Atelier B has turned twenty
What happened during the last 20 years?

Focus on some aspects:
• Modelling
• Proving
• Generating code

Not a raw description of the tool
More how its usage has evolved
Driven by projects
Atelier B initially developed by Fernando Mejia

- Development funded for Driverless metro in Paris (L14)
- First customer: University of Sherbrooke
- METEOR released
- Creation of ClearSy (full ownership)
- Atelier B Community + Maintenance

Tools

- B / C code generator
- B / HIA code generator
- B / Ada code generator
- Automatic refiner
- B model animator
- Event-B animator
- Formal data validation support
- Traceable proof obligation generator
- B / Ada code generator
- B / HIA code generator
- B / Ladder code generator
- B / C code generator
- Event-B / VHDL code generator
- B / generic C code generator
- B / binary (HEX) code generator

Applications

- First metro automatic pilot METEOR (Paris L14)
- First EAL5+ (CC 2.3) smartcard certificate
- First Platform screen door Controller (Paris L3)
- First symbolic calculus engine in B
- First safety assessment in Event-B (New York L7)
- First (xml) compiler in B

Timeline:

- 1994
- 1995
- 1996
- 1997
- 1998
- 1999
- 2000
- 2001
- 2002
- 2003
- 2004
- 2005
- 2006
- 2007
- 2008
- 2009
- 2010
- 2011
- 2012
- 2013
- 2014
- 2015
- 2016

Platforms:

- Sun
- HP
- Linux
- MacOS
- Windows
- Linux
- MacOS
Modelling
Modelling Random Walk

Safety Critical Railway Apps

- Automatic Refinement
- Simplified Abstract Modelling
- Formal Data Validation
- Real & Floating Point Numbers

B

Event-B

Safety Applications

- Component-Based System Description
- Security & Common Criteria
- Safety Demonstration

BART
B Automatic Refiner Tool

EMA
Abstract Model Editor

Composys

RIP 2012
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)

≡ +30% automatic metros in the world

_B initial success is due to urban women fertility_

Confirm / Deny?
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
- 3 000 definitions
- 23 000 proof obligations (83 % automatic proof)
- 3 000 added user rules (85 ?? % automatic proof)

Platform Screen Doors
- Top component: 12 variables 14 operations
- 10 components, 2 kloc specification, 2 kloc 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 ...*

```c
variables : (types &
properties &
properties_train &

(loc_trainLocated = TRUE =>
0 <= loc_locationUncertainty
& kine_kineInvalid = FALSE
& loc_train_track /= {})
& first(loc_train_track) = loc_ext2Block |-> oppositeDirection(loc_ext2Dir)
& lli. (ii : 1..size(loc_train_track)-1 => loc_train_track(ii) : dom(sidb_nextBlock))
& lli. (ii : 1..size(loc_train_track)-1 => sidb_nextBlock(loc_train_track(ii)) = loc_train_track(ii))
& lli. (aa : 1..size(loc_train_track) & prj1(t_block,t_direction)(loc_train_track(aa))

& loc_rearBlock = { c_cabin1 |-> loc_ext2Block, c_cabin2 |-> loc_ext1Block, c_none |-> loc_rearDir|c_up}
& (not (loc_rearBlock = c_block_init)

 => loc_rearDir - { c_cabin1 |-> oppositeDirection(loc_ext2Dir), c_cabin2 |-> c_none |-> loc_rearDir|c_up}
& loc_rearAbs = { c_cabin1 |-> loc_ext2Abs, c_cabin2 |-> loc_ext1Abs, c_none |-> 0
& loc_frontBlock = { c_cabin1 |-> loc_ext1Block, c_cabin2 |-> loc_ext2Block, c_none |-> loc_ext2Block
& loc_frontDir = { c_cabin1 |-> loc_ext1Dir, c_cabin2 |-> loc_ext2Dir, c_none |-> loc_ext2Dir
& loc_frontAbs = { c_cabin1 |-> loc_ext1Abs, c_cabin2 |-> loc_ext2Abs, c_none |-> loc_ext2Abs
```
User added rules to bootstrap the main prover:

- THEORY language (ako PROLOG)
- Pattern-matching
- Backward, forward, rewrite rules
- Added at project level or component level
- Partly reused from project
- Have to be validated to confirm that a project is 100% proven

Specific tool & GUI to validate user added rules (maintenance version)

- Translation to predicates
- Predicate prover
- Peer review
- Report
Automatic Refinement

≡ Progressive transformation of a model containing all design decisions into a model able to be translated into an imperative language

≡ Initially invented by Matra Transport
  • To support their own development methodology (abstract model first)
  • In-house tools (EdithB + Bertille) to obtain an efficient process
  • Refinement engine based on refinement rules

Modern Automatic Train Protection Software (2015)
Automatic Refinement

≡ Not a push-button tool
• Automate what the modeller does manually
• Refine variables then substitutions
• Adding rules is mandatory when automatic refinement fails (iterations)
• Rich language: type information, operation declaration, etc.

```
RULE assign_a_bool_belongs_b_c_16
REFINES
  @a := bool(@b) =>@d : @c@e
REFINEMENT
  @a := bool(@b:@c & @d:@e)
END;
```
Used rules:

RULE assign_a_union_a_b
REFINES
@a := @a V @b
WHEN
setarrayNAT(@a)
IMPLEMENTATION
TYPE_ITERATION(index => #1,
type => NAT,
body => (IMPORTED_OPERATION(out => ()
in => (#1),
pre => (#1 : NAT),
body => (@a := @a V ((#1)\(@b\)))),
invariant => (@a = @a$0(V(NAT_treated\(/}@b\))))
END

RULE add_el_to
REFINES
@a := @a V \{@b\}
WHEN
setarrayNAT(@a) &
B0EXPR(@b)
IMPLEMENTATION
@a_{l}(\@b) := TRUE
IMPLEMENT
@a
END

ABSTRACT_VARIABLES
sub1, sub2, sub3
INVARIANT
sub1 <= NAT &
sub2 <= NAT &
sub3 <= NAT
OPERATIONS
op1(ee) = PRE
ee : NAT
THEN
sub1 := sub1 V sub2 ||
sub3 := sub3 V \{ee\}
END

Generated components:

CONCRETE_VARIABLES
sub3_{i}, vg_loop
INVARIANT
sub3_{i} : NAT --> BOOL &
sub3 = sub3_{i}-\{TRUE\} &
vg_loop : BOOL
OPERATIONS
op1(ee) = VAR I_{1} IN
vg_loop <- init_iteration_NAT ;
WHILE vg_loop = TRUE DO
vg_loop, I_{1} <- continue_iteration_NAT ;
/*? sub1 := sub1 V ((I_{1}) \ sub2) ?*/
op1_{1}(I_{1})
INVARIANT
[... ] &
sub1 = sub1$0 V (NAT_treated \ sub2)
VARIANT
card(NAT_to_treat)
END :
sub3_{i} ( ee ) := TRUE
END
≡ No need to certify the tool
• Whatever the refinement rules, in the engine or added by users
• If the refined model is incorrect, it can’t be proved

≡ Impact on projects
• Siemens claimed to divide development costs by 2 when the process is stabilized
• More little steps, more decomposition levels, more function calls, more lines of code
• Refinement patterns ease deployment of proof strategies

≡ Industrial applications
[2004] Canarsie line (ATP):
• 38k lines of handwritten B
• 115k lines of generated B

[2007] Roissy airport shuttle (ATP+ZC):
• 40k lines of handwritten B
• 225k lines of generated B

[2007] Symbolic Calculus Engine:
• 200k lines of generated B

[2015] SysML Compiler:
• 300k lines of generated B
Propose a new way of modelling

- User is incited to think in terms of functions
- Model creation, decomposition, variables allocation is left to the computer

Outline
- Functions
- Variables
- Constants

Function
- Parameters
- Predicates
- Body

Variables graph
- What is produced
- What is consumed
Simplified Abstract Modelling

≡ Two Modelling Views for the Same Project
  • Projects can me edited what ever the view
  • Providing a clear distinction between abstract model and concrete model
  • Corollary: not all projects can be analysed
  • Direction continuation of the Automatic Refinement approach

Abstract Modelling Editor (EMA) to appear in Atelier B 4.4

Atelier B current component view
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>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Name</td>
<td>ID</td>
<td>IP</td>
<td>Type</td>
<td>UpLink</td>
<td>DownLink</td>
<td>Length</td>
</tr>
<tr>
<td>2</td>
<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>
</tr>
<tr>
<td>3</td>
<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>
</tr>
<tr>
<td>4</td>
<td>Switch_w_003</td>
<td>256</td>
<td>192.16.5.55</td>
<td>S</td>
<td>Route_vs_128</td>
<td>Route_tx_006</td>
<td>23</td>
</tr>
<tr>
<td>5</td>
<td>Relay_s_004</td>
<td>12</td>
<td>192.16.4.10</td>
<td>Y</td>
<td></td>
<td></td>
<td>N 50.85 963</td>
</tr>
<tr>
<td>6</td>
<td>Relay_rx_005</td>
<td>3</td>
<td>R</td>
<td>Route_tx_006</td>
<td>Route_vs_128</td>
<td>291</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Relay_s_001</td>
<td>55</td>
<td>192.16.4.125</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Route_tx_006</td>
<td>22</td>
<td>R</td>
<td>EndLine_001</td>
<td>Route_vs_002</td>
<td>110</td>
<td></td>
</tr>
<tr>
<td>9</td>
<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>
</tr>
<tr>
<td>10</td>
<td>Switch_w_009</td>
<td>242</td>
<td>192.16.4.10</td>
<td>S</td>
<td>Route_vs_128</td>
<td>Route_tx_006</td>
<td>34</td>
</tr>
<tr>
<td>11</td>
<td>EndLine_000</td>
<td>0</td>
<td>E</td>
<td>Route_vs_002</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>EndLine_001</td>
<td>1</td>
<td>E</td>
<td>Route_vs_002</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>Signal_x_002</td>
<td>32</td>
<td>192.16.4.12</td>
<td>G</td>
<td>Route_vs_128</td>
<td></td>
<td>22</td>
</tr>
<tr>
<td>14</td>
<td>Signal_x_003</td>
<td>33</td>
<td>192.16.4.13</td>
<td>G</td>
<td>Route_tx_006</td>
<td></td>
<td>51</td>
</tr>
<tr>
<td>15</td>
<td>Balise_b_001</td>
<td>301</td>
<td>B</td>
<td>Route_vs_128</td>
<td></td>
<td></td>
<td>N 50.85 933</td>
</tr>
<tr>
<td>16</td>
<td>Balise_b_002</td>
<td>302</td>
<td>B</td>
<td>Route_tx_005</td>
<td></td>
<td></td>
<td>N 50.86 123</td>
</tr>
</tbody>
</table>

Are they
- Consistent?
- Correct?
- Safe?

Up to 100,000+ raw data chunks

Formal Data Validation in the Railways, T. Lecomte, SSS 2016
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, Ryad, Mecca

SECURE DATA VALIDATION

Project
  • [Table of verified projects with metrics and status]

powered by CLEARIS
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
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 Data Validation

≡ New feature
- Added to Atelier B 4.2
- Requires ProB to be installed
- Able to validate constants against properties
- Able to generate/compute missing values
Involved in Several R&D projects

- French proposal rejected “because Event-B mathematics do not allow to model systems” *(sic)*

- **FP5 MATISSE** (Methodologies And Technologies for Industrial Strength Systems Engineering)
  - First language draft specification
  - Software-based systems
  - Event-B to B translation to experiment
  - UML-B

- **FP5 PUSSEE** (Paradigm Unifying System Specification Environments for Proven Electronic Design)
  - Electronic-based systems
  - VHDL code-generation

- **FP6 RODIN** (Rigorous Open Development Environment for Complex Systems)

- **FP7 DEPLOY** (Industrial deployment of system engineering methods providing high dependability and productivity)
Event-B applied to a number of diverse applications:
- Banking (Société Générale)
- Ariane 5 Flight Sequencer Software (Délégation Générale de l’Armement)
- Micro-Satellites Constellation Flight (Centre National d’Etudes Spatiales)
- Air Traffic Control (Eurocontrol, CEAT)
- Press (Caisse Nationale d’Assurance Maladie)
- De-icing Helicopters (Eurocopter)
- Nuclear Plant (Electricité de France)
- Multiplexed Cars (Peugeot, Renault)
- Smartcard (STMicroelectronics)
- Etc.

No positive conclusion:
- understanding,
- compliance with existing development cycle & methods,
- existing teams,
- standards,
- etc.

Studies mostly for free!
- Bootstrap activity
- Studies mainly performed with Atelier B (tool “sufficient to handle problems”)

Except:
- Vehicles (diagnosis, assistance to integration)
- Smartcard (certification, proven product)
- Railways (safety critical systems)
Flat Behavioural Specification (maintenance, failure diagnosis)
- Event-B to describe behaviour, light proof
- Find ambiguities in the specification documents (thousands pages)
- Events are allocated to components
- Analyse model structure to deduce dependencies: where is produced, where is consumed
- Generate documentation from model and additional information

\[
F_{Opening} = \begin{cases} 
\text{SELECT} & \text{CommandOpening} = \text{TRUE} \\
\text{THEN} & \text{PSDOpen} := \text{TRUE} \\
\text{END}; & 
\end{cases}
\]

B model

\[
\text{VARIABLE PSDOpen} \\
\text{MEDIUM Internal} \\
\text{TRANSLATION} \\
\text{Status of the non locked PSD: open or not} \\
\text{END}
\]

TRANSLATION PSDOpen := TRUE
PSD are opening
END

Modelling Main Issue
- What could be done when millions of hours have been spent designing a car?
Component-Based System Description

≡ Tooling
- Scripts developed during modelling sessions turned into a standalone tool: Composys
- Based on Atelier B BCompiler

≡ Cars Modeling
- Applied to the failure modelling of 4 Peugeot cars in order to locate faulty ECU(s)
- To help mechanics to diagnose failures
- Stopped in 2005, Peugeot switched back to only provide error codes without further analysis

≡ Military Vehicule Modeling
- Applied to the design of unique vehicle designed from scratch
- To help integrating tons of systems

Reponse_ID =
SELECT
vl_couplage = normal &
Attente_response = TRUE &
PresenceIDCoteDroit = TRUE
THEN
ID_Reponse :: {REP_pirate, REP_PSA} ||
v1_couplage ::= ph_Emission_reception
END
Component-Based System Description

≡ On Going Integration

≡ The Resulting Vehicle
Component-Based System Description

≡ Complex Relationships
= Platform Screen Doors (railways)
  • New system to avoid people to die (pushed) on tracks in Paris
  • Trick: detect train arrival and door opening without direct communication
  • Composys to analyse the system and provide documentation
  • Finally an Event-B model to ensure safety with proof

= Real System on Paris L13 (proof of concept)
  • Developed from scratch in 6 months
  • Lot of missing skills to acquire
  • Software developed manually from formal specification
  • SIL4: one failure every 10 000 years
  • 99,999% reliability: one train max missed per year
  • one year successful experiment: no death, no train missed

Safe and Reliable Metro Platform Screen Doors Control/Command Systems, T. Lecomte, SBMF 2008
Component-Based System Description

≡ Proved but still unsafe …
  • Proven Event-B model presented to RATP
  • But major issue uncovered by RATP technician

≡ Specification changed: “do not open PSD in case of incorrect train position”
  • Did we select the “correct” dimensions to analyse the system?
  • What about Japanese tourists taking pictures on the other platform, birds flying through the tunnels, etc.?
≡ For the next call for tender (Paris line 1), we were ready

≡ Brama Model Animator (Rodin)
The model

≡ When it is the first time, it is the first time!
The model

≡ Connecting things with the real station

Et voilà !
Smartcard industry demands for higher security levels

- ... for marketing reasons
- First attempt:
  - Model API functions (library available to third-party)
  - Model Security Policy (define what access conditions are)
  - Prove that Security Policy is enforced by the functional specification
  - Check (traceability) that specification is implemented
  - Compliant with EAL5+ (Common Criteria 2.2)
  - 1 Event-B model, 4 levels of refinement
  - Replayed several times for different companies and products
  - Demonstration is limited to non-disturbed behaviour “When it works, it works”

Down to the hardware

- Same approach than above, Rodin not op, Event-B support added to Atelier B
- Component to study: Memory Protection Unit
- 1 Event-B model, 18 levels of refinement ready for VHDL code generation
- Development: 6 months, 183 versions, 5 major refactoring
- Final product: 20,000 gates

18 levels for an IF-THEN-ELSE ???

How many levels for Intel Core i7 ???
• 18 levels of refinement for a 20k gates micro-electronic component
• How many zillions levels required to model a metro line and address very low level details?
• Challenge: proving that a new CBTC (Thales) enforces safety on New York L7 metro
  – No structural “model of everything”
  – End-product readable and understandable by customers
  – Reusable on other lines

Safety Demonstration
Safety Demonstration

• Protection zone proof: 35 assumptions, 2 subproofs, 30 events, 1400 loc model

• 2 years to complete

• NYCT analysed positively the proof documents

• Reused on Culver line (50% effort reduction)

• Experimented on:
  – ERTMS Regional
  – Paris metro (xx lines automated until 2030)

Reference:
D. Sabatier, L. Burdy,
New York Metro Flushing line: System level formal verification, June 2013
### Real & Floating Point Numbers

≡ 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
- Floating point literals are not accepted

≡ 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>$x \leq . y$</td>
</tr>
<tr>
<td>$x \times y$</td>
<td>$x \times y$</td>
<td>$x \text{ rmul } y$</td>
<td>$x \times . Y$</td>
</tr>
<tr>
<td>$x / y$</td>
<td>$x / y$</td>
<td>$x \text{ rdiv } y$</td>
<td>$x / . y$</td>
</tr>
<tr>
<td>$x \times y$</td>
<td>$x \times y$</td>
<td>$x \text{ rpow } y$</td>
<td>Invalid</td>
</tr>
<tr>
<td>min$(x)$</td>
<td>min$(x)$</td>
<td>rmin$(x)$</td>
<td>Invalid</td>
</tr>
<tr>
<td>max$(x)$</td>
<td>max$(x)$</td>
<td>rmax$(x)$</td>
<td>Invalid</td>
</tr>
<tr>
<td>SIGMA$(x) \cdot (y \mid z)$</td>
<td>SIGMA$(x) \cdot (y \mid z)$</td>
<td>rSIGMA$(x) \cdot (y \mid z)$</td>
<td>Invalid</td>
</tr>
<tr>
<td>PI$(x) \cdot (y \mid z)$</td>
<td>PI$(x) \cdot (y \mid z)$</td>
<td>rPI$(x) \cdot (y \mid z)$</td>
<td>Invalid</td>
</tr>
</tbody>
</table>
≡ New proof obligations
• Not currently handled by any prover
• Bware project would contribute to it

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_CNAV_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 }

IN
Proving

Predicate Prover ➔ Proof Tools

Manual Proof

Main Prover

Mono Lemma ➔ Rodin

Model Checking ➔ ProB

Bware, LCHIP ➔ Alt-Ergo

Why3

Interactive Proof

Formal Data Validation

PredicateB++
≡ Rule-based Proof Engine
• Predicates are decomposed and simplified
• Hypotheses are generated and simplified
• Set of rules includes Alsthom and Matra rules

≡ State of the Art
• Invented by a signalling engineer
• Improved by electrotechnician and robotic engineers

Interactive Prover
Optimized Proof Obligation Loading

Processing Hypotheses
Processing Goal

Patchprover
SolvePred
Skolemisation
Equalities

Contradiction if goal false
DED zone: H => G
Simplification \exists x.P
SolvePred
Rule package selection
Surtype any new goal
Heuristics for \neg P, \exists x.P
Apply equalities in goal
Generate new hypotheses
Proof by case
Patchprover
≡ Ad-hoc programming language: THEORY language
  • Prolog-like
  • Built-in B parsers
  • Programs transformed in bytecode programs executed by a VM
  • Integrated to Rodin as a plugin (Mono Lemma version)

≡ Fixed point
  • 1 proof obligation to demonstrate manually == 35 €
  • 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

Testimony:
a rule modified 9 times by 2 experts while being constantly wrong

≈ The resulting validation forms
- 28 cm

A detailed demonstration

A concise one!

(*) « Obvious »
≡ Tableau 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
• Integrated to Rodin as a Plugin

```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);
```

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

≡ Added value
• Quickly identify errors
Interactive Automatic Proof or Automatic Interactive Proof

- Remember: 1 PO == 35 €
- Improve demonstrations efficiency
  - Abstract and reuse demonstrations
  - Fine grained tactics
- Motto: do not lose any proof work

Rule packages
- Backward
- Forward
- Rewriting

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)

OK Cancel

Do not lose any proof work

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)
Connecting New 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

≡ Alternate provers

- Bware research project (http://bware.lri.fr/)
- Connected through Why3
- +100k Proof obligations (obfuscated) for benchmark
- 78% -> 99% automatic proof for one significant project

*Increasing Proofs Automation Rate of Atelier-B Thanks to Alt-Ergo, S. Conchon, M. Iguernlala, RSSR 2016*
Connecting New Provers

Example of Why3 fragment

≡ Alternate provers (bis)
- LCHIP research project (to start in Q4 2016)
- Connected through Why3, still
- Alt-Ergo integrated to Atelier B, Cubicle model-checker for evaluation
Generating
≡ Long sequence of implementations
  • Paris L14 (1998) : 80 kloc
  • Paris Roissy Airport Shuttle (2006): 186 kloc onboard, 50 kloc trackside
  • Lille (2015): 300 kloc xml compiler

≡ Using different technologies (redundancy)
  • encoding (FIDARE): 2 instances
    • inputs, outputs, variables, instructions are encoded
    • the coding key is different for each instance
  • diversity (inverse mirror): 2 instances
    • one instance uses big endian model,
    • the other one executes little endian model
  • specific hardware (coded secure processor): 1 instance
    • Variables have two fields: value and code. Instructions (OPELS) modified both accordingly. Mismatching value and code indicates a memory corruption.
    • Compensation tables contains tags for all possible path (masks). An unexpected value indicates a program counter corruption
≡ Privileged code generation path
  • Code generator that produces:
    • Readable code
    • Compliant with (railways) standards: limited use of pointers
    • Restrictions on accepted B0 (no multi instance)

≡ Modes
  • Code generation for a:
    • Machine: skeleton for the component
    • Implementation: code for the component
    • Project: makefile and code for the components
  • Profiles:
    • Code generation is highly dependant on the target
    • Predefined translation profiles allow adapt generated code

≡ INT
  • INT can be redefined at project level to adapt to algorithm precision
≡ Applications
• C code embedded at Sao Paulo L15 and Stockholm Citybanan

≡ Embedded systems
• Aimed at low cost hardware (from 6€ to 100€)
• Resources available in the future at https://github.com/TProver/TRBM: Training Resources for the B Method
== Programming PLCs ==
- One language (out of 5) for programming PLCs (IEC61131-3)
- Very similar to electric circuits

\[
\begin{array}{c}
\text{1. } \quad \text{x} & \quad \text{y} & \quad \text{s} \\
\text{z} & \\
\end{array}
\]

The above realises the function: \( S = X \text{ AND } (Y \text{ OR } Z) \)

== Mandatory IDE ==
- Step7 IDE required to program Siemens-S7 to keep the SIL3 certificate
- Ladder code generator is not connected to Step7
- Ladder programs have to be manually entered into Step7 with the GUI
- Cross-read required to check identity between programs
- Solution abandoned
- Translator not published because too specific
≡ Programming PLCs
  • One language (out of 5) for programming PLCs (IEC61131-3)
  • Kind of assembly language for PLCs

≡ Mandatory IDE
  • Step7 IDE accept Instruction List copy/paste actions
  • Simpler, safer, no further cross-verification
  • Translator not published because too specific

≡ Applications
  • Bridge Crane Lyon (2013)
Bridge cranes (SNCF Lyon)

- Safe movements over catenary lines
≡ Objectives
• Code generator producing VHDL from proven Event-B models
• Seamless integration to Atelier B

≡ Development and fine-tuning
• Co-developed with STMicroelectronics Rousset (Smartcard division)
• Event-B model describes synchronous and asynchronous behaviour
• Code generator produces VHDL with a different structure from hand written one
• Applied to smartcard EAL6+ certified product (Memory Protection Unit)
  – Similar size (number of gates), less components, better debug vectors
  – Publication on this subject: A proved “correct by construction” realistic digital circuit [1]

≡ Exploitation (B4SYN)
• Free extension to Atelier B 4.4 (ETA Q4 2016)
• Level crossing controller being developed for French Railways (FPGA, SIL4

Event-B model

Constants

Variables

Events

Parameters

Constants

Variables

Events

Event-B model

Constants

• bit level operation

Variables

• reset
• clock
• inputs
• combinatorial outputs
• synchronous outputs

Events

• reset
• phi (input acquisition)
• psi (combinatorial propagation)
• update registers and synchronous outputs

Prove the non-divergence of new events (VARIANT clause)

Prove coverage

Prove exclusivity
Objectives
- Code generator producing LLVM Internal Representation (IR) from proven B models
- LLVM IR to be translated in any language supported by LLVM (JS, Java, C, Ada, etc.)
- Seamless integration to Atelier B

Development and fine-tuning
- Co-developed with UFRN (Brazil)
- B model describes software functions (no interrupt, no parallel execution)
- Code generator supports C4B B language subset
- Proof of concept demonstrated on several existing SIL4 software

Exploitation (B₂LLVM)
- Free extension to Atelier B 4.4 (ETA Q4 2016)
B Specification

\[
\begin{align*}
\text{result} & \leftarrow \text{pow}(\text{numberA}, \text{numberB}) = \\
& \text{BEGIN} \\
& \text{IF} \ \text{numberB} = 0 \ \text{THEN} \\
& \text{ELSE IF} \ \text{numberB} = 1 \ \text{THEN} \\
& \text{ELSE} \\
& \text{VAR} \ ii, \ exp \ IN \\
& \text{ii} := 1 \\
& \text{exp} := \text{numberA} \\
& \text{WHILE} \ \text{ii} < \text{numberB} \ \text{DO} \\
& \text{ii} := \text{ii} + 1 \\
& \text{INVARIANT} \\
& \text{ii} : \ INT \ & \text{ii} \leq \text{numberB} \ & \text{exp} \leq \text{numberA}^\text{numberB} \\
& \text{result} := \text{exp} \\
& \text{END} \\
& \text{END}
\end{align*}
\]

B Implementation

LLVM IR code

```c
define void @Calculator$pow(i32 NumberA, i32 NumberB, i32* Result) {
  entry:
    Nii = alloc i32
    Nexp = alloc i32
    NumberA = icmp sge i32 NumberB, 0
    br i1 N0, label Llabel0, label Lnext
  label0:
    N1 = icmp eq i32 NumberB, 0
    br i1 N1, label Llabel1, label Llabel2
  label1:
    store i32 1, i32* Result
    br label Lnext
  label2:
    store i32 NumberA, i32* Nresult
    br label Lnext
  label3:
    store i32 1, i32* Nexp
    store i32 NumberA, i32* Nexp
    br label Llabel0
  label4:
    N3 = load i32 Nii
    N4 = load slt i32 Nii, NumberB
    br i1 N4, label Llabel1, label Llabel5
  label5:
    N5 = load i32 Nexp
    N6 = mul i32 N5, NumberA
    store i32 N6, i32* Nexp
    N7 = load i32 Nii
    N8 = add i32 N7, 1
    store i32 N6, i32* Nii
    br label Llabel6
  label6:
    N9 = load i32 Nexp
    store i32 N9, i32* Result
    br label Lnext
  exit:
    ret void
}
```
Modèle

IMPLEMENTATION
machine_i

REFINES
machine

OPERATIONS
operation =
BEGIN
  IF TRUE=TRUE THEN
    skip
  ELIF TRUE=TRUE THEN
    skip
  ELSE
    skip
  END
END

Mapping
- 0x000018000 machine_initialisation
- 0x000018028 operation

ASM Mips32

Hex

≡ C32

• Generate Hex file from B0 implementation
Future
**Low Cost High Integrity Platform**

Input Model

- DSLs

Formal software development toolchain

- Formal model
  - Source code
  - Binary code
  - Application

- Feedback:
  - Proof
  - Translation

- Error backpropagation

Formal model

- Deployment:
  - Secure bootloading

- Secure execution

- Detection of divergent behaviour

≡ Automatic generation of proven code

- Formal part hidden behind the curtain
- Support for bounded algorithm complexity
- Domain Specific Languages: grafcet, relays drawings, etc (IEC61131-3) to keep development perturbation low

- To ease SIL4 certificate achievement:
  - Developer focused on functional aspects
  - Even non high grade engineers could participate

Funded by bpiFrance
1 B model (application)
- Data
  - Integer arithmetic
  - Boolean equations
  - State machine
  - Variables typing and properties
- Cyclic software, no interrupt modifying state variables

Application proved vs requirements
- Implementation doesn’t contradict specification
- A minima, no programming error:
  - overflow,
  - div 0,
  - tables

1 C code generator + commercial compiler => Hex

1 ASM MIPS / Hex code generator

Low level software ensuring safety developed in B
- Once for all
- Safety features out of reach of the developers
- No requirement to certify the tools
Low Cost High Integrity Platform

≡ Starter-kit @ Q3 2017
  • Mother board (I/Os, maintenance processor, network sockets)
  • Daughter card, embedding the double-processor
  • IDE v1, based on Atelier B

≡ IDE v2 @ Q2 2018

≡ IDE v3 @ Q2 2019
Conclusion & Perspectives

- **Atelier B still alive**
  - +10B passengers transported by « B inside » metros - Lille 2015, Paris L4 2018
  - +100M passengers going through doors operated by B software - Sao Paulo L15 2016, Stockholm Citybanan 2017

- **Is not only a tool for piloting metros**
  - Opening doors safely
  - Validating parameters
  - Contributing to the certification of microcircuits

- **New usage**
  - Integrated to PLC
  - Applications to (nuclear) energy automation

- **Forthcoming evolutions**
  - Linked with industrial needs
  - Funded by industrial & collaborative R&D projects

- Let’s meet for ABZ 2026!

**formal methods are only a tiny part of the whole picture**
Thank you for your attention