Formal Data Validation

Tutorial

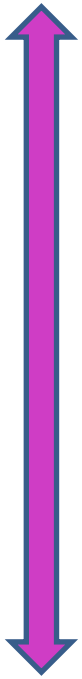
June 3rd 2014 2:00pm : 5:30pm

Organizers:

- Thierry Lecomte (ClearSy)
- Dr Michael Leuschel (University Dusseldorf)

Agenda

2:00



05:30

- Introduction
- Formal Data Validation in a Nutshell
- Practical Examples
- The approach:
  - B mathematical language
  - B tools
- Case studies
- Conclusion

Introduction

- **Data validation definition**

  "*The process of ensuring that a program operates on clean, correct and useful data.*" (wikipedia)

  "*An Excel feature that you can use to define restrictions on what data can or should be entered in a cell.*" (Microsoft)

Trivial when related to typing, range verification or simple conditions.
Human-based or software non-formal verification.

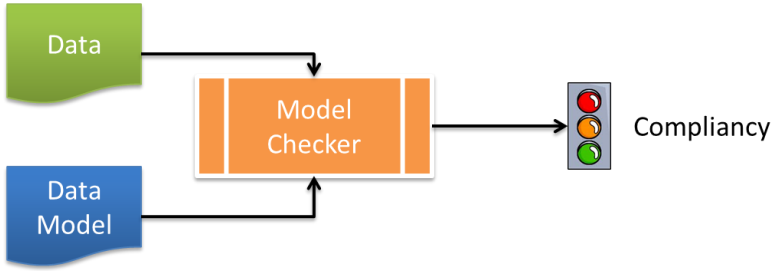
Ex: Excel macros

```
=AND(D1=0,D2<40000)
=ISTEXT(B2)
=AND(LEFT(B5, 3) ="ID-",LEN(B5) > 9)
=COUNTIF($A$1:$A$20,A1)=1
```

Introduction

- **Formal Data validation definition**

  Data validation performed with formal means i.e. checking that data comply with a formal data model.



- **Rationale**
  - Safety critical software require safety critical parameters
  - Intricate data structure can't be safely checked visually

  Ex:
  - Alstom : B/ProB for metro line description (see "Practical Examples")
  - Airbus : Uml/OCL for embedded software

FDV In a Nutshell

Formal Data validation (for this tutorial) ≡

Automatic check of large data sets against properties

100,000+ raw data chunks

	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

Expressed using B mathematical language

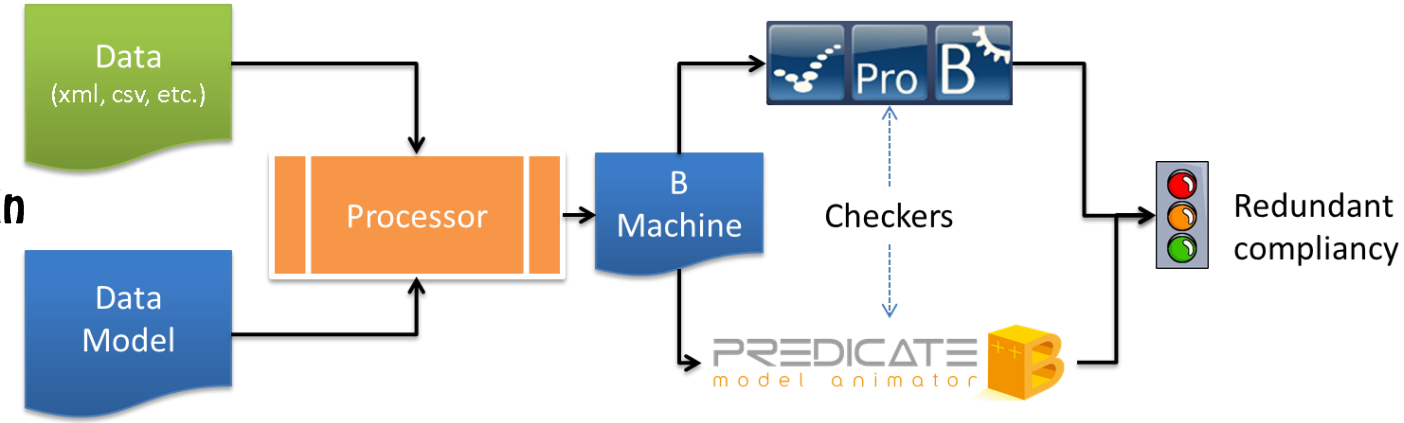
Are they
- Consistent ?
- Correct ?
- Safe ?

Model-checker (no human in the loop)

FDV In a Nutshell

- **Why B / ProB ?**
  - Experimented with success during DEPLOY project for Siemens safety critical software (deliverable D41):
    - San Juan metro: 1 MM human verification vs 1-minute computation
    - Paris Line 1 metro: 2024 assertions checked in 5 minutes
    - Sao Paulo line 4: 2500+ assertions checked in a day
    - Barcelona line 9, Paris Charles de Gaulle airport shuttle
  - Extensively used for metro lines verification

Ex: redundant tool chain



Practical Examples

- Metro lines topology data validation
- Grafcet compiler binary file verification

Practical Examples

Metro lines topology data validation

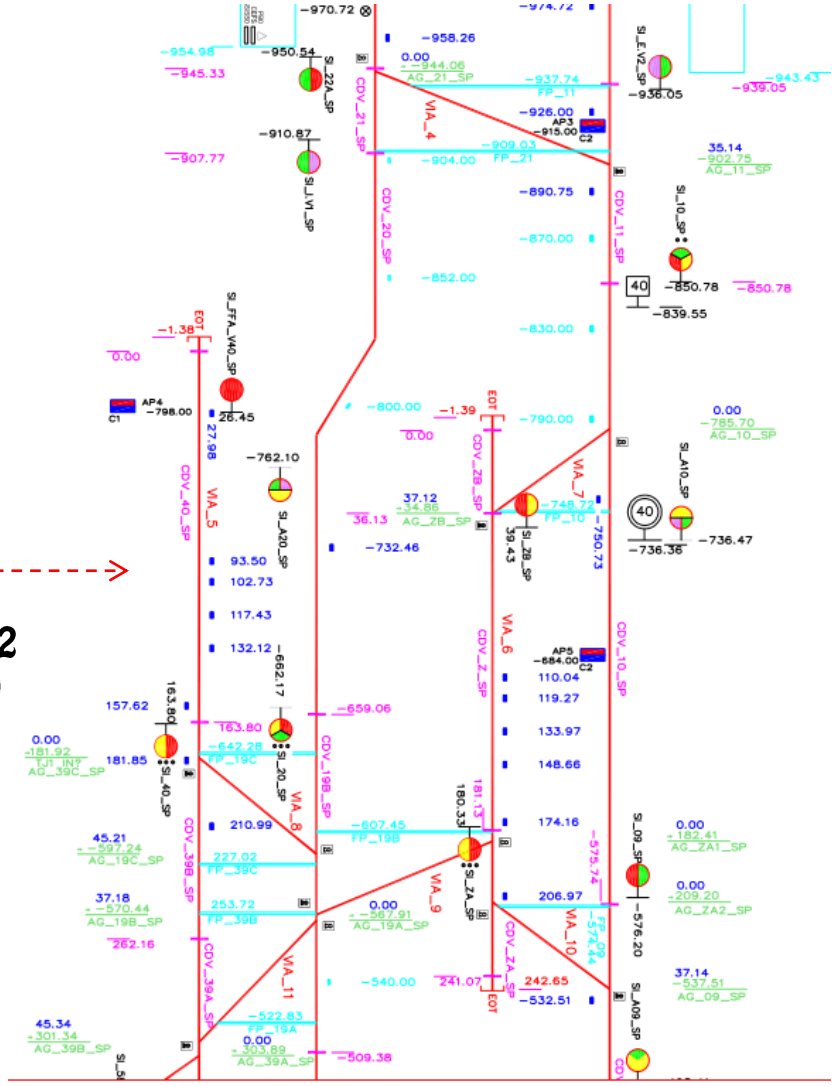
Data sets ≡

Data describing the topology of the tracks
Software and equipment parameters

- **Addressing plan:** networked equipment, IP addresses
- **Scheme plan :** 100+ similar drawings
- **System Data :** around 50,000 data (tables) for Mexico L12 (track circuits, signals, switches, links etc.)

Data model ≡

Consistency (track circuits connected),
signalling rules (signals protecting switches)

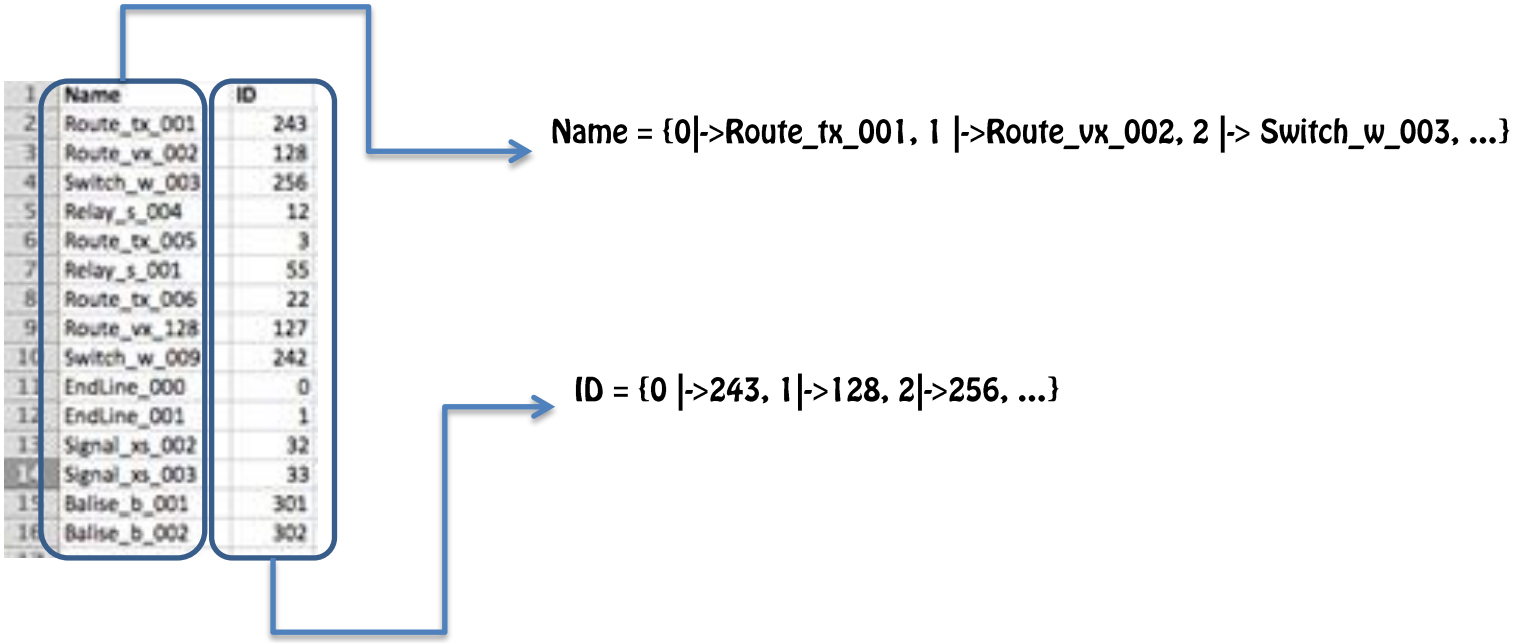


Practical Examples

Data sets ≡

Raw data as inputs

- csv files, every csv column is a **constant** in the B model (total function)



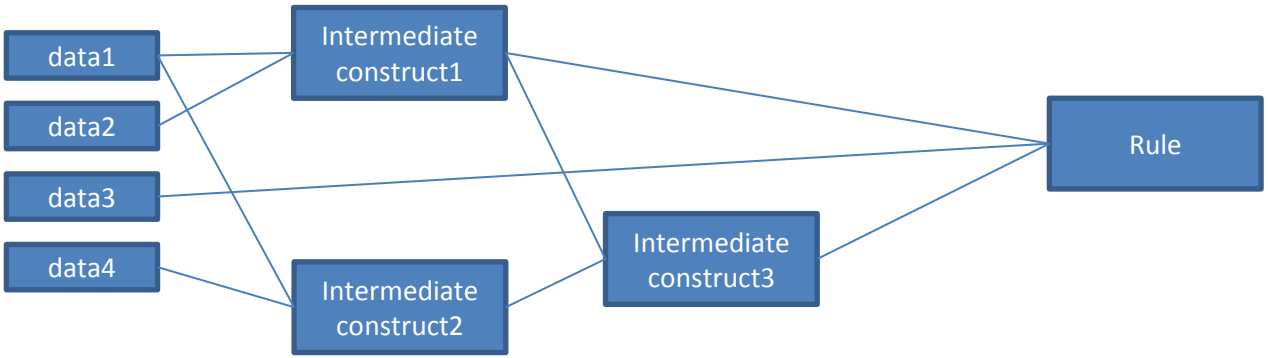
- data are not preprocessed: everything is modelled in B mathematical language
- Supported types: BOOL, INT , STRING, seq(INT), seq(STRING).

Practical Examples

Data model ≡

Relationships between the data

- Properties expressed with B mathematical language and decorated with substitution-like syntax
- Use of intermediate constructs (sets definition) to factorize development



- Simple specification and detection of counter-examples

Ex:
- « grey tracks circuits are track circuits that are associated to a grey signal » (intermediate construct)
- « a grey track, made of track circuits, contains at least one grey track circuit » (rule)

Practical Examples

```
RULE DB_GENERAL.3
COUNTEREXAMPLE
   the name %1 is the name of an equipment of type ZC but is not in table ZC
ANY
   name1, ind2
TYPE
   STRING, INT
WHERE
   ind2 : dom(ATC_Equipments_Cap!Name) &
   ATC_Equipments_Cap!ATC_Equipment_Type(ind2) = "ZC" &
   ATC_Equipments_Cap!Name(ind2)=name1
EXPECTED
   #ind1.( ind1 : dom(ZCs_Cap!Name) & name1=ZCs_Cap!Name(ind1))
END
```

Rule name

Values to search for

Sheet name

Data name

Conditions to fulfill

If not fulfilled, counterexample is found and error message is displayed

execution

RULE NAME	STATUS	COUNTEREXAMPLES
Rule_DB_General	KO	2

COUNTEREXAMPLE_0
the name ZC_A is the name of an equipment of type ZC but is not in table ZC

COUNTEREXAMPLE_1
the name ZC_AB is in table ZC but is not the name of an equipment of type ZC

A rule can be made of several sequential searches for counterexamples

Practical Examples

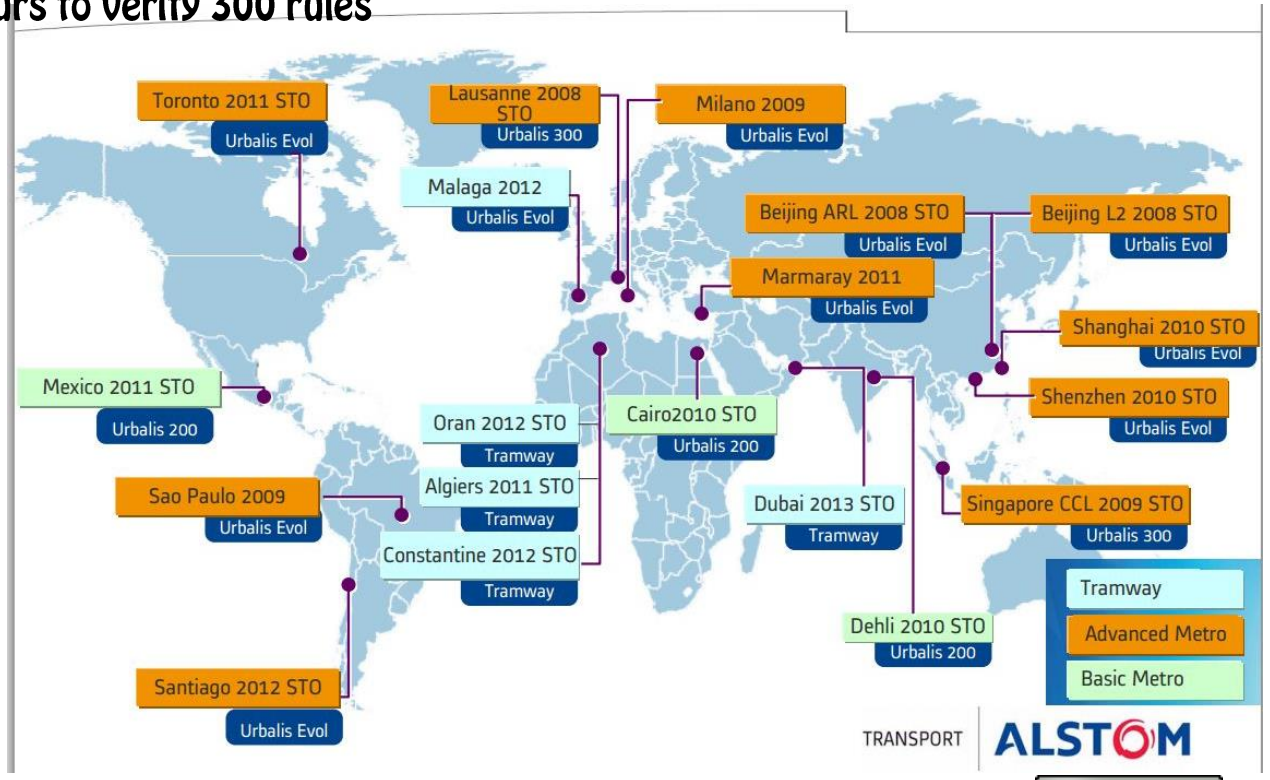
Rules to verify:

- 1.000 rules per metro line (generic / specific corpus)
- 150 intermediate constructs

Time schedule:

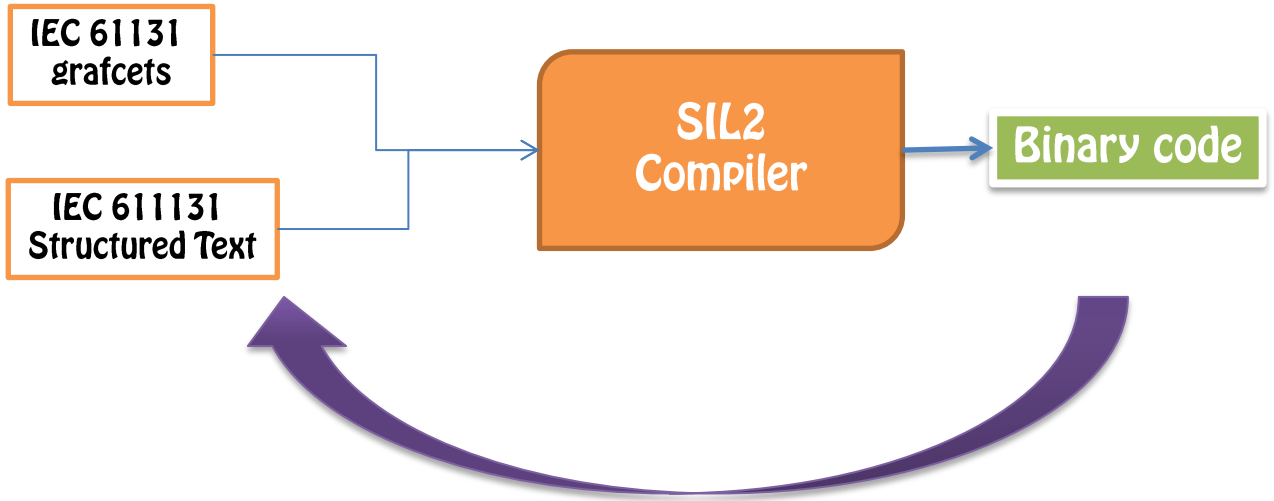
- Manual: 30 days to verify 300 rules
- Data validation process: few hours to verify 300 rules

Application to all these projects (+Panama) for each major release



Practical Examples

Grafcet compiler binary file verification



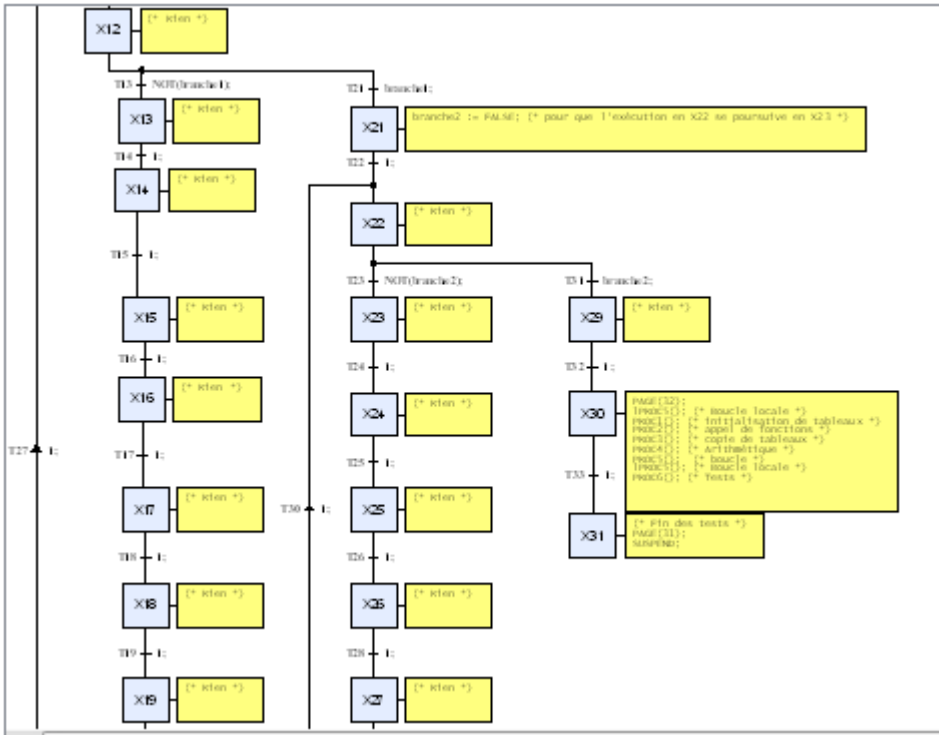
Check that binary code complies with input models

~80 properties identified related to 1200 variables and 800 kB code

- P03 Sub-grafcets called in the binary file should comply with sub-grafcets activated in input models

Practical Examples

IEC 61131 -3
grafcets



IEC 61131 -3
Structured Text

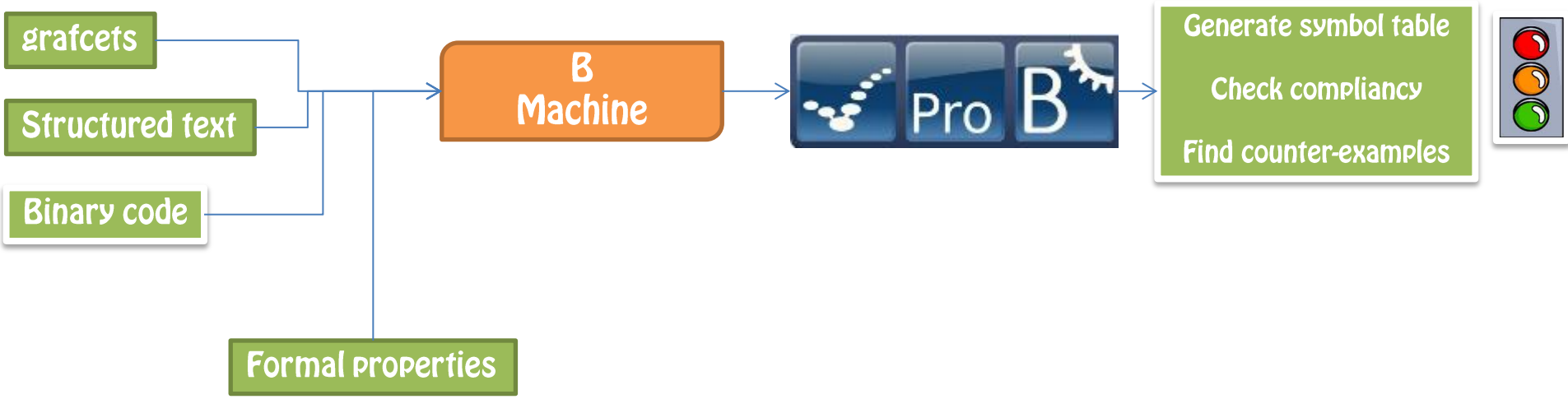
```
169-    FUNCTION FI001
170         gI1 := gcI1;
171         gtI1[10] := gI1;
172         gtI1[1..3] := [ 3 (gcI1)];
173         gtI1[1..3] := gtI1[11..13];
174         gI1 := (gI1 MOD 42) + 18 - (gI1 * 23) / gI1;
175         gI2 := -gI1;
176         gI2 := -gI2;
177     END_FUNCTION
```

Binary code

```
[0x001404] ADR_VL_VL_DR TRANSFERT 0x00d4 0x0000 0x82be
[0x00140c] ADR_DR_VL_DR TRANSFERT 0x82be 0x0000 0x8306
[0x001414] ADR_VL_VL_DR RMP_TABLE 0x0003 0x00d4 0x82f4
[0x00141c] ADR_VL_VL_DR TFR_TABLE 0x0003 0x8308 0x4004
[0x001424] ADR_DR_VL_DR MODULO 0x82be 0x002a 0x82ae
[0x00142c] ADR_DR_VL_DR ADDITION 0x82ae 0x0012 0x82ae
[0x001434] ADR_DR_VL_DR MULTIPLIE 0x82be 0x0017 0x82b0
[0x00143c] ADR_DR_DR_DR DIVISE 0x82b0 0x82be 0x82b0
[0x001444] ADR_DR_DR_DR SOUSTRAC 0x82ae 0x82b0 0x82be
[0x00144c] ADR_DR_VL_DR SOUSTRAC 0x82be 0x0000 0x82c0
[0x001454] ADR_DR_VL_DR SOUSTRAC 0x82c0 0x0000 0x82c0
[0x00145c] RETOUR 0x00
```

Practical Examples

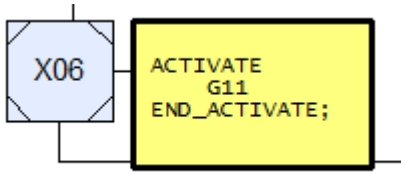
~30,000 lines of code
162 grafcets
1000+ steps
800 kB binary code



Practical Examples

- P03 Sub-grafcets called in the binary file should comply with sub-grafcets activated in input models

List grafcet activations (old models)



Build B model of activations

G7 = {main, g1, g2, g3, g4, …. }
next: G7 <-> G7
next = { …, g7 |-> g11, …}

List grafcet activations (binary)

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

Build B model of activations

ADR = {0x01, 0x13, 0x15, …}
suiv: ADR <-> ADR
suiv = { … , 0x10 |-> 0x15, …}

there exists a bijection bij that associates to a node of G7 a node of ADR such as children of both nodes match

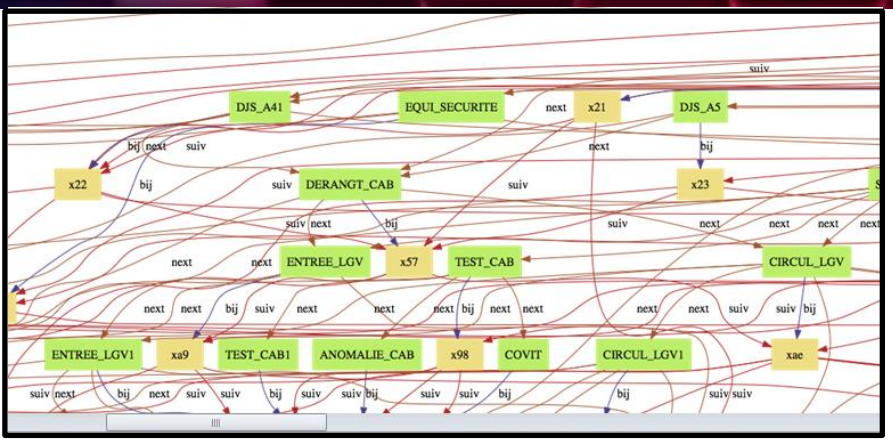
bij: G7 >->> ADR &!xx.(xx: G7 => bij[next[{xx}]] = suiv[bij[{xx}]])

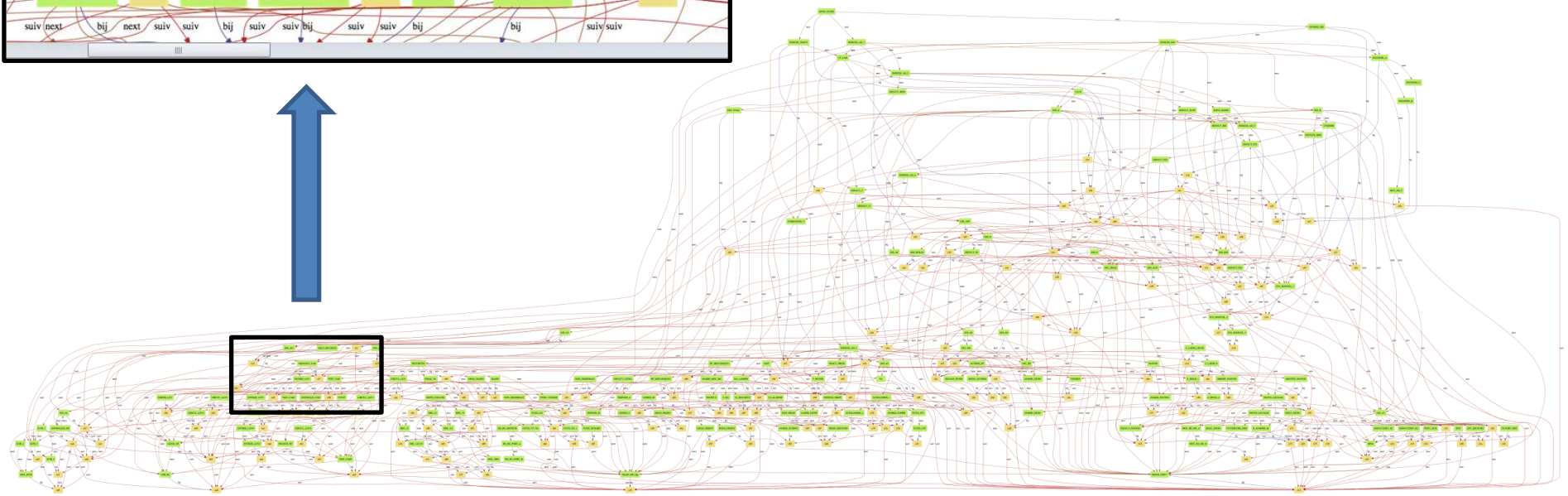
Practical Examples

there exists a bijection bij that associates to a node of G7 a node of ADR such as children of both nodes match (each node is given an address)

bij: G7 >->> ADR &!xx.(xx: G7 => bij[next[{xx}]] = suiv[bij[{xx}]])

Practical Examples

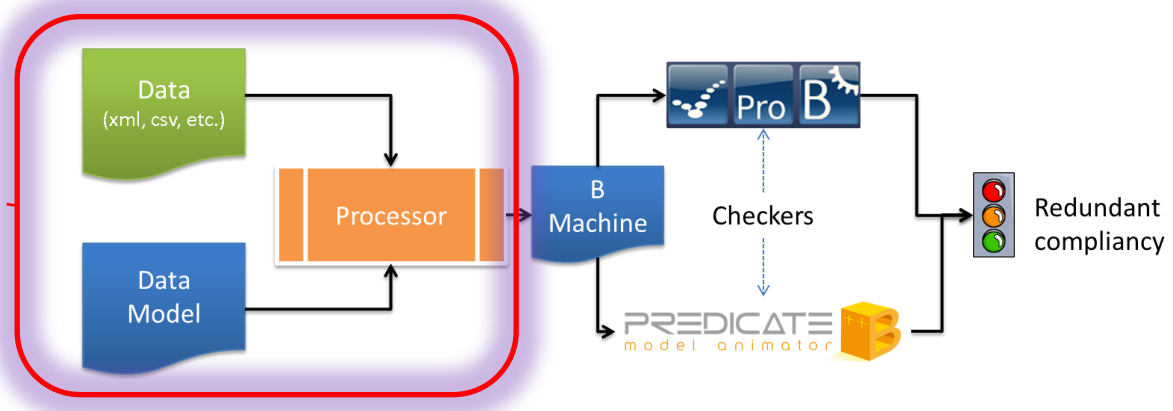




162! =
1229694218739449434110178928491750176572300599427169306620762521167814540117728965860988098467051531783599507442
9904709708273401807824365415928975695099566042246320538220924308010459938381430588227927174194100982189204709615
2931983263907734109259038720000000000000000000000000000000000000000

Approach: language & tools

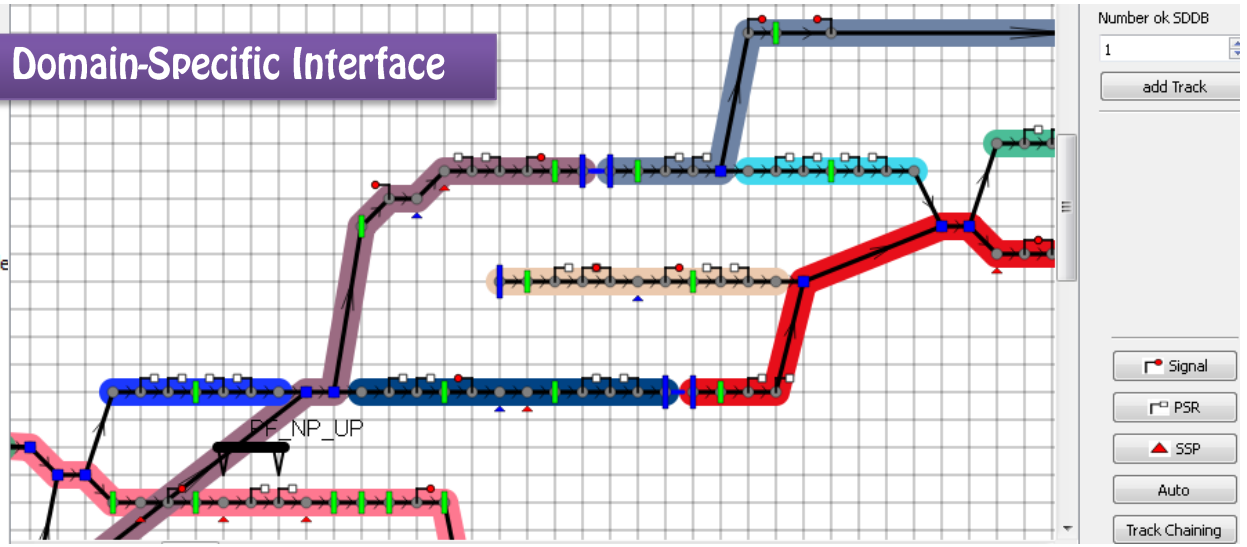
Pre-processing is domain specific
and is aimed at easing rule writing



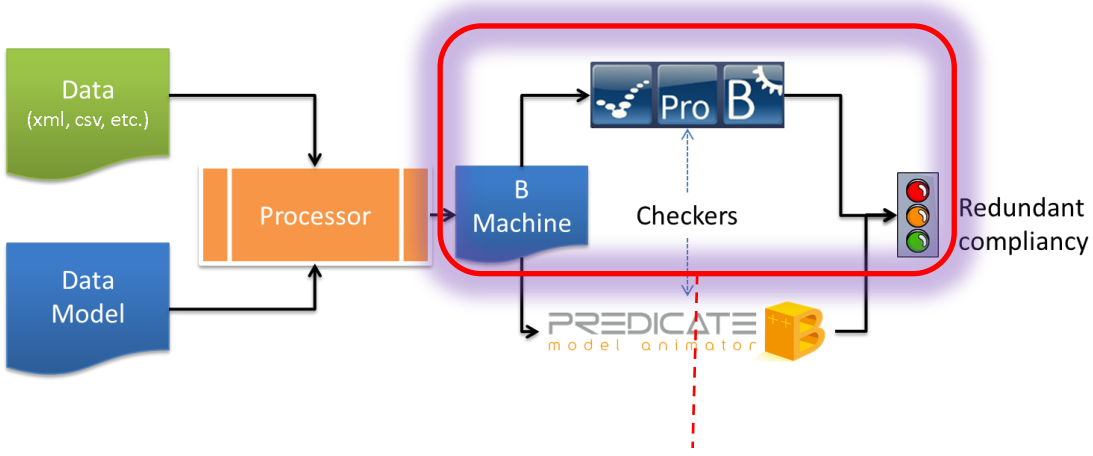
Example of Railways Domain-Specific Interface

```
Rule_DB_ATCEQUIP_0007
/* For a given project, there shall
COUNTEREXAMPLE
  %1 local OMAP found
ANY
  nbLocOMAP
TYPE
  INT
WHERE
  nbLocOMAP = card(ATC_Equipments_Cap!ATC_Equipment_Type|>{"Trackside
EXPECTED
  nbLocOMAP <= 10
END;

COUNTEREXAMPLE
  ZC %1 has %3 defined trackside OMAP
ANY
  urbalisSectorID, idZC, nb, nameZC
TYPE
  STRING, INT, INT, STRING
WHERE
  idZC : ATC_Equipments_Cap!ATC_Equipment_Type~[{"ZC"}] &
  urbalisSectorID = ATC_Equipments_Cap!Urbalis_Sector_ID(idZC) &
  nameZC = ATC_Equipments_Cap!Name(idZC) &
  nb = card(dom(ATC_Equipments_Cap!Urbalis_Sector_ID|>{urbalisSectorID}) /\ dom(ATC_Equipments_Cap!ATC_Equipment_Type|>{"Trackside OMAP"}))
EXPECTED
  nb = 1
END
```



Approach: language & tools



What is required:

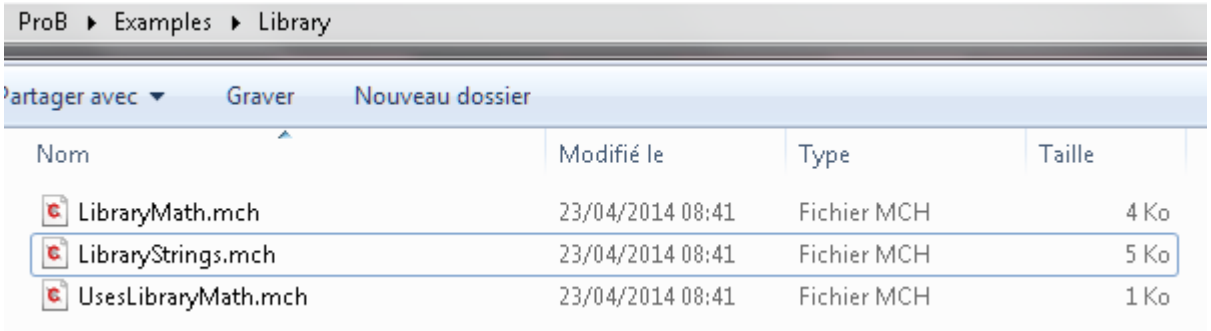
- ProB + your preferred text editor
- B mathematical language + basic substitutions
- Rule writing strategy

Approach: language & tools

http://nightly.cobra.cs.uni-duesseldorf.de/tcl/

ProB Nightly Build

Name	Last modified	Size	Description
Parent Directory		-	ProB Release Folder
ProB.linux.tar.gz	27-Mar-2014 20:05	24M	ProB Linux Tarball
ProB.linux32.tar.gz	31-May-2014 20:07	24M	ProB Release Folder
ProB.linux64.tar.gz	31-May-2014 20:09	25M	ProB Linux 64Bit Tarball
ProB.mac_os.10.6.8.x86_64.tar.gz	31-May-2014 20:04	26M	ProB MacOS X Tarball
ProB.windows.zip	31-May-2014 20:09	25M	ProB Release Folder



Strings and mathematic libraries

Approach: language & tools

B Mathematical Language

See http://www.tools.clearsy.com/resources/documents/
« B Language Reference Manual »
Symboles.pdf: ascii / math characters

aa = bb
not(aa=bb)
C1 & C2 or C3
P1 => P2
P1 <=> P2
!xx.(P(xx)) (universal)
#xx.(P(xx)) (existential)
aa /\ bb \/ cc (intersection, union)
{} (empty set)
aa <: bb (inclusion)
aa : AA (belongs)
aa : AA +-> BB (partial function)
aa : AA --> BB (total function)
aa : AA <-> BB (relation)
aa : AA >-> BB (injection)

aa: AA -->> BB (surjection)
aa : AA >->> BB (bijection)
rr[AA] (relational image)
rr~ (inverse function)
{aa} <| rr (domain restriction)
rr |> {bb} (range restriction)
AA – BB (set difference)
dom(rr) (domain)
ran(rr) (range)
rr;ss (composition)
AA*BB (cartesian product)

Approach: language & tools

B basic substitutions

See http://www.tools.clearsy.com/resources/documents/
« B Language Reference Manual »
Symboles.pdf: ascii / math characters

Var := value	xx := {0|->10, 1|->2} || yy := {} || zz := 1
Var := {xx | P(xx)}	aa:= {ab, cd | ab: dom(xx) & cd: INT}

```
SELECT conditions THEN
   substitution
END
```

```
SELECT zz > 1 THEN
   xx := {zz} <| xx
END
```

```
ANY variables WHERE
   conditions
THEN
   substitution
END
```

```
ANY bb, cc WHERE
   bb = xx(zz) &cc <: aa
THEN
   aa := cc || zz := bb
END
```

Approach: language & tools

Rule writing strategy

Data to verify are modelled as constant sequences

For each property to verify, a variable is created, initialized as empty then populated with elements that verify Property1 and not(Property2)
The variable finally contains all counterexamples

```
ANY VariableList
TYPE
  TypeList
WHERE Property1
EXPECTED
  Property2
END
```

Given that data = {“abc”,”defg”,”hi”,”jklm”}

Check that data doesn’t contain strings with more than 4 characters

```
CONSTANTS
   data
PROPERTIES
   data : seq(STRING) & data = [“abc”,”defg”,”hi”,”jklm”]
VARIABLES
   prop_1
INVARIANT
   prop_1 : POW(STRING*INT)
INITIALISATION
   prop_1 := {}
OPERATIONS
   compute_prop_1 =
      prop_1 := {dd,nn | dd: ran(data)  & nn=length(dd) &  not(nn<=4) }
END
```

Approach: language & tools

Lists are modelled as sequences
("abc","defg","hi","jklm") = {0|->"abc",1|->"defg", 2|->"hi", 3|->"jklm"}

```
CONSTANTS
   data
PROPERTIES
   data : seq(STRING) & data = ["abc","defg","hi","jklm"]
VARIABLES
   prop_1
INVARIANT
   prop_1 : POW(STRING*INT)
INITIALISATION
   prop_1 := {}
OPERATIONS
   compute_prop_1 =
      prop_1 := {dd,nn | dd: ran(data)  & nn=length(dd) &  not(nn<=4) }
END
```

ProB library function
See librarystrings.mch

Approach: language & tools

Machine name and link with ProB library machine

```
MACHINE
  Rule_prop_1
SEES LibraryStrings
CONSTANTS
  data
PROPERTIES
  data : seq(STRING) & data = [“abc”,”defg”,”hi”,”jklm”]
VARIABLES
  prop_1
INVARIANT
  prop_1 : POW(STRING*INT)
INITIALISATION
  prop_1 := {}
OPERATIONS
  compute_prop_1 =
    prop_1 := {dd,nn | dd: ran(data)  & nn=length(dd) &  not(nn<=4)
};
  rule_1 =
   ANY dd, nn WHERE
     dd |-> nn : prop_1
   THEN
     prop_1 := prop_1 – {dd |-> nn  }
   END
END
```

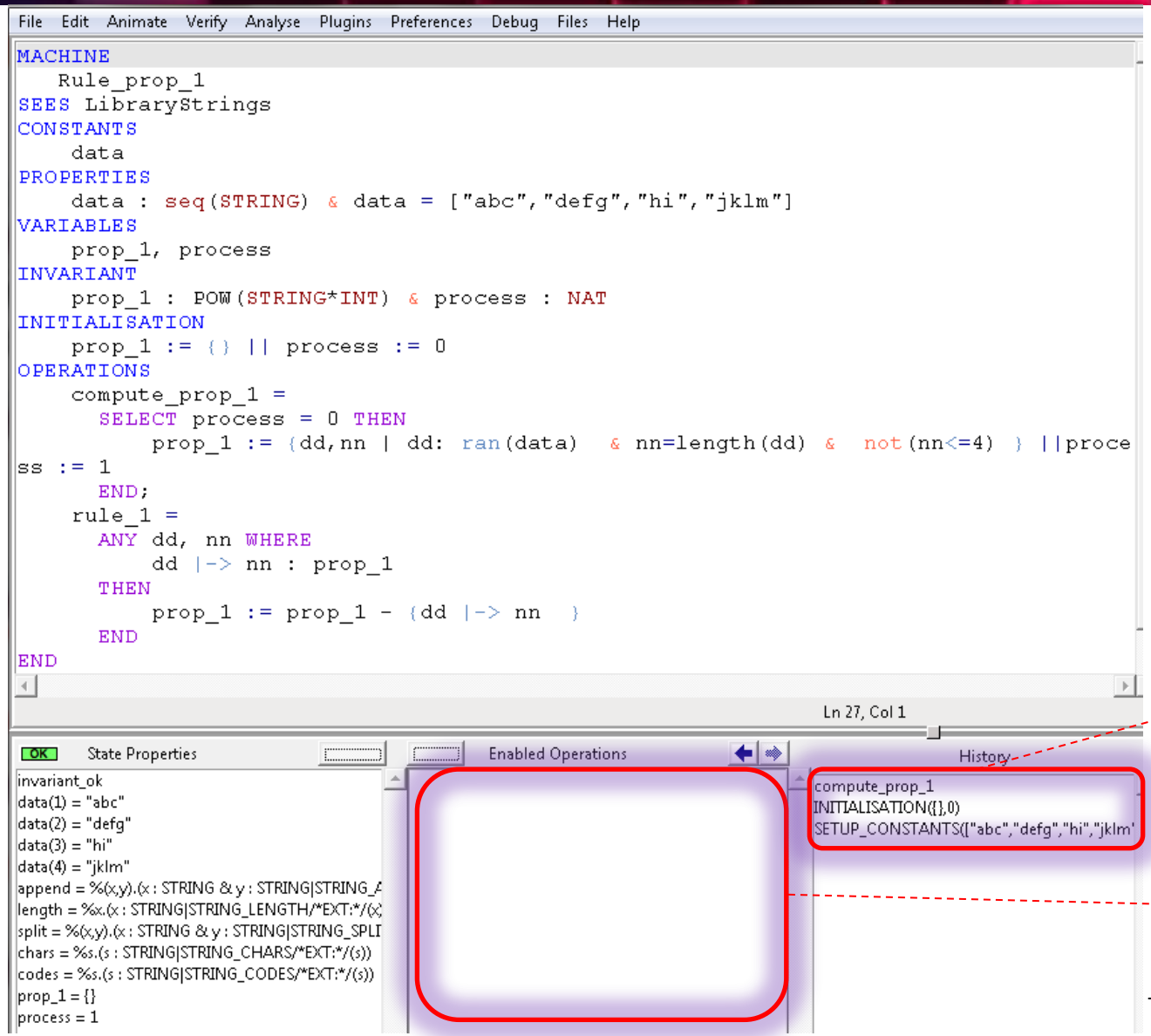
When animating with ProB, rule_1 allows to get all counterexamples, one by one

Approach: language & tools

Addition of a sequencing variable to execute compute_prop_1 only once

```
MACHINE
  Rule_prop_1
SEES LibraryStrings
CONSTANTS
   data
PROPERTIES
   data : seq(STRING) & data = [“abc”,”defg”,”hi”,”jklm”]
VARIABLES
   prop_1, process
INVARIANT
   prop_1 : POW(STRING*INT) & process : NAT
INITIALISATION
   prop_1 := {} ||process := 0
OPERATIONS
   compute_prop_1 =
     SELECT process = 0 THEN
        prop_1 := {dd,nn | dd: ran(data)  & nn=length(dd) &  not(nn<=4) } || process :=1
     END;
   rule_1 =
    ANY dd, nn WHERE
       dd |-> nn : prop_1
    THEN
       prop_1 := prop_1 – {dd |-> nn  }
    END
END
```

Approach: language & tools



Animating the machine with ProB

Compute_prop_1 operation triggered once

Rule_1 operation can't be triggered: no counterexample -> property verified

Approach: language & tools

Sequencing verifications

In case of several verifications to perform sequentially, sequencing variable helps to sort operation execution

```
VARIABLES
   process, prop_1, prop_2, prop_3
INVARIANT
   process : NAT &
   prop_1 : POW(STRING) & prop_2: POW(INT) & prop_3: POW(INT)
INITIALISATION
   process := 0 || prop_1 := {} || prop_2 := {} || prop_3 := {}
OPERATIONS
   compute_prop_1 =
      SELECT process =0 THEN
         prop_1 := { ... } || process := 1
      END;
   compute_prop_2 =
      SELECT process =1 THEN
         prop_2 := { ... } || process := 2
      END;
   compute_prop_3=
      SELECT process =2 THEN
         prop_3 := { ... } || process := 3
      END;
   rule_1 = ANY dd WHERE dd : prop_1   THEN ... END;
   rule_2 = ANY dd WHERE dd : prop_2   THEN ... END;
   rule_3 = ANY dd WHERE dd : prop_3   THEN ... END;
END
```

Case-studies

Case-studies

Ex 1:

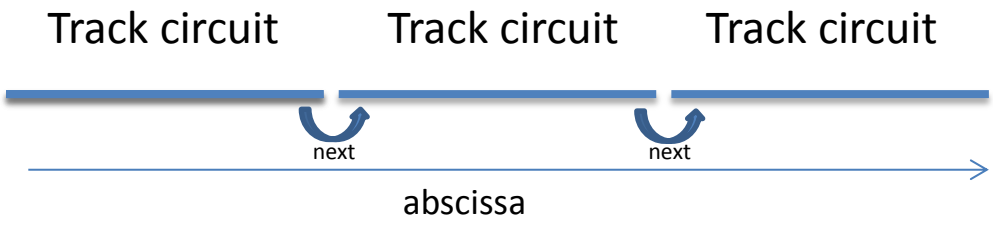
- Type in the machine
- Animate it with ProB
- Verify that rule_1 is never activated
- Introduce an error in data (a string with more than 5 characters)
- Verify that the case is found by ProB

```
MACHINE     Rule_prop_1
SEES        LibraryStrings
CONSTANTS   data
PROPERTIES  data : seq(STRING) & data = ["abc","defg","hi","jklm"]
VARIABLES   prop_1, process
INVARIANT
  prop_1 : POW(STRING*INT) & process : NAT
INITIALISATION
  prop_1 := {} || process := 0
OPERATIONS
  compute_prop_1 =
   SELECT process = 0 THEN
      prop_1 := {dd,nn | dd: ran(data)  & nn=length(dd) &  not(nn<=4) } ||process := 1
   END;
  rule_1 =
   ANY dd, nn WHERE
      dd |-> nn : prop_1
   THEN
      prop_1 := prop_1 - {dd |-> nn  }
   END
END
```

Case-studies

Ex 2:

- Type in the machine
- Animate it with ProB
- Verify that rule_1 is never activated
- Introduce an error in data Verify that the case is found by ProB



TC n°	0	1	2	3	4	5	6	7
TC name	tc1	tc2	tc3	tc4	tc5	tc6	tc7	tc8
TC abs	0	10	250	320	400	600	700	800
TC next	tc2	tc3	tc4	tc5	tc6	tc7	tc8	tc8

Property:

a track circuit next to other one should have increasing abscissa

Case-studies

```
MACHINE
  Rule_prop_2
SEES LibraryStrings
CONSTANTS
   tc, nxt, kp
PROPERTIES
   tc : seq(STRING) & tc = ["tc1", "tc2", "tc3", "tc4", "tc5", "tc6", "tc7", "tc8"] &
   kp : seq(INT) & kp = [0, 10, 250, 320, 400, 600, 700, 800] &
   nxt : seq(STRING) & nxt = ["tc2", "tc3", "tc4", "tc5", "tc6", "tc7", "tc8", "tc8"]
VARIABLES
   prop_2, process
INVARIANT
   prop_2 : POW(INT*INT*INT*INT) & process : NAT
INITIALISATION
   prop_2 := {} || process := 0
OPERATIONS
   compute_prop_2 =
    SELECT process = 0 THEN
       prop_2 := {i1, i2, k1, k2 | i1 : dom(tc) & i2 : dom(tc) & i1 < i2 & k1=kp(i1) &
              k2=kp(i2) & not(k1<=k2) } ||
       process := 1
    END;
   rule_2 =
    ANY i1, i2, k1, k2 WHERE
       i1 |-> i2 |-> k1 |-> k2 : prop_2
    THEN
       prop_2 := prop_2 - {i1 |-> i2 |-> k1 |-> k2}
    END
END
```

Case-studies

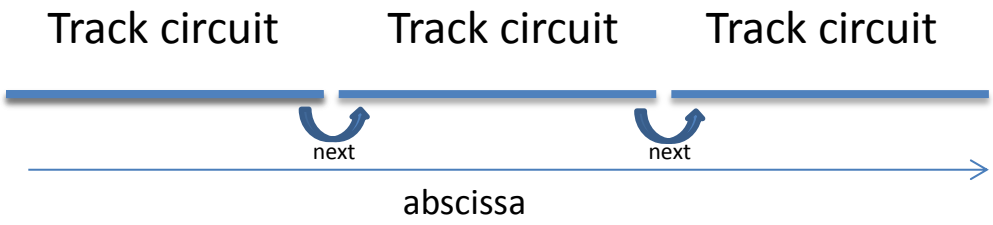
```
MACHINE
  Rule_prop_2b
SEES LibraryStrings
CONSTANTS
   tc, nxt, kp
PROPERTIES
   tc : seq(STRING) & tc = ["tc1", "tc2", "tc3", "tc4", "tc5", "tc6", "tc7", "tc8"] &
   kp : seq(INT) & kp = [0, 10, 250, 320, 400, 600, 700, 800] &
   nxt : seq(STRING) & nxt = ["tc2", "tc3", "tc4", "tc5", "tc6", "tc7", "tc8", "tc8"]
VARIABLES
   prop_2, process
INVARIANT
   prop_2 : POW(STRING*STRING*INT*INT) & process : NAT
INITIALISATION
   prop_2 := {} || process := 0
OPERATIONS
   compute_prop_2 =
    SELECT process = 0 THEN
       prop_2 := {n1, n2, k1, k2| n2 : ran(tc) & n1 : tc[nxt~[{n2}]] &
                                   k1 = kp(tc~(n1)) &  k2 = kp(tc~(n2))  & not(k1 <=k2)} ||
       process := 1
    END;
   rule_2 =
    ANY n1, n2, k1, k2 WHERE
       n1 |-> n2 |-> k1 |-> k2 : prop_2
    THEN
       prop_2 := prop_2 - {n1 |-> n2 |-> k1 |-> k2}
    END
END
END
```

```
prop_2 := {n1, n2, k1, k2|
    n2 : ran(tc) &
    n1 : tc[nxt~[{n2}]] &
    k1 = kp(tc~(n1)) &
    k2 = kp(tc~(n2))  &
    not(k1 <=k2)
    }
```

Case-studies

Ex 3:

- Type in the machine
- Animate it with ProB
- Verify that rule_1 is never activated
- Introduce an error in data Verify that the case is found by ProB



TC n°	0	1	2	3	4	5	6	7
TC name	tc1	tc2	tc3	tc4	tc5	tc6	tc7	tc8
TC abs	0	10	250	320	400	600	700	800
TC next	tc2	tc3 tc5	tc4	tc5	tc6 tc8	tc7	tc8	tc8

Property:

a track circuit next to other one should have increasing abscissa

Case-studies

```
MACHINE
  Rule_prop_3
SEES LibraryStrings
CONSTANTS
  tc, nxt, kp
PROPERTIES
  tc : seq(STRING) & tc = ["tc1", "tc2", "tc3", "tc4", "tc5", "tc6", "tc7", "tc8"] &
  kp : seq(INT) & kp = [0, 10, 250, 320, 400, 600, 700, 800] &
  nxt : seq(seq(STRING)) & nxt = [["tc2"], ["tc3", "tc5"], ["tc4"], ["tc5"], ["tc6", "tc8"], ["tc7"], ["tc8"], ["tc8"]]
VARIABLES
  prop_3, process
INVARIANT
  prop_3 : POW(STRING*STRING*INT*INT) & process : NAT
INITIALISATION
  prop_3 := {} || process := 0
OPERATIONS
  compute_prop_3 =
   SELECT process = 0 THEN
      prop_3 := {n1, n2, k1, k2 |  n1 : ran(tc) & n2: ran(nxt(tc~(n1))) & k1 = kp(tc~(n1)) & k2 = kp(tc~(n2))
& not(k1 <= k2)} ||
      process := 1
   END
   ;
  rule_3 =
   ANY n1, n2, k1, k2 WHERE
      n1 |-> n2 |-> k1 |-> k2 : prop_3
   THEN
      prop_3 := prop_3 - {n1 |-> n2 |-> k1 |-> k2}
   END
END
```

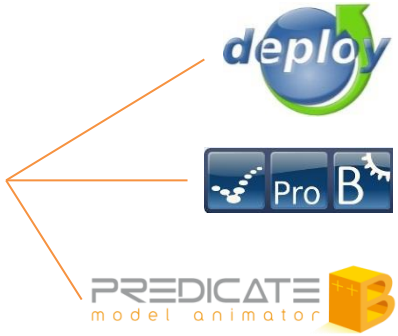
Conclusion

Data validation & data generation able to deal with industrial problems

- Data validation time divided by 10 at least
- Automation slightly improves the level of confidence
- It's "only model checking" but it does the job
- Industry loves push-button approach

Technology is mature

- Several R&D projects to assess and improve tools and methods
- Daily production on worldwide applications not restricted to B applications
  - Proprietary tools
  - Atelier B 4.1+ integrates data validation projects



THANK YOU FOR YOUR ATTENTION