CLEARSY
Safety Solutions Designer

ETMF
DEC2024

Math Saves Lives

How Formal Methods Keep Trains on Track and Software from Going Off the Rails

Thierry Lecomte
R&D Director

THIERRY.LECOMTE@CLEARSY.COM

« The presentation provides a comprehensive overview of the role of formal methods in ensuring the safety in the railways.»

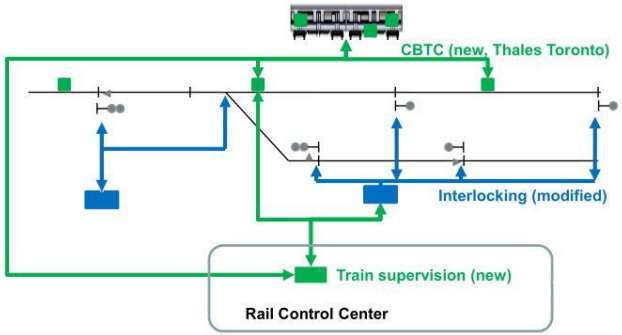
Art mostly generated with *ChatGPT or similar*

CLEARSY

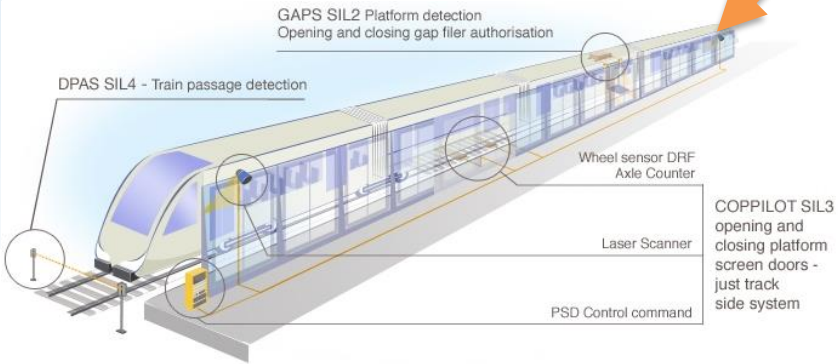
Safety Solutions Designer

Metro automatic pilots

SME created in 2001
150+ engineers
~120 MR$ turnover 2022
Aix, Lyon, Paris, Strasbourg



System formal engineering, RAMS, services



SIL2 - SIL4 safety critical systems

ATELIER B

Editor

SIL4 products

http://www.clearsy.com/

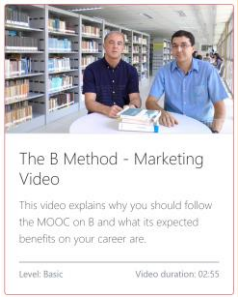
My CV

Development of Atelier B and its proof tools

Applied Formal methods
Teached at IMD

R&D and education

28 EU and French R&D projects
CLEARSY Safety Platform
Courses in 30 universities and schools
MOOC on B



Applications to industry

Smartcard certification
ProB for formal data validation
Railways signalling software
SP metrô: platform screen doors control

My Subjects

Safety

How not to kill people ?

Autonomous Mobility

How to remove control from human?

Cybersecurity

How to protect secrets ?

Artificial Intelligence

How to do smart things?

Common Thread

We are going to

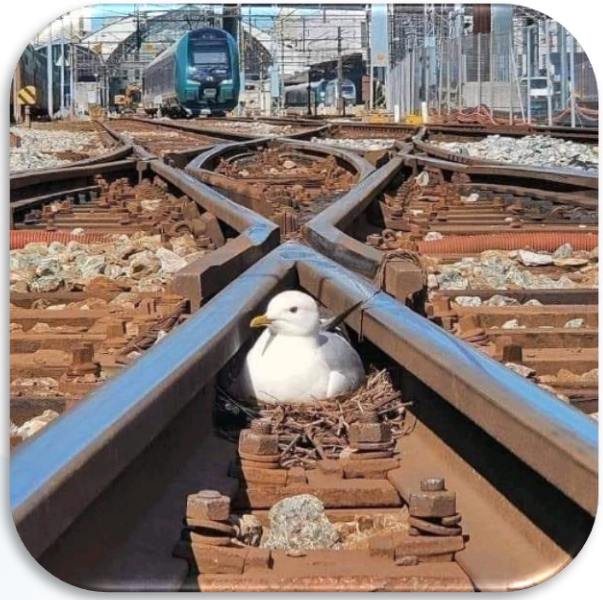
design a train line

with different configurations

and see

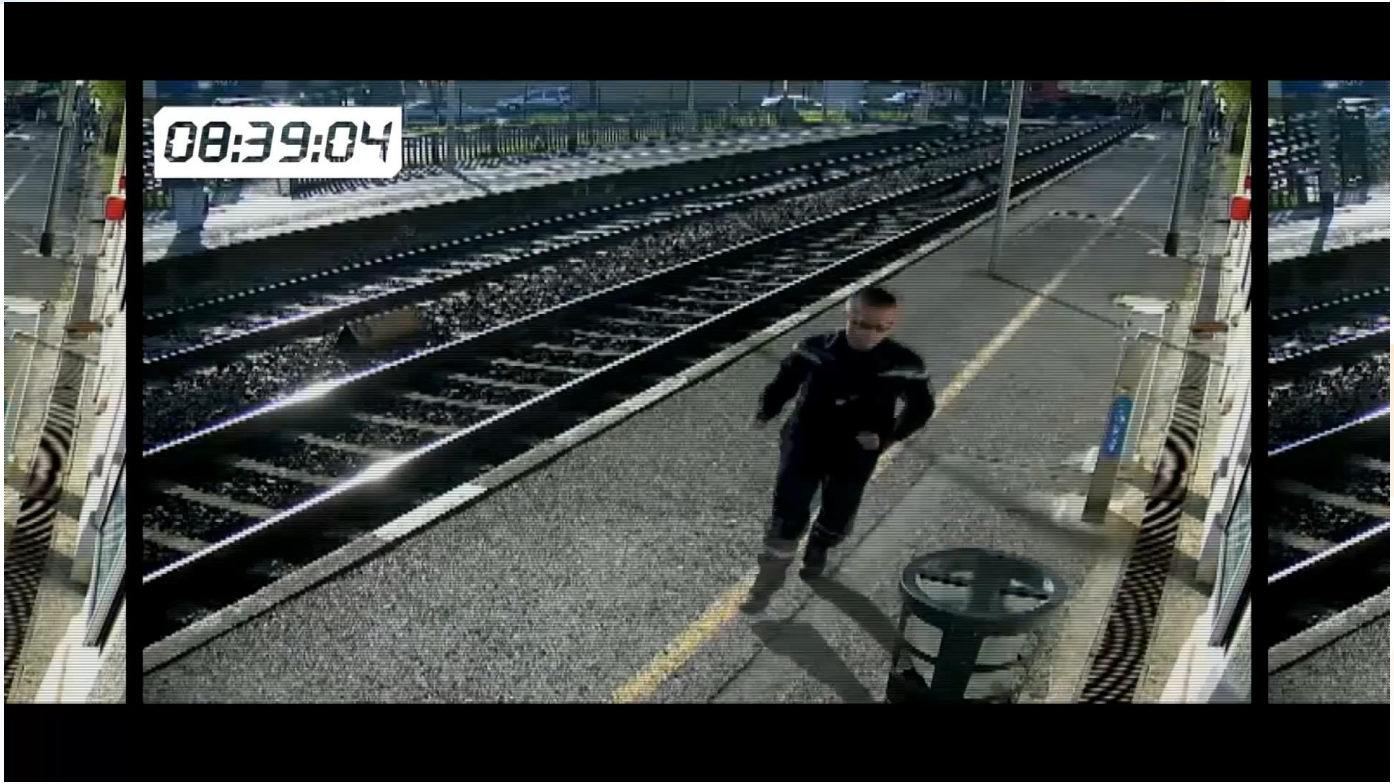
how formal methods can help

SAFETY

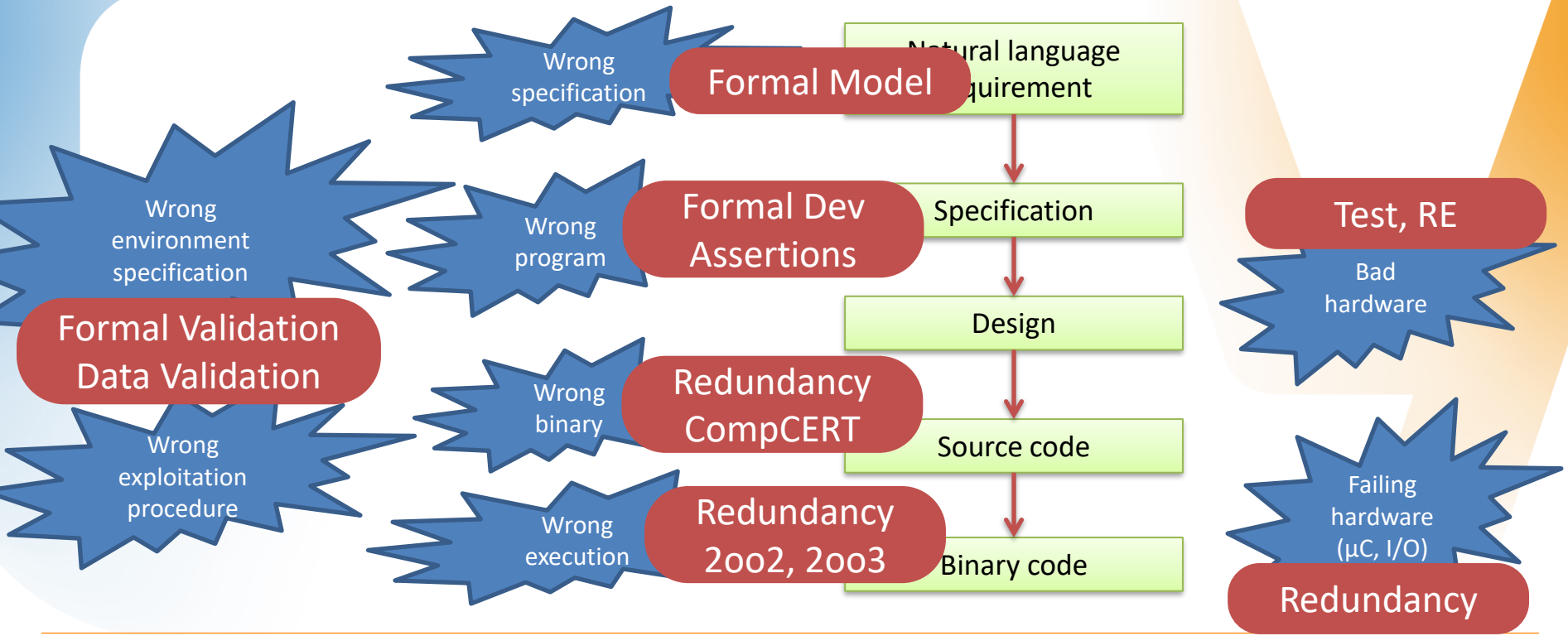


- Failing systems
- Safety critical
- Standards

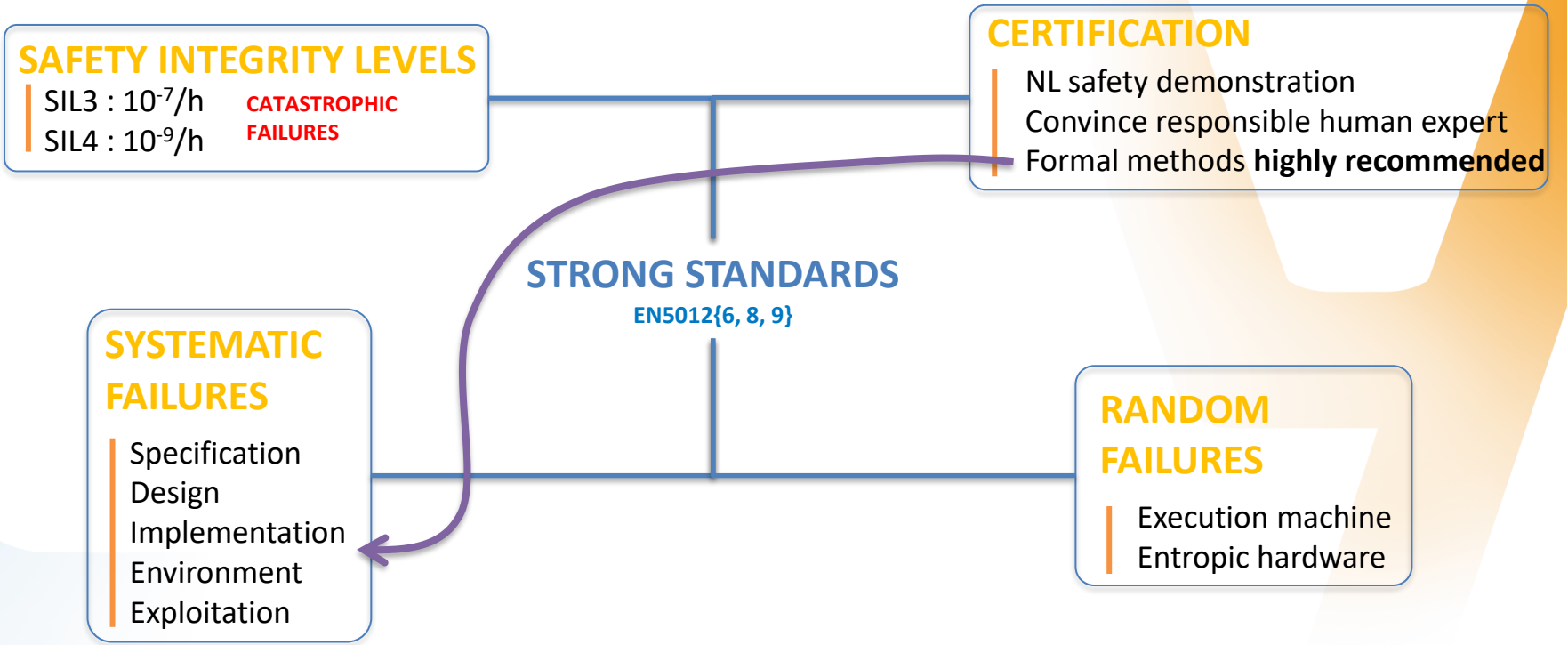
Safety is about things that happen 1 in 1,000,000



Failing Software-Based Systems



Safety @ Railways

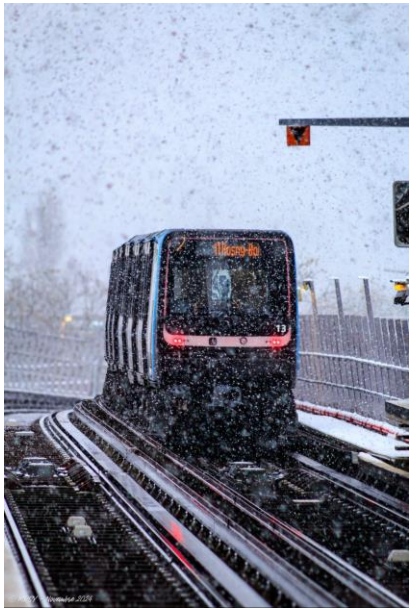


Railways in a Nutshell

- Move persons / goods from point A to B
  - using predefined paths
- Trams, Metros, Freight train, Passenger Trains, High Speed Train

Safety Properties

- No collision
- No overspeed
- No derailment



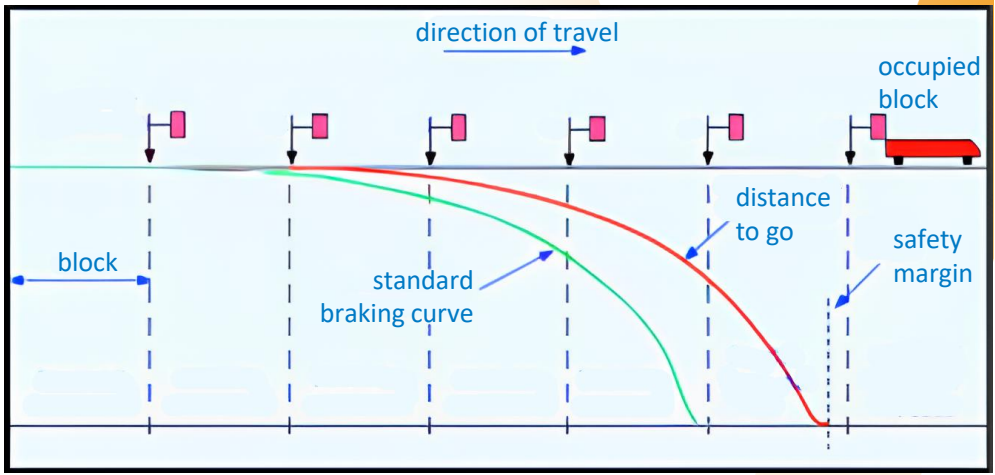
Whatever the conditions

No collision

- Collision with trains
- Collision with obstacles
- Collision with animals

No collision

- Collision with trains
  - Braking curves
- Collision with obstacles
  - Driver, tunnels, sensors
- Collision with animals
  - Driver, fences, tunnels, sensors



Braking curves

No Overspeed

- Emergency braking if speed over speed limit
  - Shutdown engine, trigger break
  - Alarms (sound) when approaching speed limit
- Means to measure speed
  - Odometer, GNSS out
- Resume travel when at full stop

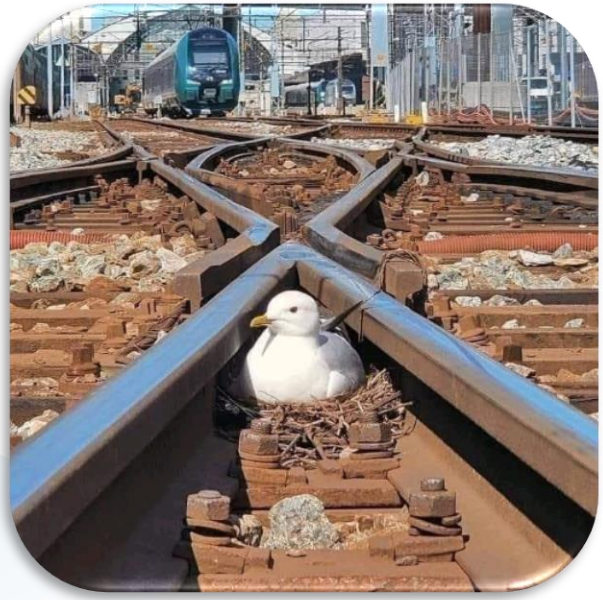
No Derailment

- Prevent switch to move when a train is passing by

No Derailment

- Routes are allocated to trains by interlocking (IXL)
  - Switch fixed until the train leaves
- The absence of derailment is not always guaranteed
  - Ex: after an earthquake

CASE-STUDY



- Monocab
- Configuration 1 (intro to B)
- Configuration 2
- Configuration 3
- Data Validation

- ❑ Adding decisions, details
- ❑ Perform some (simplified) verifications of different nature

MONOCAB

(https://www.monocab-owl.de/english-language/)

- Compact, autonomous monorail vehicle designed to revitalise disused railway lines
- Offers an innovative transport alternative in rural areas.
- Reuses existing German railway infrastructure

MONOCAB

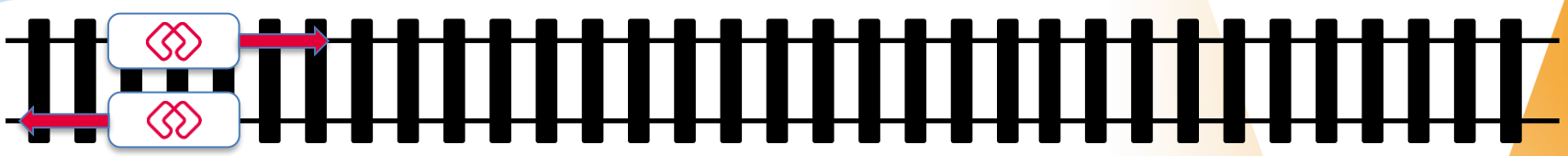
(https://www.monocab-owl.de/english-language/)

- Uses advanced gyroscopic technology to stabilise itself on a single conventional track,
- Vehicles travelling in opposite directions pass on the same track.

MONOCAB ao vivo

CONFIGURATION 1

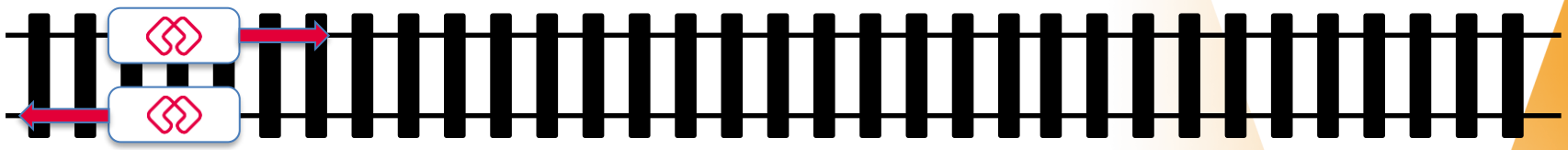
Configuration 1



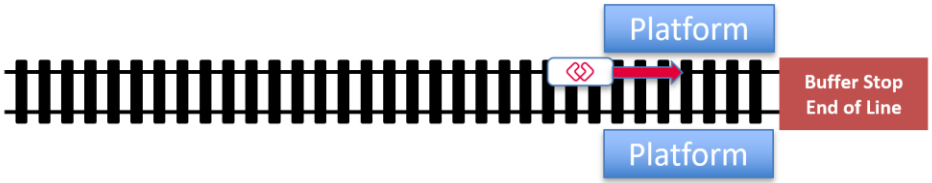
- 5 km line between only 2 stations
- Capacity: 8 passengers per car
- Operating speed: 50 km/h
- Slope: 0%
- No crossroads

- 2 trains in opposite phase on separate rail
- Hypotheses:
  - 6’ for a ride;
  - 1’ for passengers transfer
- Traffic:
  - 8.5 travels / h / train
  - 136 passengers / h both ways

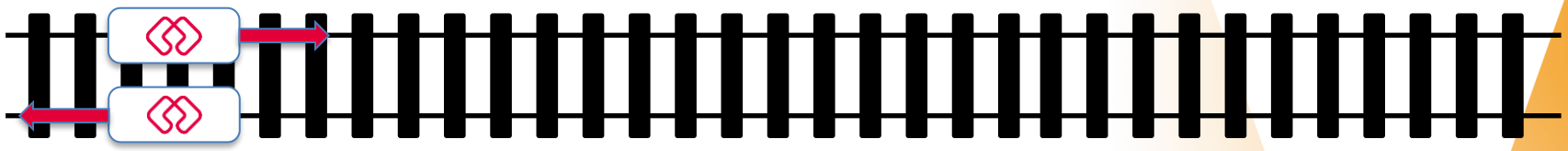
Configuration 1: safety



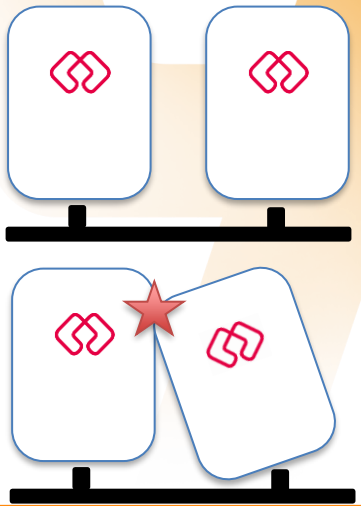
- No collision with train - on same rail
  - 1 train per rail
- No collision with train - on other rail ?
- No collision with obstacles
  - Flat ground, no slope, no trees around
  - Collision with buffer stop
- No collision with animals
  - Fences on both side, no crossroads
- No overspeed
  - Need reliable speedometer, minimum guaranteed braking capabilities
- ~~No derailment (no switch)~~
- NB: we are not controlling train doors opening but we could reduce speed if someone open a door during travel



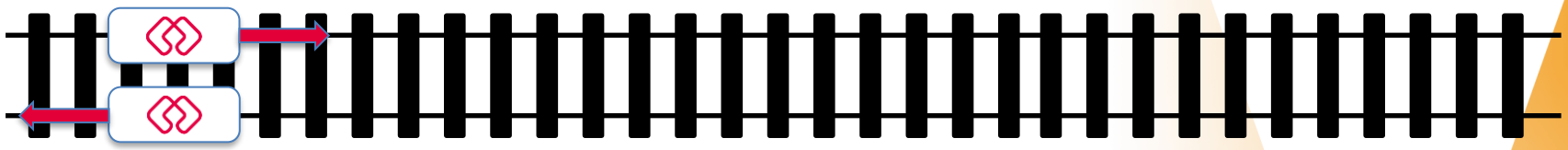
Configuration 1: safety



- No collision with train - on other rail ?
  - If gyroscopic mechanism fails
  - If unexpected conditions (shaky passengers, strong lateral wind)
- How to limit risks ?
  - ~~Reduce speed when trains “meet” (how ? when ?)~~
  - Reduce operation speed when windy



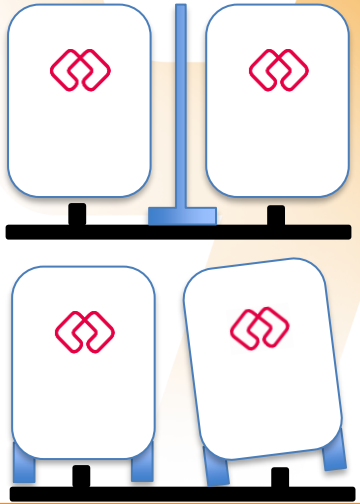
Configuration 1: safety



- How to avoid ?
  - Physical obstacle between cars (wall, pole) – and a way to detect the situation (contact, angle, speed)

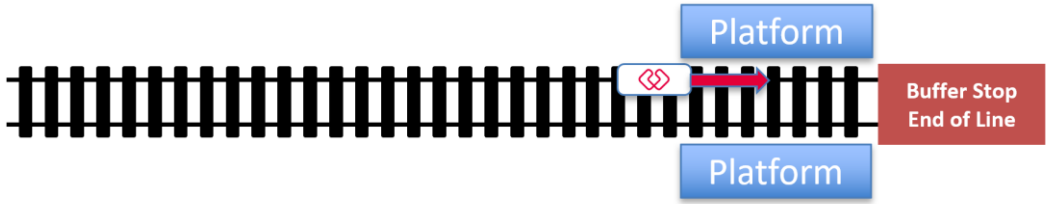
DECISION

  - “Feets” (as in the video) to limit angular deviation (but risk of hitting a low obstacle on the track)
  - Telescopic “feets” that are deployed when angular deviation is detected (such a sensor is needed) – and train is stopped



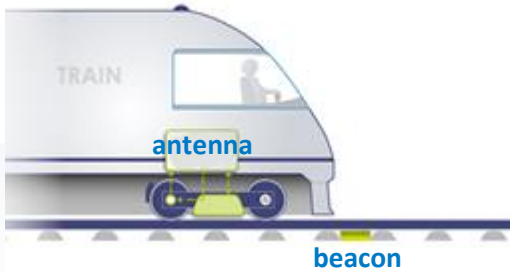
Configuration 1: safety

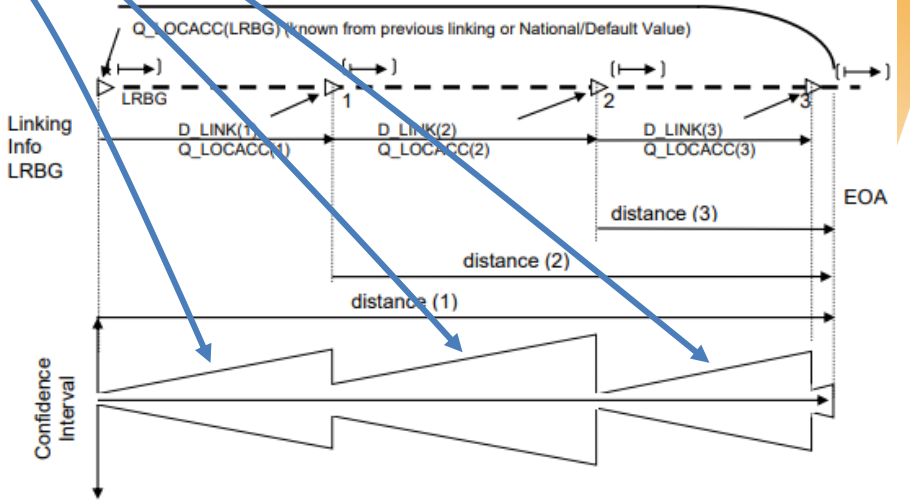
- No collision with obstacle – buffer stop
  - We need to know when the train is approaching the platform to slow down and avoid collision



Configuration 1: safety

- No collision with obstacle – buffer stop
  - Use an odometer but increasing error over time
  - Beacons to provide location when confidence is too low

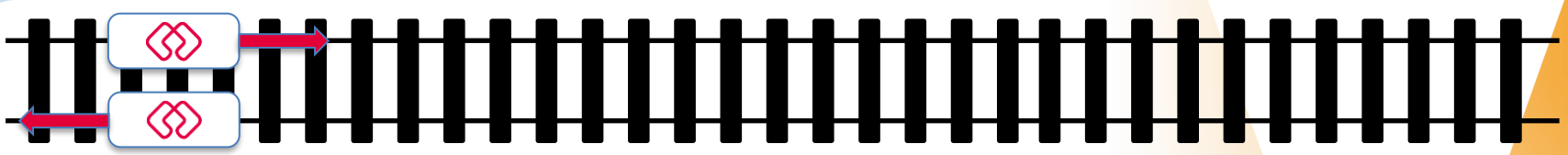


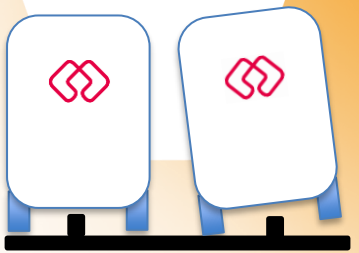


Configuration 1: safety

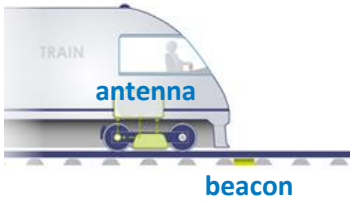
- No collision with obstacle – buffer stop
  - Beacons can be missed or misread (electronic failure, beacons moved, covered by object)
  - We need diversity to estimate if we are close to the end of the line
  - It could be a laser measuring the distance with the platform (but could be misaligned and function becomes critical) **DECISION**
  - An internal timer could activate a watchdog if 6' – 30'' have passed since the start of the trip, the speed limit is decreased.
  - To stop the train, we could add gravels at the end of the line

Configuration 1: safety

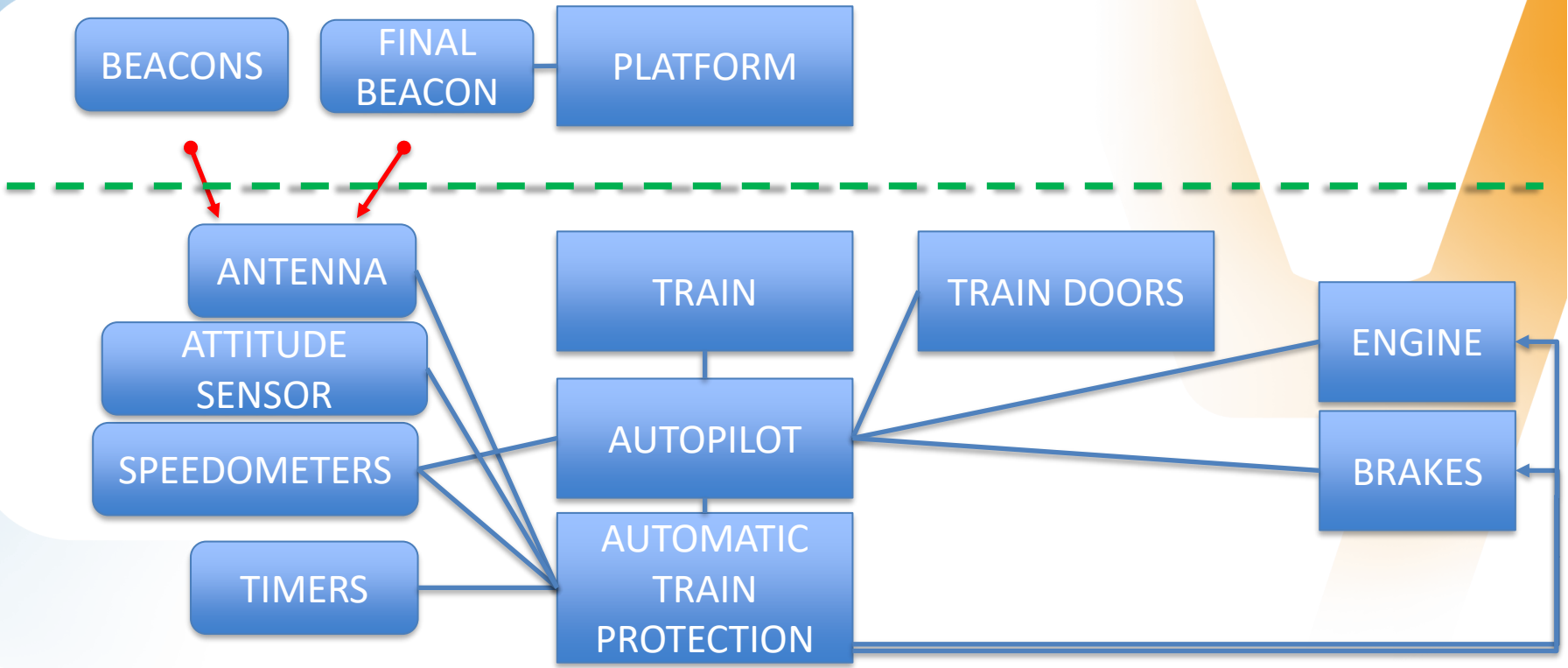




- Summary
  - 1 train per rail, 2 trains total operated independently, no slope, no crossroads, fences on both sides
  - Reduce speed operation when windy (experiments needed to define speed limits / wind speed)
  - Fixed feets to limit angular deviation
  - Use beacons and timer to determine if the train is approaching the end of line (engineering to determine number of beacons and locations, speed profile)
  - Need reliable speedometer, minimum guaranteed braking capabilities



System Analysis



SHORT INTRODUCTION TO B

« 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;
```

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)

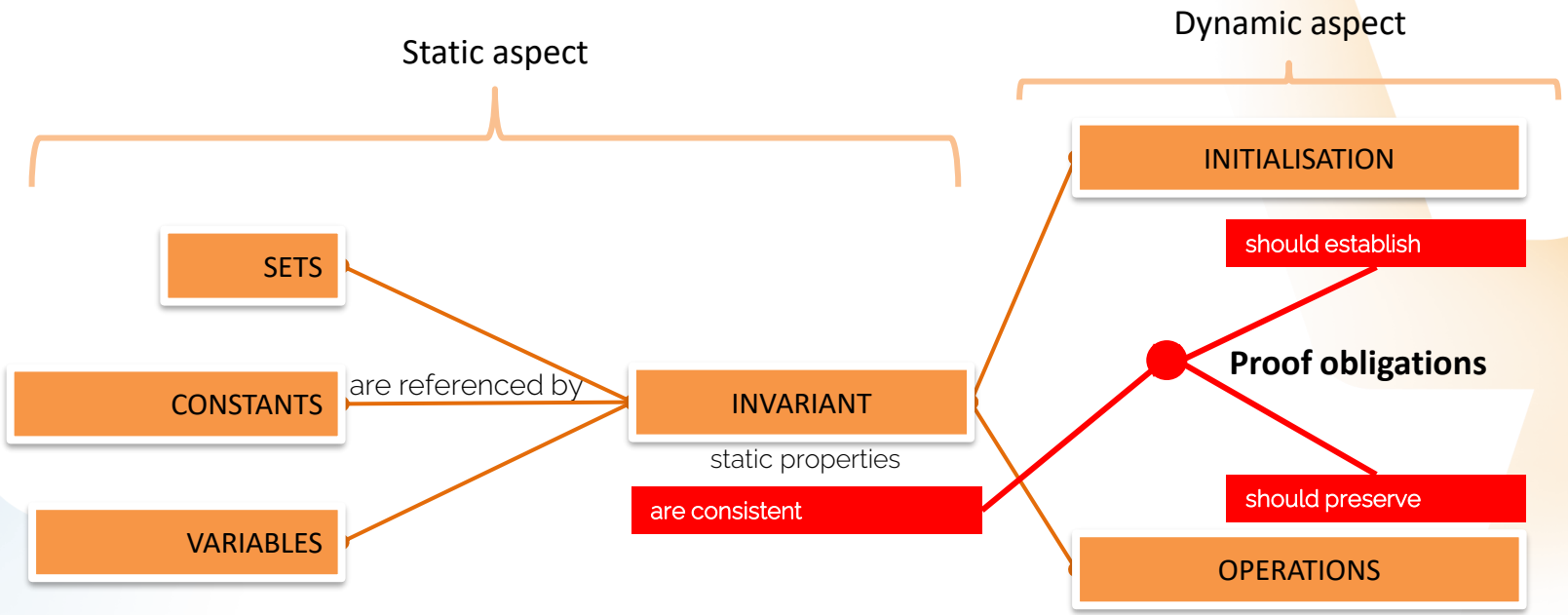
```
void M0__activation_sequence(void)
{
    CTX__SEQUENCES sequ;

    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

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

```
11  static int32_t M0__xx;
12  /* Clause INITIALISATION */
13  void M0__INITIALISATION(void)
14  {
15
16      M0__xx = 0;
17  }
18
19  /* Clause OPERATIONS */
20
21  void M0__inc(void)
22  {
23      if(M0__xx == 2147483647)
24      {
25          M0__xx = 0;
26      }
27      else
28      {
29          M0__xx = M0__xx+1;
30      }
31  }
```

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

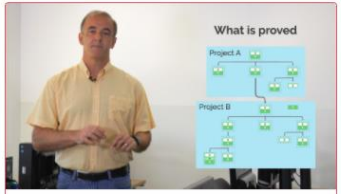
This video presents the structure of the course, provides an overview of the different kinds of formal methods and specification styles, and tells us some myths on formal methods

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)

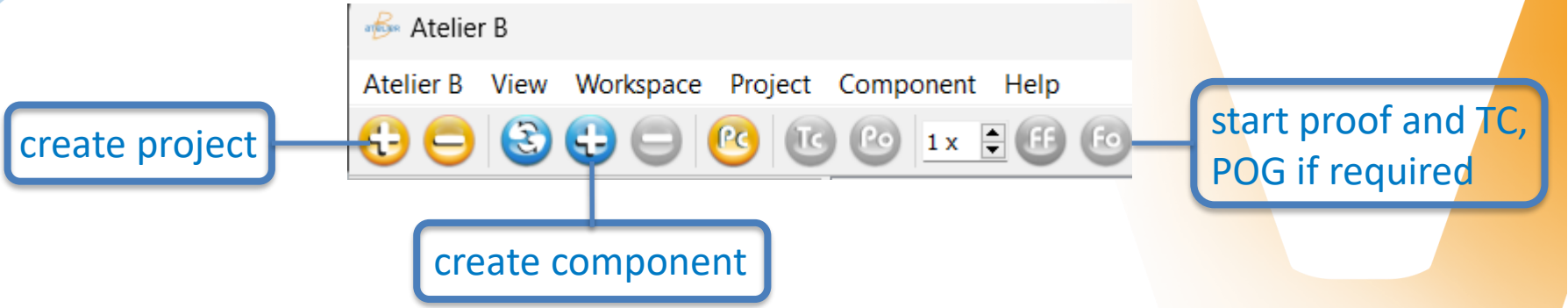
2024-2030

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

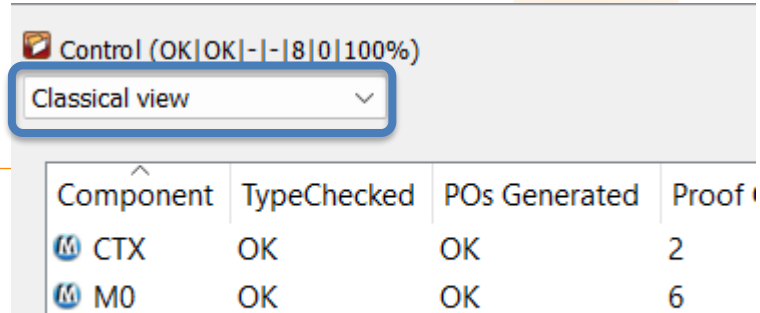
Formal IDE for B

- ATELIER B https://www.atelierb.eu/en/atelier-b-support-maintenance/download-atelier-b/
  - Model software (B) and systems (Event-B)
  - Automatic and interactive proof
  - Code generation
  - Simple installer
- PROB https://prob.hhu.de/w/index.php?title=Download
  - Same support but simple B models architecture
  - Model checking, verification
  - Animation
  - Requires Java, Tcl/Tk

ATELIERB

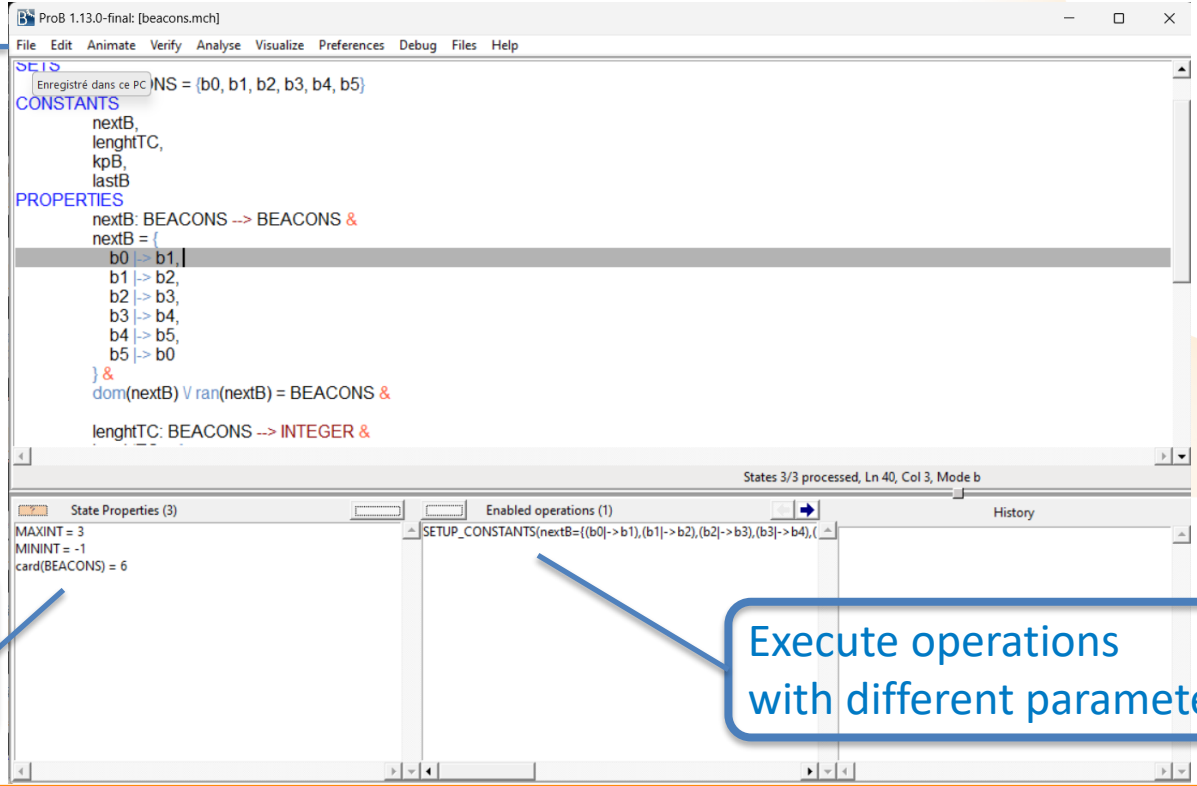


- If asked, relocate your projects in a directory where you can write
- Create a project (software development)
- Add component with « + » blue button or drag/drop file from Windows explorer (Classical view)



PROB

Load file



Observe values of variables and constants

Execute operations with different parameters

M0.mch CTX.mch

```
1  MACHINE
2      CTX
3  SETS
4      BEACONS = {b0_stop, b1_leave, b2_approach, b3_approach, b4_enter, b5_stop}
5  CONSTANTS
6      S_MANOEUVER, /* minimum speed limit - garanteed not to injure passengers */
7      S_MAX, /* maximum speed limit */
8      S_BEACONS, /* associate to each beacon a speed limit */
9      DELAY_TRAVEL_APPROACH, /* delay where we consider that
10                               first approach beacon has been missed */
11     NEXT_BEACONS /* What is the order of reading beacons - for simulation */
12 PROPERTIES
13     S_MANOEUVER: INTEGER &
14     S_MAX : INTEGER &
15     S_MANOEUVER > 0 &
16     S_MAX >= S_MANOEUVER &
17     S_BEACONS : BEACONS --> INTEGER &
18     ran(S_BEACONS) <: S_MANOEUVER..S_MAX &
19     S_BEACONS = {
20         b0_stop |-> S_MANOEUVER,
21         b1_leave |-> S_MAX,
22         b2_approach |-> S_MAX,
23         b3_approach |-> S_MAX,
24         b4_enter |-> S_MANOEUVER,
25         b5_stop |-> S_MANOEUVER
26     } &
```

Context machine, No variable, Only sets and constants

Typing + properties

Total function

Codomain is included in interval

Speed limit per beacon

https://github.com/CLEARSY/ETMF2024/

Directory: Configuration1

Model: CTX.mch

M0.mch CTX.mch

```
28        DELAY_TRAVEL_APPROACH : NATURAL1 &
29        DELAY_TRAVEL_APPROACH <= 10 &
30        NEXT_BEACONS : BEACONS --> POW(BEACONS) &
31-       NEXT_BEACONS = {
32            b0_stop |-> {b0_stop, b1_leave, b2_approach, b3_approach, b4_enter, b5_stop},
33            b1_leave |-> {b1_leave, b2_approach, b3_approach, b4_enter, b5_stop},
34            b2_approach |-> {b2_approach, b3_approach, b4_enter, b5_stop},
35            b3_approach |-> {b3_approach, b4_enter, b5_stop},
36            b4_enter |-> {b4_enter, b5_stop},
37            b5_stop |-> {b5_stop}
38        }
39    END
```

Value only to shorten the simulation

The order of beacon reading,
Only for simulation

M0.mch CTX.mch

```
1     MACHINE M0
2-    SEES CTX
3
4-    VARIABLES
5         current_speed, /* current speed - input */
6         last_beacon_read, /* id of last beacon read - input */
7         current_speed_limit, /* current speed limit computed */
8         emergency_braking, /* trigger for emergency braking - output */
9         travel_time, /* time (cycles) measured since the beginning of travel */
10        travel_completed
11-   INVARIANT
12 1/1    current_speed : NATURAL &
13        last_beacon_read : BEACONS &
14 1/1    current_speed_limit : S_MANOEUVER..S_MAX &
15        emergency_braking : BOOL &
16        travel_time : NATURAL &
17        travel_completed : BOOL &
18        /* Safety p1: overspeed implis emergency braking */
19        (current_speed > current_speed_limit => emergency_braking = TRUE)
20
21-   INITIALISATION
22 1/1    current_speed := 0 ||
23        last_beacon_read := b0_stop ||
24 1/1    current_speed_limit := S_MANOEUVER ||
25        emergency_braking := TRUE || /* safety position
26        we leave this state if conditions met */
27        travel_time := 0 ||
28        travel_completed := FALSE
```

Variables

Typing +properties

Initial values all set in //

https://github.com/CLEARSY/ETMF2024/

Directory: Configuration1

Model: M0.mch

```
30-  OPERATIONS
31       cycle_b0_b5 =
32-      PRE travel_completed = FALSE THEN
33           current_speed,
34           emergency_braking,
35   2/2     last_beacon_read,
36-          travel_time: (
37               current_speed : NATURAL & /* input */
38               last_beacon_read : BEACONS & /* input */
39               /* to ensure that past beacons are read */
40   2/2         last_beacon_read : NEXT_BEACONS(last_beacon_read$0) &
41               emergency_braking : BOOL & /* output */
42               travel_time : NATURAL &
43               travel_time > travel_time$0 &
44
45-              (travel_time > DELAY_TRAVEL_APPROACH =>
46                   current_speed_limit = S_MANOEUVER) &
47-              (travel_time <= DELAY_TRAVEL_APPROACH
48   2/2             => current_speed_limit = S_BEACONS(last_beacon_read)) &
49
50-              (emergency_braking = TRUE => current_speed
51                   <= current_speed$0 & current_speed >= 0) &
52               /* safety p1 */
53-              (current_speed > current_speed_limit
54                   => emergency_braking = TRUE) &
55               /* safety p2 */
56-              (emergency_braking$0 = TRUE & current_speed = 0
57                   => emergency_braking = FALSE) &
58               /* safety p2 */
59-              (emergency_braking$0 = TRUE & current_speed > 0
60                   => emergency_braking = TRUE)
61           )
62       END;
```

Precondition

Substitution: variables become such as …

Var$0 is the value of Var before the exécution of the operation

Time is passing

Once the approach delay is over, speed limit is set to minimum

Logic for the emergency braking

M0.mch CTX.mch

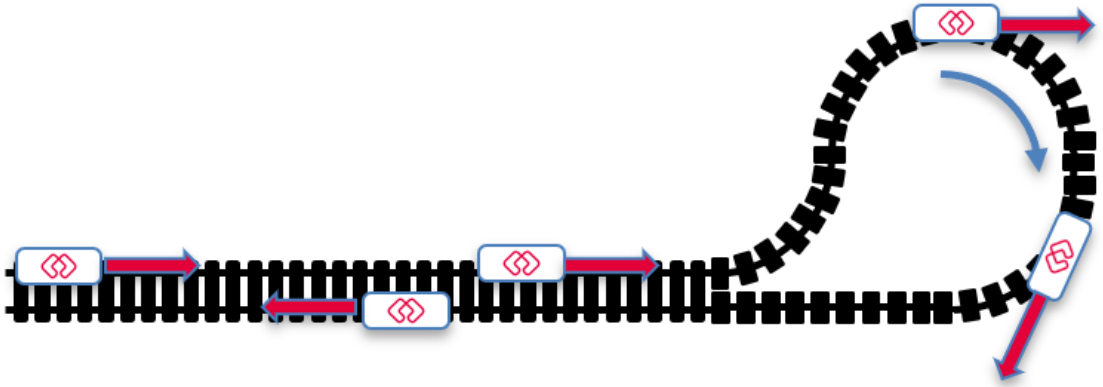
```
64          end_travel =
65-         PRE
66              travel_completed = FALSE &
67              last_beacon_read = b5_stop &
68              current_speed = 0
69-         THEN
70              travel_completed := TRUE
71          END
72
73      END
```

For simulation

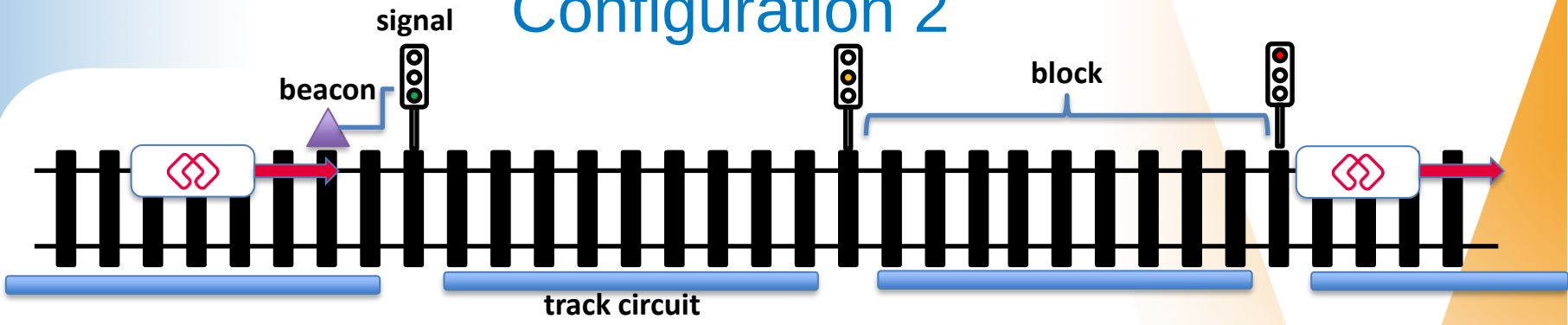
CONFIGURATION 2

Configuration 2

- Higher capacity
- Several trains
- Looped lines
- To avoid collisions
  - Signals
  - Interlocking

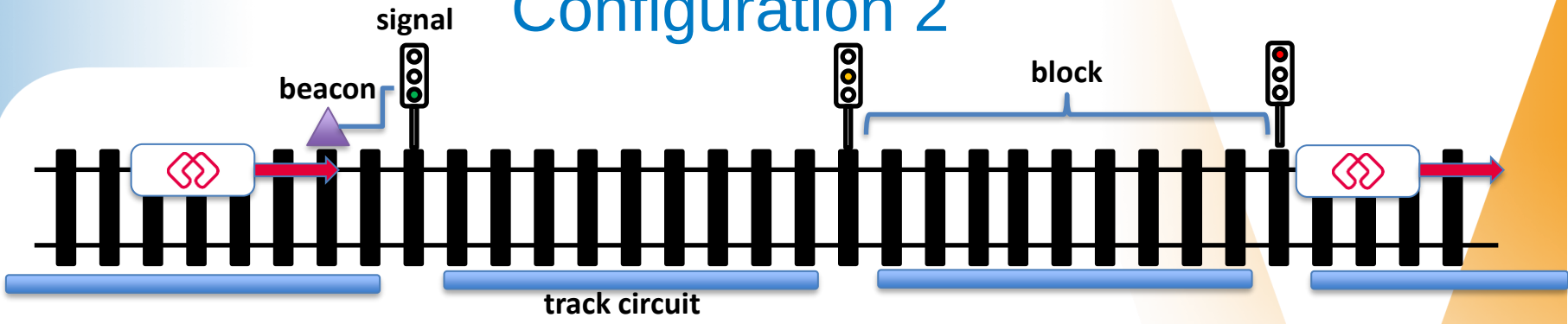


Configuration 2



- Definition of fixed blocks
- Track circuit to detect if there is a train in the block
- One train maximum per block
- Beacon: similar to position beacon, but message is variable

Configuration 2



- Signal (virtual)
  - Red: next block is occupied, stop before next signal
  - Orange: next next block is occupied, slow down, prepare to stop
  - Green: next two blocks are free, continue
- Interlocking:
  - Detect train with track circuit, position signal and beacon message
  - When a train goes over a beacon, reads the status of the virtual signal
  - ATO to slow down, ATP to check that the train is able to stop before red signal

https://github.com/CLEARSY/ETMF2024/

Directory: Configuration2

Model: CTX.mch

CTX.mch

```
MACHINE
    CTX
SETS
    TRACK_CIRCUITS = {tc1, tc2, tc3, tc4, tc5, tc6, tc7, tc8, tc9};
    SIGNALS = {s1, s2, s3, s4, s5, s6, s7, s8, s9};
    STATUS = {GREEN, RED}
CONSTANTS
    IS_PROTECTED_BY
PROPERTIES
    IS_PROTECTED_BY : TRACK_CIRCUITS +-> SIGNALS &
    IS_PROTECTED_BY = {
        tc1 |-> s1,
        tc2 |-> s2,
        tc3 |-> s3,
        tc4 |-> s4,
        tc5 |-> s5,
        tc6 |-> s6,
        tc7 |-> s7,
        tc8 |-> s8,
        tc9 |-> s9
    }
END
```

Static configuration

As many signals as TC

Each signal protects a TC

https://github.com/CLEARSY/ETMF2024/

Directory: Configuration2

Model: IXL.mch

IXL.mch

```
1-      MACHINE
2           IXL
3       SEES CTX
4-      VARIABLES
5           is_occupied,
6           signal_status
7-      INVARIANT
8   1/1     is_occupied <: TRACK_CIRCUITS &
9   1/1     signal_status : SIGNALS --> STATUS
10-     INITIALISATION
11  1/1     is_occupied :: POW(TRACK_CIRCUITS) ||
12  1/1     signal_status := SIGNALS * {RED}
13-     OPERATIONS
14          update_protection =
15-         BEGIN
16-             signal_status: (
17                  signal_status : SIGNALS --> STATUS &
18                  /* all signals protecting an occupied TC should be red */
19                  signal_status[IS_PROTECTED_BY[is_occupied]] = {RED}
20              )
21          END
22      END
```

TC occupancy (input)

Signal status (output)

Set of occupied TC

Signals state

Any subset of TRACK_CIRCUITS

All signals are RED

Not deterministic specification

All signals may remain RED all the time

No problem: it is safe !

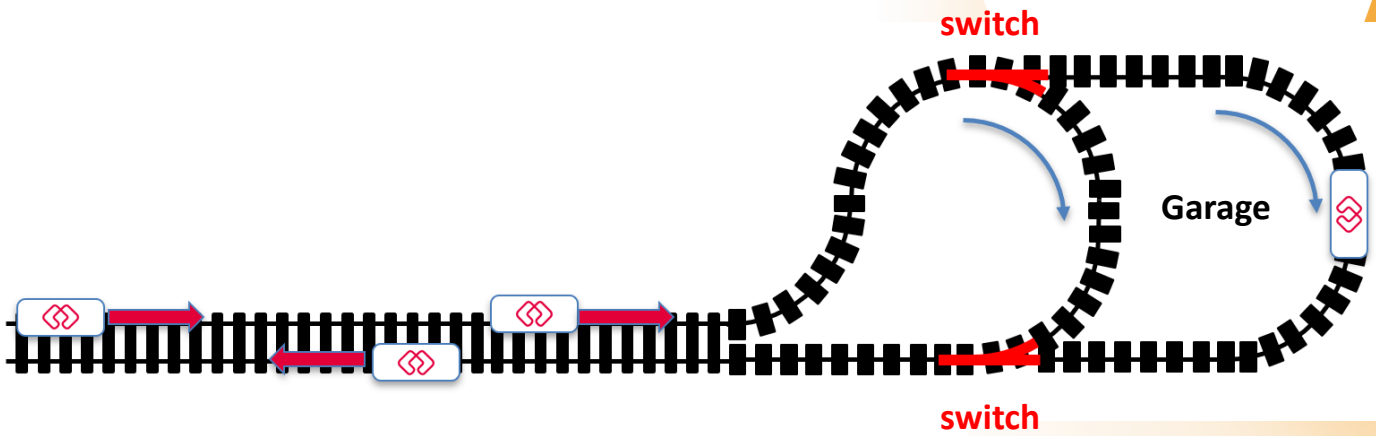
Controlling Signals

- We cannot specify where the trains are as we do not have this information
- Sensor information is made redundant and diverse
  - track circuits,
  - border detectors
- Real processing more complex with
  - history from cycle to another
  - 3 colors: Red, Orange, Green for optimisation

CONFIGURATION 3

Configuration 3

- Garage
- Switches



- Interlocking control switches position (**left, right, unknown**)
- Redundant sensors to detect switch position
- Switch cannot move when associated track circuit is occupied (=> derailment) (orientation is ignored here)

Detecting the position of a switch

- With 3 contact sensors !
- Each sensor returns a measure:
  - Right,
  - Left,
  - Unknow (failing sensor, switch moving during measurement)
- The position of the switch is
  - Right, at least one right, no left
  - Left, at least one left, no right
  - Unknown otherwise

https://github.com/CLEARSY/ETMF2024/ Directory: Configuration3 Model: BLADE*

BLADE.mch

```
1  MACHINE
2      BLADE
3  SETS
4      POSITION = {Left, Right, Unknown}
5  OPERATIONS
6      pos <-- estimate (s1, s2, s3) =
7      PRE
8          s1 : POSITION &
9          s2 : POSITION &
10         s3 : POSITION
11     THEN
12         IF Right: {s1, s2, s3} THEN
13             IF Left: {s1, s2, s3} THEN
14                 pos := Unknown
15             ELSE
16                 pos := Right
17             END
18         ELSIF Left: {s1, s2, s3} THEN
19             pos := Left
20         ELSE
21             pos := Unknown
22         END
23     END
24 END
```

Specification component

Fully proved

Stateless component
No variable

Parameters typing
in preconditions

Set theory based specification

BLADE_i.imp | BLADE2_i.imp

Implementation component

```
1       IMPLEMENTATION BLADE2_i
2       REFINES BLADE
3-      OPERATIONS
4   24/24   pos <-- estimate (s1, s2, s3) =
5-      IF s1 = Left THEN
6           IF s2 = Right or s3 = Right
7     5/5   THEN pos := Unknown ELSE pos := Left
8           END
9-      ELSIF s1 = Right THEN
10          IF s2 = Left or s3 = Left
11    5/5   THEN pos := Unknown ELSE pos := Right
12          END
13-     ELSE
14          IF s2 = Left
15-         THEN
16    2/2       IF s3 = Right THEN pos := Unknown
17    3/3       ELSE pos := s2
18              END
19          ELSIF s2 = Right
20-         THEN
21    2/2       IF s3 = Left THEN pos := Unknown
22    3/3       ELSE pos := s2 END
23-         ELSE
24    4/4       pos := s3
25          END
26      END
27  END
```

Fully proved

BLADE_i.imp | BLADE2_i.imp

```
1     IMPLEMENTATION BLADE_i
2     REFINES BLADE
3-    LOCAL_OPERATIONS
4         res <-- has_pos (pos, s1, s2, s3) =
5-        PRE
6             pos : POSITION &
7             s1 : POSITION &
8             s2 : POSITION &
9             s3 : POSITION
10-       THEN
11  1/1       res := bool (pos : {s1, s2, s3})
12        END
13-   OPERATIONS
14  1/1   res <-- has_pos (pos, s1, s2, s3) =
15-       BEGIN
16  1/1       res := bool (s1 = pos or s2 = pos or s3
      = pos)
17        END
18        ;
```

Other implementation component

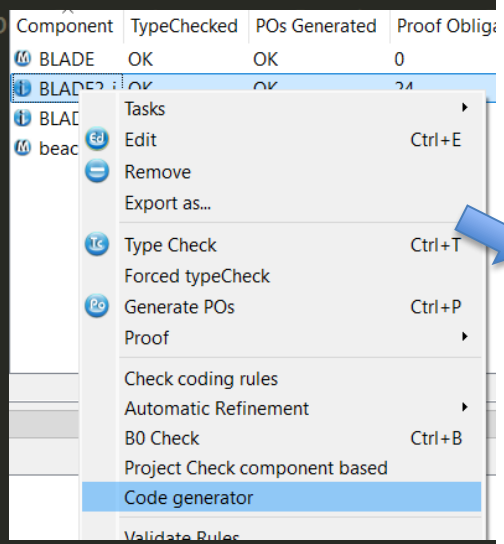
Local operation specification

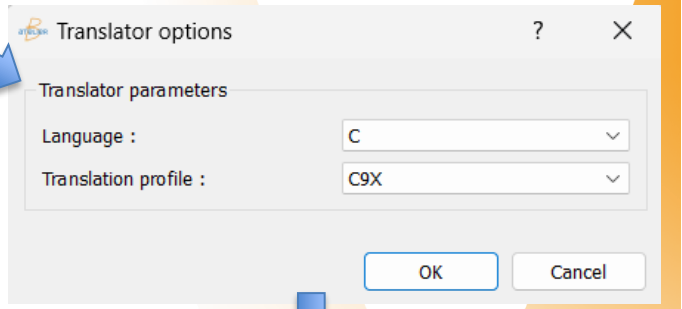
Local operation implementation

Operation implementation
Using local operation

```
19  2/2     pos <-- estimate (s1, s2, s3) =
20-         VAR
21              has_left, has_right
22-         IN
23              has_left <-- has_pos (Left, s1, s2, s3);
24              has_right <-- has_pos (Right, s1, s2, s3);
25-             IF
26                  has_left = TRUE & has_right = FALSE
27-             THEN
28                  pos := Left
29-             ELSIF
30                  has_left = FALSE & has_right = TRUE
31-             THEN
32                  pos := Right
33-             ELSE
34  2/2             pos := Unknown
35              END
36          END
37  END
```

```c
/* WARNING if type checker is not p
#include "BLADE.h"

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

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

/* Clause INITIALISATION */
void BLADE__INITIALISATION(void)
{

}

/* Clause OPERATIONS */

void BLADE__estimate(BLADE__POSITION s1, BLADE__POSITION s2,
{
    if(s1 == BLADE__Left)
    {
        if((s2 == BLADE__Right) ||
        (s3 == BLADE__Right))
        {
            (*pos) = BLADE__Unknown;
        }
```





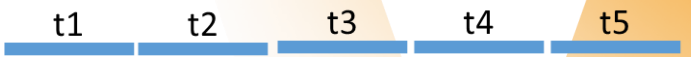
To access the code

- Select the project
- Click open folder
- Go to lang/C directory

DATA VALIDATION

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}

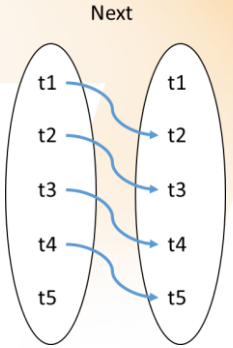
Let the function Next $\in$ TrackCircuit $\nrightarrow$ TrackCircuit

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))

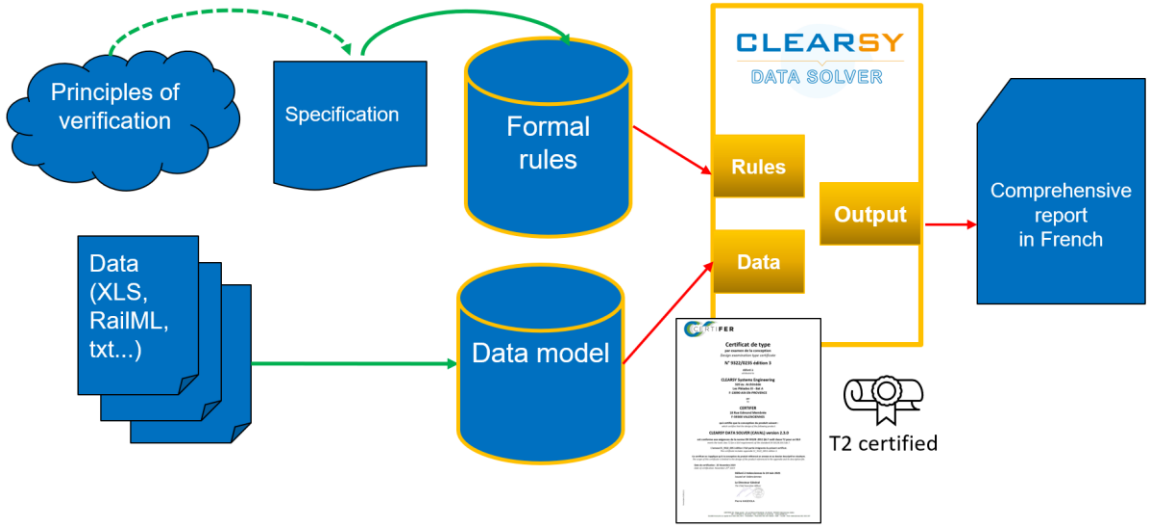


Formal Data Validation

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/

Achievements

2003

First tool to verify embedded topology data
For Certification

2012

First tool integrated into CBTC metro dev process

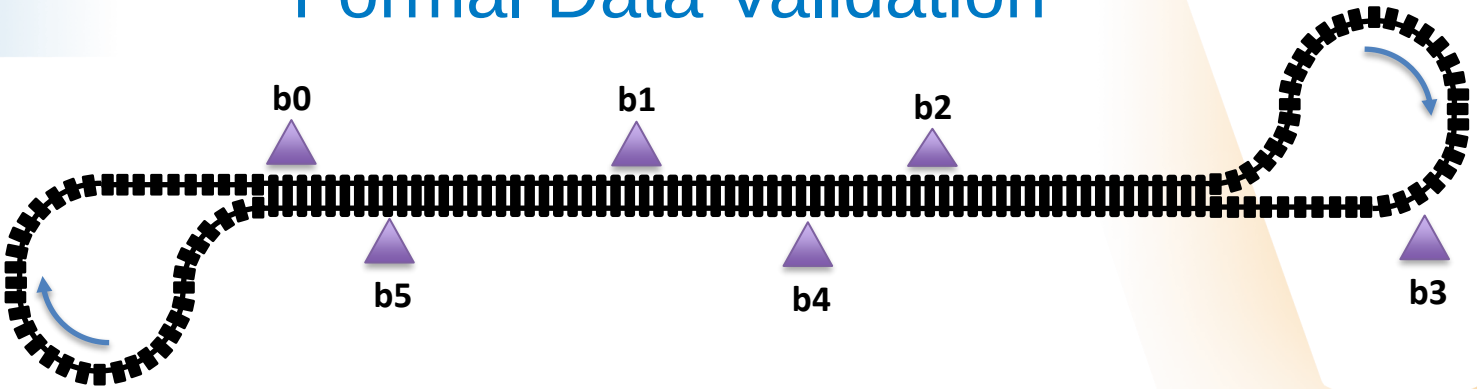
2018

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



- Beacons should be known from the onboard software
  - Location, attached information (speed limit, target distance)
  - Specific properties have to be checked
  - A formal model and a model-checker to obtain this

https://github.com/CLEARSY/ETMF2024/ Directory: DataValidation Model: beacons.mch

beacons.mch

```
1-  MACHINE beacons
2
3-  SETS
4       BEACONS = {b0, b1, b2, b3, b4, b5}
5-  CONSTANTS
6       nextB,
7       lenghtTC,
8       kpB,
9       lastB
10- PROPERTIES
11      nextB: BEACONS --> BEACONS &
12-     nextB = {
13          b0 |-> b1,
14          b1 |-> b2,
15          b2 |-> b3,
16          b3 |-> b4,
17          b4 |-> b5,
18          b5 |-> b0
19      } &
20      dom(nextB) \/ ran(nextB) = BEACONS &
```

Set of beacon ids, no order

Beacon successor, total function

All beacon ids used, no « dead code »

beacons.mch

```
22          lenghtTC: BEACONS --> INTEGER &
23-         lenghtTC = {
24              b0 |-> 1000,
25              b1 |-> 1000,
26              b2 |-> 2000,
27              b3 |-> 2000,
28              b4 |-> 1000,
29              b5 |-> 1000
30          } &
31
32          kpB : BEACONS --> INTEGER &
33-         !bc.(bc: BEACONS =>
34    2/2       (bc = b0 => kpB(bc) = 0) &
35    4/7       (not(bc=b0) => kpB(bc) = lenghtTC(nextB~(bc)) + kpB(nextB~(bc)))
36          ) &
37          lastB: BEACONS &
38          lastB = b5 &
39    5/5   !bc.(bc: BEACONS - {lastB} => kpB(bc) <= kpB(nextB(bc)))
40    END
```

TC length

Fully calculated with ProB

State Properties (20)

OK

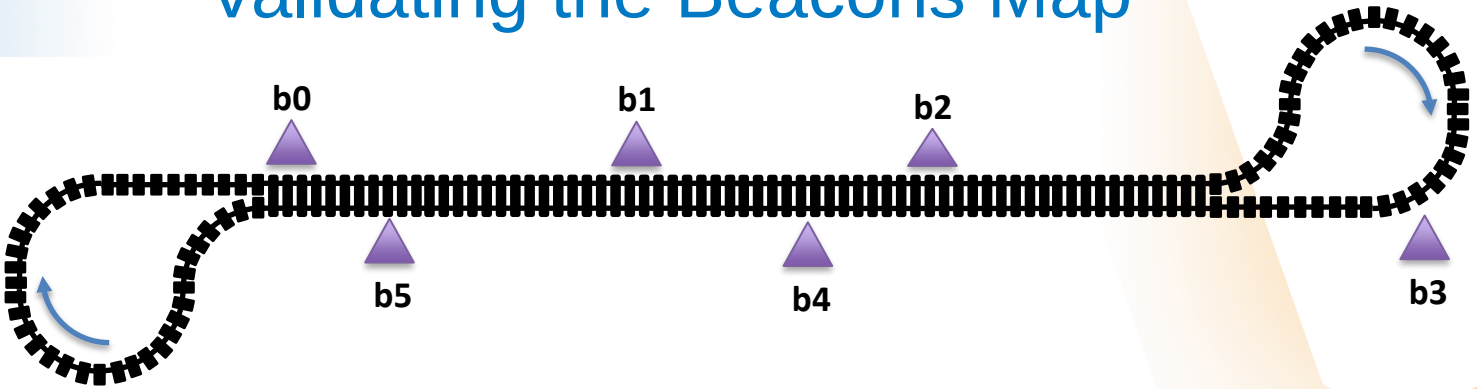
```
lenghtTC(b0) = 1000
lenghtTC(b1) = 1000
lenghtTC(b2) = 2000
lenghtTC(b3) = 2000
lenghtTC(b4) = 1000
lenghtTC(b5) = 1000
kpB(b0) = 0
kpB(b1) = 1000
kpB(b2) = 2000
kpB(b3) = 4000
kpB(b4) = 6000
kpB(b5) = 7000
lastB = b5
```

Calculation of TC position

Verification of location increasing with successors

Not fully proved With Atelier B

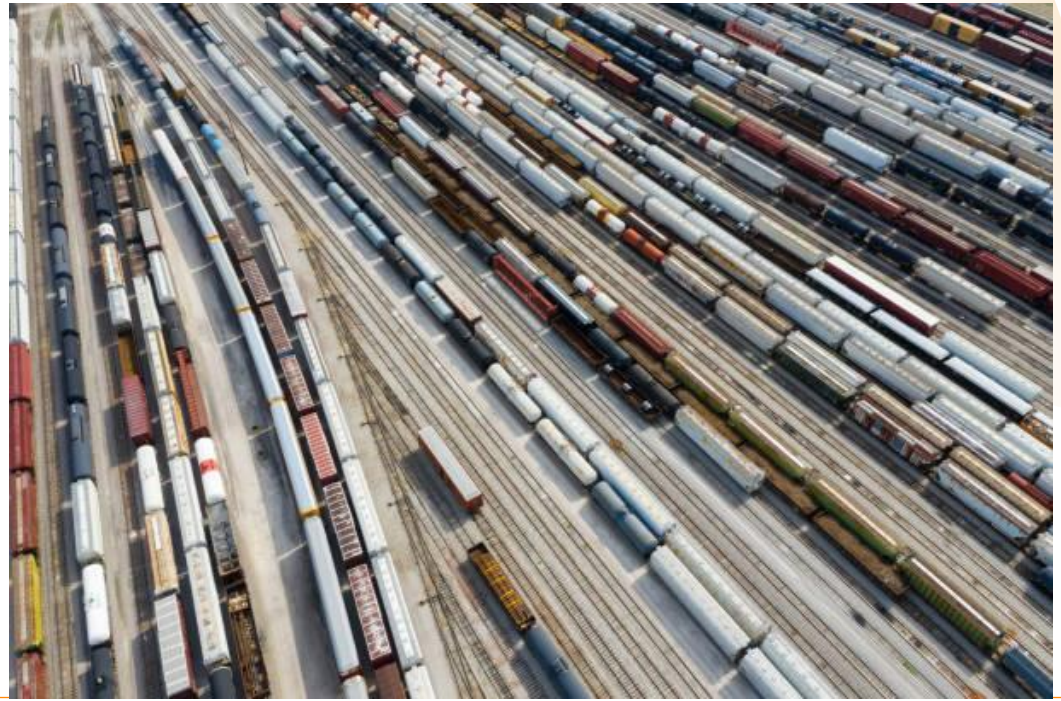
Validating the Beacons Map



- This is not all !
- We have validated an Excel file (or similar)
- We also need to verify the real distances on the track (manually, with a robot, a drone)


clearSY
Safety Solutions Designer

READY FOR REAL WORLD?



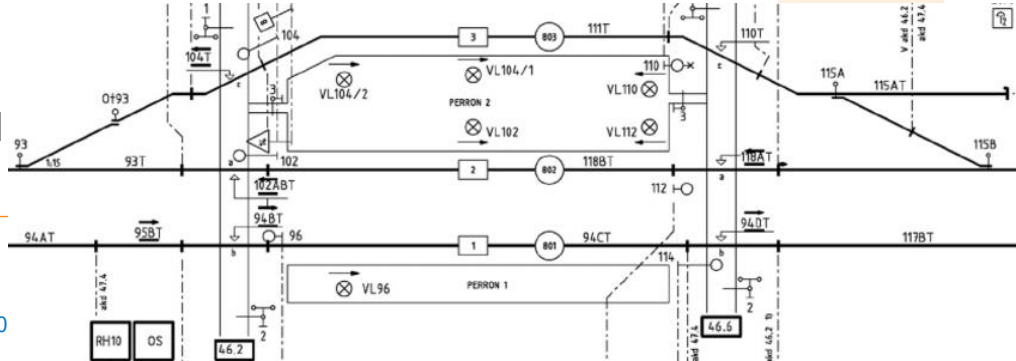
Towards the limits …

« 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))

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.

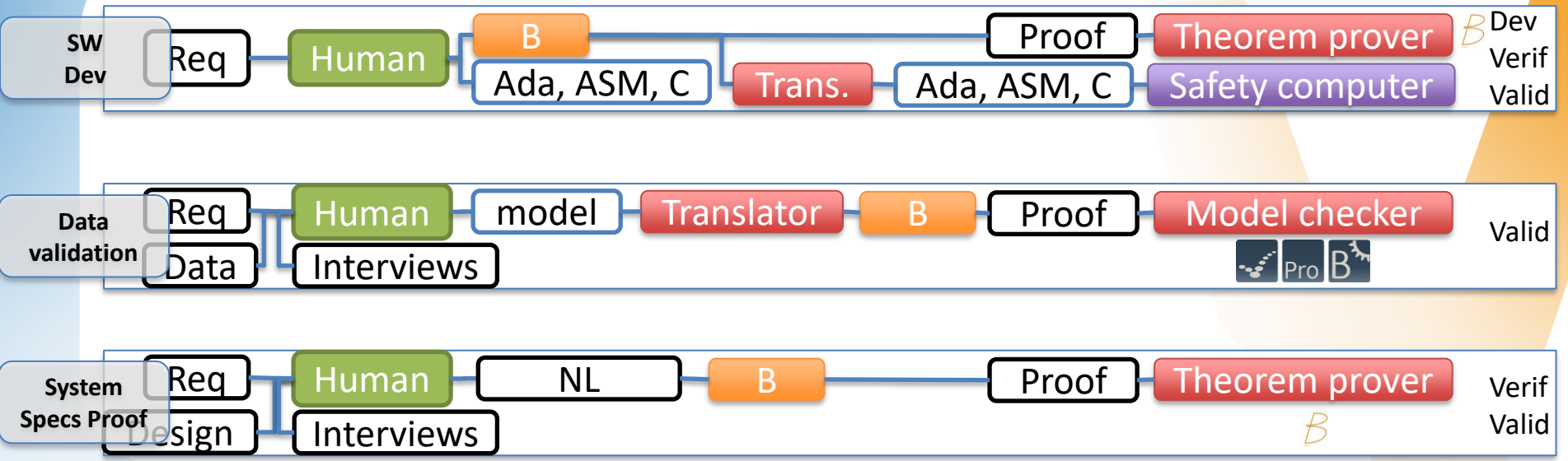
- the elevationDifference.elevationEnd of the part1 and elevationDifference.elevationStart of the part2 (reference to the above figure) are equal to elevationStart +2*radius*sin(Length1/ (2*radius))*sin(gradientStart +Length1/ (2*radius)).

- 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.

Formal Methods in Action



Conclusion

- FM can complement engineering when designing a “system”
- **Safety is by design**
  - There is no ultimate design
  - Only consequences of decisions and a reasoning
- Particularly useful when dealing with cyber-physical systems
- **Handling industry strength systems is a challenge, even with FM**

CLEARSY
Safety Solutions Designer

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