Safety Solutions Designer

AIX
LYON
PARIS
STRASBOURG

WWW.CLEARSY.COM

Hackathon
JUL2025

Airlock
Access Control

ADDENDUM

PART I

One Solution

Thierry Lecomte
R&D Director

THIERRY.LECOMTE@CLEARSY.COM

Handling constants with Atelier B and ProB

- Constants are seen and used as defined by their properties
- Properties may be badly designed and contradictory
  - Ex: C1: INTEGER & C1 >10 & 2 > C1
- Properties become hypotheses for all related predicates
  - C1 > 10 & 2 > C1 => any P
- Instantiation V is required to demonstrate feasibility
  - (C1 > 10 & 2 > C1)[C1 := V]

Handling constants with Atelier B and ProB

```
MACHINE
    ACCESS_CARD
ABSTRACT_CONSTANTS
    map,
    idx_odd,
    idx_even
PROPERTIES
    map: 0..9 --> 0..9 &
    map = {
        0 |-> 0, 1 |-> 2, 2 |-> 4, 3 |-> 6, 4 |-> 8,
        5 |-> 1, 6 |-> 3, 7 |-> 5, 8 |-> 7, 9 |-> 9
    } &
    idx_odd <: 0..15 &
    idx_even <: 0..15 &
    idx_odd /\ idx_even = {} &
    idx_odd \/ idx_even = 0..15 &
    idx_odd = {1, 4, 5, 7, 9, 11, 13, 15} &
    idx_even = {0, 2, 4, 6, 8, 10, 12, 14}
```

Erreur	PROPERTIES are unsatisfiable (but all CONSTANTS valued)

Detected by ProB

This is a miracle

With Atelier B, you need to show there exists at least one value with the clause VALUES

CLEARSY
Safety Solutions Designer

AIX
LYON
PARIS
STRASBOURG

WWW.CLEARSY.COM

Hackathon
JUL2025

Airlock
Safety Controller

Thierry Lecomte
R&D Director

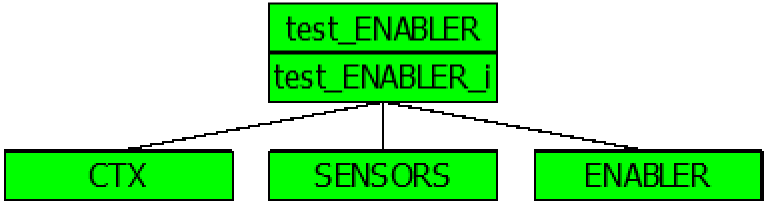
PART II

One Solution

THIERRY.LECOMTE@CLEARSY.COM

Your turn

- *Develop a B model of this enabling function [8 pts]*
  - *3 sensors to read (pressure_l, contact_a, contact_b*
  - *2 enabling variables (enable_door_a, enable_door_b)*
  - *safety properties*



Your turn

- ▶ *SENSORS model*

```
MACHINE
    SENSORS
SEES
    CTX
CONCRETE_VARIABLES
    pressure_sensor_l,
    contact_sensor_a,
    contact_sensor_b
INVARIANT
    pressure_sensor_l : PRESSURES &
    contact_sensor_a : BOOL & // TRUE means door closed
    contact_sensor_b : BOOL   // TRUE means door closed
INITIALISATION
OPERATIONS
    update_sensors_states =
    BEGIN
        pressure_sensor_l,
        contact_sensor_a,
        contact_sensor_b: (
        )
    END
END
```

Any values in their domain
but a door open has an impact on
the pressure in the airlock

Your turn

- *SENSORS model*

```
INITIALISATION
    pressure_sensor_l :: PRESSURES ||
    contact_sensor_a :: BOOL ||
    contact_sensor_b :: BOOL
```

```
pressure_sensor_l,
contact_sensor_a,
contact_sensor_b: (
    pressure_sensor_l : PRESSURES &
    contact_sensor_a : BOOL &
    contact_sensor_b : BOOL &
    (not(pressure_sensor_l = PRESSURE_A) => contact_sensor_a = TRUE) &
    (not(pressure_sensor_l = PRESSURE_B) => contact_sensor_b = TRUE)
)
```

Your turn

- *ENABLER model*

The safety property of the airlock (what do we want to ensure all the time)

The safety property of the control of each enabling variable

```
MACHINE
    ENABLER
SEES
    CTX, SENSORS
CONCRETE_VARIABLES
    enable_door_a,
    enable_door_b
INVARIANT
    enable_door_a : BOOL &
    enable_door_b : BOOL &

INITIALISATION
    enable_door_a := FALSE ||
    enable_door_b := FALSE
OPERATIONS
    compute_enabling =
    PRE
    THEN
        enable_door_a,
        enable_door_b :(
        )
    END
END
```

Your turn

- *ENABLER model*

The safety property of the airlock

```
INVARIANT
    enable_door_a : BOOL &
    enable_door_b : BOOL &
    not(enable_door_a = TRUE & enable_door_b = TRUE)
```

The property to ensure coherency

```
compute_enabling =
PRE
    (not(pressure_sensor_l = PRESSURE_A) => contact_sensor_a = TRUE) &
    (not(pressure_sensor_l = PRESSURE_B) => contact_sensor_b = TRUE)
THEN
    enable_door_a,
    enable_door_b :(
        enable_door_a : BOOL &
        enable_door_b : BOOL &
        (enable_door_a = TRUE => (pressure_sensor_l = PRESSURE_A & contact_sensor_b = TRUE)) &
        (enable_door_b = TRUE => (pressure_sensor_l = PRESSURE_B & contact_sensor_a = TRUE))
    )
END
```

The safety property of the control of each enabling variable

Your turn

- *Check the enabling with ProB [2 pts]*
  - *run the operation 10 times*
  - Save the probtrace file with the value of oks

```
MACHINE
    test_ENABLER
OPERATIONS
    test_compute_enabling = skip
END
```

```
IMPLEMENTATION test_ENABLER_i
REFINES test_ENABLER
IMPORTS CTX, SENSORS, ENABLER
OPERATIONS
    test_compute_enabling =
    BEGIN
        update_sensors_states;
        compute_enabling
    END
END
```

Your turn

- ***Optional**: what happens if we chose the INITIALISATION with both doors open ? [1 pt]*

```
INITIALISATION(contact_sensor_a:=FALSE, contact_sensor_b:=FALSE)
```

Nothing particular. It is just the initialization of the enabler system. If it switch on with already a catastrophic situation, it is not at fault.

- ***Optional**: in the operation test_compute_enabling,what happens if we forgot to update the sensors (we forget to call update_sensors_states) ? [1 pt]*

We compute something not related to reality and we may take wrong decisions

Your turn

- ***Optional**: how would you simply ensure a correct (i.e. proved) sequencing of the two operations **update_sensors_states** and* **compute_enabling** *? [3 pt]*
- *We can rely on the precondition of **compute_enabling** verified by the postcondition of **update_sensors_states***
- *We can add a notion of cycle and to check it with a precondition*

```
SETS
    PRESSURES = {
    };
    CYCLE = {ACQUISITION, COMPUTING};
```

```
CONSTANTS
    next_cycle
PROPERTIES
    next_cycle = {ACQUISITION |-> COMPUTING, COMPUTING |-> ACQUISITION}
END
```

Your turn

```
n_cycle <-- update_sensors_states(cycle) =
PRE cycle = ACQUISITION THEN
    pressure_sensor_l,
    contact_sensor_a,
    contact_sensor_b: (
        pressure_sensor_l : PRESSURES &
        contact_sensor_a : BOOL &
        contact_sensor_b : BOOL &
        (not(pressure_sensor_l = PRESSURE_A) => contact_sensor_a = TRUE) &
        (not(pressure_sensor_l = PRESSURE_B) => contact_sensor_b = TRUE)
    ) ||
    n_cycle := next_cycle(cycle)
END
```

Your turn

```
n_cycle <-- compute_enabling(cycle) =
PRE cycle = COMPUTING &
     (not(pressure_sensor_l = PRESSURE_A) => contact_sensor_a = TRUE)  &
     (not(pressure_sensor_l = PRESSURE_B) => contact_sensor_b = TRUE)
THEN
     enable_door_a,
     enable_door_b :(
          enable_door_a : BOOL &
          enable_door_b : BOOL &
          (enable_door_a = TRUE => (pressure_sensor_l = PRESSURE_A & contact_sensor_b = TRUE)) &
          (enable_door_b = TRUE => (pressure_sensor_l = PRESSURE_B & contact_sensor_a = TRUE))
     )
     ||
     n_cycle := next_cycle(cycle)
END
```

Your turn

```
IMPLEMENTATION test_ENABLER_i
REFINES test_ENABLER
IMPORTS CTX, SENSORS, ENABLER
CONCRETE_VARIABLES
    cycle
INVARIANT
    cycle : CYCLE
INITIALISATION
    cycle := ACQUISITION
OPERATIONS
    test_compute_enabling =
    BEGIN
        cycle <-- update_sensors_states(cycle);
        cycle <-- compute_enabling(cycle)
    END
END
```