Technical Session 3

The CLEARSY Safety Platform

Thierry Lecomte

ROBOSTAR

robostar.cs.york.ac.uk

11th March 2025

Royal Academy of Engineering

UKRI Engineering and Physical Sciences Research Council

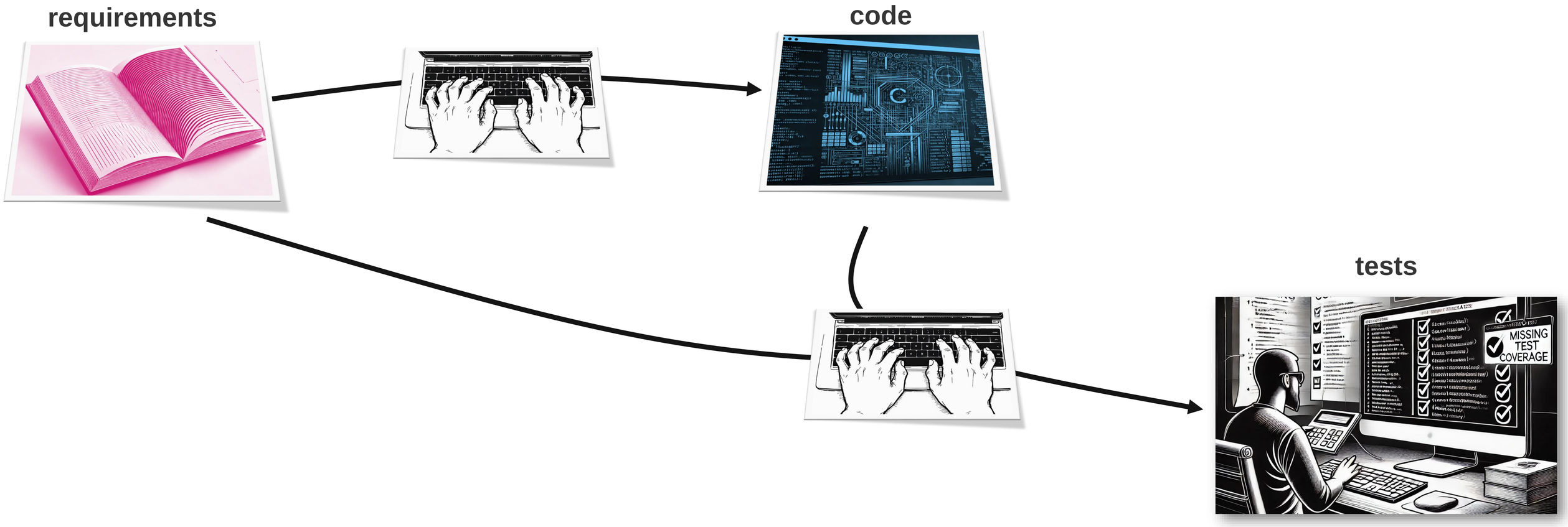
Agenda

- Introduction to proven software
- Introduction to the CLEARSY Safety Platform
- Development process (demo video)
- Bits of B
- Using the modelling interface

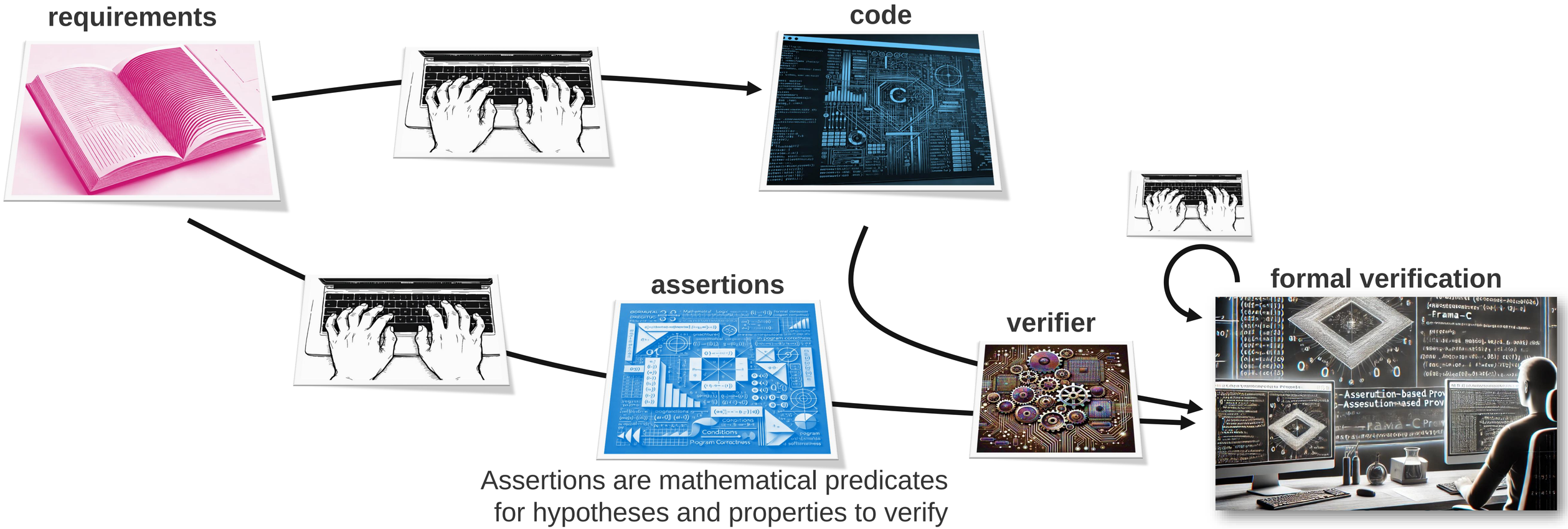
ROBOSTAR

Introduction to Proven Software

Introduction to proven software: code manually produced and tested



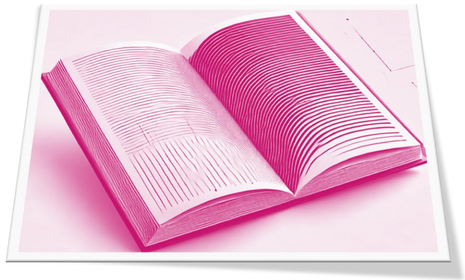
Introduction to proven software: code manually produced and **formally verified**



Assertions are mathematical predicates
for hypotheses and properties to verify

Introduction to proven software: code **formally specified and verified**

requirements



formal model

code

code generator

formal proof

prover

Formal model use mathematical notation
to specify and implement behaviour

Introduction to proven software: formal proof

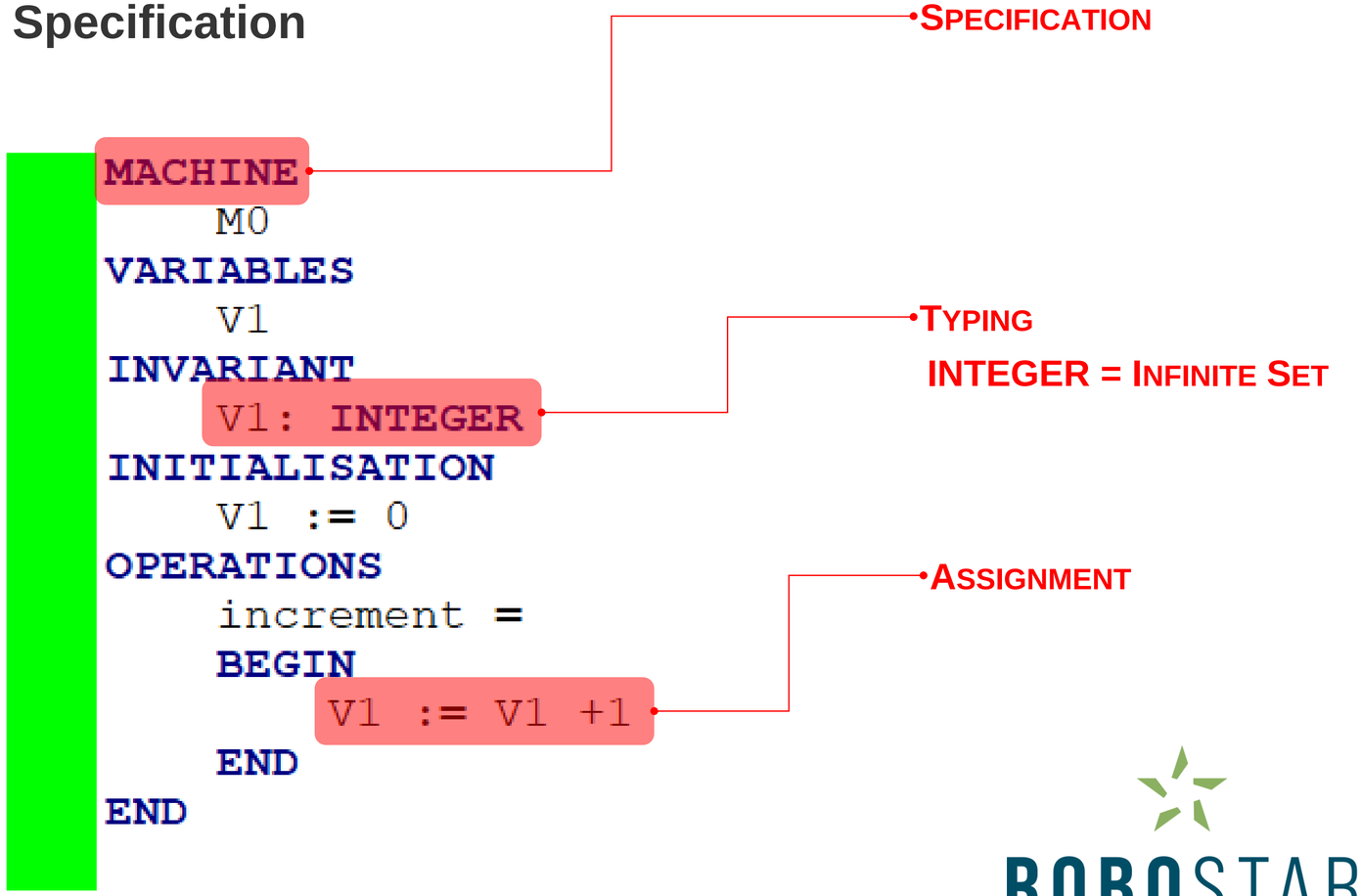
Requirement

V1 is a variable of type Integer

Its initial value is 0

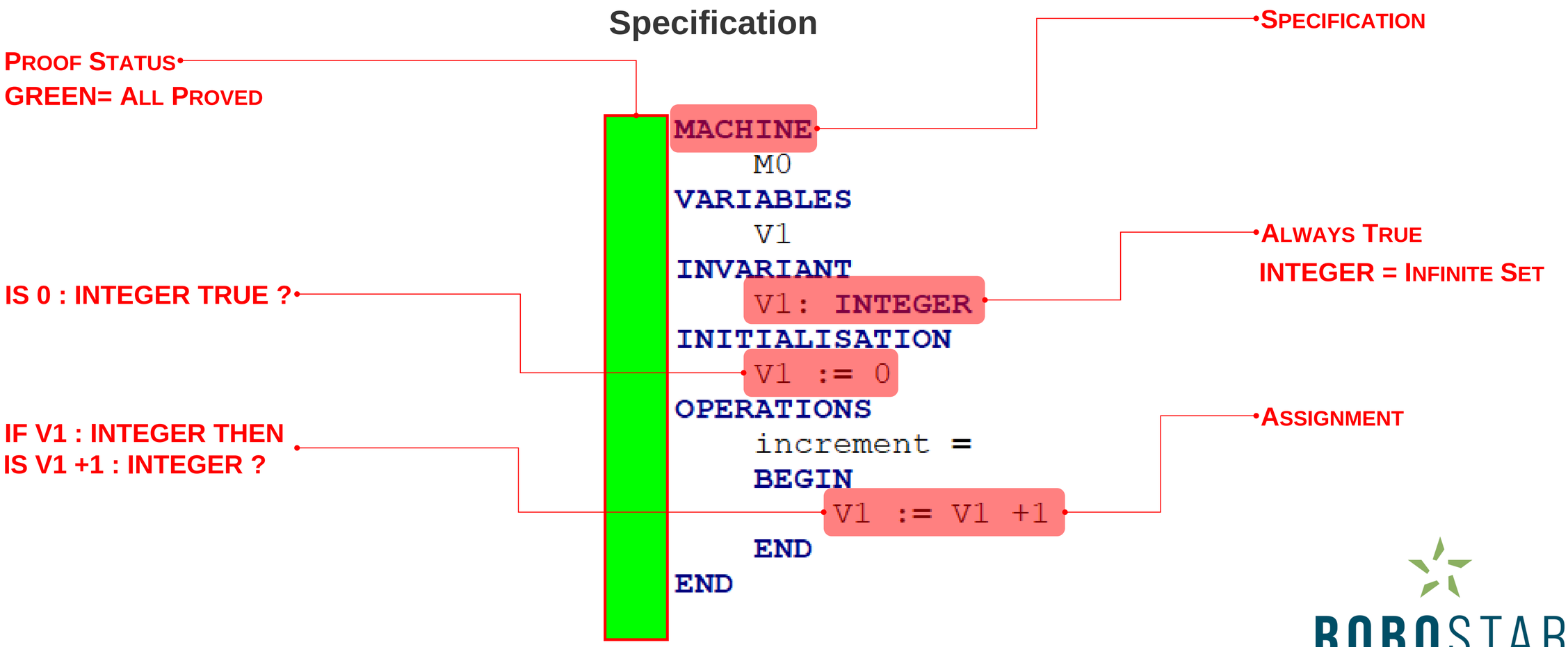
The function *increment* adds one to V1

Specification



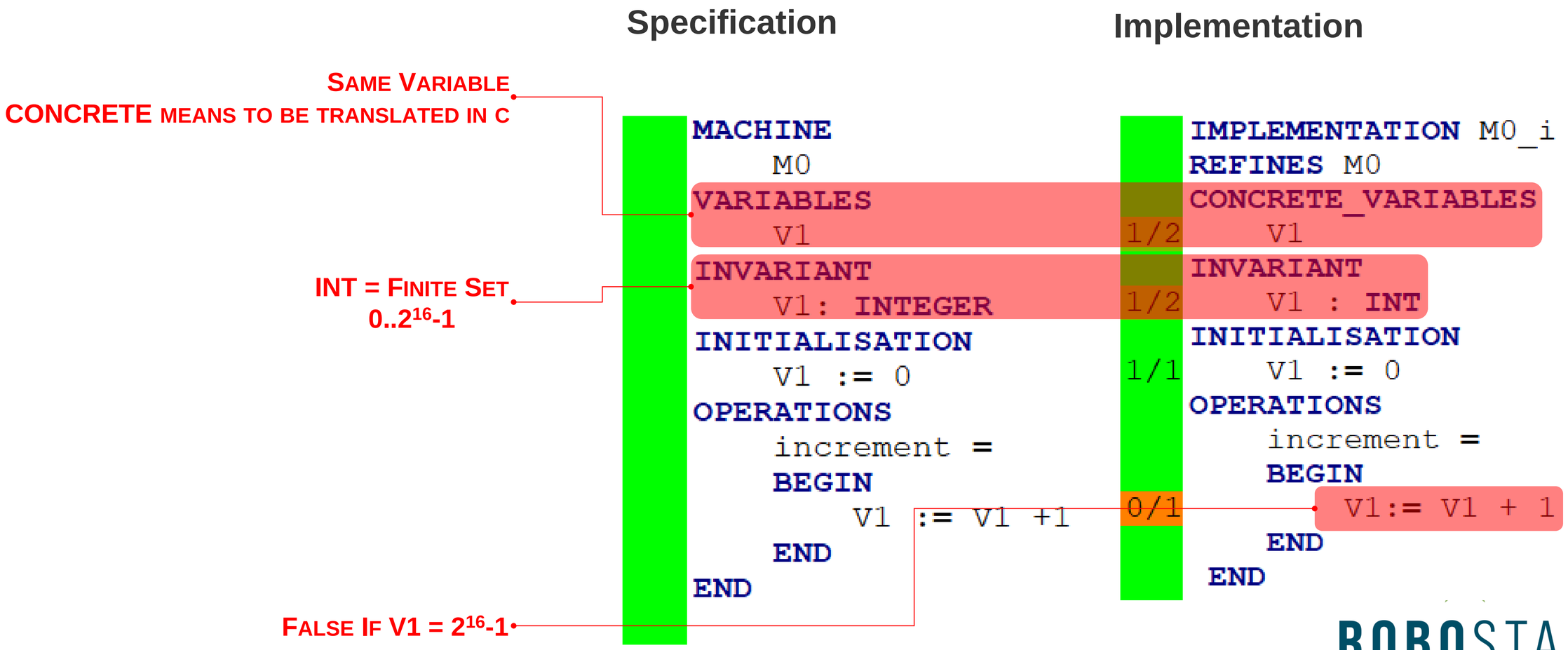
ROBOSTAR

Introduction to proven software: formal proof



ROBOSTAR

Introduction to proven software: formal proof

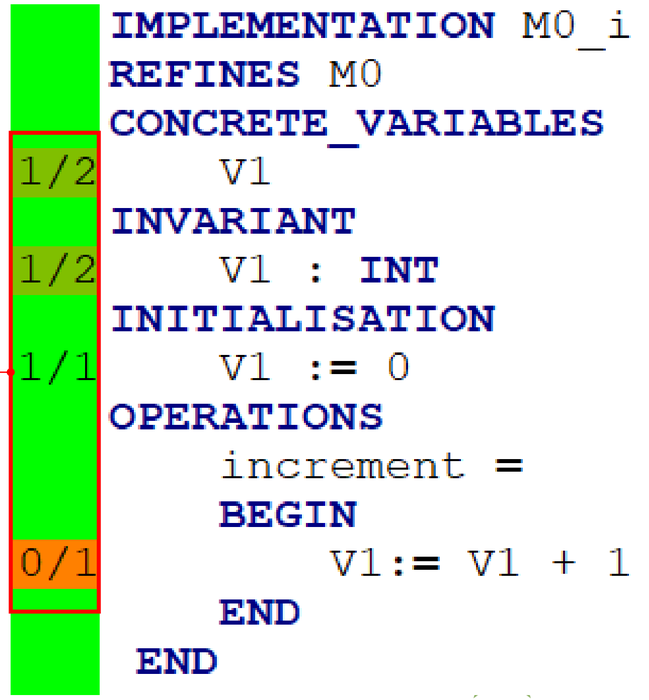


ROBOSTAR

Introduction to proven software: formal proof

IF GREEN, NO TESTING REQUIRED

Implementation



PROOF STATUS

NOT GREEN = NOT PROVED AUTOMATICALLY
EITHER WRONG OR NOT PROVABLE BY THE TOOL

ROBOSTAR

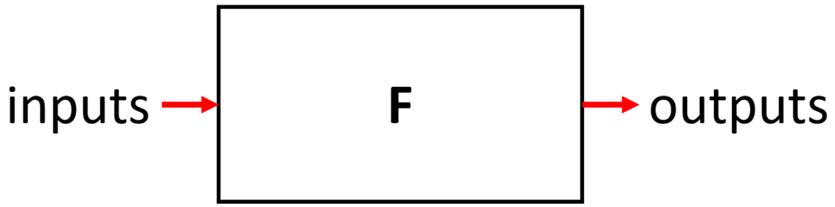
Introduction to the CLEARSY Safety Platform

Introduction to the CLEARSY Safety Platform

Summary of
https://youtu.be/A8uemvcelcU

What a Safety Computer is

Safety computer



- **F == (read inputs, compute, set outputs)***
- **F** could harm / kill people
- Ability to check if able to execute **F** properly

*: for space applications, could be a reset when computer hit by high energy particle

- F is not safe just because a safety computer is used

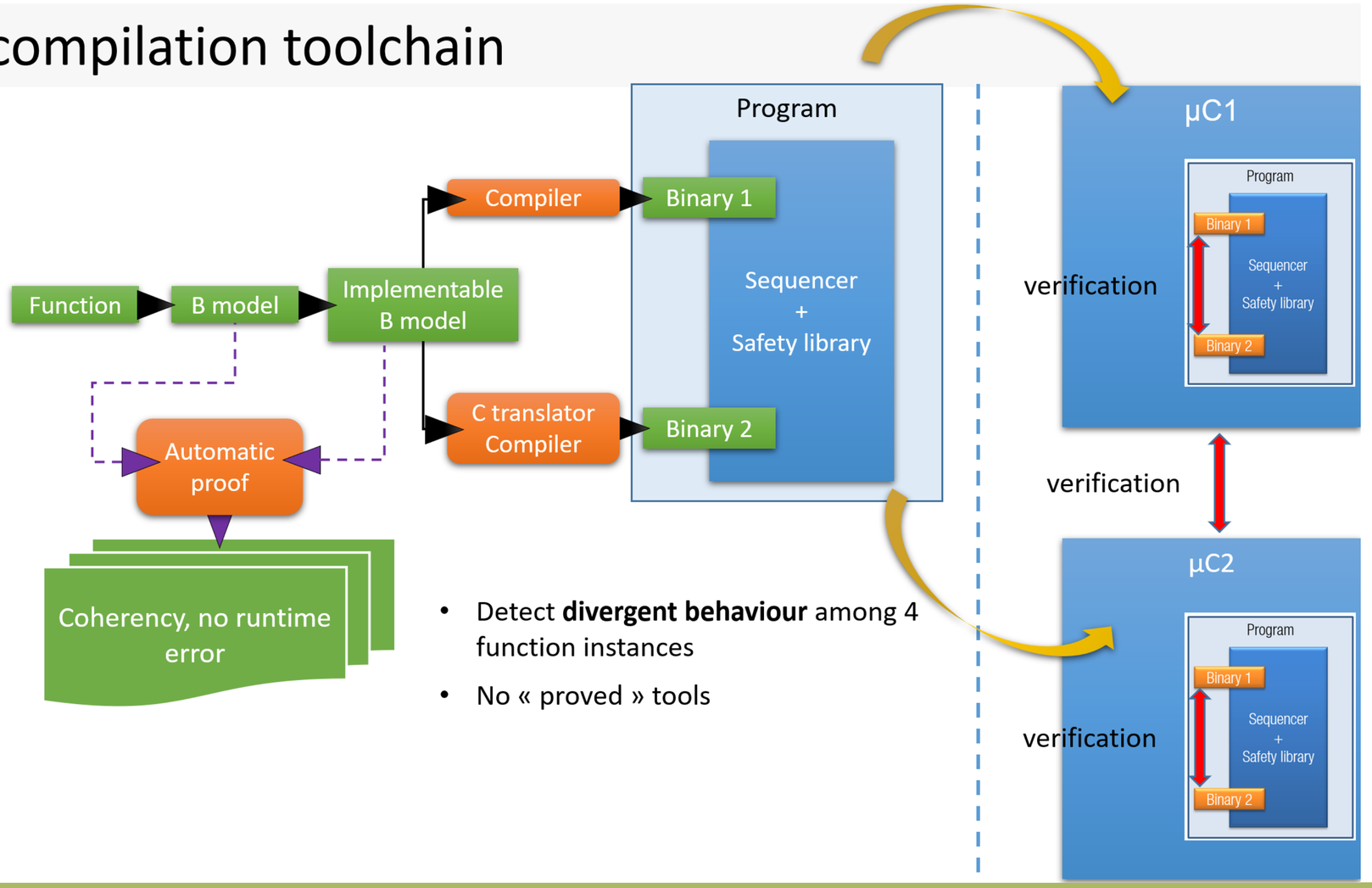
« execute the right F and execute the F right »

ROBOSTAR

Introduction to the CLEARSY Safety Platform

Summary of
https://youtu.be/A8uemvcelcU

Double compilation toolchain



- Detect **divergent behaviour** among 4 function instances
- No « proved » tools

ROBOSTAR

Introduction to the CLEARSY Safety Platform

Summary of
https://youtu.be/A8uemvcelcU

Verification

- **Starter kits for education**:
  - Programmed with B
  - SK0 available since Q1 2019: 5 digital I/O
  - SK0 software simulator (no safety)

Handle failures:

- **Systematic** (buggy code generator and compiler, etc.)
- **Random** (memory corruption, failing transistor, degrading clock, etc.)

Safety is built-in, out of reach of the developer who cannot alter it

ROBOSTAR

Introduction to the CLEARSY Safety Platform

Summary of
https://youtu.be/A8uemvcelcU

Summary

- CLEARSY Safety Platform
  - Safety computer
    - Execute 4 instances of the same function on 2 processors
    - Verify health regularly
    - Stop application execution and deactivate outputs if problem 
  - IDE
    - Application developed with B formal language
    - Model mathematically proved

ROBOSTAR

CLEARSY Safety Platform Programming Model

- The execution is cyclic
- The function is executed regularly as often as possible
  similar to arduino programming (setup(), loop())

```
init();

while (1) {
   instance1();
   instance2();
}
```

- No underlying operating system
- No interrupt()
- No predefined cycle time (if outputs are not set and cross read every **50ms**, board enters panic mode)
- No delay()
- Inputs are values captured at the beginning of a cycle (digital I/O)
- Outputs are maintained from one cycle to another (digital I/O)

- Project skeleton is generated from board description (I/O used, naming)
- Programming is specifying and implementing the function *user_logic*

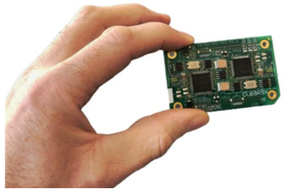
ROBOSTAR

CLEARSY Safety Platform Applications (Industrial Version)

Platform Screen Doors Controller

São Paulo, Monorail line 15
Stockholm, City Line
Brisbane, Cross River Rail

SIL4 Certified



CERTIFER

Design examination type certificate
Certificat de type par examen de la conception

N° 9594/0262 edition 2

Attributed to
Délivré à
CLEARSY
320 Av. Archimède - Pléiades III
F-13100 Aix-en-Provence
by
par
CERTIFER SA
18 Rue Edmond Membrée
F-59300 VALENCIENNES

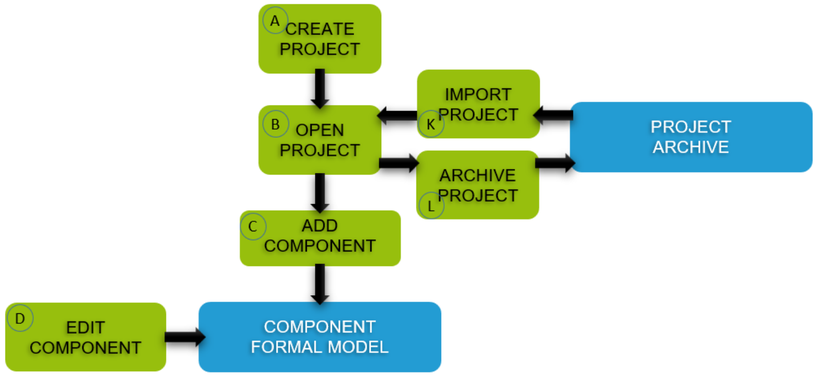
Which certifies that the design of the following product:
Qui certifie que la conception du produit suivant :
GENERIC PRODUCT
CLEARSY SAFETY PLATFORM
BASELINE 1.0.2
Meets SIL4 requirements of the standards
Est conforme aux exigences SIL4 des normes
CENELEC EN 50126:2017, EN 50129:2018, EN 50128:2011

Autonomous Train Localization

ROBOSTAR

Development Process

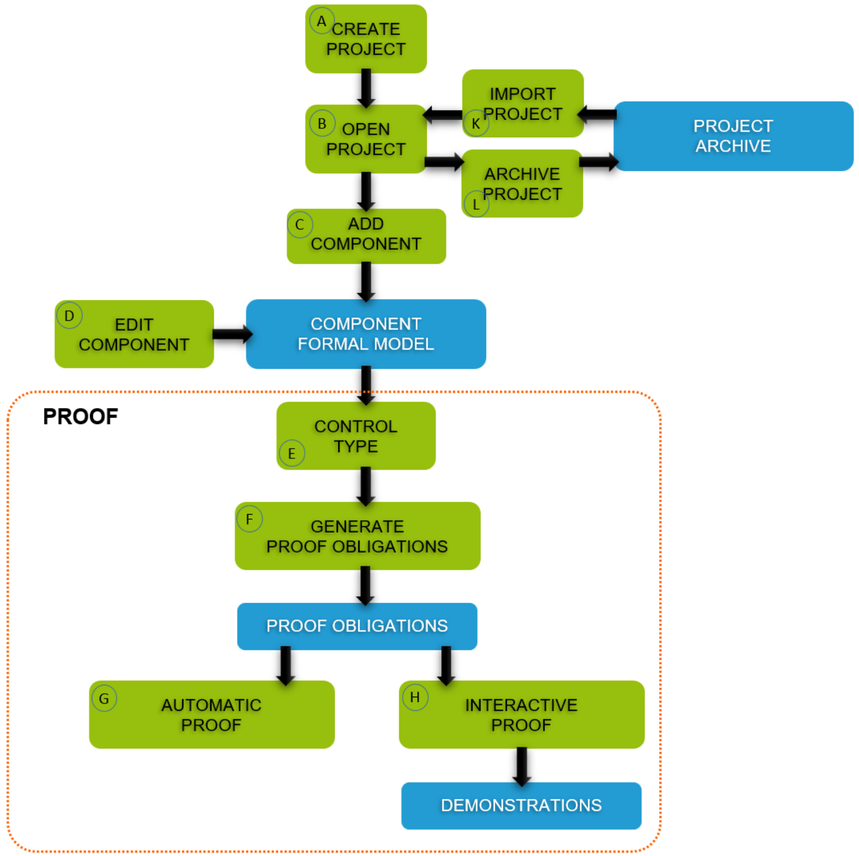
Development process



ROBOSTAR

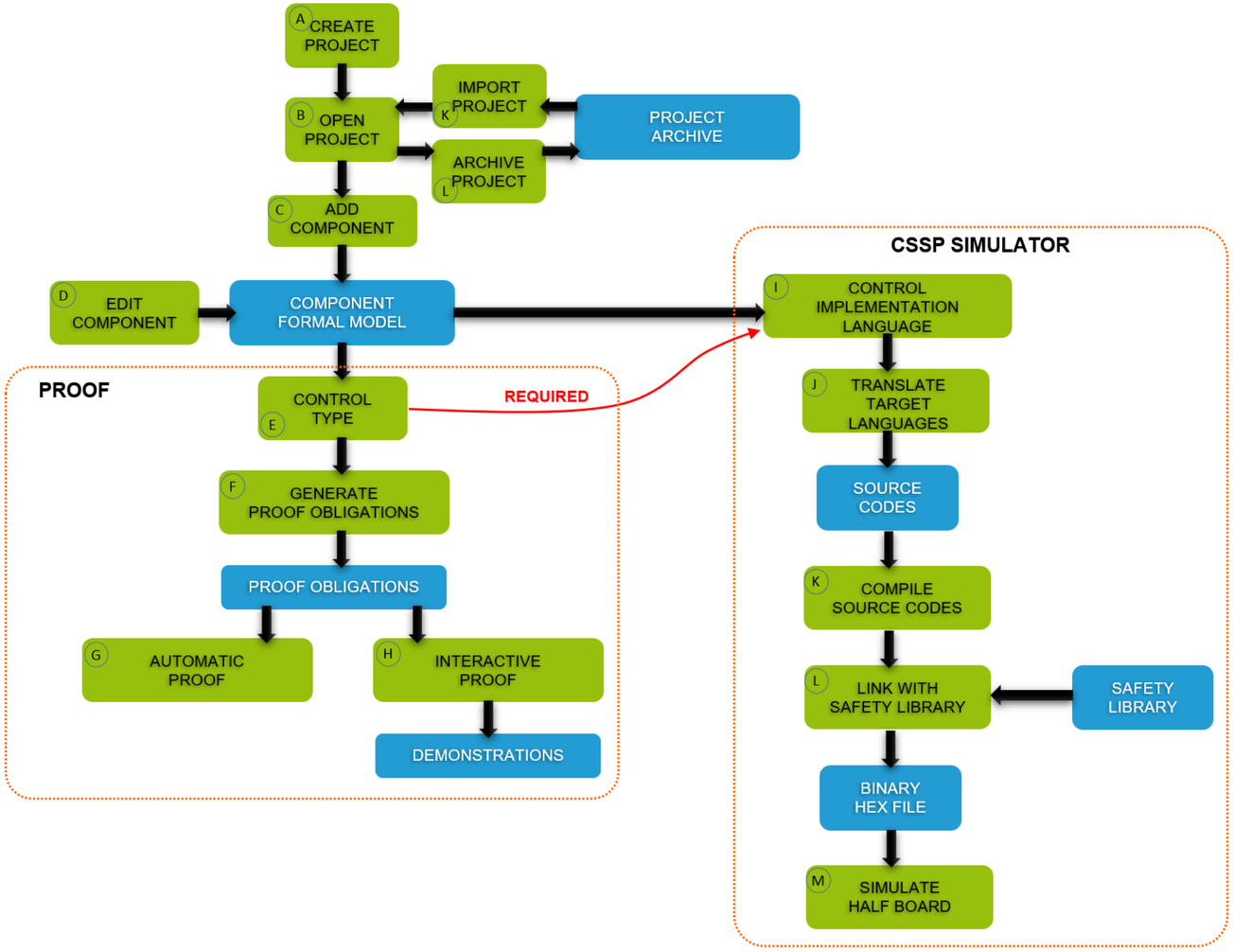
University of York, UK

Development process



ROBOSTAR

Development process



ROBOSTAR

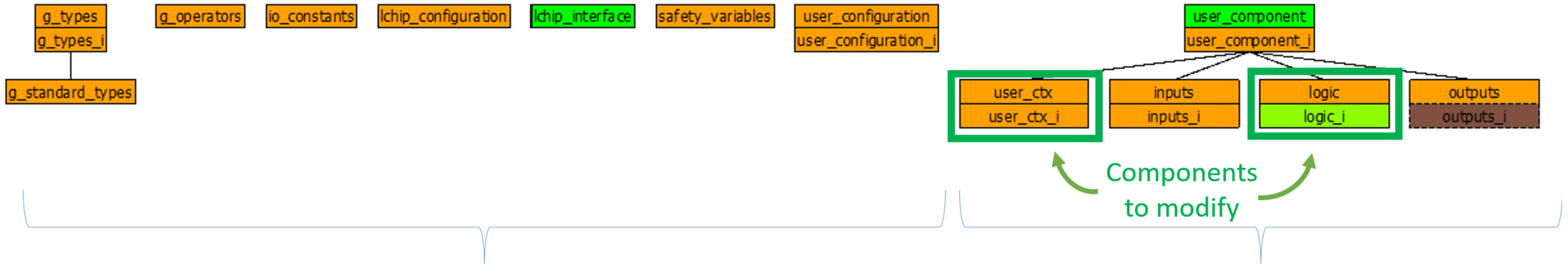
University of York, UK

CLEARSY Safety Platform Project

A CSSP project is a B project

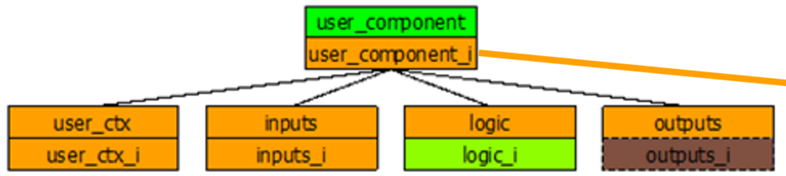
- is generated automatically from board configuration (# IOs, naming)

It contains



ROBOSTAR

Generated Models




```
user_app =
BEGIN
    divergence_test_var :=  0;
    read_inputs;
    user_logic;
    write_outputs
END;
```

Syntax:
pp <-- ff(vv)
represents a call to operation ff(vv)
that returns the value pp

call

call

call

```
read_inputs  =
BEGIN
    I0 <-- read_global_input(0);
    I1 <-- read_global_input(1);
    I2 <-- read_global_input(2)
END;

po <-- get_I0 =
BEGIN
    po <-- read_global_input(0)
END;

po <-- get_I1 =
BEGIN
    po <-- read_global_input(1)
END;

po <-- get_I2 =
BEGIN
    po <-- read_global_input(2)
END
```

used by

```
user_logic = skip;

po <-- get_O0 =
BEGIN
    po := O0
END;

po <-- get_O1 =
BEGIN
    po := O1
END
```

used by

used by

```
write_outputs =
VAR
    lsb
IN
    lsb:(lsb : uint8_t);

    lsb <-- get_O0;
    write_global_output(0, lsb);

    lsb <-- get_O1;
    write_global_output(1, lsb)
END
```

Programming CSSP is mainly implementing *user_logic* OPERATION

- Reading inputs values with *get_IO*, *get_I1* and *get_I2*
- Modifying the variables *O0* and *O1*

ROBOSTAR

Bits of B

Bits of B : B Variables Declaration

specification

```
ABSTRACT_VARIABLES
    O0,
    O1

INVARIANT
    O0 : uint8_t &
    O1 : uint8_t

INITIALISATION
    O0 :: uint8_t ||
    O1 :: uint8_t
```

: means « belongs to »

|| means « in parallel », « at the same time »

:: means « any value within »

implementation

```
// pragma SAFETY_VARS

CONCRETE_VARIABLES
    O0,
    O1,
    TIME_A,
    STATUS

INVARIANT
    O0 : uint8_t &
    O1 : uint8_t &
    TIME_A : uint32_t &
    STATUS : uint8_t

INITIALISATION
    O0 := IO_OFF;
    O1 := IO_OFF;
    TIME_A := 0;
    STATUS := SFALSE
```

Mandatory
Contains variables that will be verified

Variables local to implementation

ROBOSTAR

Bits of B : B Constants Declaration

specification

```
CONCRETE_CONSTANTS
    DELTA_T

PROPERTIES
    DELTA_T : uint32_t
```

implementation

```
// pragma CONSTANTS
```

Mandatory
Contains constants that will be verified

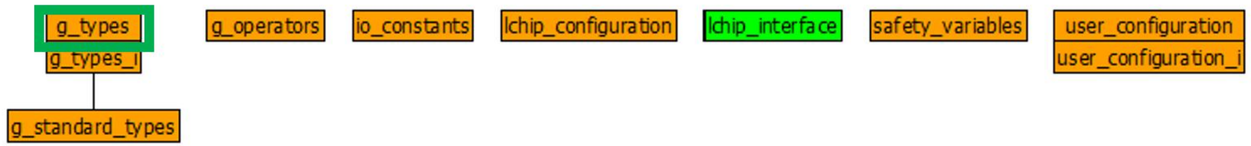
Important
A model cannot contain both variables and constants

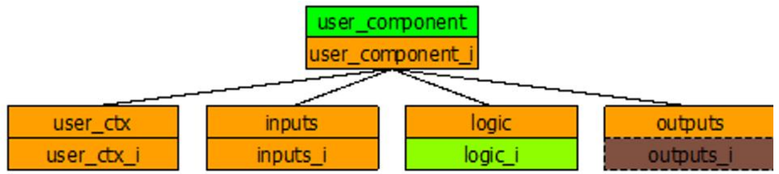
```
VALUES
    DELTA_T = 1000 // 1000 ms == 1s
```

Value that should enforce the properties

ROBOSTAR

Bits of B : CLEARSY Safety Platform Supported Types






```
CONCRETE_CONSTANTS
    uint32_t ,
    uint16_t ,
    uint8_t ,

    STRUE ,
    SFALSE ,
    MAX_UINT32 ,
    MAX_UINT16 ,
    MAX_UINT8
```

Everything is either 8, 16 or 32 bits

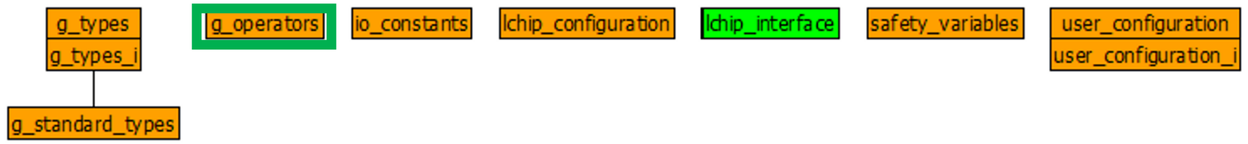
Boolean values TRUE and FALSE coded on 8 bits

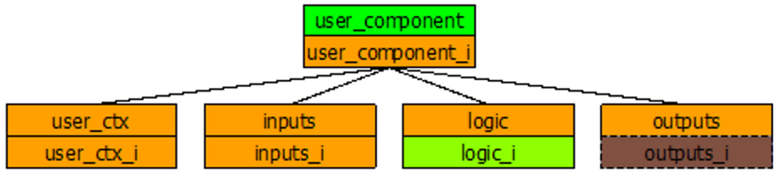
The real values for STRUE and SFALSE are not displayed but we know that

```
PROPERTIES
    uint32_t = 0 .. 4294967295 &
    uint16_t = 0 .. 65535 &
    uint8_t  = 0 .. 255  &

    MAX_UINT32 : uint32_t &
    MAX_UINT16 : uint16_t &
    MAX_UINT8  : uint8_t  &

    STRUE  : uint8_t &
    SFALSE : uint8_t &

    MAX_UINT32 = 4294967295 &
    MAX_UINT16 = 65535      &
    MAX_UINT8  = 255  &

    STRUE  : 0 .. MAX_UINT8 &
    SFALSE : 0 .. MAX_UINT8 &

    STRUE /= SFALSE  &
    SBOOL = { STRUE , SFALSE } &
    STRUE <= 2 &
    SFALSE <= 2 &
```

ROBOSTAR

Bits of B : Unsigned INT Operators






```
CONCRETE_CONSTANTS
    bitwise_not_uint32,
    bitwise_and_uint32,
    bitwise_xor_uint32,
    bitwise_not_uint16,
    bitwise_and_uint16,
    bitwise_xor_uint16,
    bitwise_or_uint16,
    bitwise_not_uint8,
    bitwise_and_uint8,
    bitwise_xor_uint8,
    bitwise_or_uint8,
    add_uint32,
    sub_uint32,
    mul_uint32,
    add_uint16,
    sub_uint16,
    mul_uint16,
    add_uint8,
    sub_uint8,
    mul_uint8
```

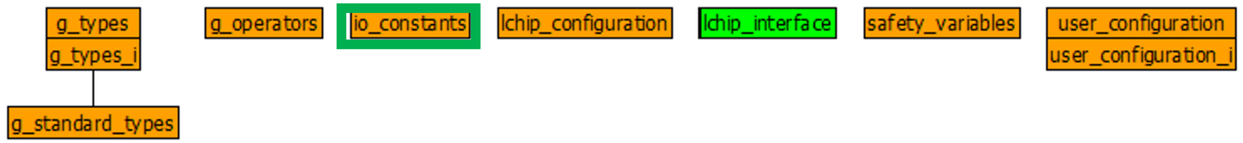
Builtin operators

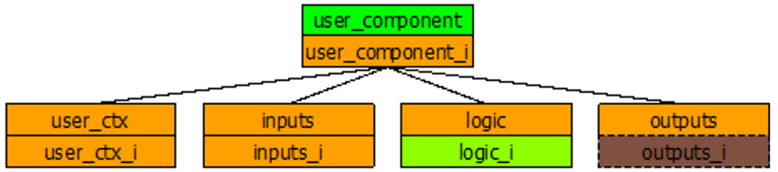
```
add_uint32 : uint32_t * uint32_t --> uint32_t &
sub_uint32 : uint32_t * uint32_t --> uint32_t &
mul_uint32 : uint32_t * uint32_t --> uint32_t &
add_uint16 : uint16_t * uint16_t --> uint16_t &
sub_uint16 : uint16_t * uint16_t --> uint16_t &
mul_uint16 : uint16_t * uint16_t --> uint16_t &
add_uint8 : uint8_t * uint8_t --> uint8_t &
sub_uint8 : uint8_t * uint8_t --> uint8_t &
mul_uint8 : uint8_t * uint8_t --> uint8_t &
```

Operators that could lead to overflow

ROBOSTAR

Bits of B : Inputs/outputs






```
ABSTRACT_CONSTANTS
    TIME,
    IO_STATE

CONCRETE_CONSTANTS
    IO_ON,
    IO_OFF

PROPERTIES
    TIME = uint32_t &
    IO_STATE = uint8_t &

    IO_ON  : uint8_t &
    IO_OFF : uint8_t &
    IO_ON /= IO_OFF &
    IO_ON : IO_STATE &
    IO_OFF : IO_STATE
```

inputs and outputs state

values used by digital inputs and outputs

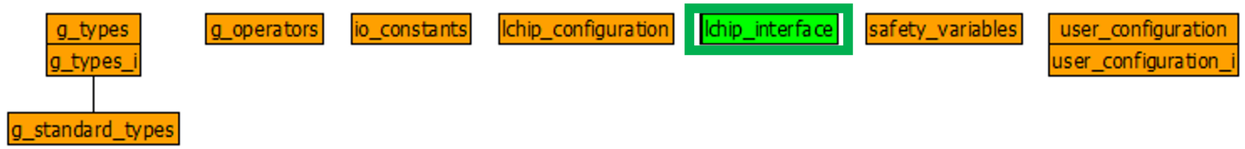
coded on 8 bits

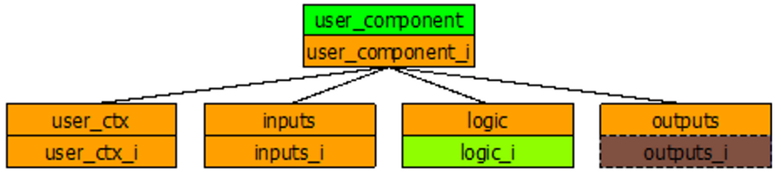
Verification

If a digital output is valued with a value different from IO_ON or IO_OFF then SK_0 stops in error mode

ROBOSTAR

Bits of B : Time






```
out <-- get_ms_tick =
PRE
    out : uint32_t
THEN
    out := ms_tick
END
```

Returns the number of milliseconds elapsed since last reboot

Bits of B: B Operations

Operations are populated with substitutions

Available substitutions in specification are different from the ones available in implementation

specification

Express the properties that the variables comply with <u>when the operation is completed</u> independently from the algorithm implemented (*post-condition*)

To simplify, always use « becomes such that substitutions »

```
user_logic  =
    BEGIN
        o0, o1 : (
            o0 : uint8_t &
            o1 : uint8_t &
            not(o0 = o1)
        )
    END;
```

Typing (mandatory)

Constraints (optional)

ROBOSTAR

Bits of B: Implementation

implementation

```
user_logic = skip;
```
do nothing

```
user_logic =
BEGIN
    O0 := IO_ON;
    O1 := IO_OFF
END;
```
valuations in sequence

```
user_logic =
BEGIN
    IF Var8 = 0 THEN
        O0 := IO_ON
    ELSE
        O1 := IO_ON
    END
END;
```
IF THEN ELSE

Important
Only single condition (no conjonction nor disjonction)
= < <= operators only

```
user_logic =
BEGIN
    VAR time_ IN
        time_ : (time_ : uint32_t);
        time_ <-- get_ms_tick;
        IF 2000 <= time_ THEN
            O1 := IO_ON
        END
    END
END;
```
Local variables declaration
Operation call

Important
Local variables have to be typed first using « becomes such that » substitution

Contraints on the language to simplify the compiler

ROBOSTAR

Bits of B: user_logic

specification

```
user_logic  = skip;
```

skip means « do no alter the variables of the model »

```
MACHINE
    logic

SEES
    g_types,
    g_operators,
    io_constants,
    lchip_interface

ABSTRACT_VARIABLES
    O1,
    O2

INVARIANT
    O1 : uint8_t &
    O2 : uint8_t

INITIALISATION
    O1 :: uint8_t ||
    O2 :: uint8_t

OPERATIONS
    user_logic  = skip;

    po <-- get_O1 =
    PRE
        po : uint8_t
    THEN
        po := O1
    END;

    po <-- get_O2 =
    PRE
        po : uint8_t
    THEN
        po := O2
    END
END
```

implementation

```
user_logic  = skip;
```

Minimum example:

- do nothing; outputs remain in their initial state (INITIALISATION)

```
IMPLEMENTATION logic_i

REFINES logic

SEES
    g_types,
    g_operators,
    io_constants,
    lchip_interface,
    inputs

    // pragma SAFETY_VARS

CONCRETE_VARIABLES
    O1,
    O2

INVARIANT
    O1 : uint8_t &
    O2 : uint8_t

INITIALISATION
    O1 := IO_OFF;
    O2 := IO_OFF

OPERATIONS
    user_logic = skip;

    po <-- get_O1 =
    BEGIN
        po := O1
    END;

    po <-- get_O2 =
    BEGIN
        po := O2
    END
END
```

ROBOSTAR

Bits of B: user_logic

specification

```
user_logic  =
BEGIN
    O0 :: uint8_t ||
    O1 :: uint8_t
END
```

O0 and O1 belong to their type

```
user_logic  =
    BEGIN
        O0, O1 : (
            O0 : uint8_t &
            O1 : uint8_t &
            not(O0 = O1)
        )
    END
```

:( ) means « becomes such that »

O0 and O1 belong to their type
and O0 is different from O1

```
user_logic  =
BEGIN
    O0 := IO_ON ||
    O1 := IO_OFF
END
```

Set O0 and reset O1

implementation

```
user_logic =
BEGIN
    O0 := IO_ON;
    O1 := IO_OFF
END
```

Set O0 then reset O1

« then » is related to the valuation of O0 regarding O1
O0 and O1 will be positioned at the same time at the end of the cycle

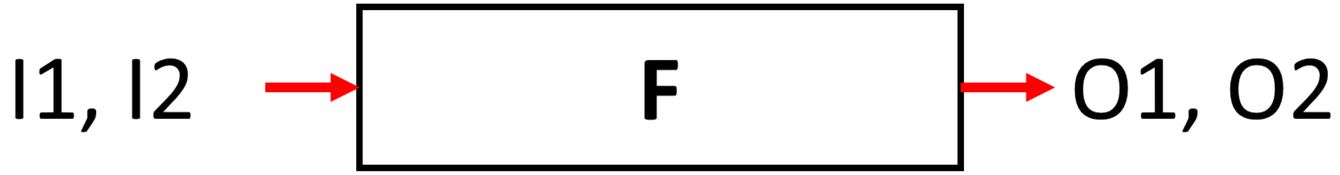
ROBOSTAR

Using the Modelling Interface

Using the modelling interface

- I1, I2, O1, O2 belongs to {IO_OFF, IO_ON}
- O1 is IO_ON *iff* both I1 and I2 are IO_ON
- O2 is the complement of O1

Using the modelling interface



- I1, I2, O1, O2 belongs to {IO_OFF, IO_ON} — is unsigned 8 bit integer enumeration
- O1 is IO_ON *iff* both I1 and I2 are IO_ON
- O2 is the complement of O1

```
user_logic =
BEGIN
    O1, O2: (
        O1 : uint8_t &
        O2 : uint8_t
    )
END;
```

« 01, 02 become such that
- Type of O1 is unsigned 8 bit integer
- Type of O2 is unsigned 8 bit integer »

Minimum specification :« 01 and 02 are modified in accordance with their type »

ROBOSTAR

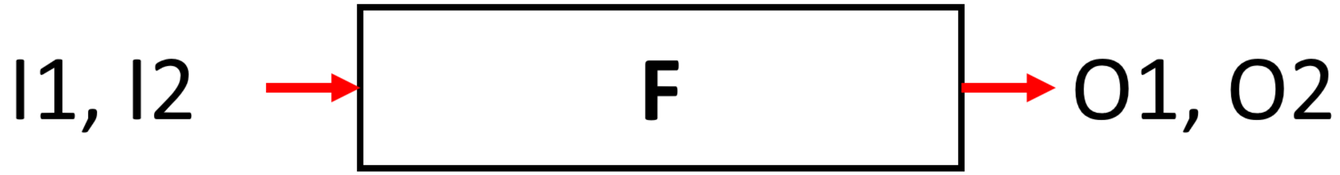
Using the modelling interface

- I1, I2, O1, O2 belongs to {IO_OFF, IO_ON}
- O1 is IO_ON *iff* both I1 and I2 are IO_ON
- O2 is the complement of O1

```
user_logic =
BEGIN
    O1, O2: (
        O1 : uint8_t &
        O2 : uint8_t &
        (O1=IO_ON <=> (I1=IO_ON & I2=IO_ON)) &
        not(O1 = O2)
    )
END
```

ROBOSTAR

Using the modelling interface



Green means « Implementation Fully proved Against Specification »

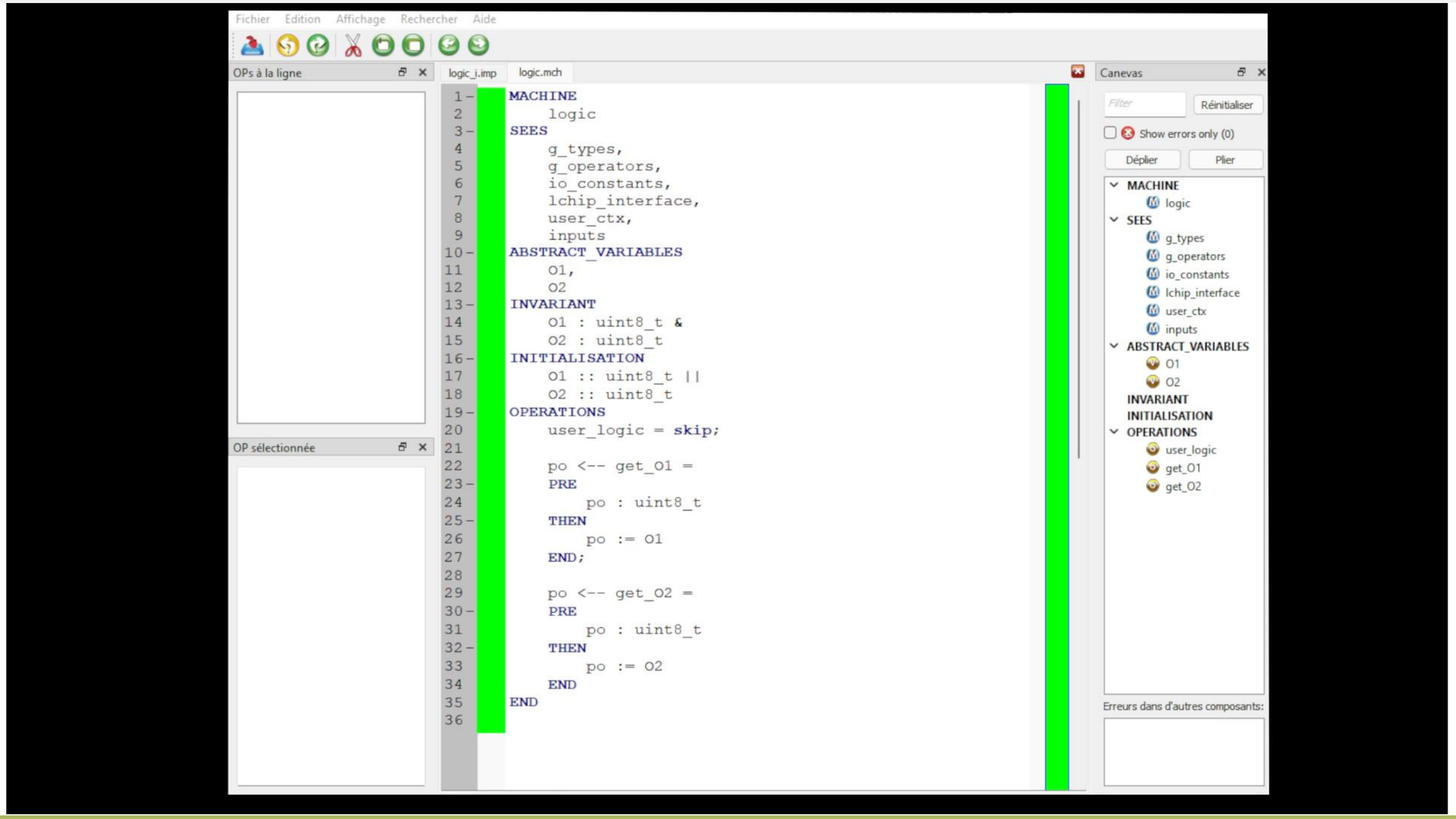
```
user_logic =
BEGIN
    VAR i1_, i2_ IN
        i1_ : (i1_ : uint8_t);
        i2_ : (i2_ : uint8_t);

        i1_ <-- get_I1;
        i2_ <-- get_I2;

        O1 := IO_OFF;
        O2 := IO_ON;
        IF i1_ = IO_ON THEN
            IF i2_ = IO_ON THEN
                O1 := IO_ON;
                O2 := IO_OFF
            END
        END
    END
END
```

1/1
1/1

Get I1 and I2 values

ROBOSTAR

Using the modelling interface



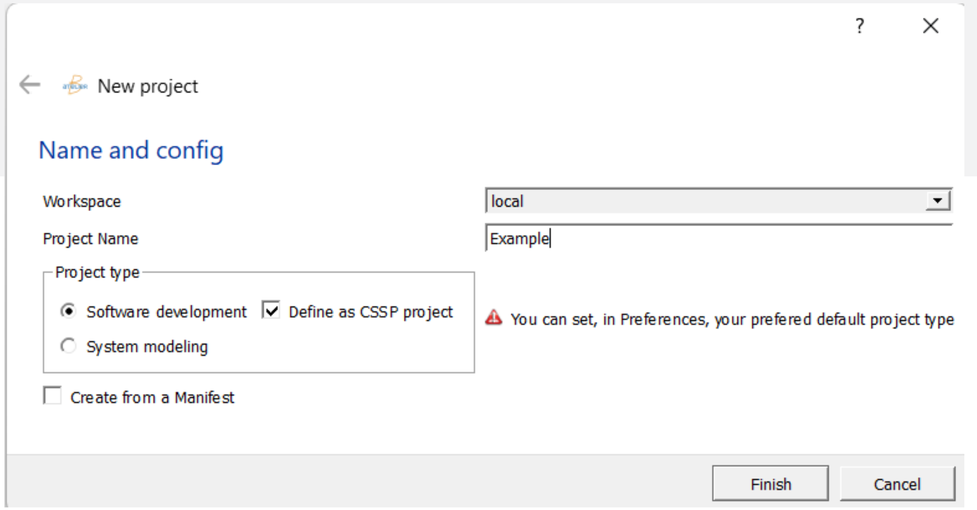
ROBOSTAR

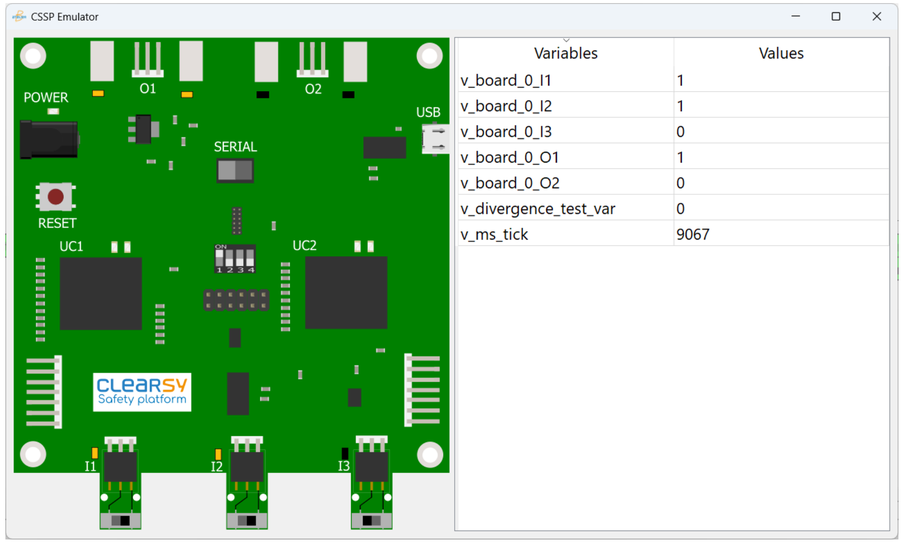
Using the modelling interface

- Edit logic.mch
- Edit logic_i.imp

- Save to check if the model
  - has a correct type
  - is proved

- Start the simulator (Ctrl+D)
  - Once the simulator has started, click on the inputs to observe the expected behaviour





ROBOSTAR

Next

Programming
The CLEARSY Safety Platform

Paulo Bezerra, Thierry Lecomte

ROBOSTAR