Formal Verification in the Industry

David Deharbe

SC-square Summer School 2017

Saarbrucken

Agenda

≡ 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*"

ClearSy Systems Engineering

- 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

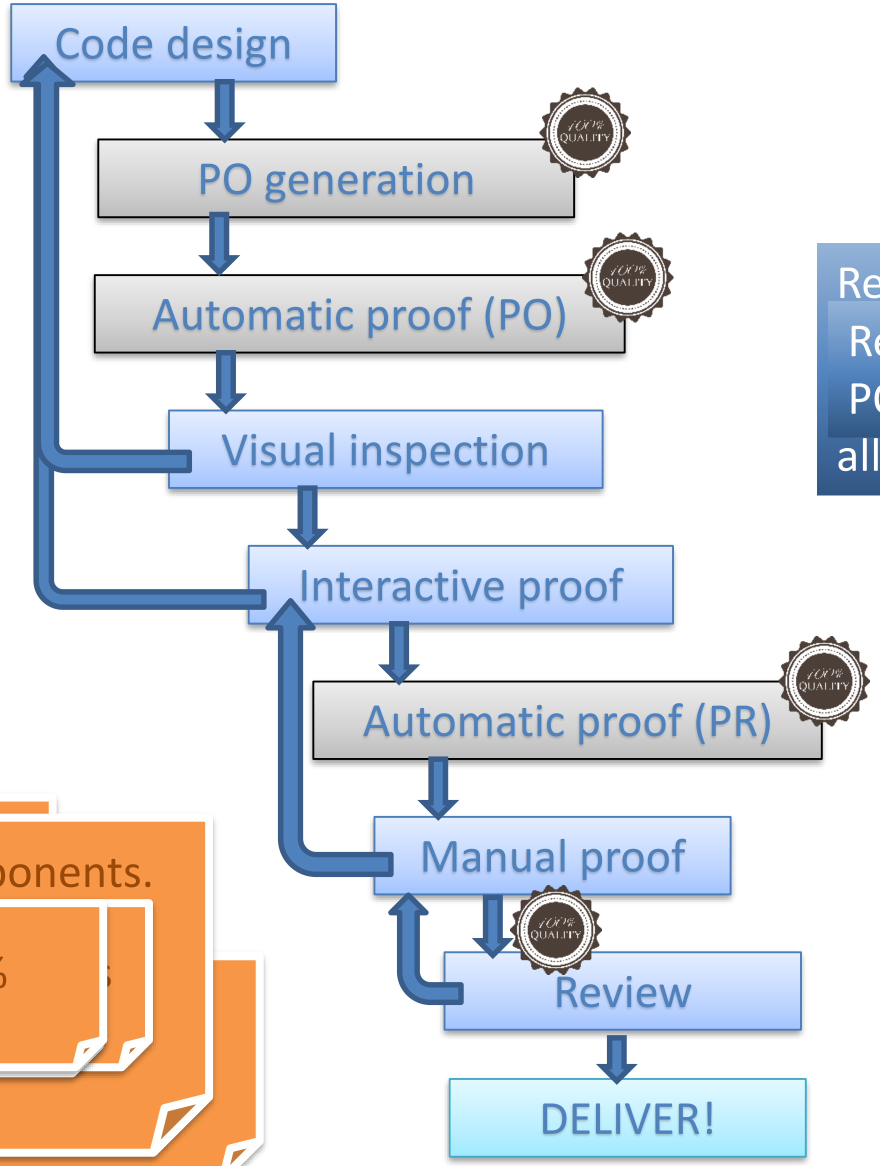
☛ **B method**

☛ **Atelier B**

B method in a nutshell

- A method to design software components
  - J.-R. Abrial. *Assigning programs to meanings*.
- 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

Workflow



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

10-100s components.

≈ 85%

B source

Proof obligations

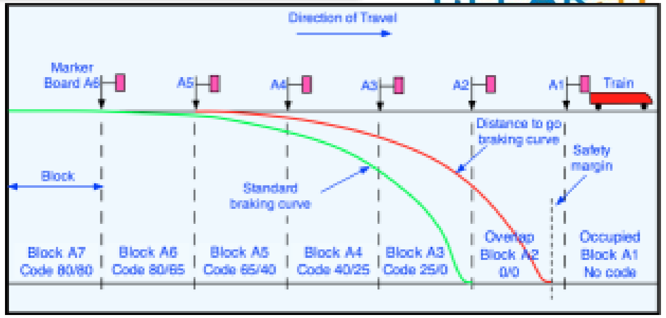
Proof scripts

Tactics

Proof rules

Human proofs

Safety Critical Railway Applications



Braking curves

- ≡ 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.

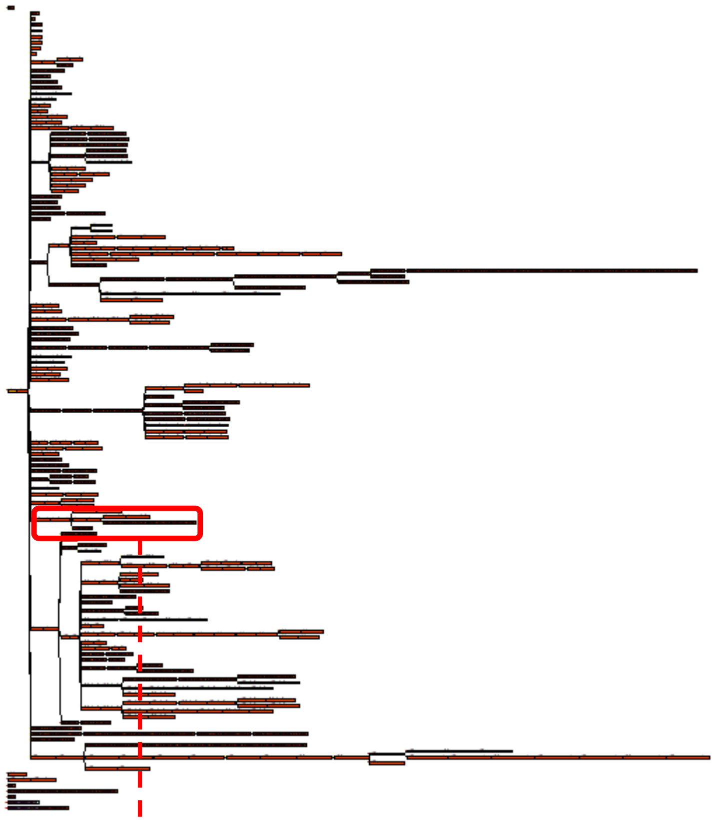
Metrics

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

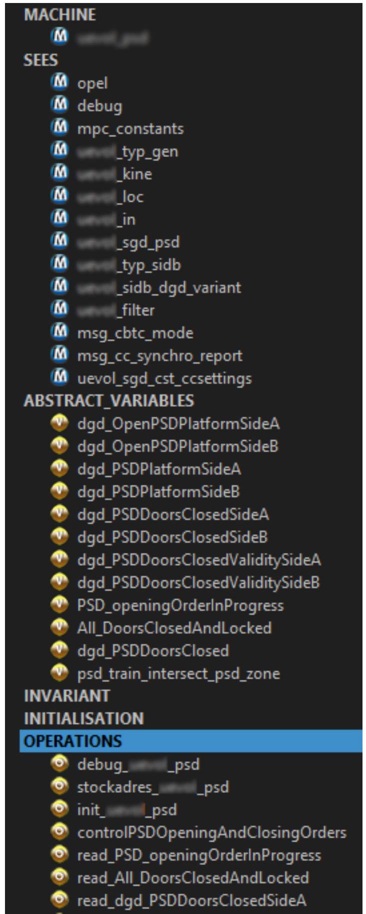
Platform Screen Doors

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

root



Modern Automatic Train Protection Software (2015)



Safety Critical Railway Applications

CLEARSY SYSTEM ENGINEERING

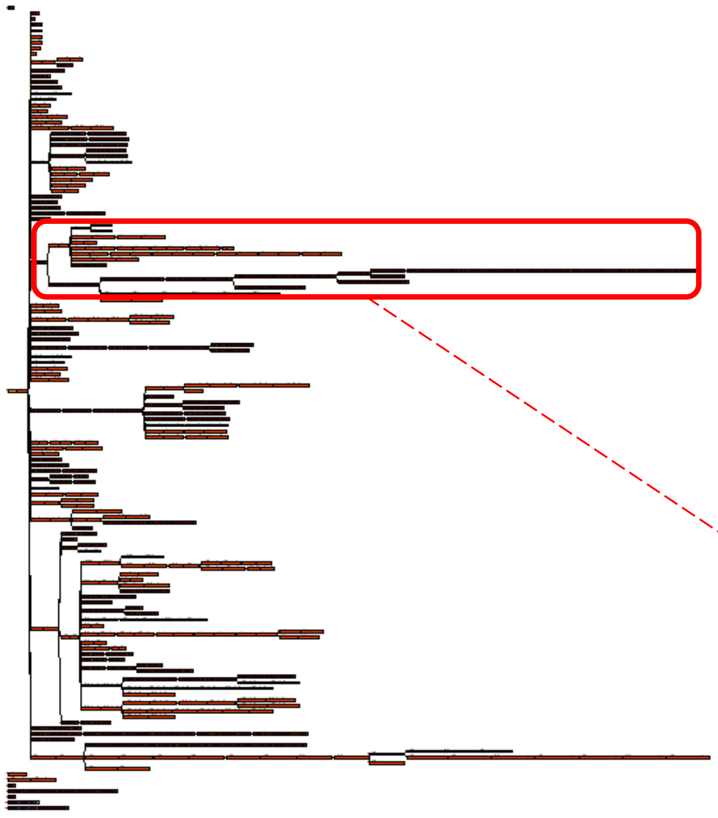
Biggest function: Localization

where is the train ?

Post-condition of one operation:

variables become such as …

root



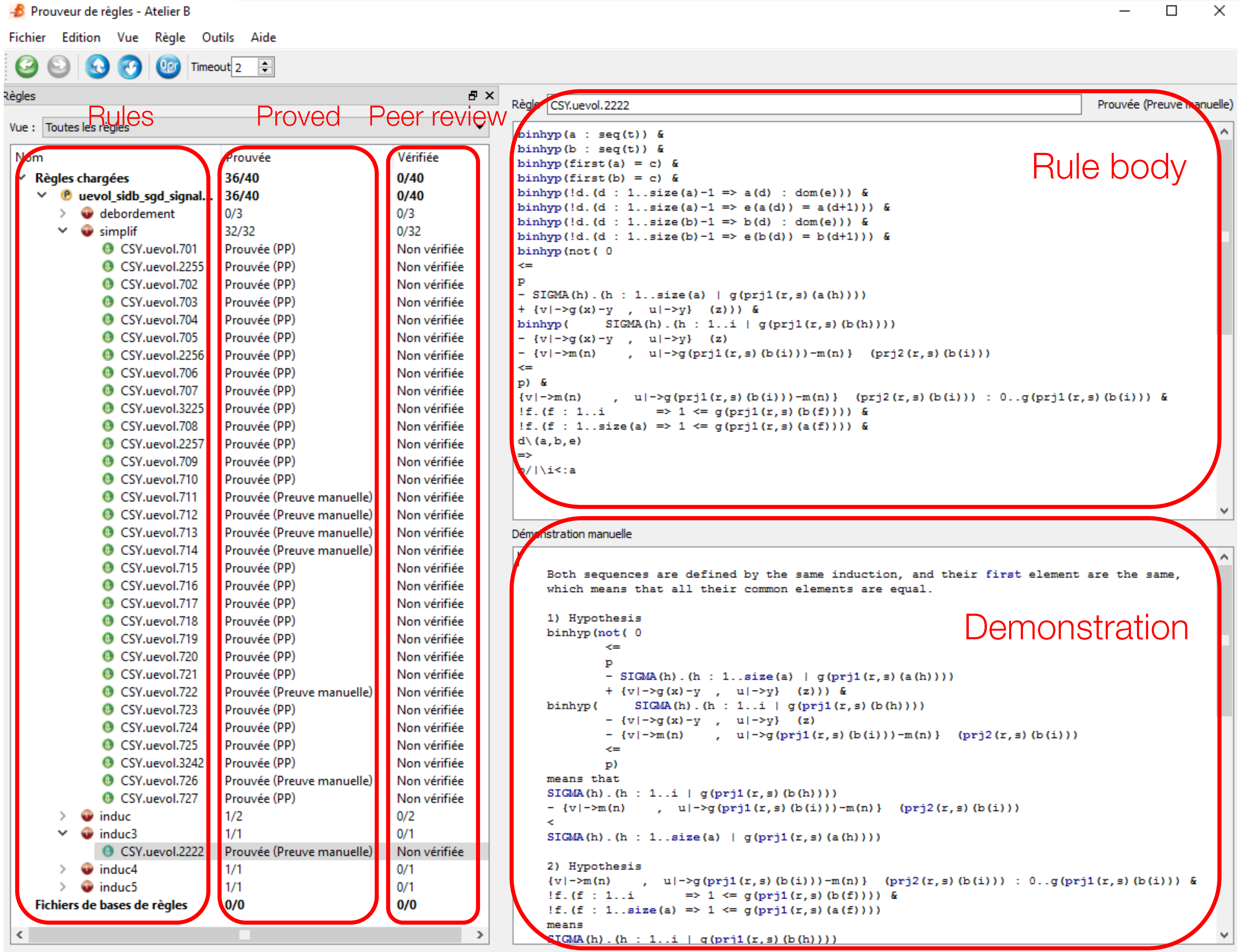
Modern Automatic Train Protection Software
(2015)

```
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)
        & !ii.(ii : 1..size(loc_train_track)-1 => loc_train_track(ii) : dom(sidb_nextBlock))
        & !ii.(ii : 1..size(loc_train_track)-1 => sidb_nextBlock (loc_train_track(ii)) = loc_
        & #aa.(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_rearBlock = c_block_init
            =>loc_rearDir=c_up)
        & ( not (loc_rearBlock = c_block_init)
            =>  loc_rearDir   = { c_cabin1 |-> oppositeDirection(loc_ext2Dir), c_cabin2 |-> c
        & 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_frontDir   = { c_cabin1 |-> loc_ext1Dir, c_cabin2 |-> loc_ext2Dir, c_none |-> c
        & loc_frontAbs   = { c_cabin1 |-> loc_ext1Abs, c_cabin2 |-> loc_ext2Abs, c_none |-> s
    ))
```

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

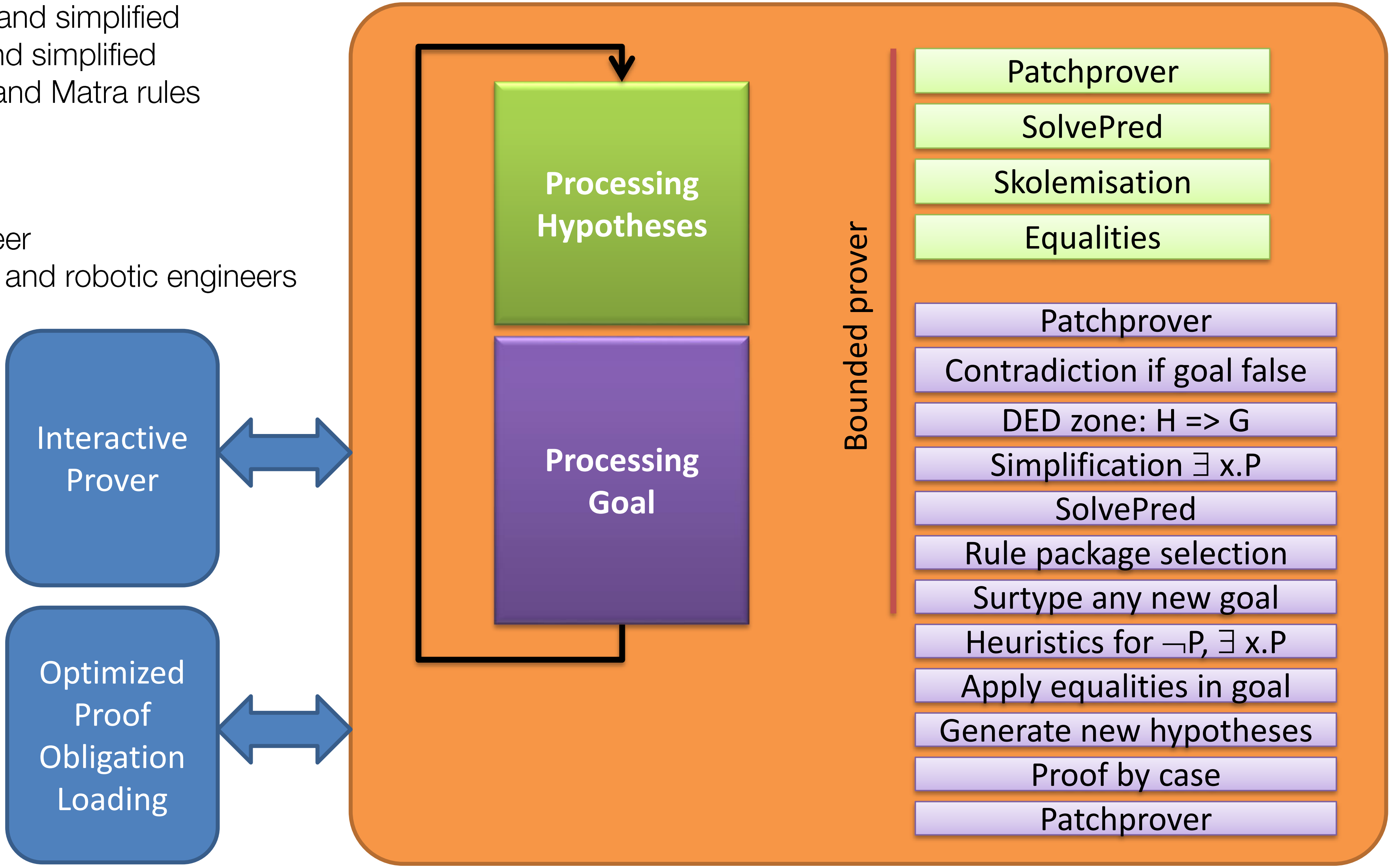
Main Prover

≡ 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

```
IDENT(InRelationXY.11)
binhyp(r: s <-> t) &
binhyp(q: s <-> t)
=>
(r/\q: s <-> t);
```

Leaf rule

```
IDENT(InRelationXY.7)
(dom(b)<<|a: s <-> t) &
(b: s <-> t)
=>
(a<+b: s <-> t);
```

Equivalence rule

```
THEORY SimplifyDisjX IS

    bcall(WRITE: bwritef("Impossible
    =>
    p;

    SimplifyDisjG(A | a | R);

    bcall(RES: bresult(R))
    =>
    SimplifyDisjG(a | a | R);

    bsearch(a, (A or bfalse), B) &
    bcall(RES: bresult(R or B))
    =>
    SimplifyDisjG(A | a | R);

    SimplifyDisjG(A | a | R)
    =>
    SimplifyDisjG(A or a| a | R);

    bsearch(a, (B or bfalse), C) &
    SimplifyDisjG(A | a | R or C)
    =>
    SimplifyDisjG(A or B | a | R)

END
```

≡ Fixed point

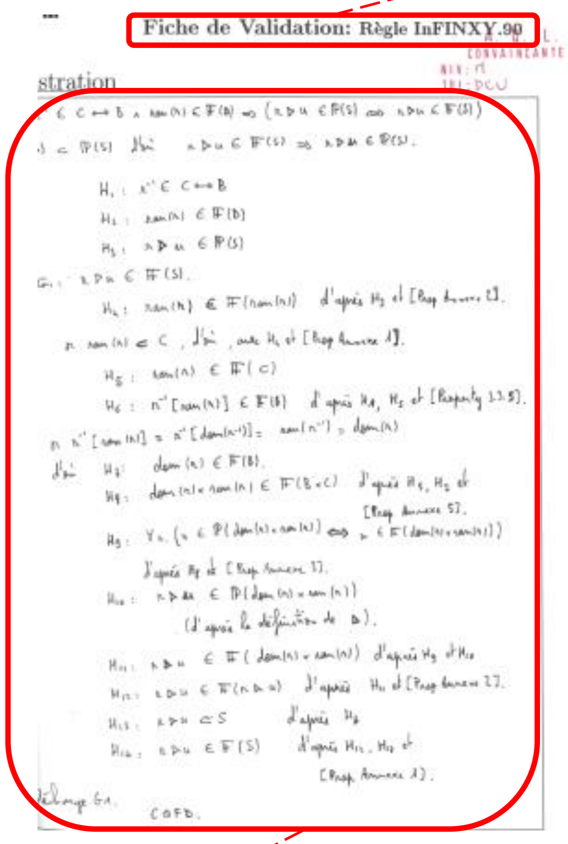
- No proof regression allowed with Atelier B releases
- Since 1998, evolutions are new interactive commands and trigger-able sets of rules

Main Prover

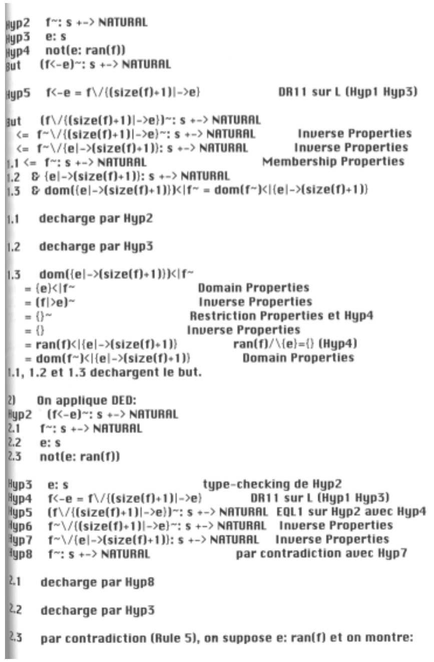
≡ 2 700 rules to validate

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

Rule name



Manual demonstration



```
Hyp2   f~: s +-> NATURAL
Hyp3   e: s
Hyp4   not(e: ran(f))
But    (f<-e)~: s +-> NATURAL

Hyp5   f<-e = f\/{(size(f)+1)|->e}                DR11 sur L (Hyp1 Hyp3)

But    (f\/{(size(f)+1)|->e})~: s +-> NATURAL
  <=  f~\/{(size(f)+1)|->e}~: s +-> NATURAL        Inverse Properties
  <=  f~\/{e|->(size(f)+1)}: s +-> NATURAL         Inverse Properties
1.1 <=  f~: s +-> NATURAL                         Membership Properties
1.2   & {e|->(size(f)+1)}: s +-> NATURAL
1.3   & dom({e|->(size(f)+1)})<|f~ = dom(f~)<|{e|->(size(f)+1)}

1.1    decharge par Hyp2

1.2    decharge par Hyp3

1.3    dom({e|->(size(f)+1)})<|f~
   = {e}<|f~                            Domain Properties
   = (f|>e)~                            Inverse Properties
   = {}~                                Restriction Properties et Hyp4
   = {}                                 Inverse Properties
   = ran(f)<|{e|->(size(f)+1)}          ran(f)/\{e}={} (Hyp4)
   = dom(f~)<|{e|->(size(f)+1)}         Domain Properties
1.1, 1.2 et 1.3 dechargent le but.

2)     On applique DED:
Hyp2   (f<-e)~: s +-> NATURAL
2.1    f~: s +-> NATURAL
2.2    e: s
2.3    not(e: ran(f))

Hyp3   e: s                                       type-checking de Hyp2
Hyp4   f<-e = f\/{(size(f)+1)|->e}                DR11 sur L (Hyp1 Hyp3)
Hyp5   (f\/{(size(f)+1)|->e})~: s +-> NATURAL  EQL1 sur Hyp2 avec Hyp4
Hyp6   f~\/{(size(f)+1)|->e}~: s +-> NATURAL   Inverse Properties
Hyp7   f~\/{e|->(size(f)+1)}: s +-> NATURAL    Inverse Properties
Hyp8   f~: s +-> NATURAL                       par contradiction avec Hyp7

2.1    decharge par Hyp8

2.2    decharge par Hyp3

2.3    par contradiction (Rule 5), on suppose e: ran(f) et on montre:
```

A detailed demonstration

évident (*)

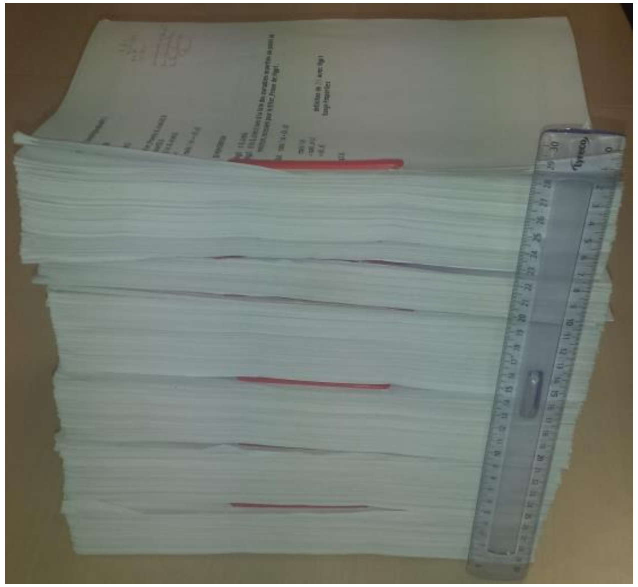
JRA

A concise one !

(*) « Obvious »

≡ The resulting validation forms

- 28 cm



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

```
/** BOOL31 **/
        INFRULE(BOOL31)
        ((v = FALSE) => Q)
=>
        (not(v = TRUE) => Q);
```

```
/** ECTR6 **/
        INFRULE(ECTR6)
        binhyp(l=m) &
        bnot(bgoal(not(L)=>M)) &
        bnot(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 instanciation
- Limited number of rules (116)
- Efficient when the number of hypotheses is low

≡ Added value

- Quickly identify errors

Interactive Proof

- Improve demonstrations efficiency
  - Abstract and reuse demonstrations
  - Fine grained tactics
  - Motto: do not lose any proof work

Patchprover

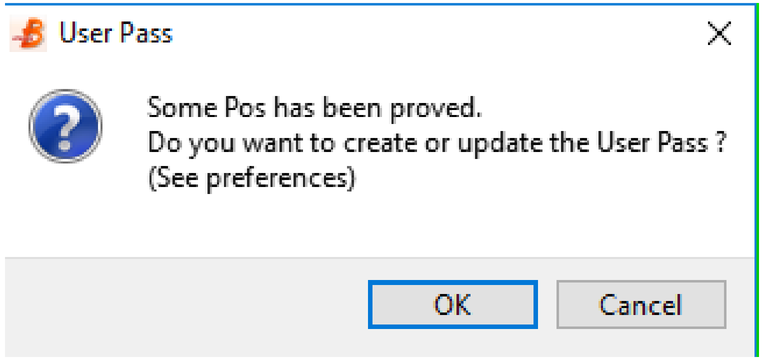
User Rule

User Tactics

Rule packages

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

Operation filter | Goal filter | Interactive commands



Do not lose any proof work

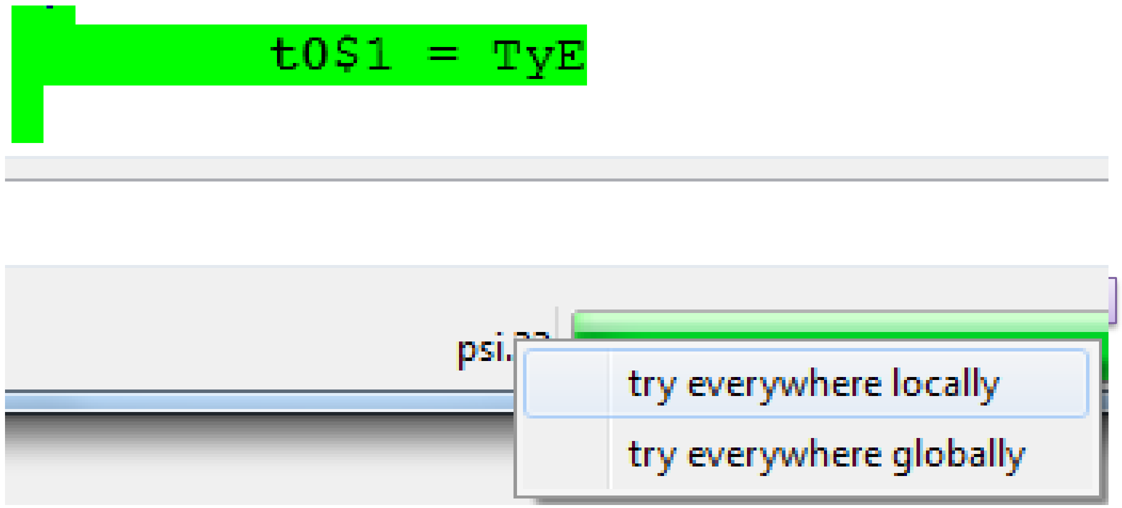
UserPass creation

preview

```
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))
     & ff(0) & dd & ah(e0 = e0$1) & ss & pp(rt.1)
```

Generated tactic from successful proof

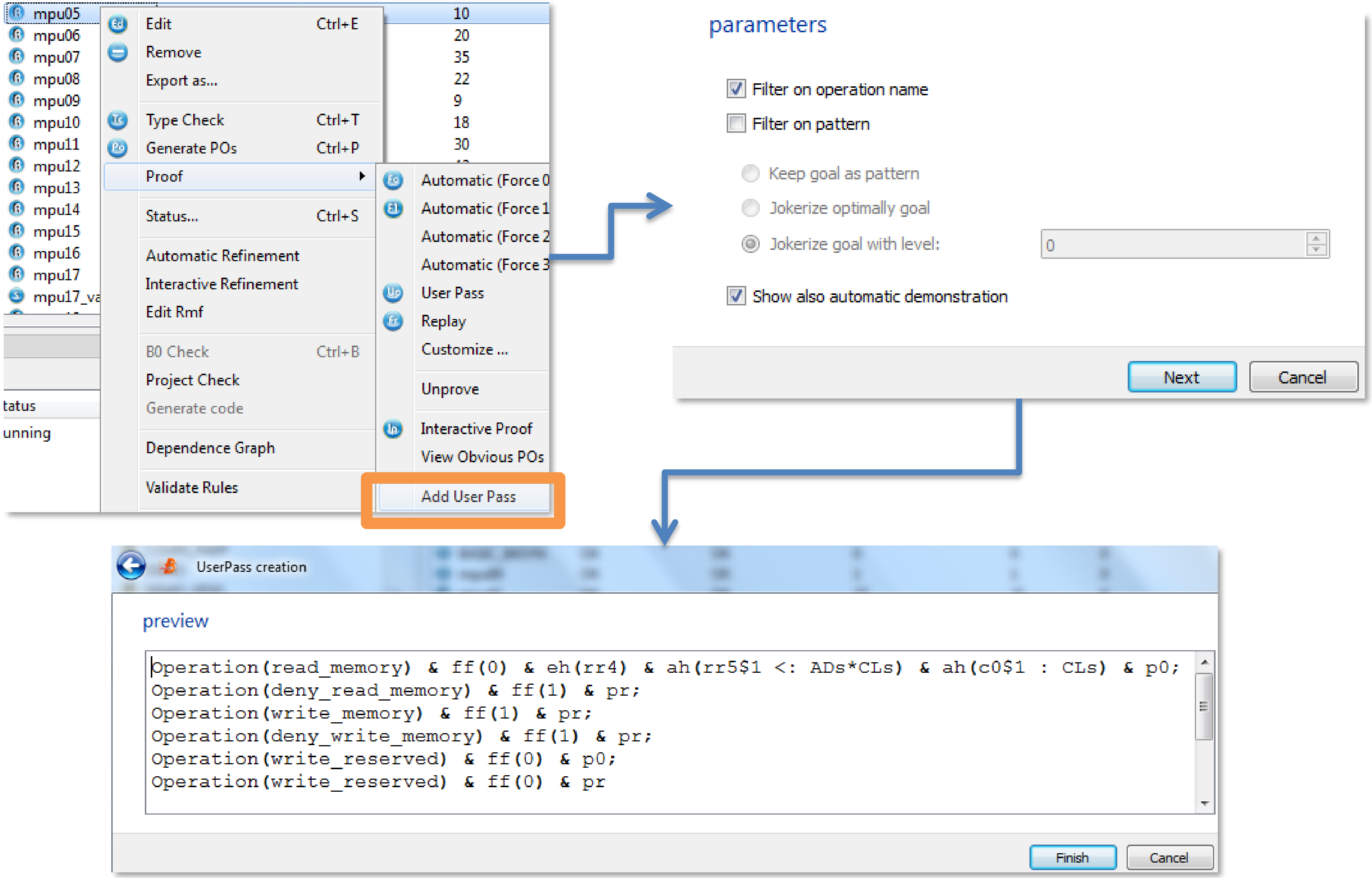
From proof script to tactic



- ≡ as is
- ≡ with minor modifications
- ≡ term abstraction

- ≡ bottom-up: avoid early generalization

From proof script to tactic



Real & Floating Point Numbers

CLEARSY SYSTEM ENGINEERING

≡ 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 **ε**

≡ 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).

Unified	Integer	Real	Float
x <= y	x <= y	x rle y	x <=. y
x < y	x < y	x rlt y	x <. y
x >= y	x >= y	x rge y	x >=. y
x > y	x > y	x rgt y	x >. y
x + y	x + y	x rplus y	x +. Y
- x	- x	0.0 rminus x	-. X
x - y	x - y	x rminus y	x <=. y
x * y	x * y	x rmul y	x *. Y
x / y	x / y	x rdiv y	x /. y
x ** y	x ** y	x rpow y	***Invalid***
min(x)	min(x)	rmin(x)	***Invalid***
max(x)	max(x)	rmax(x)	***Invalid***
SIGMA(x).(y | z)	SIGMA(x).(y | z)	rSIGMA(x).(y | z)	***Invalid***
PI(x).(y | z)	PI(x).(y | z)	rPI(x).(y | z)	***Invalid***

Real & Floating Point Numbers

≡ 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 ) =
    PRE
        v_deltaT : tFloat32 &
        p_Velocity_InitVALUE : tVect3_64 &
        p_Position_InitVALUE  : tVect3_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
```

Filter

Show errors only (0)

Expand

MACHINE
- nav

SEES
- constant
- type
- lib_diab

ABSTRACT_CONSTANTS
- LIB_Mat64_compute_matCosdirQ
- LIBNAV_velocity_calculate_gravity
- LIBNAV_position_calculate_omega_vt_v
- LIBNAV_velocity_calculate_coriolis
- LIBNAV_mech_calculate_cmd

PROPERTIES

ABSTRACT_VARIABLES
- V_nav_data
- V_LIBNAV_Sum_corr_omega_vi_v
- V_ptf_data
- V_Q_vt

INVARIANT

INITIALISATION

OPERATIONS
- LIBNAV_types_init_data
- LIBPTF_types_init_data
- IR_T1_CINAV_nav_init

Data validation

Formal Data Validation

CLEARSY SYSTEM ENGINEERING

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

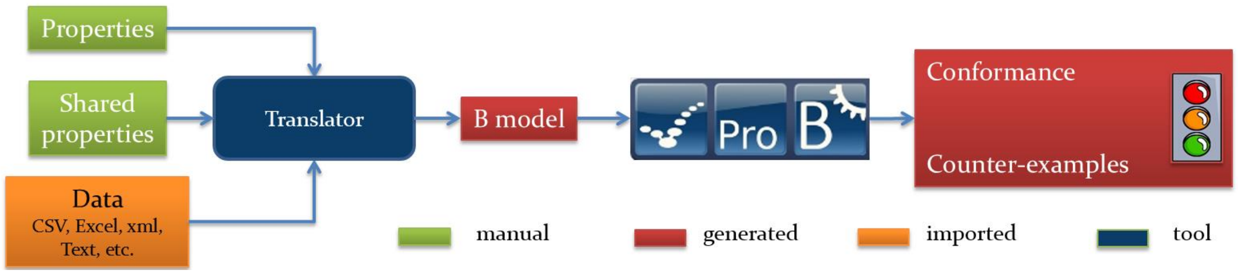
	A	B	C	D	E	F	G	H	I
1	Name	ID	IP	Type	UpLink	DownLink	Length	GPS 1	GPS 2
2	Route_tx_001	243		R	Route_tx_005	Route_vx_002	345		
3	Route_vx_002	128		R	Route_vx_002	EndLine_000	128		
4	Switch_w_003	256	192.16.4.55	S	Route_vx_128	Route_tx_006	23		
5	Relay_s_004	12	192.16.4.10	Y				N 50.85 963	O 6.84 201
6	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

Are they
- Consistent ?
- Correct ?
- Safe ?

Up to 100,000+ raw data chunks

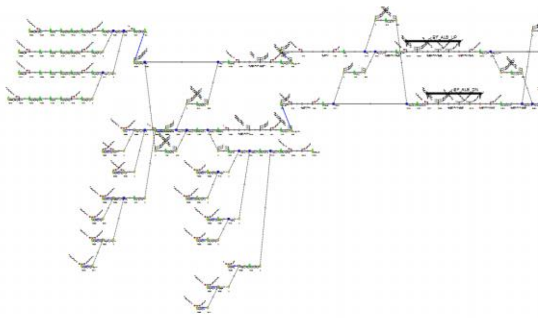
Formal Data Validation in the Railways, T. Lecomte, SSS 2016
Using B and ProB for Data Validation Projects, D. Hansen, D. Scheiner, M. Leuschel, book chapter 2016
From Animation to Data Validation: The ProB Constraint Solver 10 Years On, M. Leuschel, J. Bendisposto, I. Dobrikov, S. Krings, D. Plagge,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

Formal Data Validation



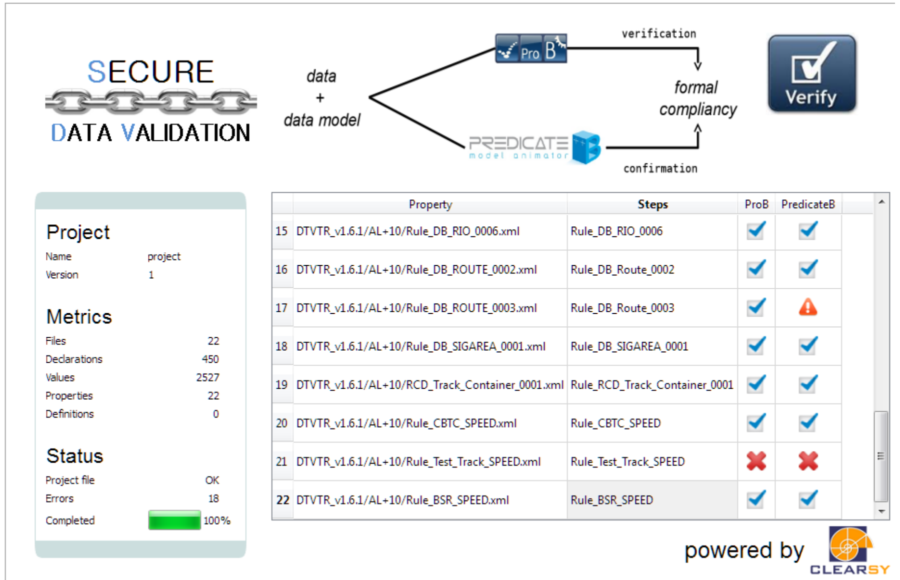
≡ Consistency, correctness, safety

- 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, Ryiad, Mecca

Formal Data Validation

Example of Properties

- Domain Specific Language to express properties
- Direct translation from the DSL to B
- This rule is simple

```
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

Connecting New Provers

≡ ProB

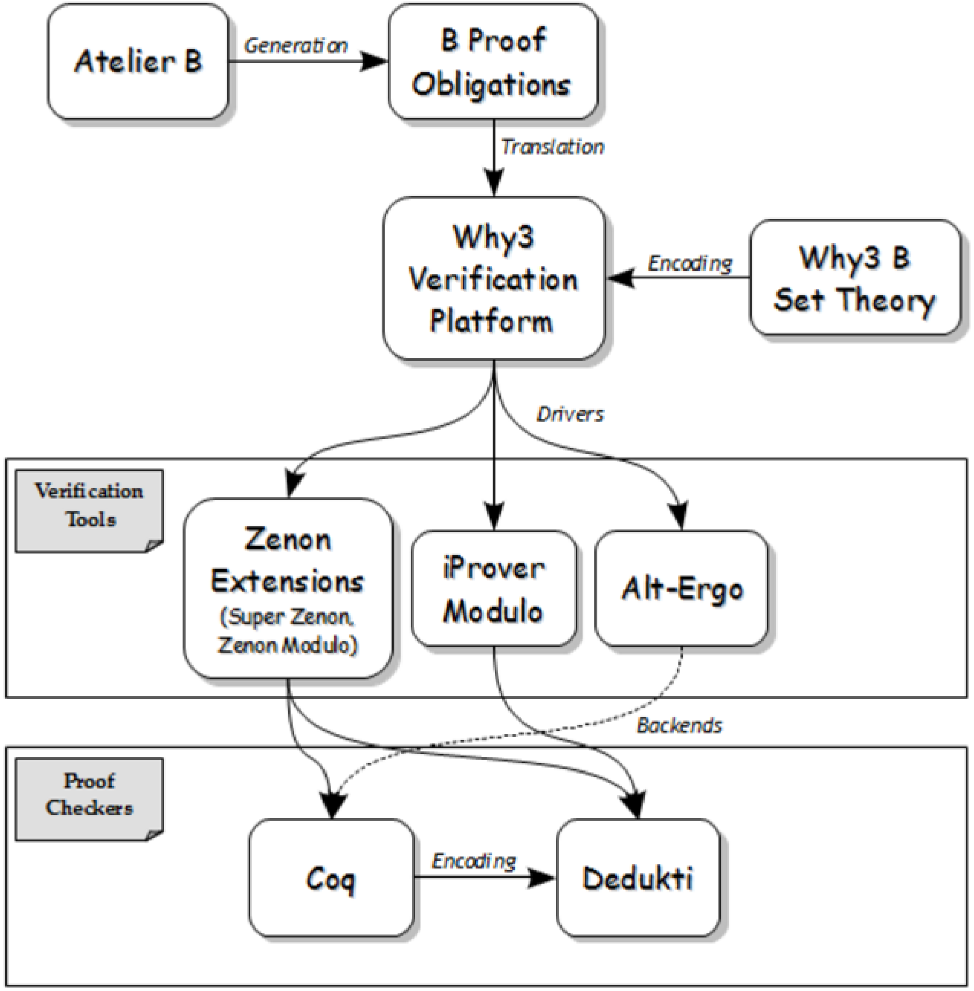
- Added to Atelier B 4.3.1 as an interactive command.
  - Syntax: prob(n) or prob(n | T).
    - T: timer in seconds,
    - n=1 selects all hypotheses that have a symbol in common with the goal
  - Add resource: ATB*PR*ProB_Path:C:\<path>\probcli.exe
- Funded by Alstom

≡ Multiprover platform

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

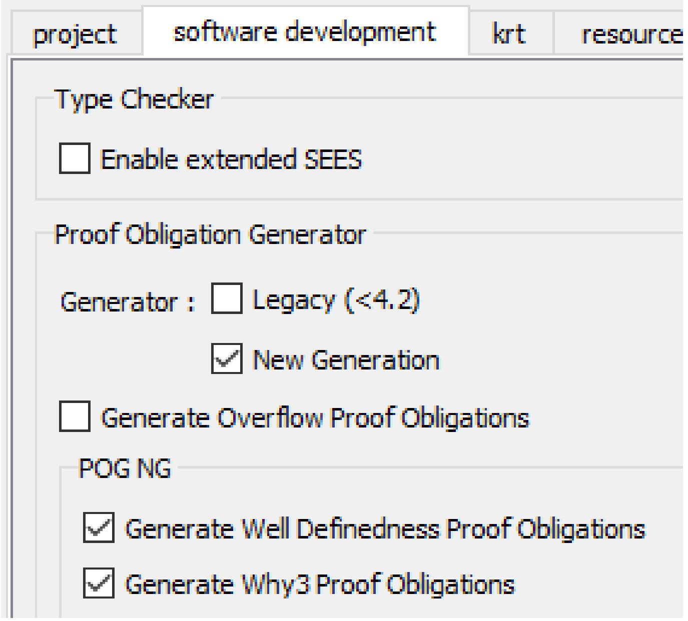
≡ “Teaching an old dog new tricks” - Drudges

- 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

iapa

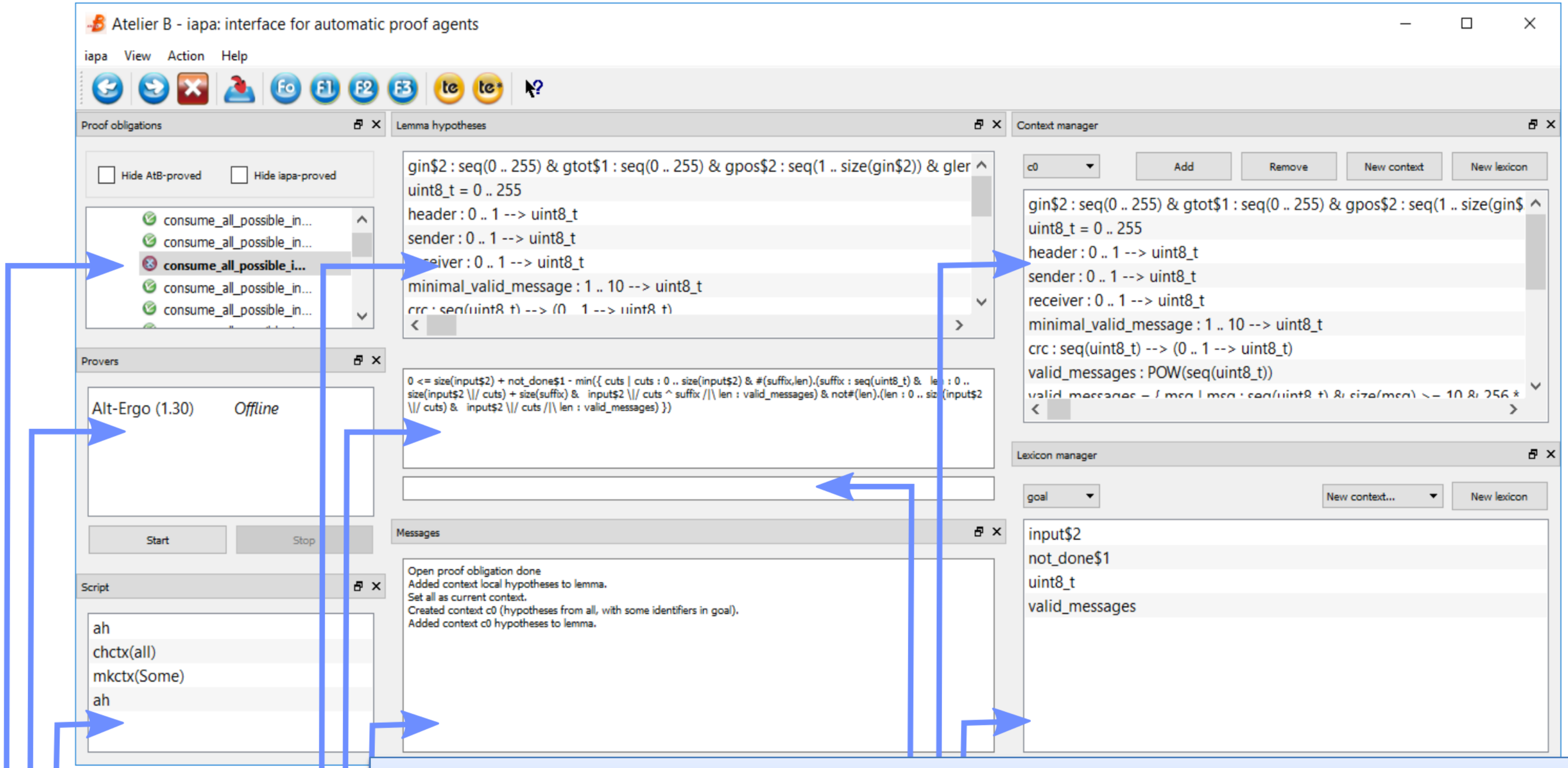



```
goal g_0 :
    forall obf_1: set int, obf_2: set int, obf_3: set int, obf_4: (set int), obf_5: int, obf_6: (set int),
    obf_7: int, obf_8: (set int), obf_9: int, obf_10: (set (int,int)), obf_11: (set (int,int)), obf_12: (
    set (int,int)), obf_13: (set (int,int)), obf_14: (set (int,int)), obf_15: (set (int,int)), obf_23: (
    set int), obf_25: (set int), obf_26: (set int), obf_16: (set int), obf_17: (set int), obf_18: (set int)
    , obf_19: (set int), obf_20: (set int), obf_21: bool.
    (define1 ) /\ (define2 obf_1 obf_2 obf_3 obf_4 obf_5 obf_6 obf_7 obf_8 obf_9 obf_10 obf_11 obf_12
    obf_13 obf_14 obf_15 ) /\ (define6 ) /\ (define7 ) /\ (define9 ) /\ (define10 ) /\ (define11 obf_23
    obf_4 obf_25 obf_26 ) /\ (define3 obf_16 obf_4 obf_17 obf_6 obf_18 obf_8 obf_19 obf_20 obf_21 ) /\ (
    lh_2_0 )
    ->
    true /\
((lh_2_1 obf_4 ) -> (exists obf_27: (set int).(mem obf_27 (power obf_4))))
```

Example of Why3 fragment

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

iapa



user actions

automatic provers

proof obligations

se

Interfacing Automatic Proof Agents in Atelier B: Introducing "iapa"

Lilian Burdy, David Deharbe, Étienne Prun.

Proceedings F-IDE 2016, Electronic Proceedings in Theoretical Computer Science 240.

The drudges of the interactive prover

≡ Problem statement

- « obvious » sub-goals in the interactive prover

≡ Solution approach

- Have third-party provers handle sub-goals

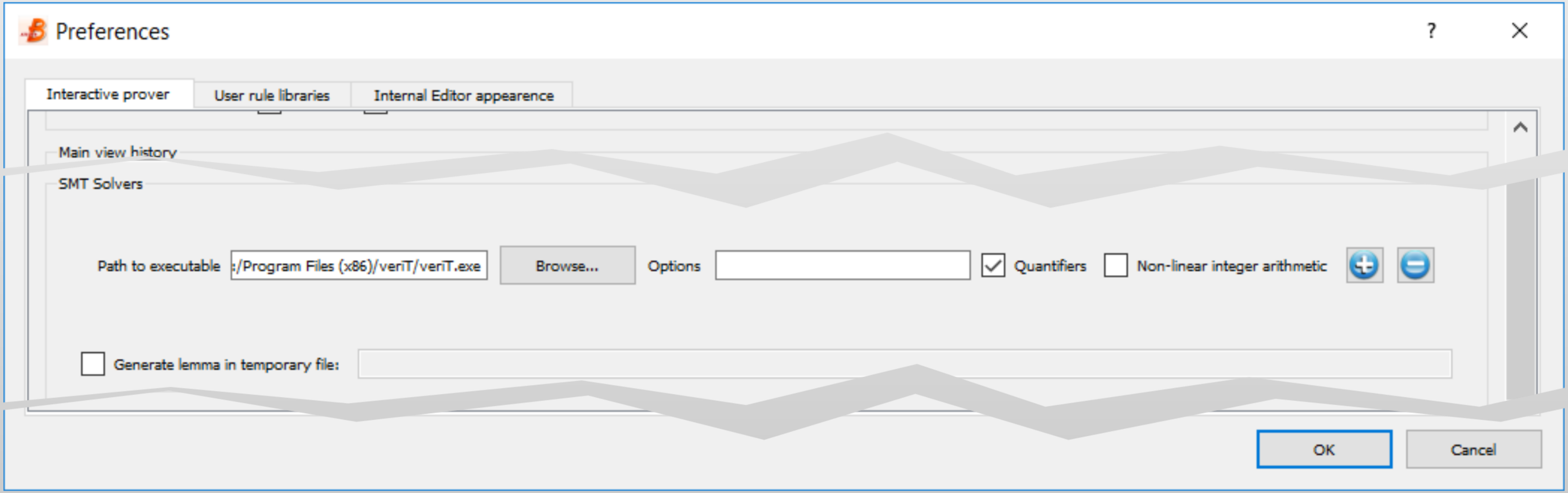
≡ Better solution approach

- Have third-party provers handle sub-goals
- and have the result translated back in proof rules and applications

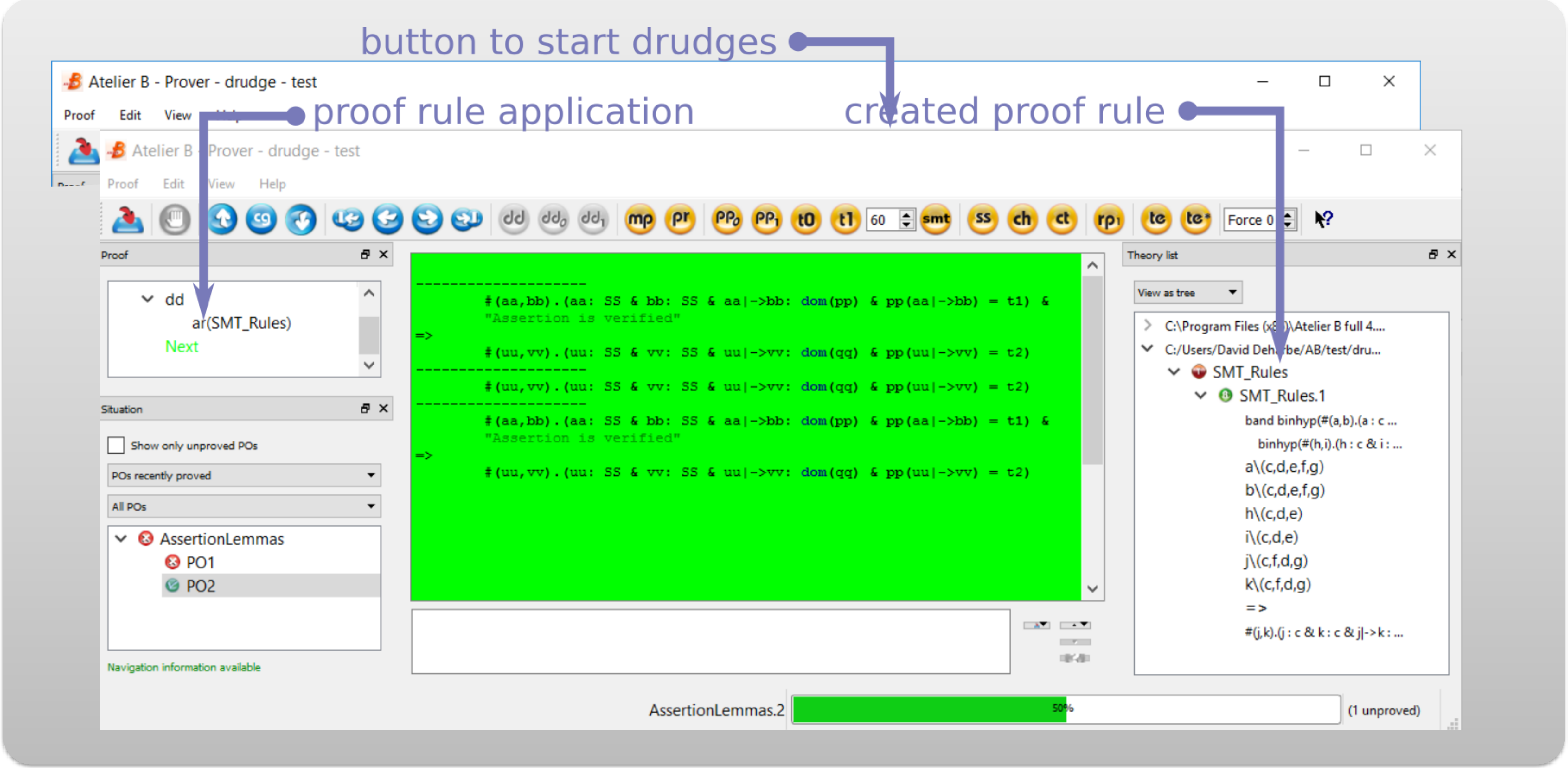
≡ Better solution implementation

- 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



The drudges of the interactive prover

NAT = 0..2147483647
INT = -2147483647..2147483647 ∧
btrue ∧SS∈ FIN(ℤ) ∧
¬(SS = ∅) ∧
TT∈ FIN(ℤ) ∧
¬(TT = ∅) ∧
pp∈ SS*SS ⇸ TT ∧
qq∈ SS*SS ⇸ TT ∧
t1∈ TT ∧
t2∈ TT ∧
∃(s1,s2).(s1∈ SS ∧ s2∈ SS ∧ s1↦s2∈ dom(pp) ∧ p
∃(s1,s2).(s1∈ SS ∧ s2∈ SS ∧ s1↦s2∈ dom(qq) ∧
∃(aa,bb).(aa∈ SS ∧ bb∈ SS ∧ aa↦bb∈ dom(pp) ∧
⇒
∃(aa,bb).(aa ∈ SS ∧ bb ∈ SS ∧ aa ↦ bb ∈ dom(pp
∃(uu,vv).(uu ∈ SS ∧ vv ∈ SS ∧ uu ↦ vv ∈ dom(q

```
(set-option :print-success false)
(set-option :produce-unsat-cores true)
(set-option :produce-proofs true)
(set-logic UF)
(declare-sort U 0)
(declare-fun TRUE () U)
(c
(c
..
(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 (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)
```

veriT stable2016 - the SMT-solver veriT (UFRN/LORIA).
unsat
(h11 h13 goal)

Searching the most generic rule

$\exists$(s1,s2).(s1$\in$ SS $\wedge$ s2$\in$ SS $\wedge$ s1$\mapsto$s2$\in$ dom(pp) $\wedge$ pp(s1$\mapsto$s2) = t1) $\Rightarrow$
$\exists$(s1,s2).(s1$\in$ SS $\wedge$ s2$\in$ SS $\wedge$ s1$\mapsto$s2$\in$ dom(qq) $\wedge$ pp(s1$\mapsto$s2) = t2) $\wedge$
$\exists$(aa,bb).(aa$\in$ SS $\wedge$ bb$\in$ SS $\wedge$ aa$\mapsto$bb$\in$ dom(pp) $\wedge$ pp(aa$\mapsto$bb) = t1)
$\Rightarrow$
$\exists$(aa,bb).(aa $\in$ SS $\wedge$ bb $\in$ SS $\wedge$ aa $\mapsto$ bb $\in$ dom(pp) $\wedge$ pp(aa $\mapsto$ bb) = t1) $\Rightarrow$
$\exists$(uu,vv).(uu $\in$ SS $\wedge$ vv $\in$ SS $\wedge$ uu $\mapsto$ vv $\in$ dom(qq) $\wedge$ pp(uu $\mapsto$ 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.

Searching the most generic rule

```
...
(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