Applying a Formal Method In Industry

A 25-Year Trajectory

Thierry Lecomte
May 09, 2018
Rio de Janeiro

CLEARSY
SYSTEMS ENGINEERING

All about Safety Critical Systems

Systems where life is at risk

Would you dare to execute your program if someone would die in case of a crash / core dump / fatal error / etc. ?

Systems = Software + hardware + environment

Specification error

Development error
Programming error
Bad compilation

Wrong execution
Processor reliability: 10^{-6} to 10^{-7} / h

2 processors in parallel (or more)
Protecting mechanisms in case of perturbation

Safety Critical Systems & standards

Standards

- Domain dependent
- Recommendations:
  - No definitive recipe to produce safe systems
  - Software and hardware, development process

IEC 61508: Software design and dev. (table A.2)

Technique/Measure	Ref	SIL1	SIL2	SIL3	SIL4
1 Fault detection and diagnosis	C.3.1	---	R	HR	HR
2 Error detecting and correcting codes	C.3.2	R	R	R	HR
3a Failure assertion programming	C.3.3	R	R	R	HR
3b Safety bag techniques	C.3.4	---	R	R	R
3c Diverse programming	C.3.5	R	R	R	HR
3d Recovery block	C.3.6	R	R	R	R
3e Backward recovery	C.3.7	R	R	R	R
3f Forward recovery	C.3.8	R	R	R	R
3g Re-try fault recovery mechanisms	C.3.9	R	R	R	HR
3h Memorising executed cases	C.3.10	---	R	R	HR
4 Graceful degradation	C.3.11	R	R	HR	HR
5 Artificial intelligence - fault correction	C.3.12	---	NR	NR	NR
6 Dynamic reconfiguration	C.3.13	---	NR	NR	NR
7a Structured methods including for example, JSD, MASCOT, SADT and Yourdon	C.2.1	HR	HR	HR	HR
7b Semi-formal methods	Table B.7	R	R	HR	HR
7c Formal methods including for example, CCS, CSP, HOL, LOTOS, OBJ, temporal logic, VDM and Z	C.2.4	---	R	R	HR
8 Computer-aided specification tools	B.2.4	R	R	HR	HR

a) Appropriate techniques/measures shall be selected according to the safety integrity level. Alternate or equivalent techniques/measures are indicated by a letter following the number. Only one of the alternate or equivalent techniques/measures has to be satisfied.

b) The measures in this table concerning fault tolerance (control of failures) should be considered with the requirements for architecture and control of failures for the hardware of the programmable electronics in part 2 of this standard.

All about Safety Critical Systems

Standards

- Safety case: a demonstration that the feared event will not happen more frequently than expected

Safety Integrity Level 3: 1 failure every 100 years

Safety Integrity Level 4: 1 failure every 10 000 years

Agenda

Formalizing software

Presenting the B method and Atelier B

Verifying formally parameters

Presenting formal data validation

Executing software on a safety hardware platform

Presenting CLEARSY Safety Platform

CLEARSY Safety Platform
Starter kit SK_0

The B Method in a Nutshell

≡ Is a formal method to develop software mathematically proved to comply with its specification

≡ [Formal] : it relies on a mathematical model of the software, containing both

- what the software is expected to do and
- its algorithm

≡ [Method] : Model is decomposed into smaller models in order to manage complexity ("*divide and conquer*")

≡ The model is mathematically [proved]: the algorithm doesn't contradict its specification

≡ The software code is generated from the model

- Code is readable, very close to the model and is easily checked

≡ The final software application is made of parts developed with B and parts not developed formally

B: A path from requirements to code

Natural language requirement

« Only inactive sequences can be added to the active sequences execution queue. »

B Specification

Behaviour + properties

```
activation_sequence = /* Activation d'une séquence non active */
PRE ¬(sequences = sequences_actives) THEN
    ANY sequ WHERE
        sequ ∈ sequences - sequences_actives
    THEN
        sequences_actives := sequences_actives ∪ {sequ}
    END
END;
```

B Implementation

Behaviour + properties

```
activation_sequence = /* Activation d'une séquence non active */
VAR sequ IN
    sequ <-- indexSequenceInactive;
    activeSequence(sequ)
END;
```

C generated code

```
void M0__activation_sequence(void)
{
    CTX__SEQUENCES sequ;

    sequence_manager__indexSequenceInactive(&sequ);
    sequence_manager__activeSequence(sequ);
}
```

Binary code

```
0x01F970 | FFFF 8B4C 2440 89C5 8D7D 0C8B 4110 89CE
0x01F980 | 83C6 0C8D 1485 0000 0000 8D42 0883 F807
0x01F990 | 7617 F7C7 0400 0000 740F 8B41 0C8D 7D10
0x01F9A0 | 83C6 0489 450C 8D42 04FC 89C1 C1E9 02F3
```

A path from requirements to code

« Only inactive sequences can be added to the active sequences execution queue. »

Natural language requirement

```
activation_sequence = /* Activation d'une séquence non active */
PRE ¬(sequences = sequences_actives) THEN
    ANY sequ WHERE
        sequ ∈ sequences - sequences_actives
    THEN
        sequences_actives := sequences_actives ∪ {sequ}
    END
END;
```

B Specification

Proof (coherence)

Proof (refinement)

```
activation_sequence = /* Activation d'une séquence non active */
VAR sequ IN
    sequ <-- indexSequenceInactive;
    activeSequence(sequ)
END;
```

B Implementation

Proof (coherence)

```
void M0__activation_sequence(void)
{
    CTX__SEQUENCES sequ;

    sequence_manager__indexSequenceInactive(&sequ);
    sequence_manager__activeSequence(sequ);
}
```

C generated code

0x01F970	FFFF	8B4C	2440	89C5	8D7D	0C8B	4110	89CE
0x01F980	83C6	0C8D	1485	0000	0000	8D42	0883	F807
0x01F990	7617	F7C7	0400	0000	740F	8B41	0C8D	7D10
0x01F9A0	83C6	0489	450C	8D42	04FC	89C1	C1E9	02F3

Binary code

Cyclic software single-thread

The B method in one example

MACHINE = specification

M0.mch*

```
MACHINE
    M0
VARIABLES
    xx
INVARIANT
    xx: INTEGER
INITIALISATION
    xx := 0
OPERATIONS
    inc =

END
```

Variables

Invariant

- Types for variables (set, function, scalar, boolean, integer, table, etc.)
- Constraints for variables
- Invariant is always true
  - Established by initialization
  - If true before operation execution, still true after

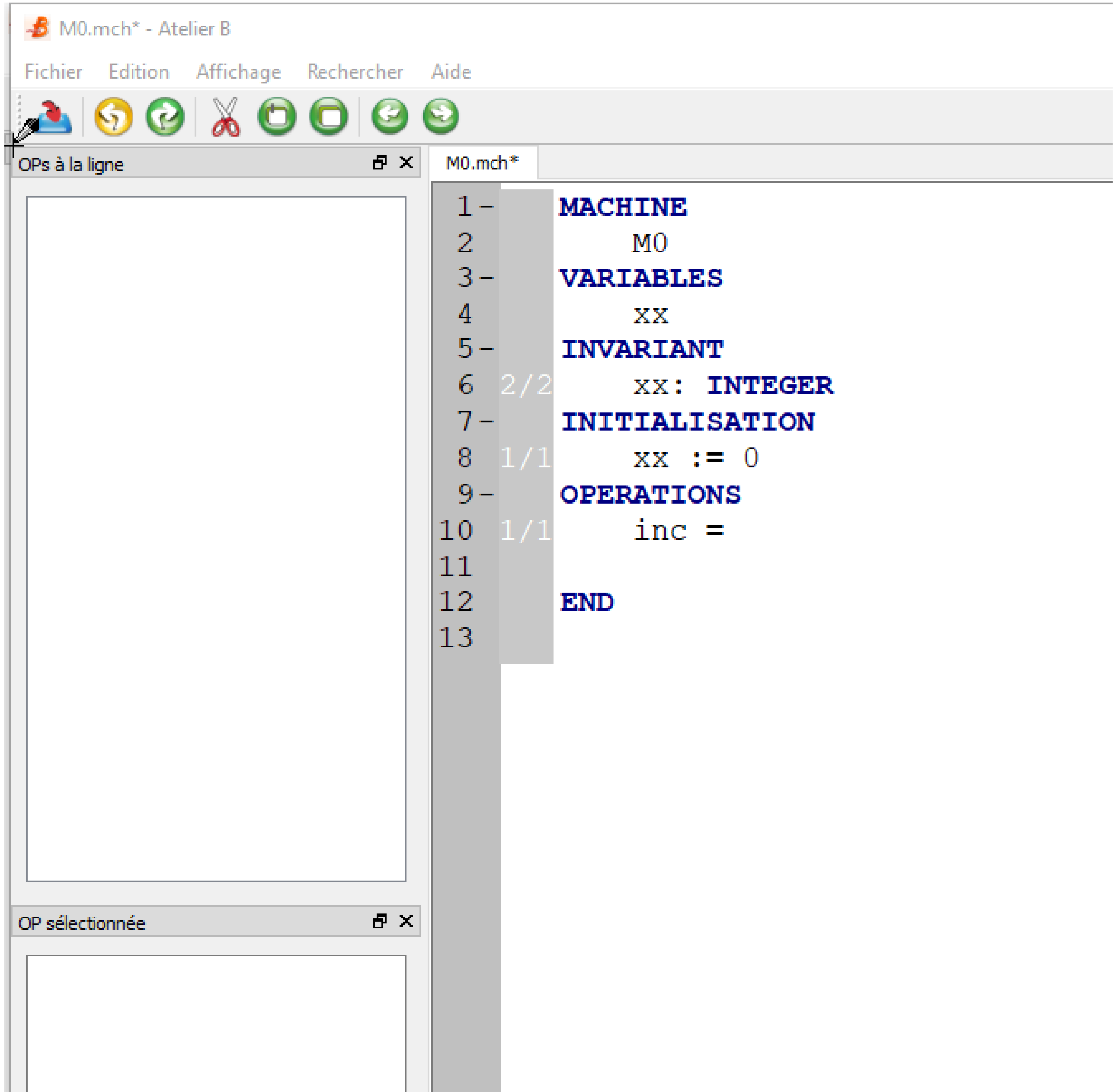
Initialization

- First value for variables
- May be non-deterministic
- All variables initialized at the same time

Operation

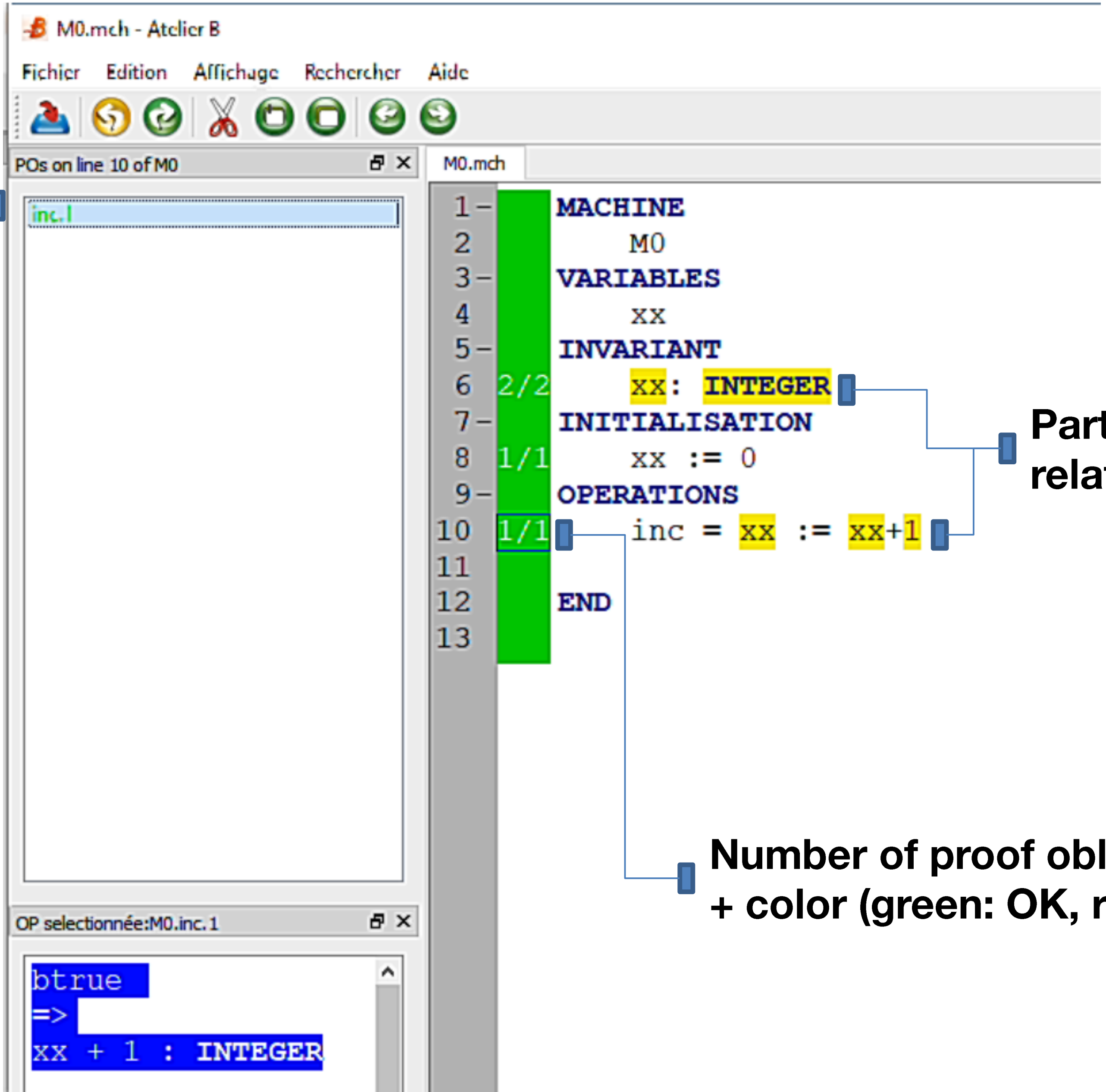
- Modify variables
- Specified as a transfer function (no algorithm)

The B method in one example



The B method in one example

Proof obligations list



Parts of the model related to the proof

Number of proof obligations related to the line + color (green: OK, red: NOK)

Proof obligation

The B method in one example

MACHINE = specification

M0.mch*

```
MACHINE
    M0
VARIABLES
    xx
INVARIANT
    xx: INTEGER
INITIALISATION
    xx := 0
OPERATIONS
    inc =

END
```

IMPLEMENTATION = algorithm

M0.mch | M0_i.imp*

```
IMPLEMENTATION M0_i
REFINES M0

CONCRETE_VARIABLES
    xx
INVARIANT
    xx: INT
INITIALISATION
    xx := 0

OPERATIONS
    inc =

END
```

- REFINES M0 → **Implement its specification**
- CONCRETE_VARIABLES → **Implemented variable**
- xx: INT → **Implementable type**

The B method in one example

POs on line 12 of M0_i

inc.1

OP selectionnée:M0_i.inc.1

```
btrue &
btrue
=>
xx$1 + 1 : INT
```

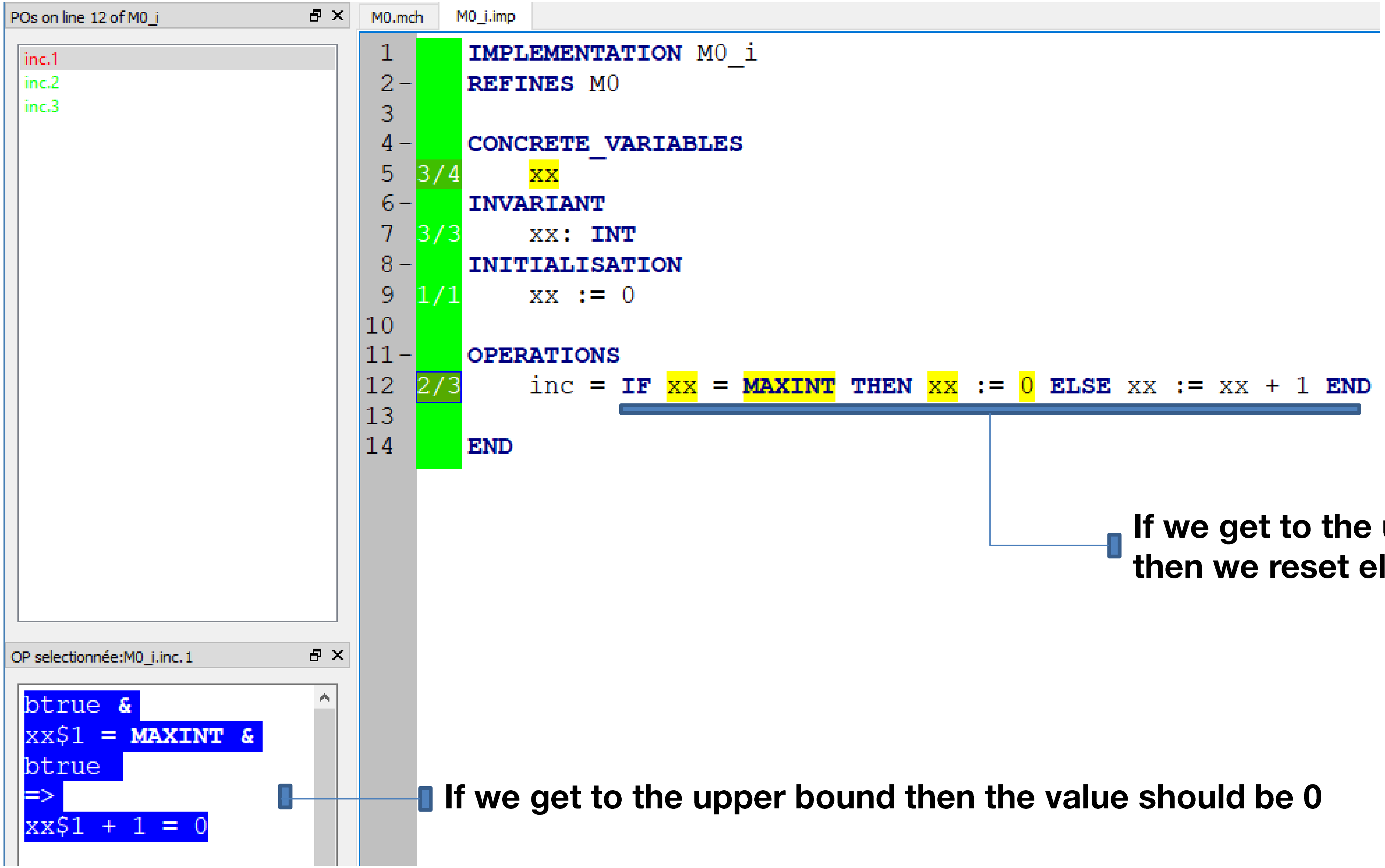
M0.mch | M0_i.imp

```
1          IMPLEMENTATION M0_i
2          REFINES M0
3
4          CONCRETE_VARIABLES
5   1/2        xx
6          INVARIANT
7   1/2        xx: INT
8          INITIALISATION
9   1/1        xx := 0
10
11         OPERATIONS
12  0/1        inc = xx := xx + 1
13
14         END
```

Endless increment

If we increment enough , we get out of type INT

The B method in one example



If we get to the upper bound (MAXINT) then we reset else we increment

If we get to the upper bound then the value should be 0

The B method in one example

M0.mch | M0_i.imp

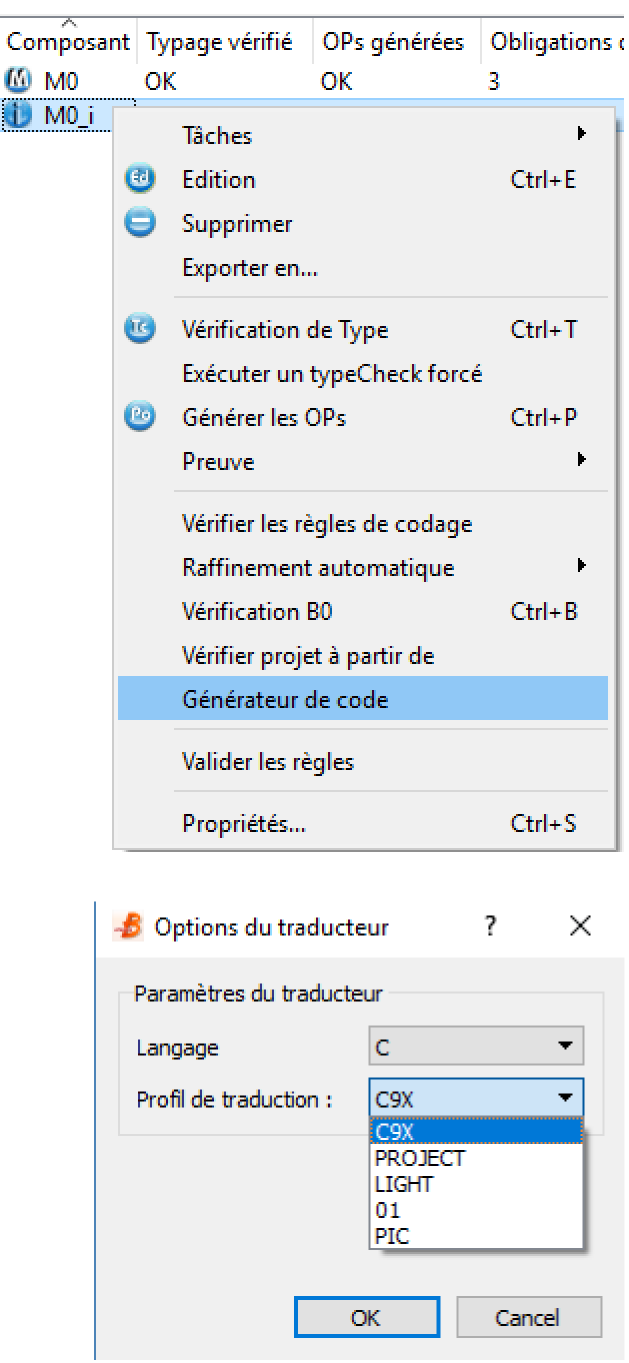
```
MACHINE
     M0
VARIABLES
     xx
INVARIANT
     xx: INTEGER
INITIALISATION
     xx := 0
OPERATIONS
     inc = CHOICE xx := 0 OR xx := xx + 1 END

END
```

Either we increment or we reset

The 2 models
are compatible

M0.mch | M0_i.imp

```
IMPLEMENTATION M0_i
REFINES M0

CONCRETE_VARIABLES
     xx
INVARIANT
     xx: INT
INITIALISATION
     xx := 0

OPERATIONS
     inc = IF xx = MAXINT THEN xx:=0 ELSE xx := xx +1 END
END
```




```
#ifndef _M0_h
#define _M0_h

#include <stdint.h>
#include <stdbool.h>
#ifdef __cplusplus
extern "C" {
#endif /* __cplusplus */


/* Clause SETS */

/* Clause CONCRETE_VARIABLES */


/* Clause CONCRETE_CONSTANTS */
/* Basic constants */
/* Array and record constants */
extern void M0__INITIALISATION(void);

/* Clause OPERATIONS */

extern void M0__inc(void);

#ifdef __cplusplus
}
#endif /* __cplusplus */


#endif /* _M0_h */
```

```
/* WARNING if type checker is not p

#include "M0.h"

/* Clause CONCRETE_CONSTANTS */
/* Basic constants */

/* Array and record constants */
/* Clause CONCRETE_VARIABLES */

static int32_t M0__xx;
/* Clause INITIALISATION */
void M0__INITIALISATION(void)
{

    M0__xx = 0;
}

/* Clause OPERATIONS */

void M0__inc(void)
{
    if(M0__xx == 2147483647)
    {
        M0__xx = 0;
    }
    else
    {
        M0__xx = M0__xx+1;
    }
}
```

Guaranty: no programming error

```
    IMPLEMENTATION M0_i
    REFINES M0

    CONCRETE_VARIABLES
3/6     xx,yy,zz,
0/1     tab

    INITIALISATION
1/1         xx := 0;
            yy := 0;
            zz := 0;
        tab := (0..10)*{0}

    OPERATIONS
        op =
        BEGIN
0/2         xx := yy / zz;
1/3         yy := MAXINT;
1/2         xx := yy + 1;
0/1         tab(yy) := 0
            END
     |
     END
```

Programming error

- no overflow,
- no division by zero,
- no access to a table outside its domain

3 integers

1 table of 11 elements: tab(0), …, tab(10)

Division by 0

Overflow

Table used outside its domain

Guaranty: no programming error

```
3/6  0/1  1/1  0/2  1/3  1/2  0/1
IMPLEMENTATION M0_i
REFINES M0

CONCRETE_VARIABLES
3/6     xx,yy,zz,
0/1     tab

INITIALISATION
1/1             xx := 0;
                yy := 0;
                zz := 0;
        tab := (0..10)*{0}

OPERATIONS
        op =
        BEGIN
0/2             xx := yy / zz;
1/3             yy := MAXINT;
1/2             xx := yy + 1;
0/1             tab(yy) := 0
                END

 END
```

When we save the model, a number of proof obligations are not automatically proved

Guaranty: no programming error

op.1
WellDefinedness_op.1

Overflow is detected because not provable

Selected PO : M0_i.op.1

```
btrue &
btrue
=>
MAXINT + 1 : INT
```

```
    IMPLEMENTATION M0_i
    REFINES M0

    CONCRETE_VARIABLES
3/6     xx,yy,zz,
0/1     tab

    INITIALISATION
1/1         xx := 0;
            yy := 0;
            zz := 0;
        tab := (0..10)*{0}

    OPERATIONS
        op =
        BEGIN
0/2         xx := yy / zz;
1/3         yy := MAXINT;
1/2         xx := yy + 1;
0/1         tab(yy) := 0
            END

     END
```

Guaranty: no programming error

op.1
WellDefinedness_op.1

Division by 0 is detected
because not provable

Selected PO : M0_i.WellDefinedness_op.1

```
not(zz$1 = 0)
```

```
      IMPLEMENTATION M0_i
      REFINES M0

      CONCRETE_VARIABLES
3/6       xx,yy,zz,
0/1       tab

      INITIALISATION
1/1           xx := 0;
              yy := 0;
              zz := 0;
          tab := (0..10)*{0}

      OPERATIONS
          op =
          BEGIN
0/2           xx := yy / zz;
1/3           yy := MAXINT;
1/2           xx := yy + 1;
0/1           tab(yy) := 0
              END

       END
```

Guaranty: no programming error

op.4

Access to table outside its domain is detected because not provable

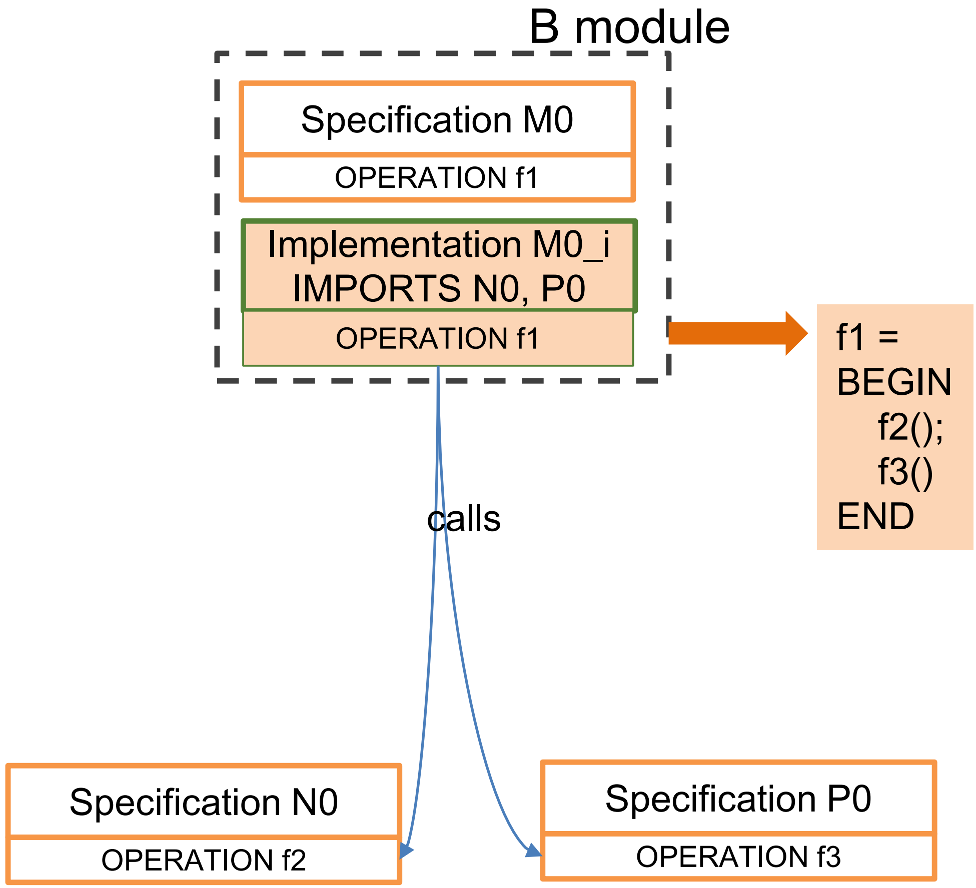
Selected PO : M0_i.op.4

```
btrue &
btrue
=>
tab$1 <+ {MAXINT |-> 0} : 0 .. 10 --> INT
```

```
IMPLEMENTATION M0_i
REFINES M0

CONCRETE_VARIABLES
     xx,yy,zz,
     tab

INITIALISATION
         xx := 0;
         yy := 0;
         zz := 0;
     tab := (0..10)*{0}

OPERATIONS
     op =
     BEGIN
         xx := yy / zz;
         yy := MAXINT;
         xx := yy + 1;
         tab(yy) := 0
         END

 END
```

Models Architecture

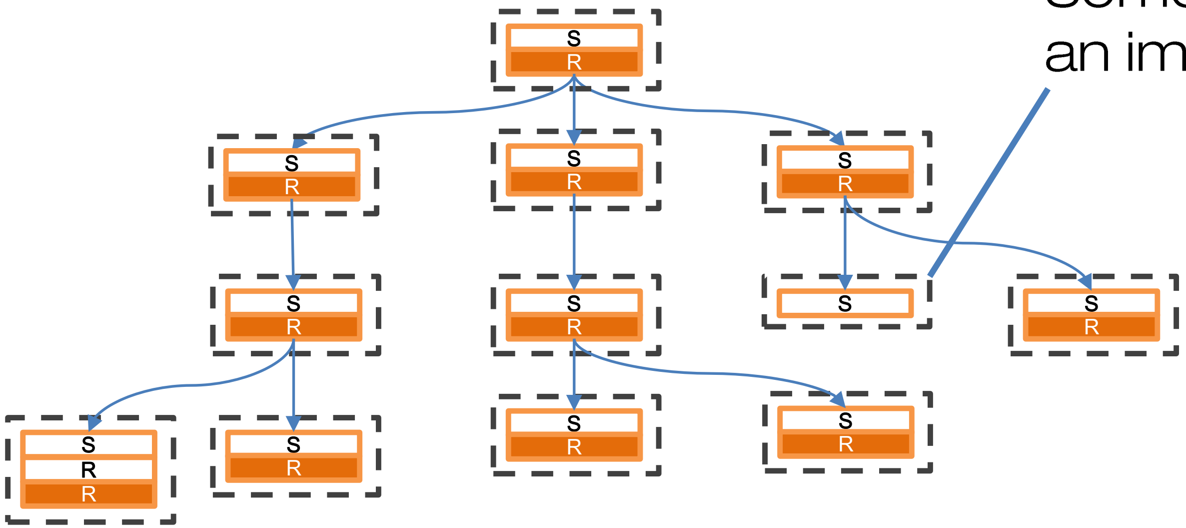
- One operation cannot call other operations from the same implementation
- One operation can call operations defined in other machines
- These machines have to be imported in an implementation
- Variables defined in imported machines have to be different: a variable cannot be modified in 2 components



Models Architecture

IMPORTS have to be applied iteratively to obtain the target decomposition

The decomposition graph should be a tree

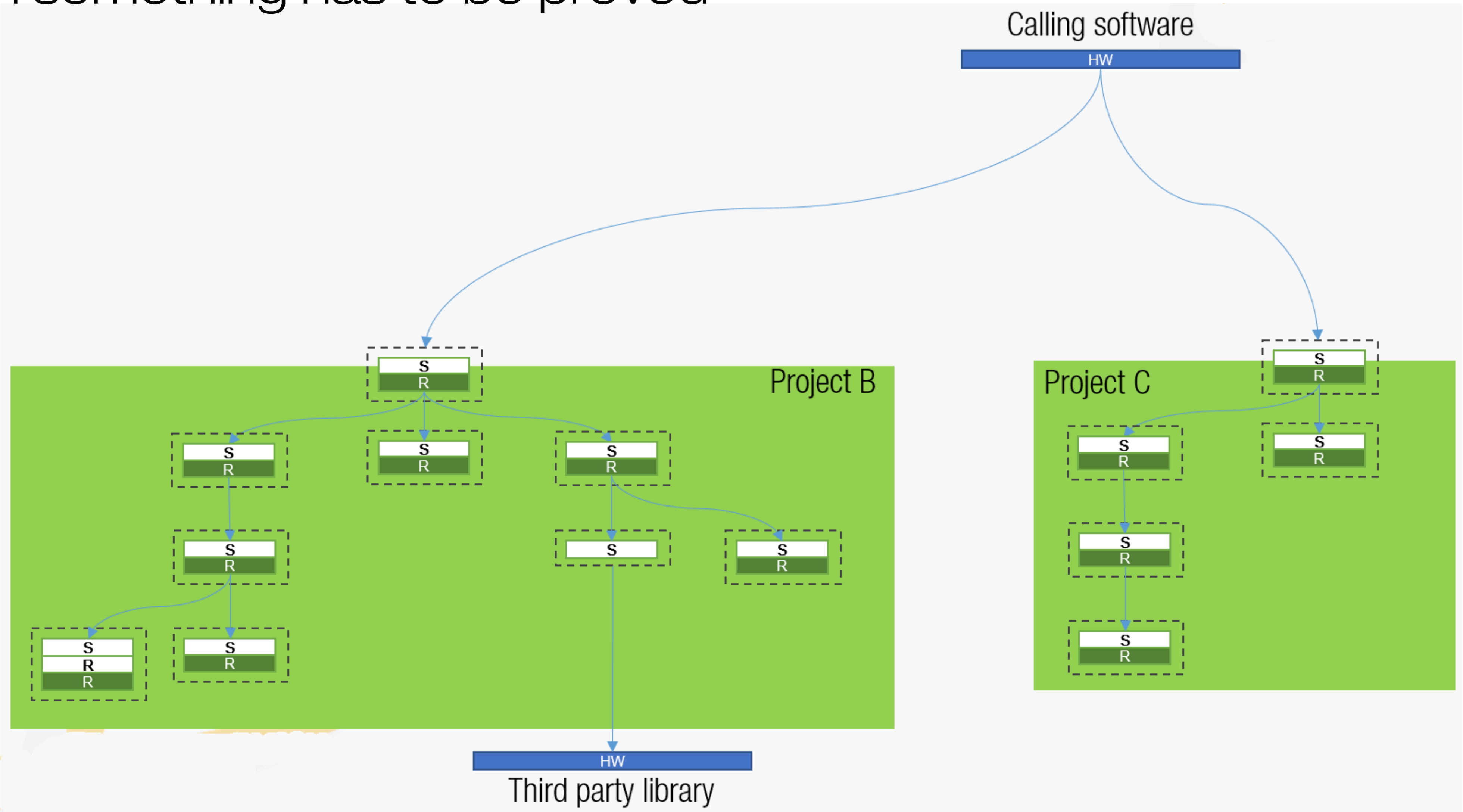


Some components do not have an implementation in B

Integrating B to a Software Project

≡ Compatible with any SW development cycle

To be used when something has to be proved



Why using B /1

≡ Improved level of confidence

Brought by mathematical formalism and proof

≡ Ambiguities are removed

The mathematical model captures the meaning of the software

≡ Early error detection

Errors are discovered

- by proof during the modelling and
- not by test once the software is built

Why using B /2

- Most testing useless

Testing is replaced by proof

Mathematical proof is exhaustive, testing is not

- Avoid redundant software development

For highest safety integrity level, only one B model required, compared with two software developed more traditionally by two independent teams

- Accepted for certification

Several industrial standards recommend or strongly recommend the use of formal methods (EN50128, IEC61508)

Why not using B ?

≡ Numerical algorithms

- FFT, Kalman, etc.

≡ Non functional properties (cannot be proved within B)

- "The software terminates in less than xxx milliseconds"

≡ Optimality

- "The software is the smallest / fastest / most efficient one"

≡ Nothing to prove

- No properties: no other specification than "my software should do A then B then C"

Atelier B

≡ A tool, freely available and fully functional, for
- developing [safety critical] software (B language, B method)
- modelling systems (Event-B language)

≡ Includes editor, proof tools and code generator

≡ Dedicated support for maintenance contract holders

≡ Used for certified applications up to SIL4

≡ Version 4.5 to be published in June 2018

http://www.atelierb.eu/en

Educational resources

Lecture 0: Marketing video

This video explains why you should follow the MOOC on B and what its expected benefits on your career are.

Level: Basic — Video duration: 02:55

Lecture 1: Course Introduction

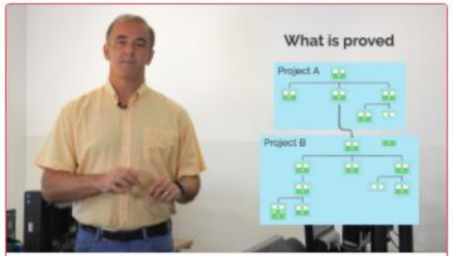
This video presents the structure of the course, provides an overview of the different kinds of formal methods and specification styles, and tells us some myths on formal methods

Level: Basic — Video duration: 08:43

Lecture 2: Overview of the B method

This video briefly introduces the tool Atelier-B, the B and Event-B languages, and some industrial references. The main concepts of B are exposed.

Level: Basic — Video duration: 15:03



Lecture 3: The concepts of B

This video presents the founding notions of B: projects, libraries, modules, components, abstract machine, refinement, implementation , and proof.

Level: Basic — Video duration: 09:29

Lecture 4 : introduction to Abstract Machines

This video introduces the notion of abstract machines, based on an example that is verified, animated and for which C source code is generated.

Level: Basic — Video duration: 16:31

MOOC
massive open online course

http://mooc.imd.ufrn.br

MOOC on B (under construction)

- 20 lectures covering B for software development
- Collaboration with UFRN/IMD (Natal)

Industrial applications

Atelier B

3M€ to strengthen Alstom Atelier B and fully support the B method

Funded by RATP, SNCF, INRETS

Tool qualification by RATP

accepted by standards

2 requirements:

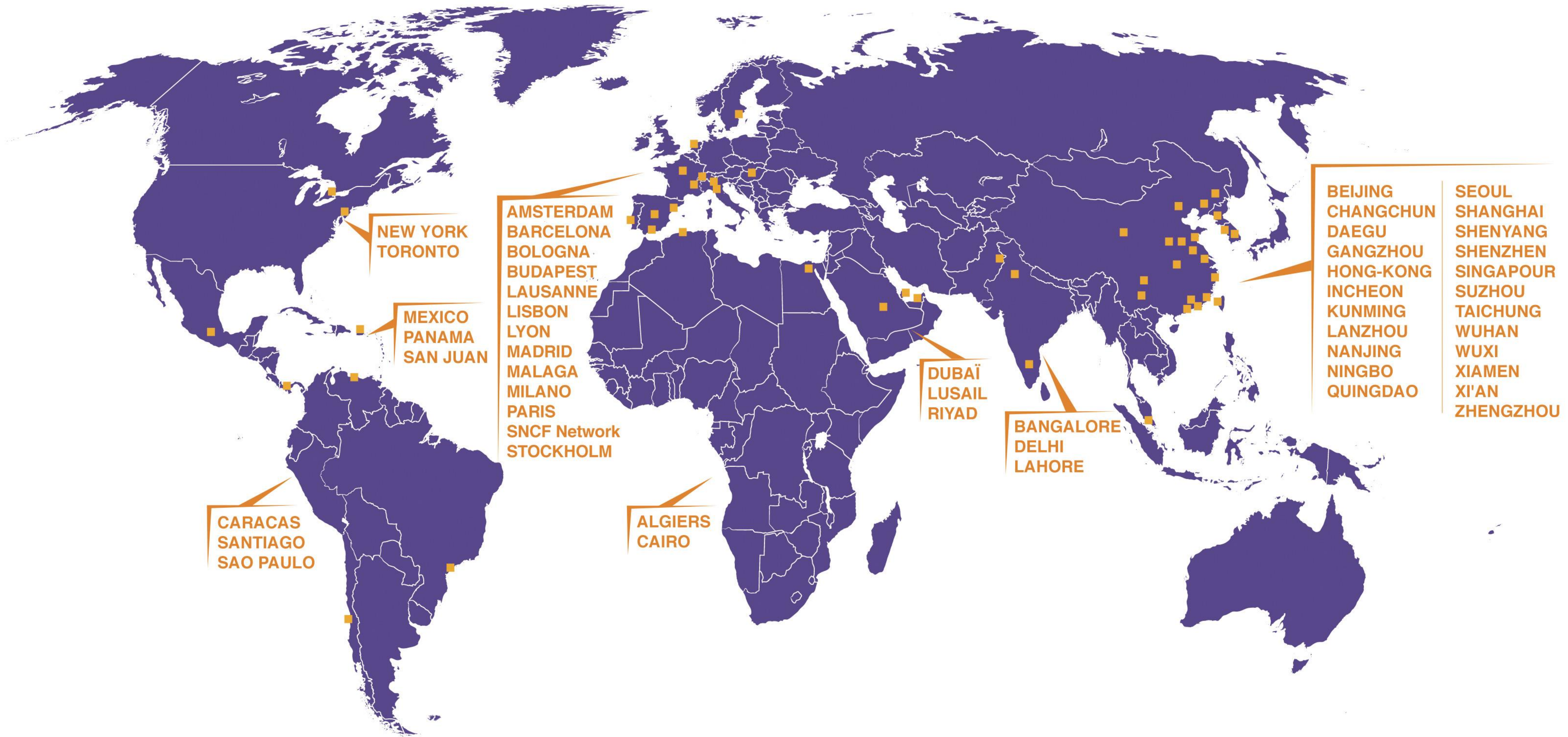
- 10s to prove 1 proof obligation
- Ability to support the development of the first driverless metro

Paris Line 14 driverless metro ready by end 1998

- 110 kloc B model, 83% automatic proof
- 86 kloc Ada code
- Still in version 1.0 after 19 years exploitation

ATELIER B

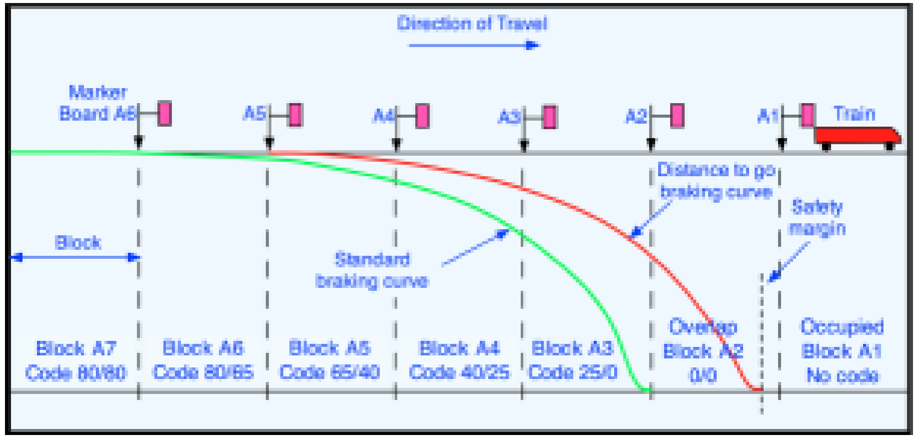
METROS AND TRAINS EQUIPPED WITH **B SIL4 SOFTWARE**



Software Developer

≡ 30%+ of the driverless metro in the world (Alstom, Siemens)

≡ Automatic Train Protection System



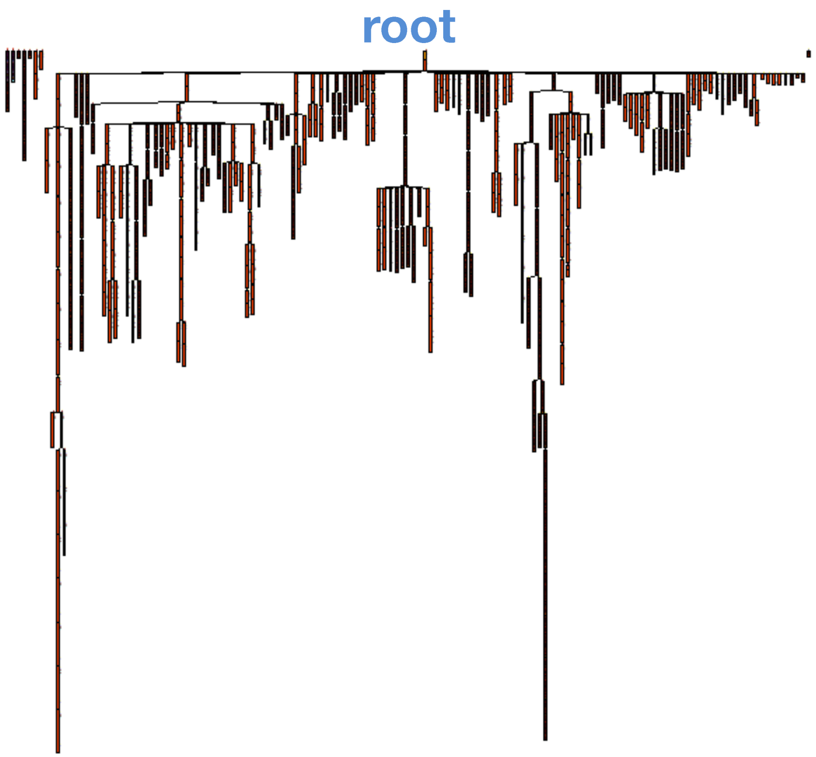
What is proved:
if the train is not allowed to go forward,
then the emergency brake should be triggered

≡ Automation to reduce intervals between trains (from 120s to 90s / 75s)

Passive security not sufficient (power off)
Active security is required (trains have to brake when emergency)

We cannot prove absence of collision.
100% safe systems are miracles

Safety Critical Railway Applications



Modern Automatic Train Protection Software (2015)

Top level implementation

- Imports 55 components
- Specify top level one-cycle function:
  - Compute location, manage kinetic energy, control PSD, trigger emergency braking, etc.

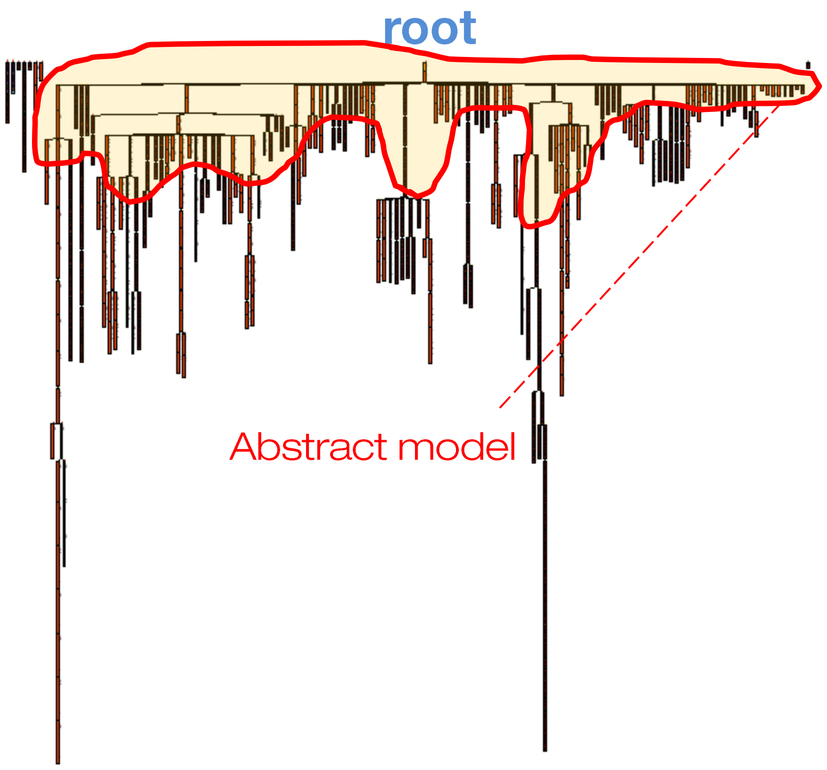
Localization: graph-based algorithms
Energy control: integer arithmetic (braking curve)
Emergency braking: Boolean predicates

Metrics

- 233 machines, 50 kloc
- 46 refinements, 6 kloc
- 213 implementations, 45 kloc
- 3 000 definitions
- 23 000 proof obligations (83 % automatic proof)
- 3 000 added user rules (85 % automatic proof)

Unit and integration testing useless

Automatic Refinement



Modern Automatic Train Protection Software (2015)

≡ Progressive transformation of a model containing all design decisions into a model able to be translated into an imperative language

≡ Refinement engine based on refinement rules

- Abstract model is hand-written
- Concrete model is generated

[2004] Canarsie line (ATP):	*38k lines of handwriten B,* *115k lines of generated B*
[2007] Roissy airport shuttle (ATP+ZC):	*40k lines of handwritten B* *225k lines of generated B*
[2007] Symbolic Calculus Engine:	*200k lines of generated B*
[2015] SysML Compiler :	*300k lines of generated B*

Development cost / 2

Automatic Refinement, L. Burdy, J.M. Meynadier, FM 1999
Return of Experience on Automated Refinement in B, T. Lecomte, SETS 2014

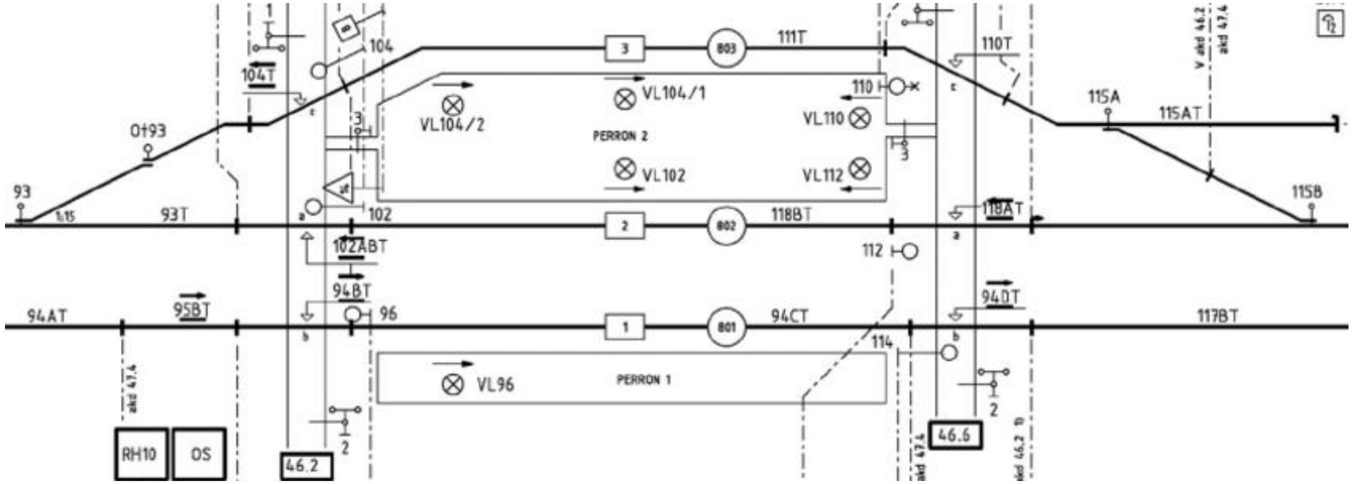
Over Energy

« There is overEnergy iff I can find a track section starting at X2M, complying with the dynamic chaining of blocks, on which I can

- either find a restriction belonging to a block such as the energy on that restriction, computed by summing deltas of energy of all restrictions located between X2MRes and this restriction, is greater than the energy associated to this restriction,

- or find 2 restrictions belonging to the EOA block, one being before the track section under consideration, the other after the track section, such as the energy associated to the EOA by using these restrictions is positive. »

[Extract from Automatic Train Protection specification]



The Corresponding Predicate

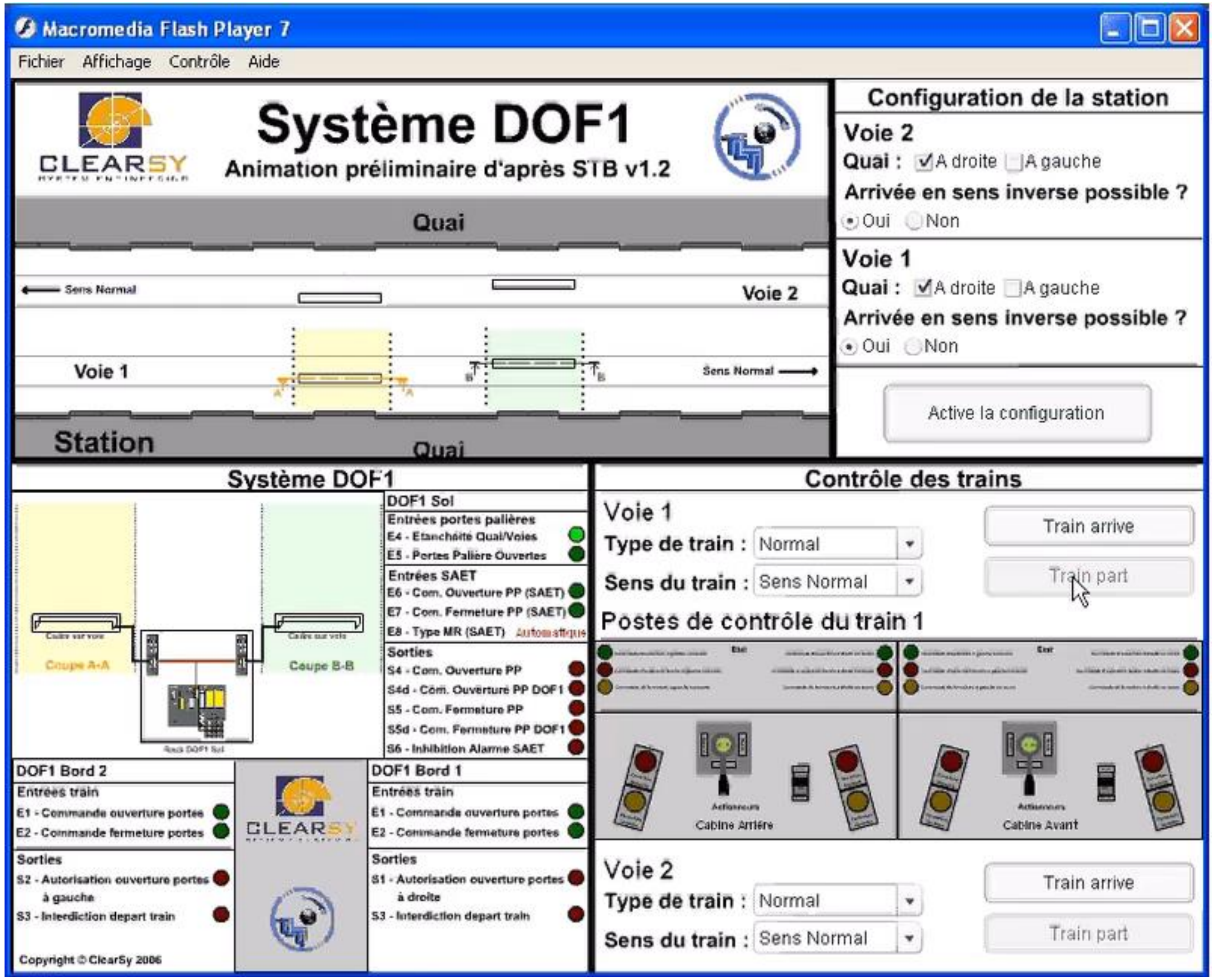
p_over := bool (# (over_track) . ((over_track : seq (t_block * t_direction) & over_track /= {} & first (over_track) = p_X2MBlock |> p_X2MDir & ! ii . (ii : 1 .. size (over_track) - 1 => (over_track) (ii) : dom (sidb_nextBlock)) & ! ii . (ii : 1 .. size (over_track) => sidb_nextBlock ((over_track) (ii)) = (over_track) (ii + 1))) &(# (over_res) . ((over_res : sidb_restrictionApplicable & (# ii . (ii : dom (over_track) & ((prj2 (t_block , t_direction) (over_track (ii))) = c_up => over_res : ran (sgd_blockUpRestrictionSeq ((prj1 (t_block , t_direction) (over_track (ii)))))) & ((prj2 (t_block , t_direction) (over_track (ii))) = c_down => over_res : ran(sgd_blockDownRestrictionSeq ((prj1 (t_block , t_direction) (over_track (ii)))))) & (ii = 1 => not (over_res <= p_X2MRes)) & p_X2MSSWorst + p_X2MDSS + (SIGMA(jj) . (jj : 1 .. ii | SIGMA (pre_res) . (pre_res : t_restriction & ((prj2 (t_block , t_direction) (over_track (jj))) = c_up => pre_res : ran (sgd_blockUpRestrictionSeq ((prj1 (t_block, t_direction) (over_track (jj)))))) & ((prj2 (t_block , t_direction) (over_track (jj))) = c_down => pre_res : ran (sgd_blockDownRestrictionSeq ((prj1 (t_block , t_direction) (over_track (jj)))))) & (jj = 1 => not (pre_res <= p_X2MRes)) & (jj = ii => not (pre_res >= over_res)) | sgd_restrictionDeltaSqSpeed (pre_res)))) > sgd_restrictionSquareSpeed (over_res) & (over_res : sgd_restrictionFront => p_X2MResDist + ((SIGMA (ti) . (ti : 1 .. ii | sgd_blockLength ((prj1 (t_block , t_direction)((over_track) (ti)))))) ({ c_down |>sgd_blockLength (p_X2MBlock) sgd_restrictionAbs (p_X2MRes) , c_up |>sgd_restrictionAbs (p_X2MRes) } (p_X2MDir)) ({ c_down |>sgd_restrictionAbs (over_res) , c_up |>sgd_blockLength ((prj1 (t_block , t_direction) ((over_track) (ii)))) sgd_restrictionAbs (over_res) } ((prj2 (t_block ,t_direction) ((over_track) (ii)))))) + sgd_restrictionLength (over_res) > loc_locationUncertainty + c_trainLength))))) or (# (eoa_res , res_after_eoa , ii) . (eoa_res : t_restriction & res_after_eoa : t_restriction & ii : dom (over_track) & p_EOABlock = (prj1 (t_block , t_direction)(o

Formalities do not save us from complexity

p => eoa_res : ran (sgd_blockUpRestrictionS … eoa_res) <= p_EOAAbs & p_EOAAbs < sgd_restrict … fter_eoa <= ri)) & ((prj2 (t_block , t_direction) (over_track (ii))) = c_down => eoa_res : ran (sgd_blockDownRestrictionSeq (p_EOABlock)) & res_after_eoa : ran (sgd_blockDownRestrictionSeq (p_EOABlock)) & sgd_restrictionAbs (eoa_res) >= p_EOAAbs & p_EOAAbs > sgd_restrictionAbs (res_after_eoa) & ! ri . (ri : ran (sgd_blockDownRestrictionSeq (p_EOABlock)) => ri <= eoa_res or res_after_eoa <= ri)) & p_X2MSSWorst + p_X2MDSS + (SIGMA (jj) . (jj : 1 .. ii | SIGMA (pre_res) . (pre_res : t_restriction & ((prj2 (t_block , t_direction) (over_track (jj))) = c_up => pre_res : ran (sgd_blockUpRestrictionSeq ((prj1 (t_block , t_direction) (over_track (jj)))))) & ((prj2 (t_block , t_direction) (over_track (jj))) = c_down => pre_res : ran(sgd_blockDownRestrictionSeq ((prj1 (t_block , t_direction) (over_track (jj)))))) & (jj = 1 => not (pre_res <= p_X2MRes)) & (jj = ii => pre_res <= eoa_res) | sgd_restrictionDeltaSqSpeed (pre_res))))) ({ c_up |>(sgd_restrictionAccel (eoa_res) * ((sgd_restrictionAbs (res_after_eoa) p_EOAAbs) / 1024)) / 2 , c_down |>(sgd_restrictionAccel (eoa_res) * ((p_EOAAbs sgd_restrictionAbs (res_after_eoa)) / 1024)) / 2 } ((prj2 (t_block , t_direction) (over_track (ii)))))) > 0)) or (# (eoa_res , ii) . (eoa_res : t_restriction & ii : dom (over_track) & (ii = 1 => not (eoa_res <= p_X2MRes)) & p_EOABlock = (prj1 (t_block , t_direction) (over_track (ii))) & ((prj2 (t_block , t_direction) (over_track (ii))) = c_up => eoa_res : ran (sgd_blockUpRestrictionSeq (p_EOABlock)) & eoa_res = last(sgd_blockUpRestrictionSeq (p_EOABlock)) & sgd_restrictionAbs (eoa_res) <= p_EOAAbs) & ((prj2 (t_block , t_direction) (over_track (ii))) = c_down => eoa_res : ran(sgd_blockDownRestrictionSeq (p_EOABlock)) & eoa_res = last (sgd_blockDownRestrictionSeq (p_EOABlock)) & sgd_restrictionAbs (eoa_res) >= p_EOAAbs) & p_X2MSSWorst + p_X2MDSS + (SIGMA (jj) . (jj : 1 .. ii | SIGMA (pre_res) . (pre_res : t_restriction & ((prj2 (t_block , t_direction) (over_track (jj))) = c_up => pre_res : ran(sgd_blockUpRestrictionSeq ((prj1 (t_block , t_direction) (over_track (jj)))))) & ((prj2 (t_block , t_direction) (over_track (jj))) = c_down => pre_res : ran(sgd_blockDownRestrictionSeq ((prj1 (t_block , t_direction) (over_track (jj)))))) & (jj = 1 => not (pre_res <= p_X2MRes)) & (jj = ii => not (pre_res >= eoa_res)) | sgd_restrictionDeltaSqSpeed (pre_res)))) + ({ c_up |> (sgd_restrictionAccel (eoa_res) * ((p_EOAAbs sgd_restrictionAbs (eoa_res)) / 1024)) / 2 , c_down |> (sgd_restrictionAccel (eoa_res) * ((sgd_restrictionAbs (eoa_res) p_EOAAbs) / 1024)) / 2 } ((prj2 (t_block , t_direction) (over_track (ii))))) > 0))

Systems

Platform screen doors (PSD)

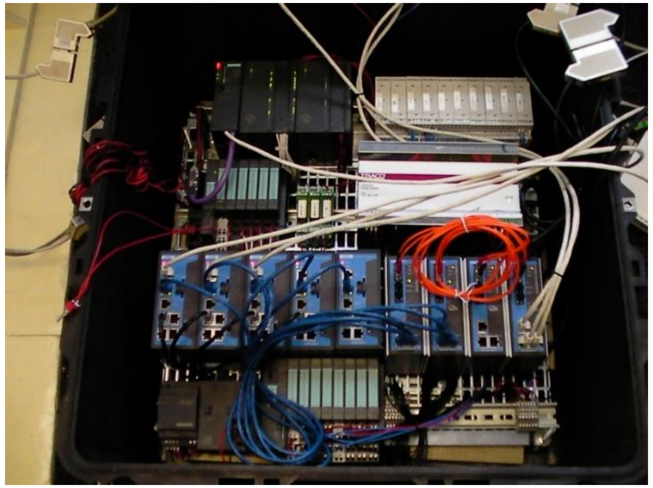


Paris line 1- PSD model animation (**Event-B**)

Platform Screen Doors Control

- System to install to prepare driverless operation
- Avoid people to die (pushed) on tracks

- No direct communication with the train: train arrival and door opening to be detected with diverse sensors
- SIL4: <u>one failure every 10 000 years</u>
- 99,999% reliability: <u>one train max missed per year</u>
- To be developed from scratch in 6 months



Safe and Reliable Metro Platform Screen Doors Control/Command Systems, T. Lecomte, SBMF 2008

Virtual becomes real

≡ Connecting things with the real station

Et voilà !

Formal Data Validation

≡ **Proving parameters (constants)**

- What is the use of a formally proven software if some of its (non trivial) parameters are wrong ?
- Initially metro line static data used by the automatic pilot (software) to drive safely

≡ **Data Validation**

- Automatic check of large data sets against properties
- Properties : international standards, national regulations, manufacturer habits, customer requirements
- Initially metro line static data used by the automatic pilot (software) to drive safely
- Model-checking applied to

	A	B	C	D	E	F	G	H	I
1	Name	ID	IP	Type	UpLink	DownLink	Length	GPS 1	GPS 2
2	Route_tx_001	243		R	Route_tx_005	Route_vx_002	345		
3	Route_vx_002	128		R	Route_vx_002	EndLine_000	128		
4	Switch_w_003	256	192.16.4.55	S	Route_vx_128	Route_tx_006	23		
5	Relay_s_004	12	192.16.4.10	Y				N 50.85 963	O 6.84 201
6	Route_tx_005	3		R	Route_tx_006	Route_vx_128	291		
7	Relay_s_001	55	192.16.4.125	Y					
8	Route_tx_006	22		R	EndLine_001	Route_vx_002	110		
9	Route_vx_128	127		R	Route_tx_006	Route_vx_002	145		
10	Switch_w_009	242	192.16.4.10	S	Route_vx_128	Route_tx_005	34		
11	EndLine_000	0		E		Route_vx_002	1		
12	EndLine_001	1		E	Route_vx_002		1		
13	Signal_xs_002	32	192.16.4.12	G	Route_vx_128		22		
14	Signal_xs_003	33	192.16.4.13	G	Route_tx_006		51		
15	Balise_b_001	301		B	Route_vx_128		0	N 50.85 933	O 6.84 508
16	Balise_b_002	302		B	Route_tx_005		0	N 50.86 123	O 6.84 550

Up to 100,000+ raw data chunks

Are they

- Consistent ?
- Correct ?
- Safe ?

Formal Data Validation in the Railways, T. Lecomte, SSS 2016

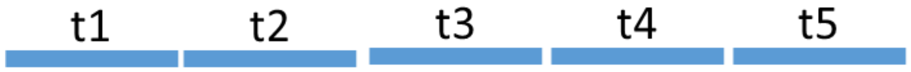
Using B and ProB for Data Validation Projects, D. Hansen, D. Scheiner, M. Leuschel, book chapter 2016

From Animation to Data Validation: The ProB Constraint Solver 10 Years On, M. Leuschel, J. Bendisposto, I. Dobrikov, S. Krings, D. Plagge, book chapter 2014

Modelling Language

≡ Modelling language based on set theory and first order predicates logic (B mathematical language)

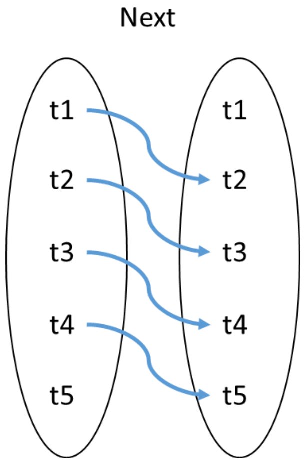
Let the set $TrackCircuit = \{t1, t2, t3, t4, t5\}$



Let the function $Next \in TrackCircuit \nrightarrow TrackCircuit$

Example: $Next(t1) = t2$, $Next(t2) = t3$, $Next(t3) = t4$, $Next(t4) = t5$

$Next = \{t1 \mapsto t2, t2 \mapsto t3, t3 \mapsto t4, t4 \mapsto t5\}$



One may define $FirstTrackCircuit \in TrackCircuit - ran(Next)$

Let the function $KpAbs \in TrackCircuit \rightarrow \mathbb{N}$

$\forall x \cdot (x \in TrackCircuit \land x \in dom(Next) \Rightarrow KpAbs(Next(x)) > KpAbs(x))$

Rule

- A rule requires *a minima*:
  - A selection predicate
  - A verification predicate
  - An error message if the verification fails

T_r^0	T_r^1	T_r^2	T_r^3
S_0	S_2	S_3	S_5
S_1		S_4	

Let S_g be the signals of the track T_r

Each signal S_i of S_g is defined in the table T_r

« The signal % used by track % is not defined », S_i, T_r

S_g	S_0	S_1	S_2	S_3	S_4	S_5

ANY
  S, i
WHERE
  i ∈ dom(Tr) &
  S ∈ Tr(i)
EXPECTED
  S ∈ Sg
END

Tr is a function defined for the values 0 to 3 (its domain).

$Tr(0) = \{S_0, S_1\}$

$Tr(1) = \{S_2\}$

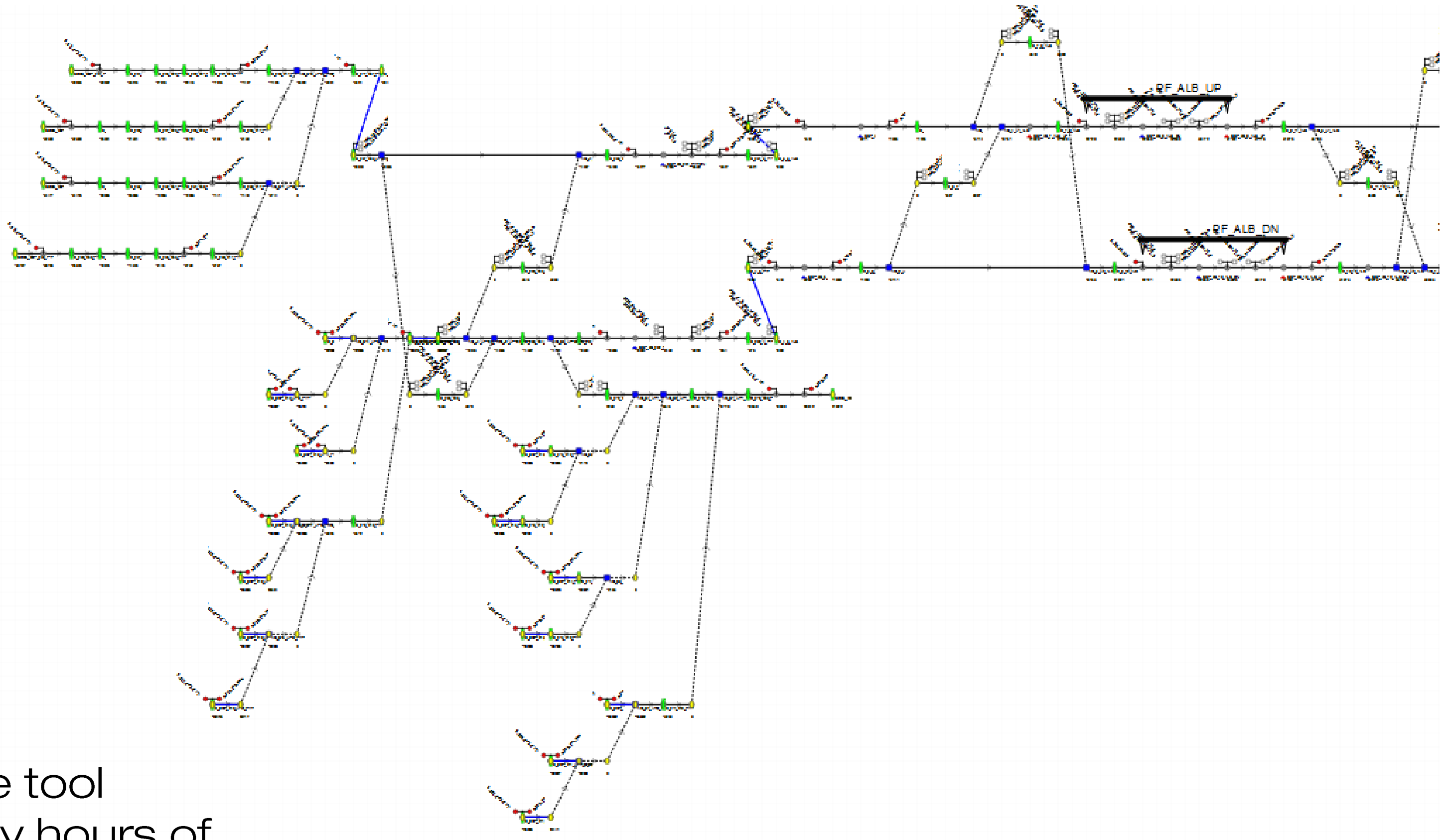
$Tr(2) = \{S_3, S_4\}$

$Tr(3) = \{S_5\}$

Formal Data Validation

Consistency, correctness, safety

- Expressed with the mathematical language of B
- Work well with graph-based properties
- Provide counter examples when errors found



Model-checking

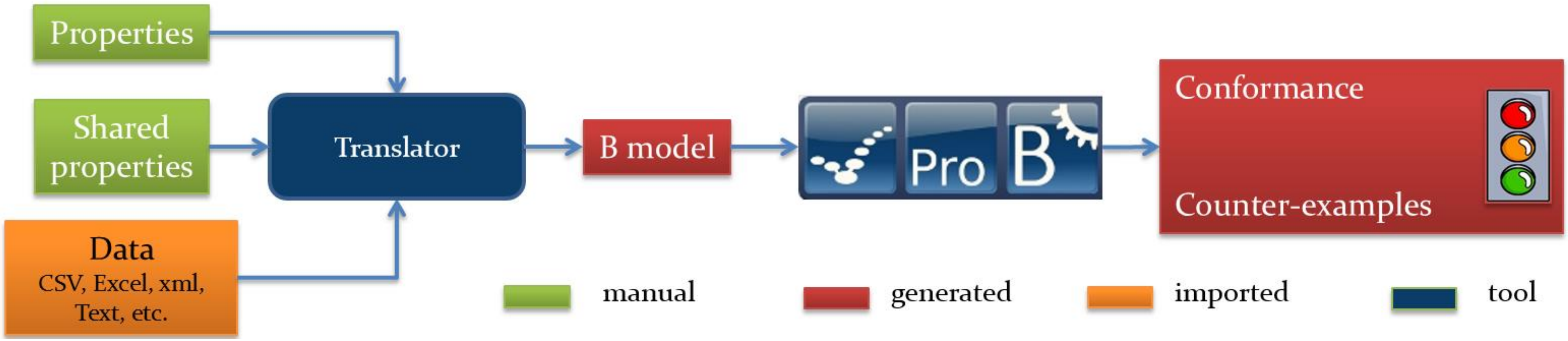
- Performed with ProB
- Rodin (Siemens) and Alstom have funded development and validation to obtain mature tool
- Replace months of (boring) engineer work by hours of computer verification
- Engineer models properties (1 000/line)

Formal Data Validation

≡ Industry-Ready **[PUSH-BUTTON !]**

- Deal with permanent changes in data and properties
- Redundant tools to obtain diversity
- Rules reused from one project to another
- More than 30 sites verified including:
    - Singapore, Panama, Ryad, Mecca

Improve the level of confidence of the V&V



Formal Data Validation

- Verbosity, implicit made explicit

For each GradientTopology (GradientTopology.BOT-Zone) totally included in a segment, a Gradient (Gradient.BOT-Zone) is created with the same attributes.

For GradientTopology intersecting different segments, several Gradients (Gradient.BOT-Zone) are created so that each of them is located in only one segment.

When the gradient is constant (GradientTopology.isConstant = Yes):

- the variable gradient information (Gradient.VariableGradient) is not set.
- the constant gradient information is set with the same information of GradientTopology for both parts.
- the elevationDifference.elevationEnd of the part1 and elevationDifference.elevationStart of the part2 (reference to the above figure) are equal to elevationStart + gradient*Length1.
- the inf

Formalities do not save us from complexity

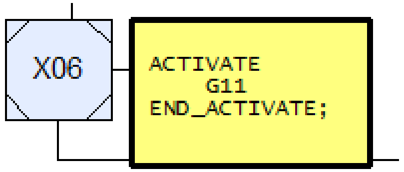
When the gradient is not constant (GradientTopology.isConstant = No):

- the constant gradient information (ConstantGradient) is not set.
- the elevationDifference.elevationEnd of the part1 and elevationDifference.elevationStart of the part2 (reference to the above figure) are equal to elevationStart +2*radius*sin(Length1/(2*radius))*sin(gradientStart +Length1/ (2*radius)).
- the information radius and transitionCurveType of the variableGradient information are the same for both parts (as initial GradientTopology information) .
- the information gradientEnd for part1 and gradientStart of part2 for variableGradient information are set to (gradientEnd-gradientStart)/(Length1 +Length2)*Length1 + gradientStart.
- the information isConstant is set to No for both Part.

Other applications

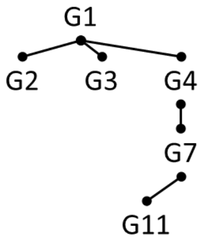
≡ Validation of compilation principles of a SIL2 IDE (TGV Atlantique)

- Compilation of hierarchical grafcets into binary file



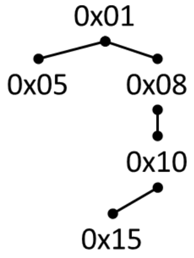

```
[0x00198c] LANCE_GRAF 0x15
```

- Property: sub-grafcets activated in the binary file should correspond to sub-grafcets activated in the models



GRF = {main, G1, G2, G3, G4, …. }
next $\in$ GRF $\leftrightarrow$ GRF
next = { …, G7 $\mapsto$ G11, …}

ADR = {0x01, 0x13, 0x15, …}
suiv $\in$ ADR $\leftrightarrow$ ADR
suiv = { … , 0x10 $\mapsto$ 0x15, …}

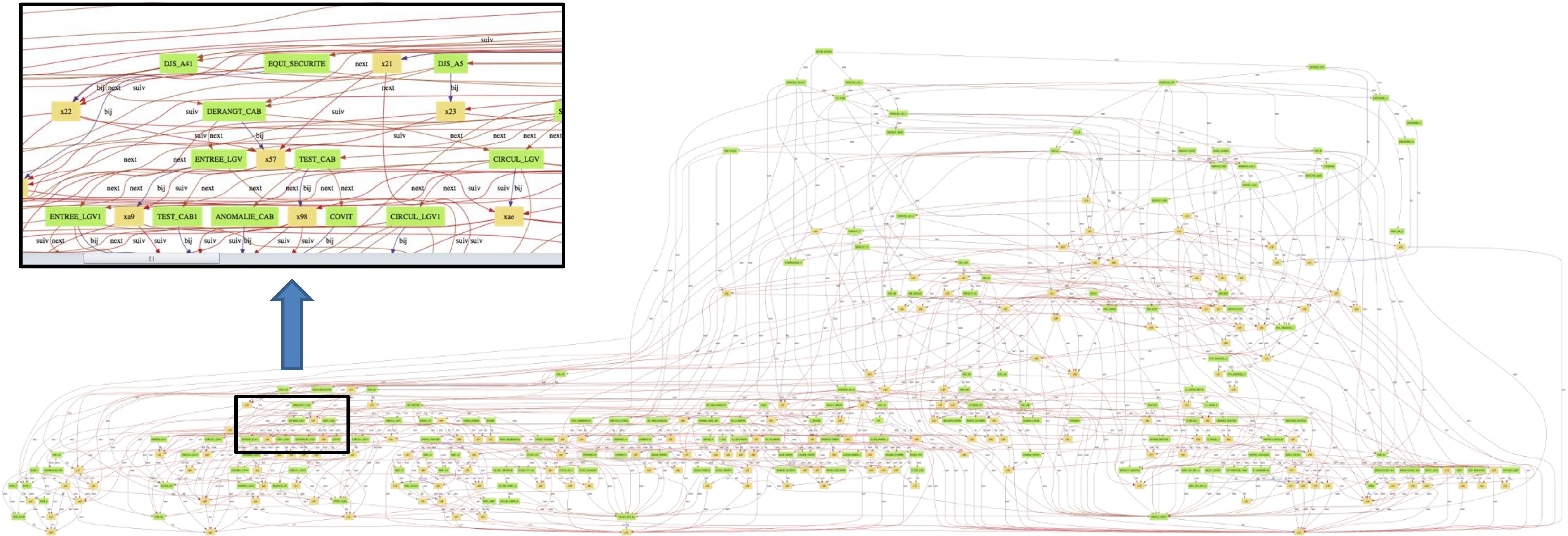


- It exists a bijection **bij** which associates to any GRF node an ADR node such as successors match.

$$\mathbf{bij} \in \mathbf{GRF} \rightarrowtail\!\!\!\!\twoheadrightarrow \mathbf{ADR} \;\&\; \forall \mathbf{xx}.(\mathbf{xx} \in \mathbf{GRF} \Rightarrow \mathbf{bij}[\mathbf{next}[\{\mathbf{xx}\}]] = \mathbf{suiv}[\mathbf{bij}[\{\mathbf{xx}\}]])$$

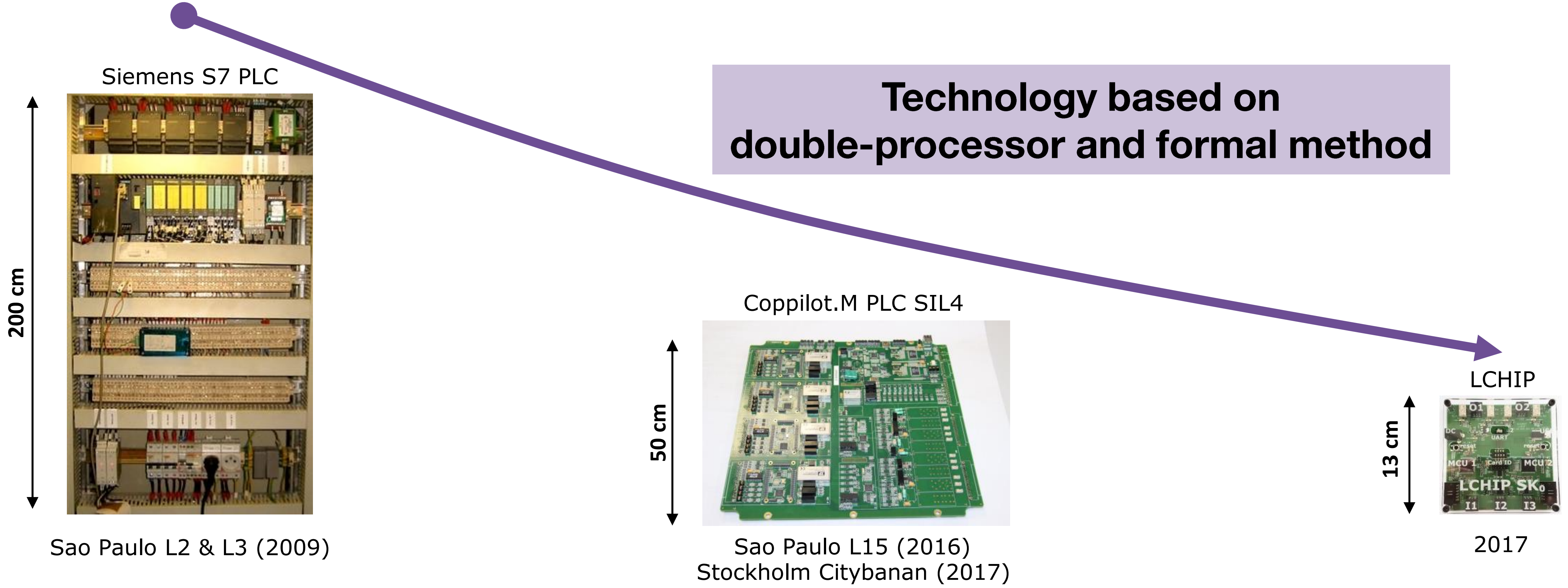
Other applications

≡ Matching between 2 graphs, containing 167 nodes, in less than 10 s



CLEARSY Safety Platform

- Need for a technical solution to overcome difficulties to develop SIL3/SIL4 systems
  - Require rare human resources to complete successfully
  - Very short delays (~6 months) to design new systems
  - Off-the-shelf block solutions difficult to adapt



Genesis

Technology experimented several times

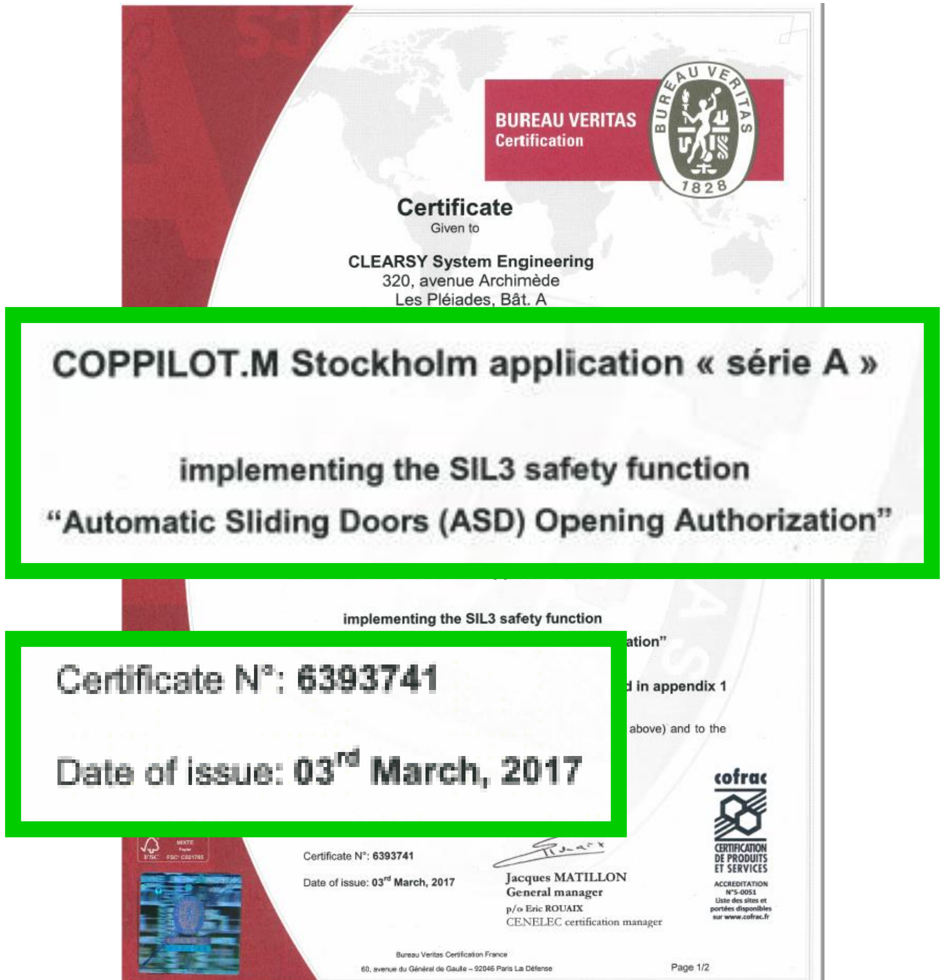
Platform screen doors controler installed in Stockholm (Citybanan)

Sao Paulo L15 (2016)

Stockholm Citybanan (2017)

Gateway SATURN (2016)

Technology certifiable



BUREAU VERITAS Certification

Certificate

Given to

CLEARSY System Engineering
320, avenue Archimède
Les Pléiades, Bât. A

COPPILOT.M Stockholm application « série A »

implementing the SIL3 safety function
"Automatic Sliding Doors (ASD) Opening Authorization"

Certificate N°: 6393741

Date of issue: 03rd March, 2017

Jacques MATILLON
General manager
p/o Eric ROUAIX
CENELEC certification manager

Page 1/2

LCHIP in a Nutshell

SOFTWARE & HARDWARE
PLATFORM
FOR SAFETY CRITICAL APPLICATI[...]

Factory
to generate a function
that executes safely

DESIGN → ***EXECUTION***

≡ ease the development of safety critical applications

- Cover the whole development cycle
- Safety principles are built-in
- Based on a formal language and related proof tools
- Mathematical proof replaces unit and integration testing
- Avoid expert transactions over multiple languages

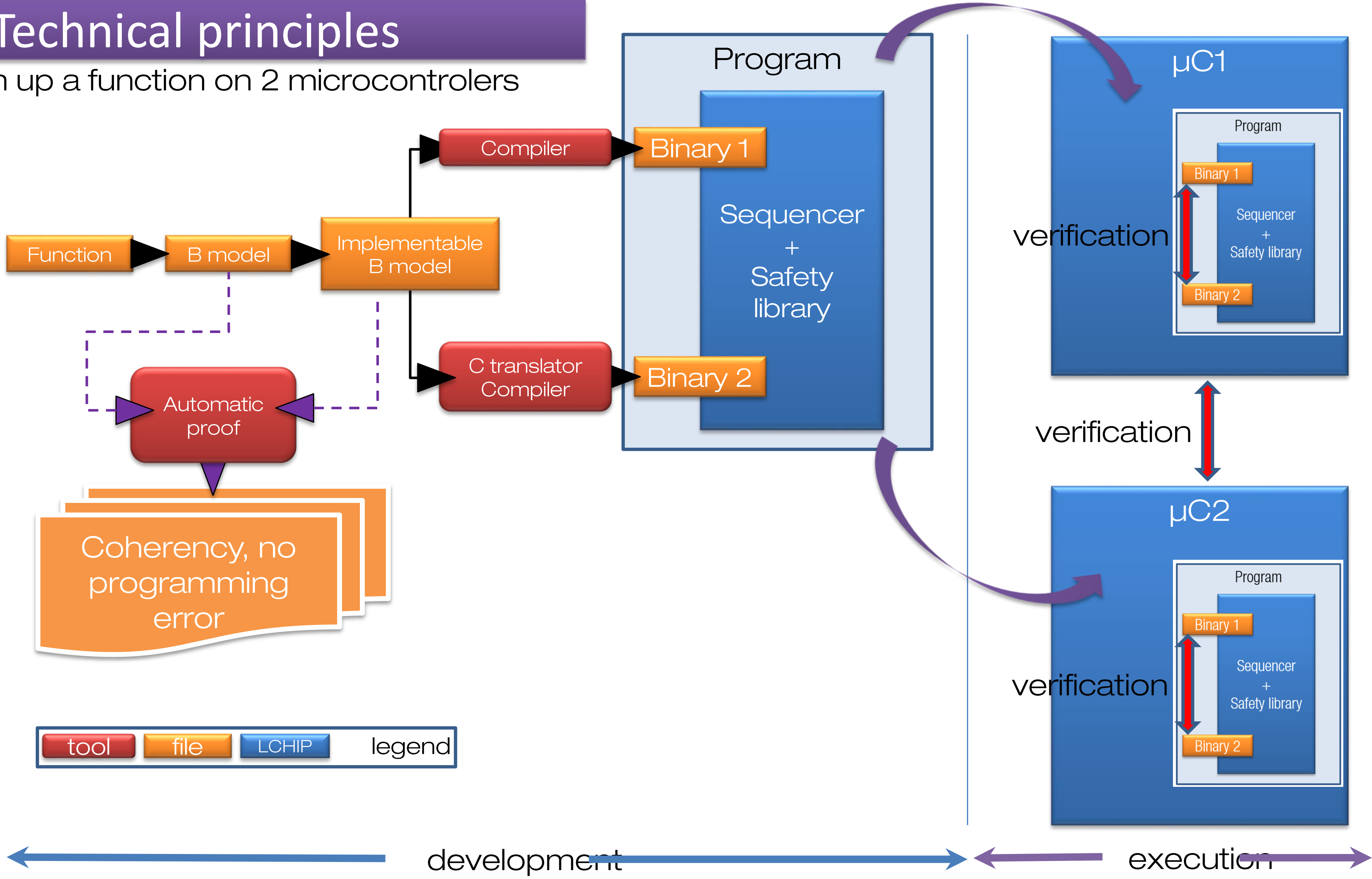
≡ ease the certification of safety critical applications

- Safety cannot be altered by the developer
- Comes along with a certification kit

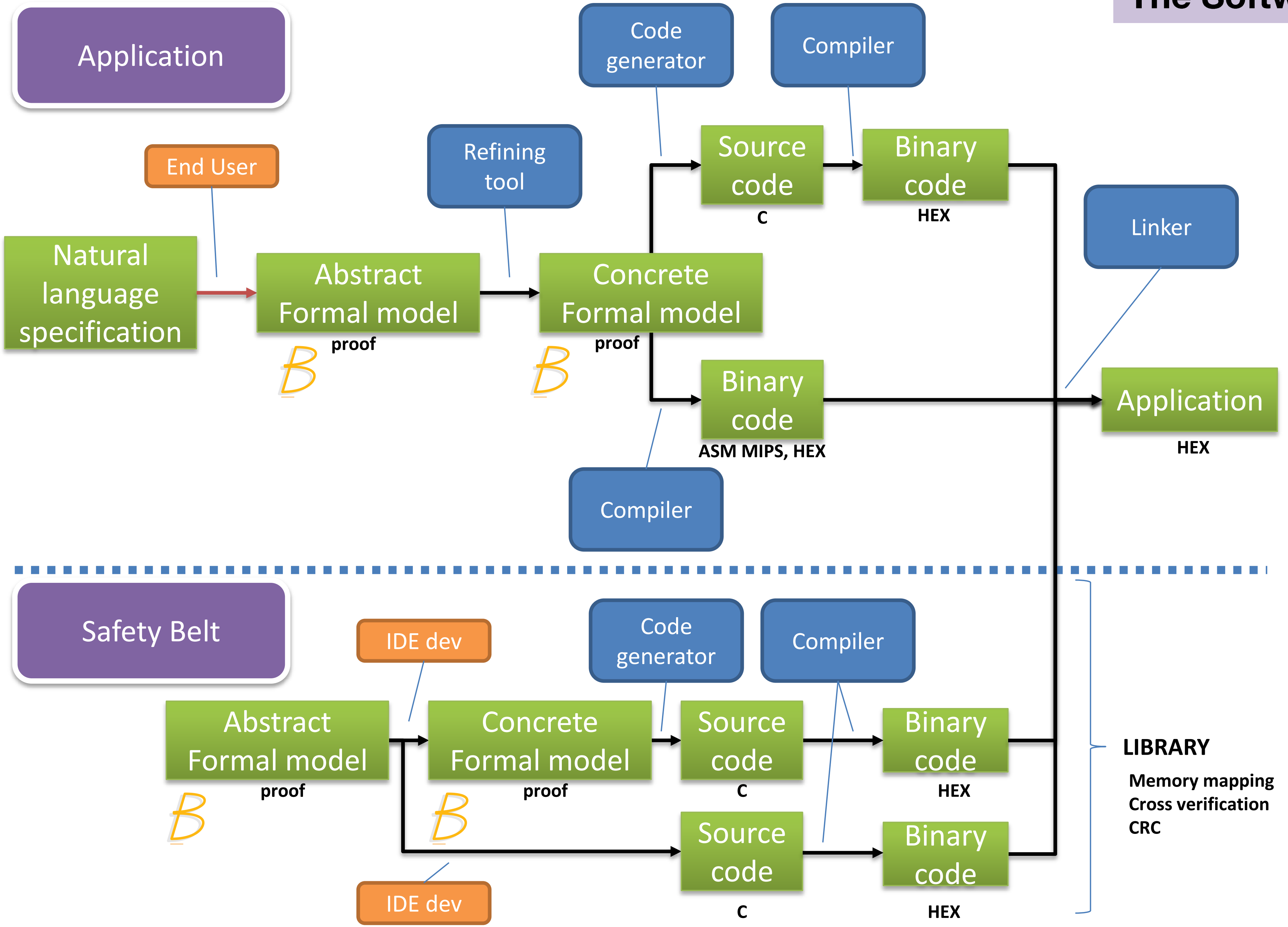
Disclaimer: The developer is responsible for the safety demonstration

Technical principles

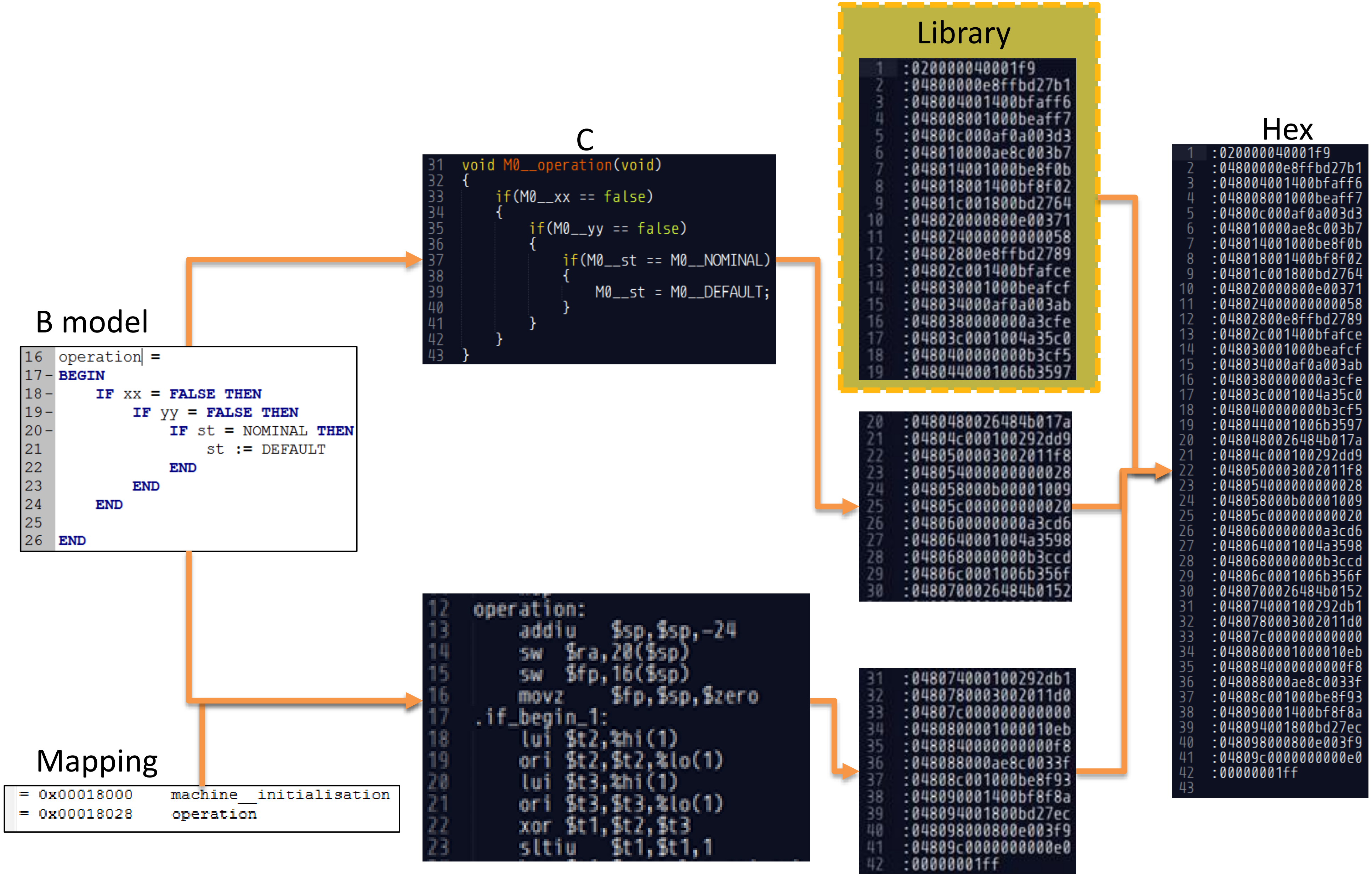
To beam up a function on 2 microcontrolers



The Software Factory

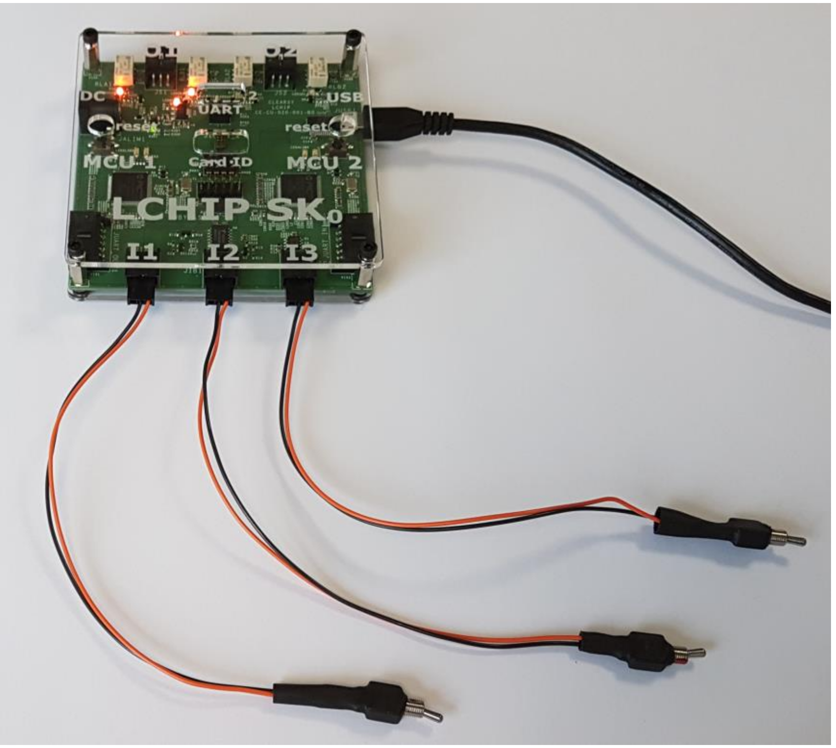


Binary



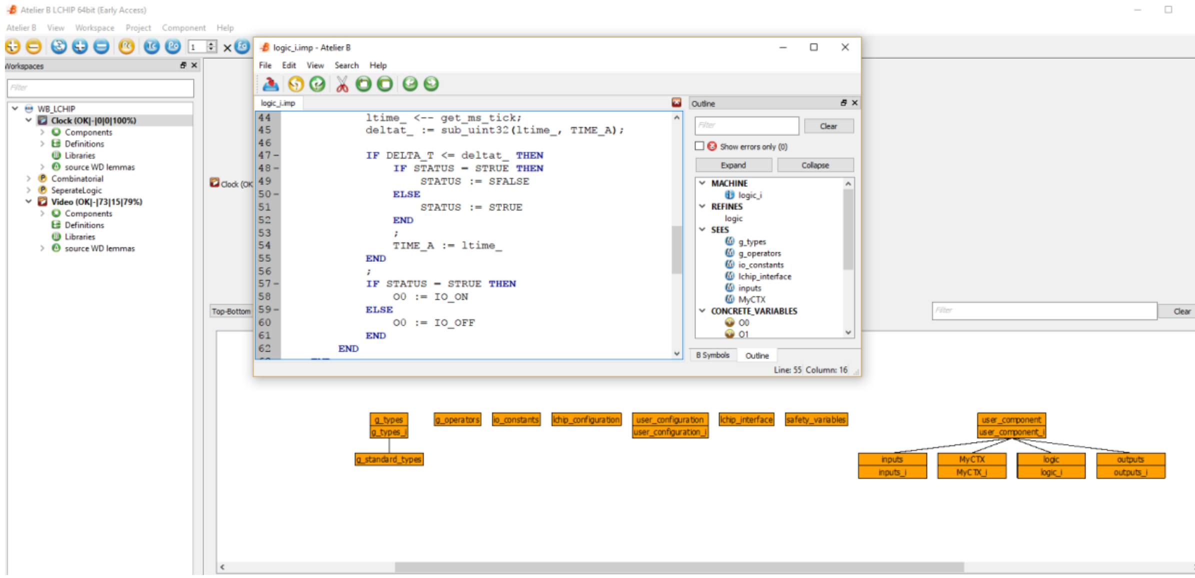
B model
operation =
BEGIN
IF xx = FALSE THEN
IF yy = FALSE THEN
IF st = NOMINAL THEN
st := DEFAULT
END
END
END
END
C
void M0__operation(void)
{
if(M0__xx == false)
{
if(M0__yy == false)
{
if(M0__st == M0__NOMINAL)
{
M0__st = M0__DEFAULT;
}
}
}
}
operation:
addiu $sp,$sp,-24
sw $ra,20($sp)
sw $fp,16($sp)
movz $fp,$sp,$zero
.if_begin_1:
lui $t2,%hi(1)
ori $t2,$t2,%lo(1)
lui $t3,%hi(1)
ori $t3,$t3,%lo(1)
xor $t1,$t2,$t3
sltiu $t1,$t1,1
Mapping
= 0x00018000 machine__initialisation
= 0x00018028 operation
Library
Hex
:020000040001f9
:04800000e8ffbd27b1
:048004001400bfaff6
:048008001000beaff7
:04800c000af0a003d3
:048010000ae8c003b7
:048014001000be8f0b
:048018001400bf8f02
:04801c001800bd2764
:048020000800e00371
:048024000000000058
:04802800e8ffbd2789
:04802c001400bfafce
:048030001000beafcf
:048034000af0a003ab
:04803800000000a3cfe
:04803c0001004a35c0
:048040000000b3cf5
:0480440001006b3597
:048048026484b017a
:04804c00100292dd9
:048050003002011f8
:048054000000000028
:048058000b00001009
:04805c000000000020
:048060000000a3cd6
:048064001004a3598
:0480680000000b3ccd
:04806c0001006b356f
:04807000026484b0152
:048074000100292db1
:04807800030002011d0
:04807c000000000000
:04808000010000010eb
:048084000000000f8
:048088000ae8c0033f
:04808c001000be8f93
:048090001400bf8f8a
:048094001800bd27ec
:048098000800e003f9
:04809c0000000000e0
:00000001ff

Starter Kit SK_0



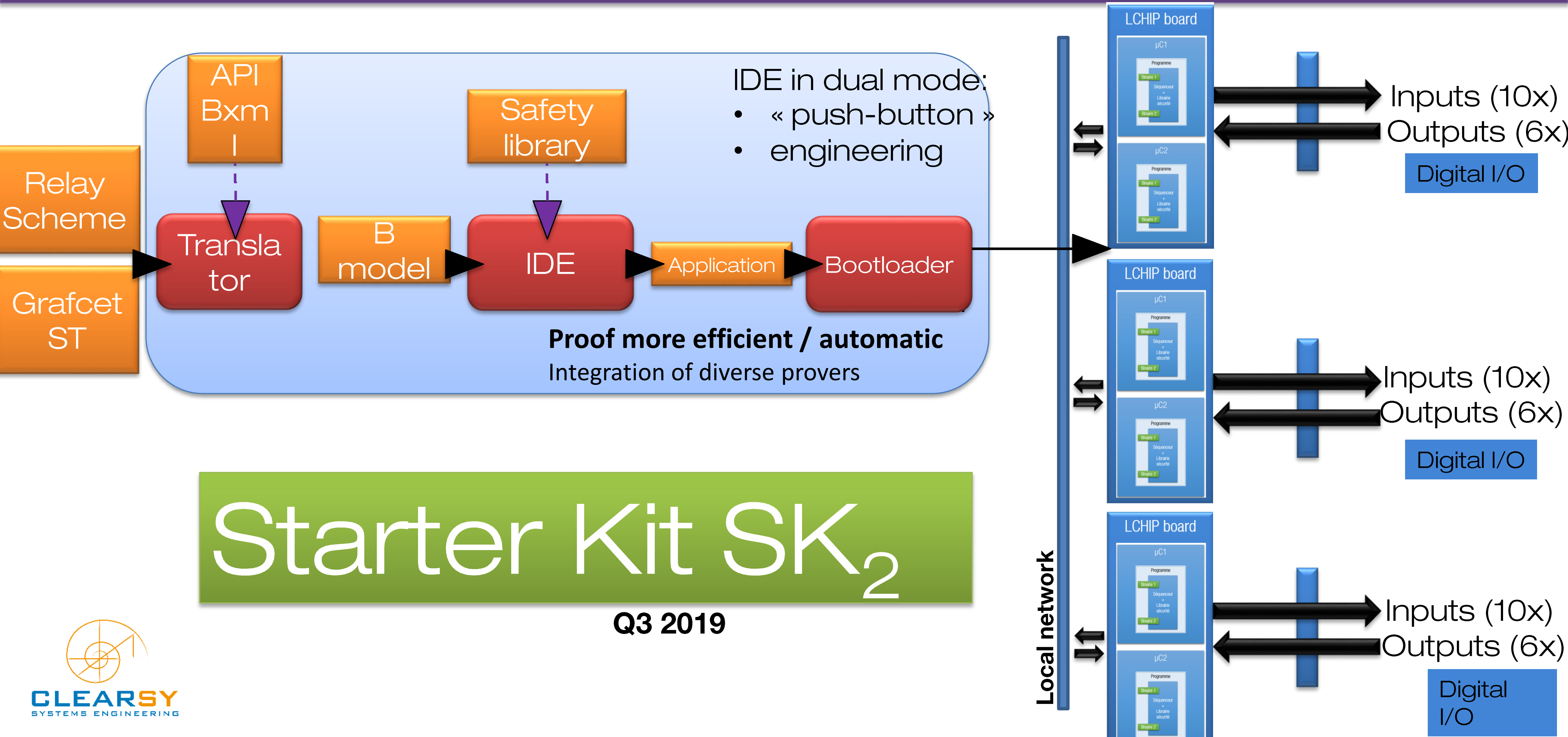
USB-C connector
Power, upload, monitor

Switches
Simulate digital inputs



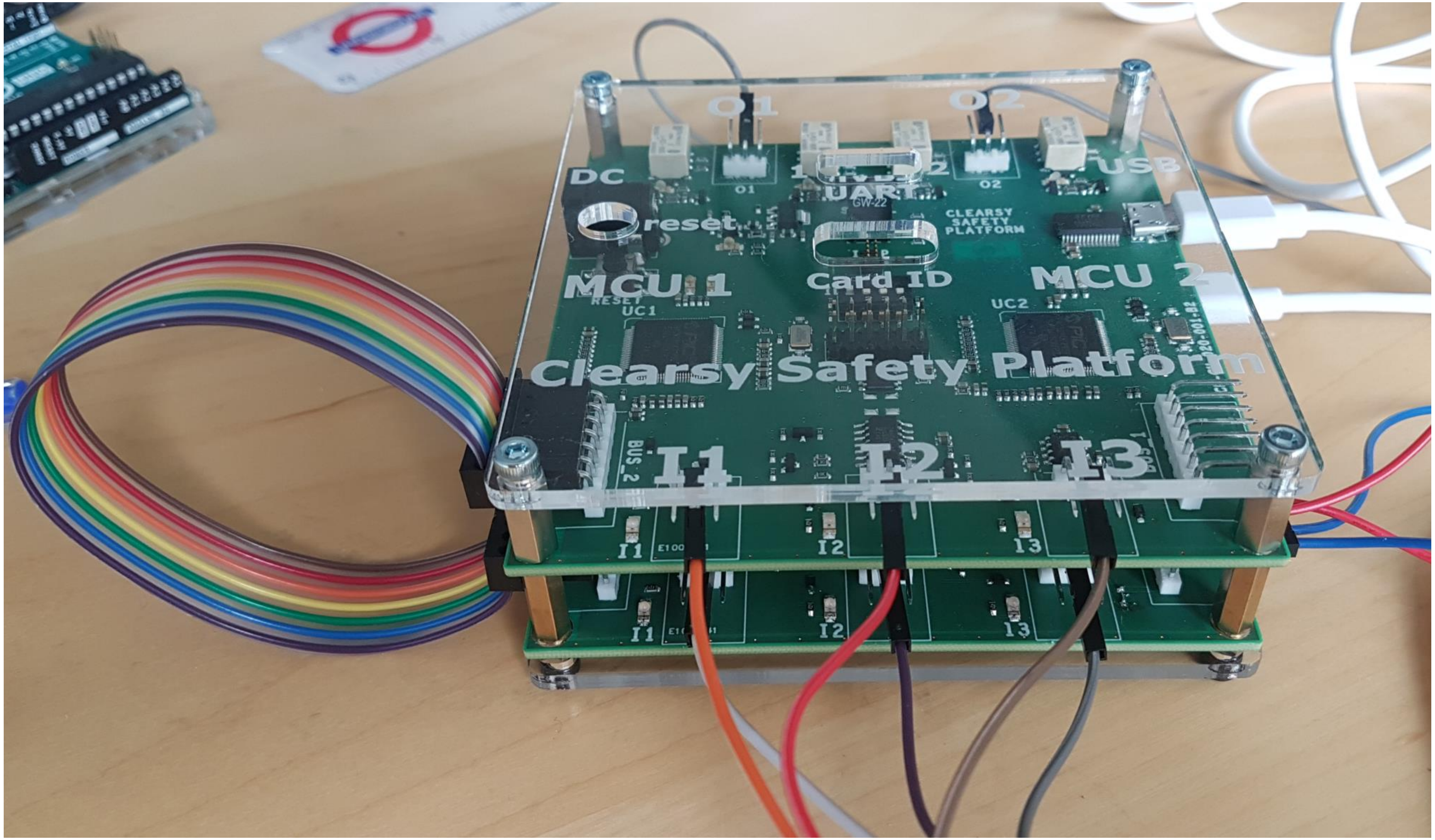
Atelier B 4.4 LCHIP Early access
Model, prove, compile, upload

Roadmap



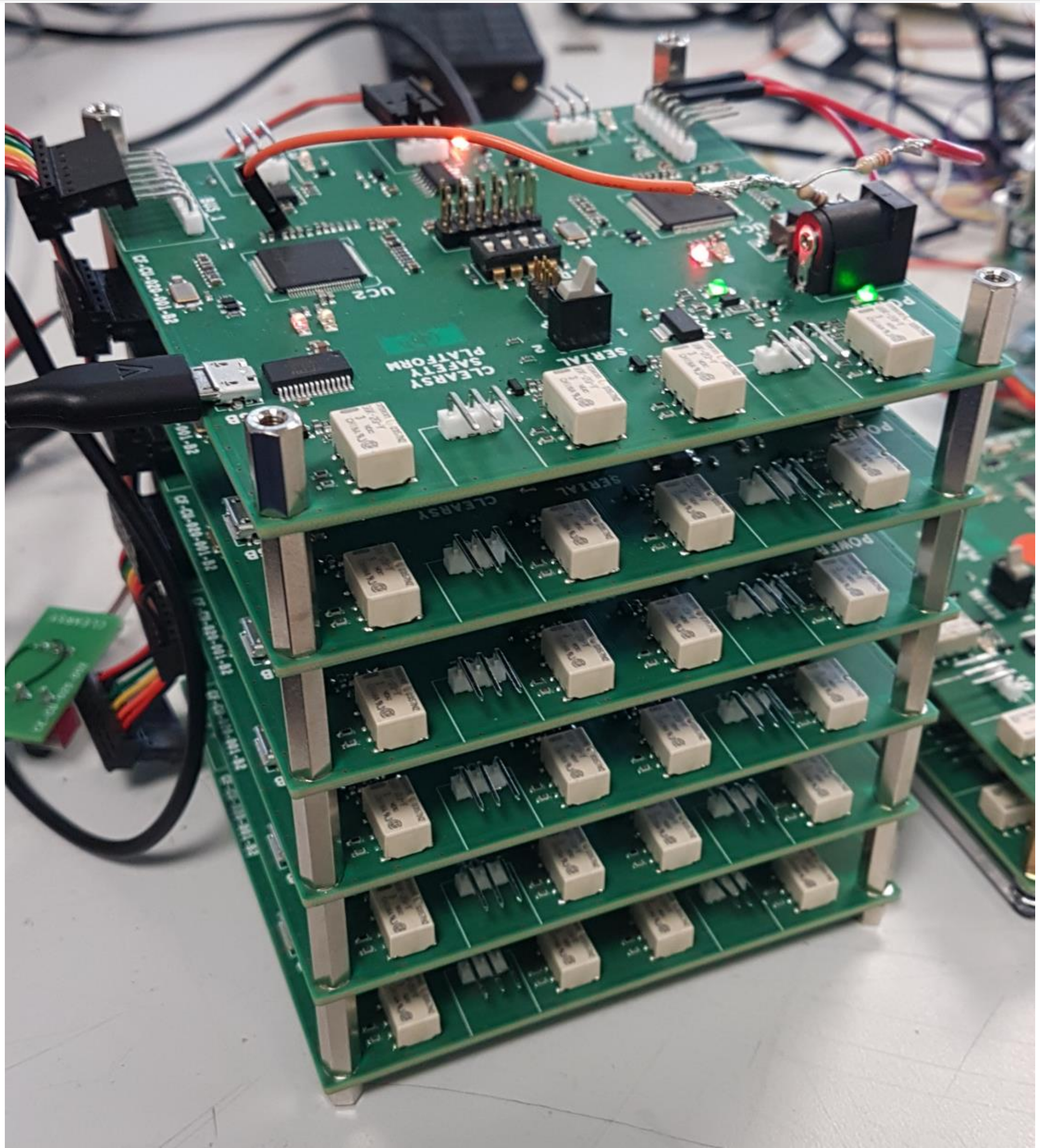
CLEARSY
SYSTEMS ENGINEERING

Roadmap



2 boards connected sharing inputs status

Roadmap



More boards …

FM successes

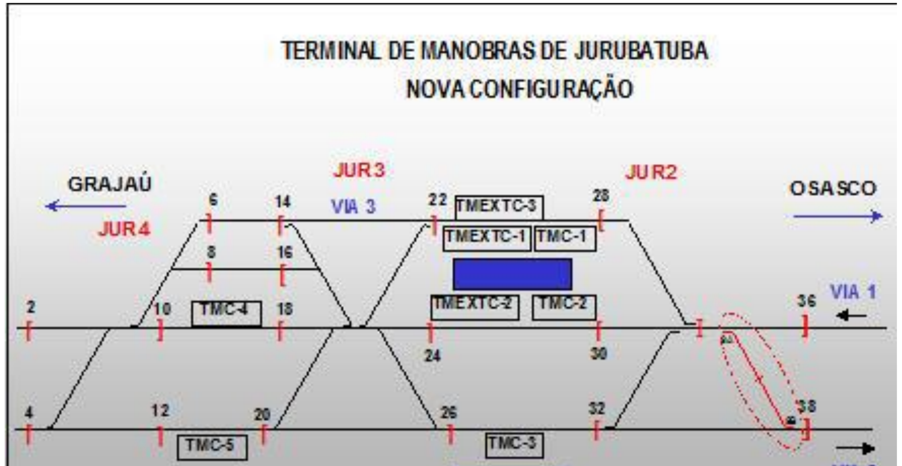
Line 1: 350 M passengers going through PSDs in 4 years, no problem
7 B+ passengers went through the PSDs we control

Line 14: ATP released in 1998, still in version 1.0, no bug detected so far
Most renovated lines in Paris are going to use B

LCHIP is being installed on-board all Thales Toronto trains (remote los)

Line 4: system-level study “without the correct by construction” stamp allowed to detect a major leak in the interoperability specification

7 independent timers related to identified situations may lead to an unwanted collision



Perspectives

Many opportunities to apply formal methods:

- Emerging domains where safety is not (yet) a subject of interest
- Increasing complexity of our (interconnected) world
- Autonomous vehicles

Many traps as well:

- A software is not safe by itself. It has to be considered in its environment
- Address the problem fully
- FM are not mandatory, most existing systems do not embed them
- Proof has to converge towards full automation
- Need for Leonard(s) da Vinci with FM skills

massive open online course

https://mooc.imd.ufrn.br/