Hackathon
JUL2023

CLEARSY
Safety Solutions Designer

AIX
LYON
PARIS
STRASBOURG

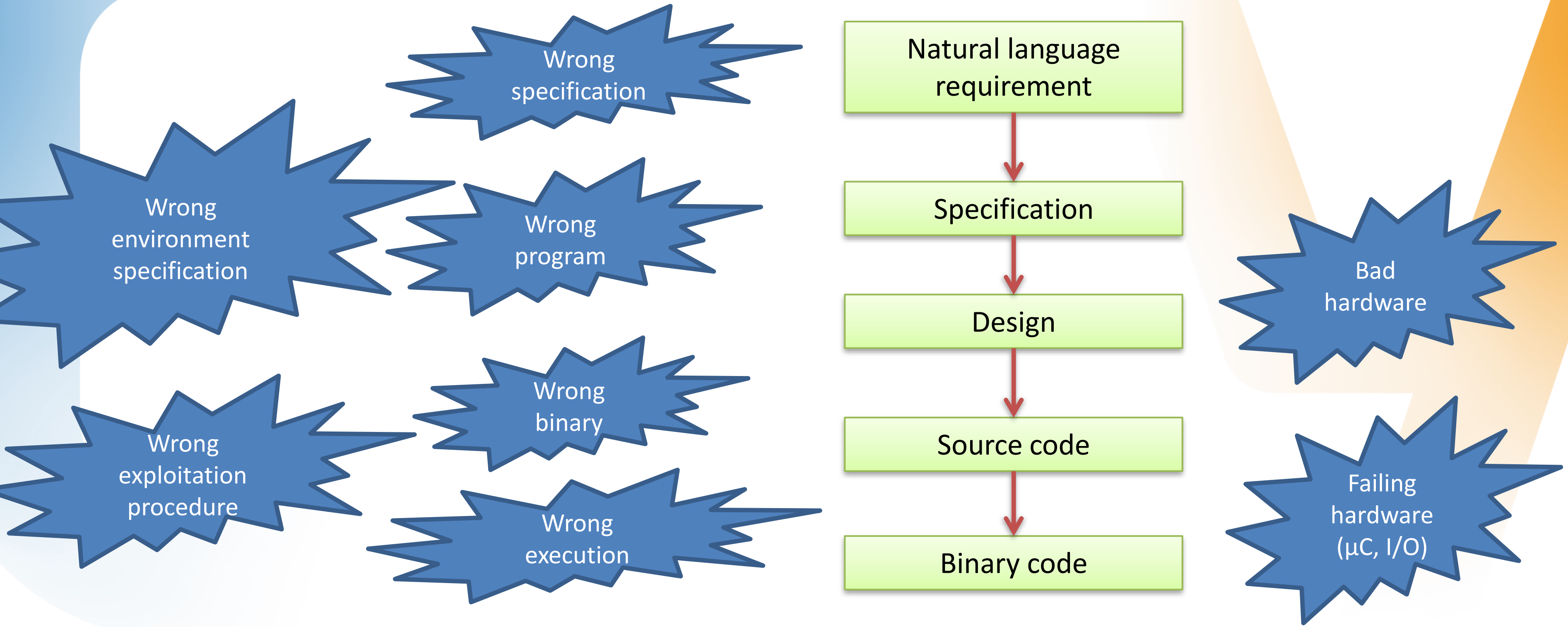
WWW.CLEARSY.COM

Introduction to Safety

Thierry Lecomte
R&D Director

THIERRY.LECOMTE@CLEARSY.COM

Safety is about Failing System



Standards for Safety Critical Systems

- Domain-specific standards
- Recommandations (REX, best practices)
  - No definitive recipe to produce safe systems
  - Cover SW, HW and development process
- Safety case
  - Demonstration that feared event won't happen more frequently than expected
  - Quality & correct development required
  - **Safety by-design !**

- FDA: healthcare
- 26262: automotive
- EN5012{6,8,9}: railways
- IEC61508: industry
- DO178: aeronautics
- CC: µelectronics

Safety Levels for Safety Critical Systems

- Safety Integrity Level (*SIL*)
  - Level 3: 1 failure every 100 years (10^{-7}/h)
  - Level 4: 1 failure every 10 000 years (10^{-9}/h)
- Systems = SW + HW + Environment
  - Specification error
  - Development error, programmation error, bad compilation
  - Wrong execution
  - **2+ processors in parallel, protecting mechanisms**, etc.

Formal Methods for Safety Critical Systems

- Accepted for certification
  - Higly recommended (EN50128, IEC61508)
  - Mandatory (Common Criteria)
- AI not recommended

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.

What for ? Avoid a 60 years old bug

```
public static int binarySearch(int[] a, int key) {
  int low = 0;
  int high = a.length - 1;

  while (low <= high) {
    int mid = (low + high) / 2;
    int midVal = a[mid];

    if (midVal < key)
      low = mid + 1
    else
      if (midVal > key) high = mid - 1;
      else return mid; // key found
  }
  return -(low + 1); // key not found
}
```

What happens if `low+high` > $2^{31} - 1$?

- 1946 : first publication of the algorithm
- 1962 : first bug-free publication
- 2006 : bug present in the Oracle JDK

Formal Methods

Mathematics (language, tool) to demonstrate “things”

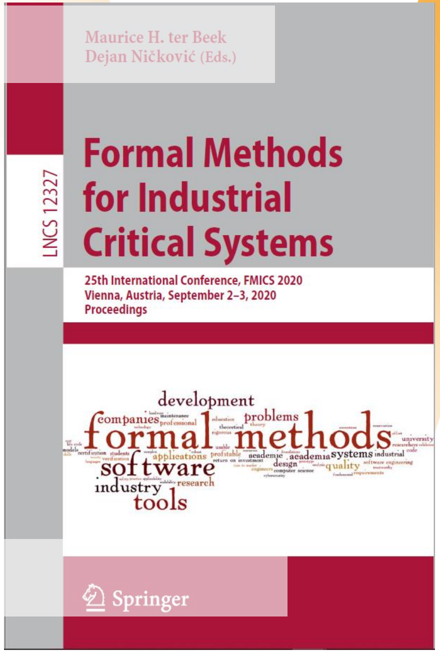
- Modelling design or verification a posteriori

100+ formalisms & formal tools *https://fme-teaching.github.io/*

- Level 0: Formal specification + informal development
- Level 1: Formal development and formal verification
- Level 2: Theorem provers for fully machine-checked proofs

- 50+ years of research
- 25+ years of research for Industrial Critical Systems

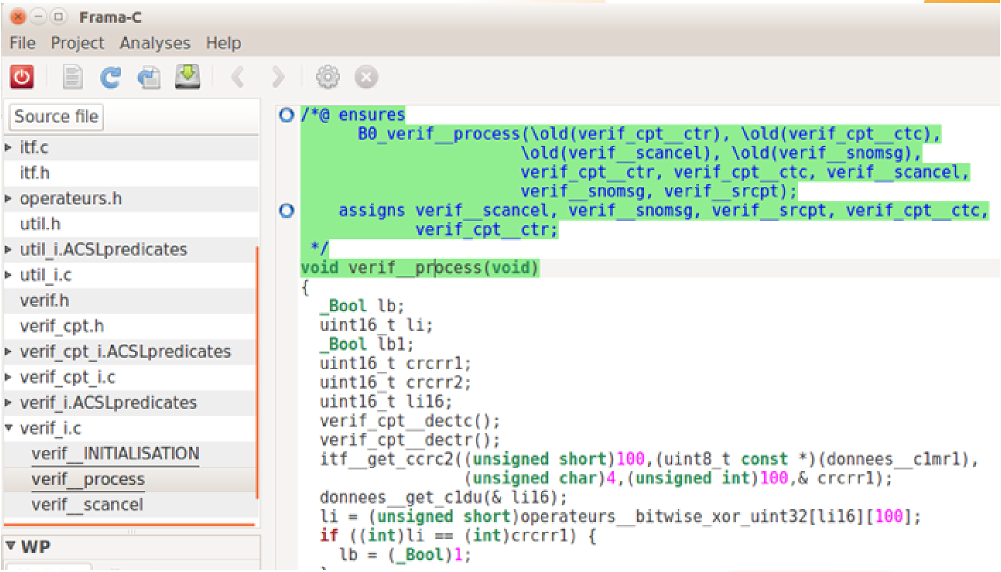
Fit for industry (planes, trains, IT, cryptocurrencies, smartcards, etc.)



Bottom-Up Formal Method

Frama-C *https://frama-c.com/*

- Assertion-based (Hoare logic)
- Source code decorated with pre and post conditions
- Works well for unit testing
- Nightmare for integration testing
- Used by Airbus and Boeing



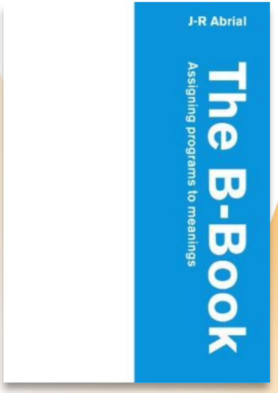
Polyspace

- Assertion-based
- Code coloured: OK, NOK, unknown, dead code.

Top-Down Formal Method

B METHOD *http://www.methode-b.com/*

- Invented by French mathematician (J. R. Abrial)
- Assigning programs to meanings
- Top-down approach
- Programs are proved to comply with their specification



ATELIER B *http://www.atelierb.eu/*

- Implement B Method
- First autonomous metro Paris L14 Meteor (1998)
- All metros in Paris to be automated (L1, L4, etc.)
- 30% of automatic metro worldwide
- Maintained & developed by CLEARSY

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

```
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;
```

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

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

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

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

Natural language requirement

B Specification

Behaviour + properties

B Implementation

Behaviour + properties

C generated code

Binary 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)

Cyclic software single-thread

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

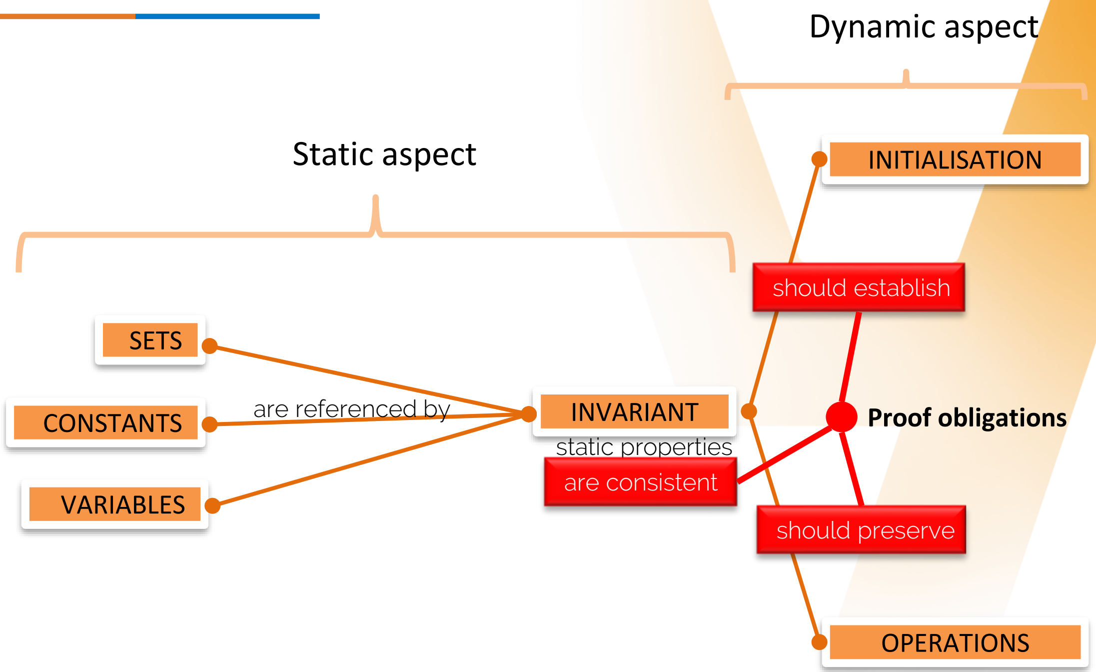
Binary code

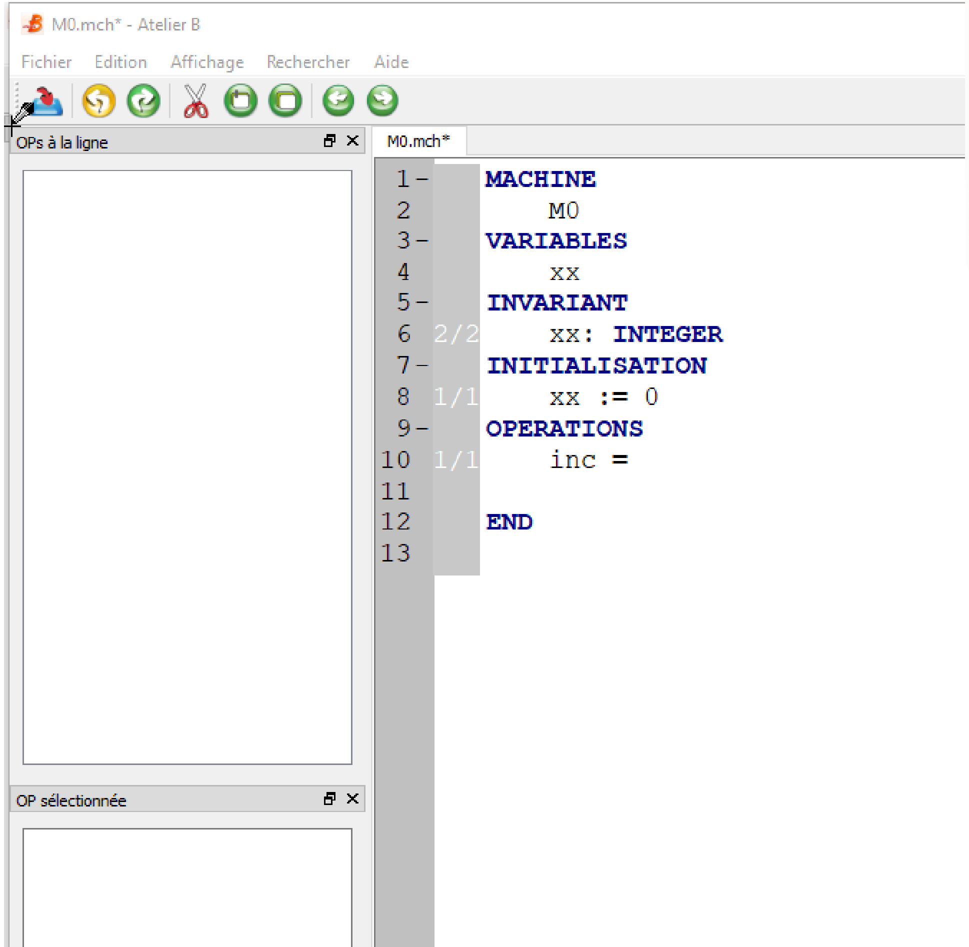
B Proof Obligations

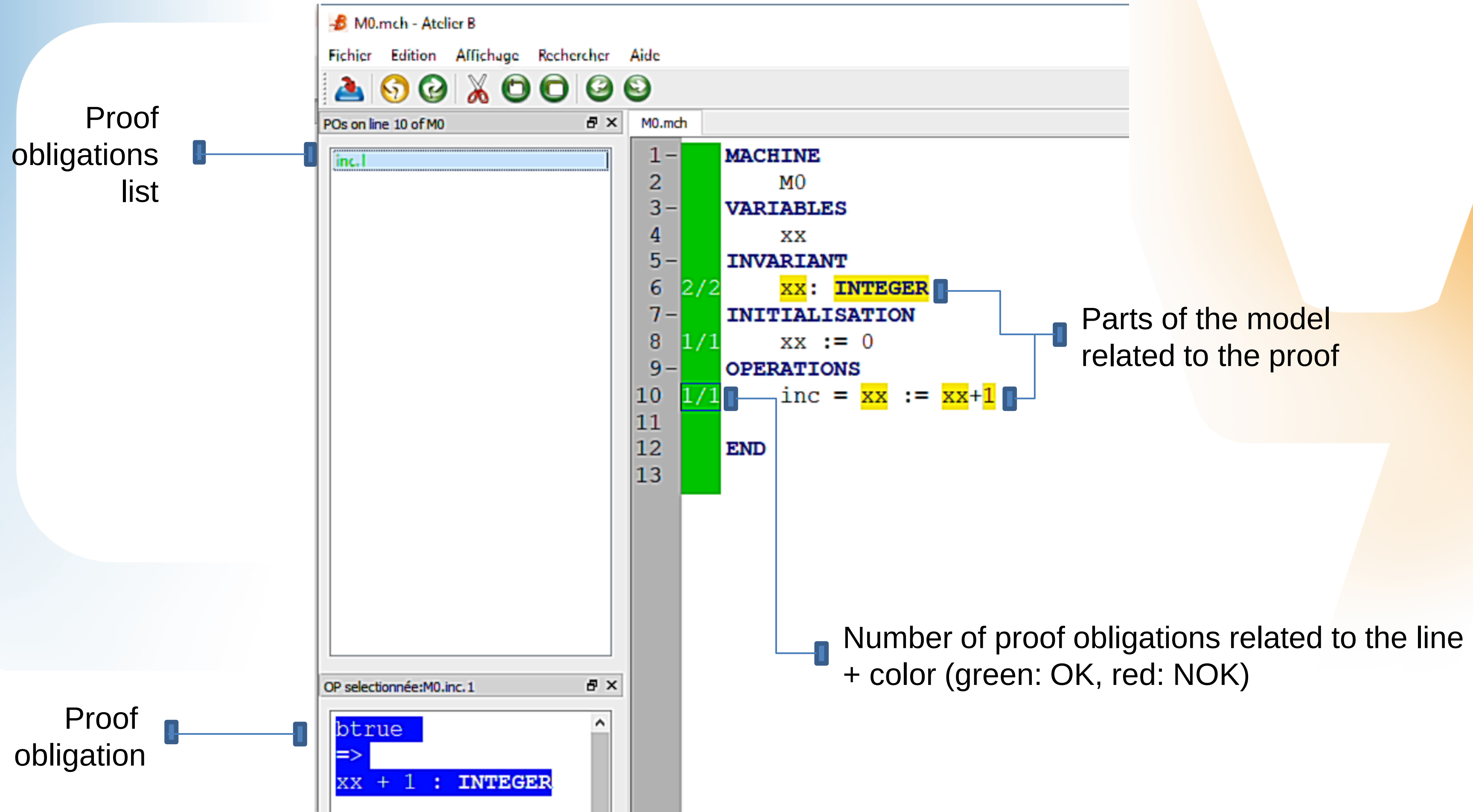
≡ Proof Obligations [POs]
Fully, automatically generated

- Functional
- Well-definedness
- Overflow (option)

≡ B Project Correct
When 100% POs are proved




M0.mch* - Atelier B
Fichier
Edition
Affichage
Rechercher
Aide
OPs à la ligne
OP sélectionnée
M0.mch*
MACHINE
M0
VARIABLES
xx
INVARIANT
xx: INTEGER
INITIALISATION
xx := 0
OPERATIONS
inc =
END


M0.mch - Atelier B
Fichier Edition Affichage Rechercher Aide
POs on line 10 of M0
inc.1
M0.mch
1 MACHINE
2 M0
3 VARIABLES
4 xx
5 INVARIANT
6 2/2 xx: INTEGER
7 INITIALISATION
8 1/1 xx := 0
9 OPERATIONS
10 1/1 inc = xx := xx+1
11
12 END
13
OP selectionnée:M0.inc.1
btrue
=>
xx + 1 : INTEGER
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

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
```

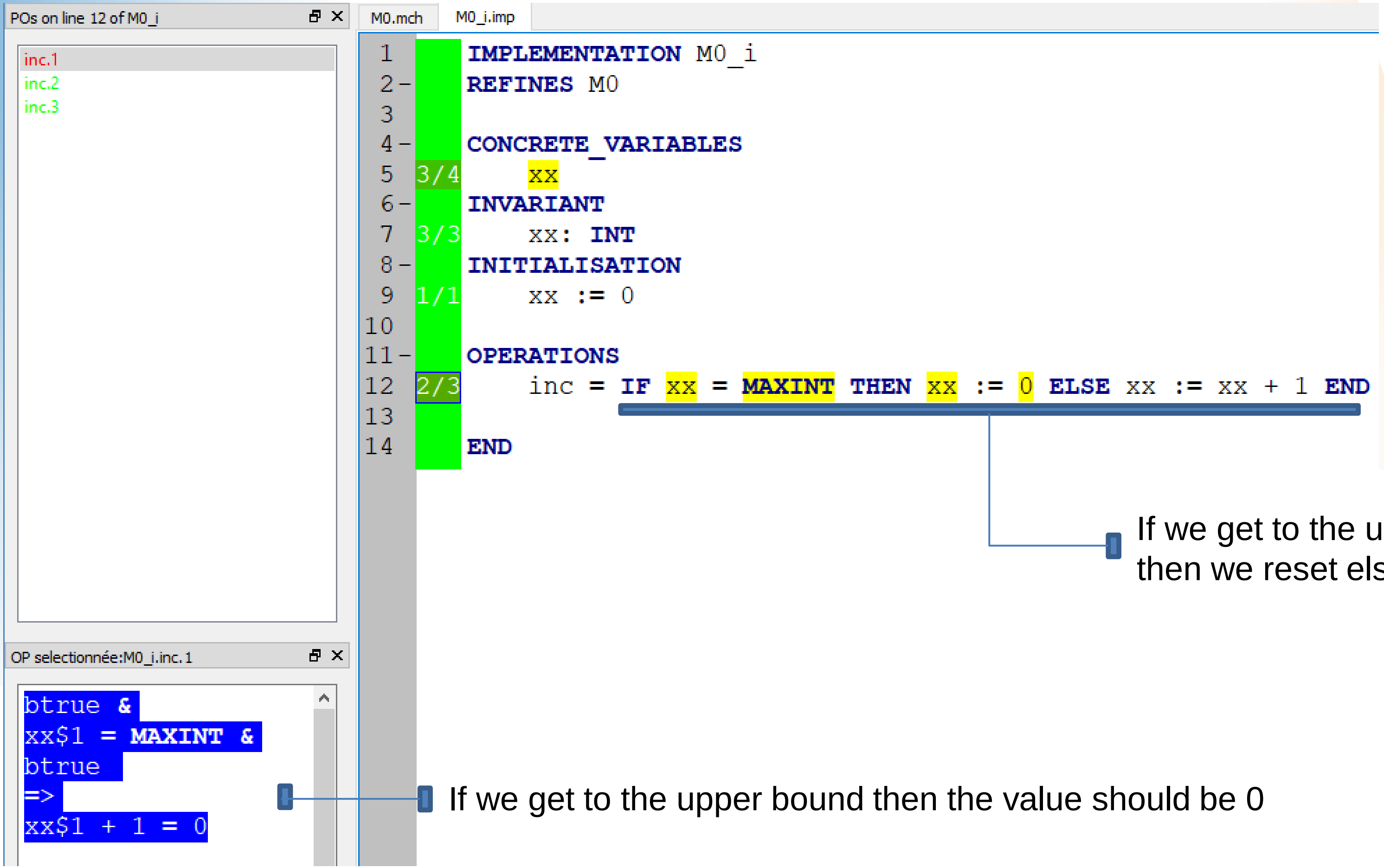
Implement its specification

Implemented variable

Implementable type

Endless increment

If we increment enough , we get out of type INT



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

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
```

Formal Methods and Railways

- Safety critical software development
  - Software formal model(specification and implementation)
  - Coherency: formal proof

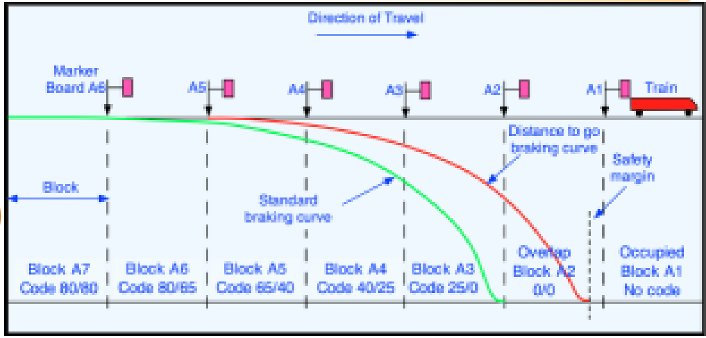
REFERENCES

[1] *Météor: A Successful Application of B in a Large Project*
FM 1999, Toulouse
Patrick Behm, Paul Benoit, Alain Faivre, Jean Marc Meynadier

[2] Using B as a High Level Programming Language in an Industrial Project: Roissy VAL
ZB 2005, Guildford
Arnaud Amelot, Frédéric Badeau

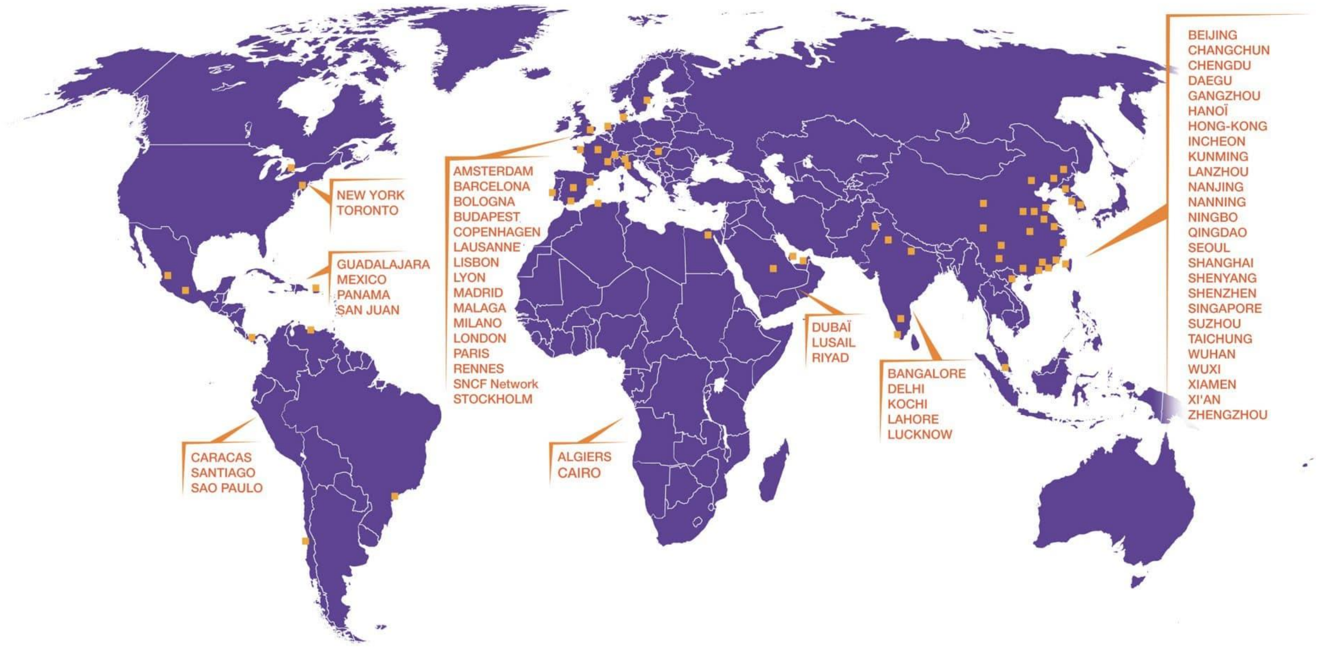
Formal Methods and Railways

- ▶ Driving is not safety related
  - ▷ No need of formal methods to drive a train
- ▶ Formal protections (B method)
  - ▷ Localization (graphs)
  - ▷ Kinetic energy control (integer)
  - ▷ Emergency braking (Boolean equations)



Braking curves

30% automatic metros worldwide



Formal Methods and Railways

▶ Data Formal Verification

- ▷ Data formal model (100k cells)
- ▷ Conformance with model-checking
- ▷ Used by most industry
- ▷ Able to handle 10 Mloc generated MACHINE

	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

REFERENCES

[1] *Formally Checking Large Data Sets in the Railways*
ICFEM 2012, Kyoto
Thierry Lecomte, Lilian Burdy, Michael Leuschel

[2] Formal Data Validation in the Railways
SSS'16, Brighton
Erwan Mottin, Thierry Lecomte

Formal Methods and Railways

- **Formal analysis of Systems**
  - Model of reasoning used for the design (rediscover the « why »)
  - Useful for « *legacy* » systems

REFERENCES

[1] *B-specification of Relay-based Railway Interlocking Systems Based on the Propositional Logic of the System State Evolution*
RSSR 2019, Lille
Dalay Israel de Almeida Pereira, David Deharbe, Matthieu Perin and Philippe Bon

[2] Safety Analysis of a CBTC System: A Rigorous Approach with Event-B
RSSR 2017, Pistoia
Mathieu Comptier, David Deharbe, Julien Molinero Perez, Denis Sabatier

Formal Methods and Railways

- ▶ Safety Computer programmed with B
  - ▷ IDE and a hardware platform that natively integrates safety principles
  - ▷ For developing cyclic applications without underlying OS

REFERENCES

[1] *The CLEARSY Safety Platform: 5 Years of Research, Development and Deployment SBMF 2019, São Paulo*

Thierry Lecomte, David Deharbe, Paulin Fournier, Marcel Oliveira

Formal Methods and Railways