

Formal Verification in the Industry

```

Force(0)
dd
  ah((0..morrow-1<|Schedule)~[task]) = (0..clock0-1<|Schedule)~[task]V((time<|Schedule)~[task])
  ah(time+1<=morrow)
  ah(clock0+1<=time)
  ah(not(clock0..morrow\dom(Schedule) = {})) => clock0..morrow\dom(Schedule) = {time}
  ah(0<=clock0)
  pp(rp.0)
dd
  eh((0..morrow-1<|Schedule)~[task]),_h,Goal)
  ar(DMS_SIG.2,Goal)
  ah(clock0+1<=time)
  pp(rp.0)
  ah((dom(spans0V{clock0}->time)<|(log0V{morrow}->victor)))~[task] = (dom(spans0<|log0)~[task])V((c
  ah(time+1<=morrow)
  ah(clock0+1<=time)
  ah(dom(spans0) = dom(log0)-{clock0})
  ah(dom(log0) <: 0..clock0)
  pp(rp.0)
dd
  eh((dom(spans0V{clock0}->time)<|(log0V{morrow}->victor)))~[task]),_h,Goal)
  ar(DMS_SIG.2,Goal)
  ah(dom(spans0) = dom(log0)-{clock0})
  pp(rp.0)
  ah(SIGMA(time$0).(time$0: (dom(spans0<|log0)~[task]) | (spans0V{clock0}->time))(time$0)-
  ar(DMS_SIG.3,Once)
  mp
  dd
    eh(SIGMA(time$0).(time$0: (dom(spans0<|log0)~[task]) | (spans0V{clock0}->time))(t
    ah(SIGMA(time$0).(time$0: ((clock0<|log0)~[task]) | (spans0V{clock0}->time))(tir
    ar(DMS_SIG.3,Once)
    ah(dom(spans0) = dom(log0)-{clock0})
    ah(spans0: INTEGER <-> INTEGER)
    pp(rp.0)
  dd
    eh(SIGMA(time$0).(time$0: ((clock0<|log0)~[task]) | (spans0V{clock0}->time))(t
    ah(SIGMA(time$0).(time$0: ((clock0<|log0)~[task]) | time-time$0)+(term0<+((
    dc(task = elected0)
    eh(elected0,task,Goal)
    ah(((clock0<|log0)~[task]) = {clock0})
    eh(task,_h,Goal)
    eh(elected0,_h,Goal)
    mp
  dd
    eh(((clock0<|log0)~[task]),_h,Goal)
    ar(SimplifyIntSIGXY.12,Goal)
    ah((term0<+((Phantom)<|task|->leftspan)V(Schec
    ah(task: dom((Phantom)<|task|->leftspan)V(S
    mp
    dd
      ar(SimplifyRelFonXY.8,Goal)
      ah(task: dom((Phantom)<|task|->le
      mp
      dd
        ar(SimplifyRelFonXY.14,Goal)
        ah(not(task = Phantom))
        mp
        pp(rp.0)
    dd
      eh((term0<+((Phantom)<|task|->leftspan)\
      ar(DMS_SIG.1,Goal)
      ah(not(term0(elected0) = 0))
      ah(elected0: Tasks)
      eh(elected0,task,Goal)
      ah(task: Tasks)
      ah(not(term0(Schedule[[time]])) =
      ah(dom(term0) = Tasks)
      ah(ran(Schedule) = Tasks)

```

```

  eh(elected0,task,Goal)
  pp(rp.0)
  ah(leftspan = term0(elected0)-(time-clock0))
  ah(elected0: Tasks)
  eh(elected0,task,Goal)
  mp
  eh(elected0,task,Goal)
  mp
dd
  dc(task: Schedule[[time]])
  dd
    ah(task: dom((Phantom)<|elected0|->leftspan)V(Schedule[[time]]<|Deadline))
    mp
    dd
      ar(SimplifyRelFonXY.8,Goal)
      ah(task: dom(Schedule[[time]]<|Deadline))
      mp
      dd
        ar(SimplifyRelFonXY.15,Goal)
        ar(SimplifyRelFonXY.3,Goal)
        ah(((time)<|Schedule)~[task]) = {time})
        ah(task: Schedule[[time]])
        pp(rp.0)
      dd
        eh(((time)<|Schedule)~[task]),(time,Goal)
        ar(SimplifyIntSIGXY.12,Goal)
        ar(DMS_SIG.1,Goal)
        ah(not(task = elected0))
        eh(elected0,_h,Goal)
        mp
      ss
        ah(not(term0(Schedule[[time]]) = {})) => term0(Schedule[[time]]) = {0})
        ah(task: Schedule[[time]])
        ah(ran(Schedule) = Tasks)
        ah(ran(Schedule) = Tasks)
        ah(dom(term0) = Tasks)
        pp(rp.0)
    dd
      ah(not(task: dom((Phantom)<|elected0|->leftspan)V(Schedule[[time]]<|Deadline)))
      mp
      dd
        ar(SimplifyRelFonXY.7,Goal)
        ar(DMS_SIG.1,Goal)
        ah(not(task: Schedule[[time]]))
        pp(rp.0)
        ar(DMS_SIG.1,Goal)
        ah(not(task = elected0))
        eh(elected0,_h,Goal)
        mp
        mp
dd
  ah(SIGMA(time$0).(time$0: (dom(spans0<|log0)~[task]) | spans0(time$0)-time$0)+SIGMA(time$0).(time$0: ((clock0<|log0)~[task]) | time-time...
  mp
  dd
    eh(SIGMA(time$0).(time$0: (dom(spans0<|log0)~[task]) | spans0(time$0)-time$0)+SIGMA(time$0).(time$0: ((clock0<|log0)~[task]) | ti...
    eh(SIGMA(time$0).(time$0: ((clock0<|log0)~[task]) | time-time$0)+(term0<+((Phantom)<|elected0|->leftspan)V(Schedule[[time]]...
    ph(task,task: Tasks => 0 = SIGMA(time).(time: (dom(spans0<|log0)~[task]) | spans0(time)-time)+term0(task)-SIGMA(time)...
    mp
    ah(SIGMA(time).(time: (0..clock0-1<|Schedule)~[task]) | Deadline(task)) = SIGMA(time$0).(time$0: (0..clock0-1<|Schedule)~[t...
    ar(DMS_SIG.4,Once)
    dd
      eh(SIGMA(time).(time: (0..clock0-1<|Schedule)~[task]) | Deadline(task)),_h,Goal)
      ah(SIGMA(time).(time: (dom(spans0<|log0)~[task]) | spans0(time)-time) = SIGMA(time$0).(time$0: (dom(spans0)...
      ar(DMS_SIG.5,Once)
      dd
        eh(SIGMA(time).(time: (dom(spans0<|log0)~[task]) | spans0(time)-time),_h,Goal)
        mp

```

≡ 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

👉 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

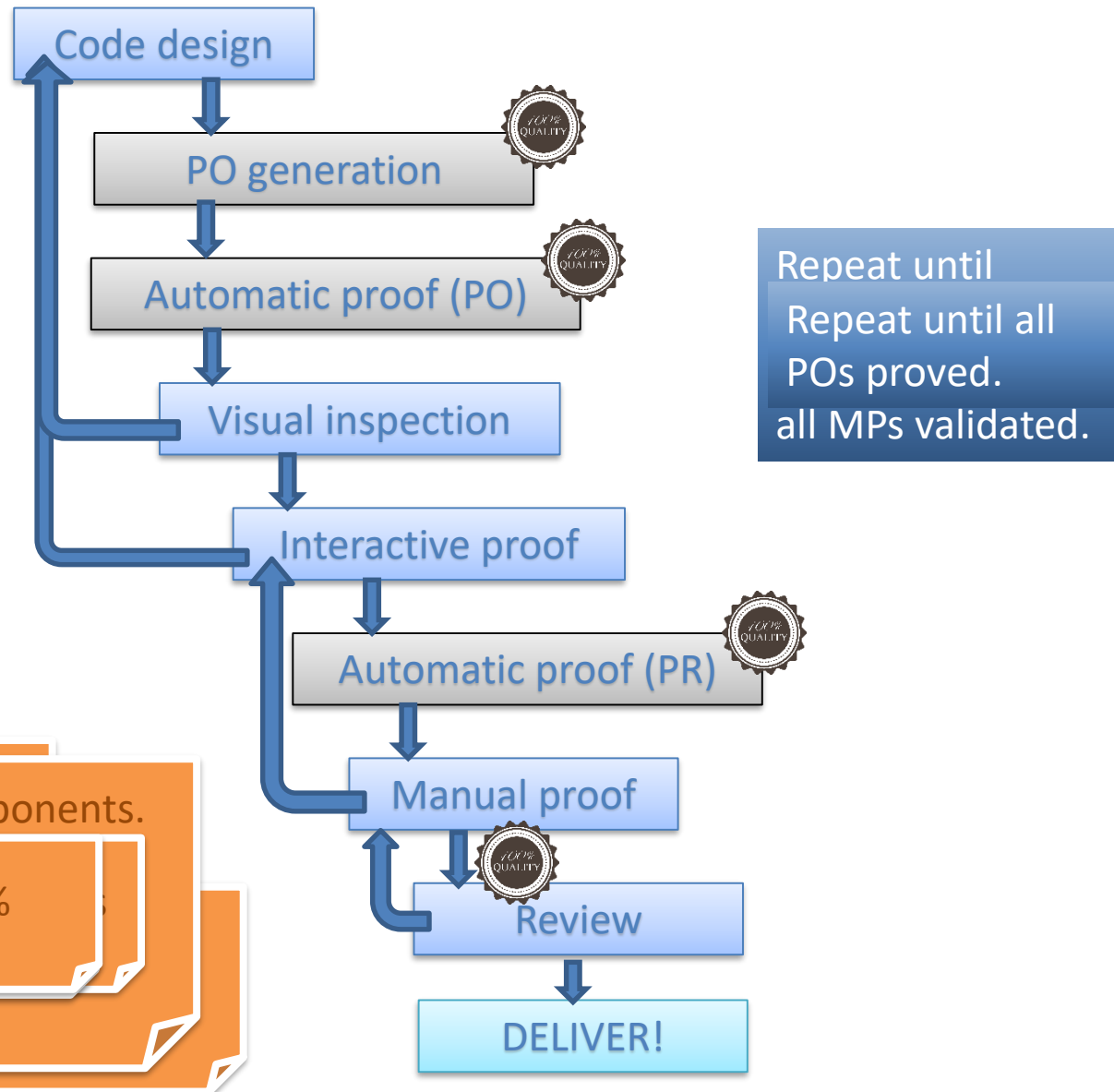
Formal verification: refinements comply with specification, well-definedness, termination

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

≡ Tool support: Atelier-B (25 year old)

≡ Spin offs: Event B, data validation

Workflow



B source

Proof
obligations

Proof scripts

Tactics

Proof rules

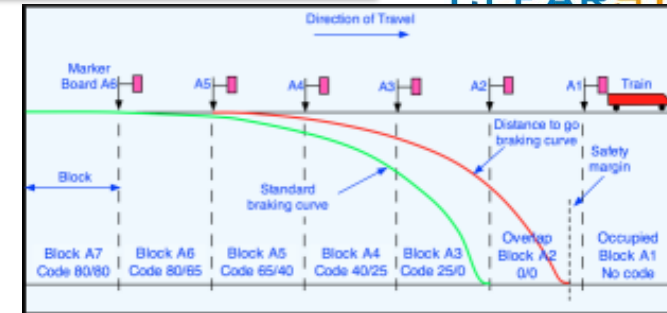
Human proofs

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)

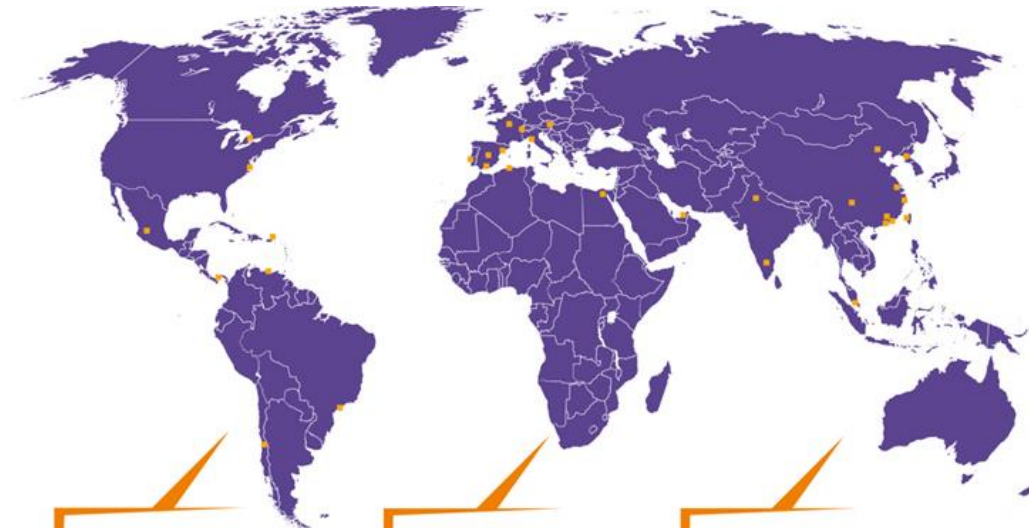


Braking curves

≡ 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



NEW YORK
SAN JUAN
MEXICO
CARACAS
SANTIAGO
SAO PAULO
PANAMA
TORONTO

LAUSANNE
MILANO
BARCELONA
MADRID
LISBON
ALGIERS
CAIRO
BUDAPEST
PARIS
MALAGA
DUBAI

BENGALORE
DELHI
SEOUL
BEIJING
SHANGAI
HONK-KONG
SINGAPOUR
NINGBO
TAICHUNG
KUNMING
SHENZHEN
GUANGHOU

Safety Critical Railway Applications

Top level implementation

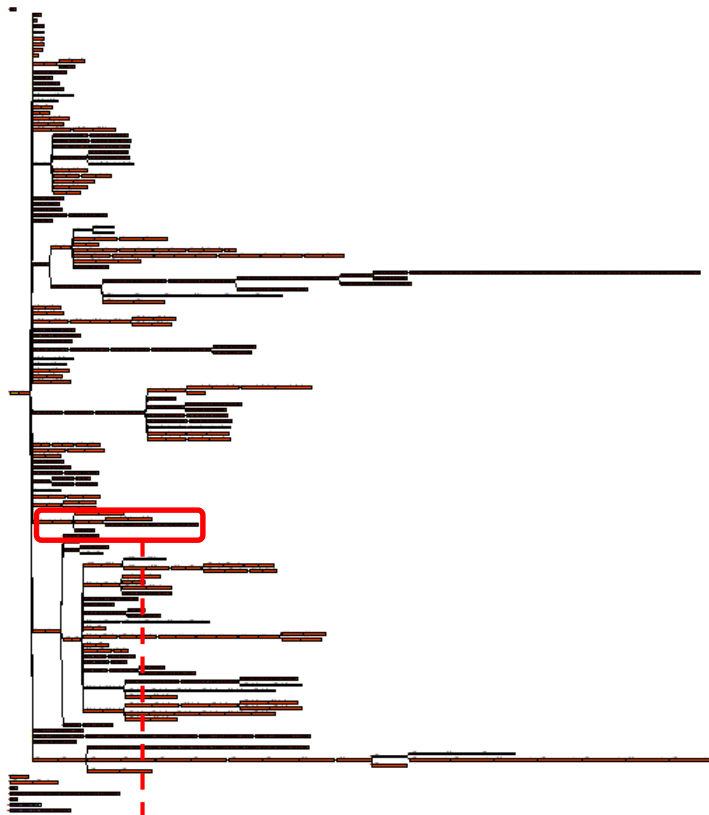
- Imports 55 components
- Specify top level one-cycle function:
 - Compute location, manage kinetic energy, control PSD, trigger emergency braking, etc.

Metrics

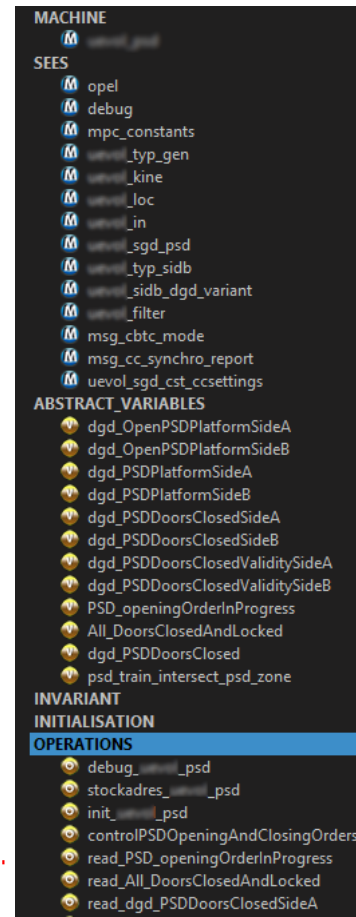
- 233 machines, 50 kloc
- 46 refinements, 6 kloc
- 213 implementations, 45 kloc
- 23 000 proof obligations
- 3 000 added user rules

Platform Screen Doors

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



Modern Automatic Train Protection Software
(2015)



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)

```
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

Prouveur de règles - Atelier B

Fichier Edition Vue Règle Outils Aide

Timeout 2

Règles

Vue : Toutes les règles

Rules	Proved	Peer review
Règles chargées		
uevol_sldb_sgd_signal	36/40	0/40
> debordement	36/40	0/40
> simplif	0/3	0/3
CSY.uevol.701	32/32	0/32
CSY.uevol.702	Prouvée (PP)	Non vérifiée
CSY.uevol.703	Prouvée (PP)	Non vérifiée
CSY.uevol.704	Prouvée (PP)	Non vérifiée
CSY.uevol.705	Prouvée (PP)	Non vérifiée
CSY.uevol.706	Prouvée (PP)	Non vérifiée
CSY.uevol.707	Prouvée (PP)	Non vérifiée
CSY.uevol.708	Prouvée (PP)	Non vérifiée
CSY.uevol.709	Prouvée (PP)	Non vérifiée
CSY.uevol.710	Prouvée (PP)	Non vérifiée
CSY.uevol.711	Prouvée (Preuve manuelle)	Non vérifiée
CSY.uevol.712	Prouvée (Preuve manuelle)	Non vérifiée
CSY.uevol.713	Prouvée (Preuve manuelle)	Non vérifiée
CSY.uevol.714	Prouvée (Preuve manuelle)	Non vérifiée
CSY.uevol.715	Prouvée (PP)	Non vérifiée
CSY.uevol.716	Prouvée (PP)	Non vérifiée
CSY.uevol.717	Prouvée (PP)	Non vérifiée
CSY.uevol.718	Prouvée (PP)	Non vérifiée
CSY.uevol.719	Prouvée (PP)	Non vérifiée
CSY.uevol.720	Prouvée (PP)	Non vérifiée
CSY.uevol.721	Prouvée (PP)	Non vérifiée
CSY.uevol.722	Prouvée (Preuve manuelle)	Non vérifiée
CSY.uevol.723	Prouvée (PP)	Non vérifiée
CSY.uevol.724	Prouvée (PP)	Non vérifiée
CSY.uevol.725	Prouvée (PP)	Non vérifiée
CSY.uevol.726	Prouvée (PP)	Non vérifiée
CSY.uevol.727	Prouvée (Preuve manuelle)	Non vérifiée
> induc	1/2	0/2
> induc3	1/1	0/1
> induc4	1/1	0/1
> induc5	1/1	0/1
Fichiers de bases de règles	0/0	0/0

Règle CSY.uevol.2222

Prouvée (Preuve manuelle)

Rule body

```
binhyp(a : seq(t)) &  
binhyp(b : seq(t)) &  
binhyp(first(a) = c) &  
binhyp(first(b) = c) &  
binhyp(!d.(d : 1..size(a)-1 => a(d) : dom(e))) &  
binhyp(!d.(d : 1..size(a)-1 => e(a(d)) = a(d+1))) &  
binhyp(!d.(d : 1..size(b)-1 => b(d) : dom(e))) &  
binhyp(!d.(d : 1..size(b)-1 => e(b(d)) = b(d+1))) &  
binhyp(not( 0  
<=  
p  
- SIGMA(h).(h : 1..size(a) | g(prj1(x,s)(a(h))))  
+ {v|->g(x)-y , u|->y} (z))) &  
binhyp( SIGMA(h).(h : 1..i | g(prj1(x,s)(b(h))))  
- {v|->g(x)-y , u|->y} (z)  
- {v|->m(n) , u|->g(prj1(x,s)(b(i)))-m(n)} (prj2(x,s)(b(i)))  
<=  
p) &  
{v|->m(n) , u|->g(prj1(x,s)(b(i)))-m(n)} (prj2(x,s)(b(i))) : 0..g(prj1(x,s)(b(i))) &  
!f.(f : 1..i => 1 <= g(prj1(x,s)(b(f)))) &  
!f.(f : 1..size(a) => 1 <= g(prj1(x,s)(a(f)))) &  
d\{a,b,e)  
=>  
p/\i<:a
```

Démonstration manuelle

Demonstration

Both sequences are defined by the same induction, and their first element are the same, which means that all their common elements are equal.

```
1) Hypothesis  
binhyp(not( 0  
<=  
p  
- SIGMA(h).(h : 1..size(a) | g(prj1(x,s)(a(h))))  
+ {v|->g(x)-y , u|->y} (z))) &  
binhyp( SIGMA(h).(h : 1..i | g(prj1(x,s)(b(h))))  
- {v|->g(x)-y , u|->y} (z)  
- {v|->m(n) , u|->g(prj1(x,s)(b(i)))-m(n)} (prj2(x,s)(b(i)))  
<=  
p)  
means that  
SIGMA(h).(h : 1..i | g(prj1(x,s)(b(h))))  
- {v|->m(n) , u|->g(prj1(x,s)(b(i)))-m(n)} (prj2(x,s)(b(i)))  
<  
SIGMA(h).(h : 1..size(a) | g(prj1(x,s)(a(h))))  
=>  
2) Hypothesis  
{v|->m(n) , u|->g(prj1(x,s)(b(i)))-m(n)} (prj2(x,s)(b(i))) : 0..g(prj1(x,s)(b(i))) &  
!f.(f : 1..i => 1 <= g(prj1(x,s)(b(f)))) &  
!f.(f : 1..size(a) => 1 <= g(prj1(x,s)(a(f))))  
means  
SIGMA(h).(h : 1..i | g(prj1(x,s)(b(h))))
```

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

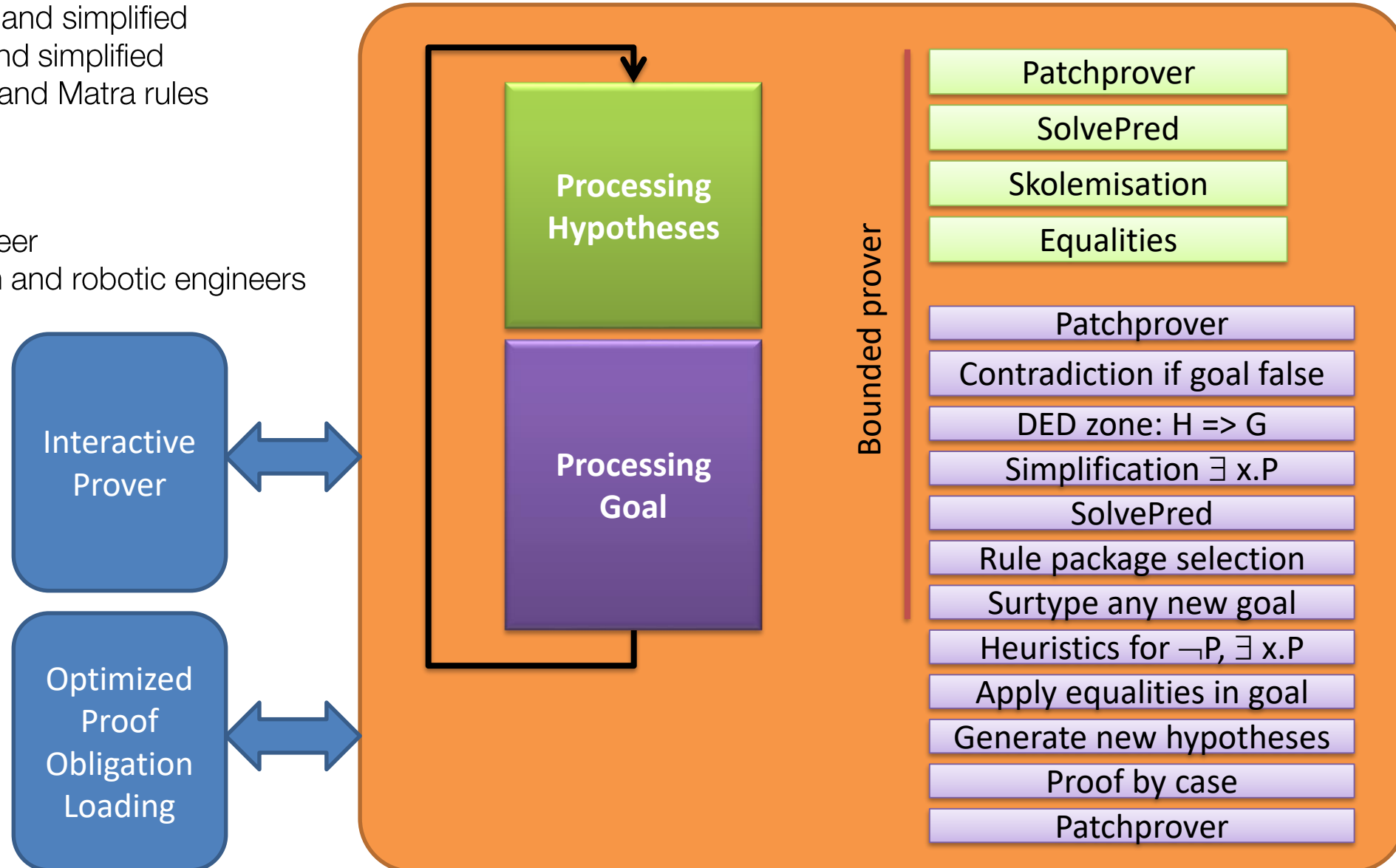
Main Prover

≡ 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

≡ Fixed point

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

```
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
```

Main Prover

≡ 2 700 rules to validate

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

Rule name

Fiche de Validation: Règle InFINXY.90

stration

$C \mapsto B \wedge \text{ran}(n) \in F(B) \Rightarrow (n \in U \in F(S) \Rightarrow n \in U \in F(S))$

$S \subset F(S) \quad \text{d'où} \quad n \in U \in F(S) \Rightarrow n \in U \in F(S)$

$H_1: n' \in C \mapsto B$

$H_2: \text{ran}(n) \in F(B)$

$H_3: n \in U \in F(S)$

$G_1: n \in U \in F(S)$

$H_4: \text{ran}(n) \in F(\text{ran}(n))$ d'après H_2 et [Prop. Annexe 1].

$n. \text{ran}(n) \in C$, d'où H_4 et [Prop. Annexe 1].

$H_5: \text{ran}(n) \in F(\emptyset)$

$H_6: n'[\text{ran}(n)] \in F(B)$ d'après H_4, H_2 et [Prop. Annexe 1].

$n. n'[\text{ran}(n)] = n'[\text{dom}(n)] = \text{ran}(n)$

d'où $H_7: \text{dom}(n) \in F(B)$

$H_8: \text{dom}(n) = \text{ran}(n) \in F(B \times C)$ d'après H_7, H_2 et [Prop. Annexe 1].

$H_9: \forall n. (n \in \mathbb{P} \{ \text{dom}(n) = \text{ran}(n) \} \Rightarrow n \in F(\text{dom}(n) \times \text{ran}(n)))$

d'après H_8 et [Prop. Annexe 1].

$H_{10}: n \in U \in F(\text{dom}(n) \times \text{ran}(n))$ d'après H_9 et H_3

$H_{11}: n \in U \in F(n \times n)$ d'après H_{10} et [Prop. Annexe 1].

$H_{12}: n \in U \subset S$ d'après H_{11}

$H_{13}: n \in U \in F(S)$ d'après H_{12} et [Prop. Annexe 1].

de l'anne 6.1. CQFD.

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~/((size(f)+1)->e))~: 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  discharge par Hyp2
1.2  discharge par Hyp3
1.3  dom((e!-(size(f)+1)))<f~
    = (e!<f~)
    = (f!>e)~
    = (!)~
    = (!)
    = 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 déchargent 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~/((size(f)+1)->e)): s -> NATURAL      Inverse Properties
hyp8  f~: s -> NATURAL      par contradiction avec Hyp7

1.1  discharge par Hyp8
1.2  discharge par Hyp3
1.3  par contradiction (Rule 5), on suppose e: ran(f) et on montre:

```

A detailed demonstration

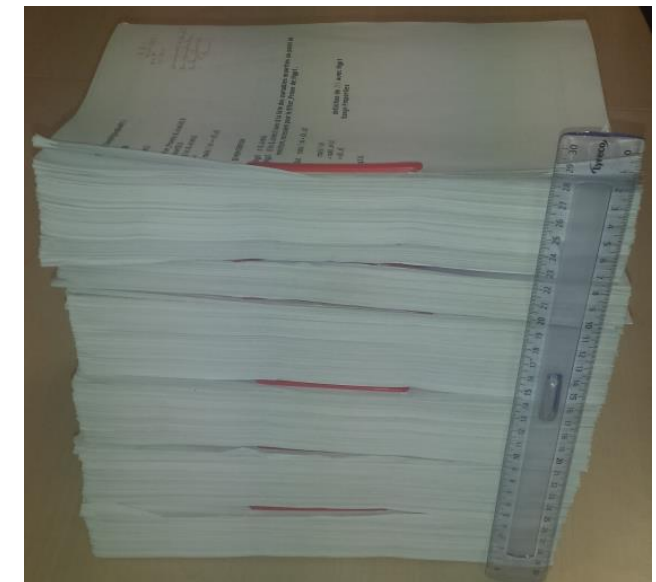
évident (*)

JRA

A concise one !

≡ The resulting validation forms

- 28 cm



(*) « Obvious »

≡ 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);
```

≡ 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

```
/** 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);
```

Interactive Proof

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

Patchprover

User Rule

User Tactics

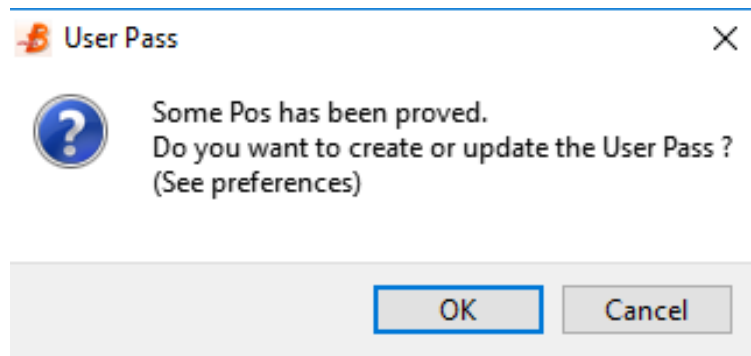
Rule packages

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

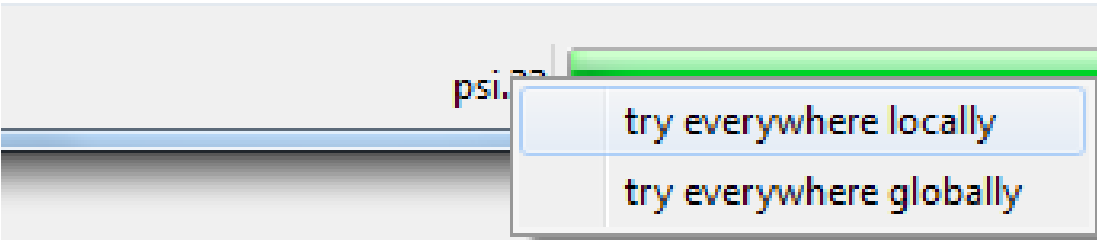
preview

```
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

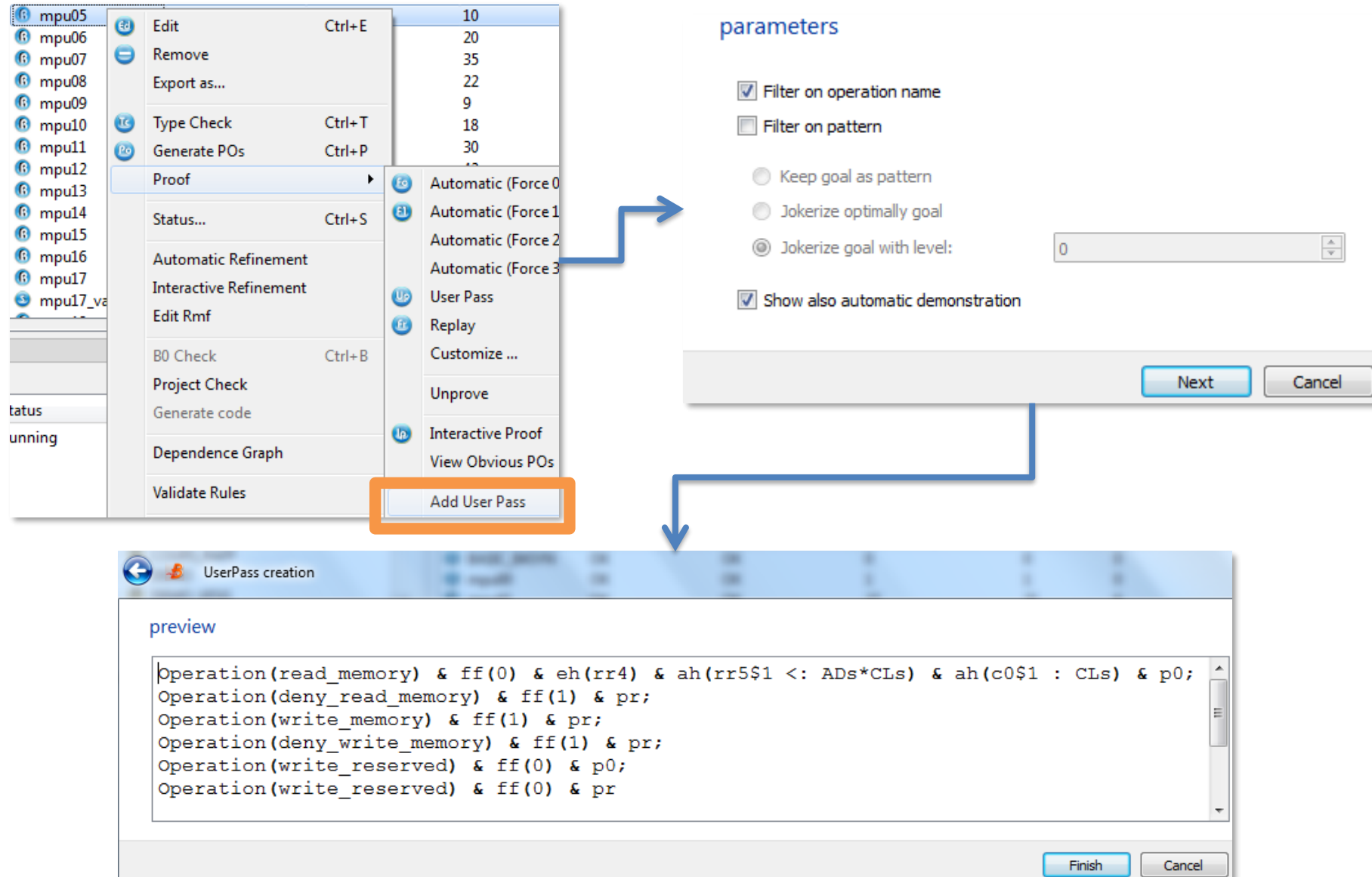
From proof script to tactic

```
t0$1 = Tye
```



- ≡ as is
- ≡ with minor modifications
- ≡ term abstraction
- ≡ bottom-up: avoid early generalization

From proof script to tactic



The screenshot illustrates the workflow from a proof script to a tactic in the CLEARSY software. It consists of three main parts:

- Left Panel (Project Explorer):** A list of components including mpu05 through mpu17, and a 'status' section with 'unning'.
- Top Panel (Menu):** A context menu for the selected component. The 'Proof' option is highlighted, and the 'Add User Pass' option is selected and highlighted with an orange box. A blue arrow points from this option to the 'parameters' dialog.
- Parameters Dialog:** A dialog box titled 'parameters' with the following options:
 - ☒ Filter on operation name
 - ☐ Filter on pattern
 - ☐ Keep goal as pattern
 - ☐ Jokerize optimally goal
 - ☒ Jokerize goal with level: 0
 - ☒ Show also automatic demonstrationThe 'Next' button is highlighted with a blue arrow pointing to the 'UserPass creation' dialog.
- UserPass creation Dialog:** A dialog box titled 'UserPass creation' with a 'preview' section showing the following code:

```
operation(read_memory) & ff(0) & eh(rr4) & ah(rr5$1 <: ADs*CLs) & ah(c0$1 : CLs) & p0;  
Operation(deny_read_memory) & ff(1) & pr;  
Operation(write_memory) & ff(1) & pr;  
Operation(deny_write_memory) & ff(1) & pr;  
Operation(write_reserved) & ff(0) & p0;  
Operation(write_reserved) & ff(0) & pr
```

The 'Finish' button is highlighted with a blue arrow.

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

☐ ☒ 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

≡ 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

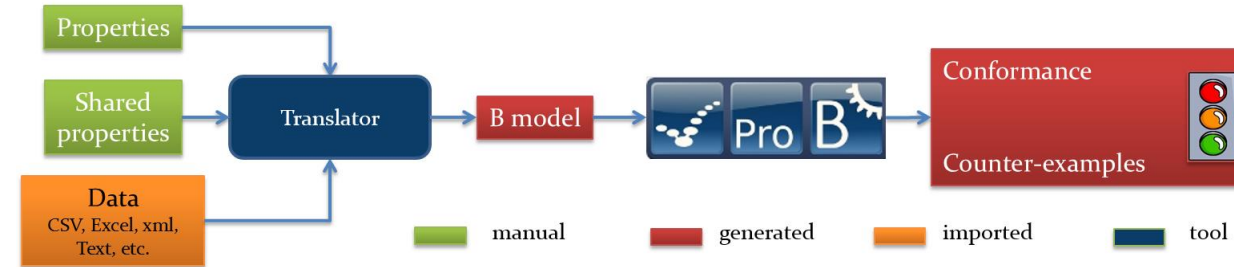
Formal Data Validation in the Railways, T. Lecomte, SSS 2016

Using B and ProB for Data Validation Projects, D. Hansen, D. Scheiner, M. Leuschel, book chapter 2016

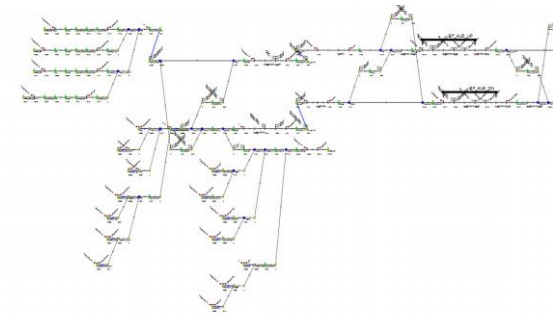
From Animation to Data Validation: The ProB Constraint Solver 10 Years On, M. Leuschel, J. Bendisposto, I. Dobrikov, S. Krings, D. Plagge, book chapter 2014

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)

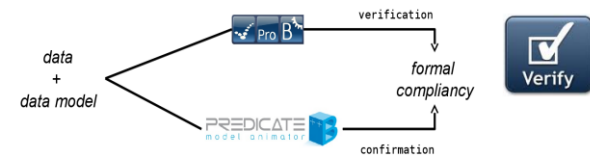


- ≡ Industry-Ready **[PUSH-BUTTON !]**
 - Deal with permanent changes in data and properties
 - Redundant tools to obtain diversity
 - Rules reused from one project to another
 - More than 30 sites verified including:
 - Singapore, Panama, Ryiad, Mecca





data + data model



verification
formal compliance
confirmation



Project

Name: project

Version: 1

Metrics

Files: 22

Declarations: 450

Values: 2527

Properties: 22

Definitions: 0

Status

Project file: OK

Errors: 18

Completed: 100%

Property	Steps	ProB	PredicateB
15 DTVTR_v1.6.1/AL+10/Rule_DB_RIO_0006.xml	Rule_DB_RIO_0006	✓	✓
16 DTVTR_v1.6.1/AL+10/Rule_DB_ROUTE_0002.xml	Rule_DB_Route_0002	✓	✓
17 DTVTR_v1.6.1/AL+10/Rule_DB_ROUTE_0003.xml	Rule_DB_Route_0003	✓	⚠
18 DTVTR_v1.6.1/AL+10/Rule_DB_SIGAREA_0001.xml	Rule_DB_SIGAREA_0001	✓	✓
19 DTVTR_v1.6.1/AL+10/RCD_Track_Container_0001.xml	Rule_RCD_Track_Container_0001	✓	✓
20 DTVTR_v1.6.1/AL+10/Rule_CBTC_SPEED.xml	Rule_CBTC_SPEED	✓	✓
21 DTVTR_v1.6.1/AL+10/Rule_Test_Track_SPEED.xml	Rule_Test_Track_SPEED	✗	✗
22 DTVTR_v1.6.1/AL+10/Rule_BSR_SPEED.xml	Rule_BSR_SPEED	✓	✓

≡ 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

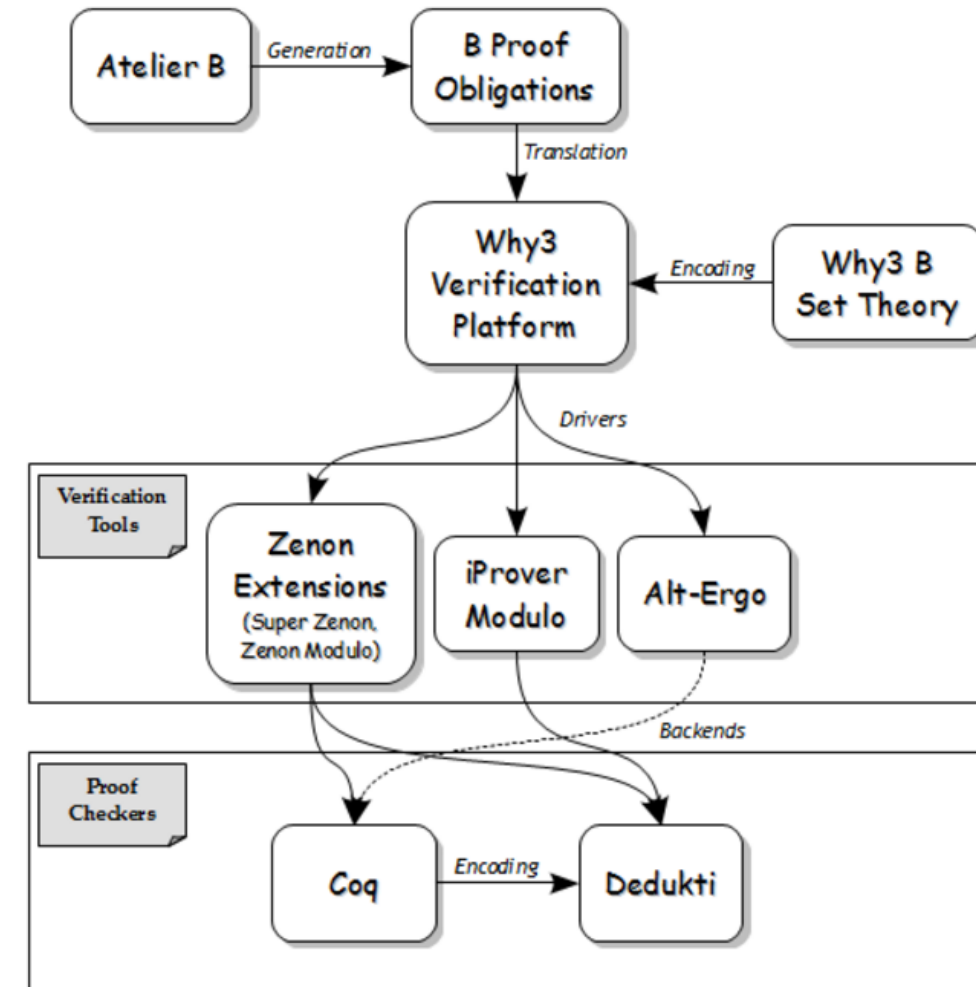
- Added to Atelier B 4.3.1 as an interactive command.
 - Syntax: prob(n) or prob(n | T).
 - T: timer in seconds,
 - n=1 selects all hypotheses that have a symbol in common with the goal
 - Add resource: ATB*PR*ProB_Path:C:\<path>\probcli.exe
- Funded by Alstom

≡ 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

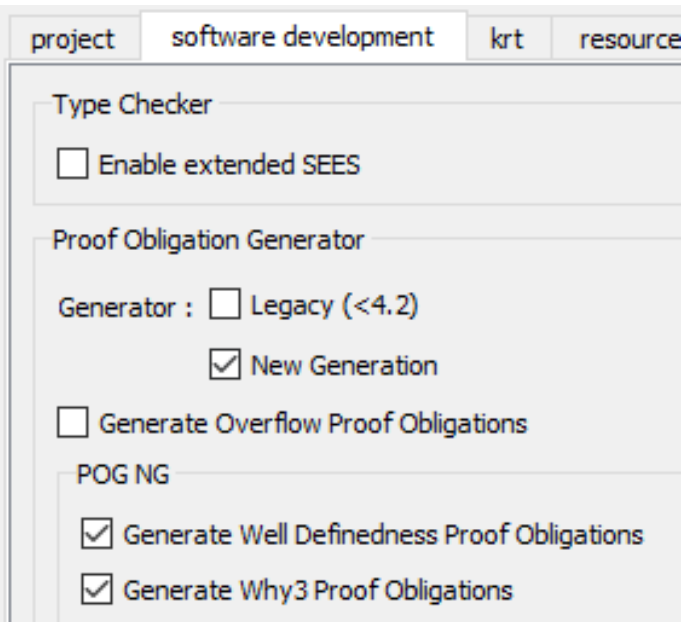
≡ “Teaching an old dog new tricks” - Drudges

- 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))))
```

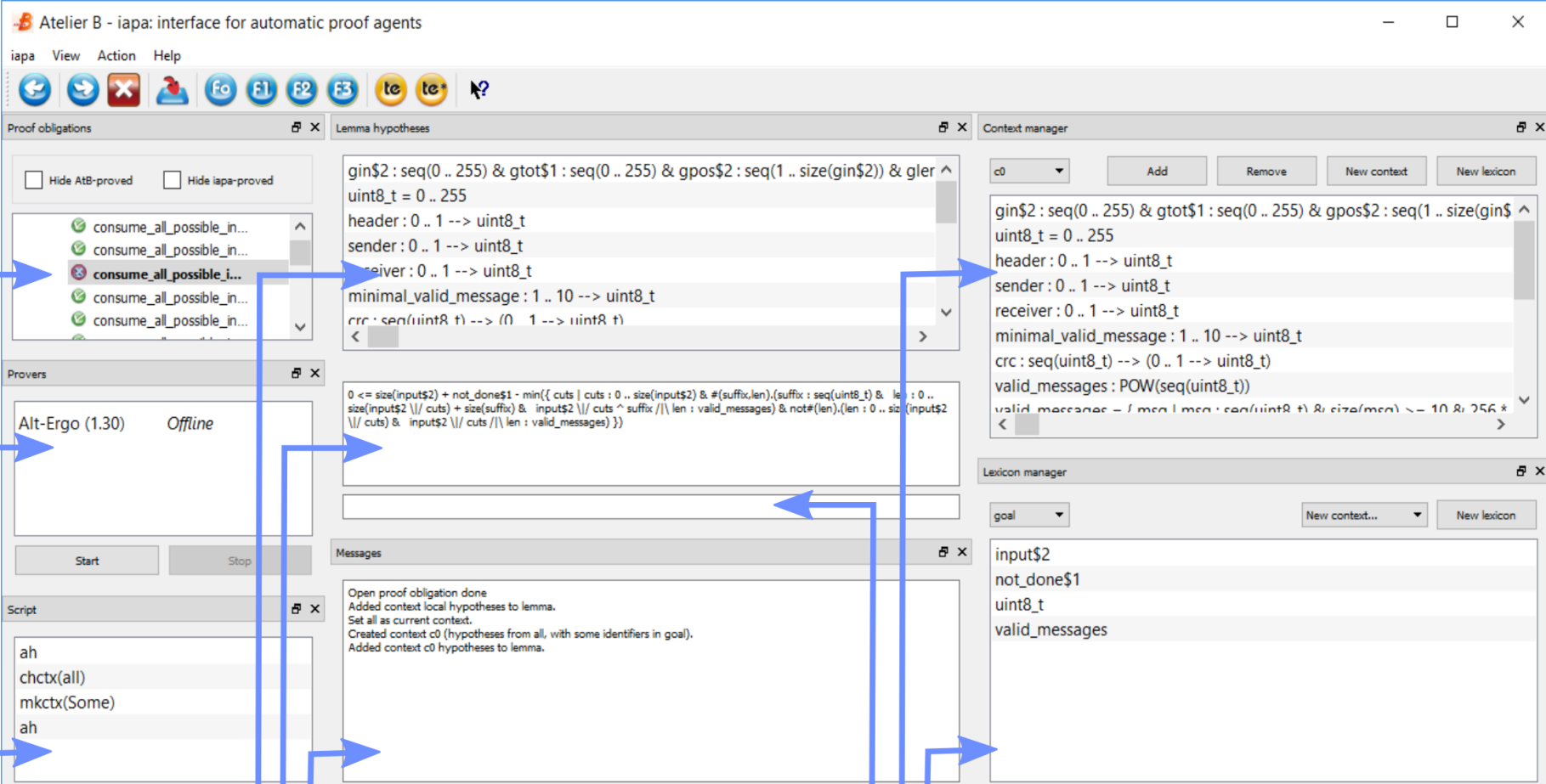
Example of Why3 fragment

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



Atelier B - iapa: interface for automatic proof agents

iapa View Action Help

Proof obligations

Lemma hypotheses

Context manager

Provers

Messages

Script

Lexicon manager

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

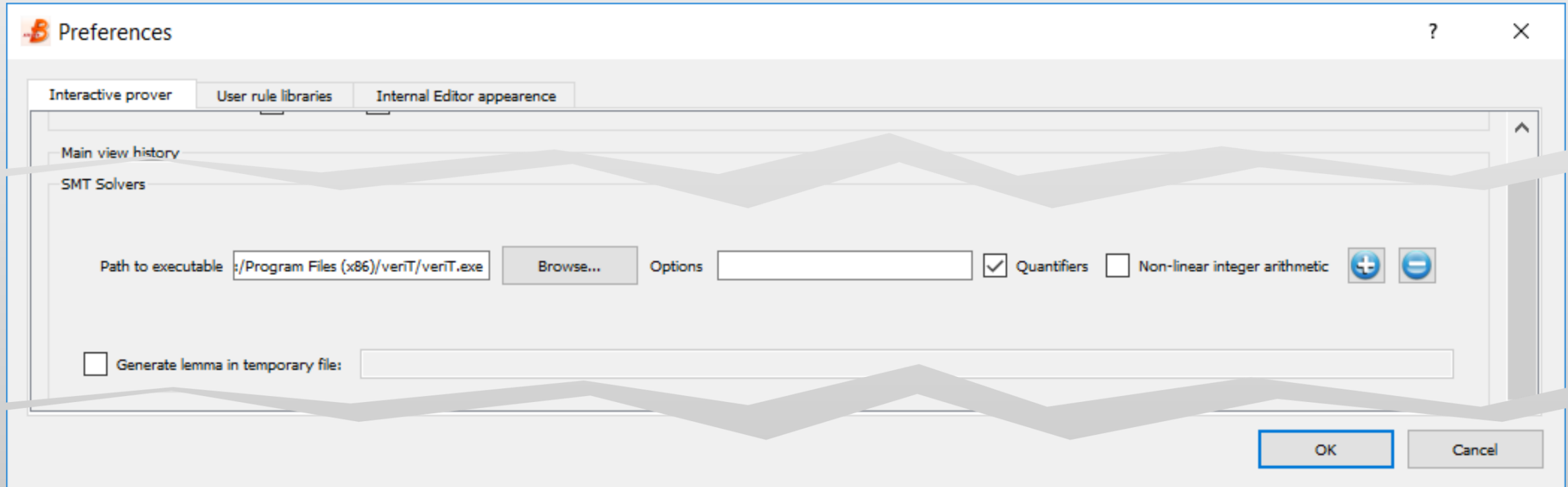
≡ Better solution approach

- 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

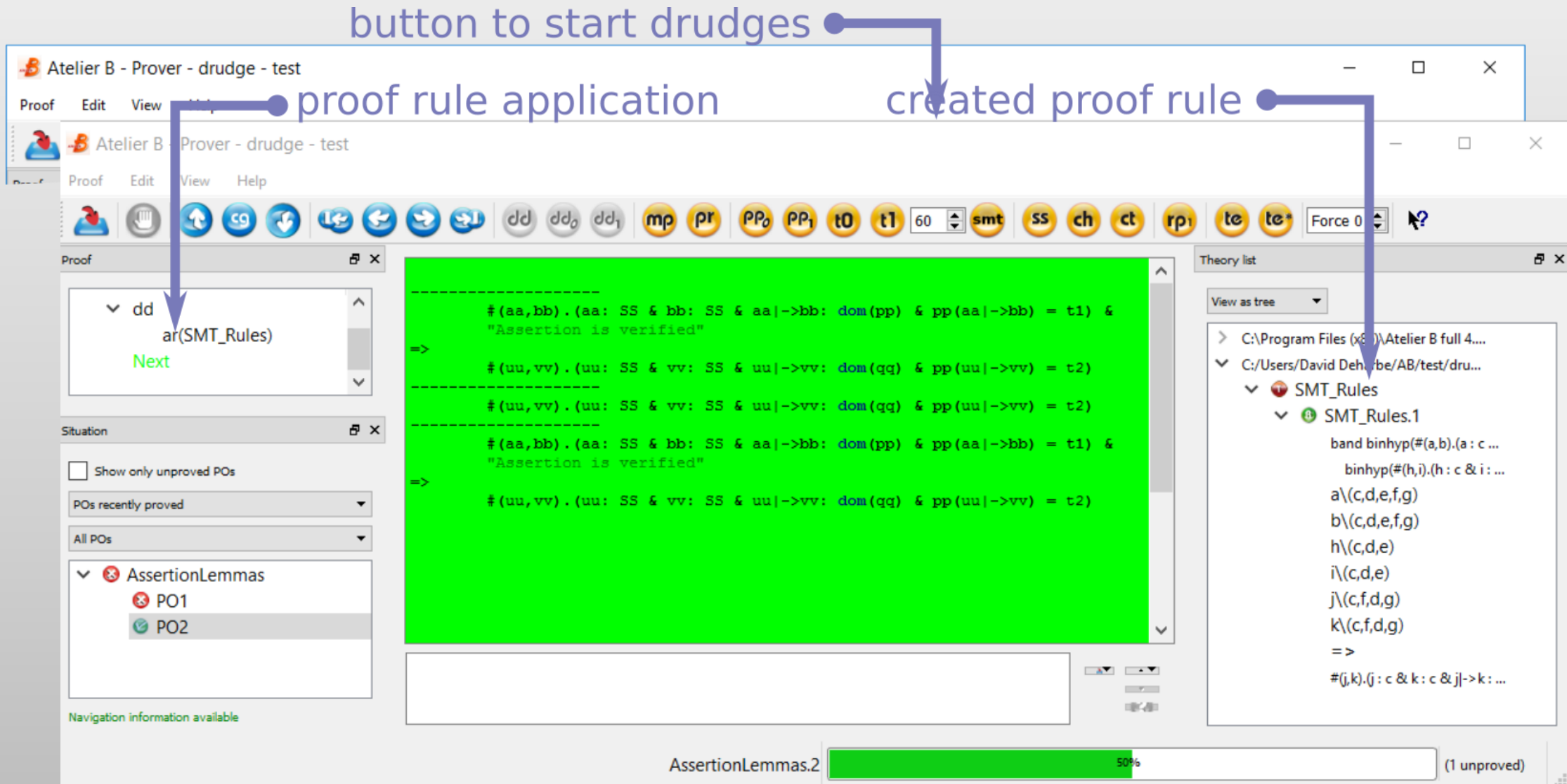


The drudges of the interactive prover

button to start drudges

proof rule application

created proof rule



Proof Edit View Help

Proof Edit View Help

Proof

dd

ar(SMT_Rules)

Next

Situation

Show only unproved POs

POs recently proved

All POs

AssertionLemmas

PO1

PO2

Navigation information available

AssertionLemmas.2

50%

(1 unproved)

```
#(aa,bb).(aa:SS & bb:SS & aa|->bb:dom(pp) & pp(aa|->bb) = t1) &
"Assertion is verified"
=>
#(uu,vv).(uu:SS & vv:SS & uu|->vv:dom(qq) & pp(uu|->vv) = t2)
-----
#(uu,vv).(uu:SS & vv:SS & uu|->vv:dom(qq) & pp(uu|->vv) = t2)
-----
#(aa,bb).(aa:SS & bb:SS & aa|->bb:dom(pp) & pp(aa|->bb) = t1) &
"Assertion is verified"
=>
#(uu,vv).(uu:SS & vv:SS & uu|->vv:dom(qq) & pp(uu|->vv) = t2)
```

Theory list

View as tree

C:\Program Files (x86)\Atelier B full 4....

C:/Users/David Deh.../AB/test/dru...

SMT_Rules

SMT_Rules.1

band binhyp(#(a,b).(a : c ...

binhyp(#(h,i).(h : c & i : ...

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 : ...


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

Searching the most generic rule

$$\begin{aligned} & \exists(s1,s2).(s1 \in SS \wedge s2 \in SS \wedge s1 \mapsto s2 \in \text{dom}(pp) \wedge pp(s1 \mapsto s2) = t1) \Rightarrow \\ & \quad \exists(s1,s2).(s1 \in SS \wedge s2 \in SS \wedge s1 \mapsto s2 \in \text{dom}(qq) \wedge pp(s1 \mapsto s2) = t2) \wedge \\ & \quad \exists(aa,bb).(aa \in SS \wedge bb \in SS \wedge aa \mapsto bb \in \text{dom}(pp) \wedge pp(aa \mapsto bb) = t1) \\ & \Rightarrow \\ & \quad \exists(aa,bb).(aa \in SS \wedge bb \in SS \wedge aa \mapsto bb \in \text{dom}(pp) \wedge pp(aa \mapsto bb) = t1) \Rightarrow \\ & \quad \quad \exists(uu,vv).(uu \in SS \wedge vv \in SS \wedge uu \mapsto vv \in \text{dom}(qq) \wedge pp(uu \mapsto vv) = t2) \end{aligned}$$

$\equiv$ 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))
```

≡ 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