Formal Verification in the Industry
≡ ClearSy
  • A company with the formal gene

≡ Verification in formal developments
  • A formal method: B
  • Workflow
  • Example

≡ An industrial theorem prover

≡ Data validation

≡ Integration of new formal verification technologies
  • iapa: interface to automatic proof agents
  • the drudges of the theorem prover or “teaching an old dog new tricks”
French SME funded in 2001
- ≈100 staff
- ≈10M€ income
- Automotive, Railway, Energy, Defense, Microelectronics
- Paris, Lyon, Aix-en-Provence, Strasbourg

Software
- Tools: Atelier B, Test runner
- Safety-critical: CBTC
- Data validation

Hardware
- SIL4 relay
- Axle counter

Systems
- SATURN: SIL0,SIL2,SIL4 I/O network
- DIL: In-gap individual detection

Services
- Safety analysis, safety demonstration
- Support for certification
B method in a nutshell

≡ A method to design software components

≡ Functional specification first
   Initial states
   Allowed state changes
   Safe states
   Formal verification: all reachable states are safe, well-definedness

≡ Stepwise refinements up to implementable realization
   Formal verification: refinements comply with specification, well-definedness, termination

≡ Poster child: Paris metro line 14 (110kloc B, 86kloc Ada)

≡ Tool support: Atelier-B (25 year old)
≡ Spin offs: Event B, data validation
Repeat until all POs proved & all PRs proved.

Repeat until all POs proved & all PRs proved & all MPs validated.

10-100s components. \( \approx 85\% \)
Safety Critical Railway Applications

- Reduce intervals between trains (from 120s to 90s / 75s)
  - Passive security not sufficient (power off)
  - Active security is required (trains have to brake when emergency)

- B instead of program proof for
  - Embedded software (Automatic Train Pilot)
  - Localization: graph-based algorithms
  - Energy control: integer arithmetic (braking curve)
  - Emergency braking: Boolean predicates
  - Trackside software (Interlocking)

- 25% automatic metros in the world
Safety Critical Railway Applications

Top level implementation

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

Metrics

- 233 machines, 50 kloc
- 46 refinements, 6 kloc
- 213 implementations, 45 kloc
- 23 000 proof obligations
- 3 000 added user rules

Platform Screen Doors

- Top component: 12 variables 14 operations
- 10 components, 2 kloc specification, 2kloc implementation
- Connection with I/O through basic machines
Safety Critical Railway Applications

Biggest function: Localization

*where is the train?*

Post-condition of one operation:

*variables become such as …*

Modern Automatic Train Protection Software
(2015)
Safety Critical Railway Applications

Proof rules:
- THEORY language (ako PROLOG)
- Pattern-matching
- Backward, forward, rewrite rules
- Added at project level or component level
- Partly reused from project

Specific tool & GUI to validate user added rules (maintenance version)
- Translation to predicates
- Predicate prover
- Peer review
- Report
An industrial theorem prover
≈ Rule-based Proof Engine
• Predicates are decomposed and simplified
• Hypotheses are generated and simplified
• Set of rules includes Alstom and Matra rules

≈ State of the Art
• Invented by a signalling engineer
• Improved by electrotechnician and robotic engineers
Main Prover

≡ Ad-hoc programming language: THEORY language
  • Prolog-like
  • Built-in B parsers
  • Programs transformed in bytecode programs executed by a VM

≡ Fixed point
  • No proof regression allowed with Atelier B releases
  • Since 1998, evolutions are new interactive commands and trigger-able sets of rules
2,700 rules to validate

- Manual demonstrations
- Cross-verification
- Third-party verification

The resulting validation forms
- 28 cm

A detailed demonstration

A concise one!

(*) « Obvious »
Predicate Prover

≡ Tableaux method
• Invented by JR Abrial to prove Atelier B rules
• Used to validate the majority of the 2700 rules of the Main Prover
• Became an interactive command with means to select reduced set of hypotheses

/* */

```c
\* BOOL31 */
INFRULE(BOOL31)
((v = FALSE) => Q)

=>
(not(v = TRUE) => Q);
```

/* */

```c
\* ECTR6 */

INFRULE(ECTR6)
binhyp(l=m) &
bnott(bgoal(not(L)=M)) &
bnott(bgoal(!x.H=>N)) &
band(binhyp(F=E),
band(bsubfrm(E,F,P,R),
binhyp(not(R))
)

=>
(P => Q);
```

≡ Tableaux method
• Hypotheses combined with not(goal) to obtain a contradiction
• Heuristics for wise instantiation
• Limited number of rules (116)
• Efficient when the number of hypotheses is low

≡ Added value
• Quickly identify errors
- Improve demonstrations efficiency
  - Abstract and reuse demonstrations
  - Fine grained tactics
  - Motto: do not lose any proof work

Interactive Proof

```
Operation( AssertionLemmas ) & Pattern( ST_7 <: E ) & dd & ah(Mhyp(ST_7: F)) & p0
```

Operation filter  Goal filter  Interactive commands

UserPass creation

```
Some Pos has been proved. Do you want to create or update the User Pass? (See preferences)
```

UserPass creation preview

1. `Operation(control) & Pattern(bool(a : {b} \/{c}\/{d})) = bool(bool(bool(a - b) = a or bool(a - c) = a) = a or bool(a = d) = a))`
2. `& ff(0) & dd & ah(e0 = e0$1) & ss & pp(rt.1)`

Generated tactic from successful proof

Do not lose any proof work
From proof script to tactic

t ⊆ 1 = TyE

≡ as is
≡ with minor modifications
≡ term abstraction
≡ bottom-up: avoid early generalization
From proof script to tactic
Following Air France Rio-Paris flight crash
- Requests for “proven software” for inertial centres
- Raw requirement: attitude matrix should keep a unit norm with a bounded error $\varepsilon$

New types: REAL, FLOAT
- FLOAT: new B0 type
- REAL: to specify ideal algorithms without considering precision

New operators
- Floating point literals are not accepted
- No predefined operator for converting float into real and float into integer (and vice-versa).

<table>
<thead>
<tr>
<th>Unified</th>
<th>Integer</th>
<th>Real</th>
<th>Float</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x \leq y$</td>
<td>$x \leq y$</td>
<td>$x \text{ rle } y$</td>
<td>$x \leq . y$</td>
</tr>
<tr>
<td>$x &lt; y$</td>
<td>$x &lt; y$</td>
<td>$x \text{ rlt } y$</td>
<td>$x &lt; . y$</td>
</tr>
<tr>
<td>$x \geq y$</td>
<td>$x \geq y$</td>
<td>$x \text{ rge } y$</td>
<td>$x \geq . y$</td>
</tr>
<tr>
<td>$x &gt; y$</td>
<td>$x &gt; y$</td>
<td>$x \text{ rgt } y$</td>
<td>$x &gt; . y$</td>
</tr>
<tr>
<td>$x + y$</td>
<td>$x + y$</td>
<td>$x \text{ rplus } y$</td>
<td>$x + . Y$</td>
</tr>
<tr>
<td>$-x$</td>
<td>$-x$</td>
<td>0.0 $\text{ rminus } x$</td>
<td>$- . X$</td>
</tr>
<tr>
<td>$x - y$</td>
<td>$x - y$</td>
<td>$x \text{ rminus } y$</td>
<td>$x &lt; . y$</td>
</tr>
<tr>
<td>$x * y$</td>
<td>$x * y$</td>
<td>$x \text{ rmul } y$</td>
<td>$x * . Y$</td>
</tr>
<tr>
<td>$x / y$</td>
<td>$x / y$</td>
<td>$x \text{ rdiv } y$</td>
<td>$x / . y$</td>
</tr>
<tr>
<td>$x ** y$</td>
<td>$x ** y$</td>
<td>$x \text{ rpow } y$</td>
<td>Invalid</td>
</tr>
<tr>
<td>$\text{min}(x)$</td>
<td>$\text{min}(x)$</td>
<td>$\text{rmin}(x)$</td>
<td>Invalid</td>
</tr>
<tr>
<td>$\text{max}(x)$</td>
<td>$\text{max}(x)$</td>
<td>$\text{rmax}(x)$</td>
<td>Invalid</td>
</tr>
<tr>
<td>$\text{SIGMA}(x).(y \mid z)$</td>
<td>$\text{SIGMA}(x).(y \mid z)$</td>
<td>$\text{rSIGMA}(x).(y \mid z)$</td>
<td>Invalid</td>
</tr>
<tr>
<td>$\text{PI}(x).(y \mid z)$</td>
<td>$\text{PI}(x).(y \mid z)$</td>
<td>$\text{rPI}(x).(y \mid z)$</td>
<td>Invalid</td>
</tr>
</tbody>
</table>
≡ New proof obligations
• Not currently handled by any prover

LIBPTF_types_init_data -

BEGIN
/* Init pure inertial attitudes */
 v_ptf_data := rec ( pure_inertial_attitudes : rec ( Roll : 0.0 ,
 Pitch : 0.0 , Azimuth : 0.0 ) ,
 T_vm : % xx . ( xx : 0 .. 2 | ( 0 .. 2 ) ^ { 0.0 } ) ,
 T_vb : % xx . ( xx : 0 .. 2 | ( 0 .. 2 ) ^ { 0.0 } ) ,
 deltaV_v : ( 0 .. 2 ) ^ { 0.0 } ,
 q_vm : ( 0 .. 3 ) ^ { 0.0 } )
END ;

IR_T1_CINAV_nav_init ( p_Velocity_InitVALUE , p_Position_InitVALUE ,
 v_vertical_velocity , v_altitude , v_deltaT ) =

PRB
 v_deltaT : tFloat32 &
 p_Velocity_InitVALUE : tVec3_64 &
 p_Position_InitVALUE : tVec3_64 &
 v_altitude : tFloat64 &
 v_vertical_velocity : tFloat64

THEN

LET
 pa_coriolis_correction ,
 v_vect_pos_init

BE
/*! Init Velocity */
 pa_coriolis_correction = ( 0 .. 2 ) ^ { 0.00 } &
 v_vect_pos_init = ( 0 .. 2 ) ^ { 0.00 }

INV
Data validation
Formal Data Validation

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

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

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>ID</td>
<td>IP</td>
<td>Type</td>
<td>UpLink</td>
<td>DownLink</td>
<td>Length</td>
<td>GPS 1</td>
<td>GPS 2</td>
</tr>
<tr>
<td>Route_tx_001</td>
<td>243</td>
<td>R</td>
<td>Route_tx_005</td>
<td>Route_vs_002</td>
<td>345</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Route_vs_002</td>
<td>128</td>
<td>R</td>
<td>Route_vs_002</td>
<td>EndLine_000</td>
<td>128</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Switch_w_003</td>
<td>256</td>
<td>S</td>
<td>Route_vs_128</td>
<td>Route_vs_006</td>
<td>23</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Relay_s_004</td>
<td>12</td>
<td>192.16.4.0</td>
<td>Y</td>
<td>Route_vs_128</td>
<td>291</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Route_rx_005</td>
<td>3</td>
<td>R</td>
<td>Route_vs_006</td>
<td>Route_vs_128</td>
<td>291</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Relay_s_001</td>
<td>3</td>
<td>192.16.4.125</td>
<td>Y</td>
<td>Route_vs_128</td>
<td>291</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Route_tx_006</td>
<td>22</td>
<td>R</td>
<td>EndLine_001</td>
<td>Route_vs_002</td>
<td>119</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Route_vs_128</td>
<td>127</td>
<td>R</td>
<td>Route_vs_006</td>
<td>Route_vs_002</td>
<td>145</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Switch_w_009</td>
<td>242</td>
<td>S</td>
<td>Route_vs_128</td>
<td>Route_vs_006</td>
<td>34</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EndLine_000</td>
<td>0</td>
<td>192.16.4.0</td>
<td>G</td>
<td>Route_vs_002</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EndLine_001</td>
<td>1</td>
<td>N</td>
<td>Route_vs_002</td>
<td>Route_vs_006</td>
<td>31</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Signal_vs_002</td>
<td>32</td>
<td>G</td>
<td>Route_vs_128</td>
<td>Route_vs_006</td>
<td>22</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Signal_vs_003</td>
<td>33</td>
<td>192.16.4.13</td>
<td>G</td>
<td>Route_vs_006</td>
<td>51</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Balise_b_001</td>
<td>301</td>
<td>B</td>
<td>Route_vs_128</td>
<td>Route_vs_005</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Balise_b_002</td>
<td>302</td>
<td>B</td>
<td>Route_vs_128</td>
<td>Route_vs_005</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Up to 100,000+ raw data chunks

Are they
- Consistent?
- Correct?
- Safe?

*Formal Data Validation in the Railways, T. Lecomte, SSS 2016*
*From Animation to Data Validation: The ProB Constraint Solver 10 Years On, M. Leuschel, J. Bendisposto, I. Dobrikov, S. Krings, D. Plagge, book chapter 2014*
Formal Data Validation

≡ Consistency, correctness, safety
  • Expressed with the mathematical language of B
  • Work well with graph-based properties
  • Provide counter examples when errors found

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

≡ Industry-Ready [PUSH-BUTTON !]
  • Deal with permanent changes in data and properties
  • Redundant tools to obtain diversity
  • Rules reused from one project to another
  • More than 30 sites verified including:
    • Singapore, Panama, Riyadh, Mecca
FOR sig, ixl

WHERE

    sig : sys_sud_er::Signal &
    sig : dom(sys_sud_er::Signal__dptId) &
    sig : dom(ic::sys_sud_er::signal_geopoint) &
    ic::sys_sud_er::signal_geopoint(sig) : ic::sys_sud_er::zone_GPZone(sys_sud_er::IXL_Core__singleZone(ixl))

THEN

VERIFY

    sys_sud_er::Signal__dptId(sig) : ran(sys_sud_er::IXL_Core__signal(ixl))

MESSAGE

    «The signal %1 belongs to IXL_Core %2 territory but is not referenced among its signals.»

ARG sys_sud_er::Signal__name(sig) TYPE STRING
ARG sys_sud_er::IXL_Core__name(ixl) TYPE STRING

ENDVERIFY

ENDFOR
Integration of new formal verification technologies

- Increase ratio of automatically proved POs
- Decrease cost of interactive proof
- Off-the-shelf automatic provers
ProB
- Added to Atelier B 4.3.1 as an interactive command.
  - Syntax: prob(n) or prob(n | T).
  - T: timer in seconds,
  - n=1 selects all hypotheses that have a symbol in common with the goal
- Add resource: ATB*PR*ProB_Path:C:\<path>\probcli.exe
- Funded by Alstom

Multiprover platform
- Bware research project (http://bware.lri.fr/)
- Connected through Why3
- +100 k Proof obligations (obfuscated) for benchmark
- 78.9% -> 99% automatic proof for one significant project
- Integration in Atelier-B: iapa

“Teaching an old dog new tricks” - Drudges
- During interactive proof, call external prover on current sub-goal
- If prover successful:
  - create proof rule
  - include rule in project
  - apply rule to discharge sub-goal

*Increasing Proofs Automation Rate of Atelier-B Thanks to Alt-Ergo,*
S. Conchon, M. Iguernlala, RSSR 2016
Discharging Proof Obligations from Atelier B using Multiple Automated Provers
David Mentré, Claude Marché, Jean-Christophe Filliâtre, Masashi Asuka.
ABZ - 3rd International Conference on Abstract State Machines, Alloy, B and Z, Jun 2012, Pisa, Italy.
Springer, 2012
Interfacing Automatic Proof Agents in Atelier B: Introducing "iapa"

Lilian Burdy, David Deharbe, Étienne Prun.

The drudges of the interactive prover

≡ Problem statement
  • “obvious” sub-goals in the interactive prover

≡ Solution approach
  • Have third-party provers handle sub-goals

≡ Better solution approach
  • Have third-party provers handle sub-goals
  • and have the result translated back in proof rules and applications

≡ Better solution implementation
  • Translate an (abstraction) of goal to a simplified logic
  • Have a SMT solver handle the abstraction
  • If successful,
    • get unsat core
    • translate back to corresponding hypotheses
    • produce proof rule from hypotheses and goal (feedback loop with solver)
    • apply rule to current goal
  • Verification of proof rules is automatic or manual.
The drudges of the interactive prover
The drudges of the interactive prover

- Button to start drudges
- Proof rule application
- Created proof rule

Atelier B - Prover - drudge - test

Proof Edit View Help

View as tree

C:\Program Files (x86)\Atelier B full 4...
NAT = 0..2147483647
INT = -2147483647..2147483647 ∧
btrue ∧ SS ∈ FIN(ℤ) ∧
¬(SS = Ø) ∧
TT ∈ FIN(ℤ) ∧
¬(TT = Ø) ∧
pp ∈ SS*SS → TT ∧
qq ∈ SS*SS → TT ∧
t1 ∈ TT ∧
t2 ∈ TT ∧
∀(s1,s2). (s1 ∈ SS ∧ s2 ∈ SS ∧ s1 → s2 ∈ dom(pp) ∧ pp(s1 → s2) = t1) ∧
∀(s1,s2). (s1 ∈ SS ∧ s2 ∈ SS ∧ s1 → s2 ∈ dom(qq) ∧ qq(s1 → s2) = t2) ∧
∃(aa,bb). (aa ∈ SS ∧ bb ∈ SS ∧ aa → bb ∈ dom(pp) ∧
∃(aa,bb). (aa ∈ SS ∧ bb ∈ SS ∧ aa → bb ∈ dom(qq) ∧
∃(uu,vv). (uu ∈ SS ∧ vv ∈ SS ∧ uu → vv ∈ dom(qq)) →

(set-option :print-success false)
(set-option :produce-unsat-cores true)
(set-option :produce-proof true)
(set-logic UF)
(declare-sort U 0)
(declare-fun TRUE () U)
(declare-fun FALSE () U)
(declare-fun mem (U U) Bool)
(declare-fun bool (Bool) U)
(assert (not (= TRUE FALSE)))
(declare-fun p0 () U)
... (declare-fun p18 () Bool)
(assert (! (= p0 (interval p1 p2)) :named h0))
(assert (! (= p3 (interval (uminus p2) p2)) :named h1))
... (assert (! p18 :named h14))
(assert (! (not (exists ((q0 U) (q1 U)) (and (and (mem q0 p5) (mem q1 p5)) (mem (mapplet q0 q1) (dom p10))) (= (apply p9 (mapplet q0 q1) p12)) :named goal)))
(check-sat)
(get-unsat-core)
(exit)
∃(s1,s2). (s1 ∈ SS ∧ s2 ∈ SS ∧ s1 → s2 ∈ dom(pp) ∧ pp(s1 → s2) = t1) ⇒
∃(s1,s2). (s1 ∈ SS ∧ s2 ∈ SS ∧ s1 → s2 ∈ dom(qq) ∧ pp(s1 → s2) = t2) ∧
∃(aa,bb). (aa ∈ SS ∧ bb ∈ SS ∧ aa → bb ∈ dom(pp) ∧ pp(aa → bb) = t1)
⇒
∃(aa,bb). (aa ∈ SS ∧ bb ∈ SS ∧ aa → bb ∈ dom(pp) ∧ pp(aa → bb) = t1) ⇒
∃(uu,vv). (uu ∈ SS ∧ vv ∈ SS ∧ uu → vv ∈ dom(qq) ∧ pp(uu → vv) = t2)

≡ Increase formula depth until proof found

Iteration 1

... (assert (! p0 :named h0))
(assert (! p1 :named h1))
(assert (! (not p2) :named goal))
(check-sat)
(exit)
unknown

Iteration 2

... (assert (! (=> p0 p1) :named h0))
(assert (! (exists ((q0 U)) p2) :named h1))
(assert (! (not (exists ((q0 U)) p3)) :named goal))
(check-sat)
(exit)
unknown

etc.
(assert (! (=> (exists ((q0 U)(q1 U)) (and (and (and (mem q0 p5) (mem q1 p5)) (mem (mapplet q0 q1) (dom p9))) (= (apply p9 (mapplet q0 q1)) p11))) ) (exists ((q0 U)(q1 U)) (and (and (and (mem q0 p5) (mem q1 p5)) (mem (mapplet q0 q1) (dom p10))) (= (apply p9 (mapplet q0 q1)) p12))) :named h11))
(assert (! (exists ((q0 U)(q1 U)) (and (and (and (mem q0 p5) (mem q1 p5)) (mem (mapplet q0 q1) (dom p9))) (= (apply p9 (mapplet q0 q1)) p11)) ) :named h13))
(assert (! (not (exists ((q0 U)(q1 U)) (and (and (and (mem q0 p5) (mem q1 p5)) (mem (mapplet q0 q1) (dom p10))) (= (apply p9 (mapplet q0 q1)) p12))) :named goal))
(check-sat)
(exit)

≡ Translate to proof rule

band(binhyp(#(a,b).(a : c & b : c & a|->b : dom(d) & d(a|->b) = e) => #(a,b).(a : c & b : c & a|->b : dom(f) & d(a|->b) = g)),
binhyp(#(h,i).(h : c & i : c & h|->i : dom(d) & d(h|->i) = e))) &
a \(c,d,e,f,g) &
b \(c,d,e,f,g) &
h \(c,d,e) &
i \(c,d,e) &
j \(c,f,d,g) &
k \(c,f,d,g)
=>
(#(j,k).(j : c & k : c & j|->k : dom(f) & d(j|->k) = g))
Wrap-up

≡ Need to diversify the portfolio of provers
  • As stand-alone tools or in cooperation with existing prover
  • Tool or process certification in mind

≡ Next version of Atelier B will include the presented work

≡ Real arithmetics and floating point representation is still a challenge