

Technical Session 3

The CLEARSY Safety Platform

Thierry Lecomte



11th March 2025



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

University of York, UK

Introduction to Proven Software

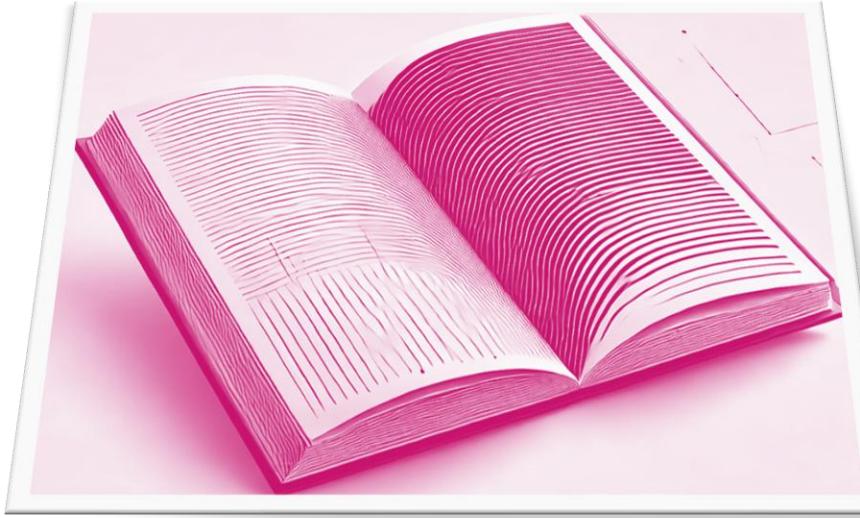


ROBOSTAR

University of York, UK

Introduction to proven software: code manually produced and tested

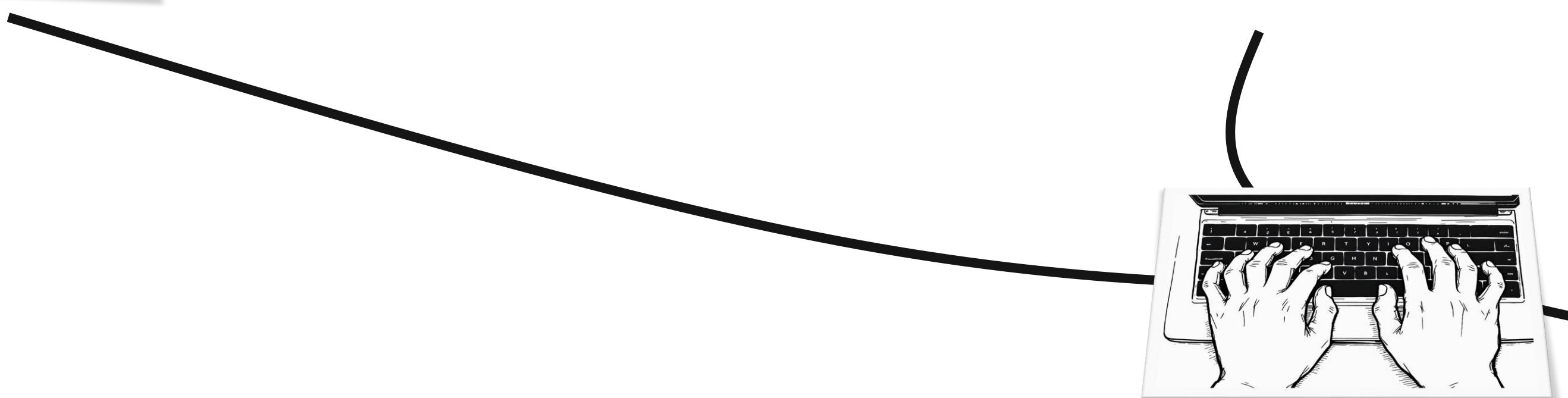
requirements



code

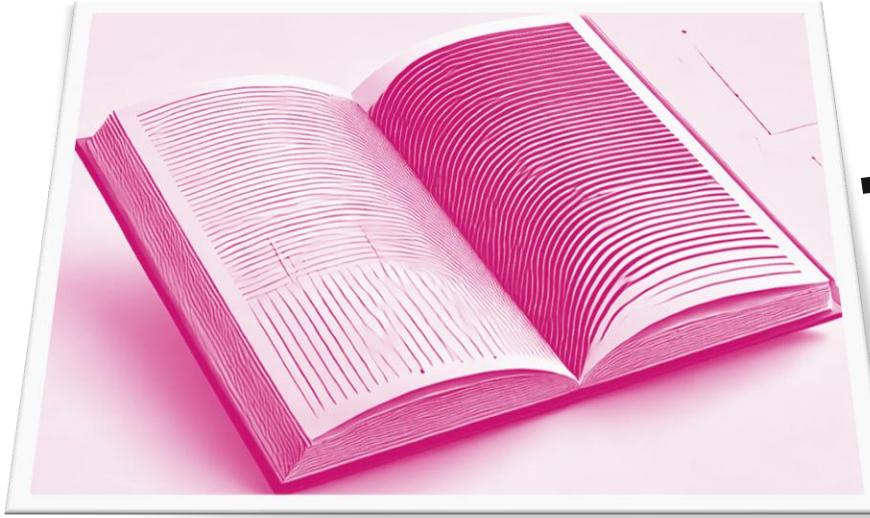


tests



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

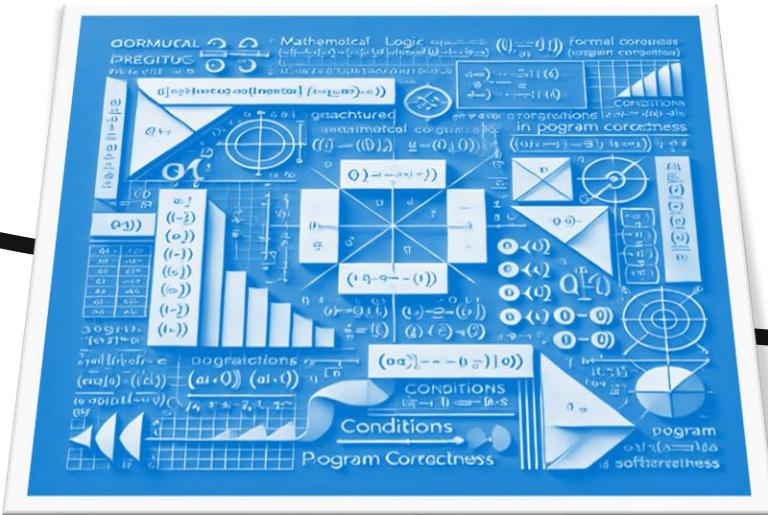
requirements



code

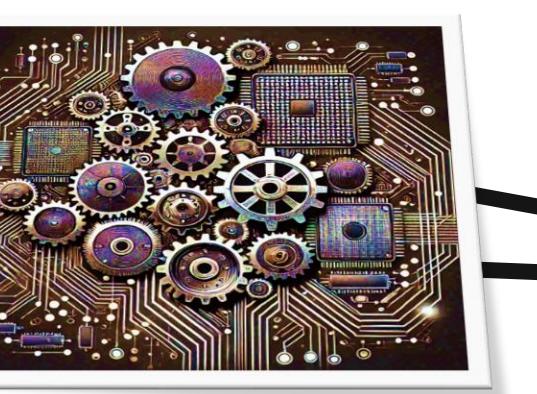


assertions

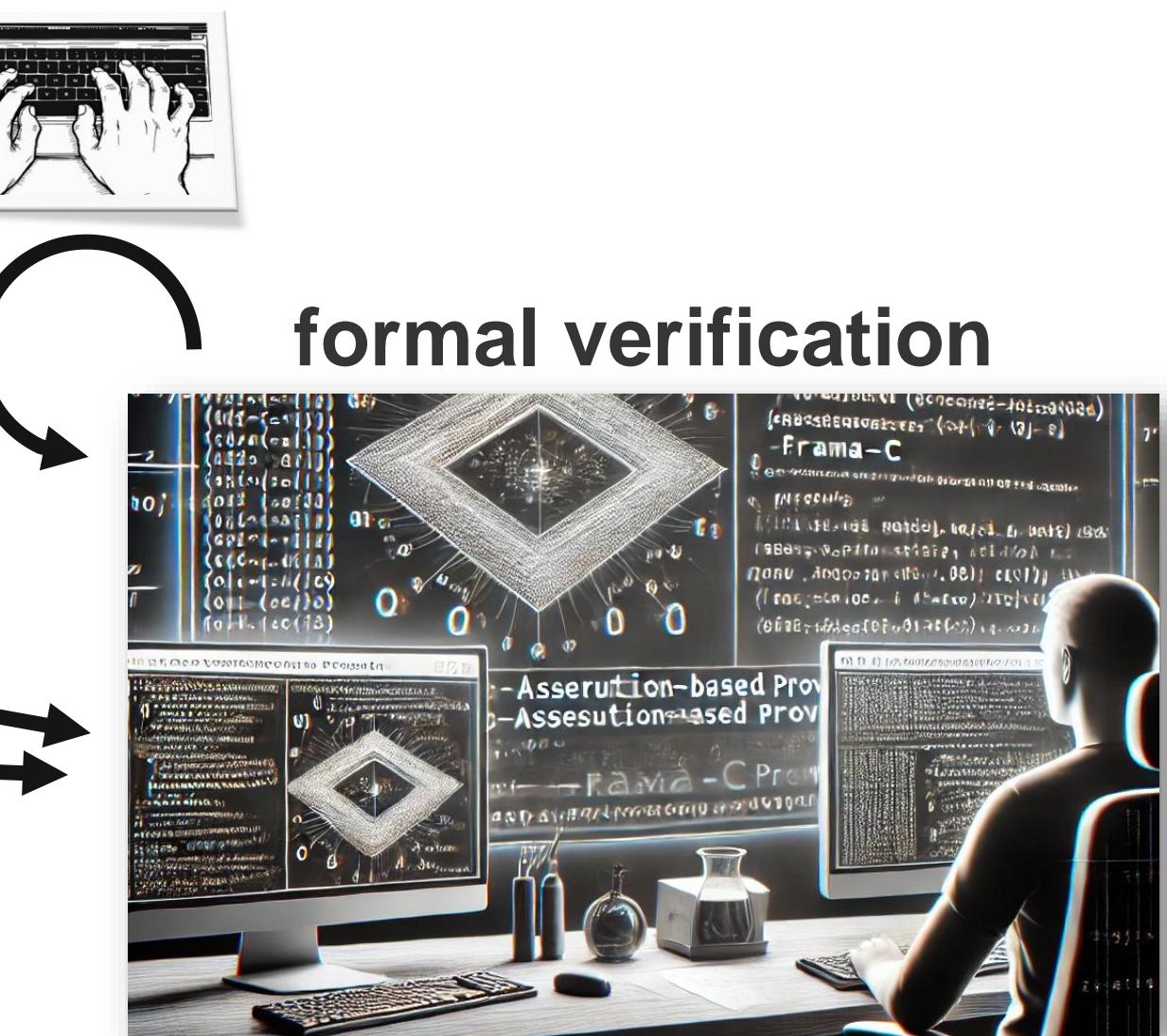


Assertions are mathematical predicates
for hypotheses and properties to verify

verifier

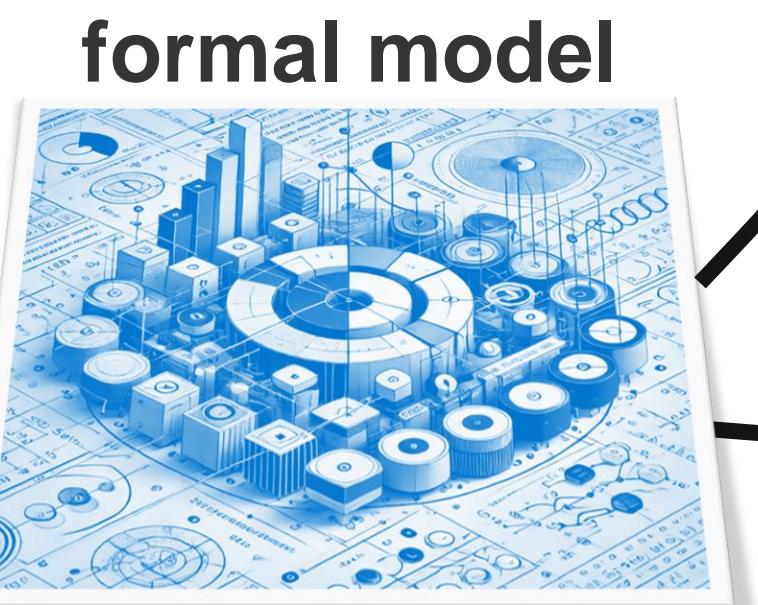
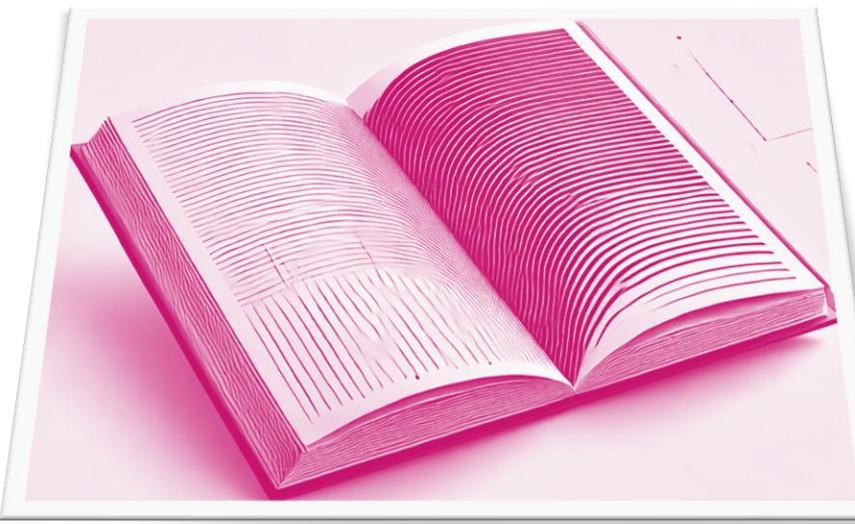


formal verification



Introduction to proven software: code formally specified and verified

requirements



Formal model use mathematical notation
to specify and implement behaviour

code



code generator



prover



formal proof



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

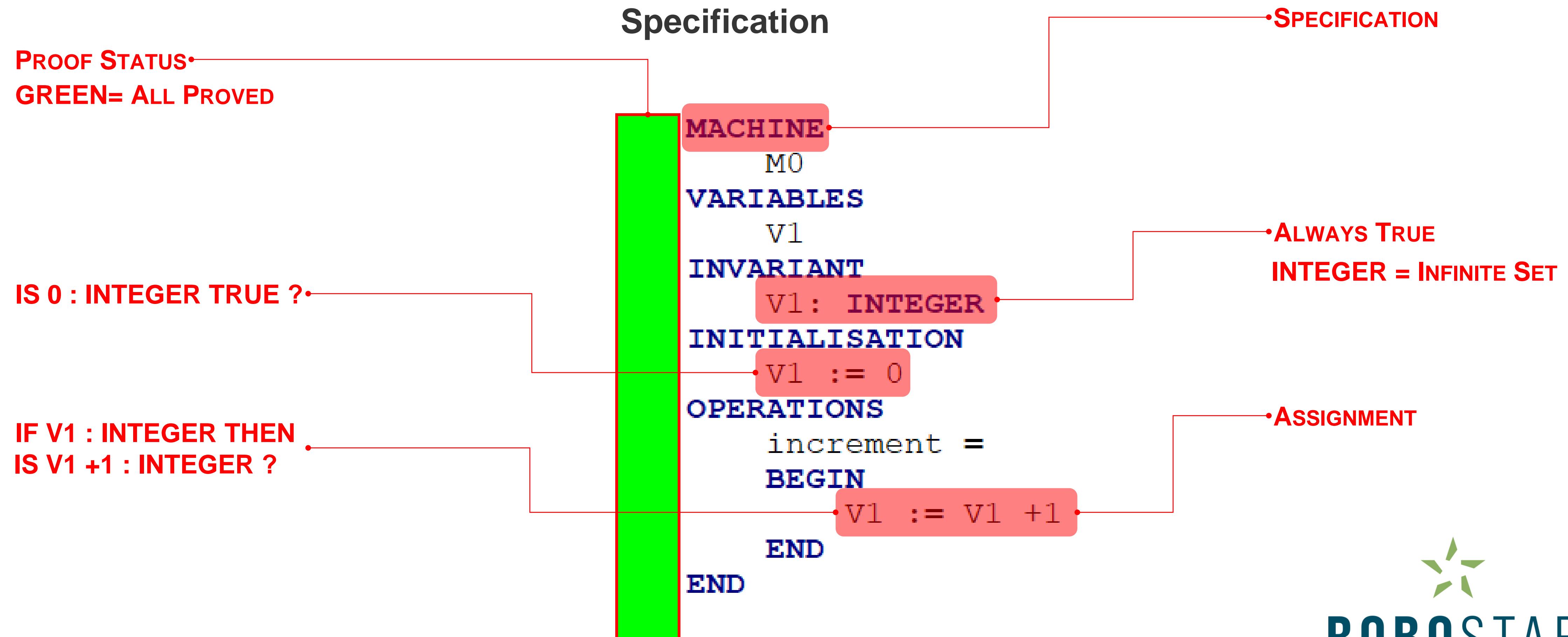
```
MACHINE  
M0  
VARIABLES  
    V1  
INVARIANT  
    V1: INTEGER  
INITIALISATION  
    V1 := 0  
OPERATIONS  
    increment =  
    BEGIN  
        V1 := V1 +1  
    END  
END
```

•SPECIFICATION

•TYPING
INTEGER = INFINITE SET

•ASSIGNMENT

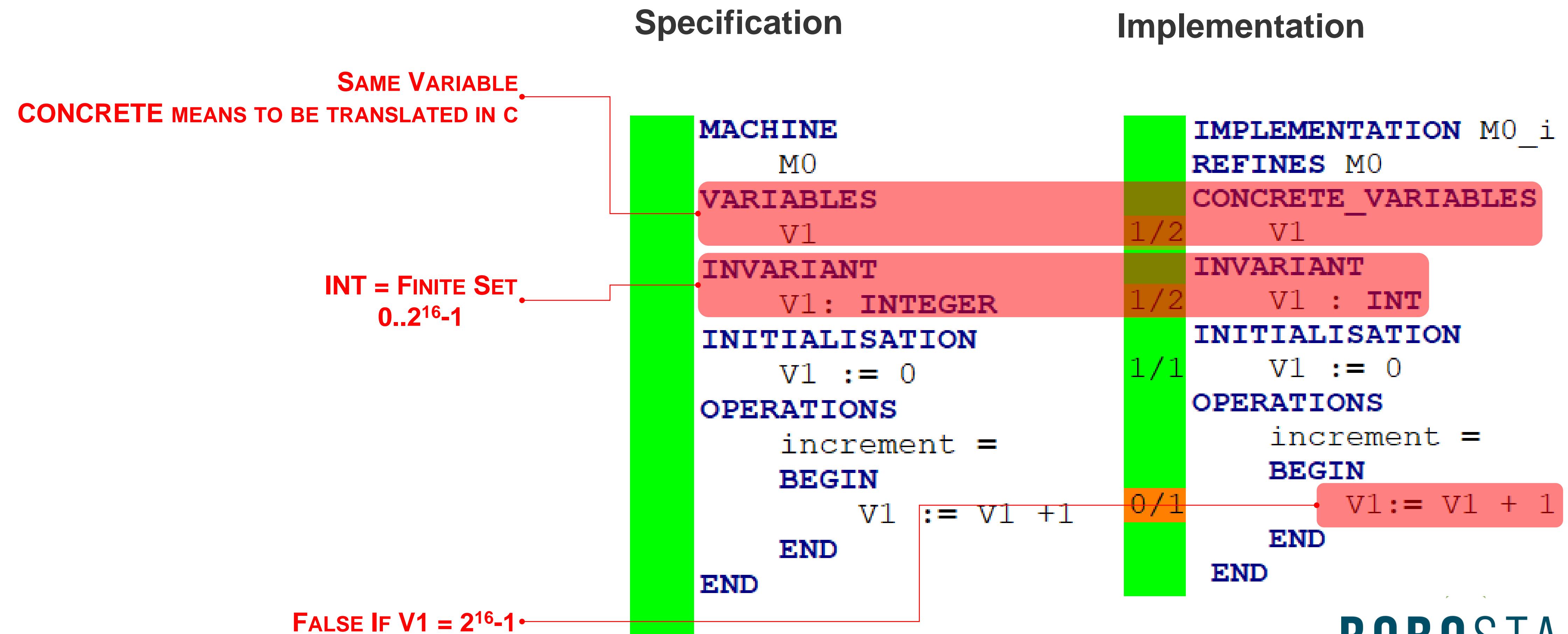
Introduction to proven software: formal proof



ROBOSTAR

University of York, UK

Introduction to proven software: formal proof



ROBOSTAR

University of York, UK

Introduction to proven software: formal proof

IF GREEN, NO TESTING REQUIRED

PROOF STATUS

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

Implementation

```
IMPLEMENTATION M0_i
REFINES M0
CONCRETE_VARIABLES
    V1
INVARIANT
    V1 : INT
INITIALISATION
    V1 := 0
OPERATIONS
    increment =
BEGIN
    V1 := V1 + 1
END
END
```

ROBOSTAR

University of York, UK

Introduction to the **CLEARSY Safety Platform**



ROBOSTAR

University of York, UK

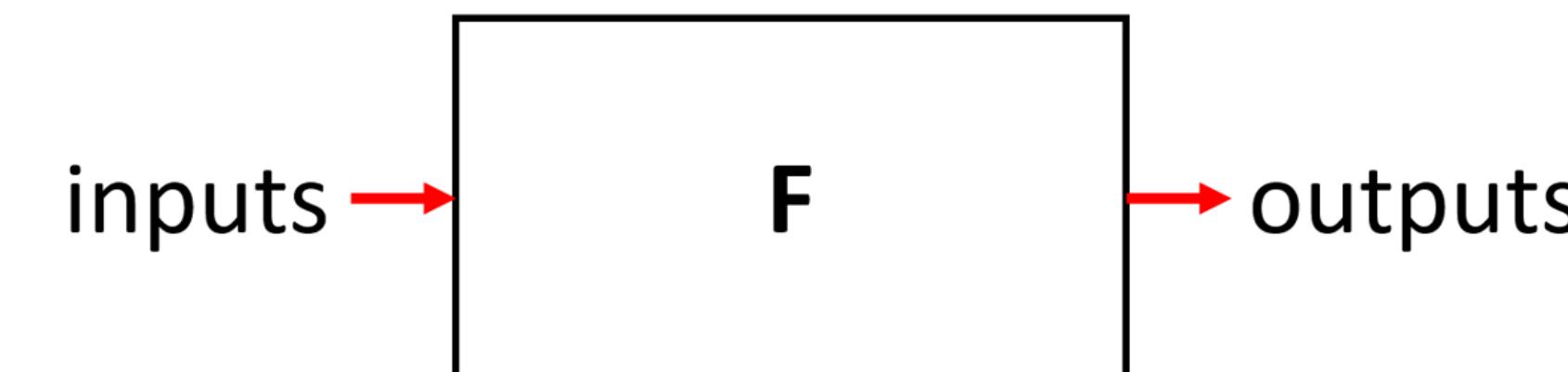
Introduction to the CLEARSY Safety Platform

Summary of
<https://youtu.be/A8uemvcclcU>

What a Safety Computer is

Safety computer

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



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

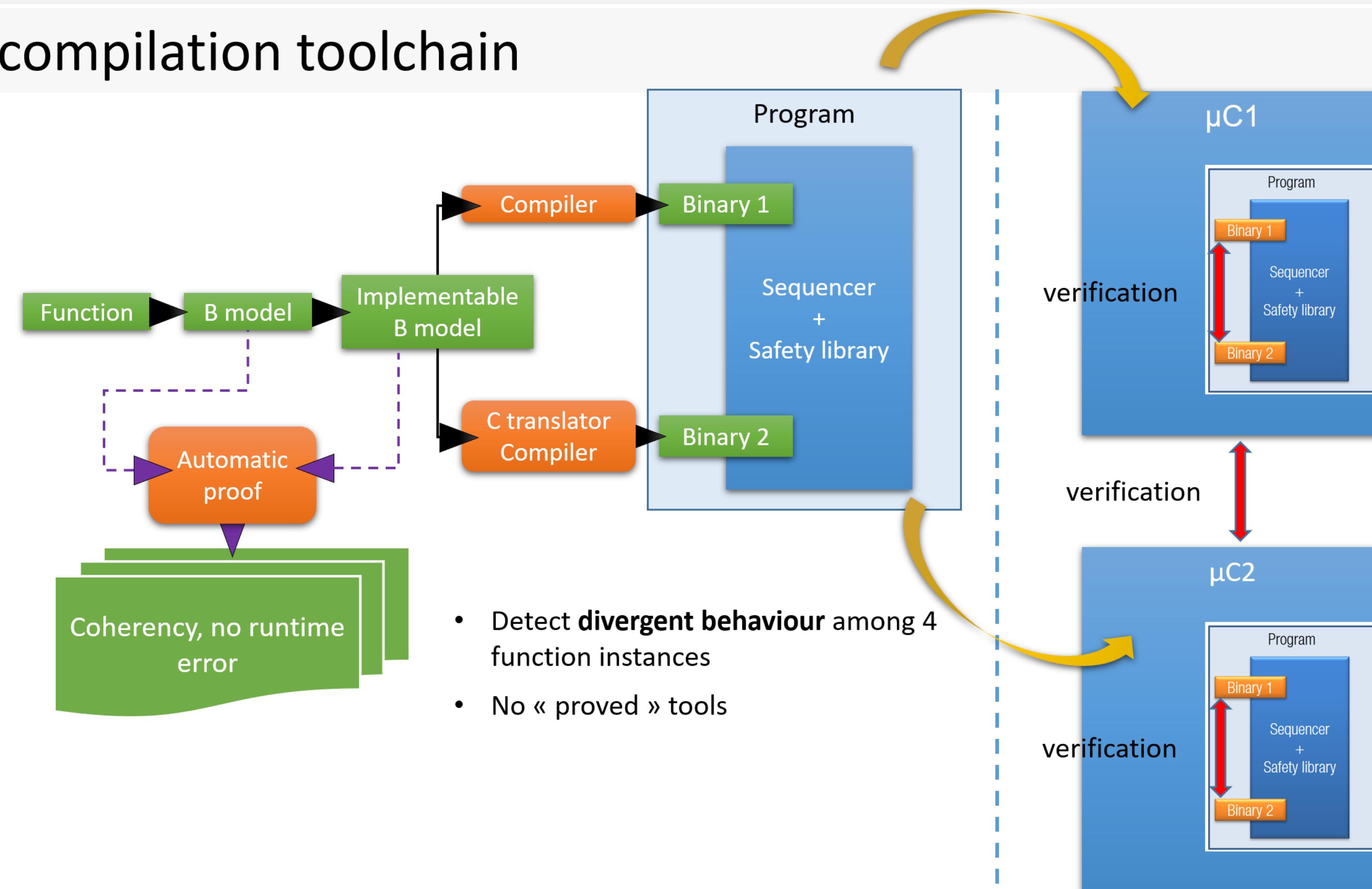
« execute the right F and execute the F right »

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

Introduction to the CLEARSY Safety Platform

Summary of
<https://youtu.be/A8uemvcclcU>

Double compilation toolchain



Introduction to the CLEARSY Safety Platform

Summary of
<https://youtu.be/A8uemvcclcU>

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

Introduction to the CLEARY Safety Platform

Summary of
<https://youtu.be/A8uemvcclcU>

Summary

- CLEARY 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



CLEARSY Safety Platform Programming Model

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

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



ROBOSTAR

University of York, UK

CLEARSY Safety Platform Applications (Industrial Version)

Platform Screen Doors Controller



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



Underwater Drone Localization



Autonomous Train Localization



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

N° 9594/0262 edition 2

Attributed to

Délivré à

CLEARSY

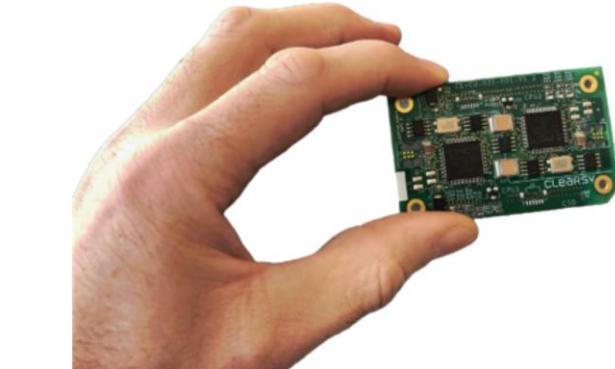
320 Av. Archimède - Pléiades III
F-13100 Aix-en-Provence

by

par

CERTIFER SA
18 Rue Edmond Membrey
F-59300 VALENCIENNES

SIL4 Certified



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

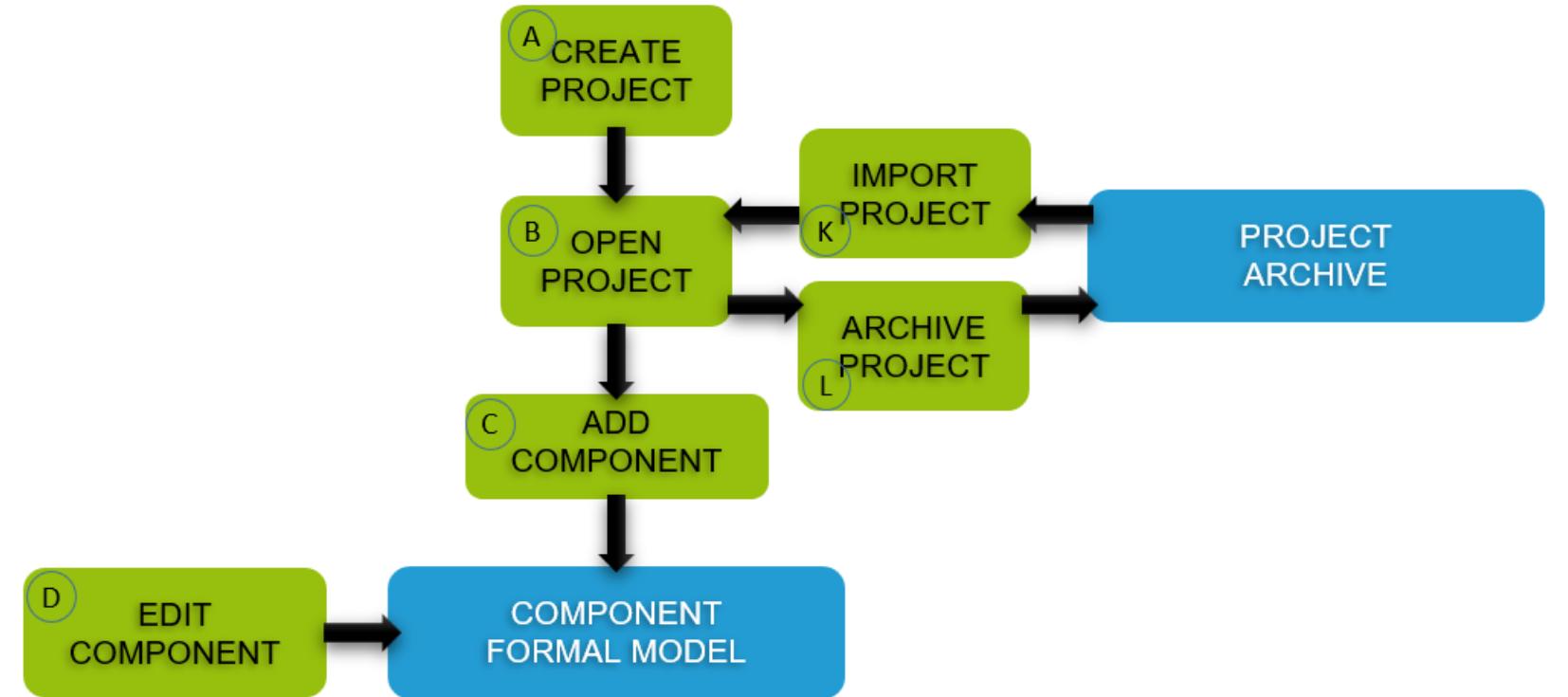
Development Process



ROBOSTAR

University of York, UK

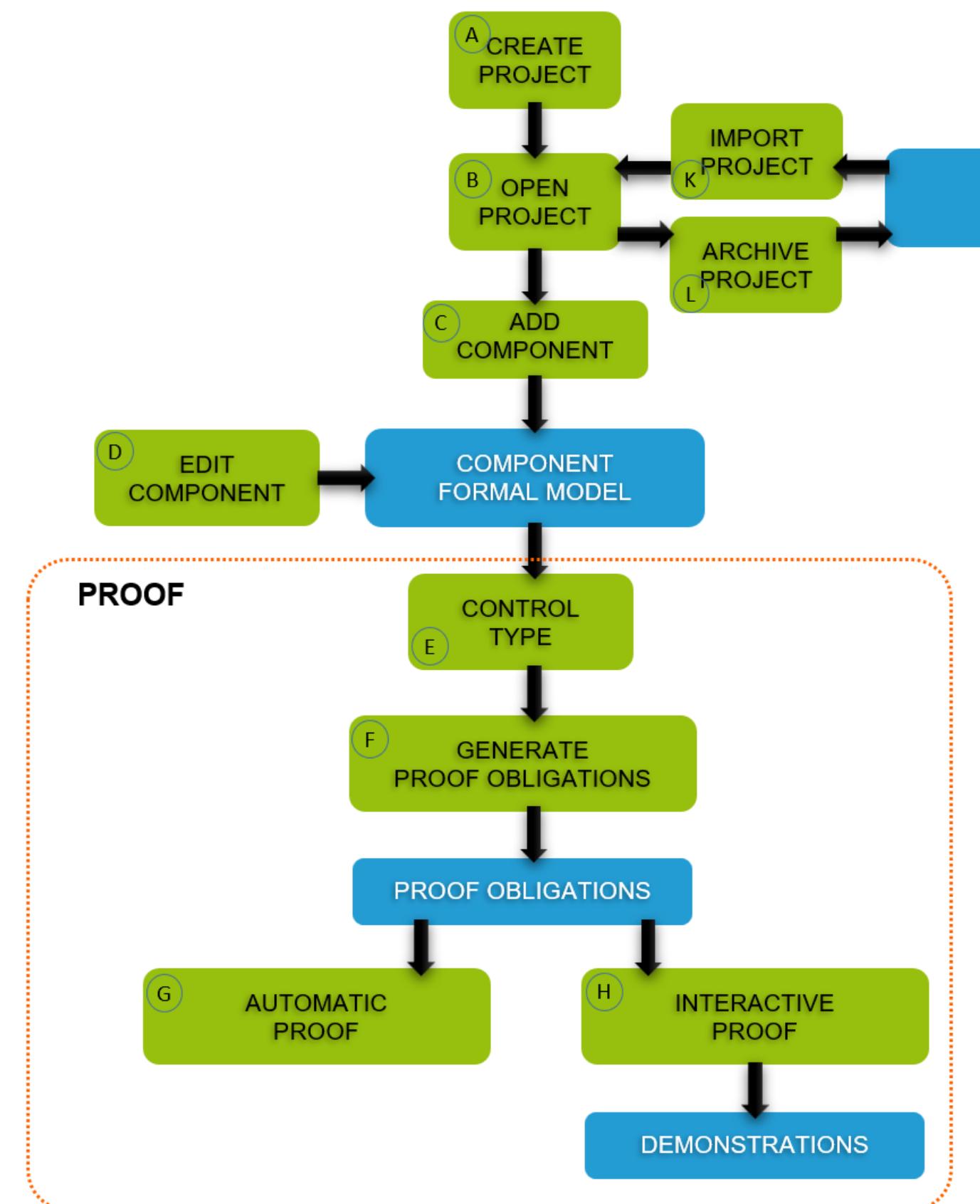
Development process



ROBOSTAR

University of York, UK

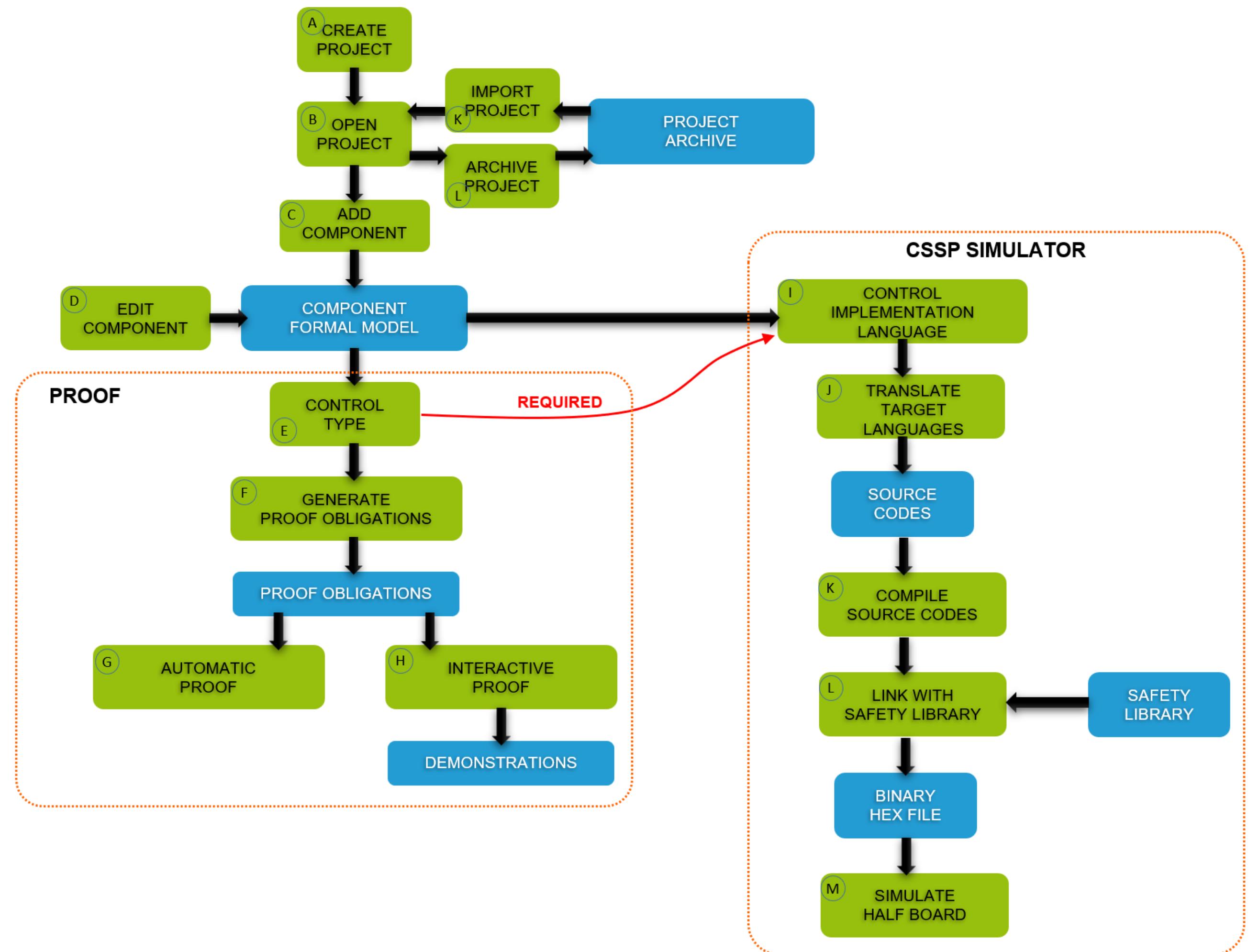
Development process



ROBOSTAR

University of York, UK

Development process



ROBOSTAR

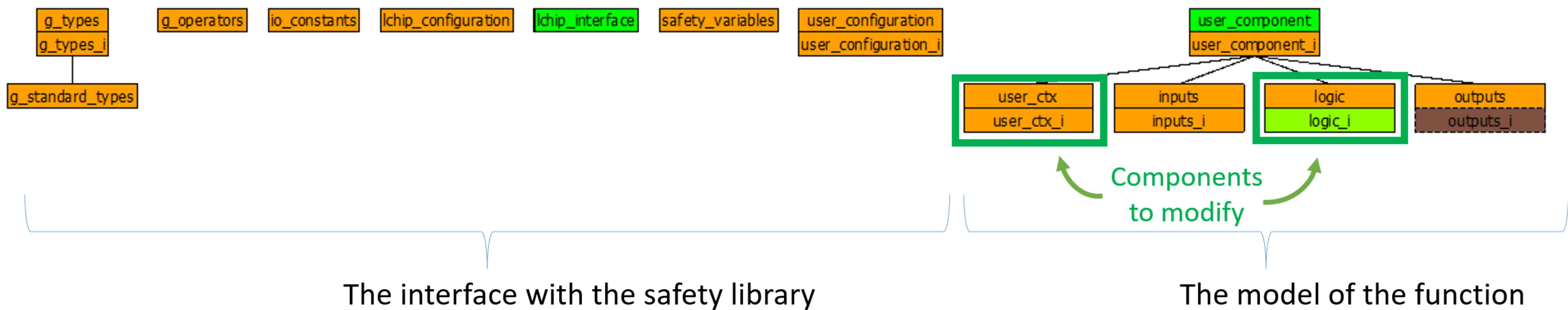
University of York, UK

CLEARSY Safety Platform Project

A CSSP project is a B project

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

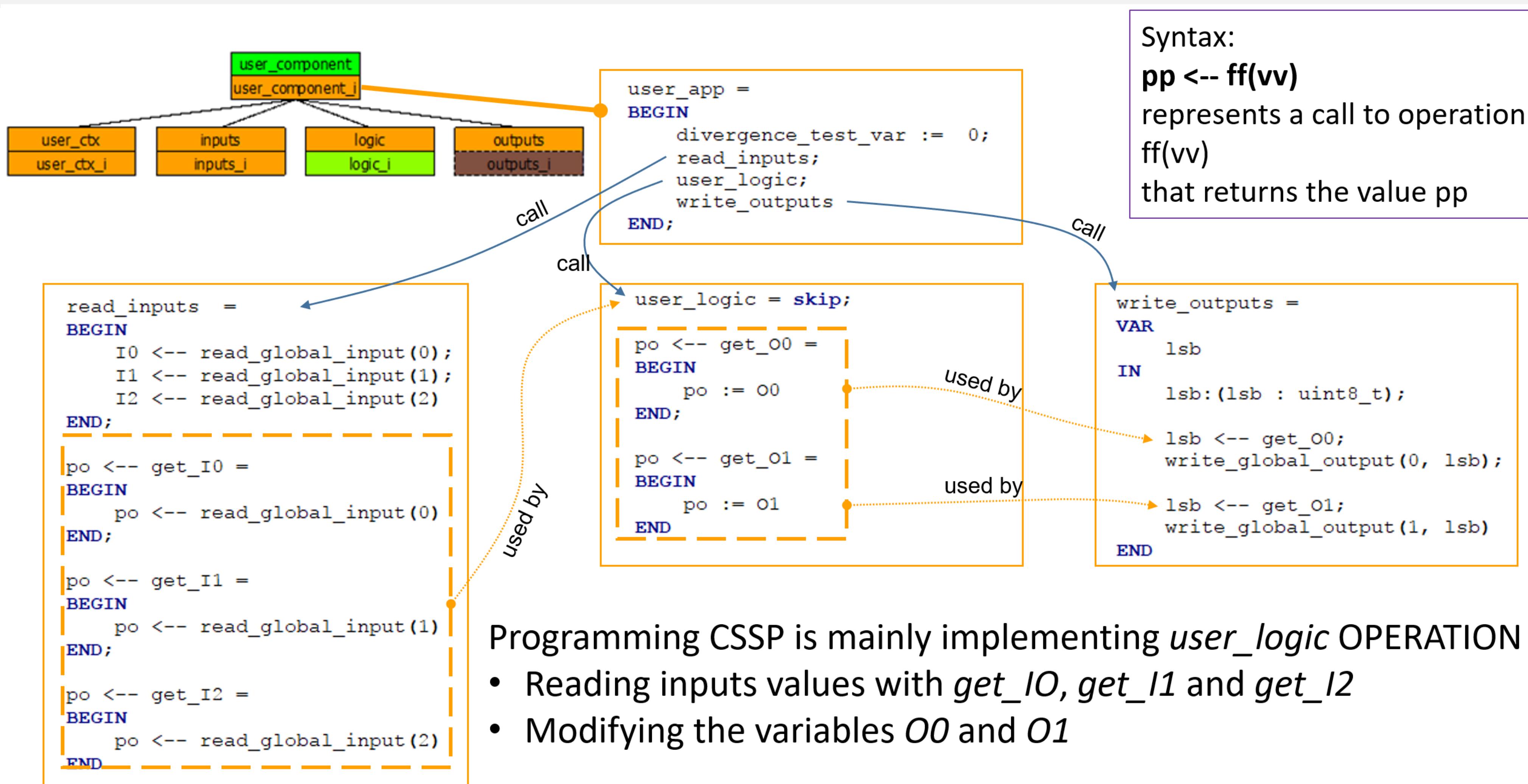
It contains



ROBOSTAR

University of York, UK

Generated Models



Bits of B



ROBOSTAR

University of York, UK

Bits of B : B Variables Declaration

specification

```
ABSTRACT_VARIABLES  
 00,  
 01
```

: means « belongs

```
INVARIANT  
 00 : uint8_t &  
 01 : uint8_t
```

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

```
INITIALISATION  
 00 :: uint8_t  
 01 :: uint8_t
```

:: means « any value within »

implementation

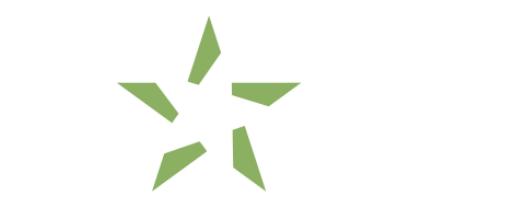
// pragma SAFETY_VARS ————— Contains variables that will be verified

```
CONCRETE_VARIABLES  
 00,  
 01,  
 TIME_A,  
 STATUS
```

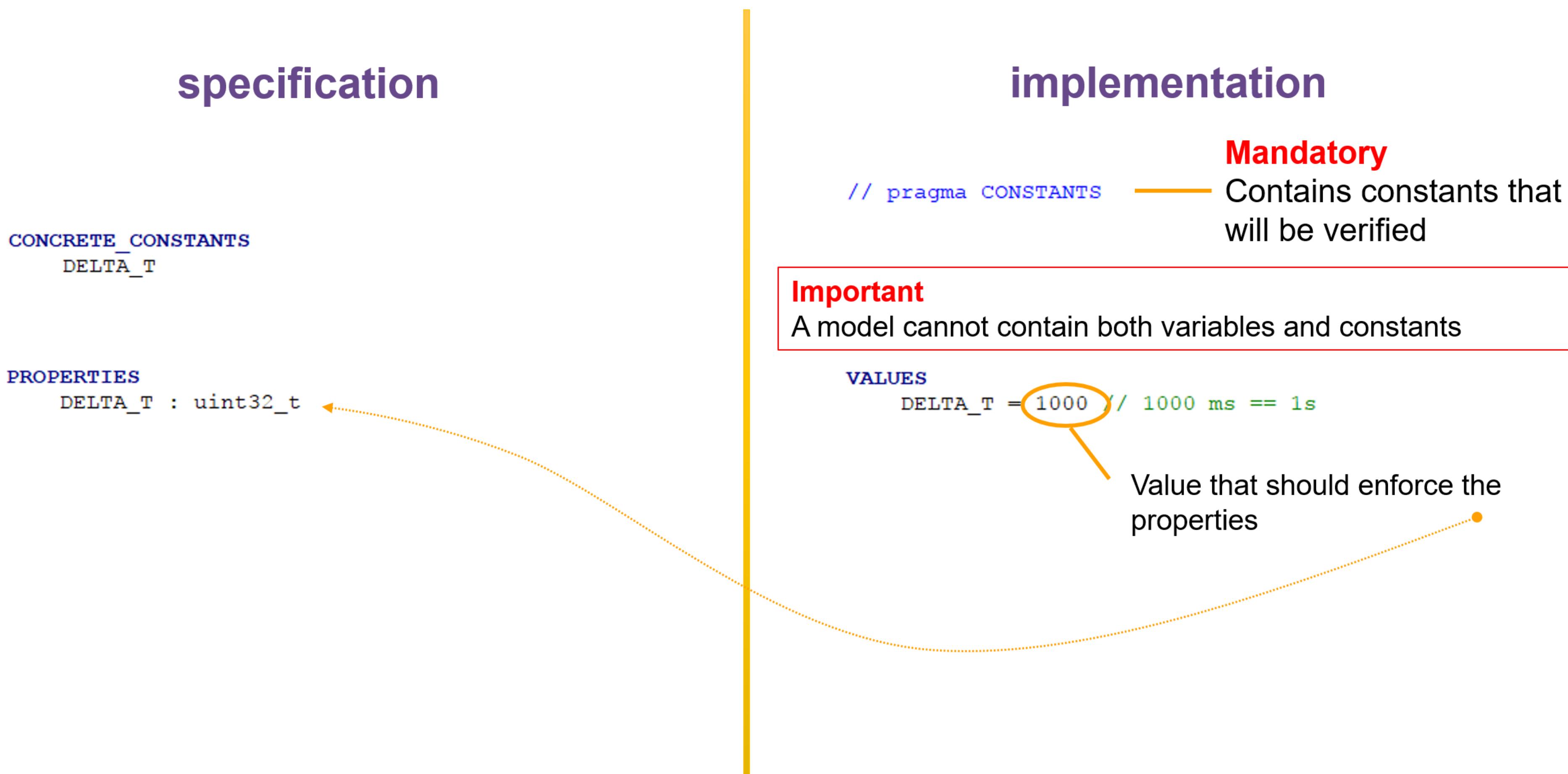
} Variables local to implementation

```
INVARIANT  
 00 : uint8_t &  
 01 : uint8_t &  
 TIME_A : uint32_t &  
 STATUS : uint8_t
```

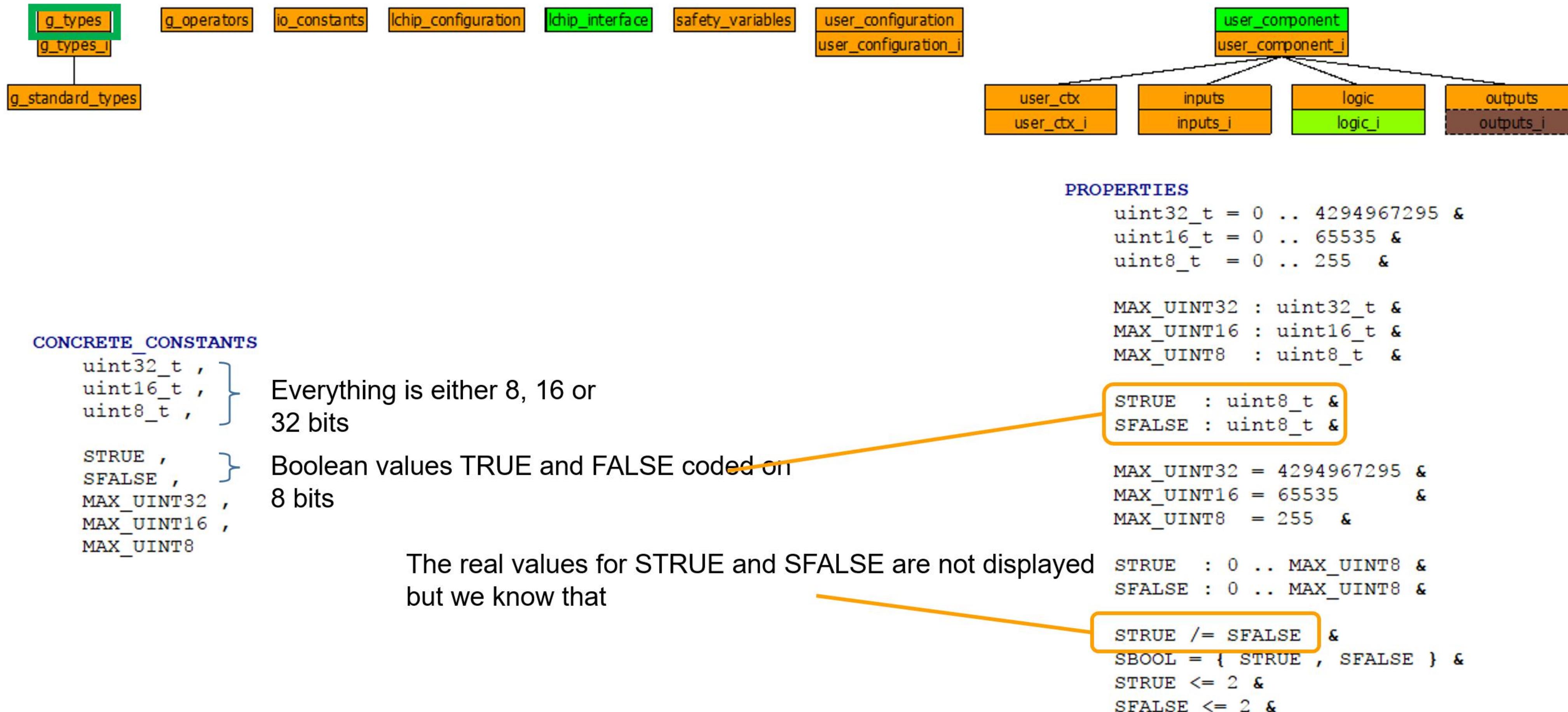
```
INITIALISATION  
 00 := IO_OFF;  
 01 := IO_OFF;  
 TIME_A := 0;  
 STATUS := SFALSE
```



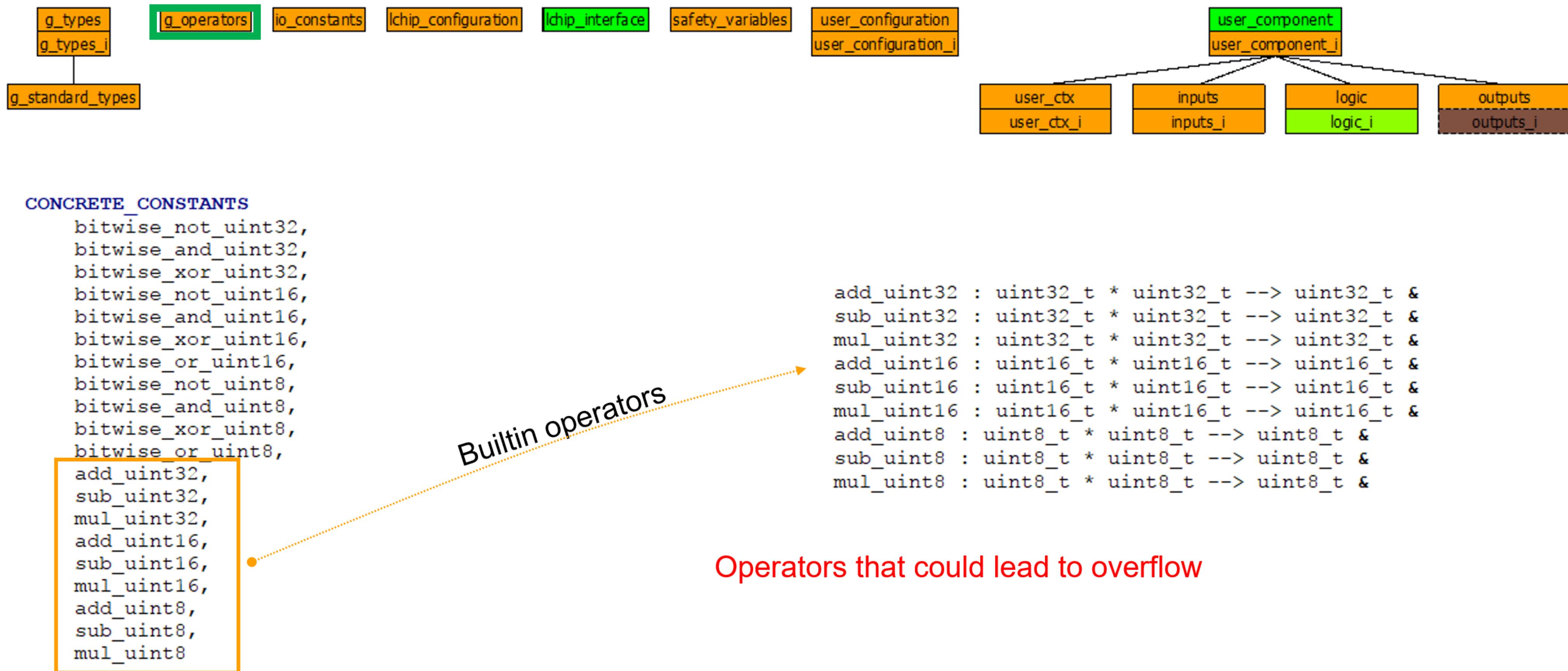
Bits of B : B Constants Declaration



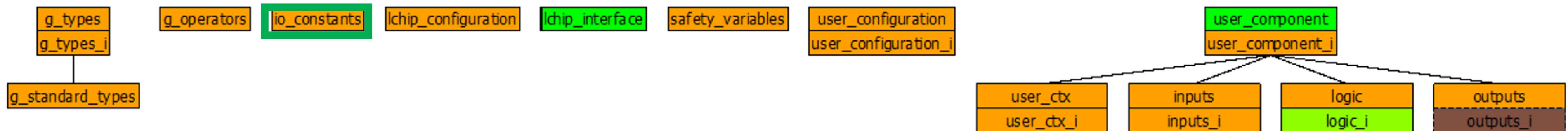
Bits of B : CLEARSY Safety Platform Supported Types



Bits of B : Unsigned INT Operators



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

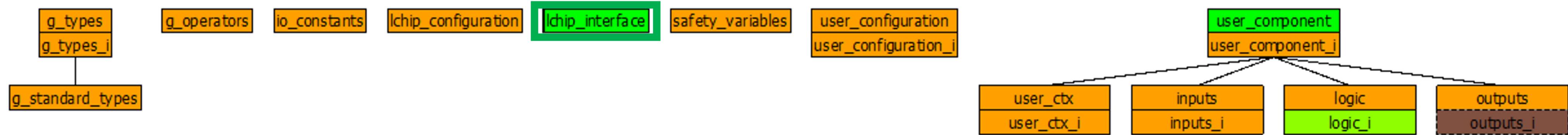
If a digital output is valued with a value different from `IO_ON` or `IO_OFF` then SK_0 stops in error mode



ROBOSTAR

University of York, UK

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 when the operation is completed independently from the algorithm implemented (*post-condition*)

To simplify, always use « becomes such that substitutions »

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

Typing (mandatory)
Constraints (optional)

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;
```

Important

Only single condition (no conjunction nor disjunction)
= < <= 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;
```

Local variables declaration
Operation call

Important

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

Constraints on the language to simplify the compiler

Bits of B: user_logic

specification

```
user_logic = skip;
```

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

```
MACHINE logic
  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
  ABSTRACT_VARIABLES
    o1,
    o2
  INVARIANT
    o1 : uint8_t &
    o2 : uint8_t
  INITIALISATION
    o1 :: uint8_t ||
    o2 :: uint8_t
  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

University of York, UK

Bits of B: user_logic

specification

```
user_logic =
BEGIN
  o0 :: uint8_t || o1 :: uint8_t
```

O0 and O1 belong to their type

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

:() 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
```

Set O0 and reset O1

implementation

```
user_logic =
BEGIN
  o0 := IO_ON;
  o1 := IO_OFF
```

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



Using the Modelling Interface



ROBOSTAR

University of York, UK

Using the modelling interface



- ▶ I_1, I_2, O_1, O_2 belongs to $\{IO_OFF, IO_ON\}$
- ▶ O_1 is IO_ON iff both I_1 and I_2 are IO_ON
- ▶ O_2 is the complement of O_1



ROBOSTAR

University of York, UK

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



- ▶ I_1, I_2, O_1, O_2 belongs to $\{IO_OFF, IO_ON\}$
- ▶ O_1 is IO_ON iff both I_1 and I_2 are IO_ON
- ▶ O_2 is the complement of O_1

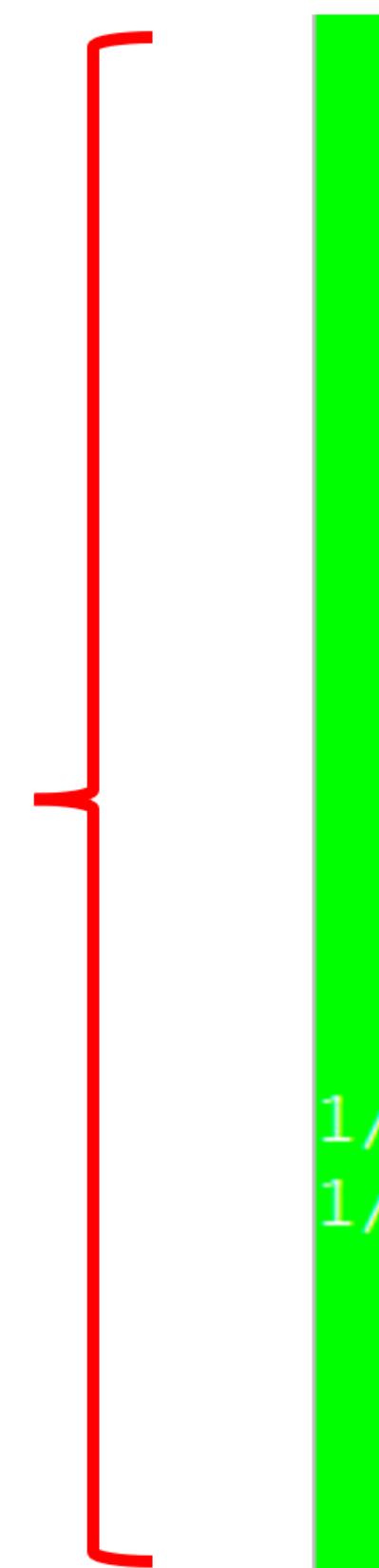
```
user_logic =  
BEGIN  
    o1, o2: (  
        o1 : uint8_t &  
        o2 : uint8_t &  
        (o1=IO_ON <=> (I1=IO_ON & I2=IO_ON)) &  
        not(o1 = o2)  
    )  
END
```



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; ] Get I1 and I2 values

    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
```

Using the modelling interface

The screenshot shows the Robostar modelling interface with the following components:

- Top Bar:** Fichier, Edition, Affichage, Rechercher, Aide.
- Toolbar:** Standard icons for file operations.
- Left Panel:** "OPs à la ligne" (Lines of OPs) window showing the source code.
- Central Editor:** "logic_i.imp" window containing the following UML-like code:

```
1 - MACHINE
2 -   logic
3 - SEES
4 -   g_types,
5 -   g_operators,
6 -   io_constants,
7 -   lchip_interface,
8 -   user_ctx,
9 -   inputs
10 - ABSTRACT_VARIABLES
11 -   O1,
12 -   O2
13 - INVARIANT
14 -   O1 : uint8_t &
15 -   O2 : uint8_t
16 - INITIALISATION
17 -   O1 ::= uint8_t ||| O2 ::= uint8_t
18 -
19 - OPERATIONS
20 -   user_logic = skip;
21 -
22 -   po <-- get_O1 =
23 -     PRE
24 -       po : uint8_t
25 -     THEN
26 -       po := O1
27 -     END;
28 -
29 -   po <-- get_O2 =
30 -     PRE
31 -       po : uint8_t
32 -     THEN
33 -       po := O2
34 -     END
35 -
36 - END
```

- Right Panel:** "Canevas" (Canvas) window showing a tree view of the model elements:

 - MACHINE (logic)
 - SEES (g_types, g_operators, io_constants, lchip_interface, user_ctx, inputs)
 - ABSTRACT_VARIABLES (O1, O2)
 - INVARIANT
 - INITIALISATION
 - OPERATIONS (user_logic, get_O1, get_O2)

- Bottom:** Error messages and navigation controls.

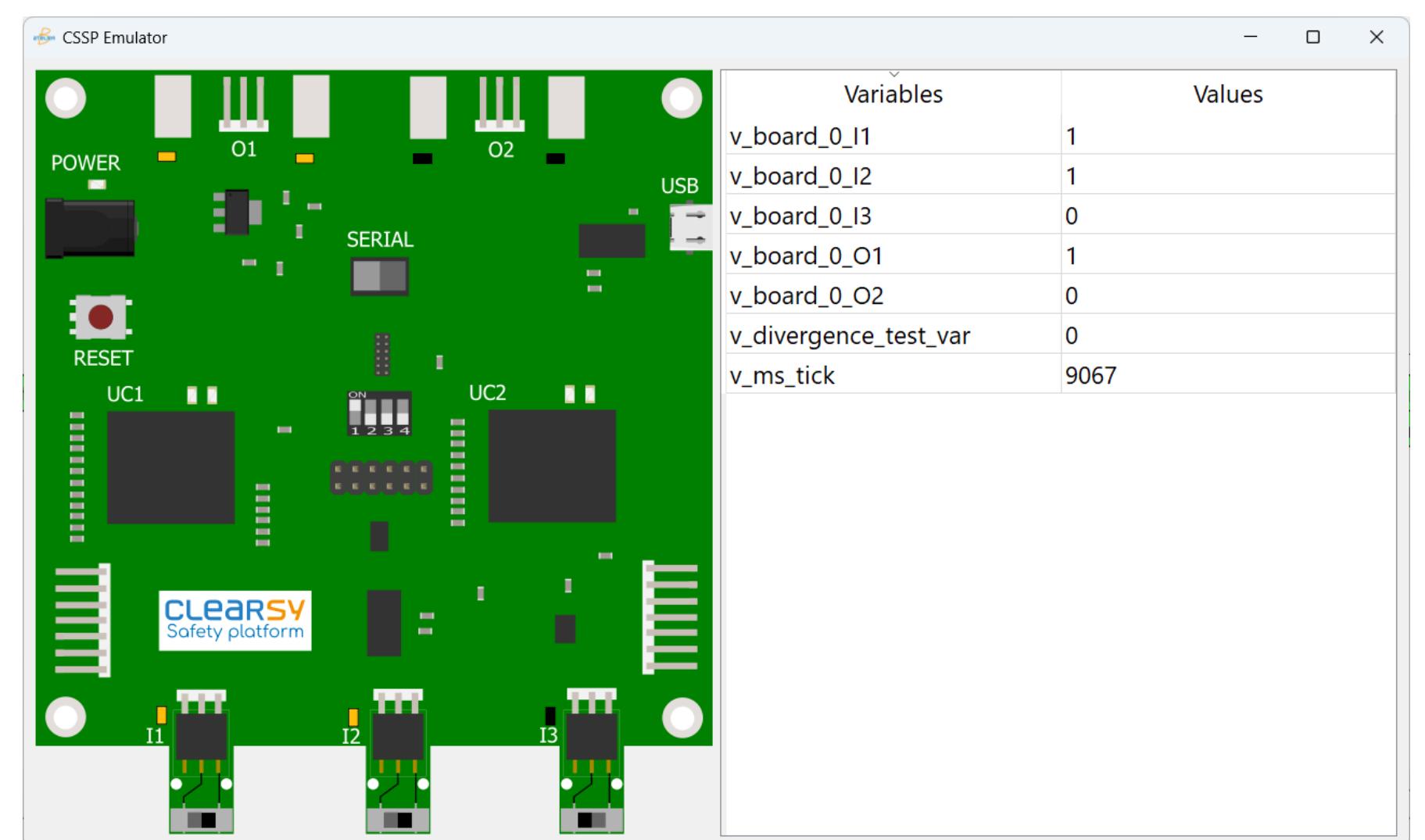
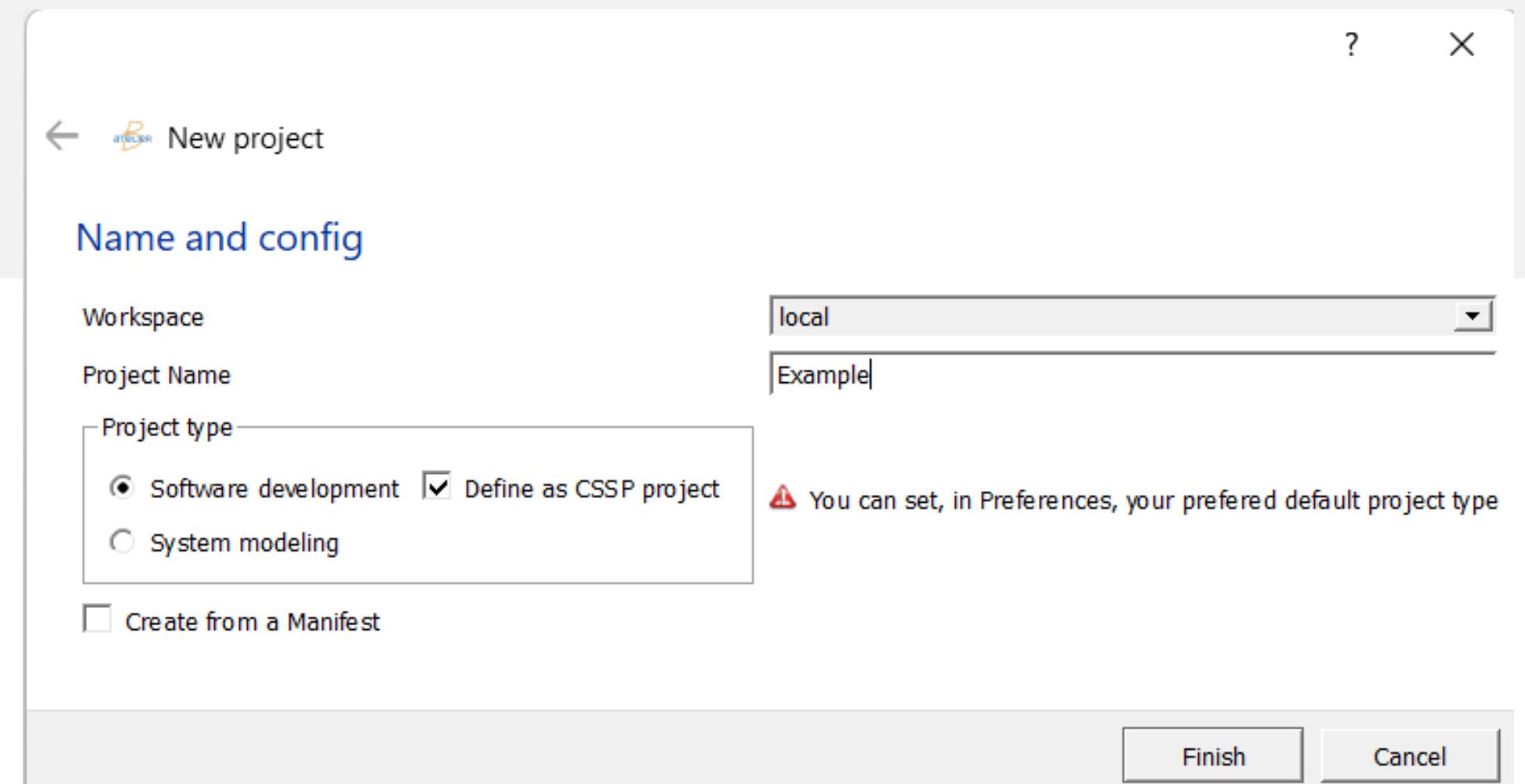


ROBOSTAR

University of York, UK

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

University of York, UK

Next

Programming The CLEARY Safety Platform

Paulo Bezerra, Thierry Lecomte



ROBOSTAR

University of York, UK