking we can't prove a property with parameters. For that, we need theorem proving method. So we need to fix the number of floors.
- 1000 seems a good choice since no building in the world have so much floors, but no model checker in the world can deal with such model.
- On the opposite, every model checker can deal with a building with only one floor, but a lift is not usefull in such a building.
- In addition, most of the properties are tautology for a one floor building.

How to modelize ?

The minimal number of floors

No requirements must be a tautology in the model. This means that we have to choose the least number for witch any requirement is not trivialy satisfy.

1. One floor is mandatory.
2. One floor is mandatory.
3. Two floors are mandatory.
4. Three floors are mandatory.
5. Two floors are mandatory.
6. Two floors are mandatory.
7. Three floors are mandatory.
8. Three (or four ?) floors are mandatory.

We choose four floors to have more confidence.

How to modelize ?

Open or close system

- An open system is a system with free inputs representing the environment's information. This type of system is use when the environment is not well described and when we want to know in which kind of environment, the system is correct.
- A close system is an open system and its environment describe as a particular component of the whole system.

Users can only push button in this system. The better way to describe users is to abstract them by the push action on button.

How to modelize ?

Architecture or functionnal design ?

- AltaRica language is enough general for the two.
- We have to convince the owner of the building. He is certainly not an engineer, nor a computer scientist.
- I think it is easier to convince him with an architecture model witch is certainly less far to the real system than the functionnale one.

How to modelize ?

The system to model

How to modelize ?

The hierarchy of the model

- To convice the owner, the hierarchy must reflect the real building.
- A top-down analyse permits to discover :
  1. Four floors and a lift.
  2. A door and four buttons in the lift.
  3. A door and a button in each floor.

Task of modelling and validation

What kind of button ?

Numerous choice for a button. Analyze of the required functionnalities is necessary :

- A push button including a light and not a switch button.
- A signal to light off the button. Is it always possible to (send/receive) this signal or not ?

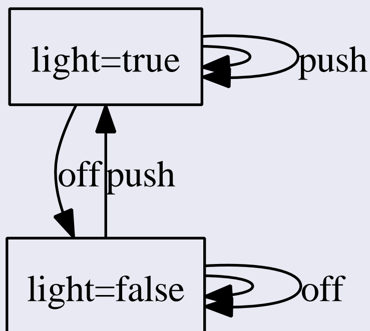
Task of modelling and validation

An AltaRica model of a button

```
/* A Button reacts to
 *   - actions of users
 *   - a signal to light off (even if it is off)
*/
node Button
  state  light : bool : public;
  event  push : public;
         off;
  trans  true |- push -> light := true;
         true |- off  -> light := false;
  init   light := false;
edon
```

Task of modelling and validation

Button's semantic



Task of modelling and validation

What kind of door ?

Numerous choice for a door. Analyze of the required functionnalities is necessary :

- An unique signal to alternativally open and close the door.
- A signal to close the door (even if the door is close), and another signal to open the door (even if the door is open).

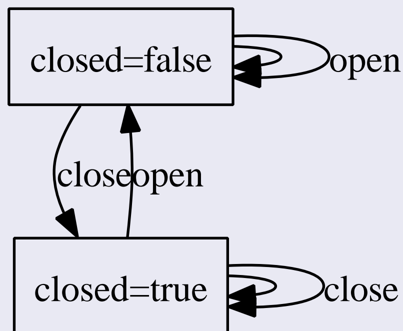
Task of modelling and validation

An AltaRica model of a door

```
/* A Door reacts to:
 *   - a signal to open the door
 *   - a signal to close the door
*/
node Door
  state
    closed : bool : public;
  event
    open, close : public;
  trans
    true |- open  -> closed := false;
    true |- close -> closed := true;
  init
    closed := true;
edon
```

Task of modelling and validation

Door's semantic



Task of modelling and validation

How to built a floor ?

- A floor contains a button and a door.
- We can send the `off` signal to the button when the corresponding request is satisfy.
- We have to chose the meaning for “the service is done”
  - The opening instant.
  - The closing instant.

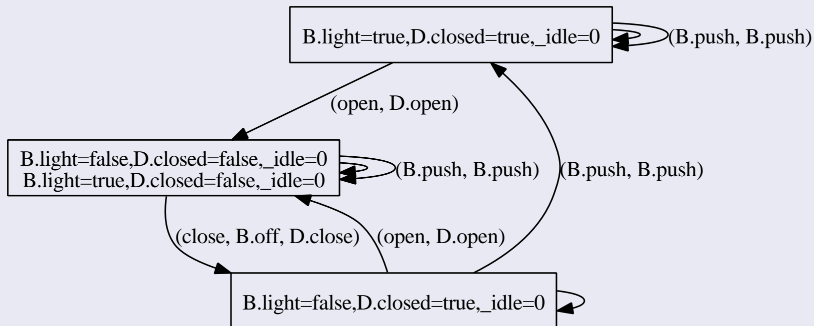
Task of modelling and validation

An AltaRica model of a floor

```
/* A floor is made of a door and a button.
 * We need a meaning for "the service is done"
 *    - it can be the opening instant
 *    - it can be the closing instant
 * We choose the closing instant
 * to send the "off" signal
*/
node Floor
  sub   B : Button;
        D : Door;
  event close, open;
  trans ~D.closed |- close -> ;
        D.closed |- open -> ;
  sync  <close, D.close, B.off>;
        <open, D.open>;
edon
```

Task of modelling and validation

Floor's semantic



Task of modelling and validation

How to built a lift ?

- A lift contains four buttons and a door.
- A lift moves only if its door is closed.
- We can send the `off` signal to the appropriate button when the corresponding request is satisfy.
- We have to chose the meaning for “the service is done”.
  - The opening instant.
  - The closing instant.

  We made the same choice as for the floor.

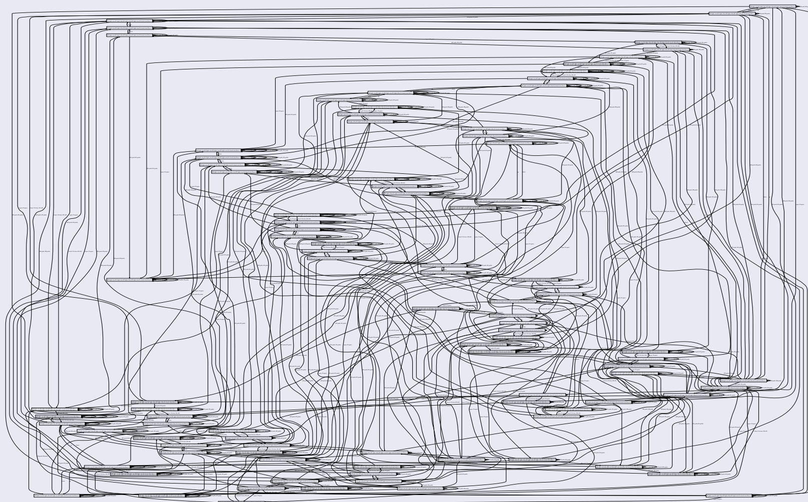
Task of modelling and validation

An AltaRica model of a lift

```
/* A lift contains one button peer floor (4) and a door.
 * Same choices as for the Floor component.
*/
node Lift
  state  floor : [0,3] : parent;    init   floor := 0;
  sub    D : Door; B0, B1, B2, B3 : Button;
  event  up, down, close0, close1, close2, close3, open;
  trans  D.closed |- up   -> floor := floor + 1;
         D.closed |- down -> floor := floor - 1;
         ~D.closed & floor = 0 |- close0 -> ;
         ~D.closed & floor = 1 |- close1 -> ;
         ~D.closed & floor = 2 |- close2 -> ;
         ~D.closed & floor = 3 |- close3 -> ;
         D.closed |- open -> ;
  sync   <close0, D.close, B0.off>;
         <close1, D.close, B1.off>;
         <close2, D.close, B2.off>;
         <close3, D.close, B3.off>;
```

Task of modelling and validation

Lift's semantic



Task of modelling and validation

Lift's validation

```
/*
 * Properties for node : Lift
 * # state properties : 2
 *
 * any_s = 128
 * initial = 1
 *
 * # trans properties : 5
 *
 * epsilon = 128
 * self_epsilon = 128
 * not_deterministic = 0
 * any_t = 864
 * self = 384
 */
TEST(dead=0) [PASSED]
TEST(notSCC=0) [PASSED]
```

Task of modelling and validation

How is the building ?

The building contains four floors and one lift.

- The lift's door and a floor's door open and close synchronously.
- open is possible at some floor only is there is some request to that floor.
- The lift move up (resp. down) only if there is an up (resp. down) request.

Task of modelling and validation

An AltaRica model of a building (1)

```
/* The building contains four floors and one lift.
 *  - The two doors open and close synchronously.
 *  - open only if some request to that floor exists.
 *  - up only if some up request exists.
 *  - down only if some down request exists.
*/
node Building1
  sub
    F0, F1, F2, F3 : Floor;
    L : Lift;
```

Task of modelling and validation

An ALTARICA model of a building (2)

```
flow
  requestUp, requestDown : bool : private;
  request0, request1, request2, request3 : bool : private;
assert
  request0 = (L.B0.light | F0.B.light);
  request1 = (L.B1.light | F1.B.light);
  request2 = (L.B2.light | F2.B.light);
  request3 = (L.B3.light | F3.B.light);
  requestUp   = ((L.floor=0&(request3|request2|request1)) |
                 (L.floor=1&(request3|request2)) |
                 (L.floor=2&(request3)));
  requestDown = ((L.floor=3&(request0|request1|request2)) |
                 (L.floor=2&(request0|request1)) |
                 (L.floor=1&(request0)));
```

Task of modelling and validation

An AltaRica model of a building (3)

```
  event down, up, open0, open1, open2, open3;
  trans (L.floor=0) & request0 |- open0 -> ;
        (L.floor=1) & request1 |- open1 -> ;
        (L.floor=2) & request2 |- open2 -> ;
        (L.floor=3) & request3 |- open3 -> ;
        requestDown |- down    -> ;
        requestUp   |- up      -> ;
  sync <up,    L.up>;
       <down,  L.down>;
       <open0, L.open, F0.open>;
       <open1, L.open, F1.open>;
       <open2, L.open, F2.open>;
       <open3, L.open, F3.open>;
       <L.close0, F0.close>;
       <L.close1, F1.close>;
       <L.close2, F2.close>;
       <L.close3, F3.close>;
edon
```

Task of modelling and validation

Too big to draw the graph.

Building's validation

```
/*
 * Properties for node : Building1
 * # state properties : 2
 *
 * any_s = 1792
 * initial = 1
 *
 * # trans properties : 5
 *
 * epsilon = 1792
 * self_epsilon = 1792
 * not_deterministic = 0
 * any_t = 19032
 * self = 9216
 */
TEST(dead=0) [PASSED]
TEST(notSCC=0) [PASSED]
```

Task of modelling and validation

Too big to draw the graph.

Building's specific validation

```
/*
 * Properties for node : Building1
 * # state properties : 8
 *
 * level0 = 448
 * level1 = 448
 * level2 = 448
 * level3 = 448
 * open0 = 192
 * open1 = 192
 * open2 = 192
 * open3 = 192
 *
 * # trans property : 0
 *
 */
```

Verification of safety properties

P1 : When a button is push, it lights.

Property P1

```
// Safety properties
with Building1, Building2, Building3, Building4DF, Building
 // When a button is push, it lights.
 notP1 := tgt(label F0.B.push)-[F0.B.light] |
          tgt(label L.B0.push)-[L.B0.light] |
          tgt(label F1.B.push)-[F1.B.light] |
          tgt(label L.B1.push)-[L.B1.light] |
          tgt(label F2.B.push)-[F2.B.light] |
          tgt(label L.B2.push)-[L.B2.light] |
          tgt(label F3.B.push)-[F3.B.light] |
          tgt(label L.B3.push)-[L.B3.light] ;
 test(notP1,0) > '$NODENAME.P1';
 traceP1     := trace(initial,any_t,notP1);
 dot(src(traceP1)|tgt(traceP1), traceP1)
            > '$NODENAME-P1.dot';
```

Verification of safety properties

P1 : When a button is push, it lights.

Building1 : Property P1

```
TEST(notP1=0) [PASSED]
```

Verification of safety properties

P2 : When the corresponding service is done, it lights off.

Property P2

```
// Safety properties
with Building1, Building2, Building3, Building4DF, Building
 // When the corresponding service is done,
 // the button lights off.
 notP2 := tgt(label F0.close)&[request0] |
          tgt(label F1.close)&[request1] |
          tgt(label F2.close)&[request2] |
          tgt(label F3.close)&[request3] ;
 test(notP2,0) > '$NODENAME.P2';
 traceP2    := trace(initial,any_t,notP2);
 dot(src(traceP2)|tgt(traceP2), traceP2)
            > '$NODENAME-P2.dot';
done
```

Verification of safety properties

P2 : When the corresponding service is done, it lights off.

Building1 : Property P2

```
TEST(notP2=0) [PASSED]
```

Verification of safety properties

P3 : At each floor, the door is close if the lift is not here.

Property P3

```
// Safety properties
with Building1, Building2, Building3, Building4DF, Building
 // At each floor, the door is close
 // if the lift is not here.
 notP3 := ([L.floor!=0] - [F0.D.closed]) |
          ([L.floor!=1] - [F1.D.closed]) |
          ([L.floor!=2] - [F2.D.closed]) |
          ([L.floor!=3] - [F3.D.closed]) ;
 test(notP3,0) > '$NODENAME.P3';
 traceP3    := trace(initial,any_t,notP3);
 dot(src(traceP3)|tgt(traceP3), traceP3)
            > '$NODENAME-P3.dot';
done
```

Verification of safety properties

P3 : At each floor, the door is close if the lift is not here.

Building1 : Property P3

```
TEST(notP3=0) [PASSED]
```

Verification of safety properties

P5 : The software opens the door at some floor only if there is some requests for that floor.

Property P5

```
// Safety properties
with Building1, Building2, Building3, Building4DF, Building
 // The software opens the door at some floor
 // only if there is some requests for that floor.
 notP5 := (label F0.D.open - rsrc([request0])) |
          (label F1.D.open - rsrc([request1])) |
          (label F2.D.open - rsrc([request2])) |
          (label F3.D.open - rsrc([request3])) ;
 test(notP5,0) > '$NODENAME.P5';
 traceP5 := trace(initial,any_t,src(notP5));
 ceP5 := reach(src(traceP5),traceP5|notP5);
 dot(ceP5, (traceP5|notP5)) > '$NODENAME-P5.dot';
done
```

Verification of safety properties

P5 : The software opens the door at some floor only if there is some requests for that floor.

Building1 : Property P5

```
TEST(notP5=0) [PASSED]
```

Verification of safety properties

P6 : If there is no request, the lift must stay at the same floor.

Property P6

```
// Safety properties
with Building1, Building2, Building3, Building4DF, Building
 // If there is no request,
 // the lift must stay at the same floor.
 notP6 := (label L.up | label L.down) -
       rsrc([request0|request1|request2|request3]);
 test(notP6,0) > '$NODENAME.P6';
 traceP6 := trace(initial,any_t,src(notP6));
 ceP6 := reach(src(traceP6),traceP6|notP6);
 dot(ceP6, traceP6|notP6) > '$NODENAME-P6.dot';
done
```

Verification of safety properties

P6 : If there is no request, the lift must stay at the same floor.

Building1 : Property P6

```
TEST(notP6=0) [PASSED]
```

Verification of safety properties

P7 : When the lift moves, it must stop where there is a request.

Property P7

```
// Safety properties
with Building1, Building2, Building3, Building4DF, Building
 // When the lift moves,
 // it must stop where there is a request.
 notP7 := (label L.up | label L.down) &
          rsrc([L.floor=0 & request0] |
               [L.floor=1 & request1] |
               [L.floor=2 & request2] |
               [L.floor=3 & request3]) ;
 test(notP7,0) > '$NODENAME.P7';
 traceP7 := trace(initial,any_t,src(notP7));
 ceP7 := reach(src(traceP7),traceP7|notP7);
 dot(ceP7, (traceP7|notP7)) > '$NODENAME-P7.dot';
done
```

Verification of safety properties

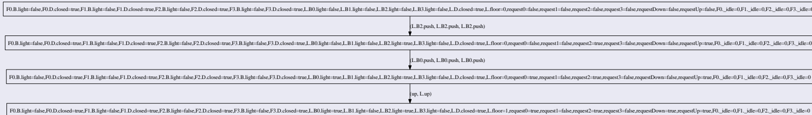
P7 : When the lift moves, it must stop where there is a request.

Building1 : Property P7

```
TEST(notP7=0) [FAILED] actual size = 1026
```

Verification of safety properties

Property P7 : a counter example



Verification of safety properties

Property P7 : a correction

```
d7 1
a7 3
/*  - the lift moves if no request for the current floor.
*/
node Building2
d25 1
a25 1
  event {down, up} < {open0, open1, open2, open3};
```

Verification of safety properties

Building2 : Properties P1, P2, P3, P5, P6 and P7

```
TEST(notP1=0) [PASSED]

TEST(notP2=0) [PASSED]

TEST(notP3=0) [PASSED]

TEST(notP5=0) [PASSED]

TEST(notP6=0) [PASSED]

TEST(notP7=0) [PASSED]
```

Verification of safety properties

P8 : When there are several requests, the software must (if necessary) continue in the same direction than its last move.

Property P8

```
// Safety properties
with Building2, Building3, Building4DF, Building4NDF, Builc
 // When there are several requests,
 // the software must (if necessary) continue
 // in the same direction than its last move.
 notP8 :=  label L.up & rsrc(src(label L.down)) |
           label L.down & rsrc(src(label L.up)) ;
 test(notP8,0) > '$NODENAME.P8';
 traceP8 := trace(initial,any_t,src(notP8));
 ceP8 := reach(src(traceP8),traceP8|notP8);
 dot(ceP8, (traceP8|notP8)) > '$NODENAME-P8.dot';
done
```

Verification of safety properties

P8 : When there are several requests, the software must (if necessary) continue in the same direction than its last move.

Building2 : Property P8

```
TEST(notP8=0) [FAILED] actual size = 180
```

Verification of safety properties

Property P8 : a counter example

Verification of safety properties

Property P8 : a correction

```
/*  - last move of the lift is record in a variable.
 *  - this variable is use to control moves
*/
node Building3
  state climb : bool; init climb := false;
    climb & requestUp                |- up   -> ;
    ~climb & requestDown             |- down -> ;
    ~climb&~requestDown&requestUp |- up   -> climb:=true;
    climb&~requestUp&requestDown |- down -> climb:=false;
```

Verification of safety properties

Building3 : Properties P1, P2, P3, P5, P6, P7 and P8

```
TEST(notP1=0) [PASSED]

TEST(notP2=0) [PASSED]

TEST(notP3=0) [PASSED]

TEST(notP5=0) [PASSED]

TEST(notP6=0) [PASSED]

TEST(notP7=0) [PASSED]

TEST(notP8=0) [PASSED]
```

Verification of liveness properties

P4 : Each request must be honored a day.

Auxilliary properties for P4

```
with Building3, Building4DF, Building4NDF, Building5NDF do
 // Preliminary properties for P4
 // we remove "self" to don't
 // take account redondancy "push" events
 waitB0 := rsrc([L.B0.light])&rtgt([L.B0.light])-self;
 waitB1 := rsrc([L.B1.light])&rtgt([L.B1.light])-self;
 waitB2 := rsrc([L.B2.light])&rtgt([L.B2.light])-self;
 waitB3 := rsrc([L.B3.light])&rtgt([L.B3.light])-self;
 waitF0 := rsrc([F0.B.light])&rtgt([F0.B.light])-self;
 waitF1 := rsrc([F1.B.light])&rtgt([F1.B.light])-self;
 waitF2 := rsrc([F2.B.light])&rtgt([F2.B.light])-self;
 waitF3 := rsrc([F3.B.light])&rtgt([F3.B.light])-self;
done
```

Verification of liveness properties

P4 : Each request must be honored a day.

Property P4

```
// Liveness properties
with Building3 do
 // Each request must be honored a day.
 notP4 := loop(any_t, waitB0) |
          loop(any_t, waitB1) |
          loop(any_t, waitB2) |
          loop(any_t, waitB3) |
          loop(any_t, waitF0) |
          loop(any_t, waitF1) |
          loop(any_t, waitF2) |
          loop(any_t, waitF3) ;
 test(notP4,0) > '$NODENAME.P4';
 traceP4 := trace(initial,any_t,src(notP4));
 ceP4 := reach(src(traceP4),traceP4|notP4);
 dot(ceP4, (traceP4|notP4)) > '$NODENAME-P4.dot';
```

Verification of liveness properties

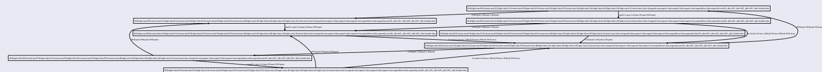
P4 : Each request must be honored a day.

Building3 : Property P4

```
TEST(notP4=0) [FAILED] actual size = 4536
```

Verification of liveness properties

Property P4 : a counter example



Verification of liveness properties

P4a : Each request must be honored a day, if the lift moves sometimes.

Property P4a

```
with Building3, Building4DF, Building4NDF, Building5NDF do
 // A new version of P4: Each request must be
 // honored a day, if the lift moves sometimes.
 move := label L.up | label L.down;
 notP4a := loop(move, waitB0) |
           loop(move, waitB1) |
           loop(move, waitB2) |
           loop(move, waitB3) |
           loop(move, waitF0) |
           loop(move, waitF1) |
           loop(move, waitF2) |
           loop(move, waitF3) ;
 test(notP4a,0) > '$NODENAME.P4a';
 traceP4a    := trace(initial,any_t,src(notP4a));
```

Verification of liveness properties

P4a : Each request must be honored a day, if the lift moves sometimes.

Building3 : Property P4a

```
TEST(notP4a=0) [PASSED]
```

Non determinism and failures

A door with explicit failures

```
/* A Door reacts to:
 *   - a signal to open the door
 *   - a signal to close the door
 * the reaction can lead to a stucked state
 * determinism is used to modelize the failure
*/
node DoorDF
  state closed : bool : public;
        stucked : bool;
  event open, close, failure : public;
  trans not stucked |- open    -> closed := false;
        not stucked |- close   -> closed := true;
        not stucked |- failure -> stucked := true;
        stucked     |- open, close ->;
  init  closed := true, stucked := false;
```

Non determinism and failures

A door with explicit failures

closed=false,stucked=false
open
close
open
failure
closed=true,stucked=false
close
failure
closed=false,stucked=true
closed=true,stucked=true
close
open

Non determinism and failures

A door with non determinism failures

```
/* A Door reacts to:
 *   - a signal to open the door
 *   - a signal to close the door
 * the reaction can lead to a stucked state
 * non determinism is used to modelize the failure
*/
node DoorNDF
  state closed : bool : public;
        stucked : bool;
  event open, close : public;
  trans not stucked |- open  -> closed := false;
        not stucked |- close -> closed := true;
        true |- open, close -> stucked := true;
  init  closed := true, stucked := false;
edon
```

Non determinism and failures

A door with non determinism failures

closed=false,stucked=false
open
close
open
close
open
closed=true,stucked=false
close
open
close
closed=false,stucked=true
closed=true,stucked=true
close
open

Non determinism and failures

A floor with explicit failures

```
/* A floor is made of a door and a button.
 * We need a meaning for "the service is done"
 *   - it can be the opening instant
 *   - it can be the closing instant
 * We choose the closing instant
 * to send the "off" signal
*/
node FloorDF
  sub   B : Button; D : DoorDF;
  event close, open;
  trans ~D.closed |- close -> ;
        D.closed |- open -> ;
  sync  <close, D.close, B.off>;
        <close, D.failure, B.off>;
        <open, D.open>;
```

Non determinism and failures

A floor with non determinism failures

```
/* A floor is made of a door and a button.
 * We need a meaning for "the service is done"
 *   - it can be the opening instant
 *   - it can be the closing instant
 * We choose the closing instant
 * to send the "off" signal
*/
node FloorNDF
  sub   B : Button; D : DoorNDF;
  event close, open;
  trans ~D.closed |- close -> ;
        D.closed |- open -> ;
  sync  <close, D.close, B.off>;
        <open, D.open>;
edon
```

Non determinism and failures

A floor with explicit failures

```
/*
 * Properties for node : FloorDF
 * # state properties : 2
 *
 * any_s = 8
 * initial = 1
 *
 * # trans properties : 5
 *
 * epsilon = 8
 * self_epsilon = 8
 * not_deterministic = 0
 * any_t = 28
 * self = 15
 */
```

Non determinism and failures

A floor with non determinism failures

```
/*
 * Properties for node : FloorNDF
 * # state properties : 2
 *
 * any_s = 8
 * initial = 1
 *
 * # trans properties : 5
 *
 * epsilon = 8
 * self_epsilon = 8
 * not_deterministic = 8
 * any_t = 28
 * self = 15
 */
```

Non determinism and failures

A lift with explicit failures

```
d4 1
a4 1
node LiftDF
d6 2
a7 3
  sub     D : DoorDF;
          B0, B1, B2, B3 : Button;
  event   up, down, close0, close1, close2, close3,open;
a18 4
          <close0, D.failure, B0.off>;
          <close1, D.failure, B1.off>;
          <close2, D.failure, B2.off>;
          <close3, D.failure, B3.off>;
a19 1
          <open, D.failure>;
```

Non determinism and failures

A lift with non determinism failures

```
d4 1
a4 1
node LiftNDF
d6 1
a6 2
  sub    D : DoorNDF;
         B0, B1, B2, B3 : Button;
```

Non determinism and failures

A building with explicit failures

```
d12 1
a12 1
node Building4DF
d14 2
a15 2
    F0, F1, F2, F3 : FloorDF;
    L : LiftDF;
```

Non determinism and failures

A building with non determinism failures

```
d12 1
a12 1
node Building4NDF
d14 2
a15 2
    F0, F1, F2, F3 : FloorNDF;
    L : LiftNDF;
```

Non determinism and failures

A building with explicit failures

```
/*
 * Properties for node : Building4DF
 * # state properties : 2
 *
 * any_s = 140952
 * initial = 1
 *
 * # trans properties : 5
 *
 * epsilon = 140952
 * self_epsilon = 140952
 * not_deterministic = 0
 * any_t = 1,45551e+06
 * self = 782142
 */
```

Non determinism and failures

A building with non determinism failures

```
/*
 * Properties for node : Building4NDF
 * # state properties : 2
 *
 * any_s = 140952
 * initial = 1
 *
 * # trans properties : 5
 *
 * epsilon = 140952
 * self_epsilon = 140952
 * not_deterministic = 98892
 * any_t = 1,45551e+06
 * self = 782142
 */
```

Non determinism and failures

Building4[DF,NDF] : Properties P1, P2, P3, P4a, P5, P6, P7 and P8

```
TEST(notP1=0) [PASSED]

TEST(notP2=0) [PASSED]

TEST(notP3=0) [FAILED] actual size = 130200

TEST(notP4a=0) [FAILED] actual size = 90324

TEST(notP5=0) [PASSED]

TEST(notP6=0) [PASSED]

TEST(notP7=0) [FAILED] actual size = 43308

TEST(notP8=0) [PASSED]
```

Non determinism and failures

Property P3 : a counter example

```
F0.B.light=false,F0.D.closed=true,F1.B.light=false,F1.D.closed=true,F2.B.light=false,F2.D.closed=true,F3.B.light=false,F3.D.closed=true,L..B0.light=false,L..B1.light=false,L..B2.light=false,L..B3.light=false,L.D.closed=true,L..floor=0,request0=false,request1=false,request2=false,request3=false,requestDown=false,requestUp=false,F0.D.stucked=false,F0._idle=0,F1.D.stucked=false,F1._idle=0,F2.D.stucked=false,F2._idle=0,F3.D.stucked=false,F3._idle=0,L.D.stucked=false,climb=false
    (L..B0.push, L..B0.push, L..B0.push)
F0.B.light=false,F0.D.closed=true,F1.B.light=false,F1.D.closed=true,F2.B.light=false,F2.D.closed=true,F3.B.light=false,F3.D.closed=true,L..B0.light=true,L..B1.light=false,L..B2.light=false,L..B3.light=false,L.D.closed=true,L..floor=0,request0=true,request1=false,request2=false,request3=false,requestDown=false,requestUp=false,F0.D.stucked=false,F0._idle=0,F1.D.stucked=false,F1._idle=0,F2.D.stucked=false,F2._idle=0,F3.D.stucked=false,F3._idle=0,L.D.stucked=false,climb=false
    (open0, L..open, L.D.open, F0.open, F0.D.open)
F0.B.light=false,F0.D.closed=false,F1.B.light=false,F1.D.closed=true,F2.B.light=false,F2.D.closed=true,F3.B.light=false,F3.D.closed=true,L..B0.light=true,L..B1.light=false,L..B2.light=false,L..B3.light=false,L.D.closed=true,L..floor=0,request0=true,request1=false,request2=false,request3=false,requestDown=false,requestUp=false,F0.D.stucked=false,F0._idle=0,F1.D.stucked=false,F1._idle=0,F2.D.stucked=false,F2._idle=0,F3.D.stucked=false,F3._idle=0,L.D.stucked=true,climb=false
    (L..B3.push, L..B3.push, L..B3.push)
F0.B.light=false,F0.D.closed=false,F1.B.light=false,F1.D.closed=true,F2.B.light=false,F2.D.closed=true,F3.B.light=false,F3.D.closed=true,L..B0.light=true,L..B1.light=false,L..B2.light=false,L..B3.light=true,L.D.closed=true,L..floor=0,request0=true,request1=false,request2=false,request3=true,requestDown=false,requestUp=true,F0.D.stucked=false,F0._idle=0,F1.D.stucked=false,F1._idle=0,F2.D.stucked=false,F2._idle=0,F3.D.stucked=false,F3._idle=0,L.D.stucked=true,climb=false
    (up, L..up)
F0.B.light=false,F0.D.closed=false,F1.B.light=false,F1.D.closed=true,F2.B.light=false,F2.D.closed=true,F3.B.light=false,F3.D.closed=true,L..B0.light=true,L..B1.light=false,L..B2.light=false,L..B3.light=true,L.D.closed=true,L..floor=1,request0=true,request1=false,request2=false,request3=true,requestDown=true,requestUp=true,F0.D.stucked=false,F0._idle=0,F1.D.stucked=false,F1._idle=0,F2.D.stucked=false,F2._idle=0,F3.D.stucked=false,F3._idle=0,L.D.stucked=true,climb=true
```

Non determinism and failures

Property P3 : a correction

```
d12 1
a12 1
node Building5NDF
a19 1
    doorsareclosed : bool : private;
a30 4
    doorsareclosed = ((L.floor=0 & F0.D.closed)|
                      (L.floor=1 & F1.D.closed)|
                      (L.floor=2 & F2.D.closed)|
                      (L.floor=3 & F3.D.closed));
d36 4
a39 4
    climb & requestUp & doorsareclosed |- up   -> ;
    ~climb & requestDown & doorsareclosed |- down -> ;
    ~climb&~requestDown&requestUp&doorsareclosed |- up
```

Non determinism and failures

Building5NDF : Properties P1, P2, P3, P4a, P5, P6, P7 and P8

```
TEST(notP1=0) [PASSED]

TEST(notP2=0) [PASSED]

TEST(notP3=0) [PASSED]

TEST(notP4a=0) [PASSED]

TEST(notP5=0) [PASSED]

TEST(notP6=0) [PASSED]

TEST(notP7=0) [PASSED]

TEST(notP8=0) [PASSED]
```

Non determinism and failures

Building5NDF : validation

```
/*
 * Properties for node : Building5NDF
 * # state properties : 2
 *
 * any_s = 10752
 * initial = 1
 *
 * # trans properties : 5
 *
 * epsilon = 10752
 * self_epsilon = 10752
 * not_deterministic = 9216
 * any_t = 109050
 * self = 56832
 */
```

Conclusion

The result

- We have to precise some details in the informal description.
  - What is a button and a door ?
  - What is the meaning of "The service is done" ?
- We have to precise some requirements.
  - What is the meaning of "Each request must be honored a day" ?
- After that, we have built a model of a lift which satisfy all the requirements.
- At the end, we have shown the power of non determinism to represent failures.

Conclusion

The different tasks

- To obtain a validate small model is not easy.
- To write logical properties is not easy too, but there is a lot of reuse.

Conclusion

Performances

```
NetBSD amd64 x86_64
       62,45 real         60,63 user          0,98 sys
           0  maximum resident set size
           0  average shared memory size
           0  average unshared data size
           0  average unshared stack size
      179729  page reclaims
          45  page faults
           0  swaps
           4  block input operations
         141  block output operations
          45  messages sent
         127  messages received
           0  signals received
         188  voluntary context switches
```