

TRAINING B – Level 3

Prove B

CLEARSY



Proving with Atelier B

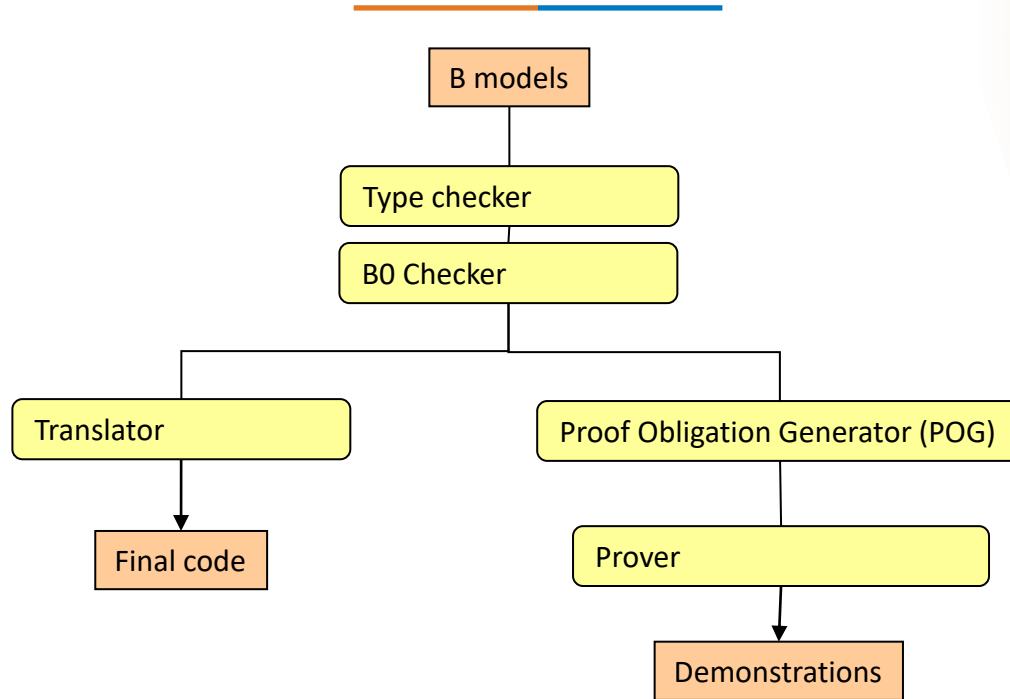
- Proof Tools - Introduction
- Proof - General Method
- Automatic Proof - Use
- Interactive Prover - Quick Tour
- Tuning Up - General Method
- Formal Proof - Project BASES
- Proof - Management and Administration



PROOF TOOLS

Introduction

Atelier B Main Tools



Proof Files

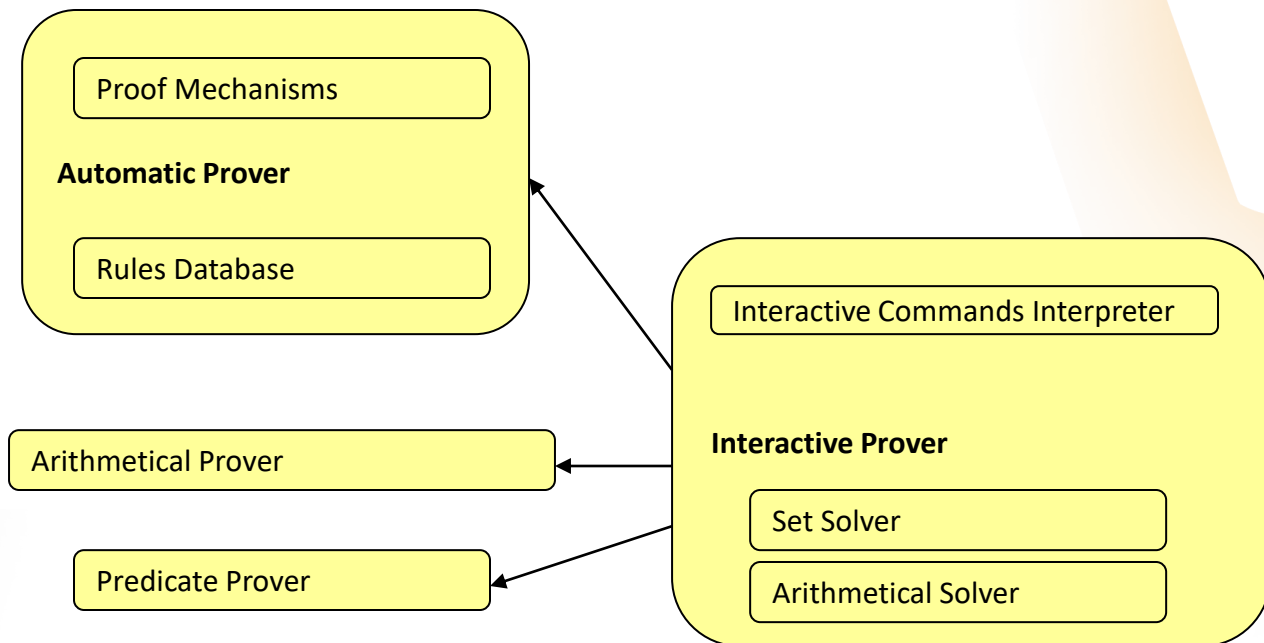
Proof obligations (<component>.po)

Demonstrations + proof status (<component>.pmi)

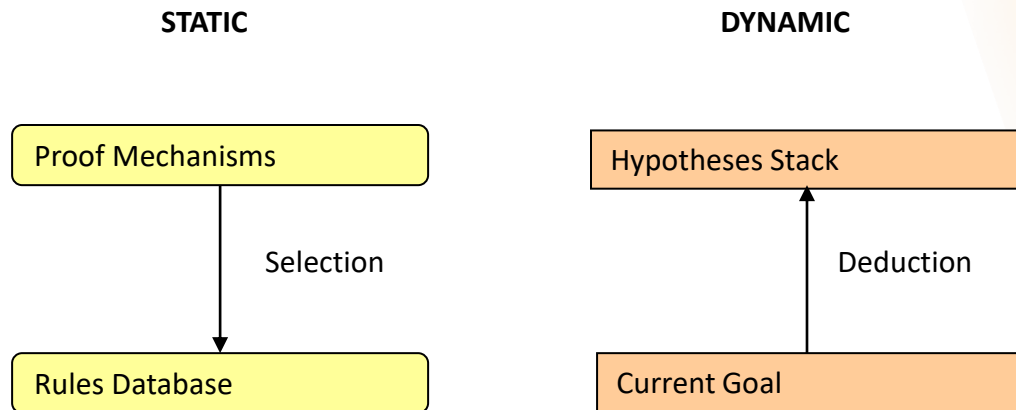
User Rules at component level (<component>.pmm)

User Rules at project level (PatchProver)

Proof Tools



Automatic Prover Internals



Once a hypothesis has been loaded into the hypotheses stack, it can not be modified



PROOF

General Method

General Method

- Do not prove to early
- Example
 - component with 300 unproved PO
 - PO n° 298 is false, because of an incorrect invariant
 - time to demonstrate first 297 PO: 5 days
- Impact of modifying invariant
 - all PO are modified
 - 5-day work lost
- Tune up before proving

Tuning Up

- checking POs to find FALSE POs
- if all PO were always true, proof would be useless
- 10% of PO are usually false in a new project

Tuning Up

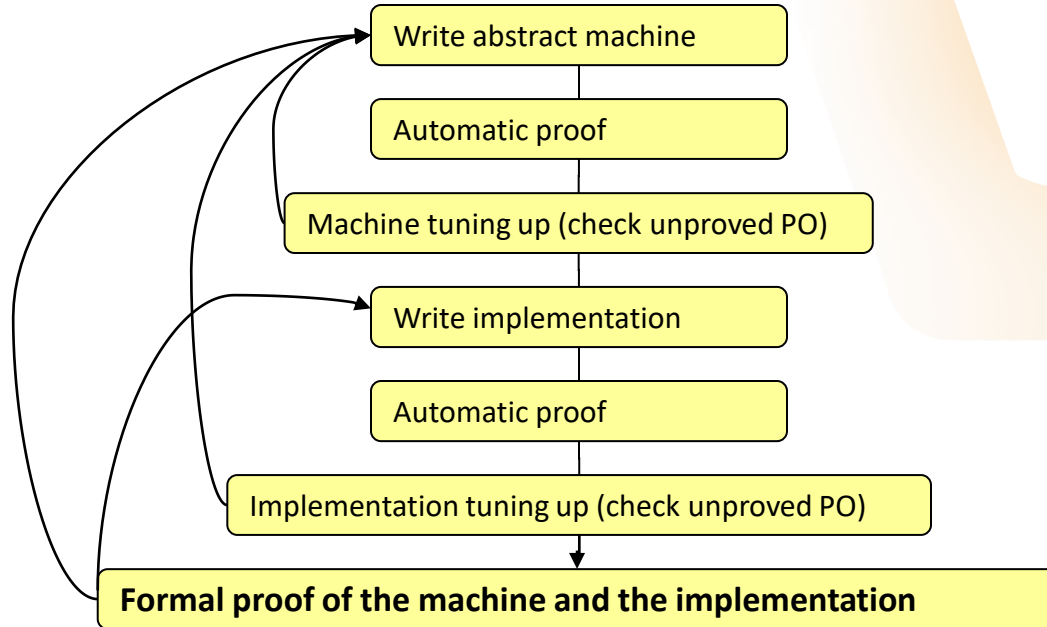
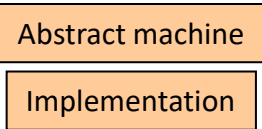
- risks while tuning up
 - spend too much time
 - undertake complex demonstrations
- precautions
 - choose a maximum time effort for each PO (10 minutes)
 - choose a maximum number of proof commands (5 commands)

Formal Proof Phase

- demonstrate remaining PO
- if a false PO is discovered
 - switch back to tuning up
- reason: in case of component modifications
 - other PO may become false
 - their demonstration may become invalid

Proof Activities

A simple project:



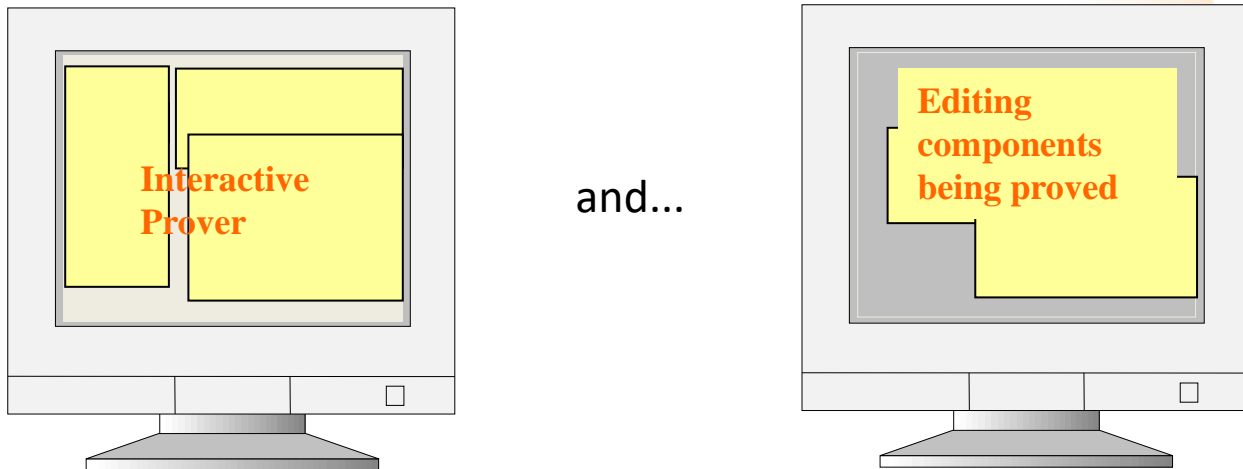
Automatic Forces

- **Fast Force:** fastest but least efficient
 - 30%, average time = 3 seconds only for large components
- **Force 0:** main force
 - 70%, average time = 10 seconds
 - to use first
 - to use for interactive proof
- **Force 1:** each hypothesis is simplified according to previously loaded hypotheses (+1%)
- **Force 2:** + derived hypotheses generation (+2%)
- **Force 3:** + attempts proof (+1%)

Documentation

- B Language Reference Manual
 - understanding symbols, properties
- B-Book
 - list of mathematical properties
- Prover Reference Manual
 - reference for proof commands
 - online help: [help](#), [help\(command\)](#)
- Prover User Manual
 - principles and user guide

Proof Tool Overview



- **Machine proof:** display
 - the machine
 - possibly its included/seen machines

- **Implementation proof:** display
 - ➔ the implementation
 - ➔ its specification
 - ➔ its imported/seen machines

Automatic Proof

Use

Forces Chaining

- with the project **BASES**, add component **Resources.mch**
- initiate proof with force 2
 - type check, PO generation
 - proof with force 0: everything is proved except 3 PO
 - proof with force 1: no PO proved
 - proof with force 2: long, 1 PO proved
- Forces 0, 1, 2 et 3 are chained
- high forces (≥ 1) may be long, but may succeed

Storing Tried Forces

- restart proof with force 0, then with force 1, then with force 2:
 - nothing happens
- the prover stores already tried forces for each PO. Here, it stores that the 2 remaining PO are not proved with forces 0 to 2
- the prover stores PO status: only unproved PO are concerned

NOTA: this status is reset when the component is (really) modified

Tried Forces Are Preserved

- Select **Resources** and use the option **Unprove** of the menu **Prove**
 - all PO are reset to unproved
- Start proof with force 0
 - only PO previously proved with force 0 are replayed, the prover displays only +
- Start proof with force 1
 - nothing happens, no PO was discharged with force 1
- Start proof with force 2
 - the only PO which is replayed is the one that was discharged with force 2
- Forces history is preserved

Merge Operation

- Modify component `Resources.mch`
 - in operation `RestoreResource`, use a `SELECT` instead of an intersection
- Start proof with force 0
 - only the PO of the modified operation lost their `Proved` status
 - the 2 unproved PO in non-modified operations are tried again
- Comparison new PO / old PO
 - POs with same goal and with stronger hypotheses keep their status

$$\begin{array}{lcl} H_1 \wedge H_2 \Rightarrow G & \xrightarrow{H_1 \wedge H_2 \wedge H_3 \Rightarrow G} & G \quad (\text{status preserved}) \\ & \xrightarrow{H_1 \wedge H_3 \Rightarrow G} & G \quad (\text{status reset}) \end{array}$$
- Forces history is initialised

Interrupting Automatic Proof

- With component **Resources**, start proof with force 3
 - the proof is long
- Go to the next PO: button **Next PO**
 - move mouse pointer into the interrupt zone: buttons turn sensitive
 - select **Next PO**: buttons turn insensitive
 - interrupt is taken into account: a - is displayed
- high forces may lead to an infinite loop
 - interrupting is sometimes necessary
- **WARNING:** a delay can be experienced in case of a huge proof/component

Interactive Demonstration

- Open with interactive prover component [Resources](#)
- Go to the next unproved PO
- Type in `pp(10)` to invoke the predicate prover with a maximum computation time of 10 seconds
- Interactive demonstration is saved when PO is quitted

Replay

- With the component **Resources**, select **Unprove**
- Start Prove Replay
 - In a single pass: all PO are replayed
 - The one who were proved are replayed with their stored demonstration (either automatic or interactive) then result is displayed (+ or -)
 - Previously unproved PO are skipped (displaying -)
- Only previously proved PO are replayed
 - for retrying a demonstration of POs which lost their “proved” status (because of hypotheses modifications - following a merge)
 - for ensuring demonstration validity after having modified user rules or after adding new hypotheses

Replay

- Modify component `Resource.mch`
 - for example, change the name of the variable `available` which becomes `free`
- Generate proof obligations
 - All PO turned to be unproved, because variable name modification prevents from finding a correspondence between old POs and new
- Start Prove Replay
 - Previous proof status is restored. Indeed, demonstrations lead to successful proofs whatever the new name of the variable
 - In particular, demonstrations using `pp` are preserved
- Prove Replay permits to retry demonstrations after an important modification of the model



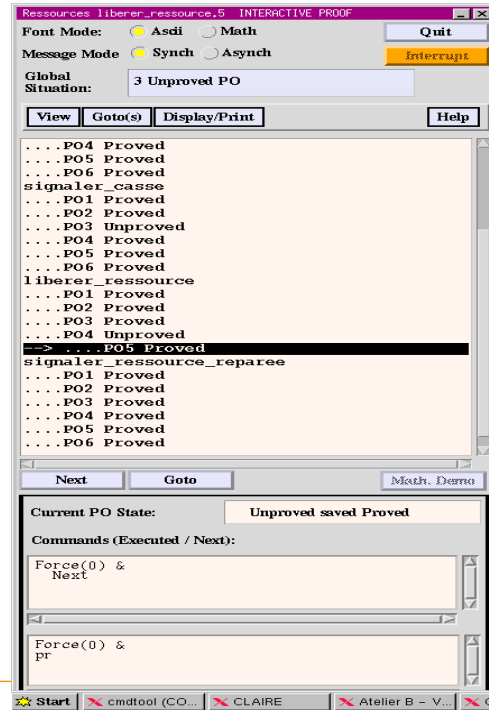
Interactive Prover

Quick Tour

Motif Interface Overview

- In the **BASE** project, add the component **Resources** and prove it with force 0
- Start the interactive prover
- Go to the first unproved PO

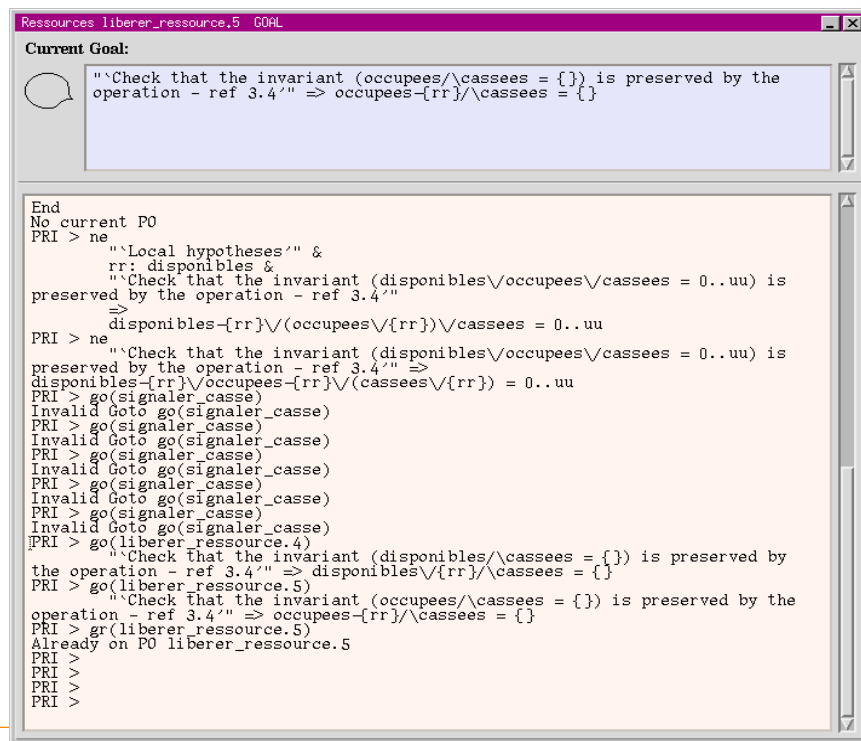
Situation Window



Situation Window

- Number of unproved PO area (becomes green when proved)
- PO selection area
 - PO display / selection menu
 - external copy-paste / rules database access
 - Help = reference manual
- Status area of the current PO
 - current / saved status
 - applied commands
 - saved commands
- quit and interrupt buttons
- special modes: math symbols / asynchronous display

Command Window

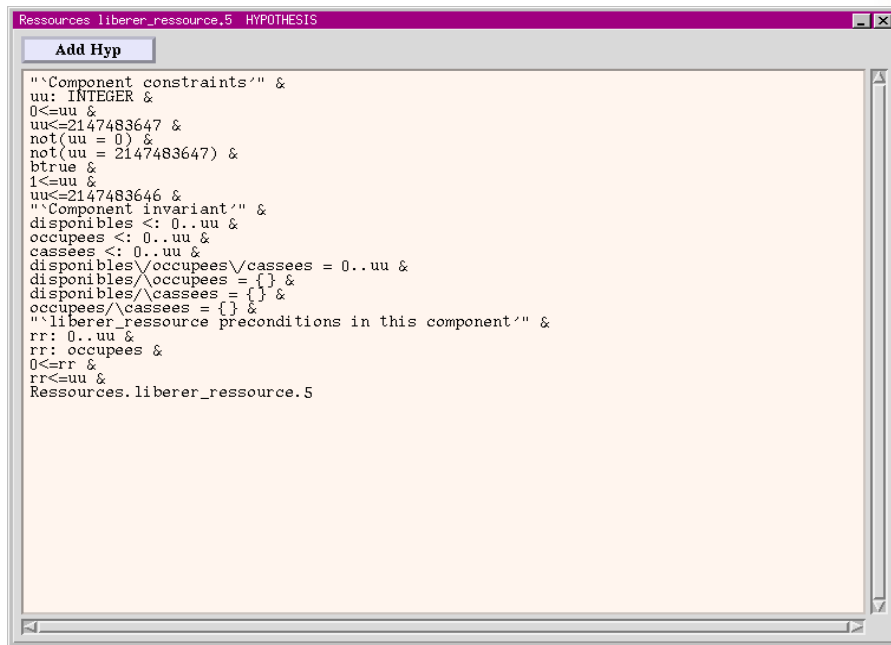


```
Resources liberer_ressource.5 GOAL

Current Goal:
"Check that the invariant (occupees/\casseees = {}) is preserved by the
operation - ref 3.4" => occupees-{rr}/\casseees = {}

End
No current P0
PRI > ne
  "Local hypotheses" &
  rr: disponibles &
  "Check that the invariant (disponibles/\occupees/\casseees = 0..uu) is
preserved by the operation - ref 3.4" =>
  disponibles-{rr}/(\occupees/{rr})/\casseees = 0..uu
PRI > ne
  "Check that the invariant (disponibles/\occupees/\casseees = 0..uu) is
preserved by the operation - ref 3.4" =>
  disponibles-{rr}/occupees-{rr}/(\casseees/{rr}) = 0..uu
PRI > go(signaler_casse)
Invalid Goto go(signaler_casse)
PRI > go(signaler_casse)
Invalid Goto go(signaler_casse)
PRI > go(signaler_casse)
Invalid Goto go(signaler_casse)
PRI > go(signaler_casse)
Invalid Goto go(signaler_casse)
PRI > go(signaler_casse)
Invalid Goto go(signaler_casse)
PRI > go(liberer_ressource.4)
  "Check that the invariant (disponibles/\casseees = {}) is preserved by
the operation - ref 3.4" => disponibles/{rr}/\casseees = {}
PRI > go(liberer_ressource.5)
  "Check that the invariant (occupees/\casseees = {}) is preserved by the
operation - ref 3.4" => occupees-{rr}/\casseees = {}
PRI > gr(liberer_ressource.5)
Already on P0 liberer_ressource.5
PRI >
PRI >
PRI >
PRI >
```

Hypotheses Window



Commands/Hypotheses Windows

- these windows appear only if we have moved to a PO
- commands window
 - goal area, turns green when the demonstration is complete
 - commands area
 - with shift + right click: search
- hypotheses window
 - one hypothesis per line (horizontal scrollbar)
 - hypotheses filter
 - with shift + right click: search
 - triple left click + right click: addition of a hypothesis (no more available)

Classical Positioning

hypotheses are visible
and can be copied

foreground easy
to commute

goal and commands

```
Ressources liberer_resource.5 HYPOTHESIS
Add Hyp
"Component constraints" &
uu: INTEGER &
0<=uu &
uu<=2147483647 &
not(uu = 0) &
7483647) &
46 &
invariant" &
<: 0..uu &
0..uu &
cassees <: 0..uu &
disponibles\occupees\casseees = 0..uu &
disponibles\occupees = {} &
disponibles\casseees = {} &
occupees\casseees = {} &
"liberer_resource preconditions in this component" &
rr: 0..uu &
rr: occupees &
0<=rr &
rr<=uu &
Ressources.liberer_resource.5

PRI > go(liberer_resource.4)
"Check that the invariant (disponibles\casseees = {}) is preserved by
the operation - ref 3.4" => disponibles\{rr}\casseees = {}
PRI > go(liberer_resource.5)
"Check that the invariant (occupees\casseees = {}) is preserved by the
operation - ref 3.4" => occupees-{rr}\casseees = {}
PRI > gr(liberer_resource.5)
Already on FO liberer_resource.5
PRI >
PRI >
PRI >
```



Tuning Up

General Method

Method

- search false PO
- do not undertake long demonstration
- display components being proved
- use Interactive Prover functions to browse PO

Searching Hypotheses

- with the **BASES** project, add **Resources0.mch**, prove with force 0
- start interactive prover, go to the first unproved PO
- type in **mp**
 - start prover, producing *a priori* a simpler goal
 - searching examples using **SearchHyp**
 - **sh(available)** search hypotheses containing this variable
 - **sh(available _and rr)** hypotheses containing both **available** and **rr**
 - **sh(a=b)** hypotheses **containing** an equality. One letter variables are “jokers”, they can match any formula
 - **sh(p, (a=b))** hypotheses which **are** an equality

Examining PO

- with component Resources0, go to the last unproved PO ([ReleaseResource.4](#))
- [dd\(0\)](#) simplify and load local hypotheses, to examine only the goal
- [rp](#) (Reduce Po) search hypotheses having a common symbol with the goal
- find a justification for the goal: here
 - `available /\ faulty = {}`
 - `rr` is not a member of `faulty`
 - so `available \/ {rr}` has no common element with `faulty` the goal is true.
- during examining PO, we can start [pp](#) (here: successful)
 - if few hypotheses

False PO with Complex Goal

- how to examine a PO containing complex expressions?
 - Use the prover to simplify expressions
- in case of bad simplification
 - use *dc* (Do Cases) to split a demonstration into several cases
- in case of presumed false PO
 - use *dc* to try values of the counter-example
- *eh(var, val, AllHyp)* (equality in hypothesis)
 - to replace *var* by *val* in all the hypotheses, to generate new hypotheses

False PO with complex goal (2)

- Example
- display components `switch.mch` and `switch_2.imp`
- add these components, prove `switch_2` with force 0
- interactive proof
 - go to next unproved PO: complex goal
 - `mp` fails to prove: goal still complex
 - `dc(m1, POSITION)` proof by cases for all the values of `m1`, `pr` first case is proved, `pr` second case is not proved
 - PO seems to be false. We start proof by contradiction: `cts`
 - From hypotheses: `m1,m2,m3 = reverse,normal,reverse`
 - `eh(m1,gauche,AllHyp)` & `eh(m2,droite,AllHyp)` & `eh(m3,gauche,AllHyp)` & `pr` still no contradiction, we probably have our counter-example.

Simplifying Expressions

Every correct project is not always reasonably provable

- reshape a project for which proof is complex
 - add refinement levels
 - decompose operations
 - use ASSERT and ASSERTIONS clauses
- use expressions as they are simplified by the prover
- search equalities containing literal terms

Using Assertions

- Assertions are predicates which are part of B models their only role is to ease proof
- 2 kinds of assertions can be used:
 - assertions from the clause **ASSERTIONS**, which are global to a component
it should be deduced from the invariant and the previous assertions (order is important)
assertions become hypothesis of other PO
 - assertions from substitutions **ASSERT**, which are local to an operation
each assertion should be proved with the properties of variables at the location of the assertion
assertions become hypothesis in the PO concerning the substitutions located after the assertion

Using Assertions (2)

- assertions of the clause ASSERTIONS
- example: machines `Assertions` et `Assertions0` (with and without assertions)
 - examine differences between these two machines,
 - the assertion is true because: `a function strictly increasing is injective`
 - add these machines and start proof with force 3,
 - statistics: `Assertions` `Assertions0`
1 unproved PO 4 unproved PO
 - all PO have the same complexity
- advantage: assertions allows to factorise proof of operations of a component.

Using Assertions (3)

- assertions of the substitution ASSERT
- example: machine `Assert` and refinements `Assert_1` et `Assert_0` (with and without assertion)
 - examine differences between these 2 refinements,
 - assertion precise how the `IF` is refined, the case `yy >= 1` corresponds with the case `xx >= 0` of the specification,
 - add these components and start proof with force 0,
 - statistics: `Assert_1` `Assert_0`
1 unproved PO 2 unproved PO
 - PO have the same complexity
- advantage: assertions allow to ease proof of an operation.

Formal Proof

Bases

Demonstrations

in Atelier B, a demonstration is a sequence of commands

- go to the PO `AcquireResource.3` of the component `Resource0`
- open in an editor the file `DemoResL`
- open the window called "`Extern Selection`" of the prover
- copy the demonstration contained in the file `DemoResL` in this window and select "`Paste`"
- in the log window, type in `Return`
 - imported demonstration can be replayed and PO is proved

Demonstrations (2)

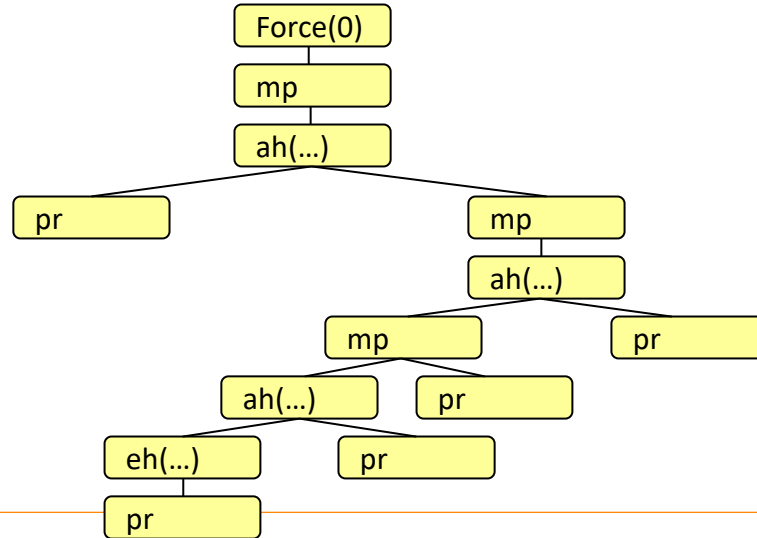
- demonstration can be seen in the window containing applied commands, with indentation as follows:

```
Force(0) &      used forced
mp &           we start with mp
  ah(available-{rr}\\(in_use\\{rr})\\faulty <: (0..nn) &
    pr &       first goal of ah: demonstrate hypothesis
    mp &       we continue with this hypothesis
  ah(0..nn <: (available-{rr}\\(in_use\\{rr})\\faulty) &
    mp &
      ah(xx: available\\in_use\\faulty) &
        eh(available\\in_use\\faulty) &
          pr &
            pr &
              pr &
                pr &
Next
```

- the content of this window can be copied in a editor

Demonstrations (3)

- such a demonstration can also be seen as a tree:
 - column number becomes line number for the tree
 - each command is linked with commands located below



Using mp

- **mp** with force 0 and 1 starts prover without proof by cases tactics
 - to use simplifying capabilities of the prover with no risk to start multiplied demonstrations.
- **example**
 - add machine **MiniPr**, start prover with force 3,
 - examine unproved PO, the goal can be simplified,
 - **mp**, in the goal the expression $0..10 \wedge \{3\}$ is simplified in $\{3\}$
 - **pp(rp.0)** the PO is proved
- **difference between mp and pr**
 - restart demonstration by replacing **mp** by **pr**: **re & pr & pp(rp.0)**
 - the remaining goal is $\text{not}(vv = 3) \Rightarrow Q$ in fact, the prover did 2 cases because of the hypothesis $vv = 3 \Rightarrow \text{not}(a = \{nn\})$, it's useless

Adding Hypothesis

- adding a useful lemma (a subgoal)
- example
 - add component `AddHyp` and start proof with force 0,
 - one PO is not proved, type in `dd(0)`
 - the PO is true because `ff` is overloaded with elements of `ff`, so the result of this overloading remains equal to `ff`,
 - the prover did not have the idea to demonstrate `0..5 <+ ff <: ff`
 - type in `ah(0..5 <+ ff <: ff) & pr & pr`
 - the PO is proved

Using pp (1)

- use `ah(existing hypotheses) & pp(rp.0)`
 - adding existing hypotheses to put then in the goal
 - `pp(rp.0)` to try to demonstrate the current goal without any other hypothesis
- add an intermediate lemma to be demonstrated by `pp`
 - from the main goal
 - to demonstrate an intermediate goal

Using pp (2)

- avoid to use **pp** without **rp** with a component that is likely to evolve
 - if the number of hypotheses increases: direct impact on the time for the predicate prover to prove
 - if maximum time allowed for proof is exceeded, pp fails
- message « **pp failed to prove** »: type checker error
 - in the console: message « can not resolve (**formula**) »
 - remove/transform (**formula**) in the lemma to prove:
often an expression like $A-B$ should be replaced by $-B+A$ (in case of arithmetical expression)

Using pp (3)

- when **pp** fails
 - lemma contains arithmetical computations, inequalities, card: abort!
 - lemma contains expressions like $f(x)$ add $x : \text{dom}(f)$
 - very often, a hypothesis $f : A \rightarrow B$ is missing.
- use **vr** (Verify Rule)
 - using **pp** for validating a user rule
 - To verify a rule before adding it to a **pmm** file
 - To quickly verify a rule used in the intuitive reasoning

ex: `vr (Back, (f: A → B => f[E-F] = f[E]-f[F]))`

Adding Hypotheses to Use a Rule

- add a hypothesis to be able to apply a rule
- restart previous example
 - type in `re & mp`,
 - search rules to transform the goal: `sr(Rewr, (f <+ g == f))`
 - the rule found is:

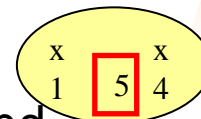
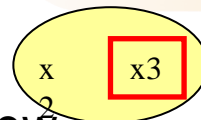
```
SimplifyRelOverXY.6
  binhyp(g <: f) ^
  binhyp(f : s +-> t) ^
  blvar(Q) &
  Q\g <: f
=>
(f <+ g == f)
```
 - the missing hypothesis is `0..5 <| ff <: ff`

Differences Between `dd` and `dd(0)`

- example
 - add machine `DDZero` and start proof with force 0, a false PO is generated (only used for demonstration)
 - type in `mp`
then `dc(bool(not(nn = 0)) = FALSE)`
 - examine differences between `dd` and `dd(0)`, type in `dd`
then `ba & dd(0)`
 - in the case of `dd(0)` new deduced hypotheses appear:
`nn = 0 & vv = 0 & ww = 0`
- `dd(0)` deduction with force 0 mechanisms
 - try also `dc(nn : 10..20)`

Simplifying Equalities

- example: machine `Eql` with a false assertion
 - examine the machine `Eql` start proof and type in `mp` for the PO related to the assertion
- variables are simplified group by group, according to equalities already loaded
- for each group, only one variable is stored, others are replaced by this variables (expressions are not processed)
 - Ex: with $a=b$ $b=c$ $c=d$ $d=e$, only one variable should be used
- loaded hypotheses can not be affected by such new simplifications
 - try to do `dc(x1 = x2)`
- this mechanism can not be used with the command `dd`



Equalities: Manual Replacements

- to replace an expression $e1$ with $e2$, under the hypothesis $e1=e2$ (or $e2=e1$)
- the replacement takes place:
 - in the Goal: $eh(e1, e2)$ shortcut $eh(e1)$ ($e2$ is the first possible value)
 - in all the hypotheses: $eh(e1, e2, AllHyp)$ to create new hypotheses
 - in hypothesis H : $eh(e1, e2, Hyp(H))$

Existential Goals

- Example: machine `Suggest` and its refinement `Suggest_1`
 - add these components in the project and start proof with force 0,
 - you should demonstrate that `ss` contains a value such as `ss` is not empty and `3 : ss`
 - the prover is not able to generate such attempts (except force 3) , it only knows how to demonstrate: $\#x . (P(x) \ \& \ x = a) \iff P(a)$
- Interactive demonstration
 - `mp & se({3}) & pr`
- Existential goals are found
 - in an ANY xx WHERE ... non directly refined
 - add a variable `lastxx := xx` in the component, keep this variable in the refinement
 - when using non refined abstract constants

Manual Creation of Derived Hypotheses

- instantiation of a « for all » predicate
 - $\text{ph}(x_0, !x.(P_x \Rightarrow Q_x))$ to particularize $!x.(P_x \Rightarrow Q_x)$ for the value x_0
 - first P_{x_0} has to be proved, then the new hypothesis Q_{x_0} is generated
- use of an « imply » hypothesis (Modus Ponens rule)
 - $\text{mh}(P \Rightarrow Q)$
 - under the hypotheses $P \Rightarrow Q$ and P , the new hypothesis Q is generated

Automatic Creation of Derived Hypotheses

- derived hypothesis: hypothesis automatically generated by mechanisms such as `dd(0)`
 - for these hypotheses, some mechanisms like equality simplification are not used (too long to be used each time)
 - to apply these mechanisms add again the hypothesis with `ah(hyp)`
- remark
 - often `mp` & `tp(Hyp)` lead to proved when derived hypotheses miss
 - `tp(Hyp)` try to prove the goal by adding hypotheses which enable to fire rules.

Manual/Mixed Demonstration

- mixed demonstration: interactive demonstration using both the mechanisms of the prover and pure interactive commands
- manual demonstration: no mechanism is applied
- example: machine **Manual** which is not proved with force 3
 - when typing **mp** hypotheses are normalized and difficult to read, in particular, the following hypotheses are no more present:
$$\begin{aligned} \text{not}(P) &\Rightarrow k = 1 \quad \& \\ P &\Rightarrow k = 0 \end{aligned}$$

Manual Demonstration

dd

dc ($aa * 10 \leq bb - 3$)

pr

dd

mh($\text{not}(aa * 10 \leq bb - 3) \Rightarrow kk = 1$)

pr

- the mechanisms of the prover are only used when they succeed at each step

Mixed Demonstration

- try: if we start the proof with $dc(aa * 10 \leq bb - 3)$, before loading hypotheses, some simplifications will perhaps occur,

$dc (aa * 10 \leq bb - 3)$

pr

pr

- mixed demonstration: can be faster
- manual demonstration: no try and a better control
- in general (depending on experience) quicker and reusable demonstrations

GOP Simplifications

- example: machine **GopSimpl**
 - this machine is similar to **Manual**, examine both machines,
 - however, in the case of **GopSimpl** all the PO are proved by the GOP
- in some cases, the GOP transforms 1 PO in many simpler PO
 - risk: memory exhaustion
 - statistics: **GopSimpl** **Manual**
11 obvious 6 obvious
0 PO 1 PO

Proof by Attempts

- proof for which the prover tries to demonstrate required hypotheses
- useful demonstrations
 - with force 0, try the following demonstration:
 `mp`
 `tp(Hyp)` (proof by attempts using hypotheses)
 - try also:
 `tp(Goal)` (hypotheses generation from the goal)

Proof by Cases with Enumerated Sets

- principle: in the case of a proof where variables belong to enumerated set, we can try to prove the goal using all the possible values of these variables.
- example: machine `switch` and refinement `switch_1`
 - in the implementation, 3 PO are not proved with force 3,
 - we prove with:
`dc(m1, POSITION) & bb(dc(m2, POSITION) &
bb(dc(m3, POSITION) & bb(pr)))`
 - in fact, we only need to type in:
`mp & bb(dc(m2, POSITION) & bb(pr))`
because `pr` generate `dc` using `m1`

Automatic Proof by Cases

- **mc** Model Checking
 - makes automatically the cases corresponding to all possible values of some variables and call **pr** for each case
 - **mc** tries to collect automatically the variables list
 - **mc(v1, v2, v3)** generates cases for the variables v1, v2 and v3
 - try it on the previous PO **re & mc**

Arithmetical Solver

- arithmetical solver simplifies arithmetical expressions, it depends on the force used for the proof:
 - force 0: simplifying arithmetical predicates: $a = b$
 - force > 0: simplifying arithmetical predicates and expressions
- example: machine **Solv** with a false PO to observe transformations
 - proof with force 0
 - $x1 \leq x3$ remains
 - $x1 + x2 + (3 - 6 * x1) - x3 + 1 \leq x2 + x3$ is simplified
 - hypothesis with \Rightarrow is simplified
 - expression **max** is not simplified
 - proof with force 1
 - expression **max** is simplified

Other Solvers / Simplifiers

- **ap** the Arithmetic Prover
 - to prove inequalities
- **ss** the set simplifier
 - to simplify goals using set expressions
 - **ss** is not a solver! if the goal is simplified into **btrue**, then just use **pr**

User Rules

- User rules should be used when everything else failed
 - their demonstration should be included in the Proof Report
 - the number of rules added should be as small as possible
- Add component **Rules** of the **BASES** project
 - generate the PO
 - start interactive proof (do not start proof with force 0)
- Go to **AssertionLemmas.5**. edit the associated rules file (**Edit PO Method**) and add:

```
THEORY MyRules IS
  0<=a &
  0<=b
  =>
  0<=a+b
END
```

User Rules (2)

- compile this file: `pc` type in `dd` to load hypotheses and apply the rule: `ar(MyRules.1, Once)`
 - the rule has been applied with `a = x1` et `b = x2`
 - goal that appear are the antecedents of the rule, demonstrate them with `pr`
- this rule is not an equivalence: if we apply this rule for `a` and `b < 0`, it leads to a stalemate
 - go to `AssertionLemmas.3` ; `dd & ar(MyRules.1, Once)`
 - the first goal generated is true but the second is false: the proof can not be terminated
- all the rules of the prover are equivalences
 - user rules do not have to be equivalences since their use is driven interactively

User Rules (3)

- joker instantiation (one-letter identifier) is only done within the goal
 - if a joker has to be instantiated by a hypothesis, `binhyp` should be used
- go to `AssertionLemmas.1`, type in `dd` to load hypotheses
- add the rule (`;` is required to separate 2 rules)
`f: S +-> T => S<|f = f`
- compile and apply this rule: `pc & ar(MyRules.2,Once)`
 - displayed goal contains a one-letter identifier: non provable.
- correct the rule
`binhyp(f: S +-> T) => S<|f = f`
- retry: it works

User Rules (4)

- a rewriting rule is used to replace an expression or a predicate
 - if the rule has antecedents, they have to be demonstrated first
- type in `re & ah((0..10)V{} = {}V{}V(0..10))` observe the goal modifications
- add the rule (‘;’ is required to separate 2 rules)
$$A \setminus \{ \} == A$$
- compile and apply this rule: `pc & ar(MyRules.3,Goal) & pr`
 - rewriting has been performed many times
 - be careful: no automatic commutativity

User Rules (5)

- type in `re & ah(max({3}V(10..5)) = 3)` observe the new goal
- add the rule
$$B = \{\} \Rightarrow A \setminus B == A$$
- compile and apply this rule: `pc & ar(MyRules.4,Goal)`
- the first goal is `10..5 = {}` prove it with `pr`
- the next goal is the result of the rewriting
- `re & dd` then `ar(MyRules)`
 - ⇒ `ar(theory name)` enables to apply many times of all the rules which match with the goal

User Rules (precautions 1)

- be careful with type checking

`x-x == 0`

→ problem if `x` is a set

- introduce only well-defined expressions

`card(A) <= n &`

`card(B) <= n`

`=>`

`card(A /\ B) <= n`

→ problem if `A` or `B` is infinite

User Rules (precautions 2)

- be careful with names capture

```
binhyp(x = 1)
```

```
=>
```

```
x-1 == 0
```

→ problem if x is a quantified variable: the x replaced may not be the one equal to 1!

```
var = 1
```

```
=>
```

```
!var.(var : INT => var - 1 : INT)
```

→ to correct this rule, we have to make sure that x is not a quantified variable

```
binhyp(x = 1) &
```

```
blvar(Q) &
```

```
Q\ $x$ -1
```

```
=>
```

```
x-1 == 0
```

Trying Out Automatically Interactive Demonstrations

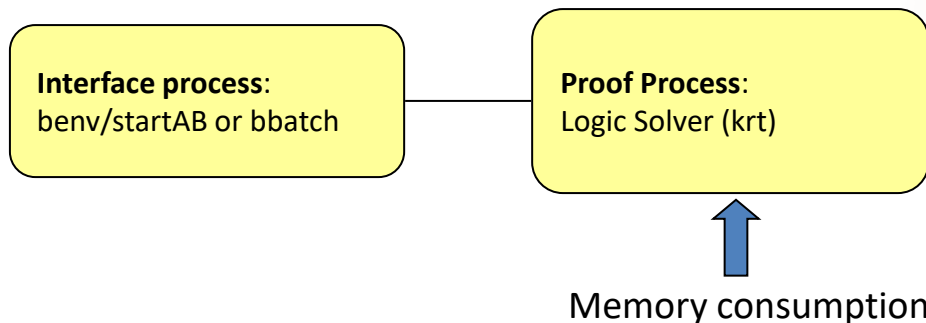
- to try a successful demonstration on other unproved PO of the same component
 - `te` (try everywhere)
 - filter the unproved PO:
 - of the same clause/of all the component
 - with a goal matching some predicate
- to try some demonstrations on components
 - some small general demonstrations: `pp`;
`dd(0)&pp(rp.1)`; `mp&tp(Hyp)`
 - in the User_Pass theory of a component pmm file
or in the PatchProver file for the whole project



PROOF

Management and Administration

Memory Management



- configure the kernel depending on the target computer
 - starting with huge memory size (quicker than dynamic allocation)
- maximum expansion factor (to stop in case of a loop)
 - how to get back memory after a proof: [Quit Project](#)

Logic Solver

- use resources (by user, by project)
- multiple defined resources are not overloaded (the first one is kept)
- prefer to choose a large memory model
 - if you notice dynamic allocation, increase the memory parameters
 - choose a greater value than the one that has been allocated

Saving Proofs

- proving is a long work ⇒ SAVES
 - preferred method: use Archive Project
 - in the project window
 - Restore Project
 - choose archive name
 - create the destination directory (the one which will contain spec, pdb, lang)
 - select this directory
 - choose a name for this project
- ⇒ project is installed and the status of the project is kept

Saving Proofs (2)

how to restore a proof status and to preserve modifications?

- restore the project in an other directory
- copy the modified files in this new directory
- **Prove** force 0 then **Prove Replay**
 - having generated the POs, the “merge” function preserves unchanged POs
 - proof with force 0 proved 70 % of the new POs
 - **Replay** retry demonstrations on the POs that have changed

Saving Proofs (3)

- simplifying - effects
 - a component is proved
 - then a new independent part is added to the invariant

since the invariant is stronger, the merger preserves proved POs
 - then this invariant part is suppressed

the invariant is weaker: **ALL POs ARE RESET TO “UNPROVED”**
(global Prove Replay required)
- it is far better to go back to the previously saved project
 - save the project regularly, when it is stable

Replay

proof replay is very important to

- final
 - verifying that a project is completely proved
 - work
 - check demonstrations with new hypotheses
- PO proved by a demonstration
the component is modified: goal remains, new hypotheses
merge: stronger hypotheses, the PO remains proved
new hypotheses can modify the proof tree and prevent to replay
the demonstration
example: demonstration with `pp`. New hypotheses: Maximum
allowed time can be reached. Use preferably `pp(rp.0)` or `pp(rp.1)`

Replay (2)

- project **MANAGEMENT**: add the component **Unknown**, prove with force 0
 - 2 PO: 1 false (problem identified, to admit), 1 needs a manual rule
- start interactive proof , 1st unproved PO
- edit user rules: rule **p** and rule **$2^{**}8 = 256$**
- **mp & ar(MyRules.2, Goal) & pr** , first PO proved
- **ne & ar(MyRules.1, Once)**, second PO admitted
- proof done: **Unprove Replay** correct
- modify manual rules: suppress **p**
- **Unprove Replay**: 0 PO proved
- rules numbers are no longer the same: demonstration of PO1 is lost!

Replay (3)

- completely proved project:
 - user rules verified
 - successful global Unprove/Replay
- advice: replay at least once a week!

Admitted Rules

Admit a PO = differ its demonstration

- not to slow automatic proof of other components
- if a PO induces a loop with automatic proof: enables to continue proof
- component **Unknown**: contains a false PO that we would like to admit
- start interactive prover, go to this PO
- **Edit PO Method**: add this rule in the theory **Admitted**

```
binhyp(c.o.n) &  
bcall(WRITE: bwritef("\nPO % admitted\n", c.o.n))  
=>  
p
```

- **pc & ar(Admitted)**, the PO is admitted, a message is displayed



PROOF

Pitfalls

Atelier B Terminal Window

- it is important to know what is going on:
look regularly at the console
- messages related to memory allocation
- messages related to type check errors (pp)

Symbols Not/Badly Covered

- some rarely used expressions are badly supported by the prover
 - *the prover is able to demonstrate only few things about them*
- component **Unknown** of **BASES** project: 1 unproved PO with force 3
- start interactive proof, go to this PO, **mp**:
 - we should demonstrate $2^{**}8 = 256$
- **sr(All,(x**y))**, searching rules related to power
 - no rule enables to compute the value (the arithmetical solver simplifies expressions like $a^{**}0$ and $a^{**}1$), try **ah(2**1 = 2)**
- manual rules should be added

Expressions Not/Badly Covered (2)

- User rules: different possibilities
 - $2^{**}8 == 256$ rule easily validated, but what about its reusability?
 - $\text{bnum}(x) \ \& \ \text{btest}(n \geq 2) \Rightarrow (x^{**}n == x^{*}(x^{**}n(n-1)))$ simple, general rule but should be applied many times: use $\text{pr}(\text{Tac}(\text{Expo})) \ \& \ \text{ah}(2^{**}1 = 2) \ \& \ \text{pr} \ \& \ \text{pr}$
 - $\text{bnum}(x) \ \& \ \text{btest}(n \geq 2) \ \& \ \text{bguard}((\text{ARI}; \text{RES}): \text{bresult}(n-1), m) \Rightarrow (x^{**}n == x^{*}(x^{**}m))$ we are programming the prover!