CLEARSY
Safety Solutions Designer

FMICS
SEP2024

B+

How to Model System Properties in a Software Formal Model

« Presentation of a more integrated use of the B method without changing the language and the tool.»

Thierry Lecomte
R&D Director

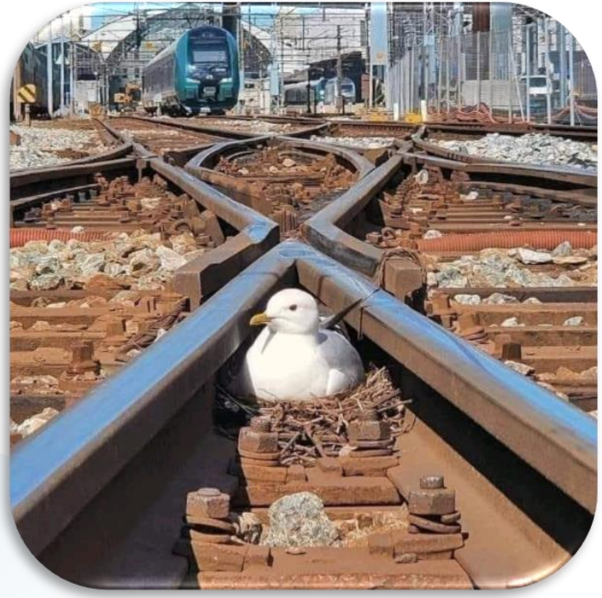
THIERRY.LECOMTE@CLEARSY.COM

Art mostly generated with *ChatGPT or similar*

Common Thread

How the last 30 years
changed our view
on safety critical
software development
in the railways

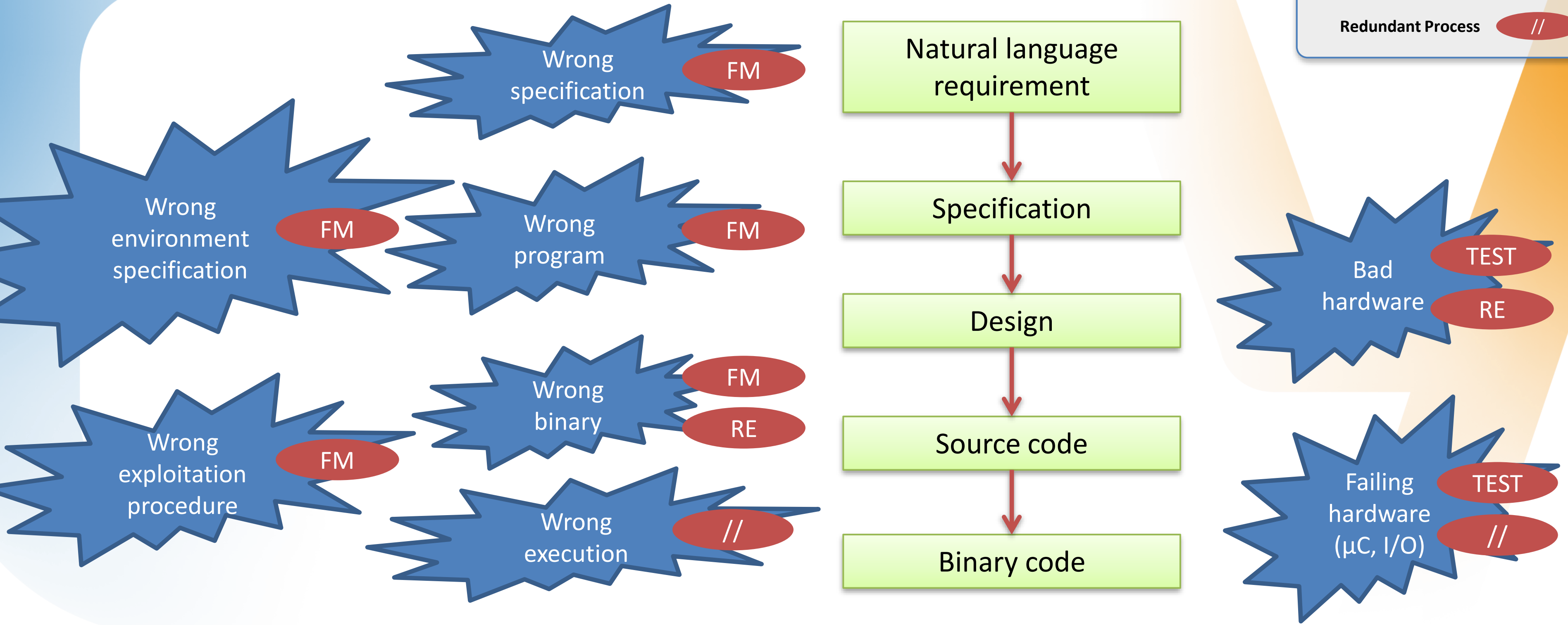
SAFETY



- Failing systems
- Safety critical
- Standards
- Safety in practice

Failing Software-Based Systems

Formal Methods — FM
Testing — TEST
Reverse Engineering — RE
Redundant Process — //



Safety @ Railways

SAFETY INTEGRITY LEVELS

- SIL3 : 10^{-7}/h
- SIL4 : 10^{-9}/h

CATASTROPHIC FAILURES

CERTIFICATION

- NL safety demonstration
- Convince responsible human expert
- Formal methods **highly recommended**

STRONG STANDARDS

EN5012{6, 8, 9}

SYSTEMATIC FAILURES

- Specification
- Design
- Implementation
- Environment
- Exploitation

RANDOM FAILURES

- Execution machine
- Entropic hardware

Safety @ Railways

CERTIFICATION

- NL safety demonstration
- Convince responsible human expert
- Formal methods **highly recommended**

APPLIED TO

SYSTEMATIC FAILURES

- Specification
- Design
- Implementation
- Environment
- Exploitation

Safety @ Railways @ CLEARSY

FORMAL SOFTWARE DEVELOPMENT
FORMAL DATA VALIDATION
FORMAL SAFETY PROOF

CERTIFICATION
- NL safety demonstration
- Convince responsible human expert
- Formal methods **highly recommended**

SYSTEMATIC FAILURES
- Specification
- Design
- Implementation
- Environment
- Exploitation

APPLIED TO

SAFETY COMPUTER

RANDOM FAILURES
- Execution machine
- Entropic hardware

Formal Methods to Handle Failing Systems

Event-B
for safety
reasoning

B
for C&C
non-threaded
safety software

Natural language
requirement

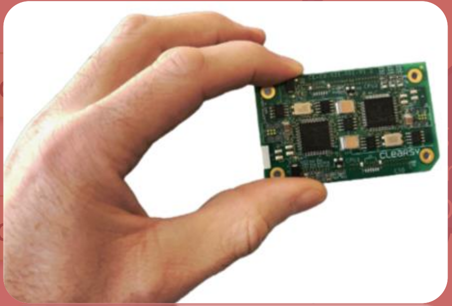
Specification

Design

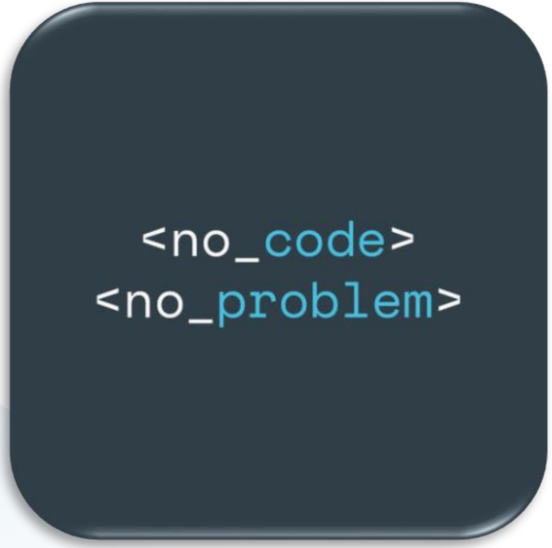
Bad
hardware

B
for data
validation

CLEARSY SAFETY
PLATFORM



B for C&C
not Threaded Safety Software



- Failing systems
- Safety critical
- Standards
- Implementation

Formal Software Development

Safety critical software
formally specified & proved

No unit test
Most integration test avoided

SET THEORY
FIRST ORDER LOGIC
INTEGER
BOOLEAN
GRAPHS

IDE DEVELOPED DURING 25+ YEARS
FREELY AVAILABLE
CERTIFIED EN50128 T2 IN 2024

https://www.atelierb.eu/en/

References:

- *The B-book - Assigning Programs to Meanings*, Cambridge Press, 2001
- *The First Twenty-Five Years of Industrial Use of the B-Method*, FMICS, 2020

« 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;
```

```c
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;
```

Cyclic software single-thread

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

B Specification

Proof (coherence)

Proof (refinement)

B Implementation

Proof (coherence)

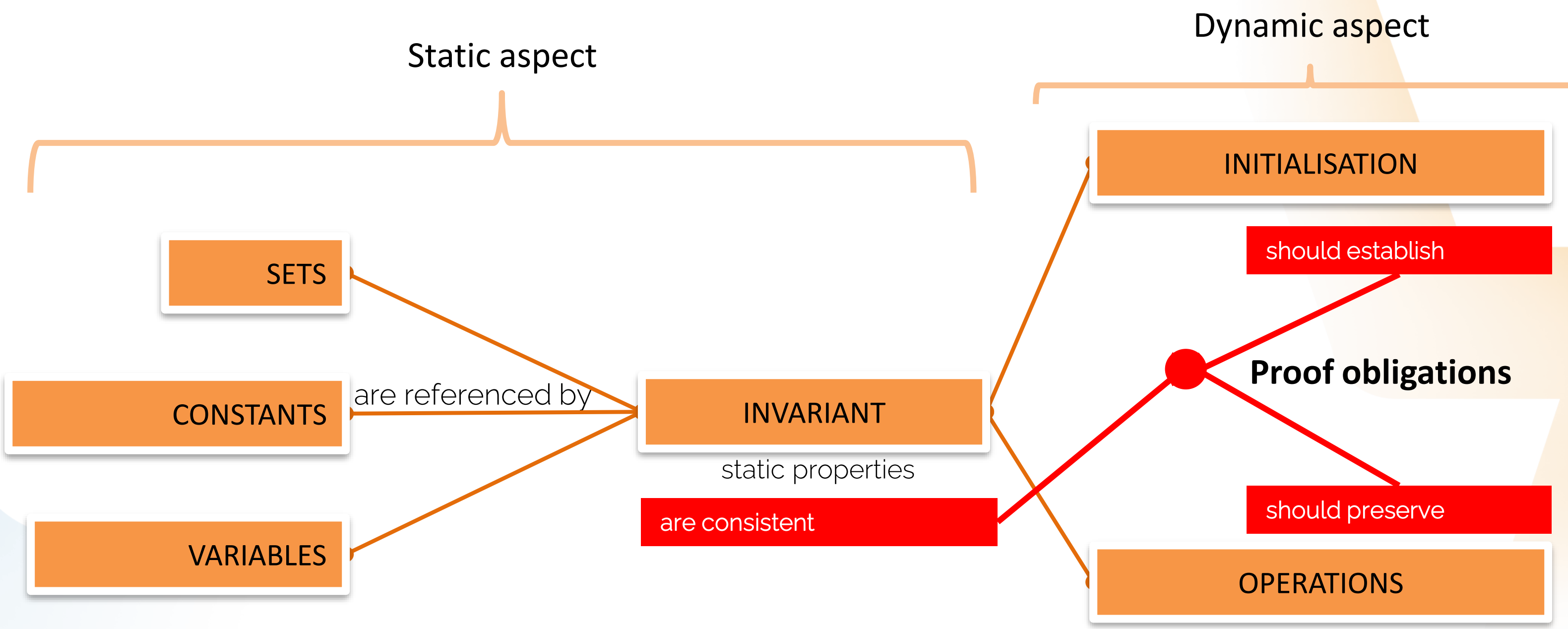
```
void M0__activation_sequence(void)
{
    CTX__SEQUENCES sequ;

    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

Proof Obligations from B Models



B Code Generation

≡ The software code is generated from the model

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

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
```

```
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;
    }
}
```

Software Formal Development

▶ Atelier B Technology [C, C++, Prolog-like]

▷ Automatic refinement based on Siemens inference engine

2006
- Integrated into Atelier B

2006-2024
- Applications up to 500 kloc for train control (NY metro, CdG shuttle) and software engineering (interpreter, compiler)

```
RULE scalar_ini0
REFINES
    @a :: @b
WHEN
    ENUM(@b) &
    @c : @b
IMPLEMENTATION
    @a := @c
END;
```

▷ Code generators:

- Ada (proprietary)(product specific)

2001-2024
- C (generic, 32-bit MCU)(*generation of Frama-C ACSL*)

2023
- Rust
- RIP: Instruction List, Ladder, LLVM, VHDL

References:

- *Applying a Formal Method in Industry: A 15-Year Trajectory*, FMICS, 2009
- *On B and Event-B: Principles, Success and Challenges*, ABZ, 2018
- *B2rust*, https://github.com/CLEARSY/b2rust

Software Formal Development

▶ Atelier B Technology [C, C++, Prolog-like]

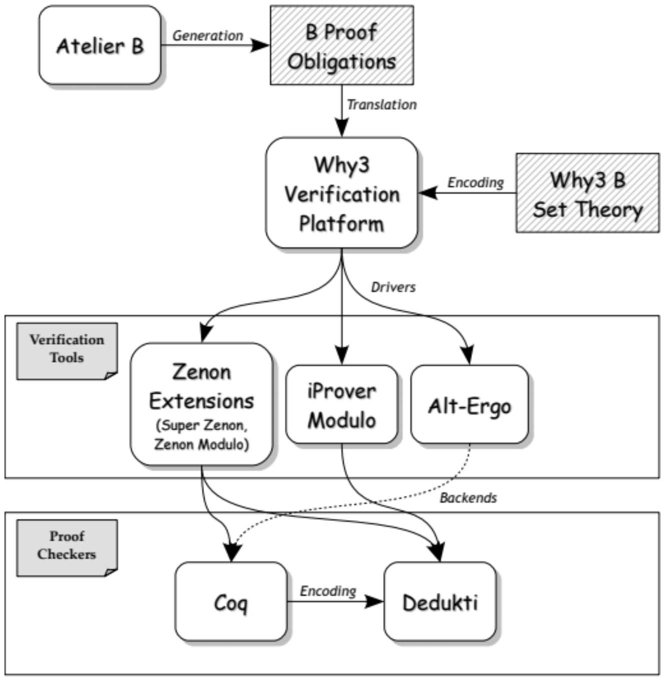
▷ Specific proof tools developed

1998
- Main prover as an inference engine with using 2600 rules
- Predicate prover to demonstrate 80% of the rules
- Main prover stuck in 1998 (interactive demos could not survive prover improvement)

1998-2024
- Extension of interactive proof language, GUI

2008-2027
- Connexion with third party provers (Alt-Ergo, CVC3, iProver, Vampire, Z3, Zenon)
- 500k proof obligations publicly available for benchmark

2022-2024
- Connexion with Generative AI for proof script generation



The BWare Platform for the Automated Verification of B Proof Obligations

References:
- ANR Projects Bware, BLASST, ICSPA - ECSEL Project AIDOaRT
- *Atelier B oPEn ResOurces*, https://github.com/CLEARSY/apero

Software Formal Development

► Atelier B Dissemination

- ▷ Continuous low frequency professional training
- ▷ Internal training for volunteers and FM profiles
- ▷ Continuous academic courses with CLEARSY Safety Platform
- ▷ Downloads:
  - **4500** / teaching semester,
  - 1300 Atelier B Prover plug-in for Rodin platform

References:

- *Programming Handbook*, https://github.com/CLEARSY/CSSP-Programming-Handbook

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

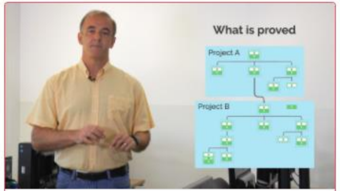
This video presents the structure of the course, provides an overview of the different kinds of formal methods and specification styles, and tells us some myths on formal methods

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

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

UFRN
UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

IMD
INSTITUTO METRÓPOLE DIGITAL

Software Formal Development

1998

Paris L14 Automatic Train Protection (ATP)
Emergency braking in case of danger (86 kloc B, 110 kloc Ada)

2000-2024

Used by ~30% radio-based control metro worldwide
CDGVAL shuttle (500 kloc / automatic refinement)

2006-2024

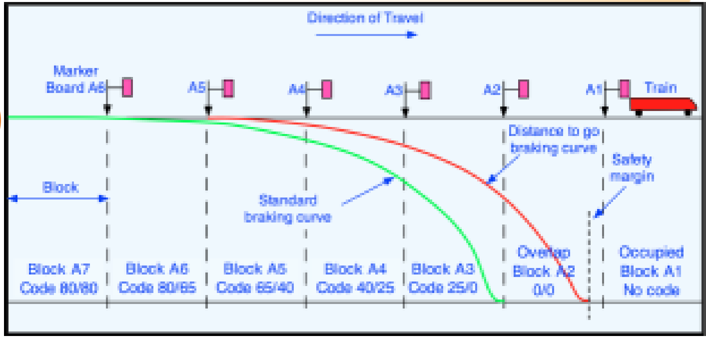
Used for Paris L1, L4, L13, L14 (Olympics)

2024-2030

To be used for Paris L15, L16, L17, L18

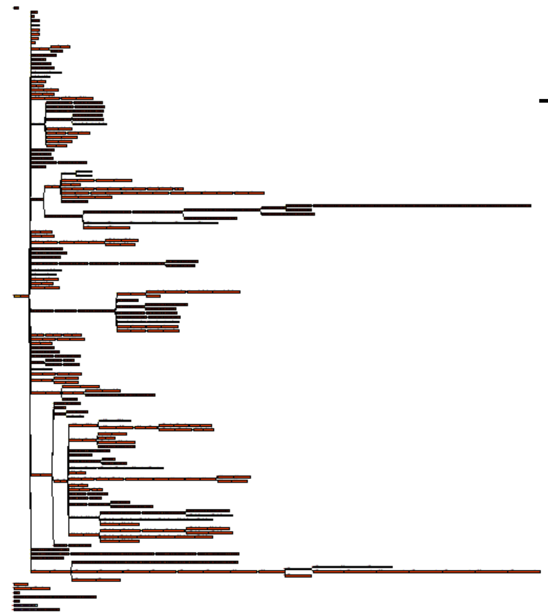
B: what for ?

- Driving is not safety related
  - No need of formal methods to drive a train
- Safeguard
  - Localization (graphs)
  - Kinetic energy control (integer)
  - Emergency braking (Boolean equations)



Braking curves

Formal Methods and Railways: metrics



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.

The specification is not fully contained in the toplevel component

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)

The specification is not « avoid collision » but « brake if not authorized to go forward »

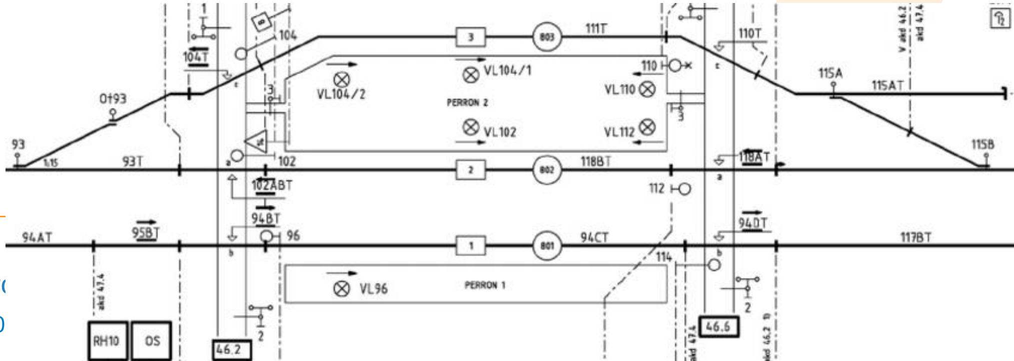
Towards the limits …

« 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]

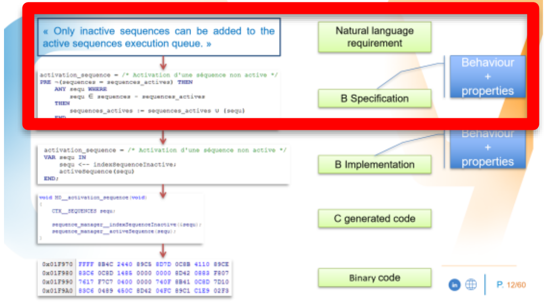


Towards the limits

```
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 )( over_track ( ii ) ) ) & ( ii = 1 => p_X2MRes <= eoa_res ) & ( ( prj2 ( t_block , t_direction ) ( over_track ( ii )
) ) = c_up => eoa_res : ran ( sgd_blockUpRestrictionSeq ( p_EOABlock ) ) & res_after_eoa : ran ( sgd_blockUpRestrictionSeq ( p_EOABlock ) ) & sgd_restrictionAbs ( eoa_res ) <=
p_EOAAbs & p_EOAAbs < sgd_restrictionAbs ( res_after_eoa ) & ! ri . ( ri : ran ( sgd_blockUpRestrictionSeq ( p_EOABlock ) ) => ri <= eoa_res or res_after_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 ) )
```

REX & Summary

- Well-oiled process in the railways
  - No programming error
  - Deliverables (models, proofs, code, V&V) accepted for certification
  - No fatality since 90s
- B mainly used for programming
  - Safety is distributed over several systems
  - Low-level Customer Specification Document
  - B model verification activity (quite) unsatisfactory



CLEARSY Safety Platform

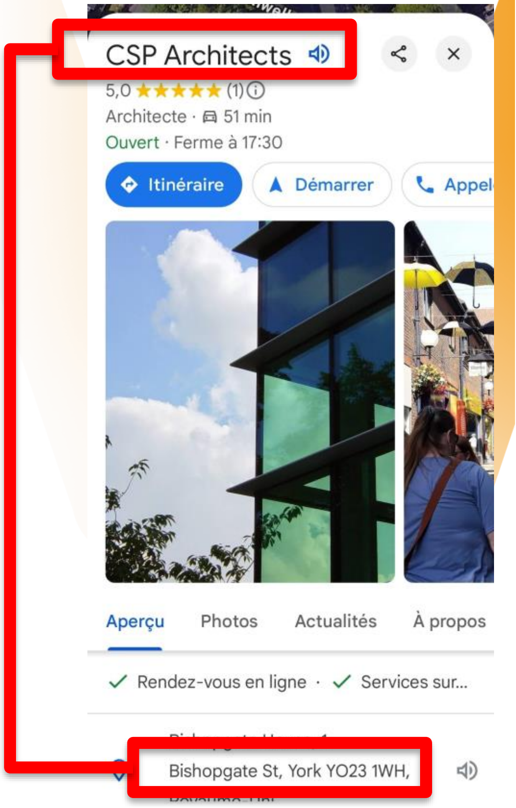
<no_code>
<no_problem>

- Safe computing
- Platform architecture
- Applications

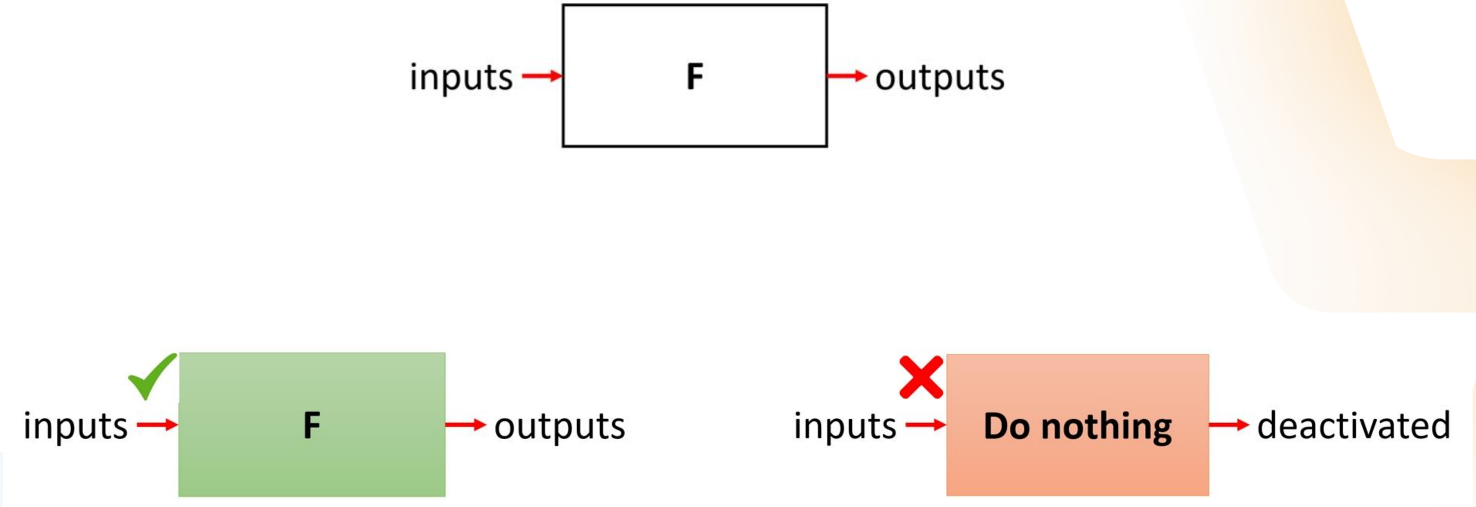
CSP or CSSP ?

- CLEARSY Safety Platorm abbreviated as CSP when there is no risk of confusion
- CSSP otherwise

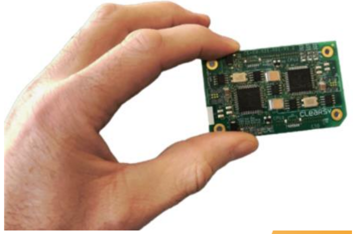
York
Best place to « CSP »



Safe Computing



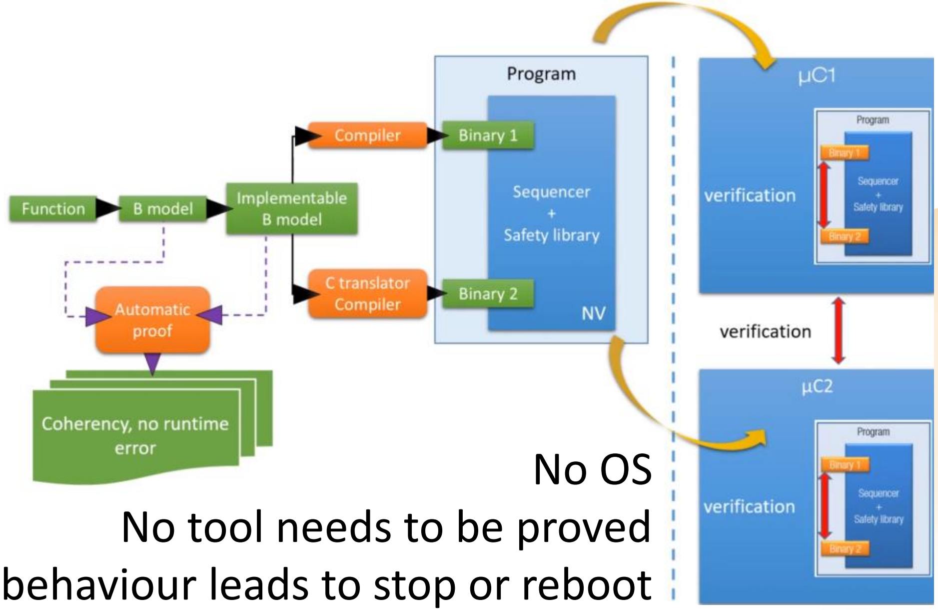
CLEARSY Safety Platform



Safety computer able to handle random failures
Programmed with B for systematic failures

4oo4 Software
2oo2 Hardware

SET THEORY
FIRST ORDER LOGIC
INTEGER
BOOLEAN
GRAPHS

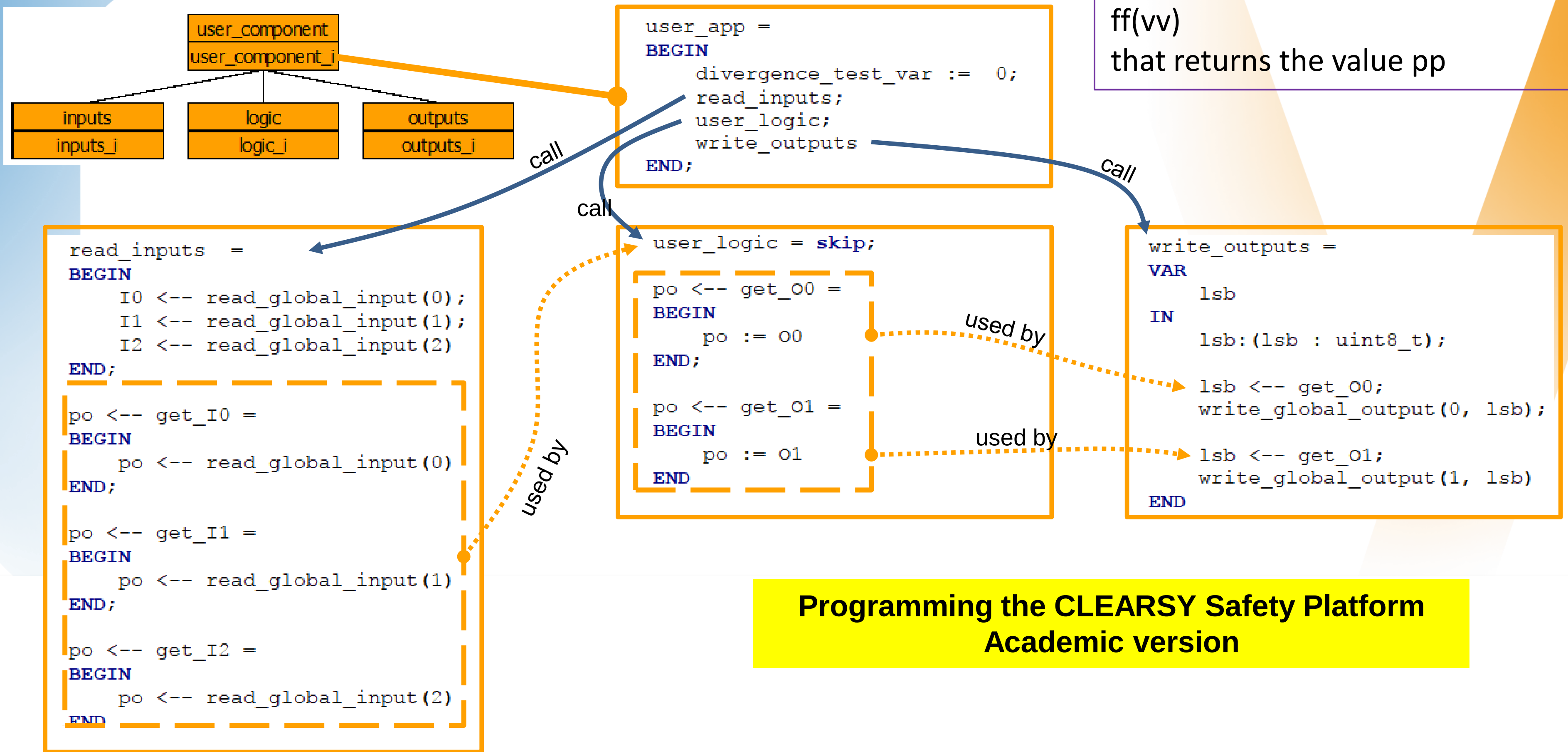


No OS
No tool needs to be proved
Divergent behaviour leads to stop or reboot

Syntax:

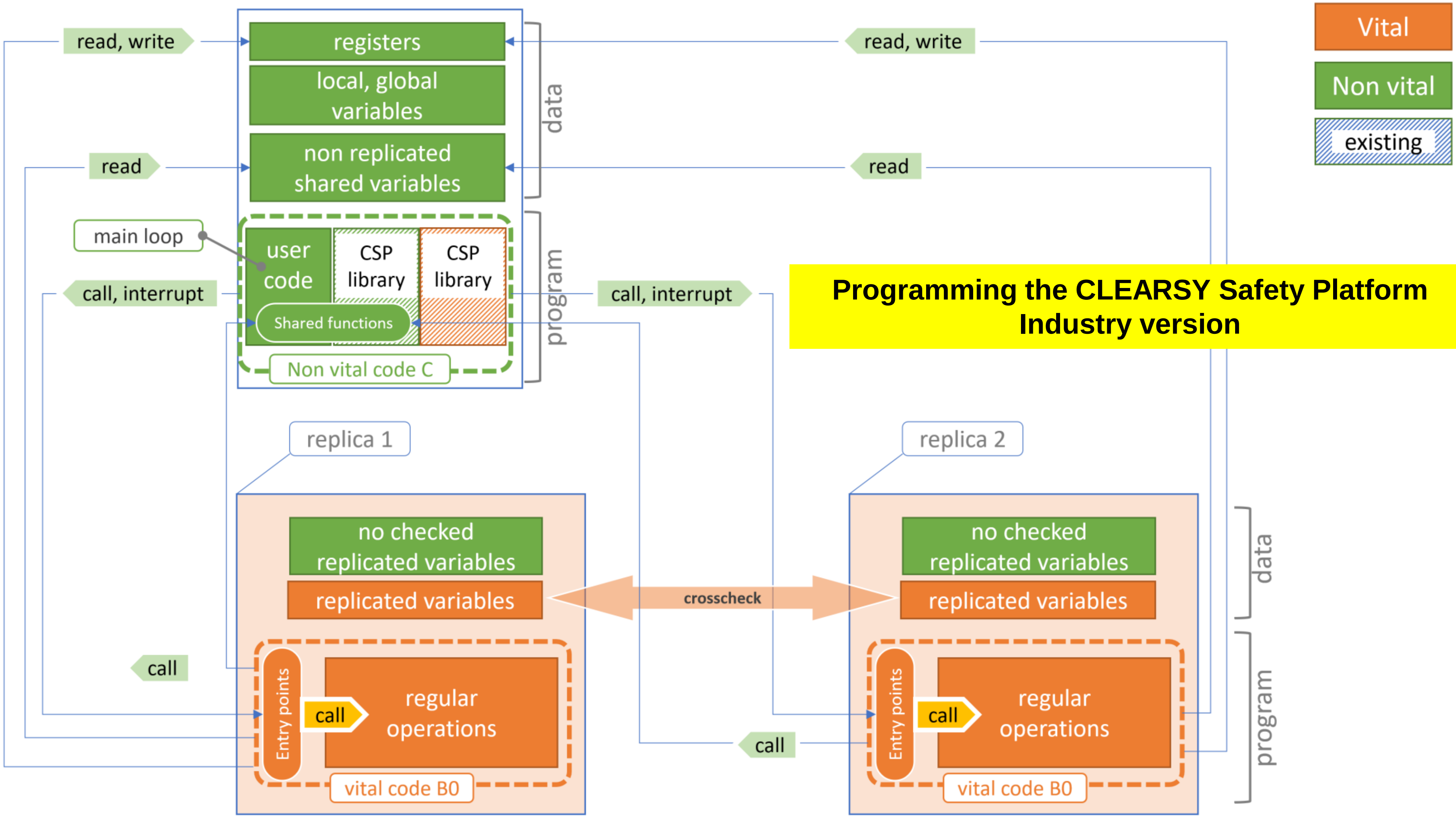
pp <-- ff(vv)

represents a call to operation ff(vv) that returns the value pp

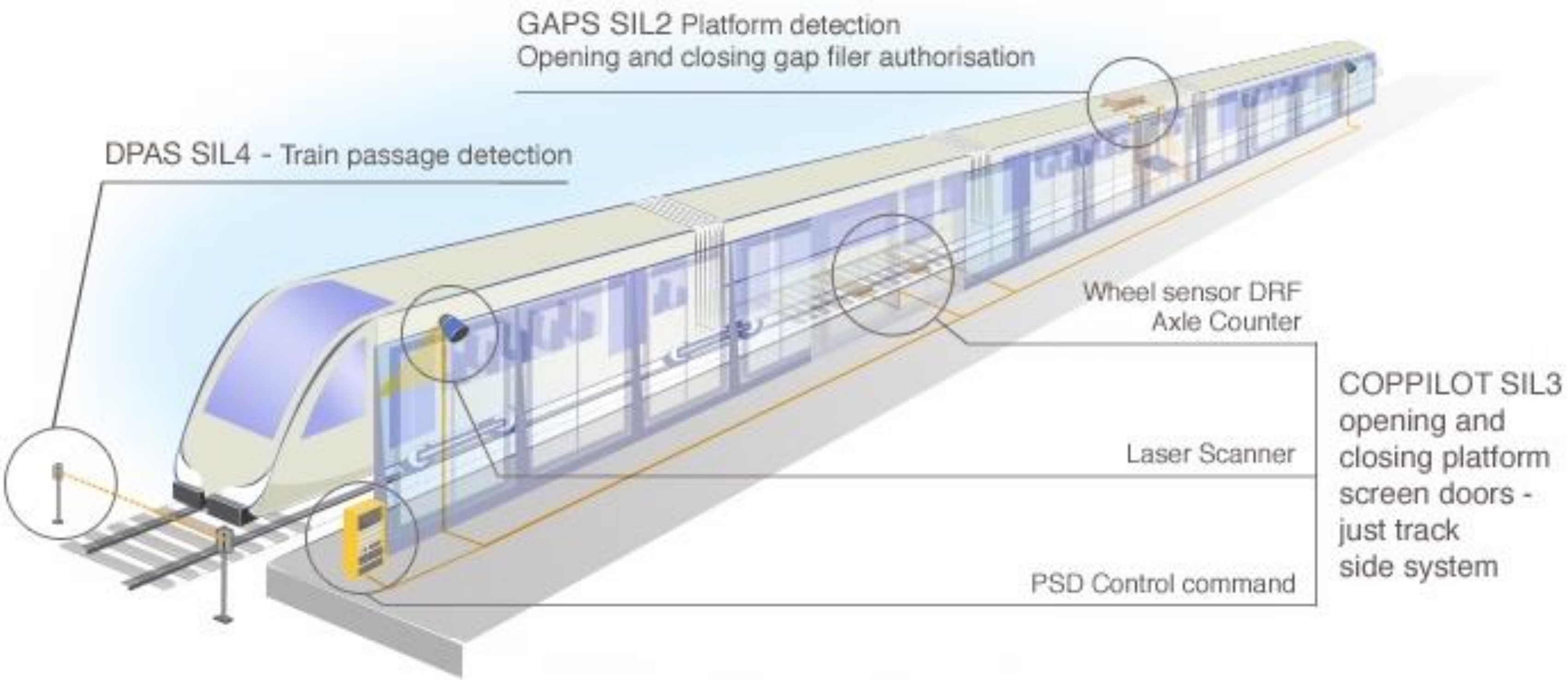


Programming the CLEARSY Safety Platform Academic version

Programming the CLEARSY Safety Platform Industry version



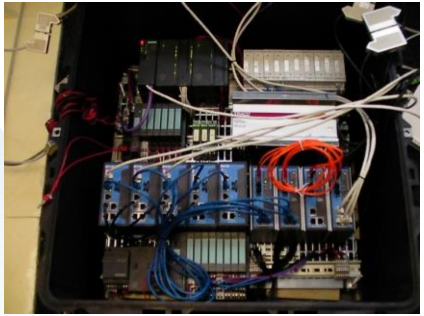
Platform screen doors: a safer system



Platform screen doors: a safer system

≡ System to install to prepare driverless operation

- No direct communication with the train: train arrival and door opening to be detected with diverse sensors
- SIL4: one failure every 10 000 years
- 99,95% reliability: one train max missed per year
- To be developed from scratch in 6 months

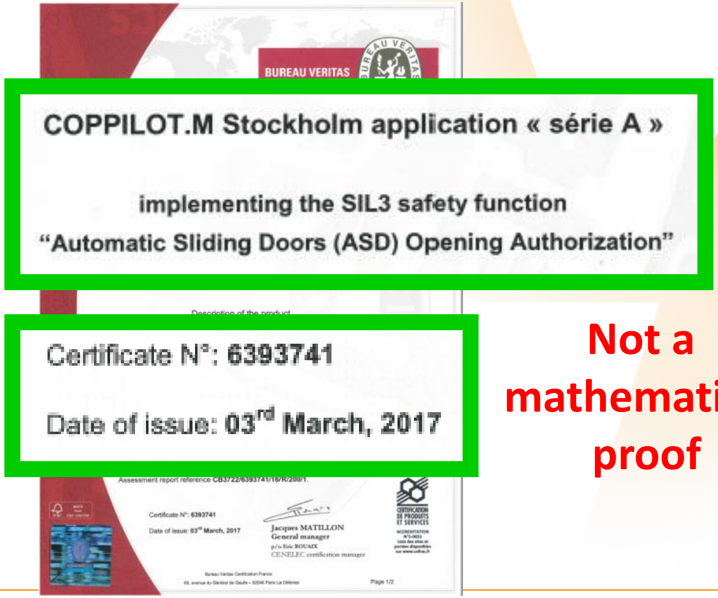


Platform screen doors: a safer system

≡ Installation on site

Platform screen doors controler installed in Stockholm (Citybanan)

≡ Certification



Not a mathematical proof

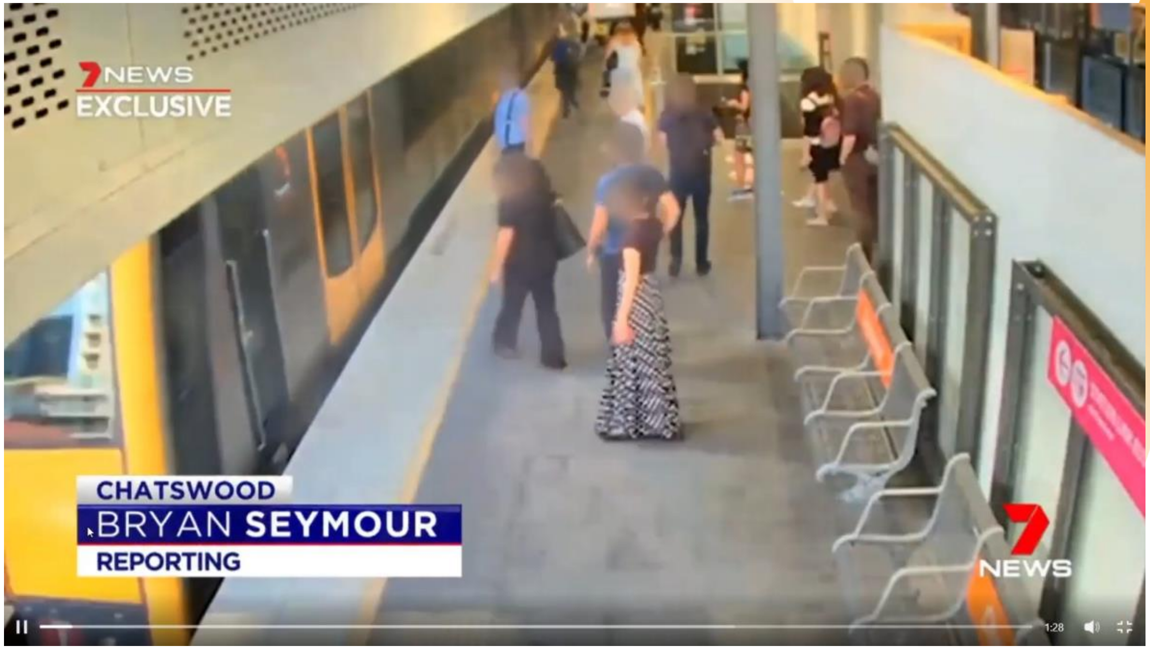
It still happens !

Paris : une femme meurt, happée par son manteau bloqué dans les portes du métro

Par M.D.

Publié le 23 avril 2023 à 14h00, mis à jour le 23 avril 2023 à 16h20

A woman dies after her coat gets caught in the metro doors



CLEARSY Safety Platform

2006-2019

Building blocks developed for platform screen doors (PSD) controllers
French R&D project for academic-version safety computer

2020

Industry-ready generic safety computer developed

2021

Platform certified EN50129 SIL4

2023-2024

Deployed in Brisbane to control PSD

2024

Deployed for ground and underwater autonomous mobility
French R&D project to add cybersecurity

REX & Summary

- Platform & B modelling accepted for certification
- Programming is still low level
- A formal method and a safe computer are not enough
  - Environment
  - Human factor
  - Modifying a system creates new risks

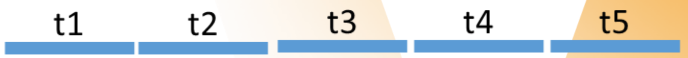
Formal Data Validation

- Mathematical Language
- Process
- Achievements
- Usability Proof

Properties with the B Mathematical Language

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

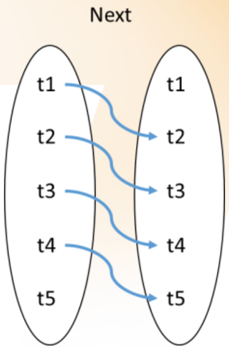
Let the set TrackCircuit = {t1, t2, t3, t4, t5}



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}



Let the function KpAbs : TrackCircuit $\rightarrow$ N

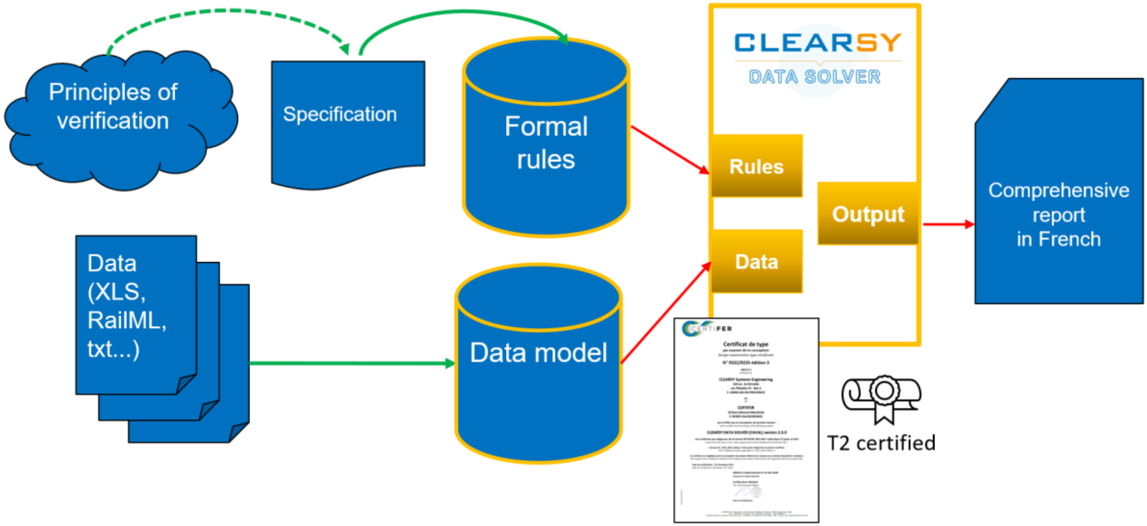
$\forall$x.(x$\in$ TrackCircuit $\wedge$ x $\in$ dom(Next) $\Rightarrow$ KpAbs (Next(x)) > KpAbs(x))

Formal Data Validation

Safety critical constant data
formally specified & model-checked

100k data chunk, up to 2k rules
Human errors avoided

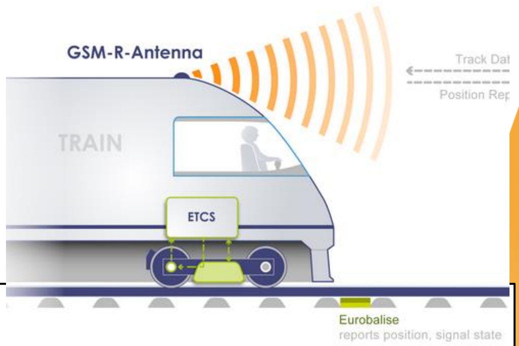
SET THEORY
FIRST ORDER LOGIC
INTEGER
BOOLEAN
GRAPHS



References:

- *Formally Checking Large Data Sets in the Railways*, ICFEM, 2012
- *ProB*, https://prob.hhu.de/

Interaction
Reasoning / Validation



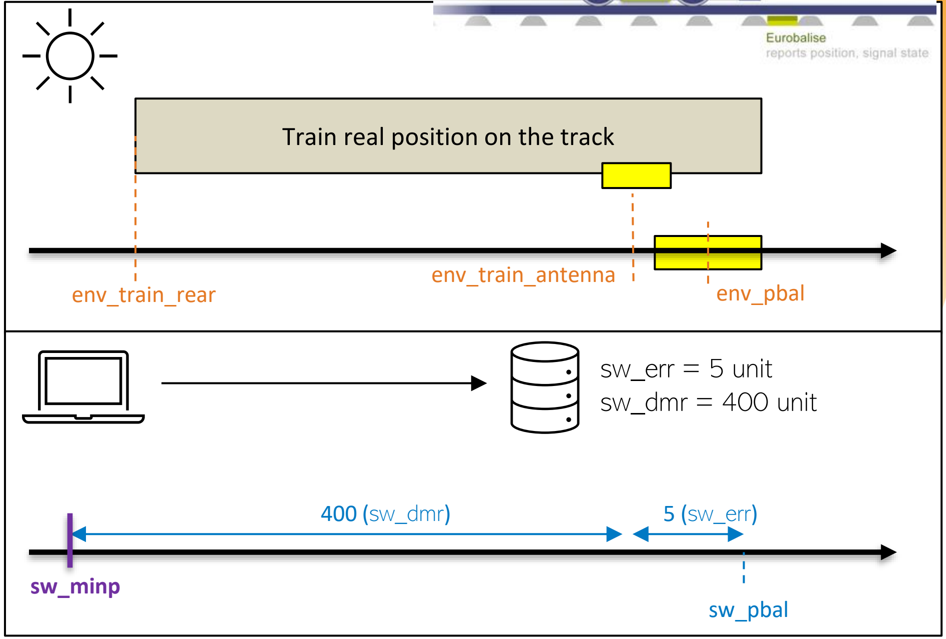
- Formalising the safety property:

  sw_minp ≤ env_train_rear

- Formalisation of hypotheses linking the environment and the software:

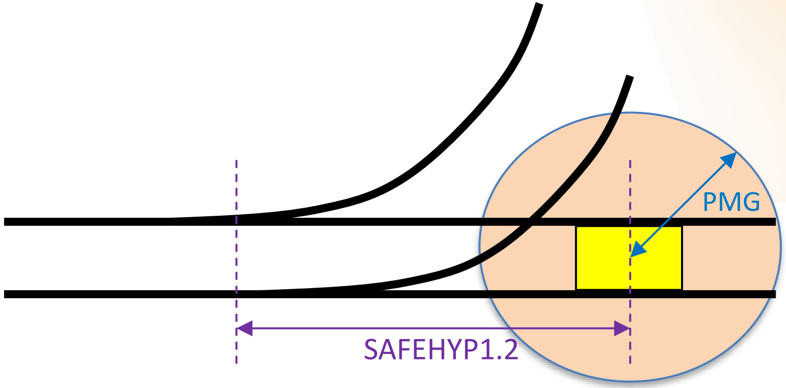
  H1) sw_pbal – sw_err ≤ **env_pbal** ≤ sw_pbal + sw_err
  H2) env_train_antenna – env_train_rear ≤ sw_dmr

- Missing concept: maximal guaranteed range



Link with the Formal Data Validation

- SAFEHYP1_2 : Balises must not be too close to switch toes on its common incident edge
  - **Allocation** : Formal validation of parameters



'Too close' can be calculated: as a function of the Maximum Guaranteed Range (MGR) and the radius of curvature.

Data Valid, step 1: formalization

SRAC2 : Balises must not be too close to switch toes on its common incident edge

formalization

Processing

- For each *SWITCH* of `switch`,
  - let *TOE_POSITION* be the position of the switch,
    - {1} verify that *TOE_POSITION* is a valid position
    - if *TOE_POSITION* is a valid position,
      - let *MIN_DIST_ZONE* be the zone covering a distance of `systemParameter::MIN_DISTANCE_TO_SWITCH_TOE` starting from *TOE_POSITION* and in direction of the COMMON edge of the switch,
        - for each element *DETECTIONPOINT* of `detectionPoint`,
          - {2} verify that *DETECTIONPOINT* is not located inside *MIN_DIST_ZONE*

Messages

- {1} "WARNING : For switch '%`switch::NAME`(*SWITCH*)%', we can't find a valid toe position from its node '%`switch::node`(*SWITCH*)%'."
- {2} "For switch '%`switch::NAME`(*SWITCH*)%', detection point '%`detectionPoint::NAME`(*DETECTIONPOINT*)%' is located at less than '%`systemParameter::XX_AC_MIN_DISTANCE_TO_SWITCH_TOE`%' from its toe '%*TOE_POSITION*%'."

Data Valid, step 2: formal validation rule design

Processing

- For each *SWITCH* of `switch`,
  - let *TOE_POSITION* be the position of the switch,
    - {1} verify that *TOE_POSITION* is a valid position
    - if *TOE_POSITION* is a valid position,
      - let *MIN_DIST_ZONE* be the zone covering a distance of `systemParameter::MIN_DISTANCE_TO_SWITCH_TOE` starting from *TOE_POSITION* and in direction of the COMMON edge of the switch,
        - for each element *DETECTIONPOINT* of `detectionPoint`,
          - {2} verify that *DETECTIONPOINT* is not located inside *MIN_DIST_ZONE*

Messages

- {1} "WARNING : For switch '%`switch::NAME`(*SWITCH*)%', we can't find a valid toe position from its node '%`switch::node`(*SWITCH*)%'."
- {2} "For switch '%`switch::NAME`(*SWITCH*)%', detection point '%`detectionPoint::NAME`(*DETECTIONPOINT*)%' is located at less than '%`systemParameter::XX_AC_MIN_DISTANCE_TO_SWITCH_TOE`%' from its toe '%*TOE_POSITION*%'."

Formal Model Design

```
FOR
    SWITCH, TOE_POSITIONS
WHERE
    SWITCH : acc::switch
    & TOE_POSITIONS = ran(%(node, refEdge, offset).(
        node = acc::switch__node(SWITCH)
        & refEdge : ran(acc::node__EDGE(node))
        & refEdge'incidence = ENU__INCIDENCE__COMMON
        & ( (refEdge'edge : dom(acc::edge__nodeBegin |> {node}) & offset = 0)
            or
            (refEdge'edge : dom(acc::edge__nodeEnd |> {node}) & offset = acc::edge__LENGTH(refEdge'edge) -1)        )
        |
        rec(edge : refEdge'edge, offset : offset)
    ))
THEN
    VERIFY
        card(TOE_POSITIONS) > 0
    MESSAGE
        "For switch '%1', we can't find a valid toe position from its node '%2'."
        ARG acc::switch__NAME(SWITCH) TYPE STRING
        ARG acc::switch__node(SWITCH) TYPE INTEGER
    DATA_VERIFIED
        DATA acc::switch__node INDEX SWITCH
    SAFETY_LEVEL
        caval::WARNING
    ENDVERIFY
    SELECT
        card(TOE_POSITIONS) > 0
    THEN
        FOR
            TOE_POSITION, SYSPARAM, MIN_DIST_ZONE,  DETECTIONPOINT
        WHERE
            TOE_POSITION : TOE_POSITIONS
            & SYSPARAM : dom(acc::systemParameter__XX_AC_MIN_DISTANCE_TO_SWITCH_TOE)
            & MIN_DIST_ZONE =
                if TOE_POSITION'offset = 0
                then
                    ic::multi_zone_union(ic::multi_DIRECTED_ZONE__to__multi_ZONE(ic::zones_distance_from_point(TOE_POSITION, ENU__UPDOWN__UP, acc::systemParameter__XX_AC_MIN_DISTANCE_TO_SWITCH_TOE(SYSPARAM))))
                else
                    ic::multi_zone_union(ic::multi_DIRECTED_ZONE__to__multi_ZONE(ic::zones_distance_from_point(TOE_POSITION, ENU__UPDOWN__DOWN, acc::systemParameter__XX_AC_MIN_DISTANCE_TO_SWITCH_TOE(SYSPARAM))))
                end
            & DETECTIONPOINT : acc::detectionPoint
        THEN
            VERIFY
                ic::is_position_in_zone(acc::detectionPoint__POSITION(DETECTIONPOINT), MIN_DIST_ZONE) = FALSE
            MESSAGE
                "For switch '%1', detection point '%2' is located at less than '%3' from its toe '%4'."
                ARG acc::switch__NAME(SWITCH) TYPE STRING
                ARG acc::detectionPoint__NAME(DETECTIONPOINT) TYPE STRING
                ARG ic::DTY__LENGTH__tostring(acc::systemParameter__XX_AC_MIN_DISTANCE_TO_SWITCH_TOE(SYSPARAM)) TYPE STRING
                ARG ic::DTY__POSITION__tostring(TOE_POSITION) TYPE STRING
            DATA_VERIFIED
                DATA acc::detectionPoint__POSITION INDEX DETECTIONPOINT
            SAFETY_LEVEL
               caval::SIL4
            ENDVERIFY
        ENDFOR
    ENDSELECT
ENDFOR
```

Achievements

2003

First tool to verify embedded topology data
For Certification

2012

First tool integrated into CBTC metro dev process

2018

First application to ERTMS (beacons)

2024

Core tool certified 50128 T2
Applied by major train manufacturers and metros
Call for tenders requiring formal data validation

Formal Data Validation: the proof !

- TGV overspeed over a switch
  - 170 km/h instead of 100 km/h in La Milesse (France)
  - due to errors not detected during **human** data validation (2019)

- BEA-TT supports FM

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

BEA-TT
Bureau d'enquêtes sur les accidents de transport terrestre

"Given the difficulty of controlling the growing quantity of parameter data, the use of validation algorithms is essential. The use of innovative formal methods, based on advanced mathematical concepts, is one answer."

References:
- https://www.bea-tt.developpement-durable.gouv.fr/rapport-d-enquete-sur-la-survitesse-d-un-tgv-le-22-a1077.html

Towards the limits again

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 information isConstant is set to Yes for both parts.

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

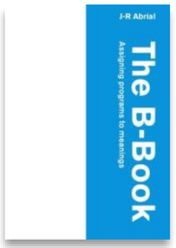
- the constant gradient information (ConstantGradient) is not set.
- the elevationDifference.elevationEnd of the part1 and elevationDifference.elevationStart of the part2 (reference to the above figure) are equal to elevationStart +2*radius*sin(Length1/ (2*radius))*sin(gradientStart +Length1/ (2*radius)).
- 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.

Safety Reasoning

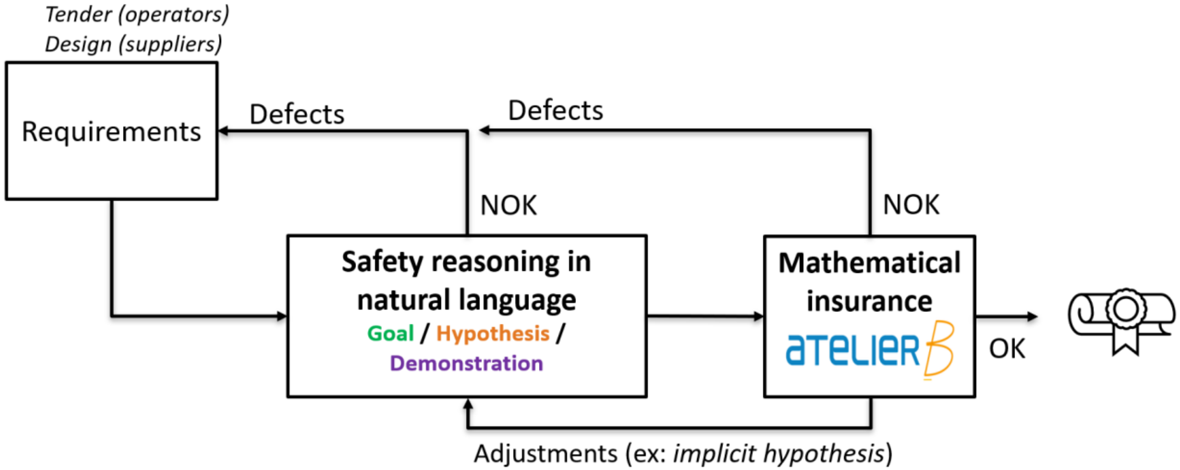
- Formal System Proof
- Achievements

Formal System Proof

Safety reasoning exhibited ("why its was designed this way")
For legacy systems and never implemented specs



SET THEORY
FIRST ORDER LOGIC
INTEGER
BOOLEAN
GRAPHS



References:
- *Formal Proofs for the NYCT Line 7 (Flushing) Modernization Project*, ABZ, 2012
- *Safety Analysis of a CBTC System: A Rigorous Approach with Event-B*, RSSR, 2017

Achievements

2010

New York City Transit (Culver, QBL line CBTC, 8th Avenue Line)
Proof of a new safety automation
Call for tender mentioned Formal Methods

2020-2024

RATP (L3, L5, L9, L6, L11)
Safety proof of OCTYS CBTC

2023-2026

SNCF (Marseille-Vintimiglia)
Safety proof of world-first ETCS L3 hybrid

2024

Calls for tender mention Formal Methods

Summary

- Dealing with the safety reasoning is worthwhile
- Works for legacy systems (safety issues found)
- Works for new, never implemented systems

Global Summary

- Safety critical doesn’t mean that nothing bad could happen
- Dealing with safety brings lots of technicalities (HW, SW, env)
- Formal Methods are tools among other tools
- Properties in the B models are often low level
- « safety problems » still happen

Question

- How can we
  - make the whole process more interesting / more efficient ?
  - increase the level of confidence ?

B+

- The Holy Grail
- The Holy Grenade of Antioch
- Implementing the Holy Grenade Launcher

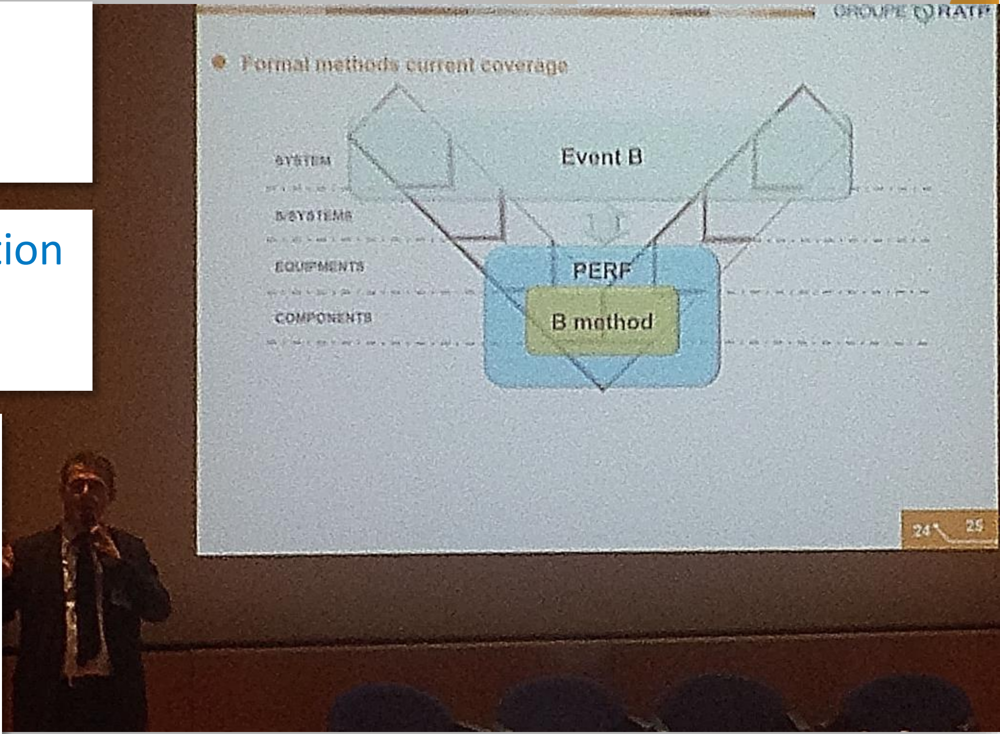
Courtesy of Lilian Burdy, CLEARSY

The Holy Grail

Formal methods as part of RATP's DNA
C. Andlauer, RATP
RSSRail 2016, Paris

The PERF Approach for Formal Verification
D. Bonvoisin, RATP
RSSRail 2016, Paris

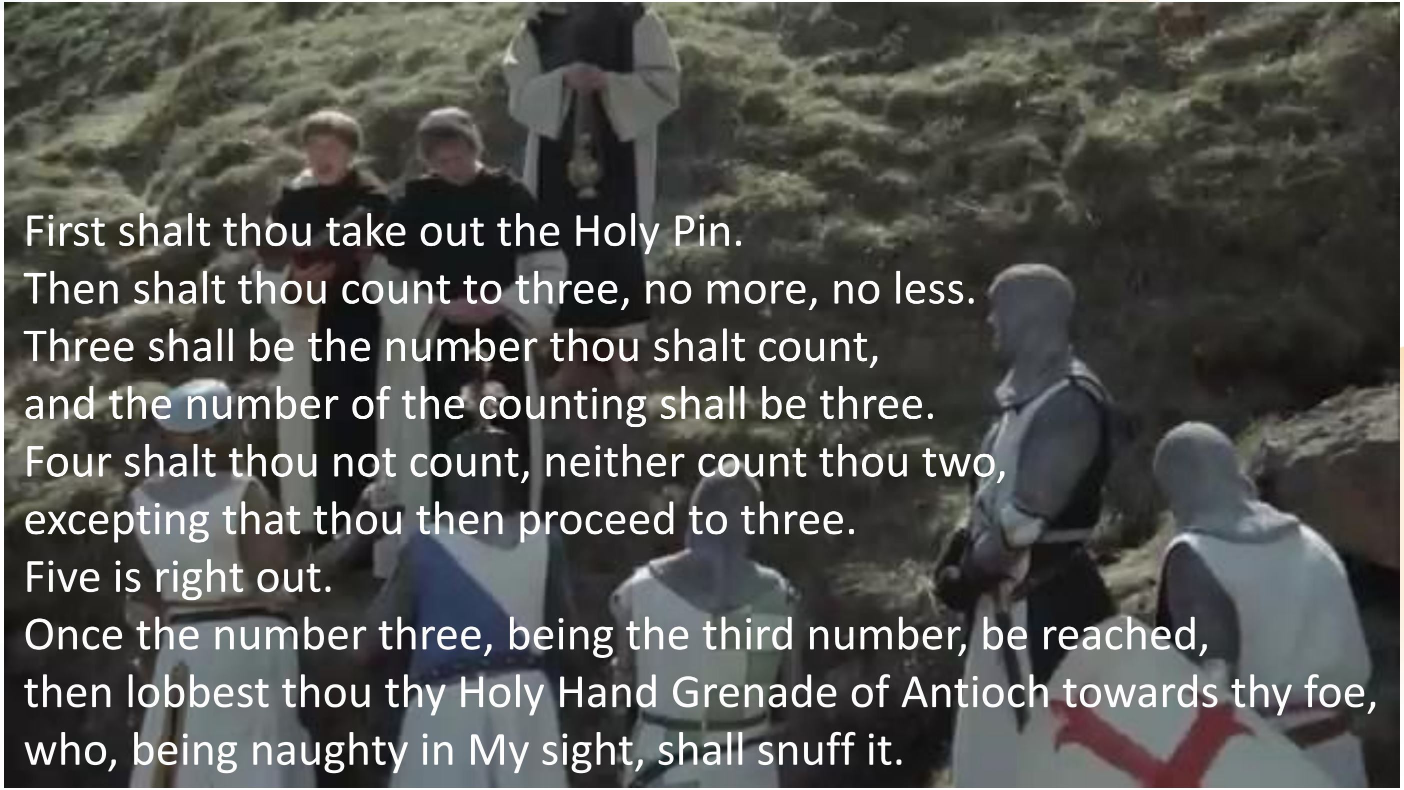
Integral Formal Proof : A Verification Approach to Bridge the Gap between System and Software Levels in Railway System
Alexandra Halchin & al
RSSRail 2023, Berlin



Monty Python and the Holy Grail.

Holy Grenade Specification

Courtesy of Monty Python



Exercise

Holy Grenade

- Determine the expected safety property
- Model it in B
- Implement catapult software that must prove that it maintains the property using unambiguous assumptions about the system's hardware components.

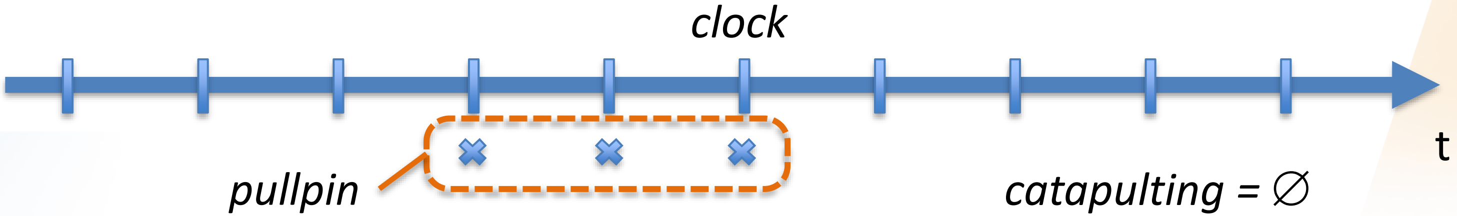
Safety Property

clock:	discretised current time
pullpin:	set of discretised instants where grenade has been pulled
catapulting:	set of discretised instants where grenade catapult is actionned

Variables of the system with a precise meaning

Safety Property

clock: discretised current time
pullpin: set of discretised instants where grenade has been pulled
catapulting: set of discretised instants where grenade catapult is actionned



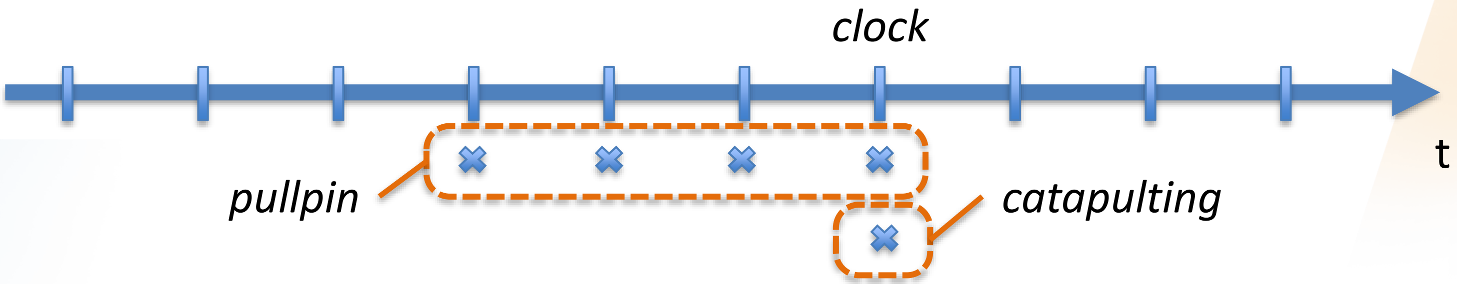
Safety Property

$$\forall t_0.(t_0 \in pullpin \wedge t_0+3 \leq clock \Rightarrow t_0..t_0+3 \cap catapulting \neq \varnothing)$$

clock: discretised current time

pullpin: set of discretised instants where grenade has been pulled

catapulting: set of discretised instants where grenade catapult is actionned



Specification

```
MACHINE The_Holy_Hand_Grenade
 VARIABLES
   pullpin,
   catapulting,
   clock
 INVARIANT
   ...
 INITIALISATION
   ...
 OPERATIONS
```

```
watchdogTimer =
BEGIN
  clock := clock + 1 ||
  pullpin :(pullpin ⊆ 1..clock+1
        ∧ pullpin - pullpin$0 ⊆ {clock+1})
END;

catapult =
BEGIN
  catapulting ∈(catapulting ⊆ 1..clock
        ∧ catapulting - catapulting $0 ⊆ {clock})
END
```

Specification

The environment

```
watchdogTimer =
BEGIN
  clock := clock + 1 ||
  pullpin :(pullpin ⊆ 1..clock+1
        ∧ pullpin - pullpin$0 ⊆ {clock+1})
END;
```

The catapult

```
catapult =
BEGIN
  catapulting ∈(catapulting ⊆ 1..clock
        ∧ catapulting - catapulting $0 ⊆ {clock})
END
```

Refinement

The catapult made more precise

```
catapult =
 BEGIN
  IF clock-2..clock ∩ pullpin ≠ ∅
  THEN
      catapulting := catapulting ∪ {clock}
  ELSE
      catapulting :: {catapulting, catapulting ∪ {clock}}
  END
 END
```

Refinement with deadline from CSP

The CLEARSY Safety Platform ensures that if *catapult* is not called frequently then
It enters a **defect** mode

The defect mode should induce **physically** a catapulting and a pullpin

```
watchdogTimer =
SELECT
  clock < catapulting_deadline
THEN
  clock := clock + 1 ||
  pullpin ∈(pullpin ⊆ 1..clock+1
        ∧ pullpin - pullpin $0 ⊆ {clock+1})
END;
catapult =
BEGIN
 IF clock-2..clock ∩ pullpin ≠ ∅
   ...
 END|| catapulting_deadline :: clock..clock + 2
END
```

Cut MACHINE for Data Acquisition

No direct link with the upper level
Identifiers are renamed

pullpin (system variable)
is linked with
input_pullpin (software variable)

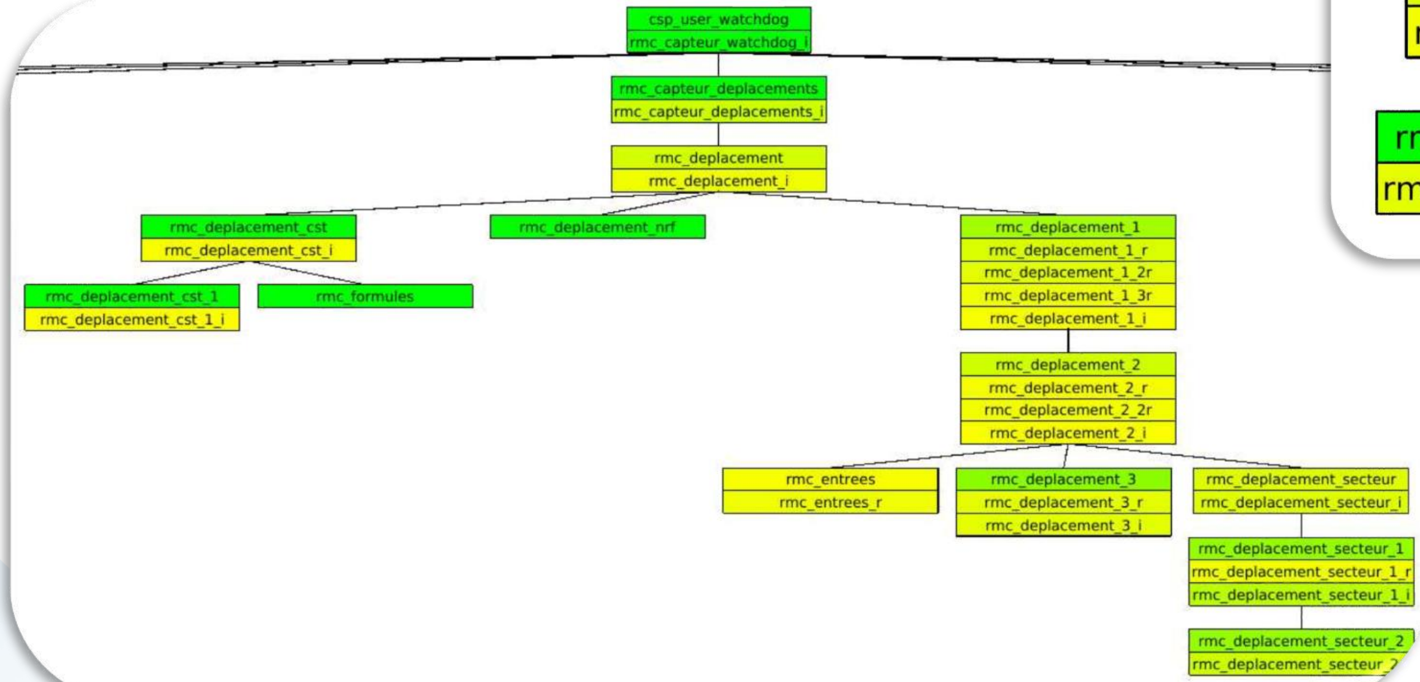
```
input_watchdogTimer =
BEGIN
  input_clock := input_clock + 1 ||
  pullpin ∈(pullpin ⊆ 1..input_clock+1
        ∧ pullpin - pullpin $0 ⊆ {input_clock+1})
END;

input_get_pullpin =
BEGIN
  input_pullpin ∈(input_pullpin ∈ BOOL &
    (input_clock-2..input_clock ∩ pullpin ≠ ∅
⇒ input_pullpin = TRUE))
END
```

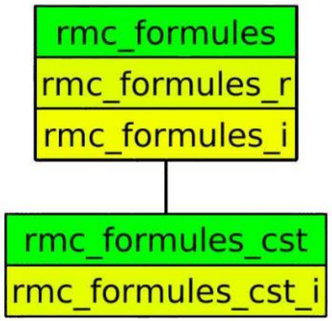
MACHINE for Catapulting

This part contains the exported constraints to this subsystem

```
catapult_watchdogTimer =
 SELECT
    clock < catapulting_deadline
 THEN
    clock := clock + 1
 END;
 catapult_catapulting =
 BEGIN
   IF input_pullpin = TRUE
   THEN
     catapulting := catapulting ∪ {clock}
   END ||  catapulting_deadline := clock + 2
 END
```

Resulting Architecture





What is missing ?

- Time between decision and effective physical catapulting
- OPERATIONS *watchdogTimer* could happen while *catapult* is being executed
- Performances as a side-note in the safety demonstration
  - Physical-arithmetic modelling would add unwanted complexity

Application to PSD Control in Brisbane (2024)

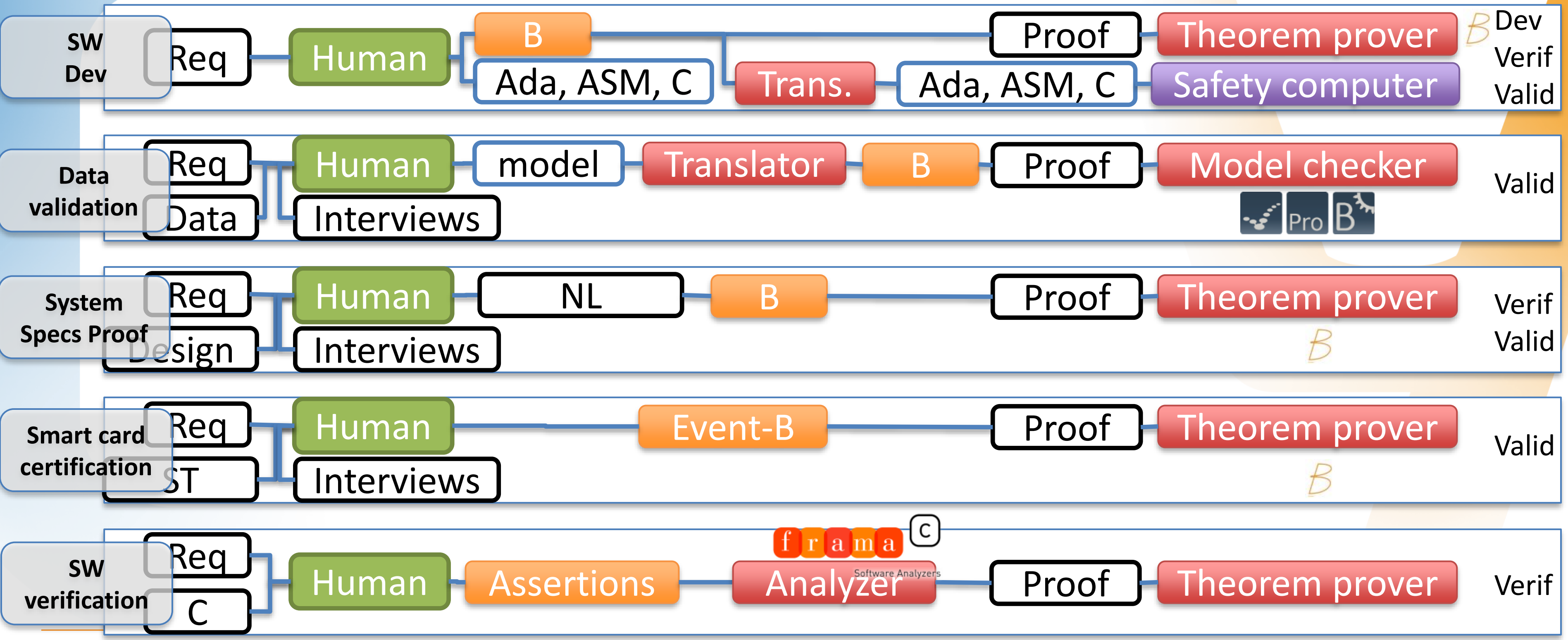
"The on-board vital software shall activate a side selection output only if a valid wayside message denoting an established communication indicates the corresponding side."

- Communication is ensured by beacons energized by the train
- Only sections with PSD have beacons
- Driver has to push a button for a side
- Beacons have ID plugs
- Valid message received recently from beacon
- Software behaviour based on system-level properties and not (only) on software defined variables

```
v_ob_trainAlignedLeftSide
:= bool(v_ob_commEstablished = TRUE
        ∧ v_ob_commRestrictive = FALSE
        ∧ v_ob_communicatingAntenna ∈
{e_TSA2,e_TSA3})
```

- SIL3 system
- 8 platforms
- 150 safety computers installed onboard
- 8 safety computers installed on trackside

Formal Methods in Action



Conclusion

- Why do we use formal methods ?
  - We are more efficient, more competitive, more flexible
  - Enhance the safety demonstration (clarity, test vs proof even if we test)
  - Help us to keep things under control
  - We find problems on existing systems / never implemented specs
- What perspective ?
  - Problem not yet « solved »: incidents, accidents still happen
  - FM requirement appears in call for tender
  - Applied also in non-safety related domains (“*do not lose the drone*”)
  - Room for improvement, contribution to SotA
  - **Human is central**
  - **Universities should produce “Leonardos”**

CLEARSY

Safety Solutions Designer

FM/FMICS
SEP2024

AIX
LYON
PARIS
STRASBOURG

WWW.CLEARSY.COM

Thank you for your attention

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

MOOC
massive open online course

THIERRY.LECOMTE@CLEARSY.COM