The Atelier B Proof System and Its Improvements

Presentation · November 2018

DOI: 10.13140/RG.2.2.35928.52480

1 author:

Thierry Lecomte
ClearSy System Engineering

The Atelier B Proof System and Its Improvements

Salvador , November 26th 2018

Thierry Lecomte
R&D Director, ClearSy

thierry.lecomte@clearsy.com

The Atelier B Proof System and Its Improvements

- Intro to B method
- Proof System
- Improvements

The Atelier B Proof System and Its Improvements

Intro to B method

Proof System

Improvements

Intro to B method

Intro to B method

B METHOD

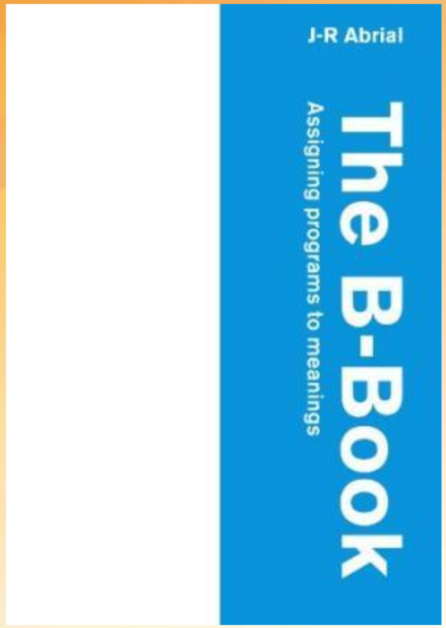
http://www.methode-b.com/

Invented by French mathematician (J. R. Abrial)

Assigning programs to meanings

Top-down approach

Programs are proved to comply with their specification



ATELIER B

http://www.atelierb.eu/

V4.5 β_{13}
for the tutorial

Windows

Implement B Method

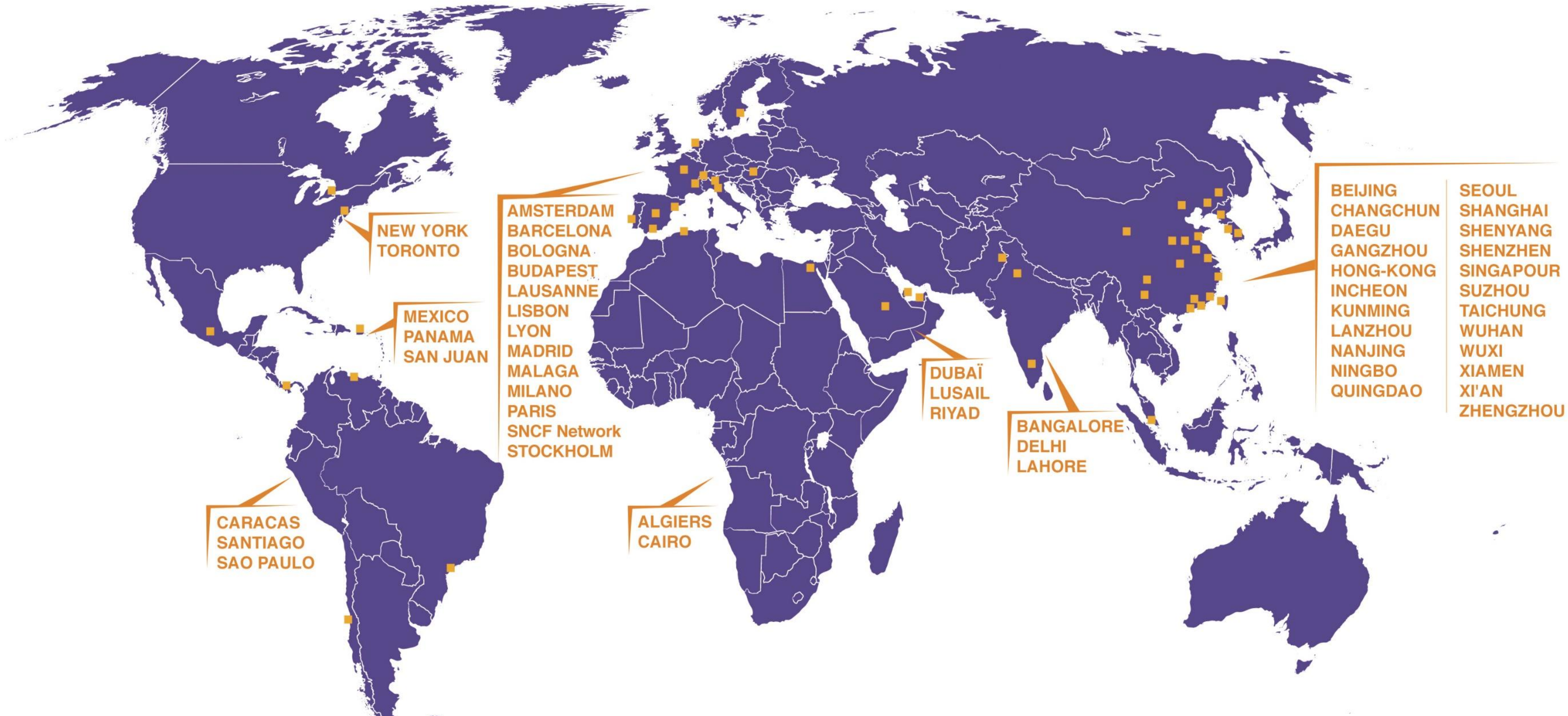
First autonomous metro Paris L14 Meteor (1998)

30% of automatic metro worldwide

Maintained & developed by CLEARSY

ATELIER B

METROS AND TRAINS EQUIPPED WITH **B SIL4 SOFTWARE**



Intro to B method

« Only inactive sequences can be added to the active sequences execution queue. »

Natural language requirement

Behaviour + properties

```
activation_sequence = /* Activation d'une séquence non active */
PRE ¬(sequences = sequences_actives) THEN
    ANY sequ WHERE
        sequ ∈ sequences - sequences_actives
    THEN
        sequences_actives := sequences_actives ∪ {sequ}
    END
END;
```

B Specification

Behaviour + properties

```
activation_sequence = /* Activation d'une séquence non active */
VAR sequ IN
    sequ <-- indexSequenceInactive;
    activeSequence(sequ)
END;
```

B Implementation

```
void M0__activation_sequence(void)
{
    CTX__SEQUENCES sequ;

    sequence_manager__indexSequenceInactive(&sequ);
    sequence_manager__activeSequence(sequ);
}
```

C generated code

```
0x01F970 FFFF 8B4C 2440 89C5 8D7D 0C8B 4110 89CE
0x01F980 83C6 0C8D 1485 0000 0000 8D42 0883 F807
0x01F990 7617 F7C7 0400 0000 740F 8B41 0C8D 7D10
0x01F9A0 83C6 0489 450C 8D42 04FC 89C1 C1E9 02F3
```

Binary code

CLEARSY
SYSTEMS ENGINEERING

Intro to B method

« Only inactive sequences can be added to the active sequences execution queue. »

Natural language requirement

```
activation_sequence = /* Activation d'une séquence non active */
PRE ¬(sequences = sequences_actives) THEN
    ANY sequ WHERE
        sequ ∈ sequences - sequences_actives
    THEN
        sequences_actives := sequences_actives ∪ {sequ}
    END
END;
```

B Specification

Proof (coherence)

Proof (refinement)

```
activation_sequence = /* Activation d'une séquence non active */
VAR sequ IN
    sequ <-- indexSequenceInactive;
    activeSequence(sequ)
END;
```

B Implementation

Proof (coherence)

```
void M0__activation_sequence(void)
{
    CTX__SEQUENCES sequ;

    sequence_manager__indexSequenceInactive(&sequ);
    sequence_manager__activeSequence(sequ);
}
```

C generated code

Cyclic software single-thread

0x01F970	FFFF	8B4C	2440	89C5	8D7D	0C8B	4110	89CE
0x01F980	83C6	0C8D	1485	0000	0000	8D42	0883	F807
0x01F990	7617	F7C7	0400	0000	740F	8B41	0C8D	7D10
0x01F9A0	83C6	0489	450C	8D42	04FC	89C1	C1E9	02F3

Binary code

CLEARSY SYSTEMS ENGINEERING

B METHOD

http://www.methode-b.com/

$\equiv$ Model is formal

- Model is text-based
- Same mathematical language (B) used for specification model and implementation model
- Uses set-theory ($A \subseteq B$) and predicates logic ($P \Rightarrow Q$)
- Static aspect: properties
- Dynamic aspect: behavior

B METHOD

http://www.methode-b.com/

≡ [Formal] : it relies on a mathematical model of the software, containing both what the software is expected to do and its algorithm

≡ The model is mathematically [proved]

The algorithm doesn't contradict its specification

$tab \in 0..9 \rightarrow \mathbb{N}$

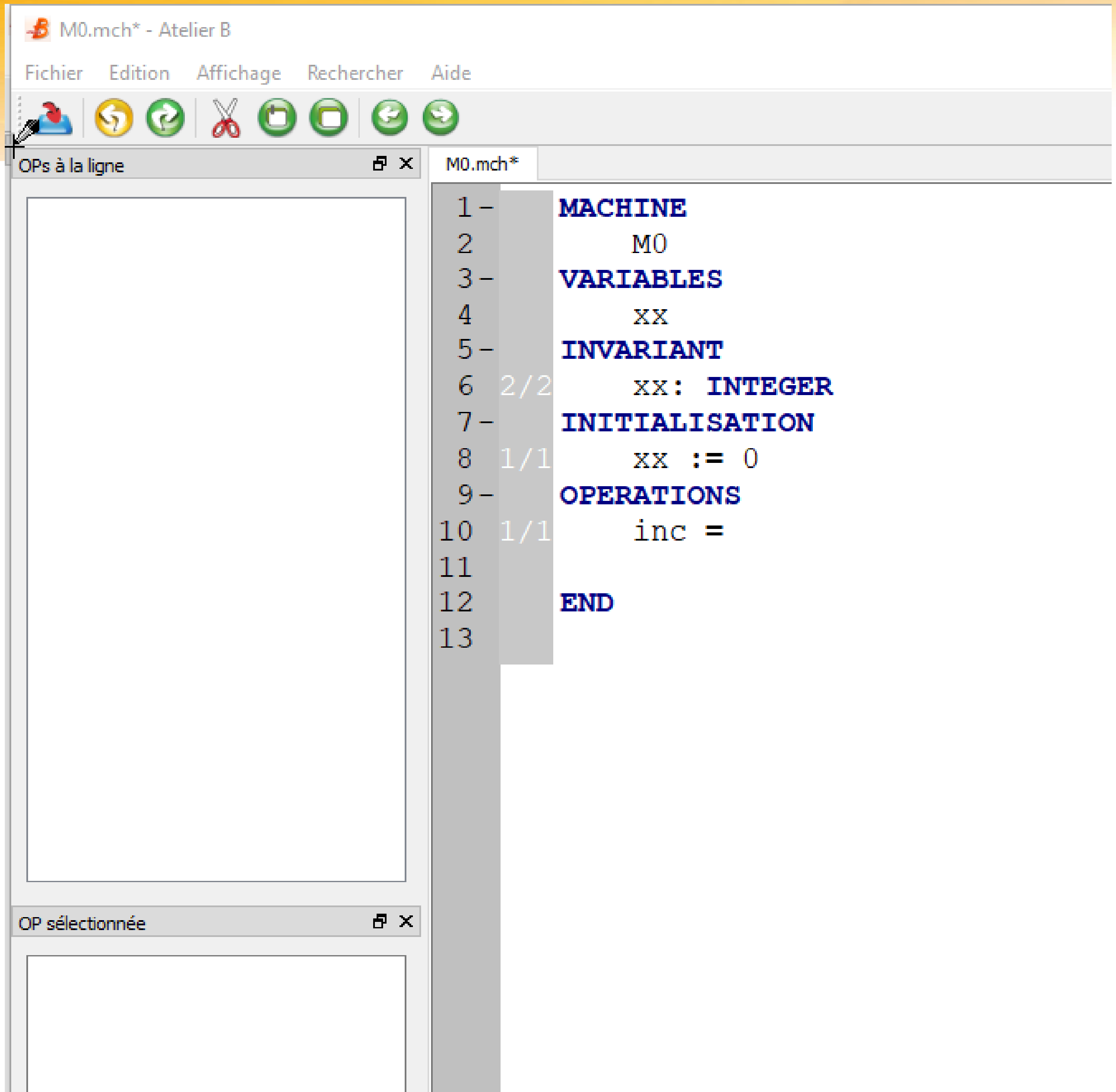
$sort \equiv tab \in (tab \in 0..9 \rightarrow \mathbb{N} \wedge \forall x.(x \in 0..8 \Rightarrow tab(i) >= tab(i+1)))$

implementation could be a bubble sort, a quick sort, etc.

≡ Loops terminate

Decreasing positive VARIANT,

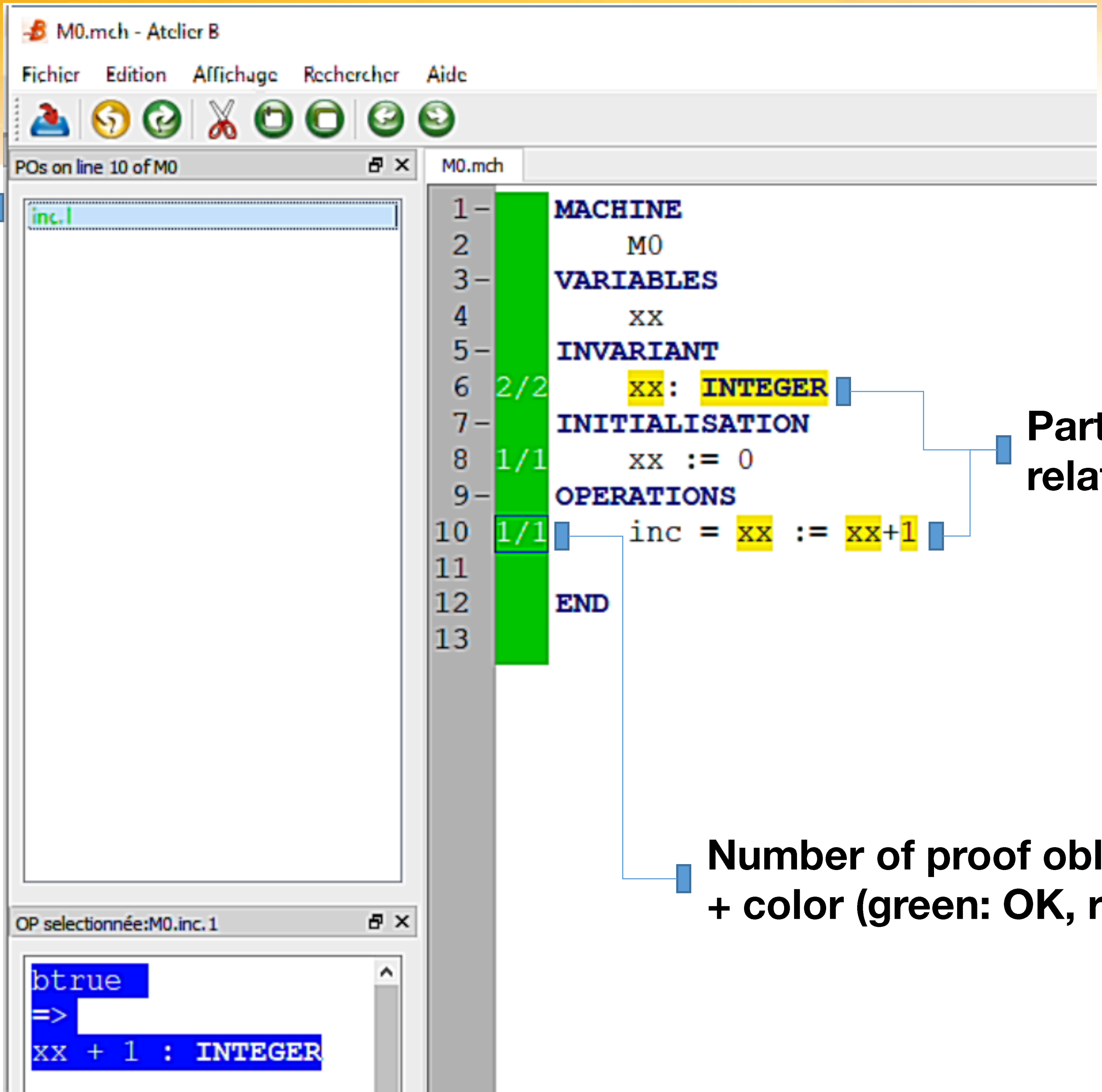
Intro to B method



Intro to B method

Proof obligations list

Proof obligation



Parts of the model related to the proof

Number of proof obligations related to the line + color (green: OK, red: NOK)

CLEARSY SYSTEMS ENGINEERING

Intro to B method

MACHINE = specification

M0.mch*

```
MACHINE
    M0
VARIABLES
    xx
INVARIANT
    xx: INTEGER
INITIALISATION
    xx := 0
OPERATIONS
    inc =

END
```

IMPLEMENTATION = algorithm

M0.mch | M0_i.imp*

```
IMPLEMENTATION M0_i
REFINES M0

CONCRETE_VARIABLES
    xx
INVARIANT
    xx: INT
INITIALISATION
    xx := 0

OPERATIONS
    inc =

END
```

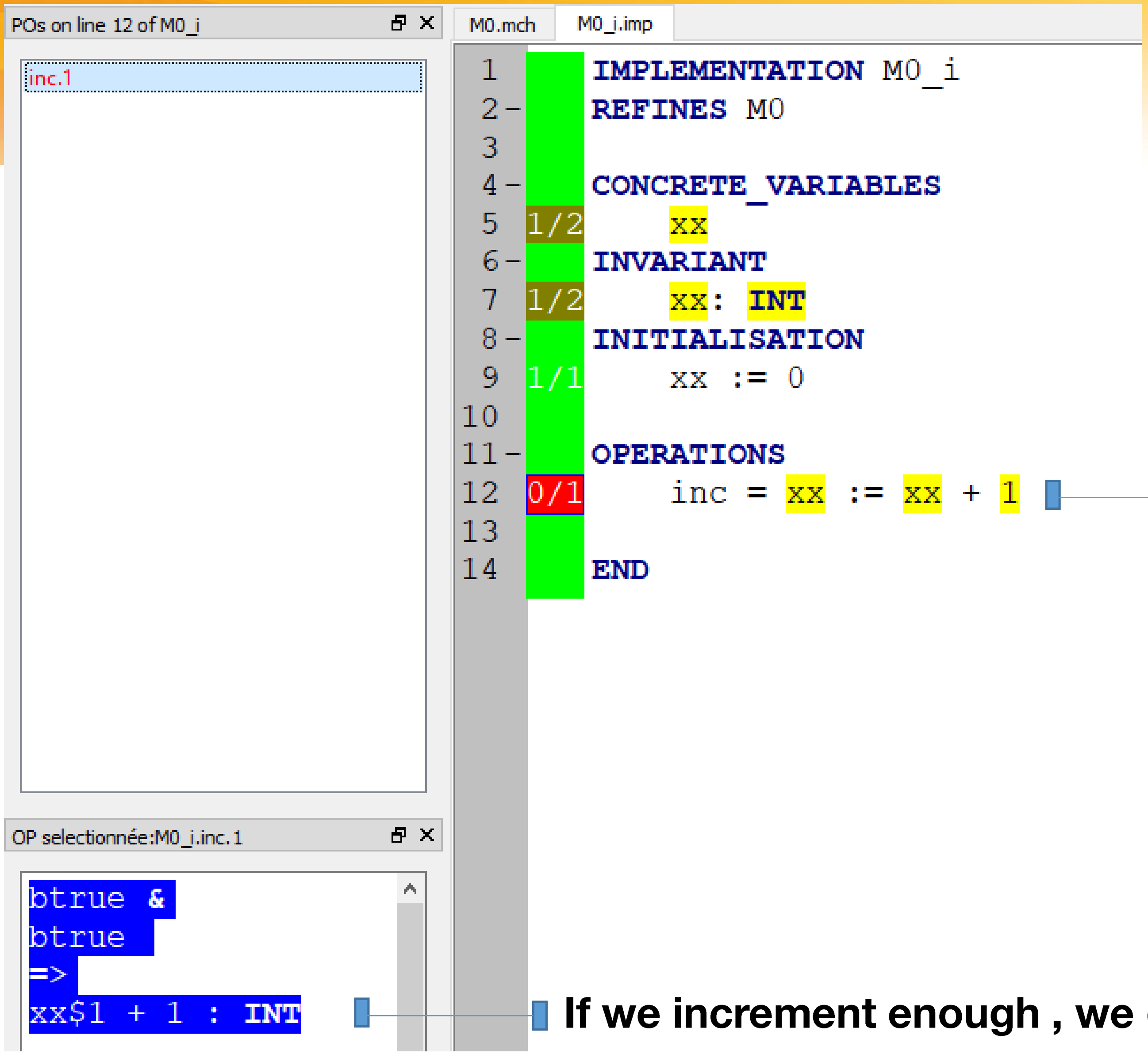
Implement its specification

Implemented variable

Implementable type

CLEARSY SYSTEMS ENGINEERING

Intro to B method



Endless increment

If we increment enough , we get out of type INT

Intro to B method

POs on line 12 of M0_i

- inc.1
- inc.2
- inc.3

OP selectionnée:M0_i.inc.1

```
btrue &
xx$1 = MAXINT &
btrue
=>
xx$1 + 1 = 0
```

M0.mch | M0_i.imp

```
1       IMPLEMENTATION M0_i
2       REFINES M0
3
4       CONCRETE_VARIABLES
5   3/4     xx
6       INVARIANT
7   3/3     xx: INT
8       INITIALISATION
9   1/1     xx := 0
10
11      OPERATIONS
12  2/3     inc = IF xx = MAXINT THEN xx := 0 ELSE xx := xx + 1 END
13
14      END
```

If we get to the upper bound (MAXINT) then we reset else we increment

If we get to the upper bound then the value should be 0

CLEARSY
SYSTEMS ENGINEERING

Intro to B method

M0.mch | M0_i.imp

```
MACHINE
     M0
VARIABLES
     xx
INVARIANT
     xx: INTEGER
INITIALISATION
     xx := 0
OPERATIONS
     inc = CHOICE xx := 0 OR xx := xx + 1 END

END
```

Either we increment or we reset

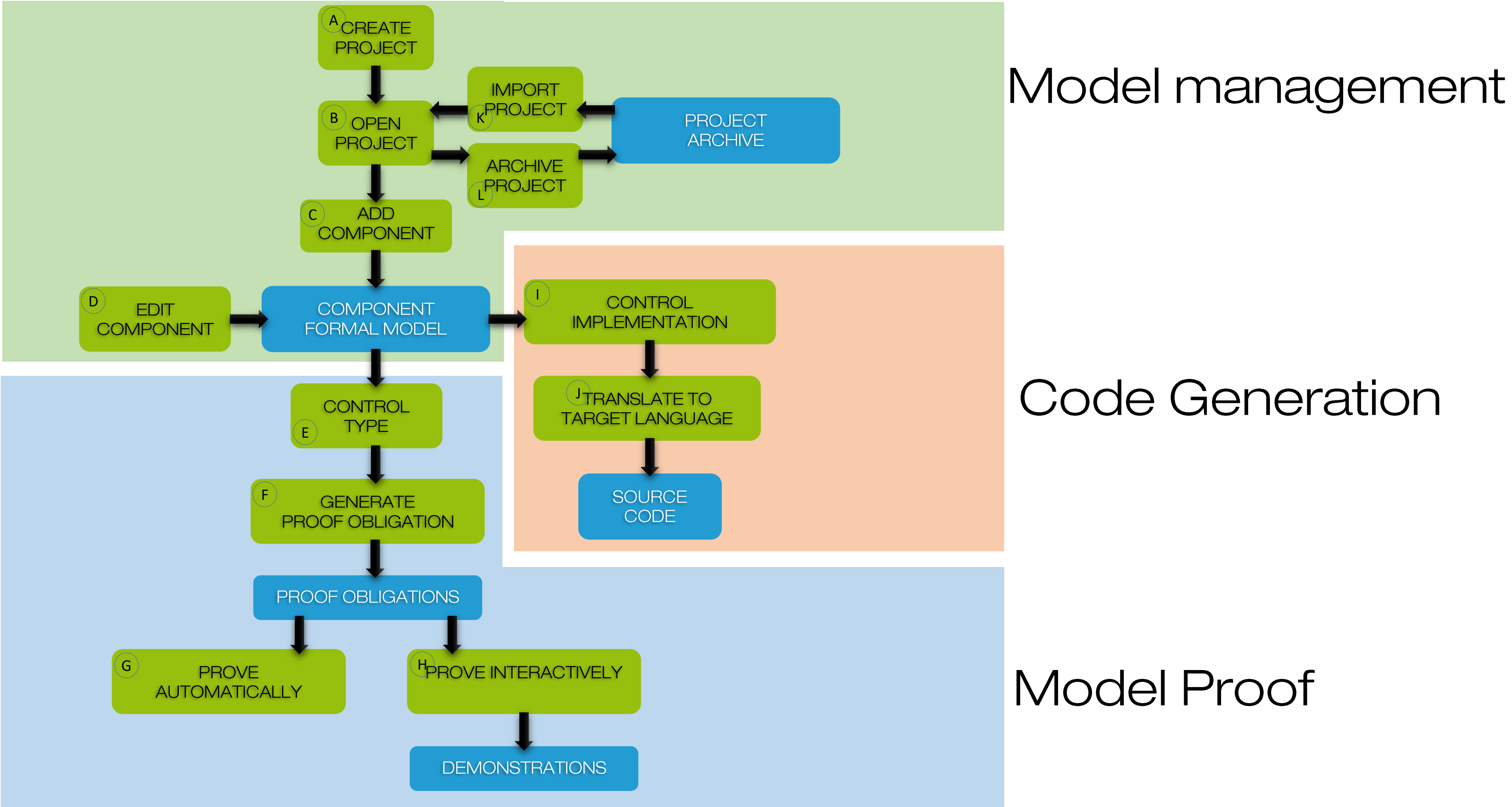
The 2 models
are compatible

M0.mch | M0_i.imp

```
IMPLEMENTATION M0_i
REFINES M0

CONCRETE_VARIABLES
     xx
INVARIANT
     xx: INT
INITIALISATION
     xx := 0

OPERATIONS
     inc = IF xx = MAXINT THEN xx:=0 ELSE xx := xx +1 END
END
```

Intro to B method

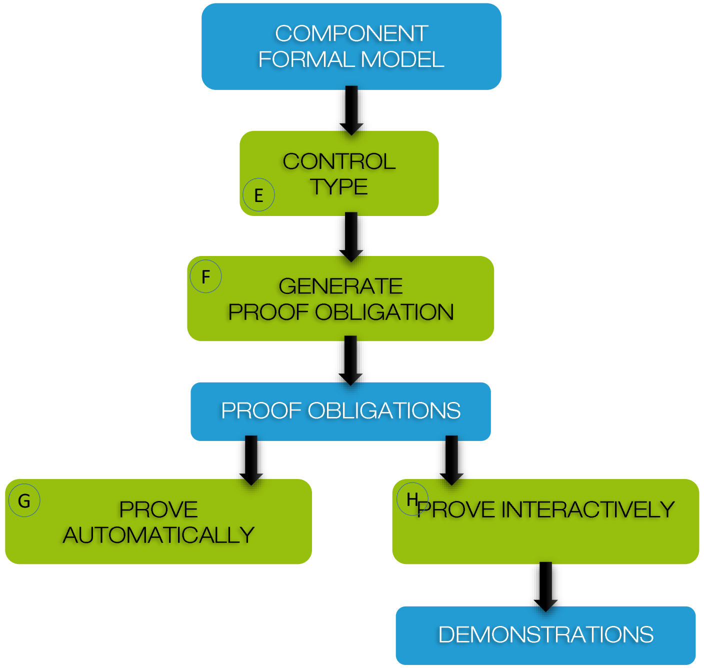
≡ Development Cycle



CLEARSY SYSTEMS ENGINEERING

Intro to B method

Development Cycle

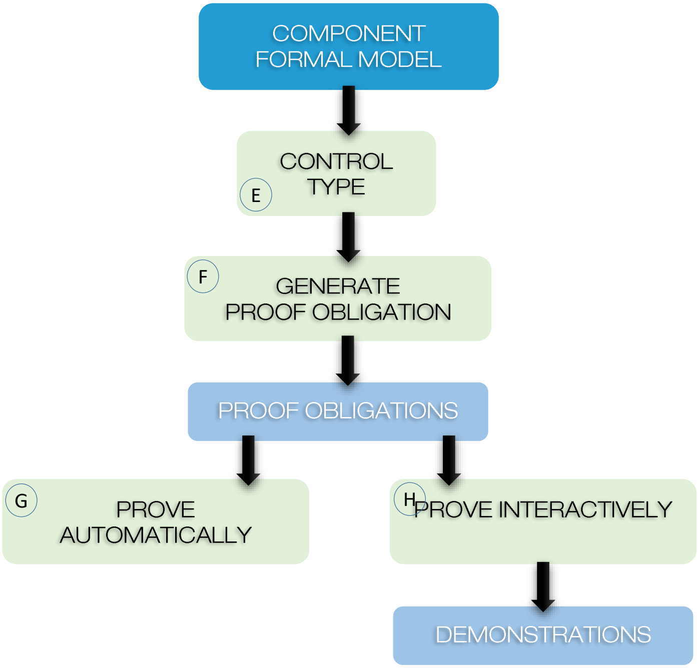


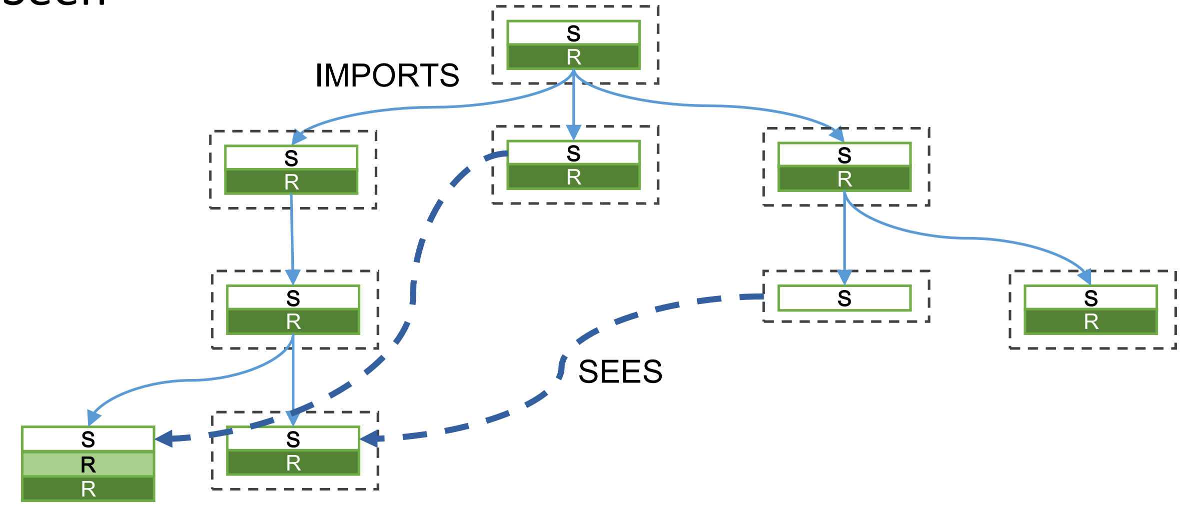
Intro to B method

≡ Development Cycle

- **A B project is made of components** (models)

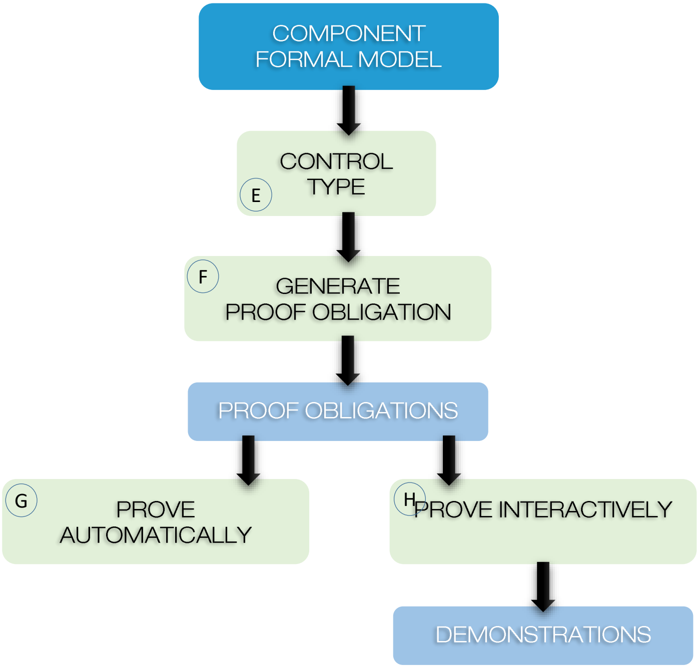
- **Models can be**:
  - Refined
  - Decomposed
  - Seen



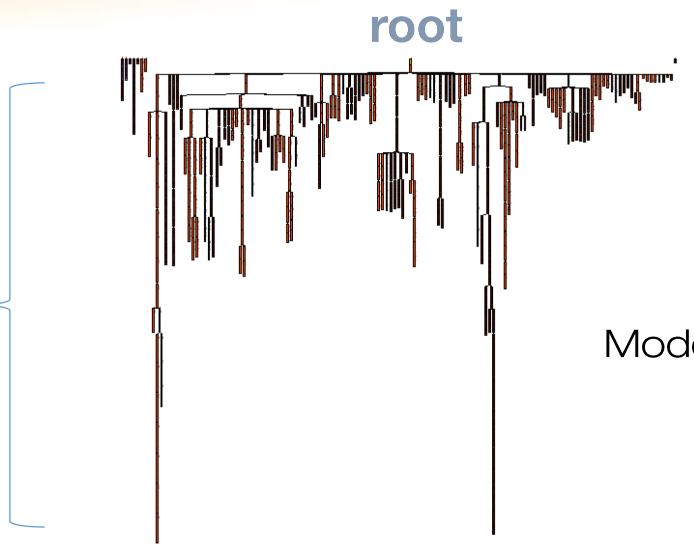


Intro to B method

Development Cycle



Decomposition graph



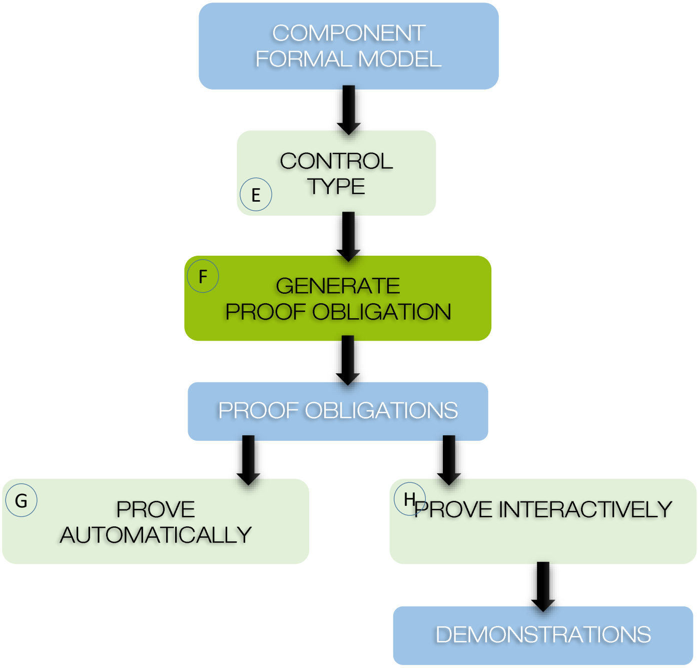
Modern Automatic Train Protection Software (2015)

Metrics

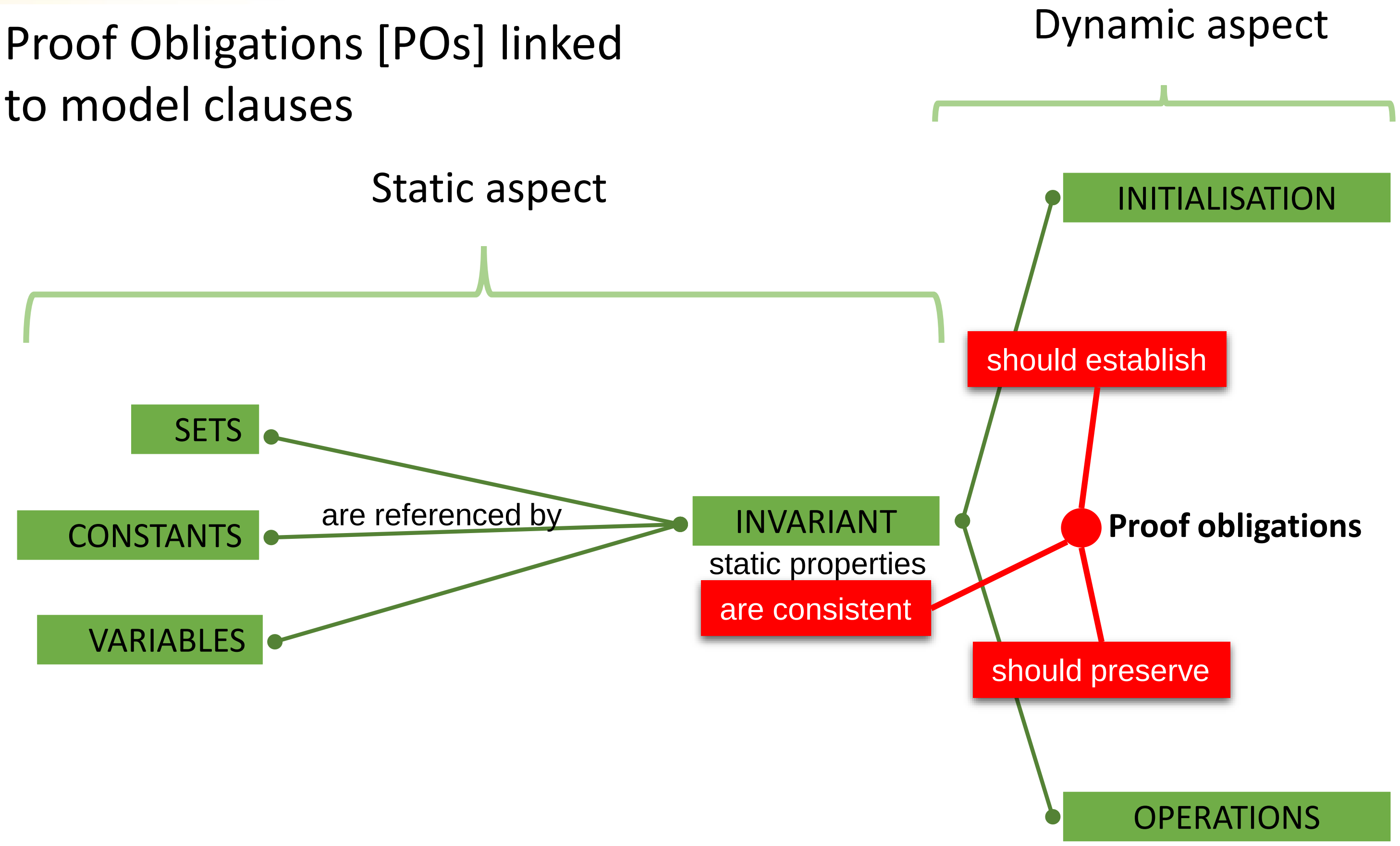
- 233 machines, 50 kloc
- 46 refinements, 6 kloc
- 213 implementations, 45 kloc
- 3 000 definitions
- 23 000 proof obligations (83 % automatic proof)
- 3 000 added user rules (85 % automatic proof)

Intro to B method

Development Cycle

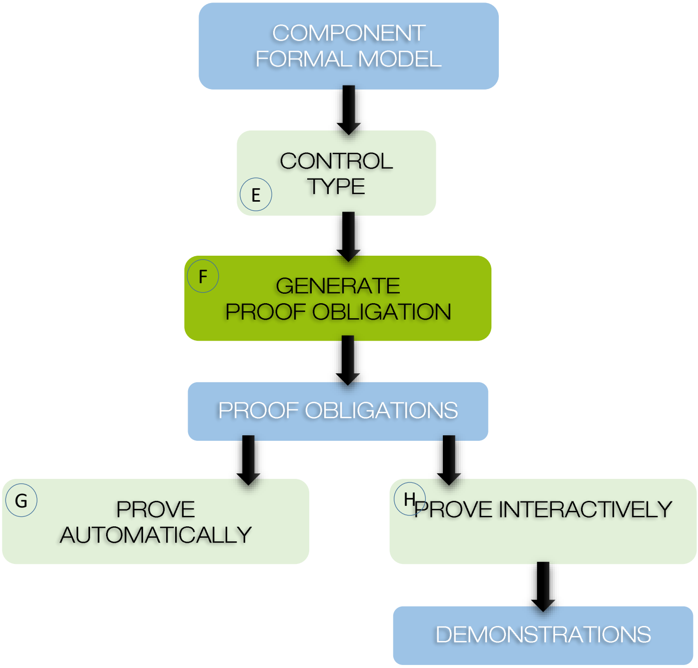


- Proof Obligations [POs] linked to model clauses

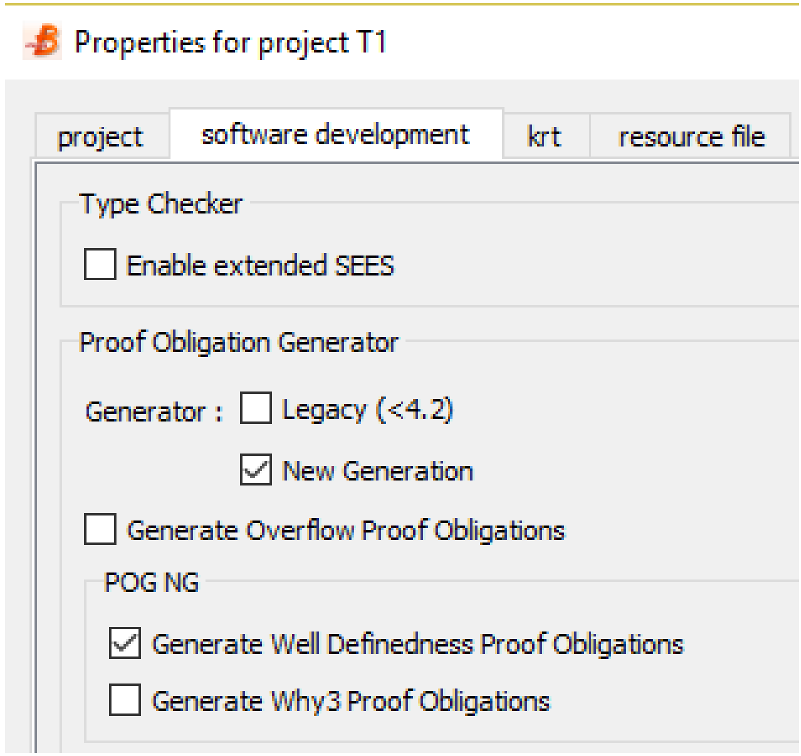


Intro to B method

≡ Development Cycle



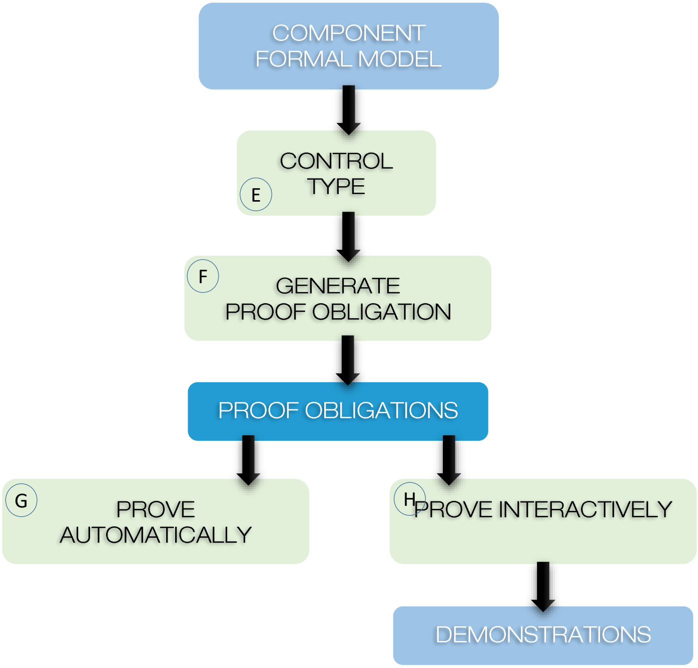
- Proof Obligations [POs] linked to model clauses
- POs **fully, automatically generated**
  - Functional
  - Well-definedness
  - Overflow (option)
- **2 PO generators**
  - < 4.2 (Legacy)
  - New Generation (default) required for PO traceability



CLEARSY
SYSTEMS ENGINEERING

Intro to B method

≡ Development Cycle



- **POs generated per component**
  - Lower impact of model modifications
- **Limitation: 3 000 POs per component**
  - Good practice (frequent modifications)
- **General form**

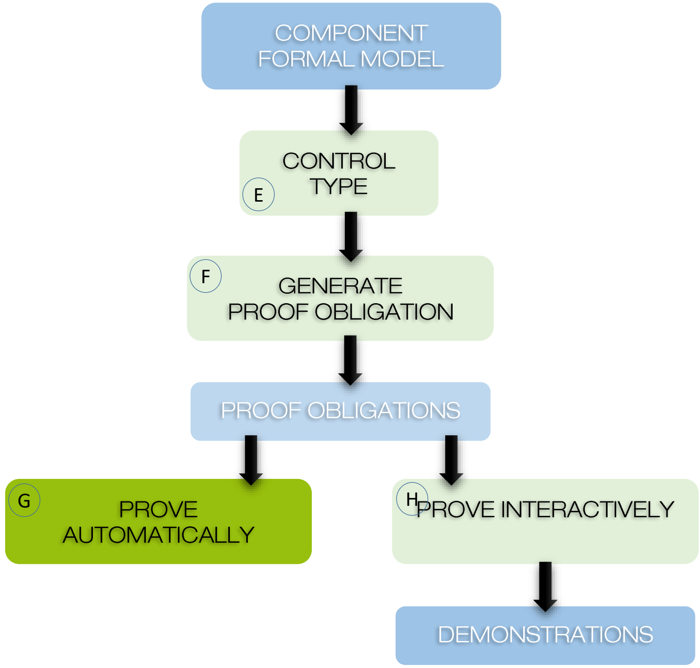
  Global hypotheses => (Local hypotheses => Goal)

  Potentially **100x (1000x) (10000x)** global hypotheses

  Most hypotheses do not help to prove
- **POs merged** when refactoring models

Intro to B method

≡ Development Cycle



- **Atelier B main prover**
  - Also used by the Rodin platform (Event-B)

- **Initial industry-ready specification**
  - Able to support full automatic train protection software proof
  - 10 seconds per PO mean time
  - Optimized PO loading per clause (design)

    Ex: when moving from one operation to another, global hypotheses are kept in memory

- **Forces from 0 to 3**
  - 1 to 3 are likely to enter infinite loops

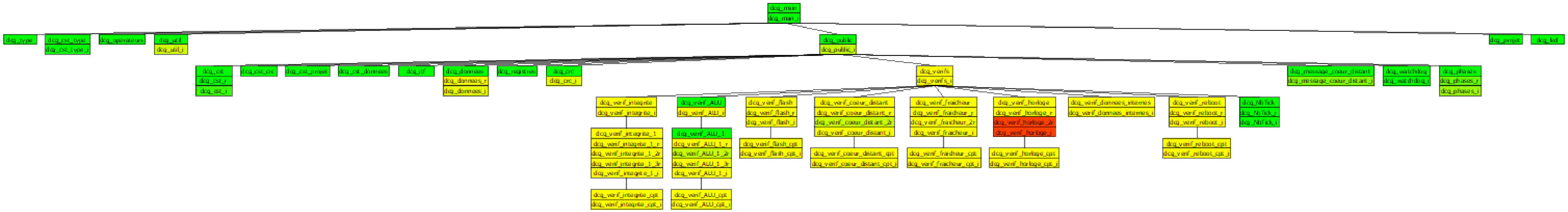
- **Proved POs are supposed true**
  - Unproved POs have to be investigated
  - Proved PO % as a quality indicator

CLEARSY
SYSTEMS ENGINEERING

≡ Proof Process

The project graphical view displays the automatic proof status of the project

- Green: fully proved – red: not proved at all



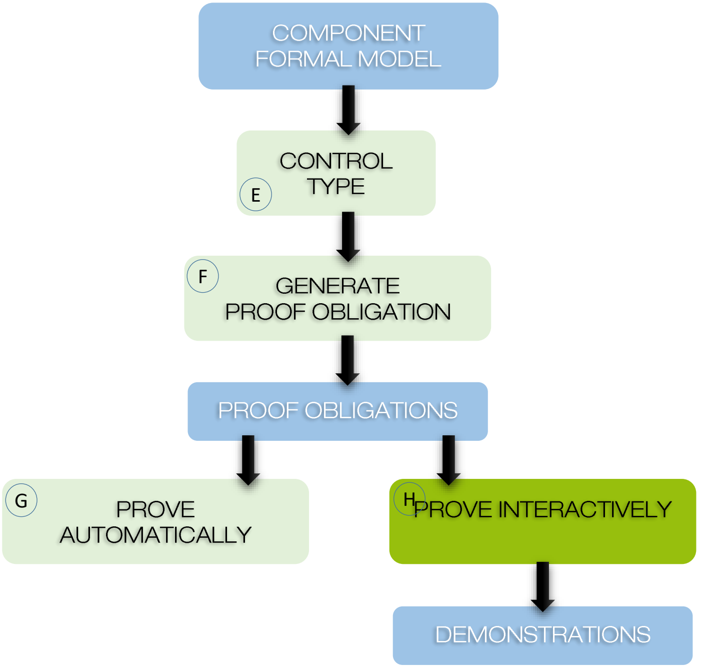
Visual inspection may then be performed on yellow, orange and red components

Intro to B method

Development Cycle

- **Interactive prover**
  - Proof commands
  - Call to automatic prover (force 0 to 3)

  - Addition of mathematical rules

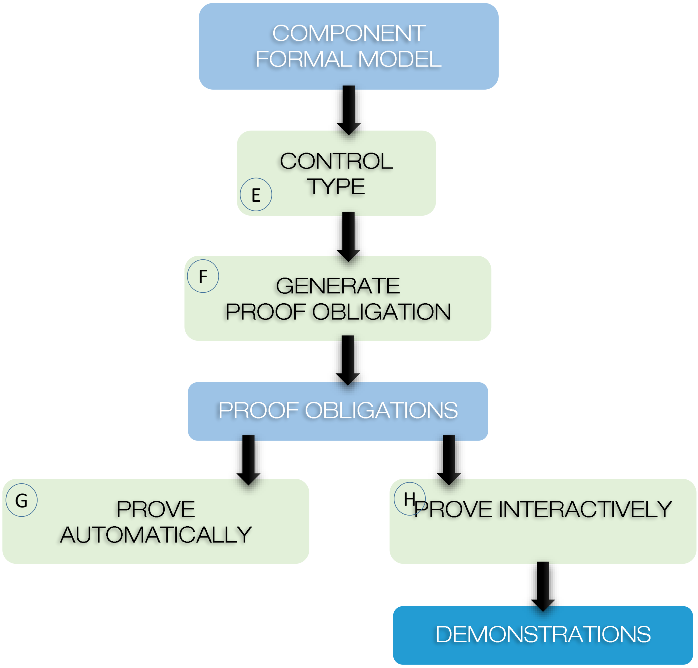


Intro to B method

Development Cycle

- **Successful proof scripts are saved**
  - Proof replay to obtain 100% proved projects
  - Avoid to lose demonstration when refactoring the models

- **Definition of generic proof scripts** (tactics)



Development Cycle

- **Industrial needs**
  - Higher level of proof automation
  - Quick interactions with the designer
  - Objective: 100% proof for a project (automatic + interactive)
  - Everything demonstrated (models, added rules)

- **Certification needs**
  - Ability to replay proof process
  - Tools certification not mandatory (only the process is evaluated) [Railways]

CLEARSY
SYSTEMS ENGINEERING

The Atelier B Proof System and Its Improvements

Intro to B method

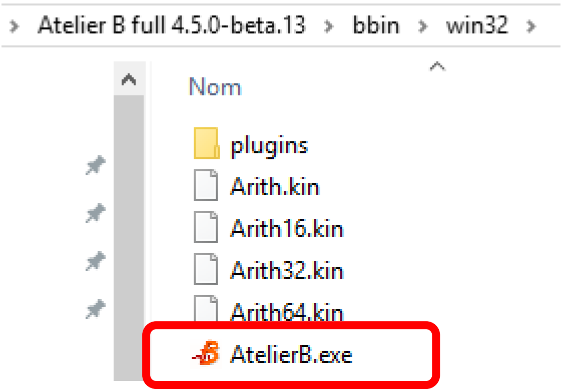
Proof System

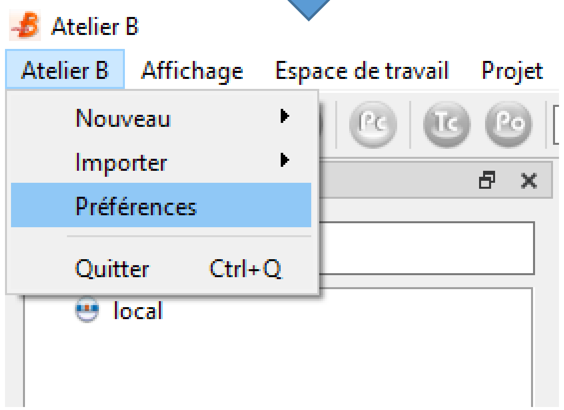
Improvements

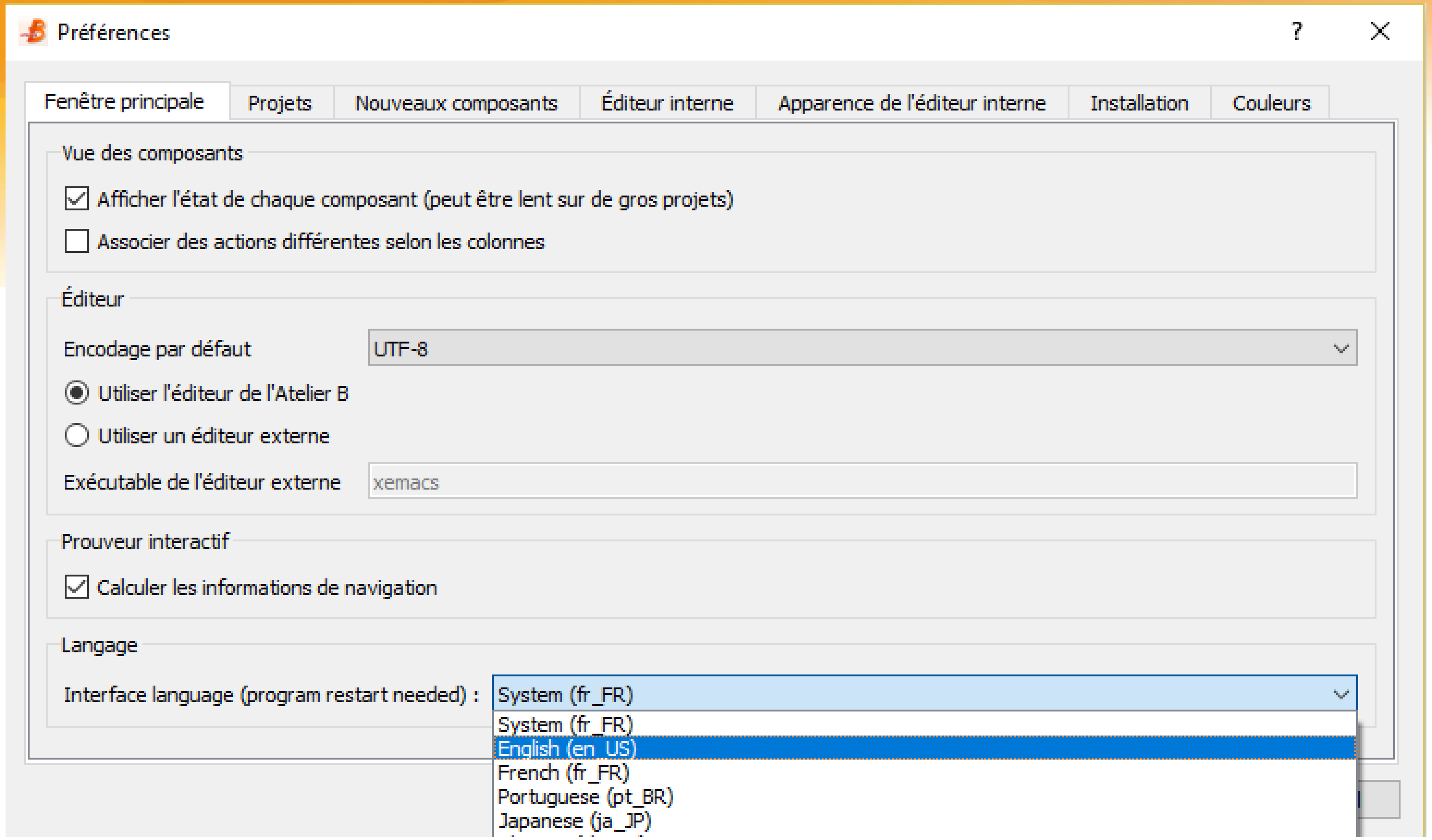
Proof System

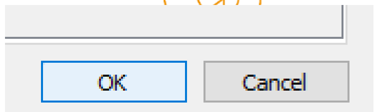
Proof System

Change UI language







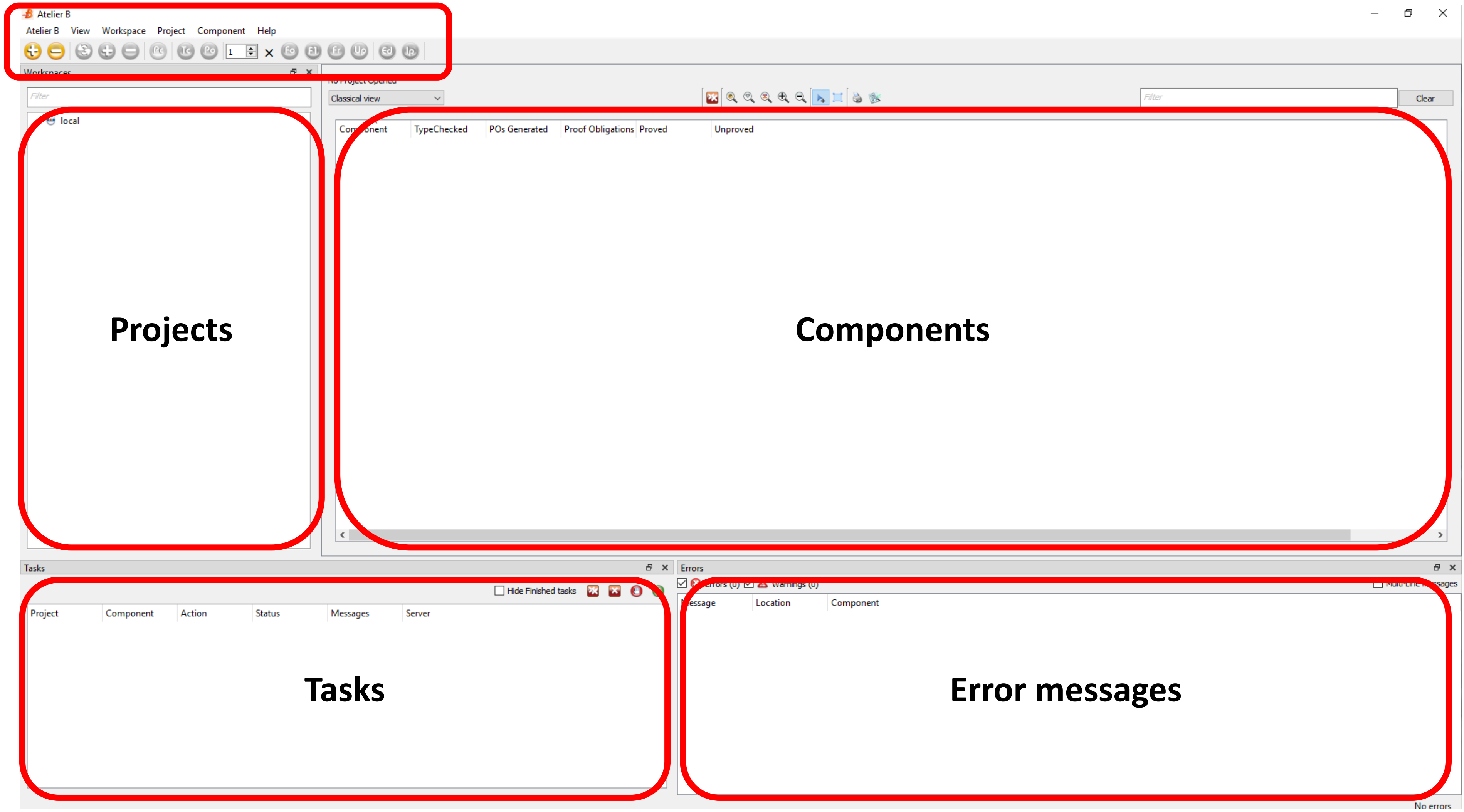


Quit Atelier B then **restart**:
the interface is now is English

Proof System

Initial UI in English

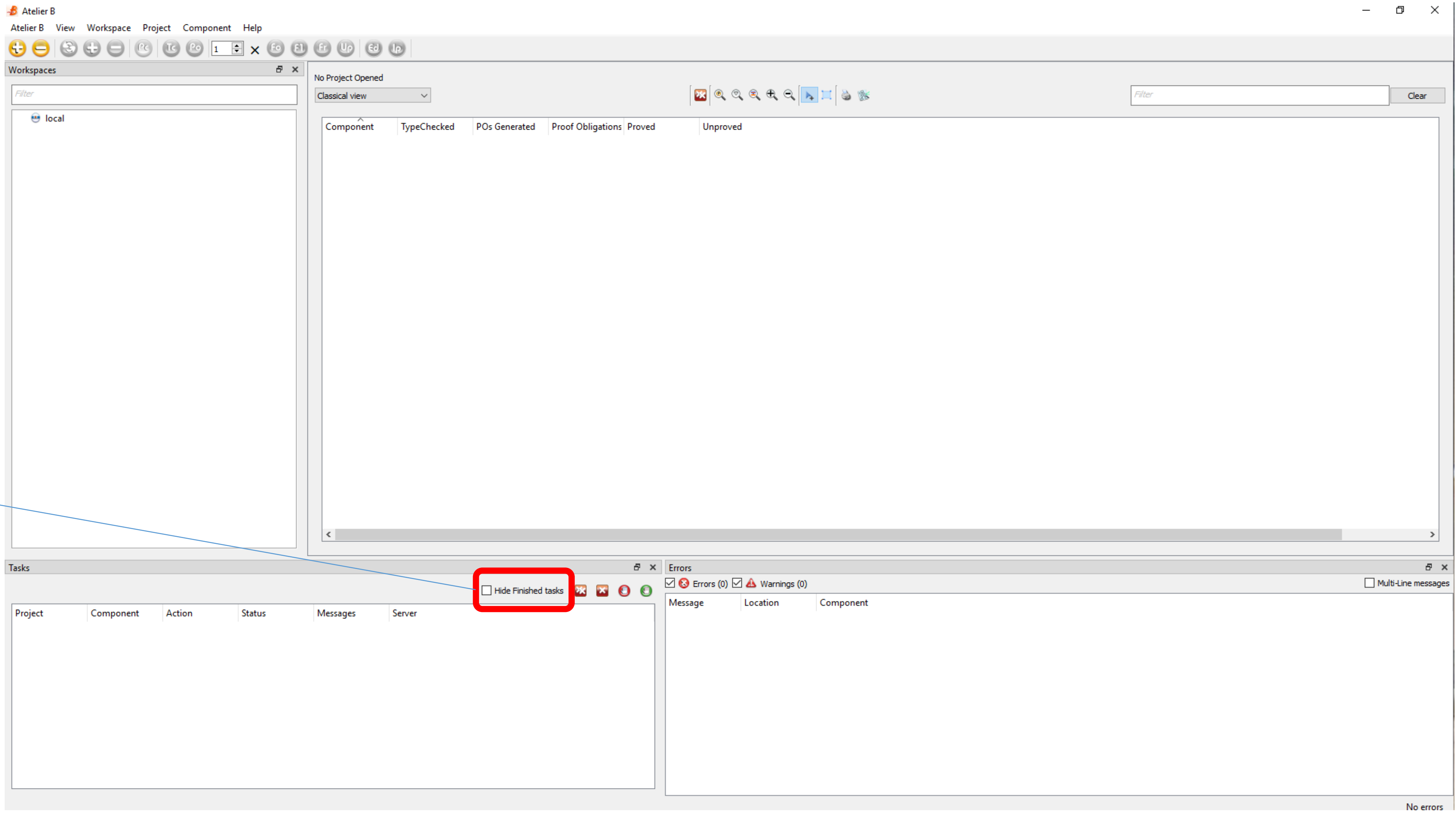
Commands



Proof System

≡ UI Tweak 1

Check « Hide finished tasks »
to avoid finished tasks cluttering



Proof System

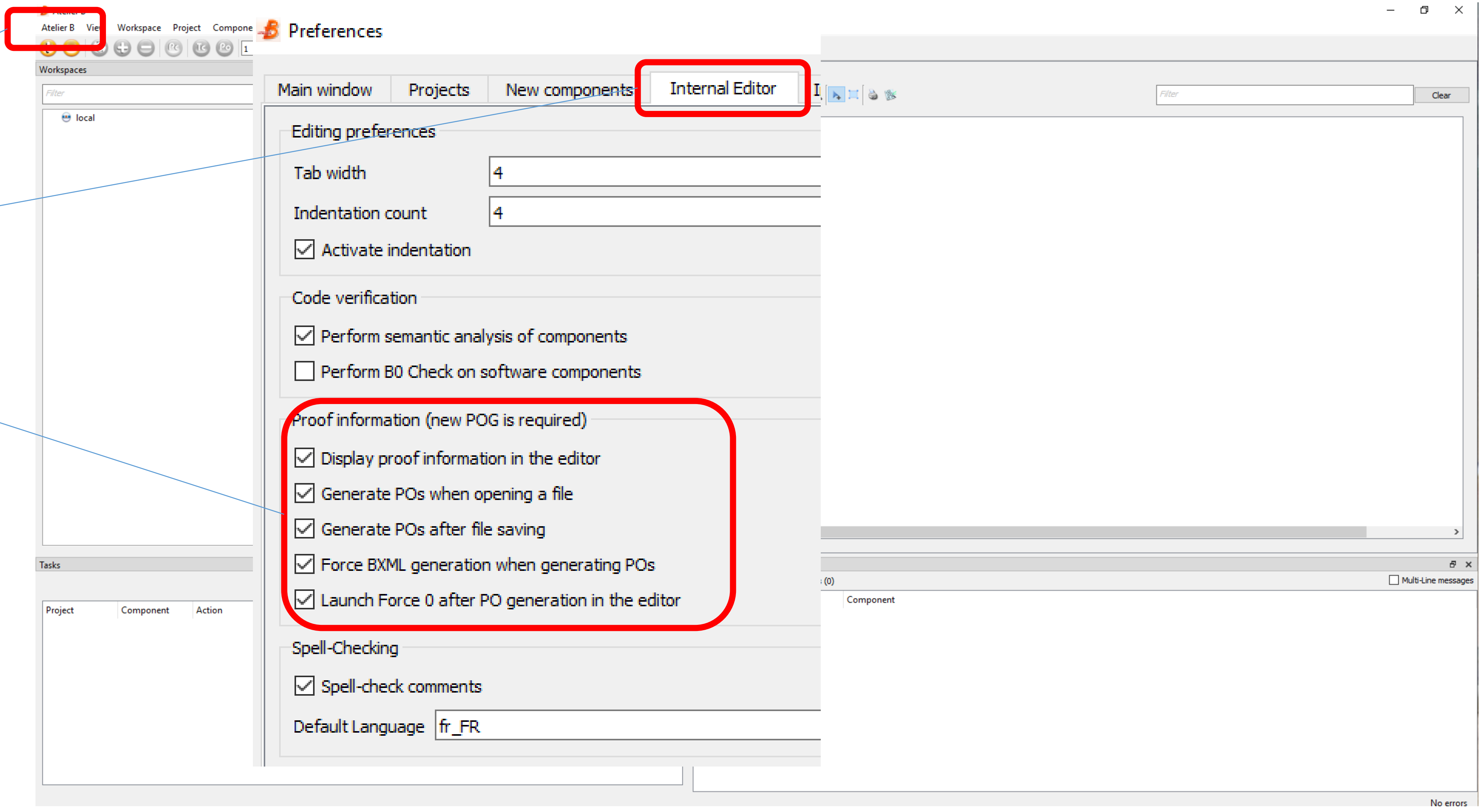
UI Tweak 2

Select « Atelier B / preferences »

Select « Internal Editor »

Check all items of « Proof information »

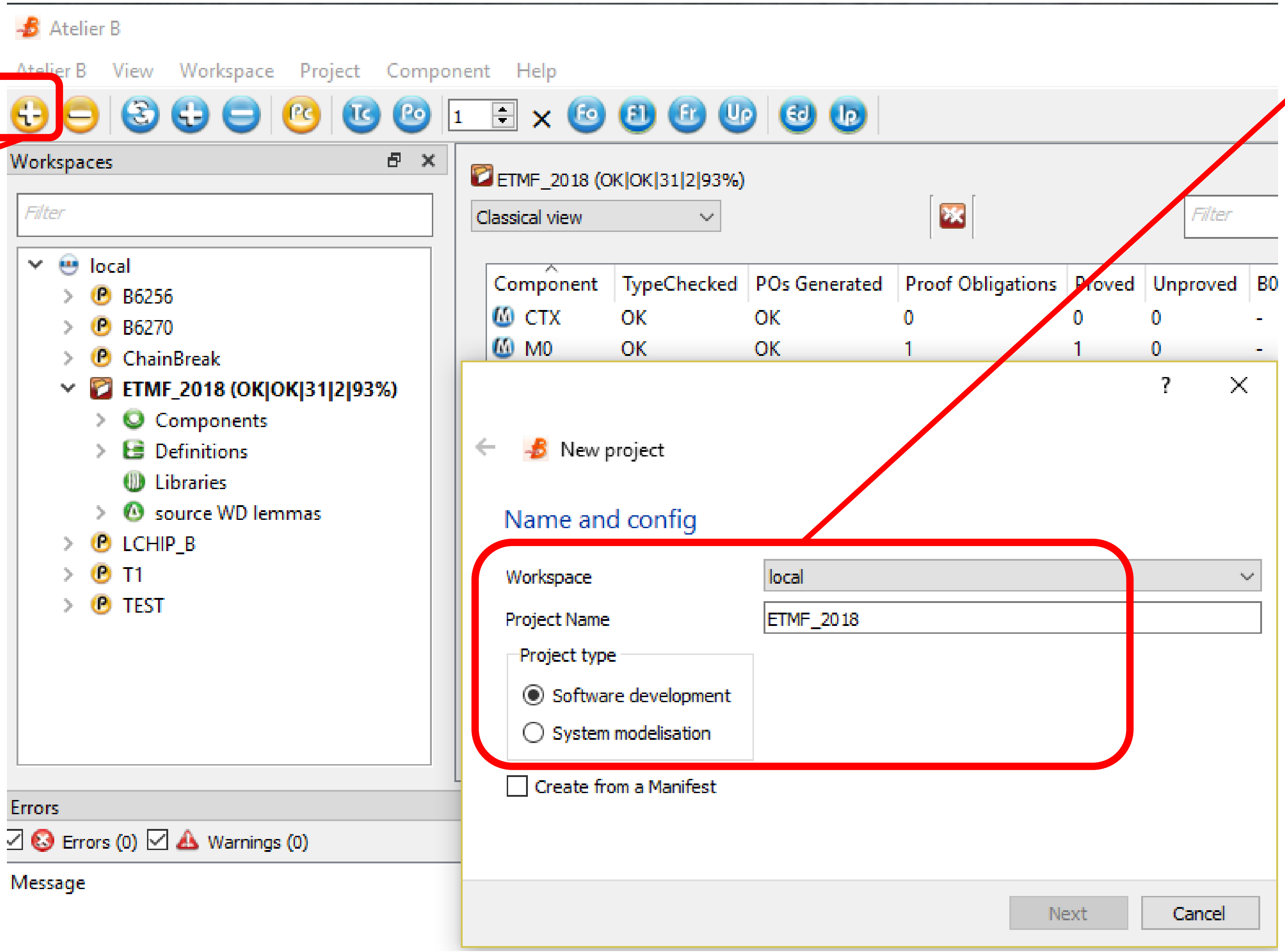
to get editor colored with proof status



Proof System

≡ Create a B Project

Click on the Yellow "+"

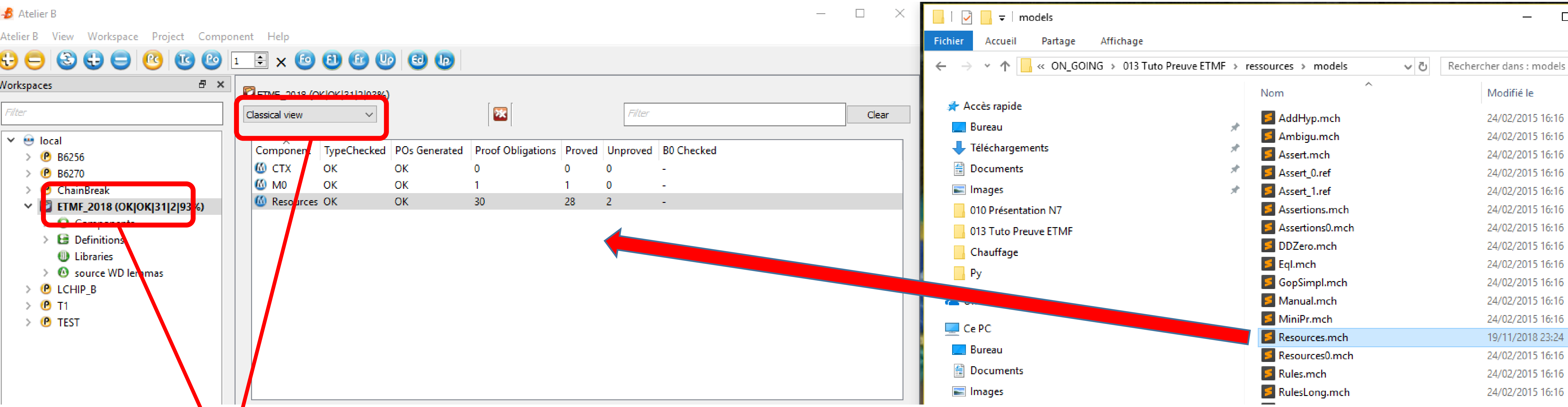


- **Enter a name Ex: "ETMF_2018"**
- **Select "Software Development"**
- **Click on "Next"**
- **On first execution, you are asked to define the project directory**
  Select any directory with R/W access
- **Click on "Finish"**

CLEARSY
SYSTEMS ENGINEERING

Proof System

≡ Add a Component and Prove it



- **Switch to "Classical view"**
  The only mode supporting "drag-n-drop"
- **Open the project by double-clicking**

- **Open an Explorer**
- **Go to "Models" directory**
- **Drag-n-Drop "Resources.mch" to the component Tab**
  New component Resources added in the list

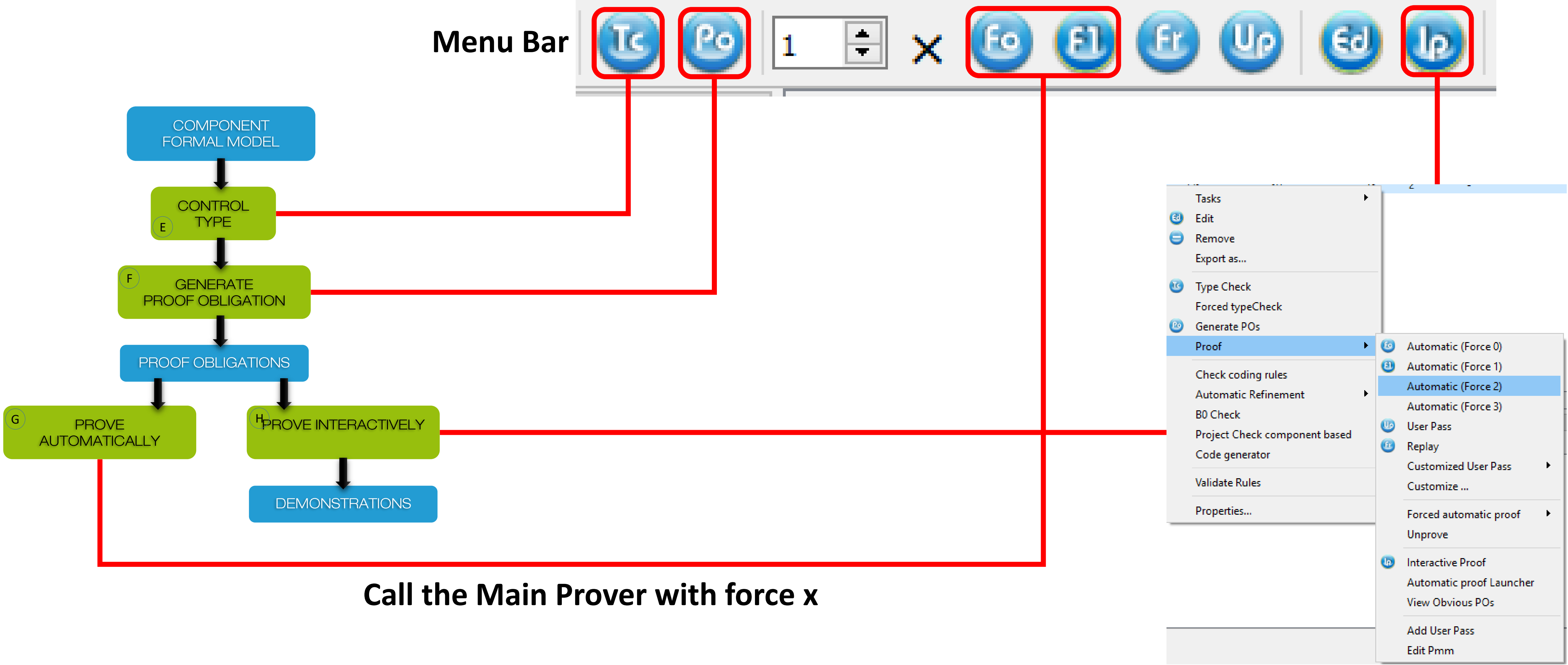
CLEARSY
SYSTEMS ENGINEERING

Proof System

To complete Proof:

- **Type in sequence: TC, PO, Fx**
- **Or type in Fx (all missing steps performed automatically)**

≡ The Proof Process



Call the Main Prover with force x

Proof System

≡ Proving “Resources”

- Select the component “resources”
- Generate Proof Obligations: 30 generated
  - All unproved

ETMF_2018 (OK|OK|31|30|3%)

Classical view

Filter

Component	TypeChecked	POs Generated	Proof Obligations	Proved	Unproved
CTX	OK	OK	0	0	0
M0	OK	OK	1	1	0
Resources	OK	OK	30	0	30

- Start Proof Force 0
  - 24 proved
- Start Proof Force 1
  - Still 24 proved

Component	TypeChecked	POs Generated	Proof Obligations	Proved	Unproved
CTX	OK	OK	0	0	0
M0	OK	OK	1	1	0
Resources	OK	OK	30	24	6

- Start Proof Force 2
  - 28 proved
- Start Proof Force 3
  - Still 28 proved

Component	TypeChecked	POs Generated	Proof Obligations	Proved	Unproved
CTX	OK	OK	0	0	0
M0	OK	OK	1	1	0
Resources	OK	OK	30	28	2

CLEARSY SYSTEMS ENGINEERING

≡ Proving “Resources”

- **The colouring of the model == proof status in force 0**
  - **Get a quick feedback about the model**
  - **To prove the model, go interactive**

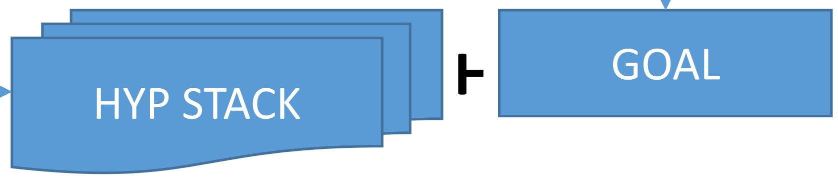
Resources.mch

```
MACHINE
    Resources(nn)

CONSTRAINTS
    nn: NAT1 &
    not(nn = MAXINT)

DEFINITIONS
    RESOURCES == 0..nn

VARIABLES
    available,in_use,faulty

INVARIANT
    available <: RESOURCES &
    in_use <: RESOURCES &
    faulty <: RESOURCES &
    available\/in_use\/faulty = RESOURCES &
    available/\in_use = {} &
    available/\faulty = {} &
    in_use/\faulty = {}

INITIALISATION
    available:=RESOURCES ||
    in_use:={} ||
    faulty:={}

OPERATIONS

    bb <-- AnyAvailable = BEGIN
        bb:=bool(not(available = {}))
    END;

    xx <-- AcquireResource = PRE
        not(available = {})
    THEN
        ANY rr WHERE
```

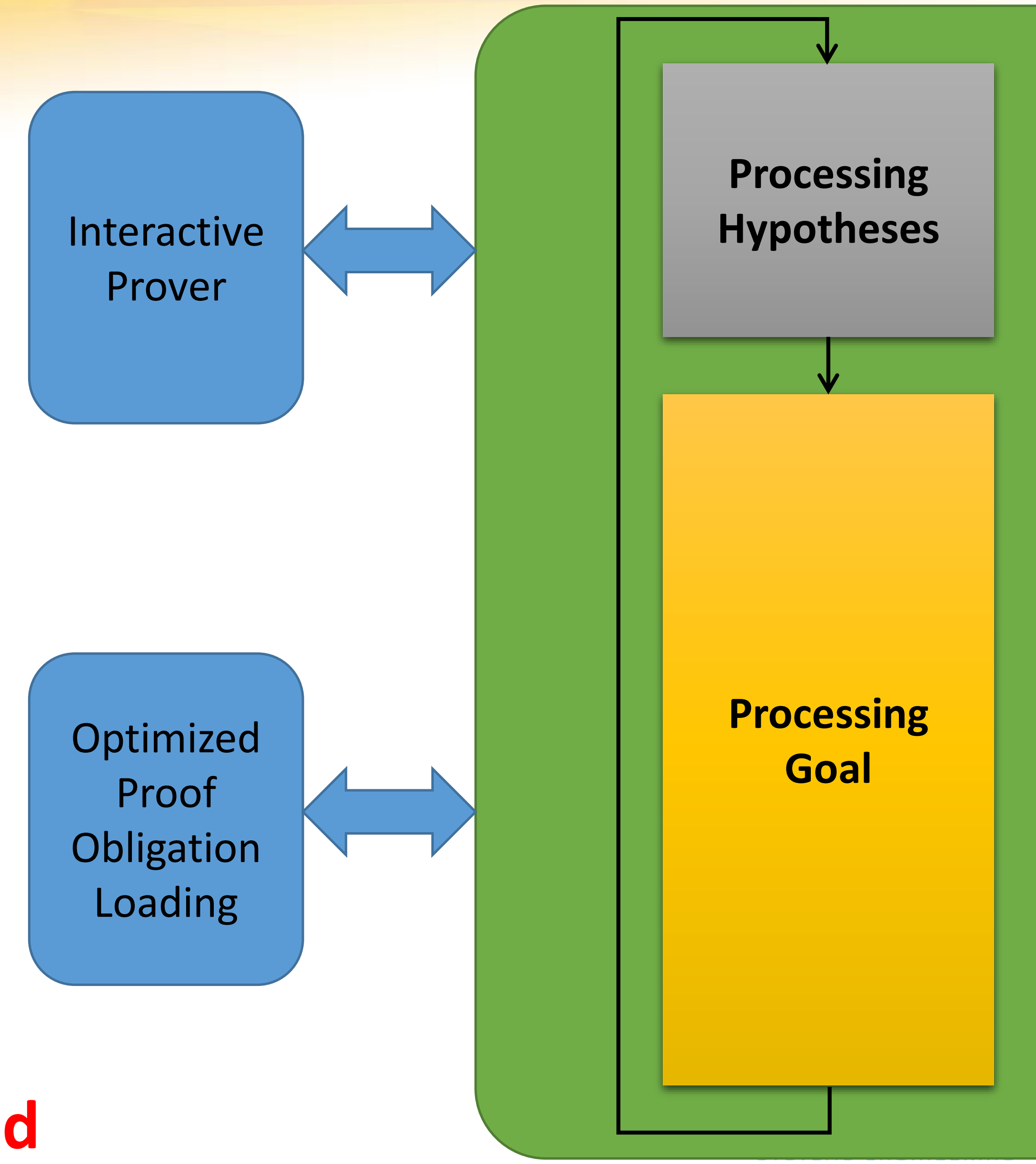
Proof System

≡ Main Prover

- Created in the early 90's by Alstom signalling engineer
- **2 main principle**s:
  - **Generate new hypotheses** (bottom-up)
    - Linked with goal
    - Linked with hypotheses in relation with the goal
  - **Simplify goal predicate** (top-down)
    - Simplification mechanisms
    - Mathematical rules, both **triggered by hypotheses**

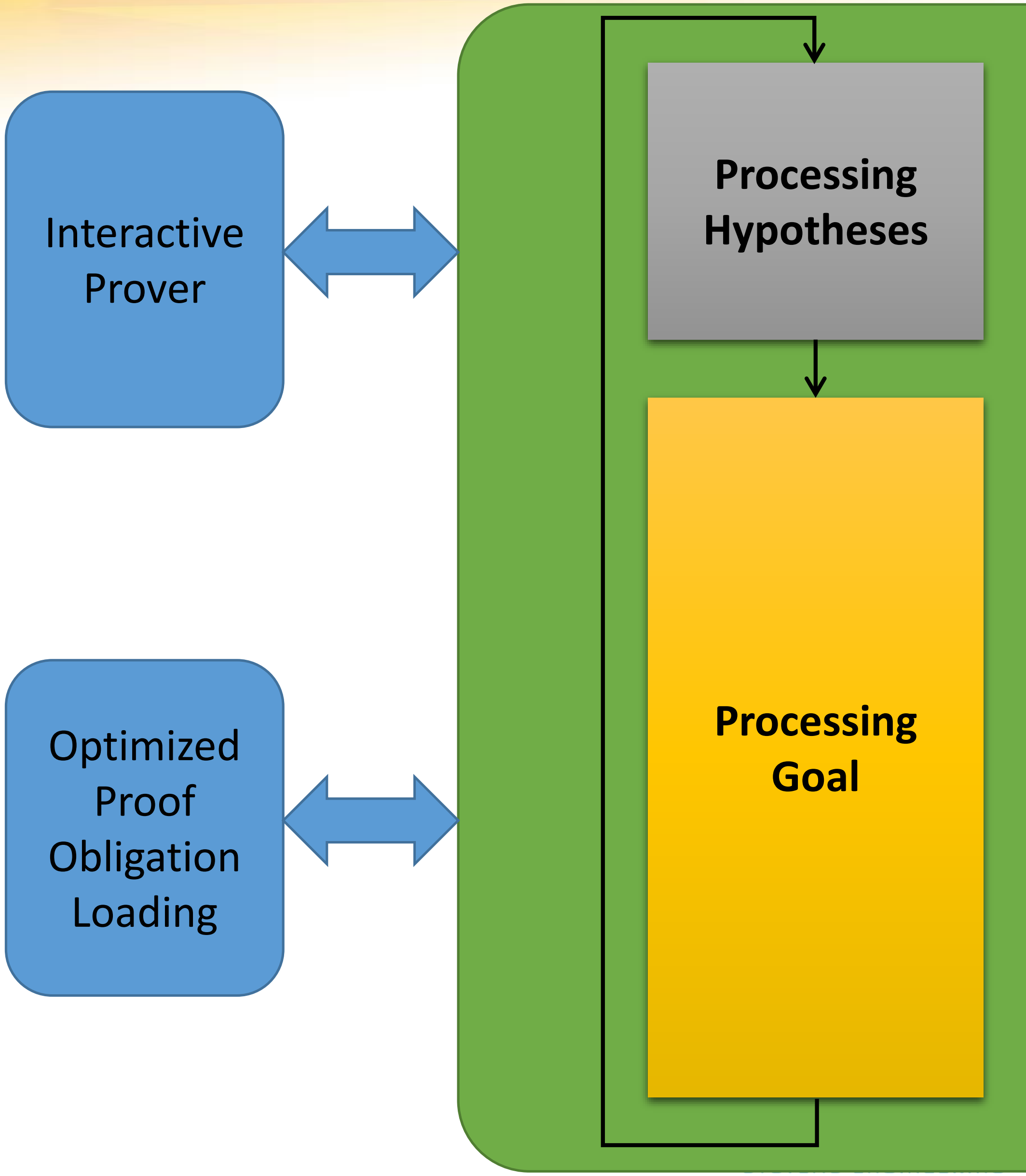


Once an HYP is in the stack, it can't be modified

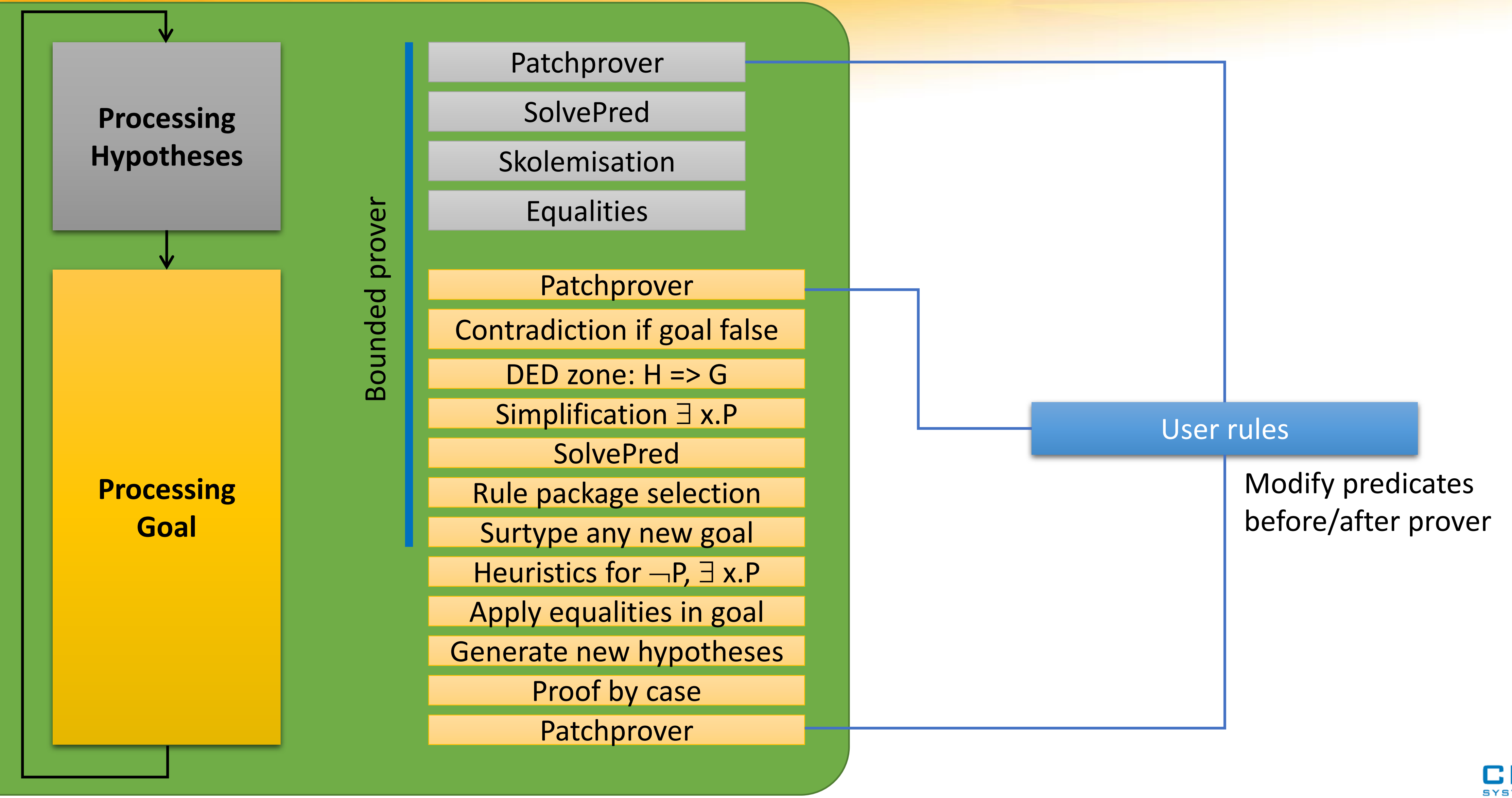


Proof System

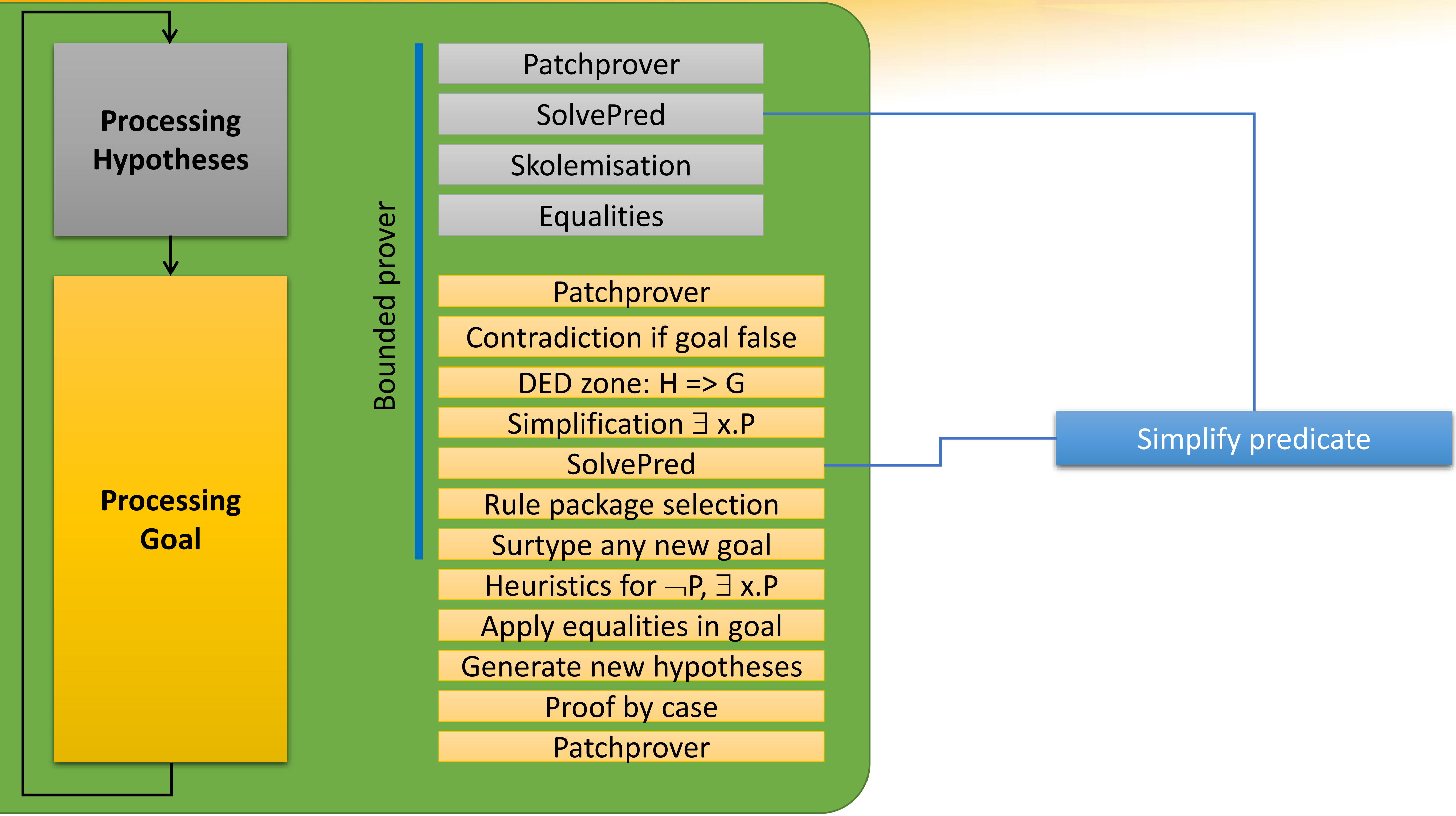
- Main Prover



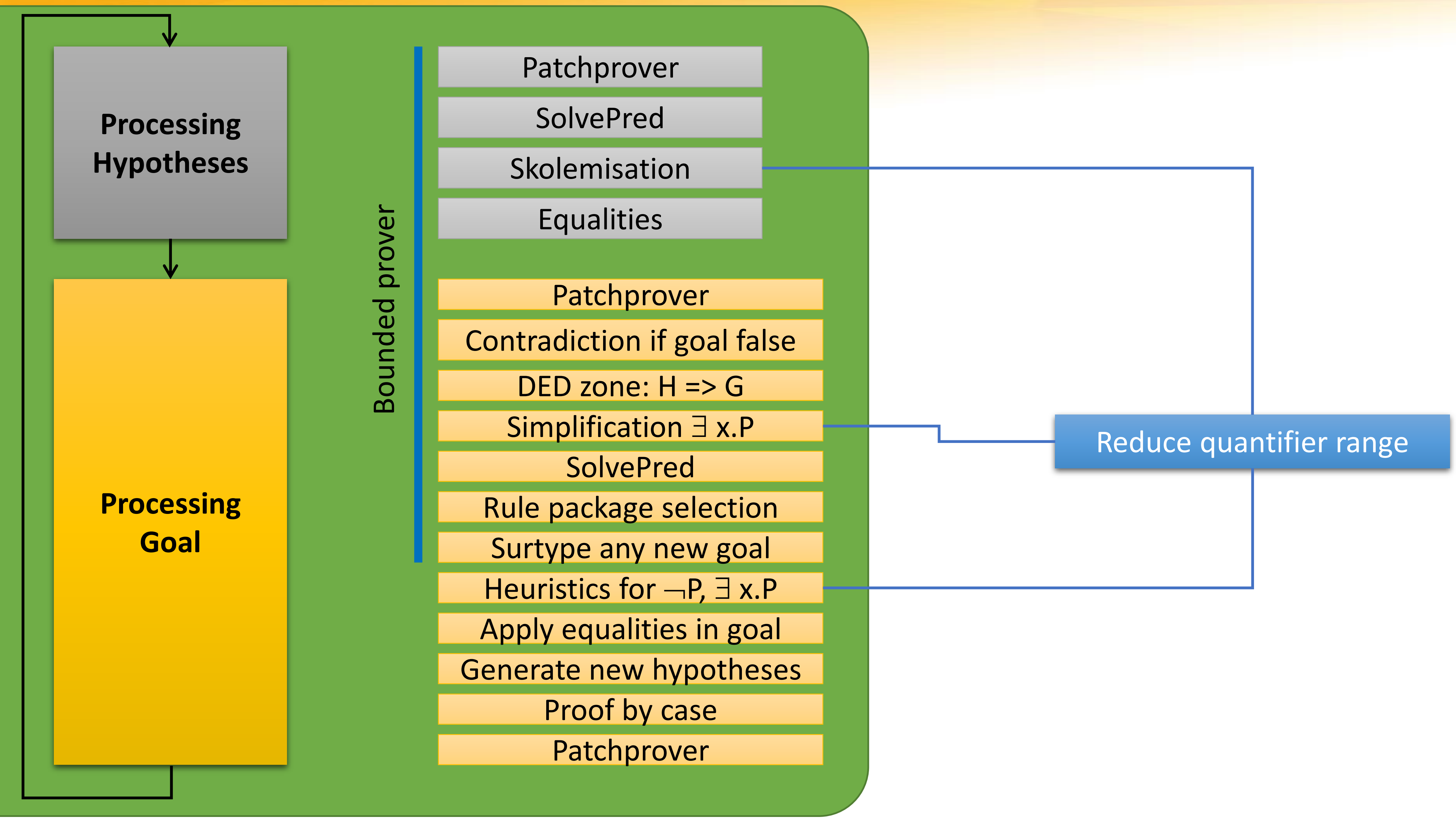
Proof System



Proof System

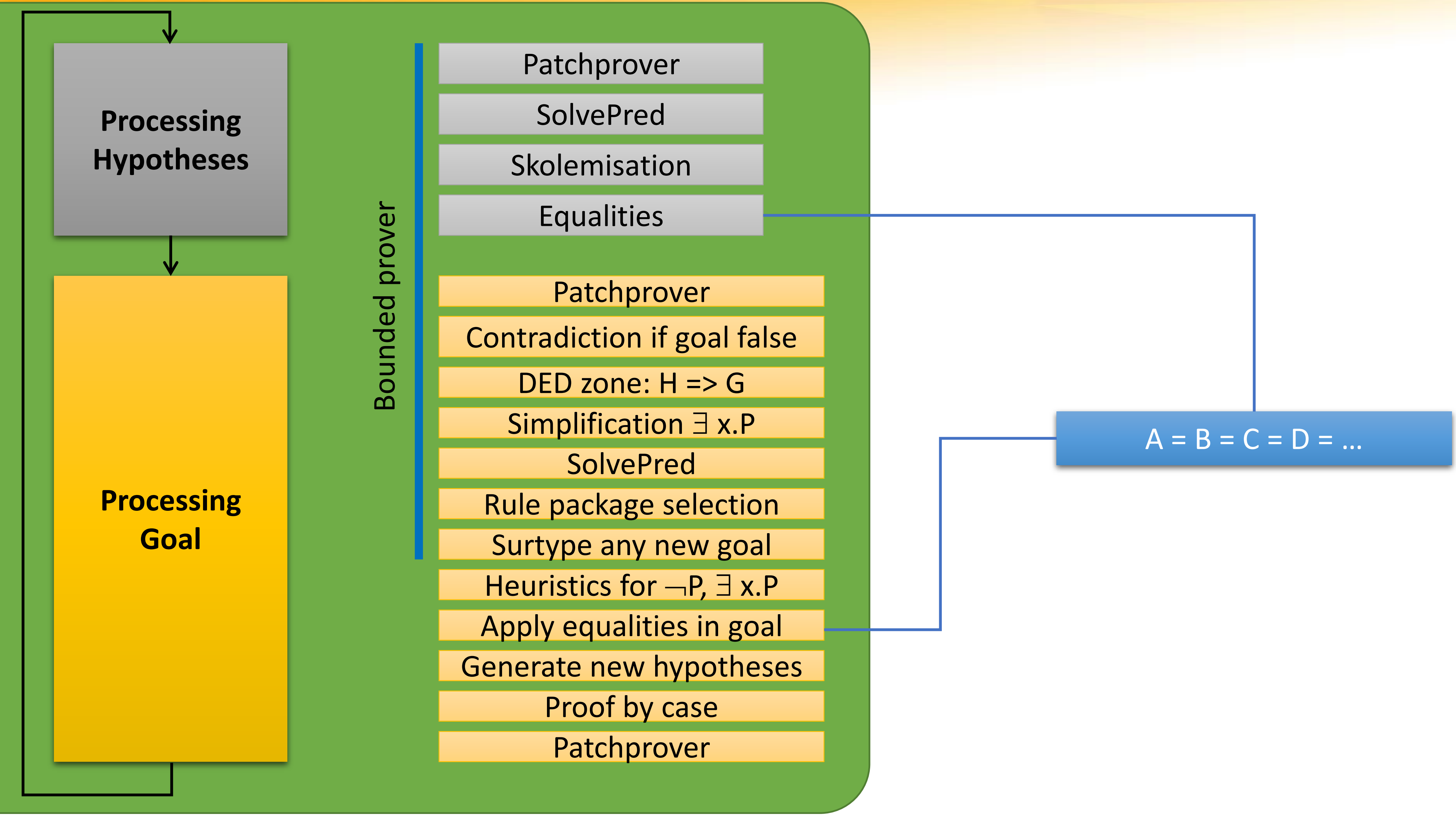


Proof System

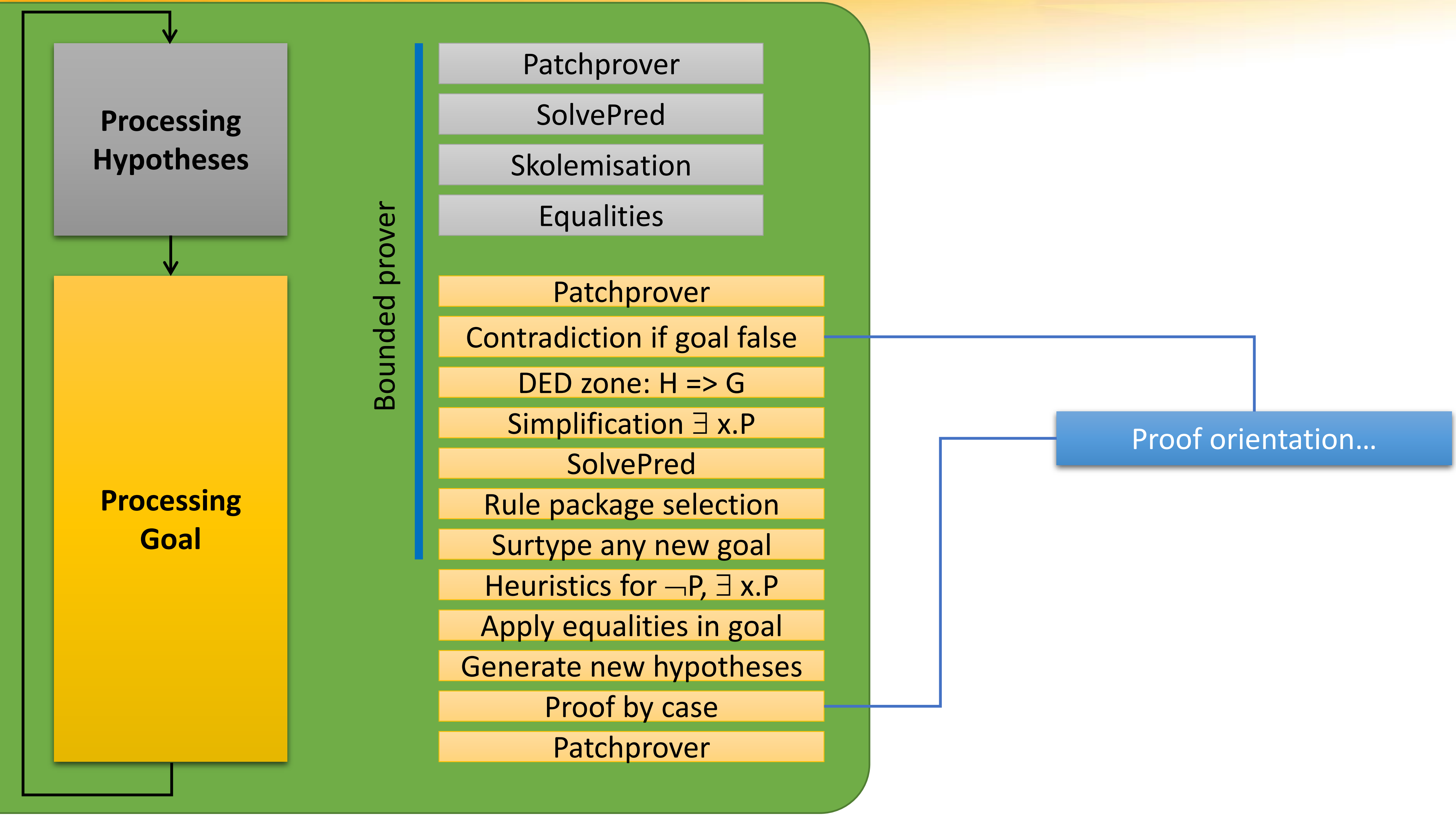


CLEARSY SYSTEMS ENGINEERING

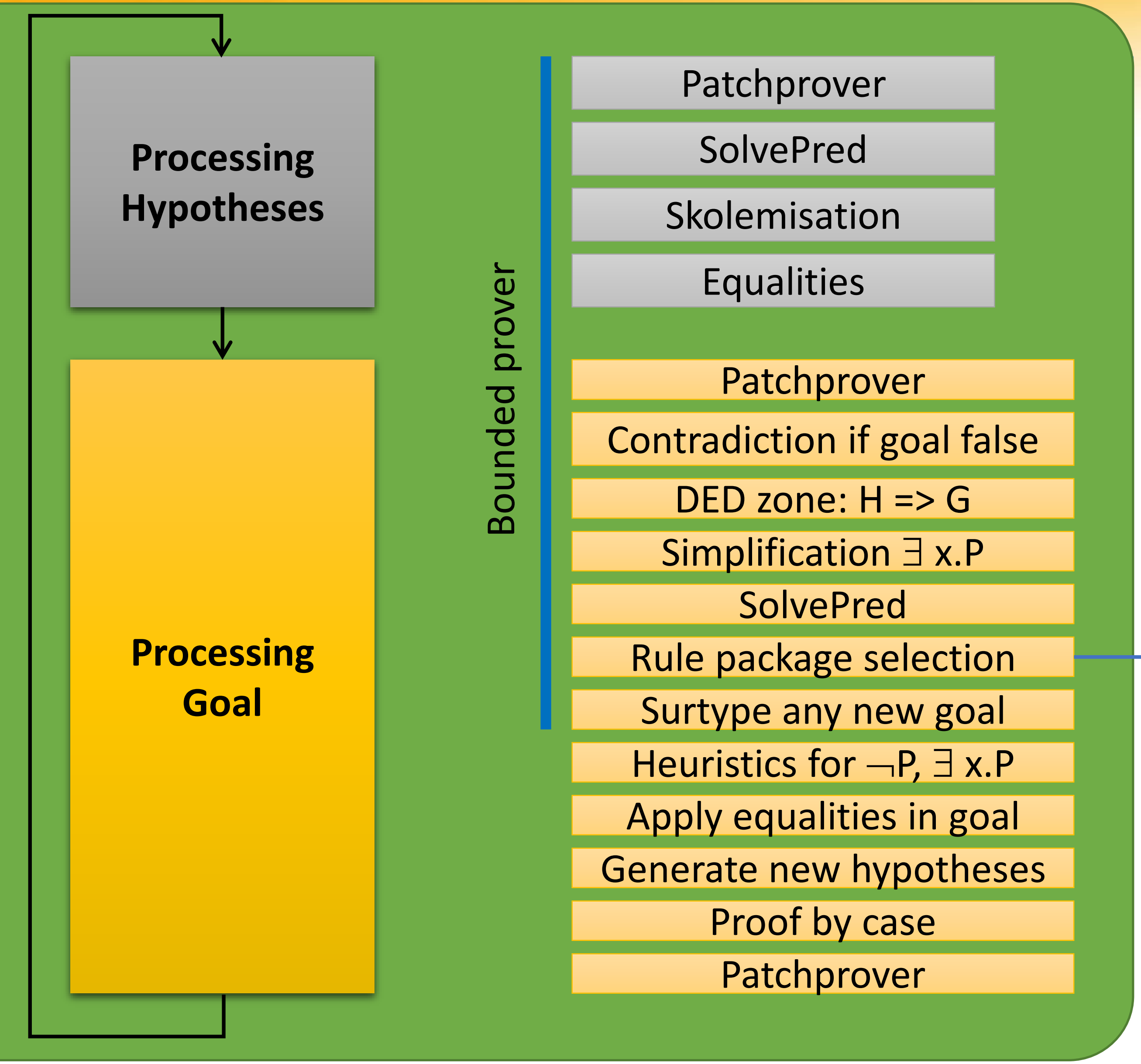
Proof System



Proof System



Proof System



f-a is a partial function from **s** to **t**

f is a partial function from **s** to **t**

a is a relation from **u** to **v**

Single letter identifiers are wildcards and may match with any valid expression

Proof with rules

Predicates are broken down into smaller/simpler predicates

Provided for information only as mechanisms are not directly activable

CLEARSY SYSTEMS ENGINEERING

Proof System

Processing Hypotheses

Processing Goal

Bounded prover

- Patchprover
- SolvePred
- Skolemisation
- Equalities

- Patchprover
- Contradiction if goal false
- DED zone: H => G
- Simplification ∃ x.P
- SolvePred
- Rule package selection
- Surtype any new goal
- Heuristics for ¬P, ∃ x.P
- Apply equalities in goal
- Generate new hypotheses
- Proof by case
- Patchprover

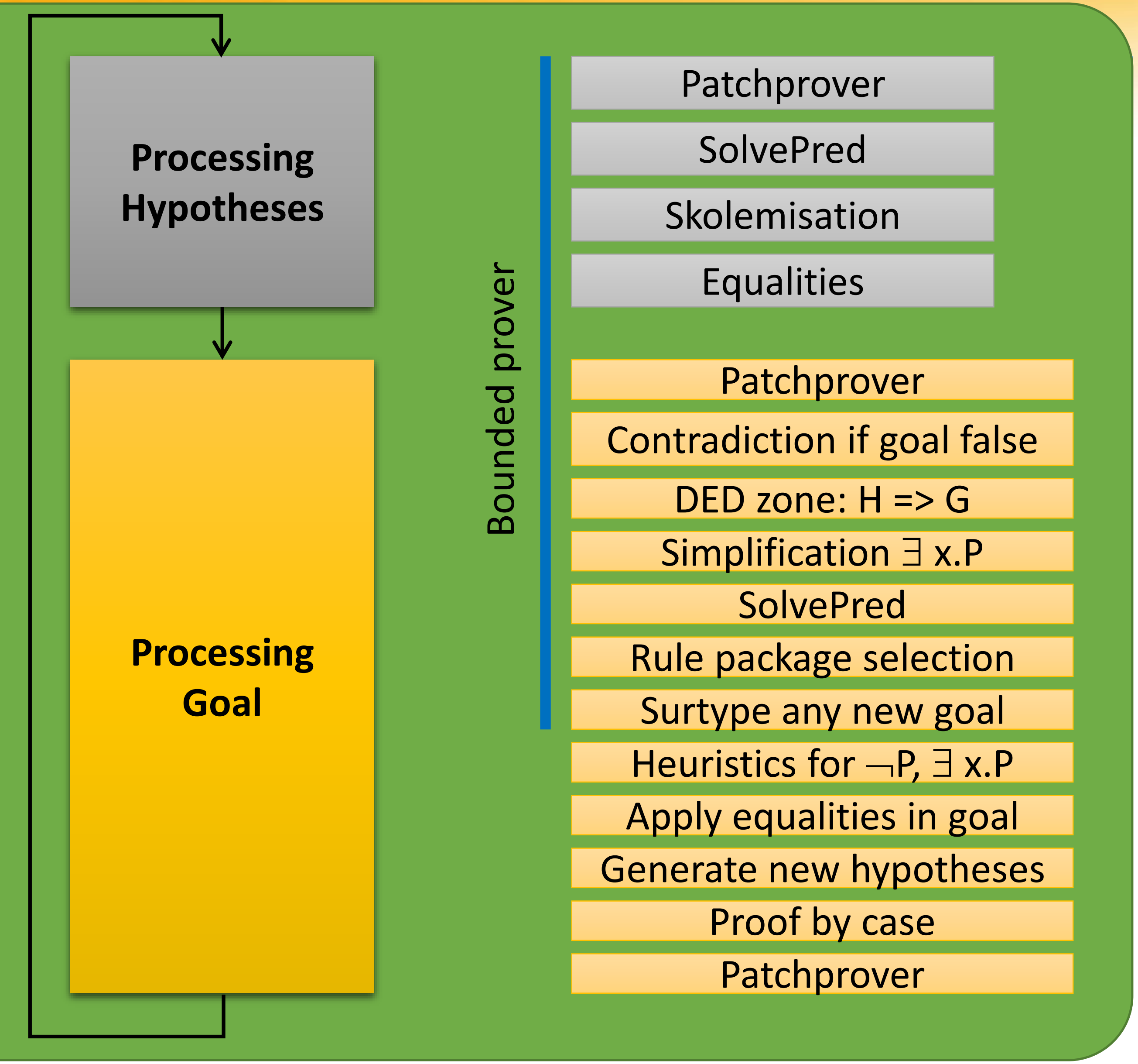
dd(x)

Deduction

If the goal is H => G, H is transformed and then added to the HYP STACK
The goal becomes G

dd(0) performs deduction in force 0
dd(3) performs deduction in force 3
dd(3) generates more new HYP than dd(0)
HYP may also be rewritten differently

Proof System



dd

Deduction (raw)

If the goal is H => G, H is added to the HYP STACK without modification
The goal becomes G

Sometimes the prover performs %!&!? transformations that are not suitable
Apply dd if you really need H in hypotheses

Proof System

Processing Hypotheses

Processing Goal

Bounded prover

Patchprover
SolvePred
Skolemisation
Equalities

Patchprover
Contradiction if goal false
DED zone: H => G
Simplification ∃ x.P
SolvePred
Rule package selection
Surtype any new goal
Heuristics for ¬P, ∃ x.P
Apply equalities in goal
Generate new hypotheses
Proof by case
Patchprover

mp

Mini Proof

Starts the bounded prover (no divergent behaviour)
Performs deduction of the current force
Triggers the mechanisms in sequence

Add new HYPS on the STACK

Succeed to produce a new goal G’ ≠ G
or
fail if no new HYP added and goal remains G

Proof System

Processing Hypotheses

Processing Goal

Bounded prover

- Patchprover
- SolvePred
- Skolemisation
- Equalities

- Patchprover
- Contradiction if goal false
- DED zone: H => G
- Simplification ∃ x.P
- SolvePred
- Rule package selection
- Surtype any new goal
- Heuristics for ¬P, ∃ x.P
- Apply equalities in goal
- Generate new hypotheses
- Proof by case
- Patchprover

pr

Proof

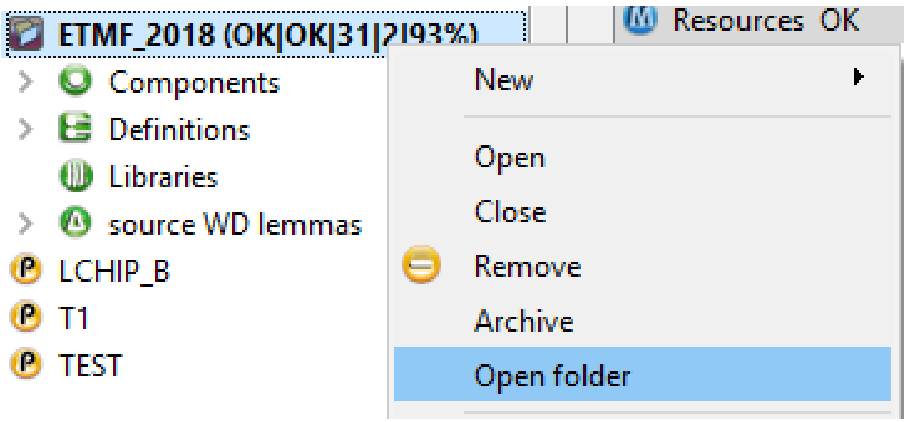
Starts the full prover
Performs deduction of the current force
Triggers the mechanisms in sequence

Add new HYPS on the STACK

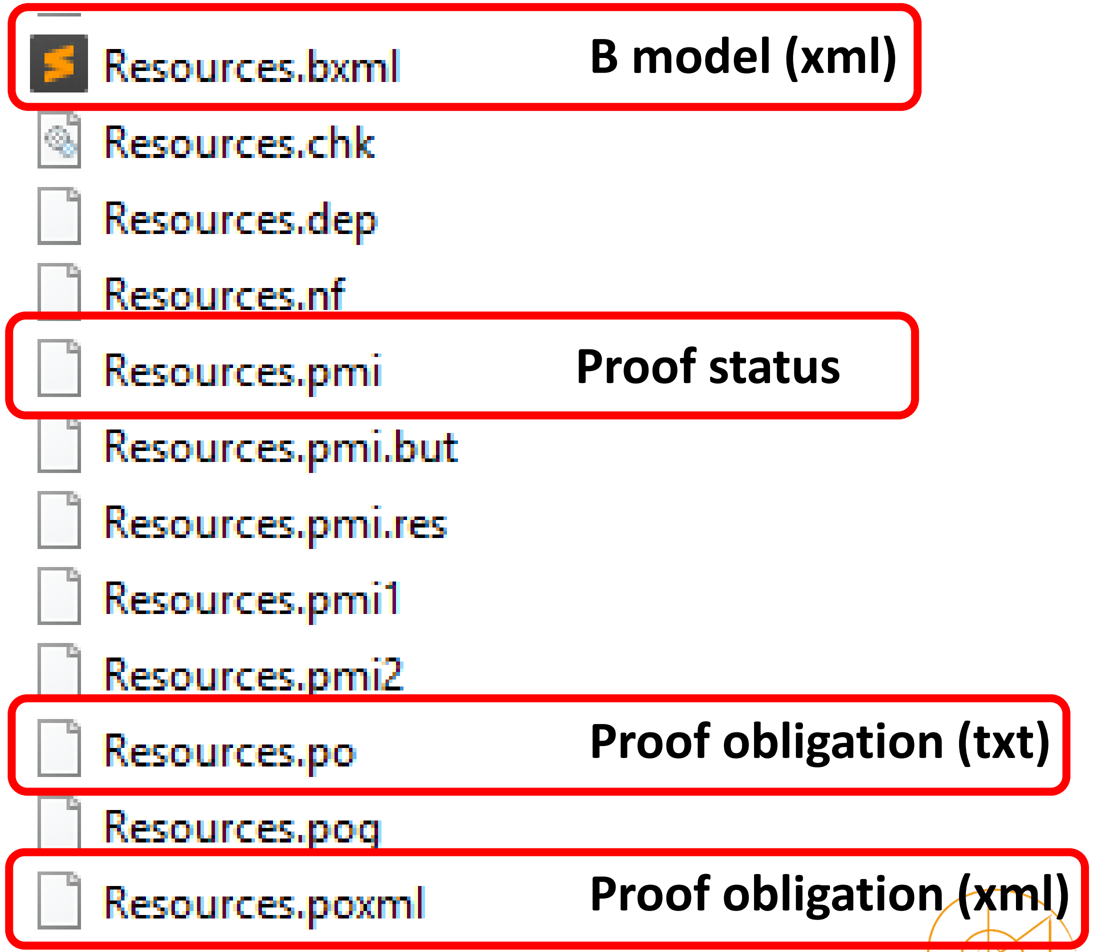
Succeed to produce a new goal G’ ≠ G
or
fail if no new HYP added and goal remains G

Proof System

≡ Behind the Curtain



- Select the Project
- Right click and select “Open Folder”
- An Explorer shows up
- Open “bdp” directory
- Several files “Resources” with different extensions



CLEARSY
SYSTEMS ENGINEERING

Proof System

Behind the Curtain (Resources.po)

- Hypotheses as packages named _f(1), _f(2), etc.
- PO definition in a line (first PO goal is _f(45))
- Moving from first PO to second PO only requires to pop _f(12) and to push _f(22)

```
THEORY ProofList IS
_f(1) & _f(2) & _f(14) & _f(25) & RestoreResource.6,(_f(39) & _f(3) & _f(12) => _f(45));
_f(1) & _f(2) & _f(14) & _f(25) & RestoreResource.5,(_f(39) & _f(3) & _f(22) => _f(44));
_f(1) & _f(2) & _f(14) & _f(25) & RestoreResource.4,(_f(39) & _f(3) & _f(10) => _f(43));
_f(1) & _f(2) & _f(14) & _f(25) & RestoreResource.3,(_f(39) & _f(3) & _f(8) => _f(42));
_f(1) & _f(2) & _f(14) & _f(25) & RestoreResource.2,(_f(39) & _f(3) & _f(27) => _f(41));
_f(1) & _f(2) & _f(14) & _f(25) & RestoreResource.1,(_f(39) & _f(3) & _f(4) => _f(40));
_f(1) & _f(2) & _f(14) & _f(33) & ReleaseResource.6,(_f(3) & _f(12) => _f(38));
_f(1) & _f(2) & _f(14) & _f(33) & ReleaseResource.5,(_f(3) & _f(22) => _f(37));
_f(1) & _f(2) & _f(14) & _f(33) & ReleaseResource.4,(_f(3) & _f(10) => _f(36));
_f(1) & _f(2) & _f(14) & _f(33) & ReleaseResource.3,(_f(3) & _f(8) => _f(35));
_f(1) & _f(2) & _f(14) & _f(33) & ReleaseResource.2,(_f(3) & _f(6) => _f(26));
_f(1) & _f(2) & _f(14) & _f(33) & ReleaseResource.1,(_f(3) & _f(4) => _f(34));
_f(1) & _f(2) & _f(14) & _f(25) & FaultyResource.7,(_f(3) & _f(12) => _f(32));
_f(1) & _f(2) & _f(14) & _f(25) & FaultyResource.6,(_f(3) & _f(22) => _f(31));
```

Proof System

Behind the Curtain (Resources.pmi)

```
THEORY ProofState IS
    Proved(0);
    Proved(0);
    Proved(0);
    Proved(0);
    Proved(0);
    Proved(0);
    Proved(0);
    Proved(2);
    Proved(0);
    Proved(0);
    Proved(0);
    Proved(0);
    Proved(2);
    Proved(2);
    Proved(0);
    Unproved;
```

- **PO status**
  - **Proved(0) : proved in force 0**
  - **Proved(2): proved in force 2**
  - **Unproved**

Proof System

Behind the Curtain (Resources.pmi)

```
THEORY MethodList IS
    pr;
    pr;
    pr;
    pr;
    pr;
    pr;
    pr;
    pr;
    pr;
    pr;
    pr;
    pr;
    pr;
    pr;
    pr;
    ?;
```

- Saved demonstrations per PO (same order)
  - pr : full prover
  - ?: nothing saved (default when file created)

- When the model is modified and the PO order changes, the merger tries to find a “correct” allocation to avoid to lose demos

Proof System

Behind the Curtain (Resources.pmi)

```
THEORY PassList IS
    Force(0),?;
    Force(0),?;
    Force(0),?;
    Force(0),?;
    Force(0),?;
    Force(0),?;
    Force(0),?;
    Force(2),(?;0;1);
    Force(0),?;
    Force(0),?;
    Force(0),?;
    Force(0),?;
    Force(2),(?;0;1);
    Force(2),(?;0;1);
    Force(0),?;
    Force(0),(?;0;1;2;3);
```

List of forces tried

- **Avoid to start again the main prover if the model has not been modified and the forces already tried without success**

Proof System

Component	TypeChecked	POs Generated	Proof Obligations	Proved	Unproved
CTX	OK	OK	0	0	0
M0	OK	OK	1	1	0
Resources	OK	OK	30	28	2

We still have 2 Unproved PO

≡ Interactive Proof

POs on line 9 of Resources

- Initialisation.1
- Initialisation.2
- Initialisation.3
- Initialisation.4
- AcquireResource.1
- AcquireResource.2
- AcquireResource.3
- FaultyResource.1
- FaultyResource.2
- FaultyResource.3
- FaultyResource.4
- ReleaseResource.1
- ReleaseResource.2
- ReleaseResource.3
- RestoreResource.1
- RestoreResource.2
- RestoreResource.3

Selected PO : Resources.AcquireResource.3

```
rr : available &
btrue
=>
available - {rr} \/
(in_use \/ {rr}) \/
faulty = 0 .. nn
```

Resources.mch

```
1         MACHINE
2             Resources(nn)
3
4         CONSTRAINTS
5             nn: NAT1 &
6             not(nn = MAXINT)
7
8         DEFINITIONS
9   15/17     RESOURCES == 0..nn
10
11        VARIABLES
12            available,in_use,faulty
13
14        INVARIANT
15    5/5     available <: RESOURCES &
16    4/4     in_use <: RESOURCES &
17    2/2     faulty <: RESOURCES &
18    3/5     available\/in_use\/faulty = RESOURCES &
19    5/5     available/\in_use = {} &
20    4/4     available/\faulty = {} &
21    5/5     in_use/\faulty = {}
22
23        INITIALISATION
24    3/3     available:=RESOURCES ||
25    4/4     in_use:={} ||
26    2/2     faulty:={}
27
28        OPERATIONS
```

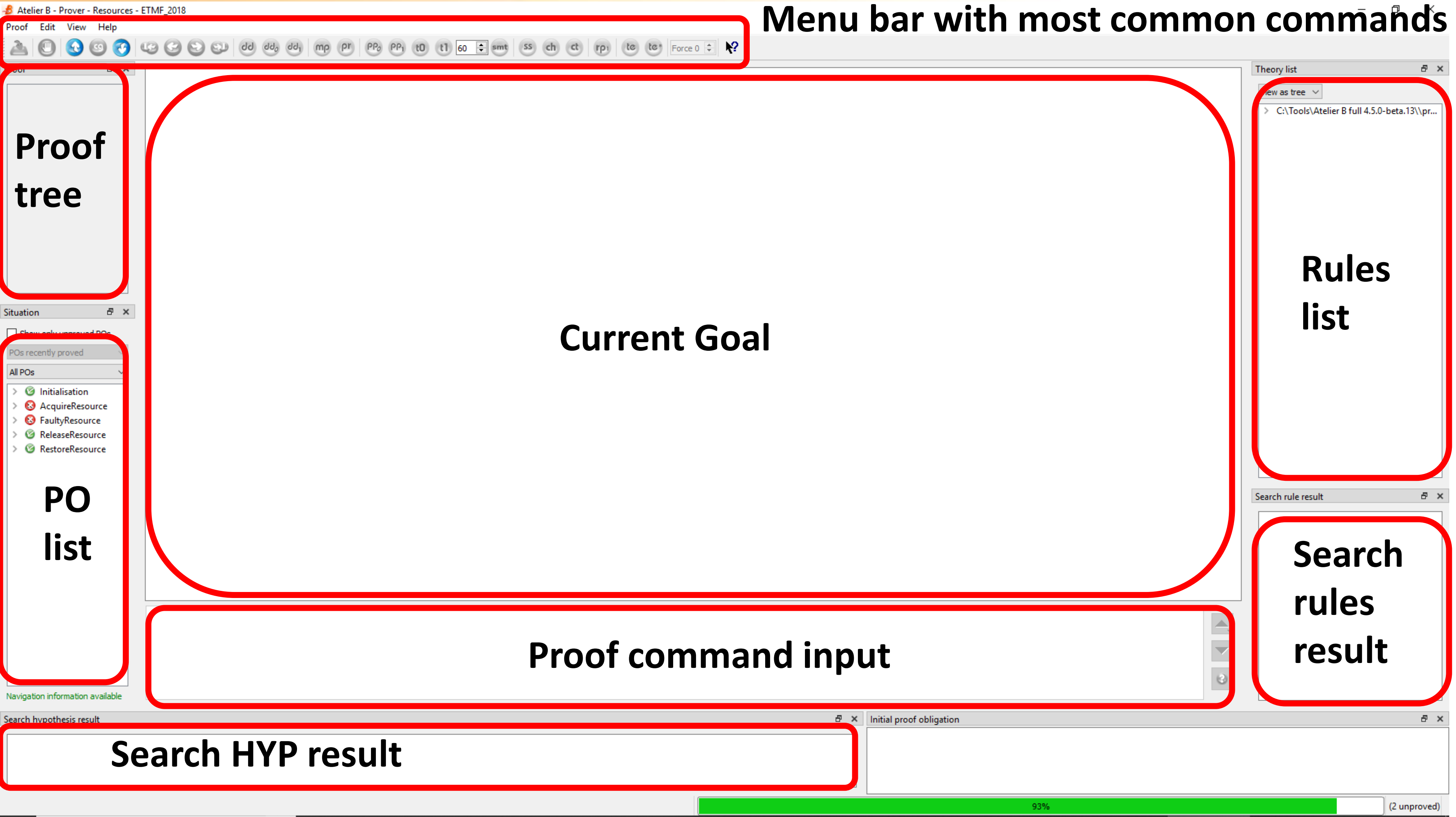
With the editor, we quickly check the 2 POs but nothing obvious

Time to start the Interactive Prover

CLEARSY
SYSTEMS ENGINEERING

Proof System

≡ Interactive Proof UI

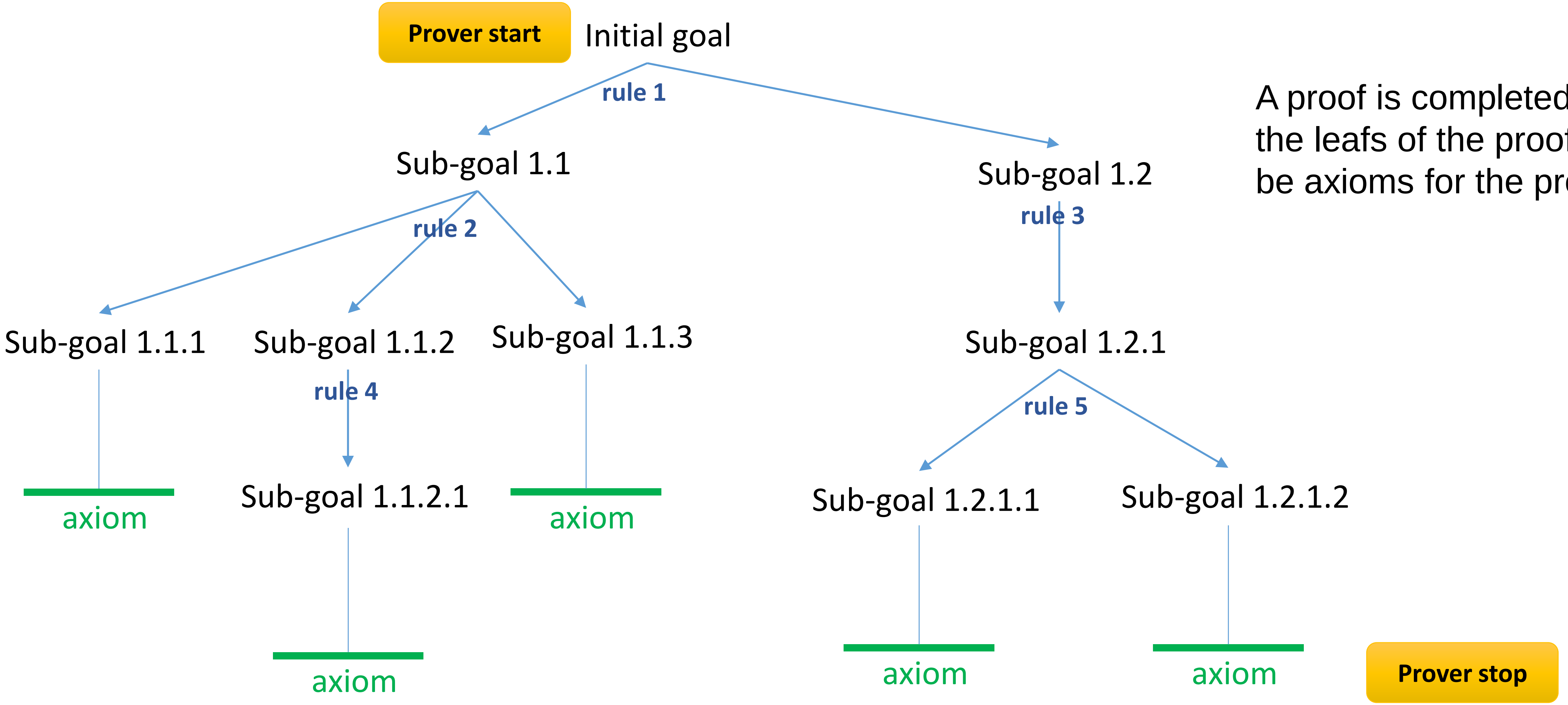


CLEARSY SYSTEMS ENGINEERING

Proof System

By applying iteratively decomposition rules, the theorem prover creates a proof tree

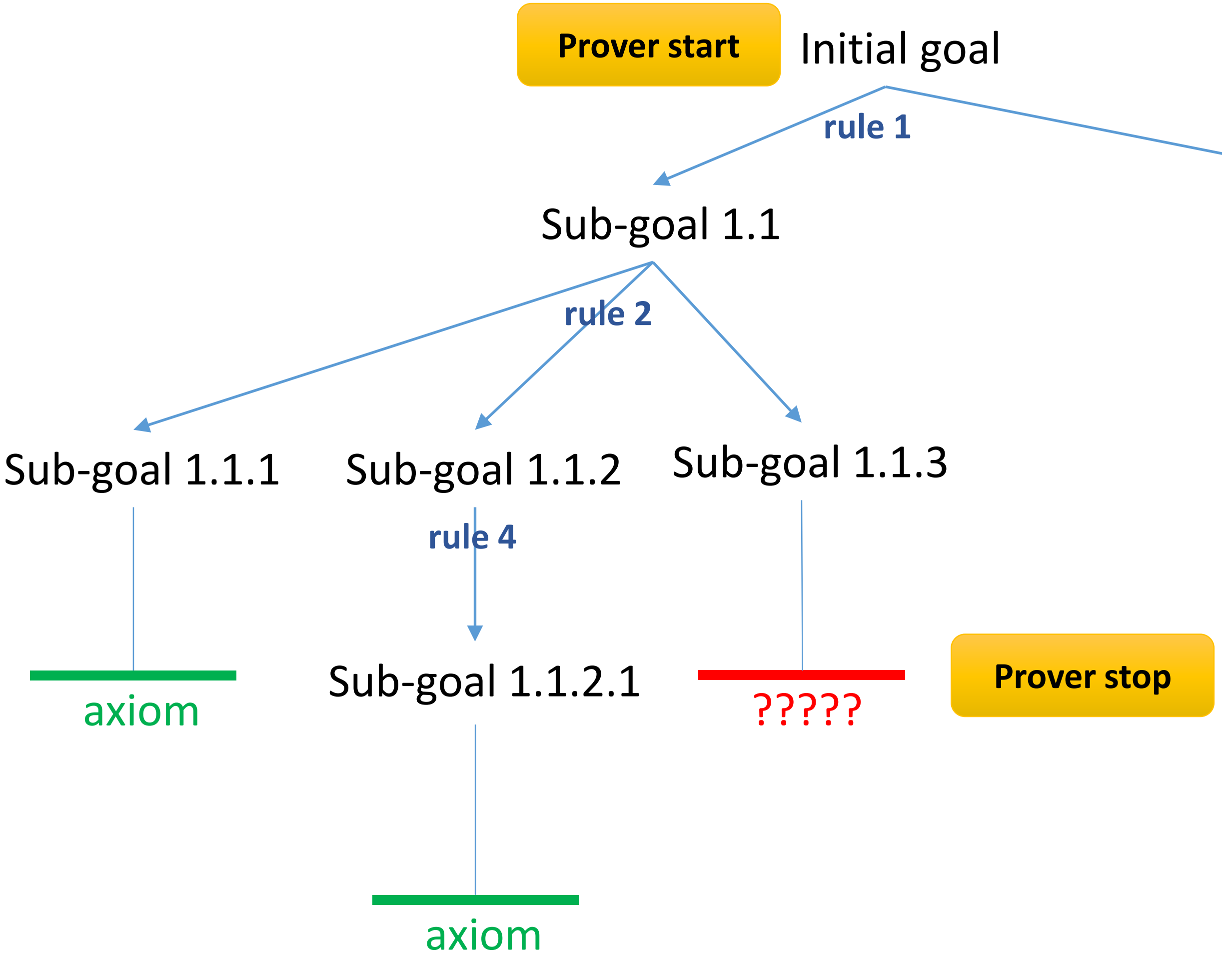
Proof Tree & demonstration



A proof is completed when all the leafs of the proof tree turn to be axioms for the prover

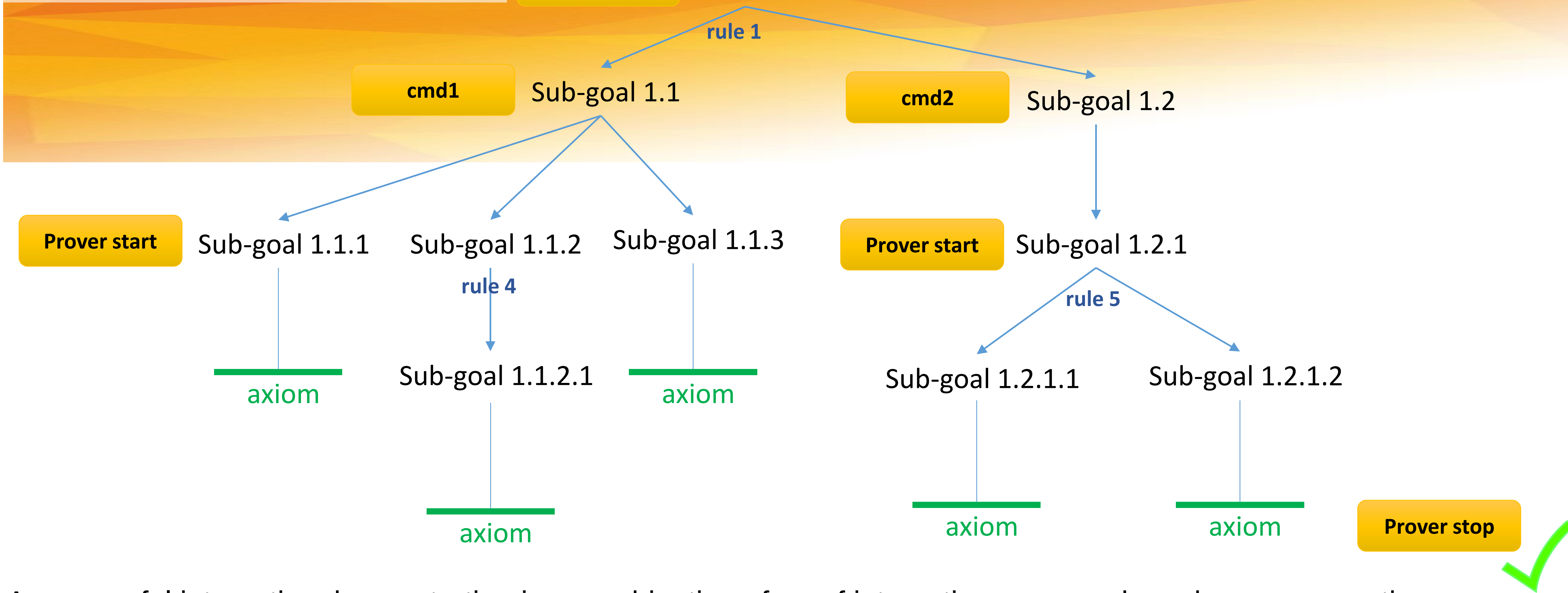
Proof System

Proof Tree & demonstration



An unproved proof obligation is represented by a proof tree where at least one leaf is not an axiom

Proof System



A successful interactive demonstration is a combination of proof interactive commands and prover execution

Prover start | **cmd1** | **Prover start** | **cmd2** | **Prover start**

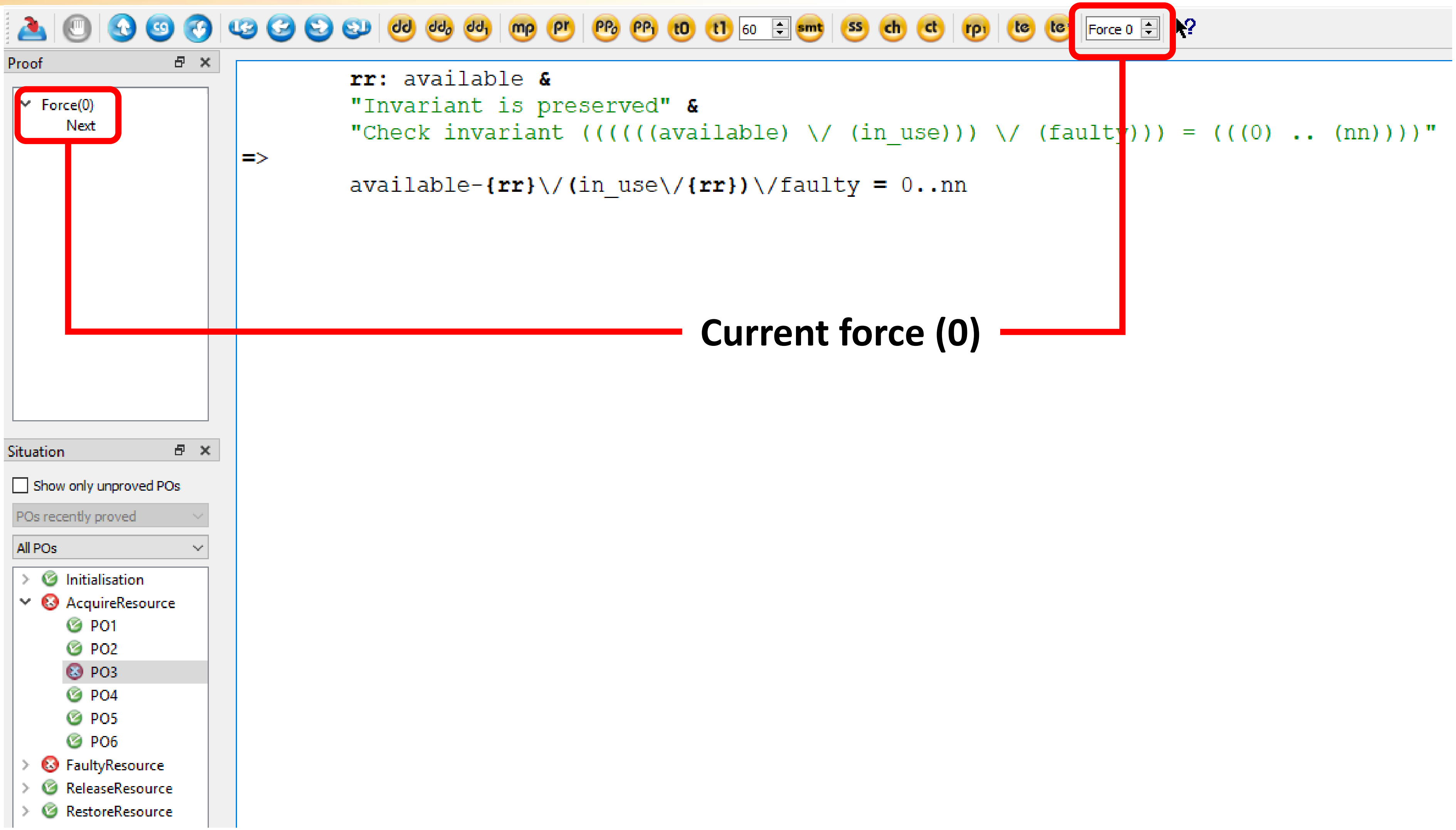
This demonstration (also called proof script) is saved and can be replayed at will to ensure that the proof obligation is true

CLEARSY
SYSTEMS ENGINEERING

Proof System

Click on the “Next PO” button

Interactive Proof UI

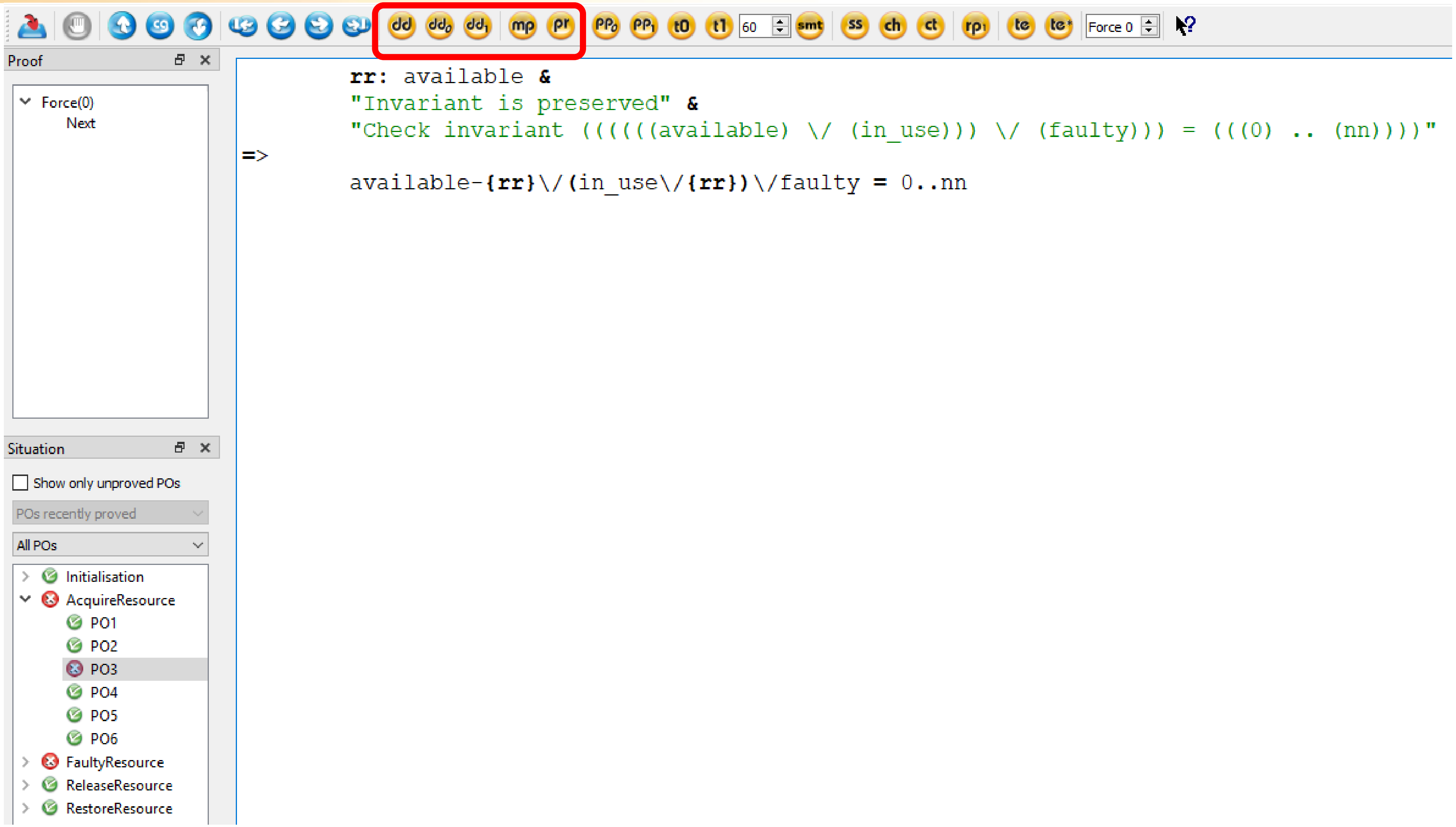


Current force (0)

Proof System

Interactive Proof UI

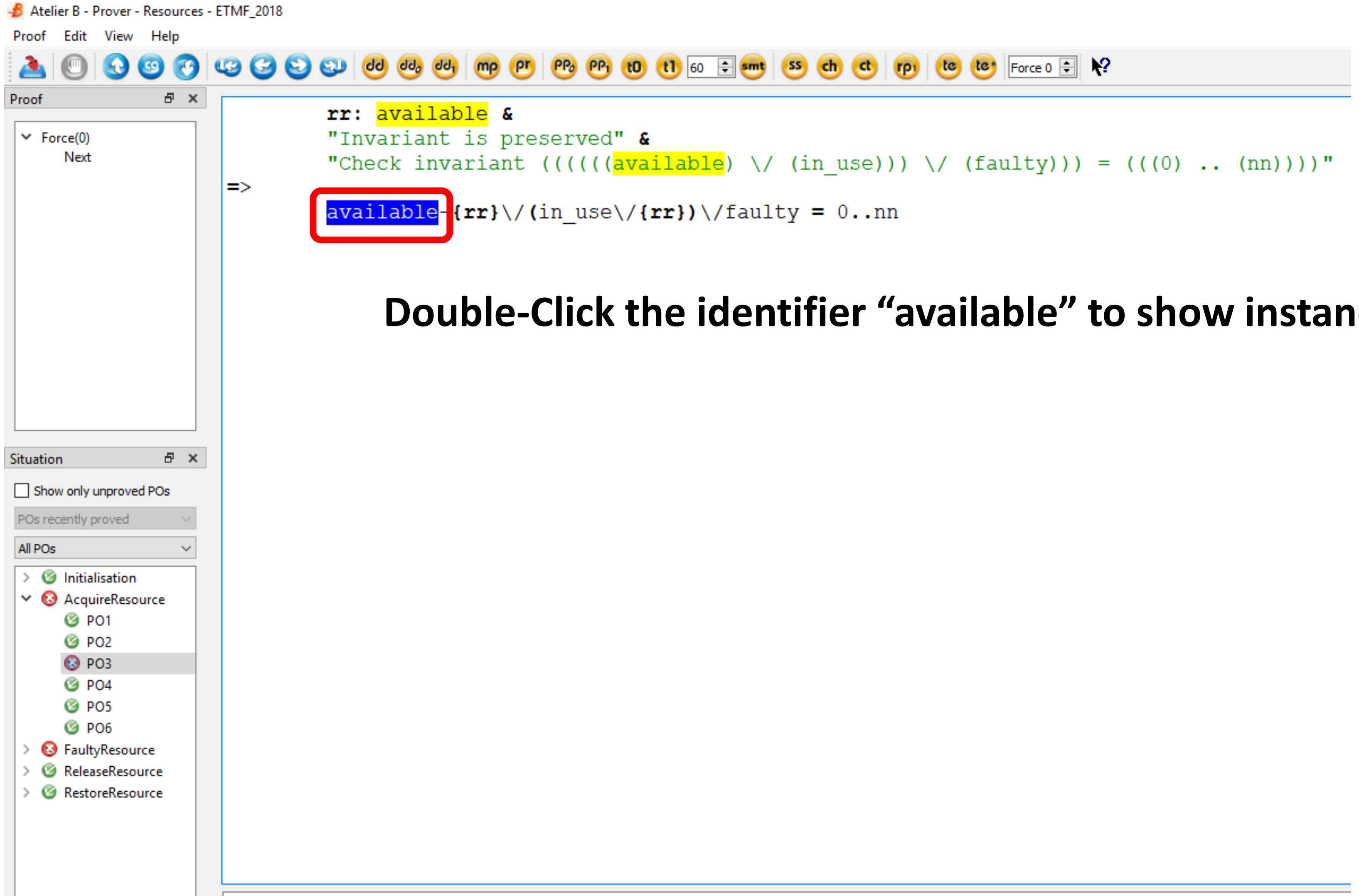
Proof commands



CLEARSY
SYSTEMS ENGINEERING

Proof System

≡ Interactive Proof UI



Double-Click the identifier “available” to show instances on the current goal

CLEARSY SYSTEMS ENGINEERING

Proof System

Interactive Proof UI

```
        rr: available &
        "Invariant is preserved" &
        "Check invariant (((((available)
=>
        avai                            r})\/f
```

- Copy
- Select All
- sh - Search Hypothesis containing
- eh(Goal) - Use Equality in Goal
- eh(Hyp) - Use Equality in Hypothesis
- ae - Add Expression (Expr = pri_varN)
- se - Suggest For Exist
- Clear window
- Open definition

Right-click the identifier "available"

Select "sh - Search Hypothesis containing"

Search hypothesis result

```
Hypothesis containing "available"
        available <: 0..nn
        available\/in_use\/faulty = 0..nn &
        available/\in_use = {} &
        available/\faulty = {} &
        not(available = {}) &
```

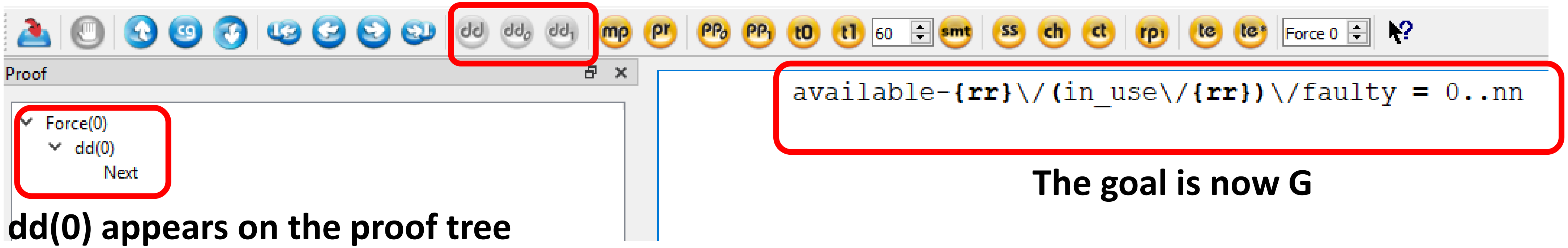
CLEARSY
SYSTEMS ENGINEERING

Proof System

I did dd(0)

≡ Interactive Proof UI

Deduction commands are not available as the goal is not H => G anymore



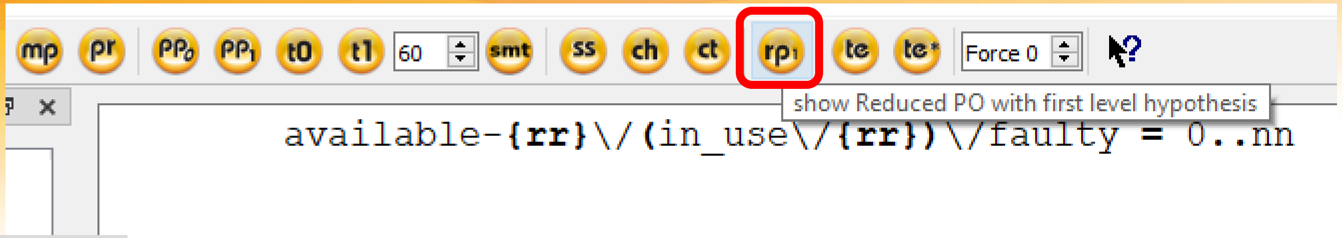
The goal is now G

dd(0) appears on the proof tree
The PO is not proved

The Next indicates the location of the next command in the proof tree

Proof System

Interactive Proof UI



Search hypothesis result

```
----------------------

        rr: available
        not(available = {}) &
        in_use/\faulty = {} &
        available/\faulty = {} &
        available/\in_use = {} &
        available\/in_use\/faulty = 0..nn &
        faulty <: 0..nn &
        in_use <: 0..nn &
        available <: 0..nn &
        nn<=2147483646 &
        1<=nn &
        not(nn = 0) &
        nn<=2147483647 &
        0<=nn &
        nn: INTEGER &
        not(nn: {0}) &
        nn: NAT &
        not(nn = 2147483647) &
        nn: NAT-{0} &
```

Select “rp1” to show all the hypotheses that have a symbol in common with the goal

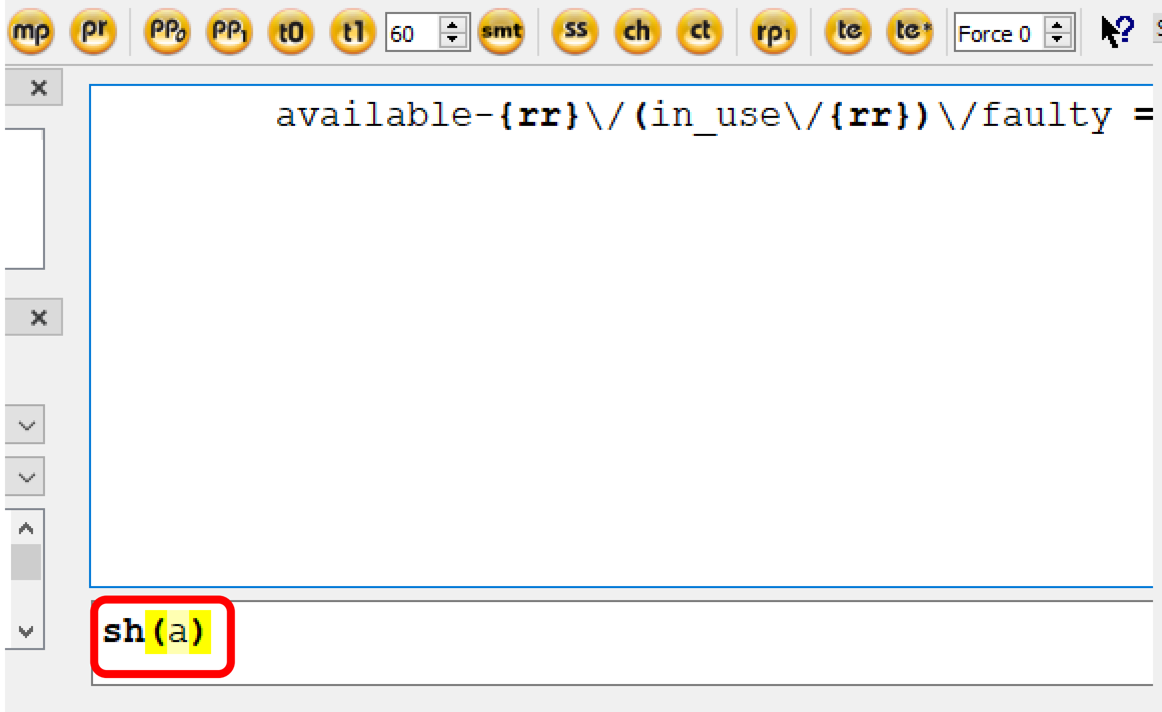
rp

rp(1)

CLEARSY SYSTEMS ENGINEERING

Proof System

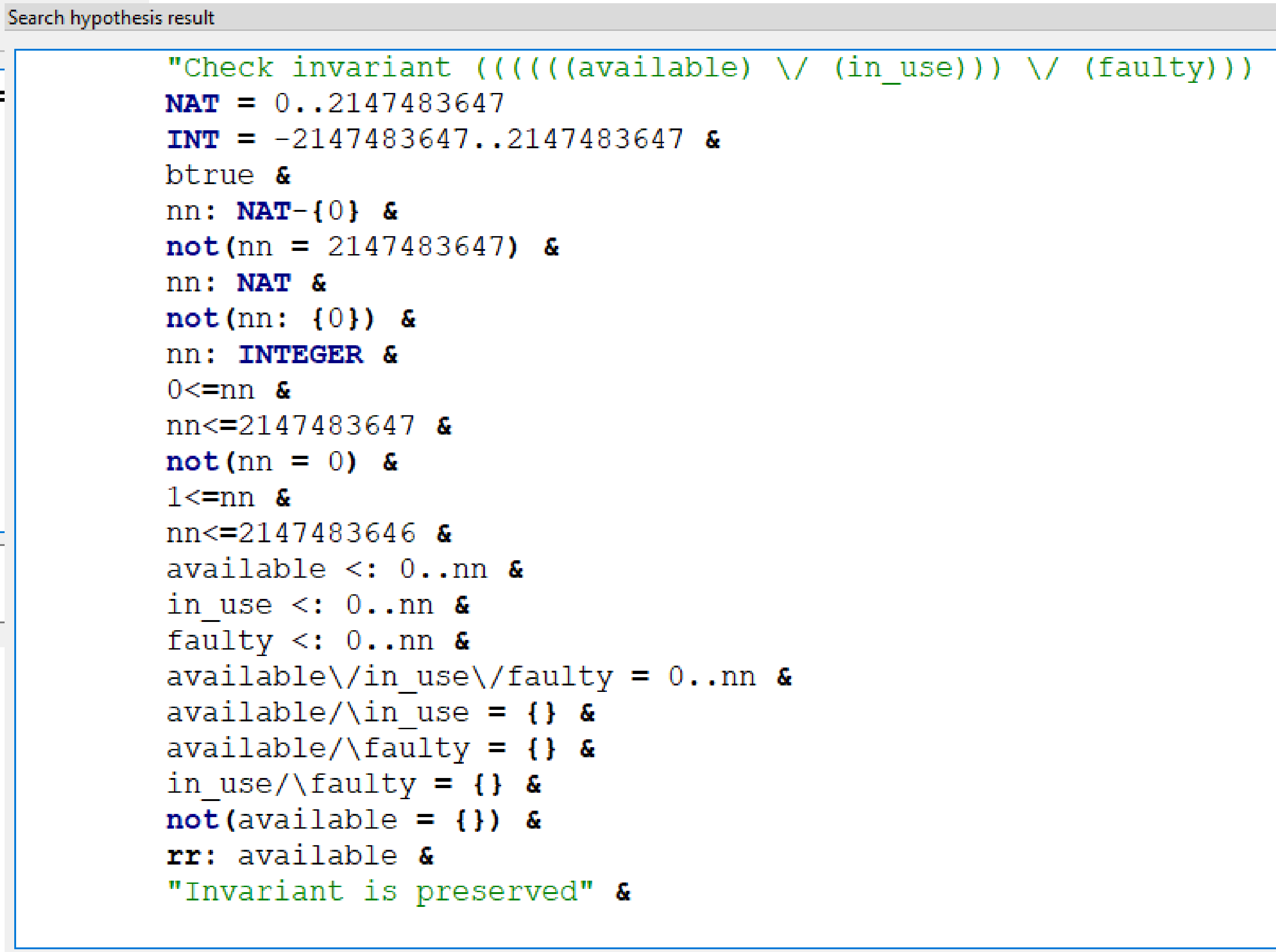
≡ Interactive Proof UI



sh(<formula>)

Search Hypothesis

Show all hypotheses matching the formula



Proof System

Interactive Proof UI

sh(a=b)

```
Hypothesis containing "a = b"
        NAT = 0..2147483647
        INT = -2147483647..2147483647 &
        not(nn = 2147483647) &
        not(nn = 0) &
        available\/in_use\/faulty = 0..nn &
        available/\in_use = {} &
        available/\faulty = {} &
        in_use/\faulty = {} &
        not(available = {}) &
```

sh(not(a))

```
Hypothesis containing "not(a)"
        not(nn = 2147483647)
        not(nn: {0}) &
        not(nn = 0) &
        not(available = {}) &
```

sh(available)

```
Hypothesis containing "available"
        available <: 0..nn
        available\/in_use\/faulty = 0..nn &
        available/\in_use = {} &
        available/\faulty = {} &
        not(available = {}) &
        rr: available &
```

sh(available _and rr)

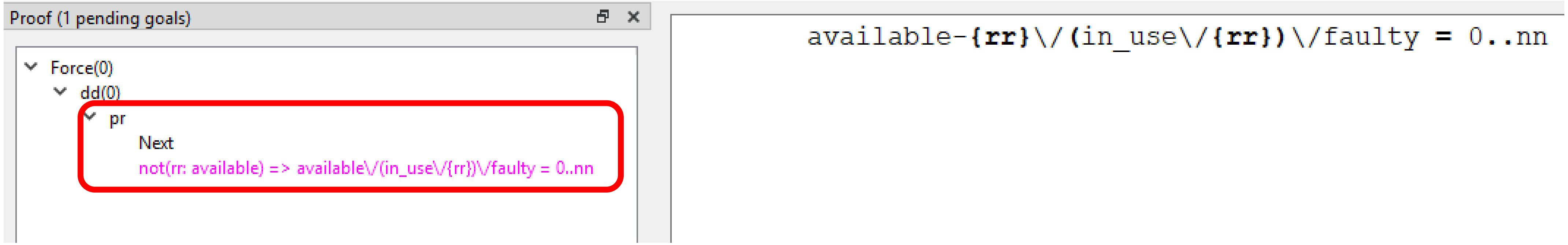
```
Hypothesis containing "available _and rr"
        rr: available
```

Your turn: search for typing hypotheses

Proof System

I did pr

≡ Interactive Proof UI



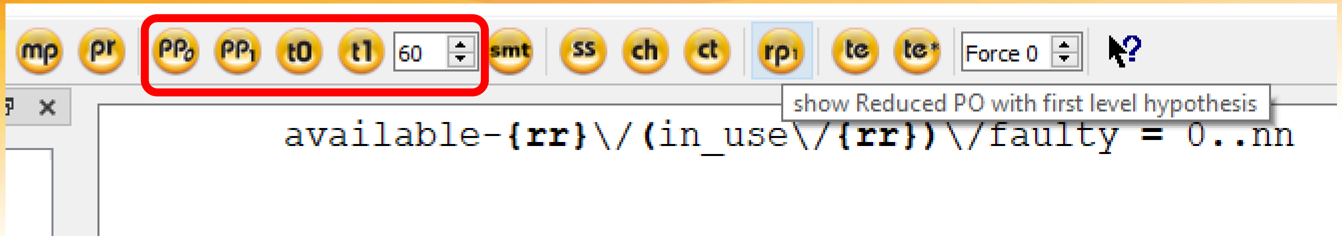
The prover has started a proof by case on rr: available

We are in the first case: rr: available => G

The second case is pending (in pink): not(rr: available) => G

Proof System

Interactive Proof UI



Predicate Prover

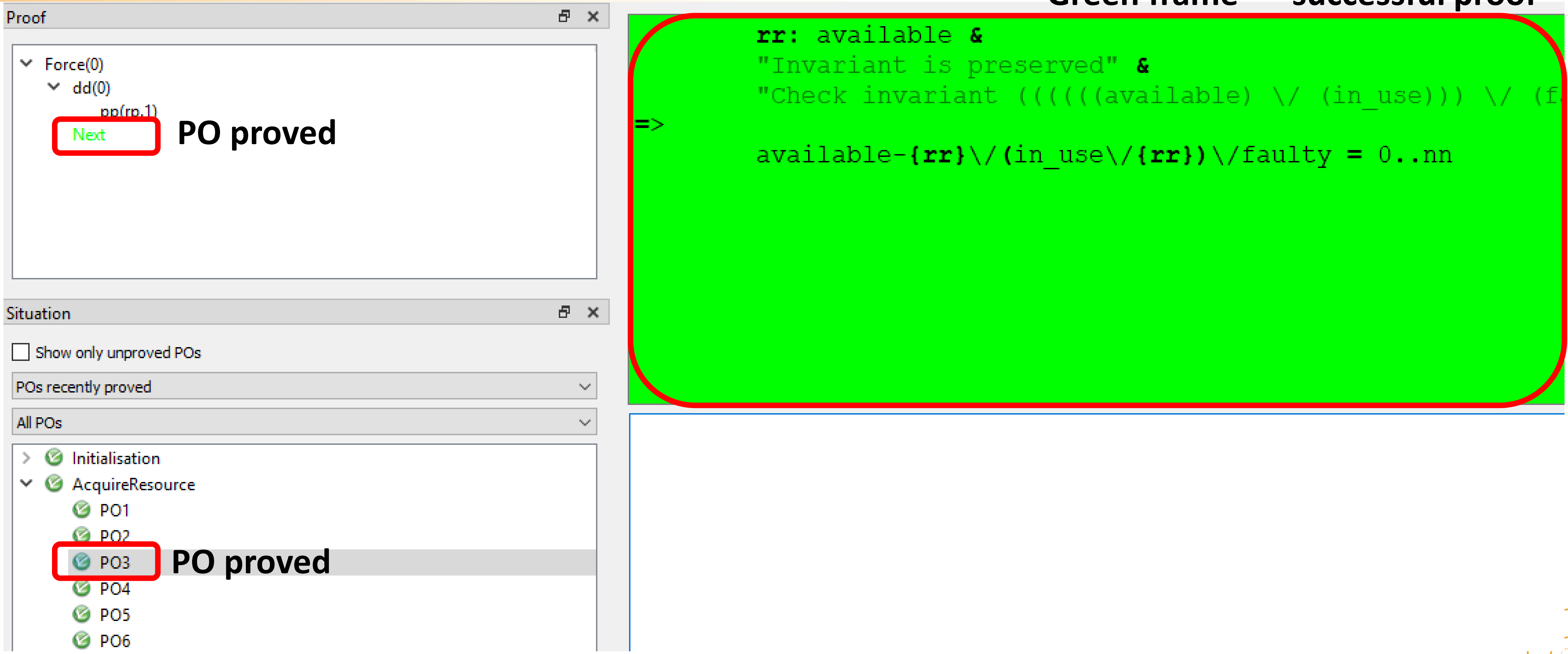
- **Based on tableau-method**
- **Used to prove predicates with few hypotheses**
- **pp0 : predicate prover on goal**
- **pp1: predicate prover with first level HYP**
- **pp(rp.0): predicate prover with typing HYP**
- **pp(rp.1): predicate prover with first level and typing HYP**

Your turn: complete the proof with pp(rp.1)

Proof System

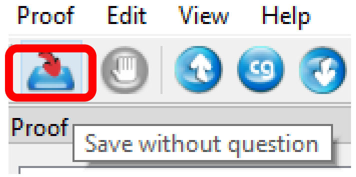
≡ Interactive Proof UI

Green frame == successful proof

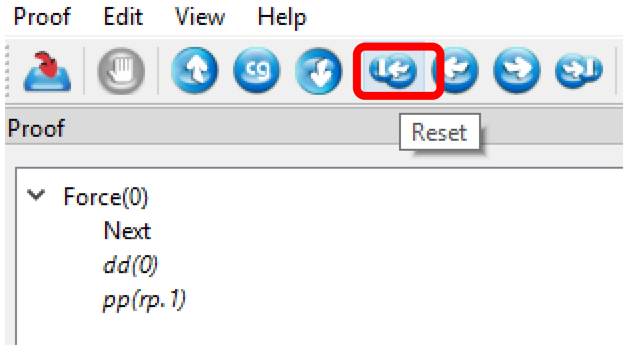


Proof System

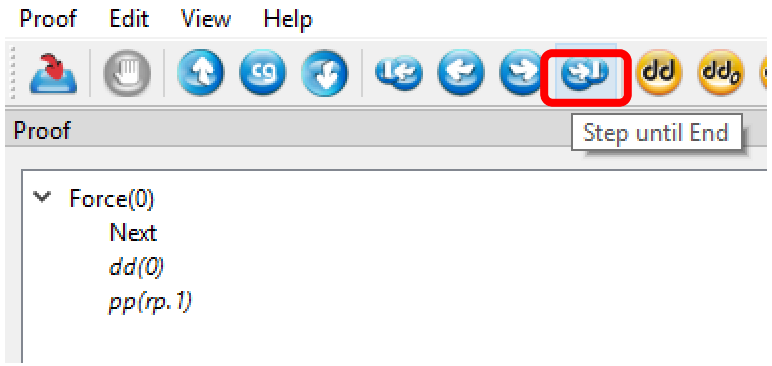
Interactive Proof UI



Save the demonstration



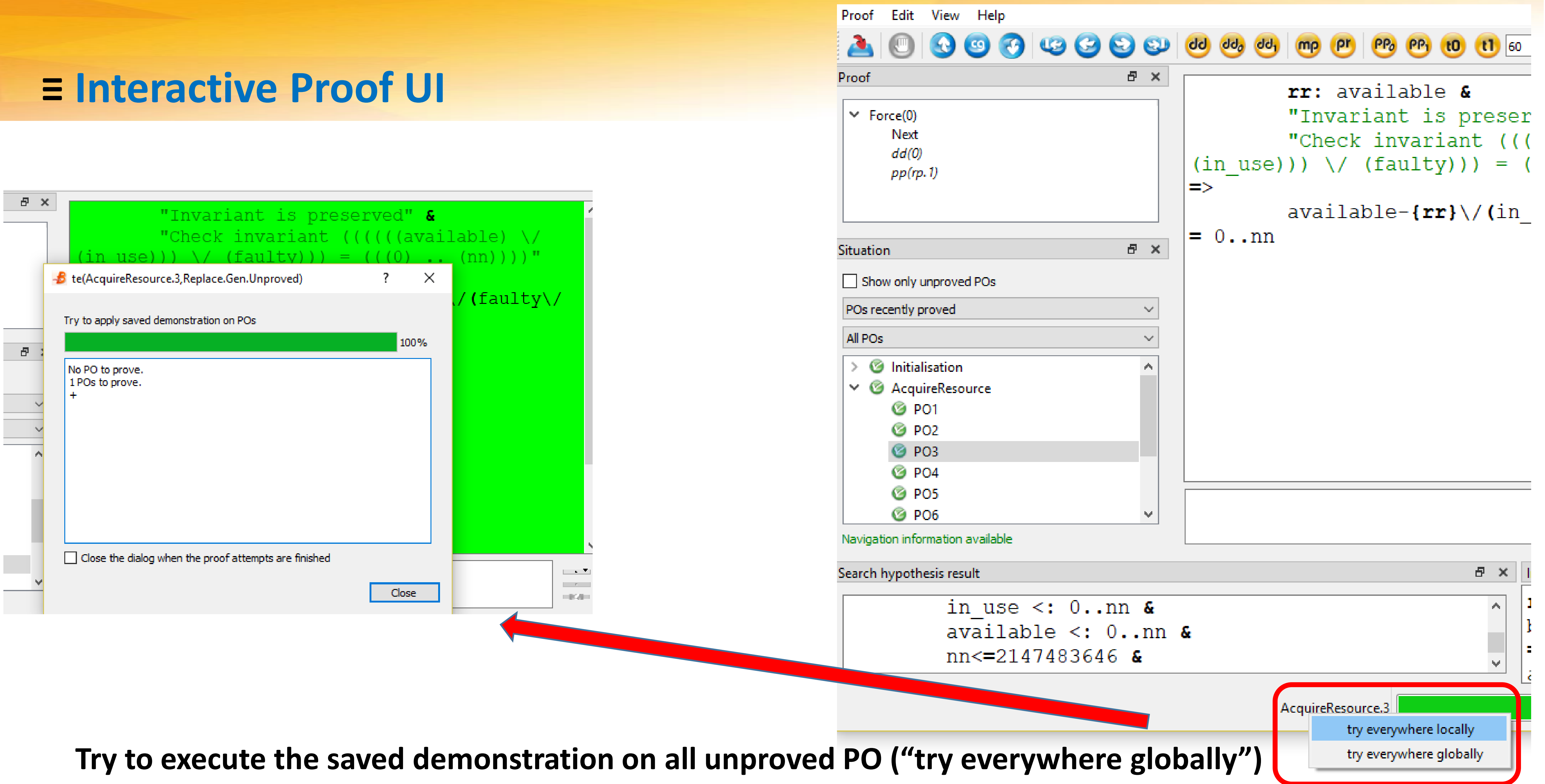
Reset the proof: the demonstration appears in italic



Step until end: replay the saved demonstration

Proof System

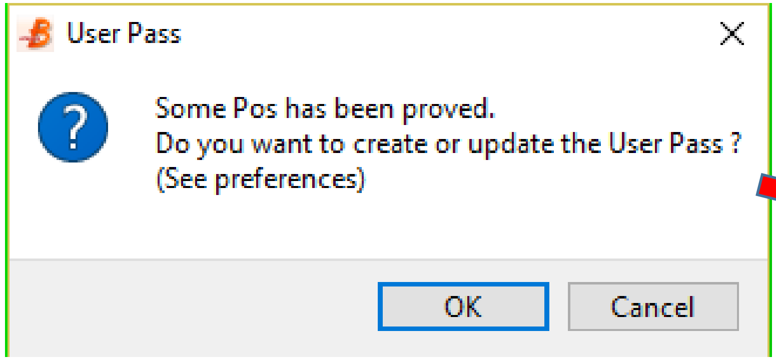
≡ Interactive Proof UI



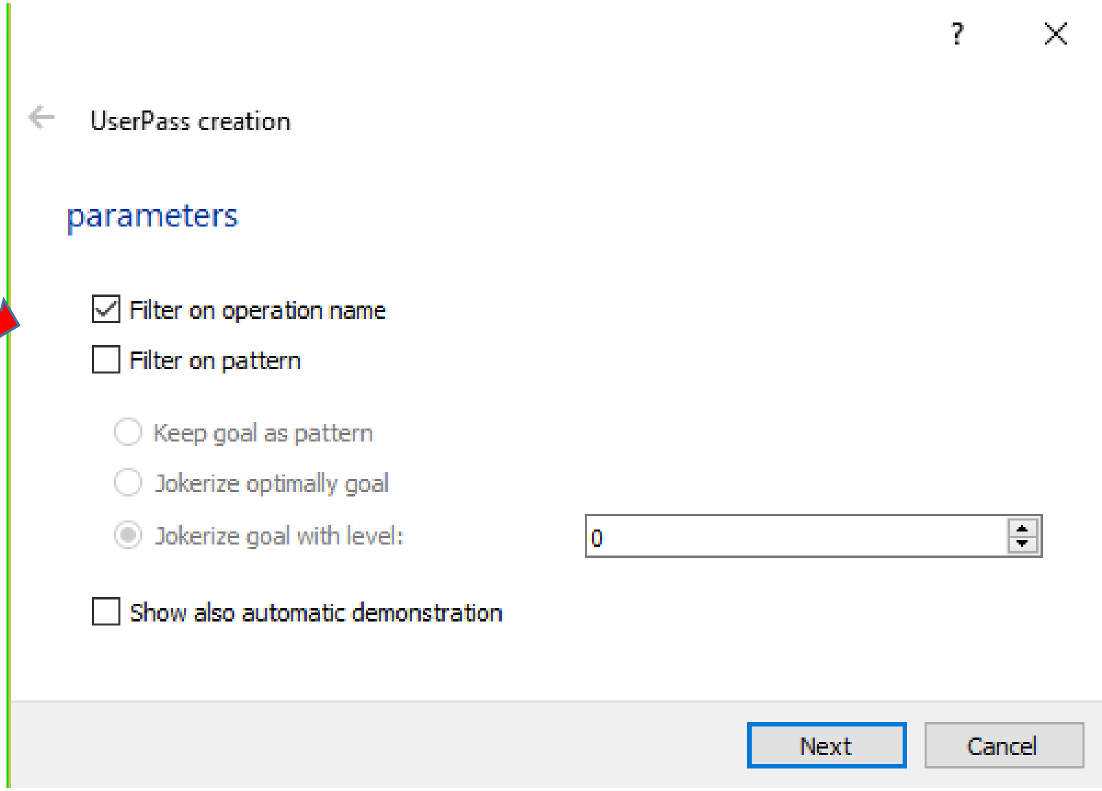
Try to execute the saved demonstration on all unproved PO ("try everywhere globally")

Proof System

Interactive Proof UI



The proof is successful
You are asked to keep the new demonstration in the User Pass

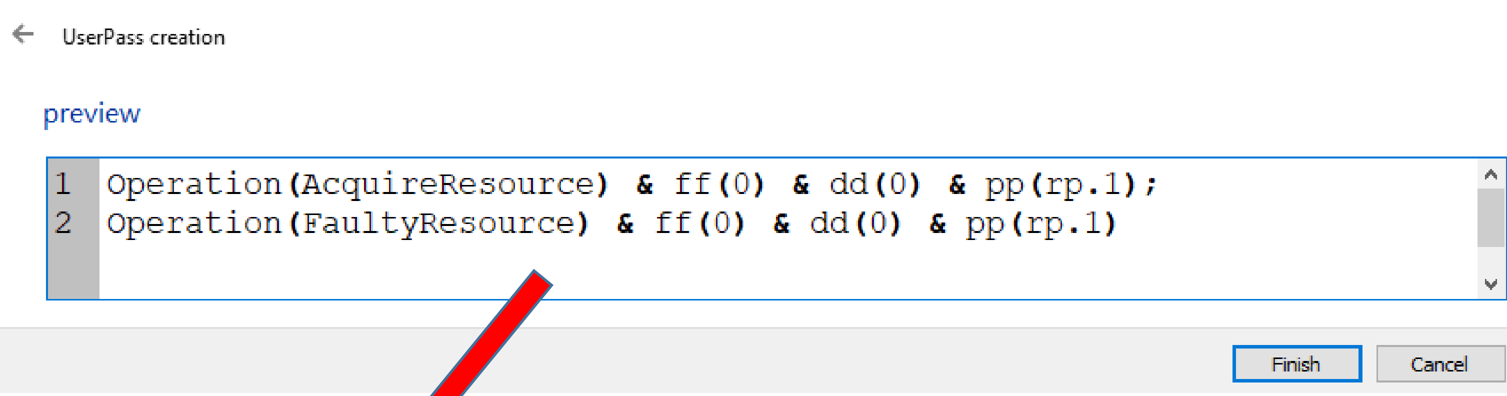


The User Pass contains all proof tactics of a component

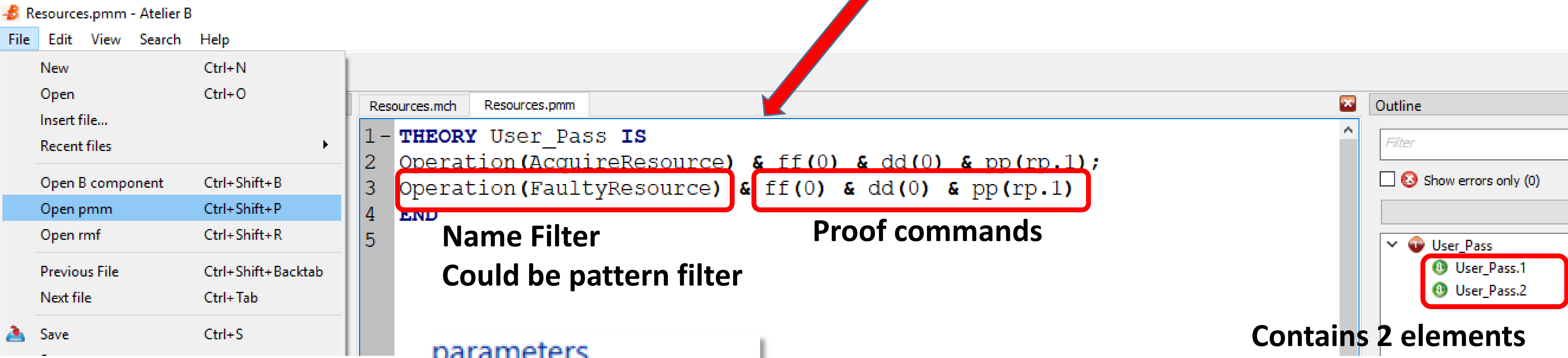
CLEARSY SYSTEMS ENGINEERING

Proof System

≡ Interactive Proof UI



- **Open the Model Editor for “Resources”**
- **Select File / Open pmm**

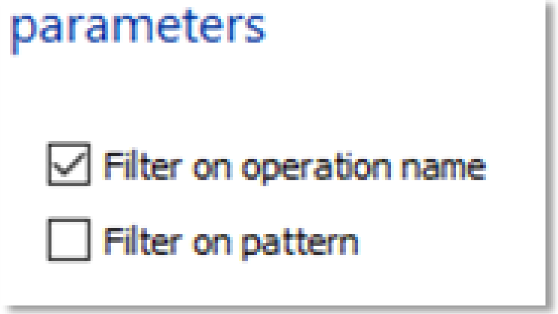


Name Filter

Could be pattern filter

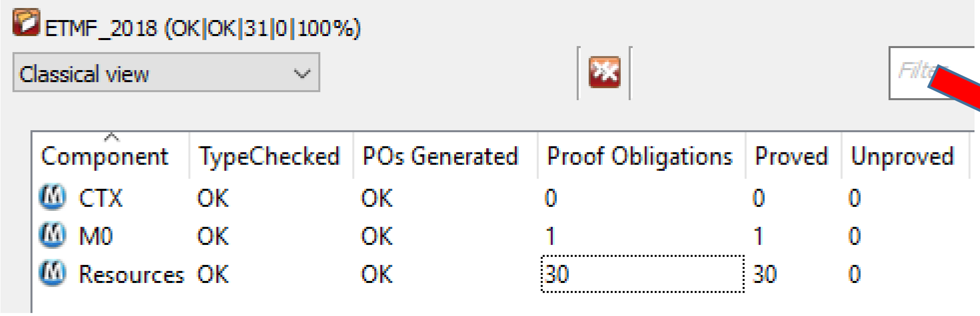
Proof commands

Contains 2 elements



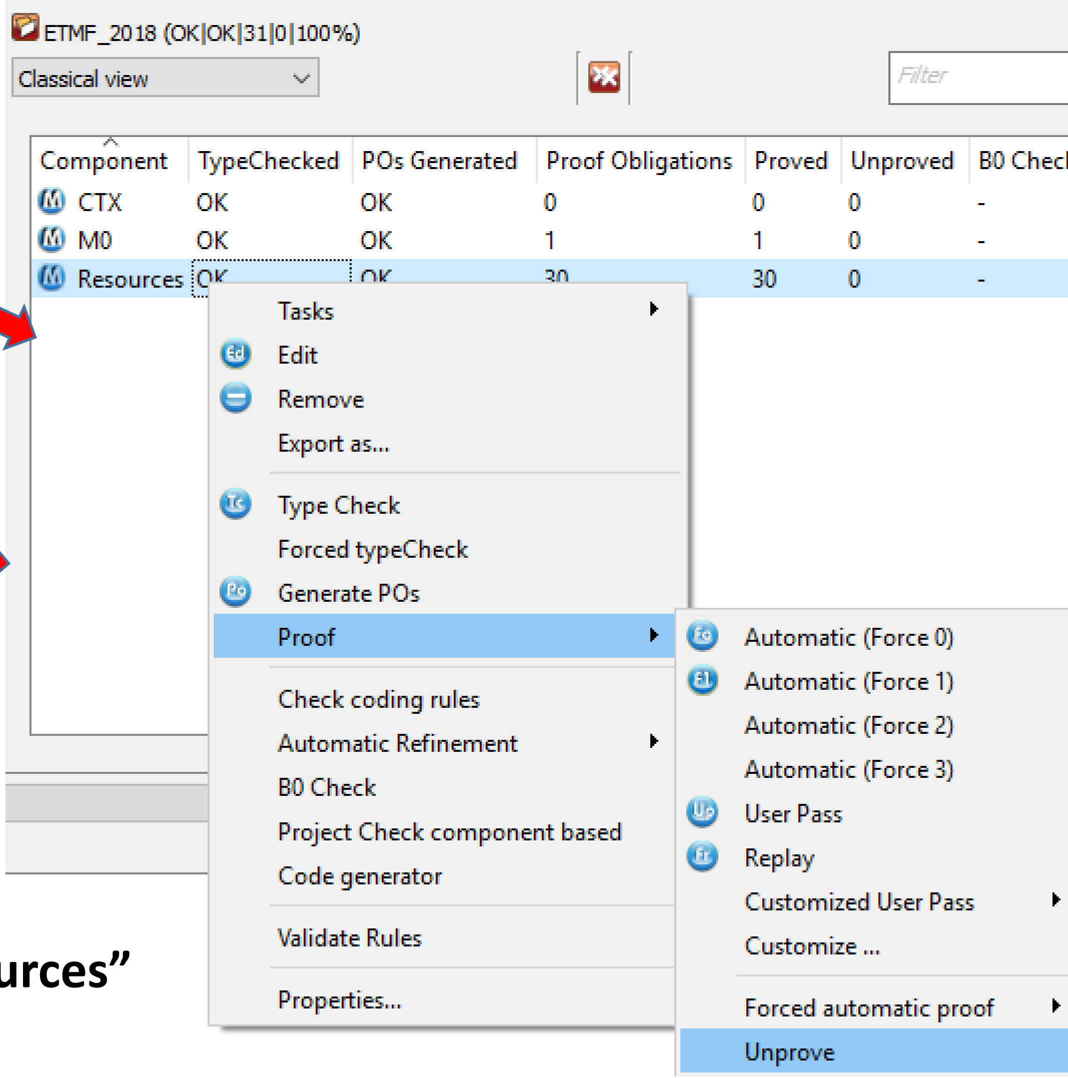
Proof System

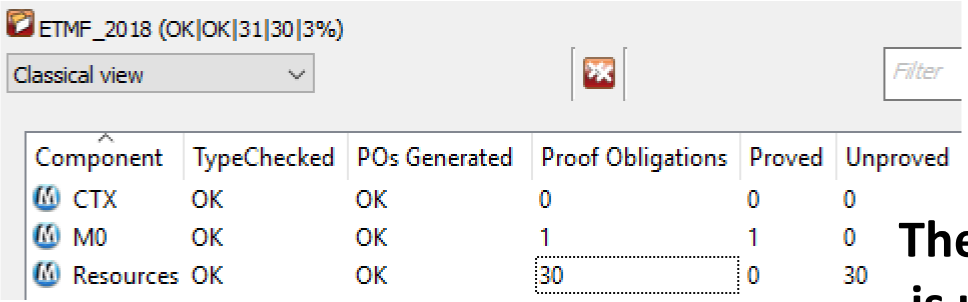
Checking Proof Replay with User Pass



The component "Resources" is fully proved

- **Select the Component "Resources"**
- **Select Component/Proof/Unprove**





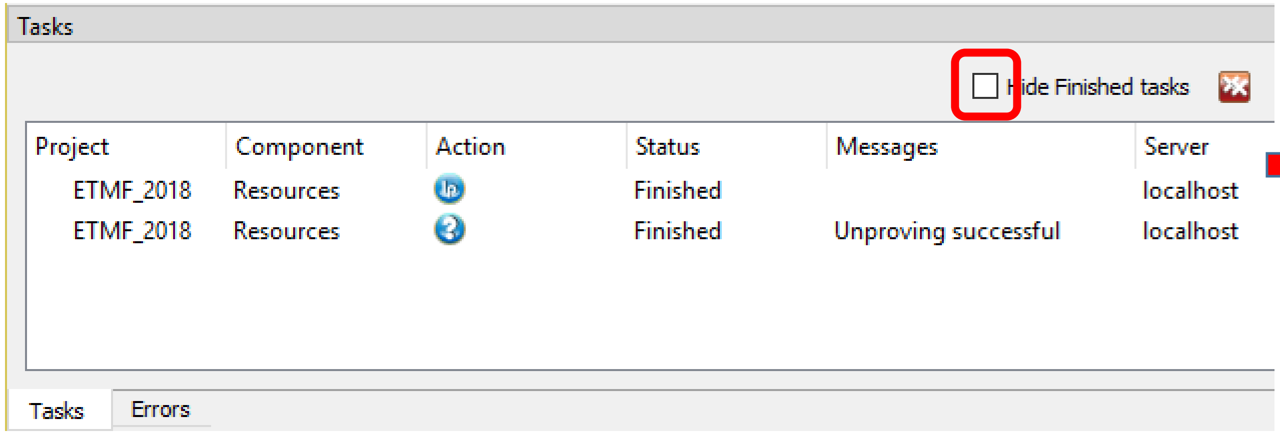
The component "Resources" is unproved

≡ Checking Proof Replay with User Pass

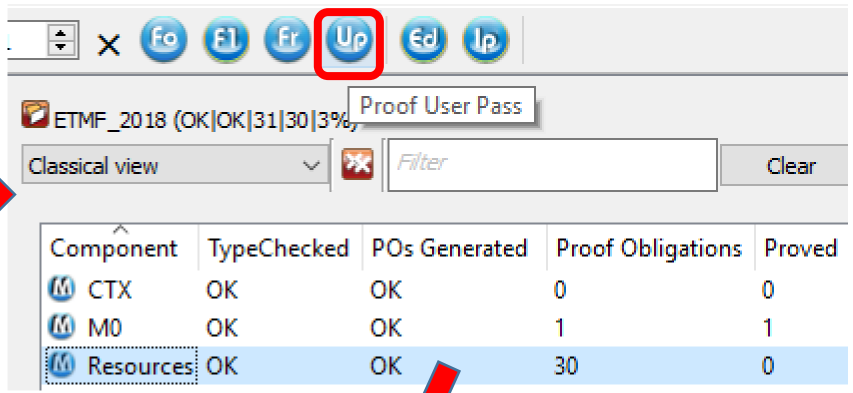
- Select the component “Resources”
- Click on “Up” (Proof User Pass)

User Pass tactics are applied sequentially on remaining POs

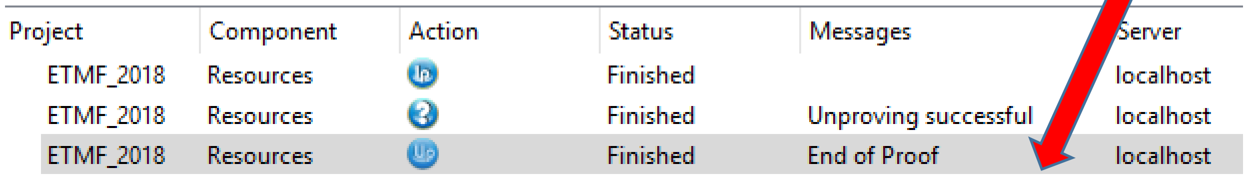
Uncheck the “hide finished tasks”



Project	Component	Action	Status	Messages	Server
ETMF_2018	Resources		Finished		localhost
ETMF_2018	Resources		Finished	Unproving successful	localhost



Component	TypeChecked	POs Generated	Proof Obligations	Proved
CTX	OK	OK	0	0
M0	OK	OK	1	1
Resources	OK	OK	30	0



Project	Component	Action	Status	Messages	Server
ETMF_2018	Resources		Finished		localhost
ETMF_2018	Resources		Finished	Unproving successful	localhost
ETMF_2018	Resources		Finished	End of Proof	localhost

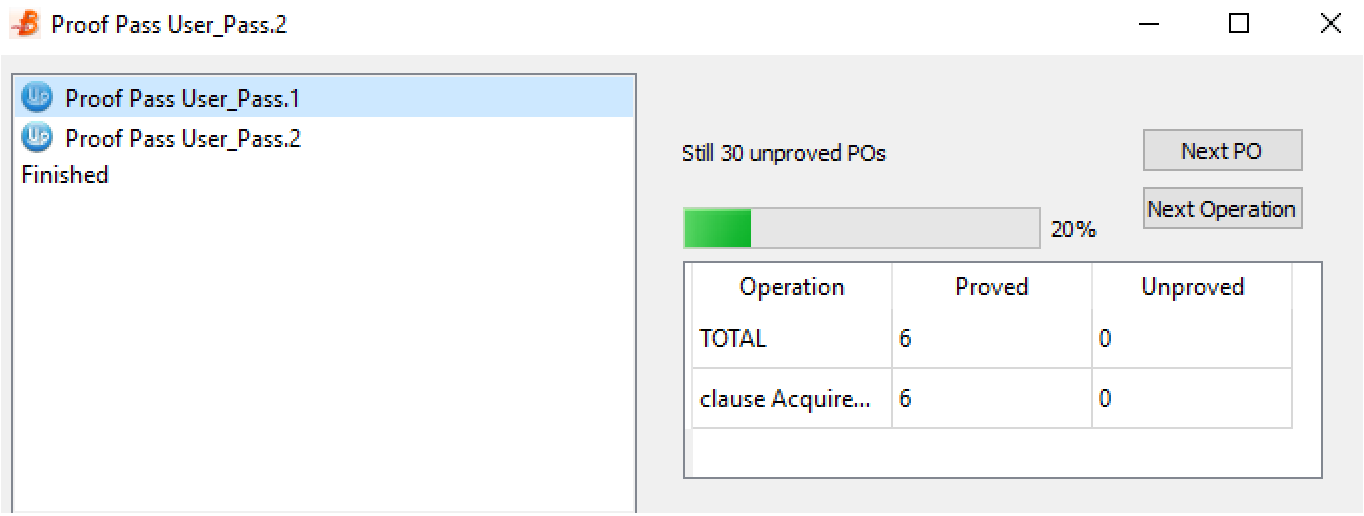
When the execution is finished, double click on the task description

CLEARSY SYSTEMS ENGINEERING

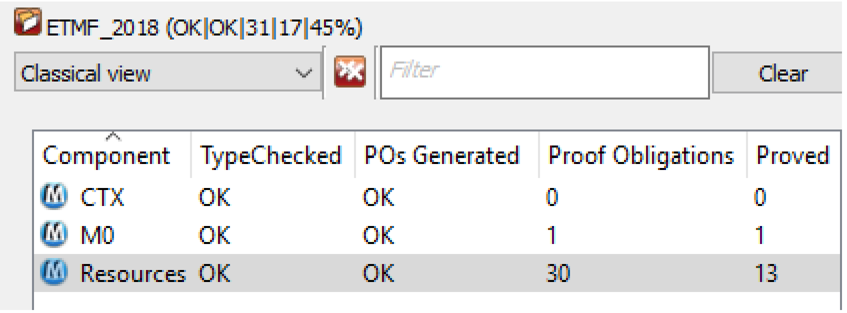
Proof System

```
1 Operation(AcquireResource) & ff(0) & dd(0) & pp(rp.1);
2 Operation(FaultyResource) & ff(0) & dd(0) & pp(rp.1)
```

≡ Checking Proof Replay with User Pass

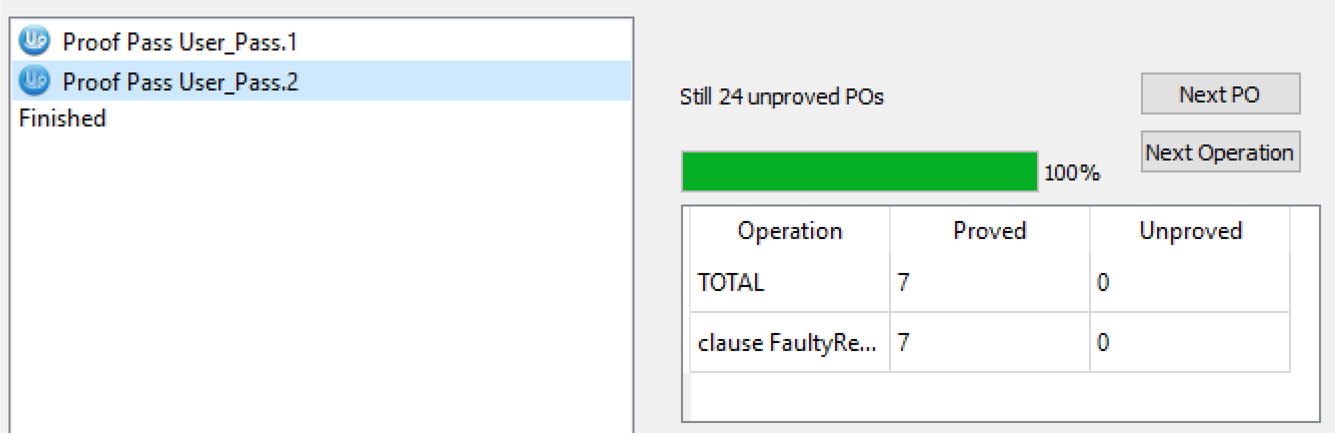


UserPass.1 proved 6 POs of AcquireResource



New Proof status

The tactics proved more than when used after pr



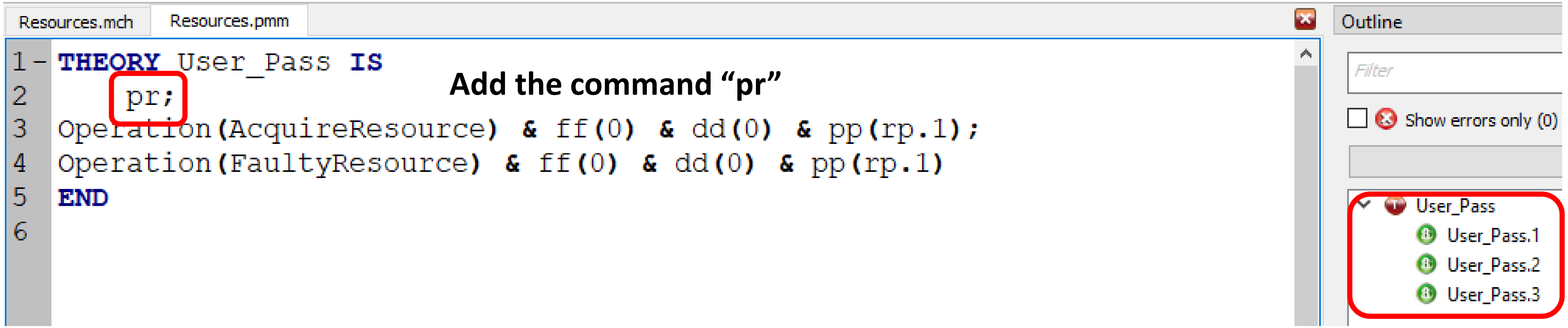
UserPass.2 proved 7 POs of FaultyResource

CLEARSY SYSTEMS ENGINEERING

Proof System

≡ Checking Proof Replay with User Pass

Let us edit the Resources UserPass

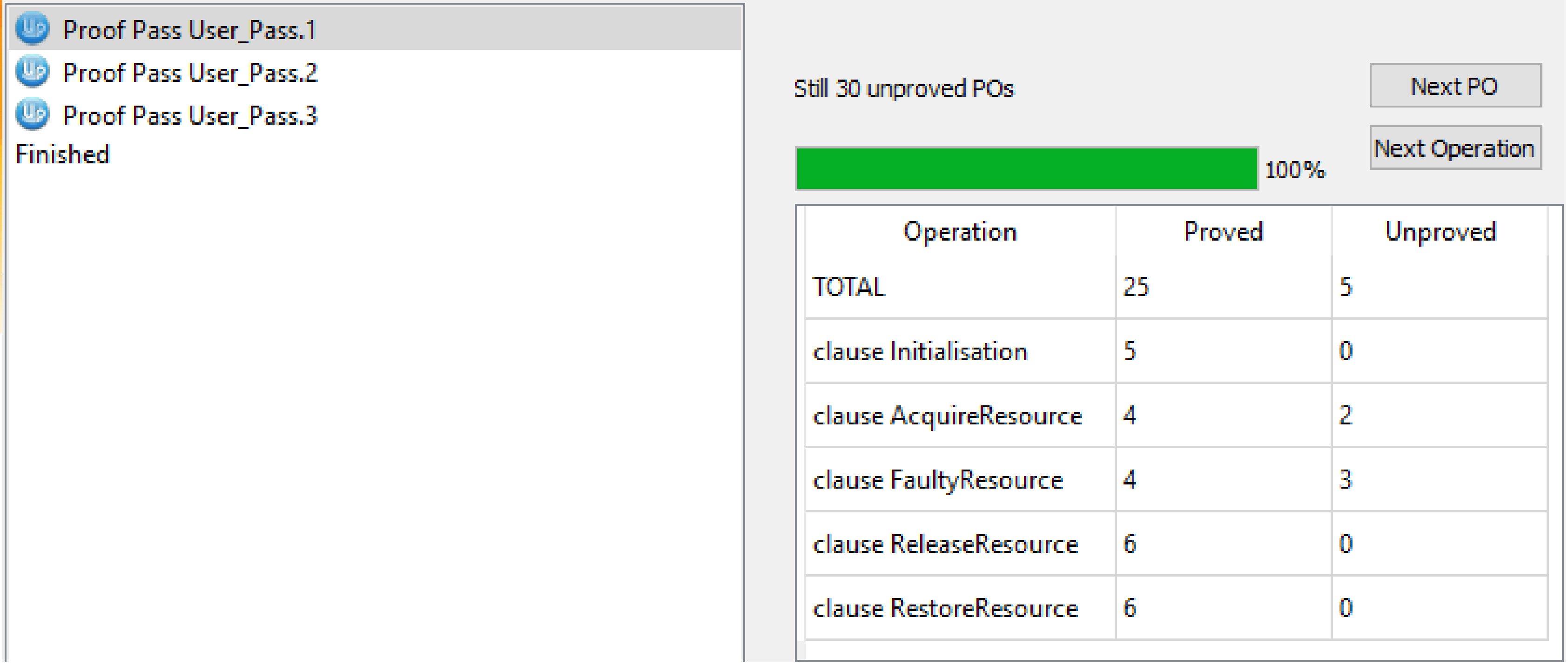


When saved, we get 3 User Passes

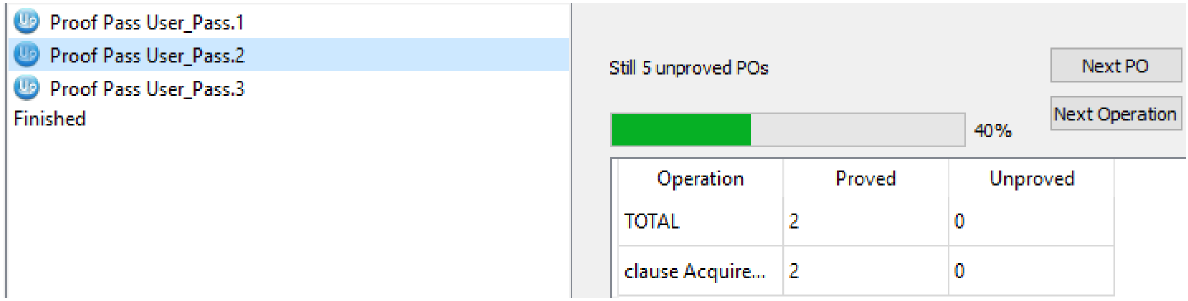
- **Unprove the component "Resources"**
- **Select "UP" (Proof User Pass)**
- **Double click the task description when completed**

Proof System

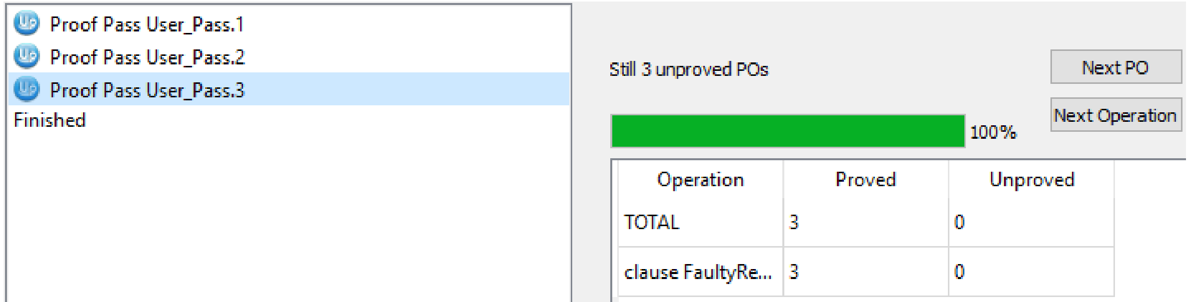
≡ Checking Proof Replay



The component is now proved in a single operation



The very idea is to avoid to lose interactive demonstration when a model is modified

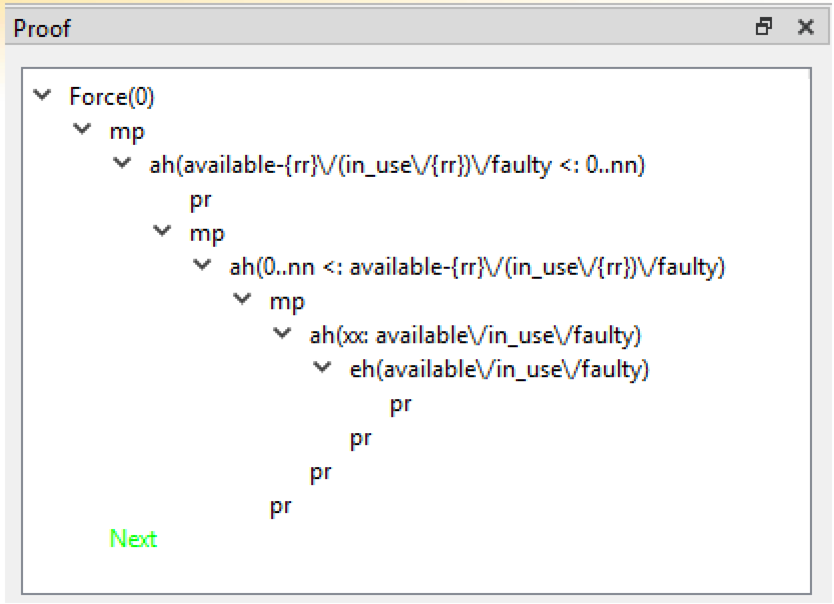


Proof System

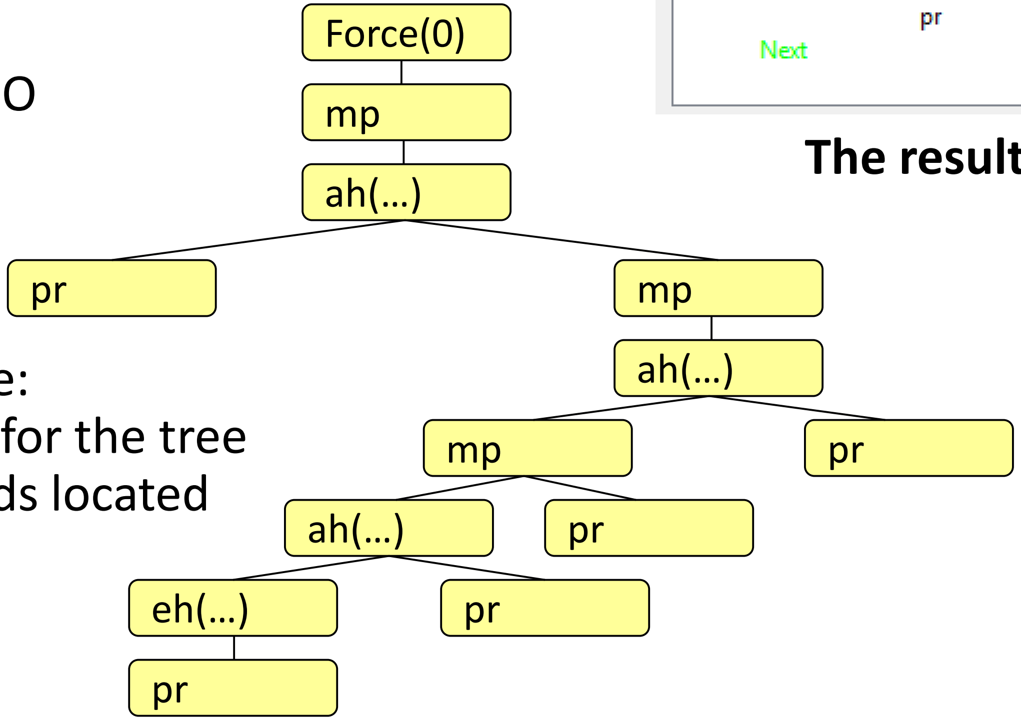
≡ Back to the Proof Tree

- Interactive Prover: go to the PO AcquireResource.3 of the component Resource0
- open in an editor the file DemoResL
- Copy the sequence of commands
- Paste it in the command pane
- Press return

These commands should proof the current PO



The resulting proof tree



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

Proof System

mp with force 0 and 1 starts prover without proof by cases tactics

≡ Using mp

example

- add machine MiniPr, start prover with force 3,
- examine unproved PO, the goal can be simplified,

```
aa-((0..10/\{3})-(8..12)) = aa-{3}\/(aa/\8..12)
```

- mp, in the goal the expression 0..10 /\ {3} in simplified in {3}
- pp(rp.0) the PO is proved

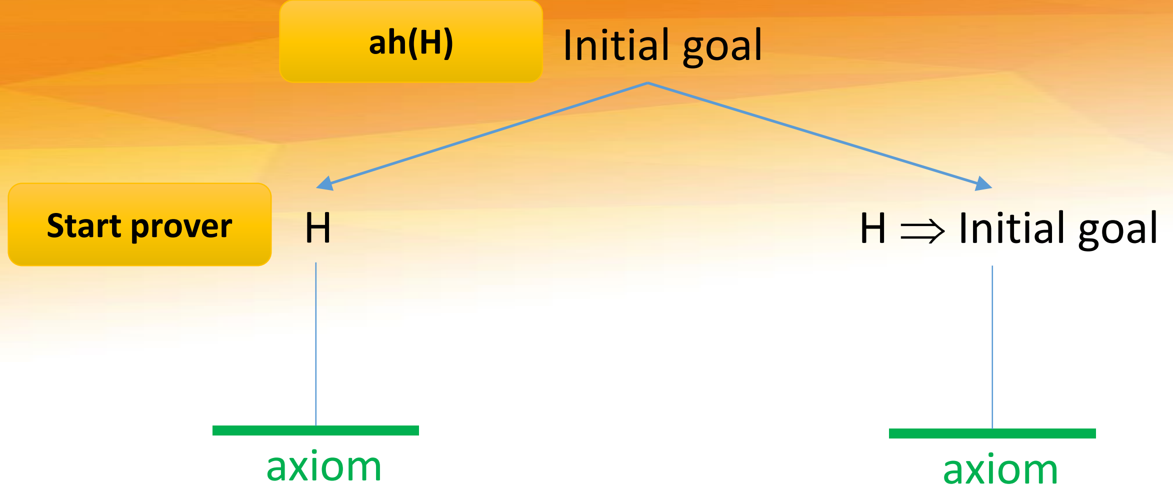
difference between mp and pr

- restart demonstration by replacing mp by pr: re & pr & pp(rp.0)
- the remaining goal is not(vv = 3) => Q
- in fact, the prover did 2 cases because of the hypothesis vv = 3 => not(a = {nn}), it's useless

```
Force(0)
  pr
    Next
    not(vv<=3) => aa-({3}-(8..12)) = aa-{3}\/(aa/\8..12)
```

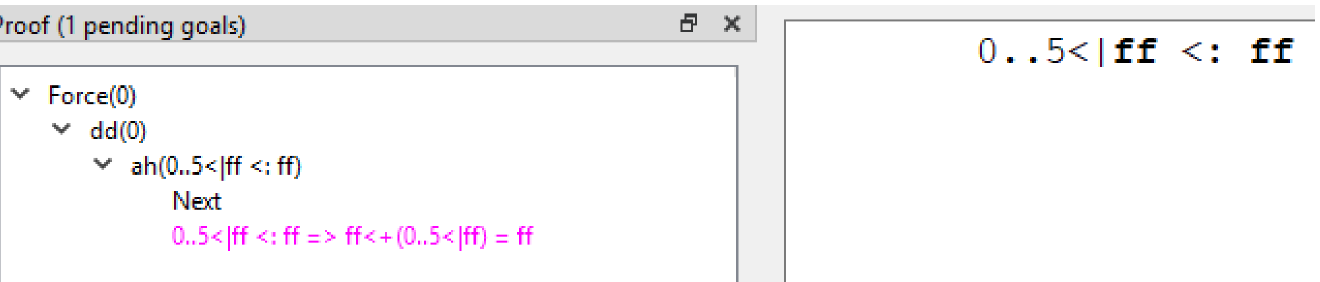
Proof System

≡ Adding Hypothesis: ah



example

- add component AddHyp and start proof with force 0 (do not use force(2), is proved)
- 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 equalities: eh

- 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))

Proof System

≡ Suggest For Exist: se

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) <=> P(a)

```
#ss.(ss <: NAT & not(ss = {}) & 3: ss)
```

Interactive demonstration

mp & se({3}) & pr

Existential goals are found

- in an ANY xx WHERE ... non directly refined
- when using non refined abstract constants

≡ Manual Creation of Hypotheses: ph & mh

instanciation of a « for all » predicate

- ph(x0, !x.(P_x => Q_x)) to particularize !x.(P_x => Q_x) for the value x0
- first P_{x0} has to be proved, then the new hypothesis Q_{x0} is generated

use of an « imply » hypothesis (Modus Ponens rule)

- mh(P => Q)
- under the hypotheses P=>Q and P, the new hypothesis Q is generated

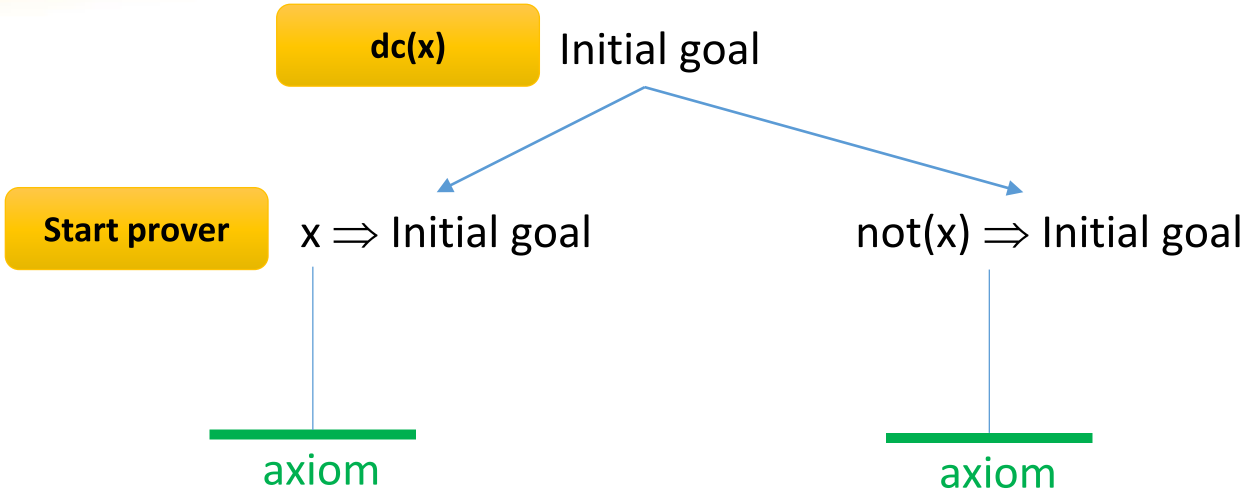
Proof System

≡ Proof by Case: dc

The interactive command dc(x) tries to prove the current goal in two cases: x and not(x)

Its action on the proof tree is as follow:

- The current goal has to be proved under the hypothesis x
- The current goal has to be proved under the hypothesis not(x)



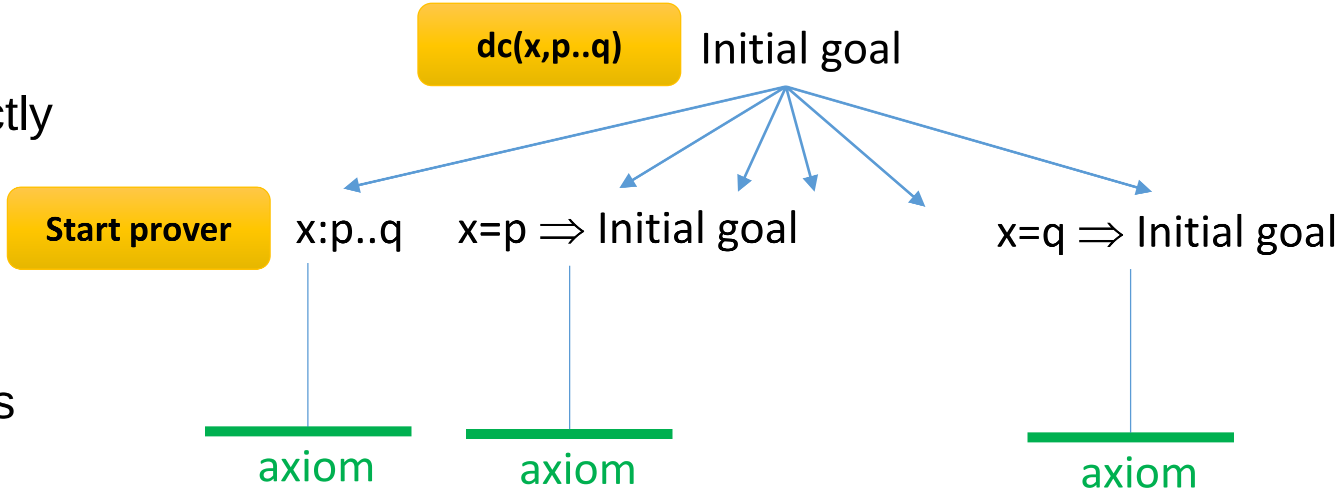
Proof System

Proof by Case: dc

The interactive command dc(x, p..q) tries to prove the current goal for all possible values of x that should strictly belong to p..q

Its action on the proof tree is as follow:

- x: p..q has to be proved
- The current goal has to be proved under the various hypotheses x=p, x=p+1, x=p+2, …, x=q



Proof System

- Mathematical Rules

Proof System

Atelier B Main Prover contains more than 2500 rules

Mathematical Rules

- Break down the current goal into smaller parts or parts easier to prove **[type 1]**

```
InRelationXY.1
dom(a) : POW(s)
ran(a) : POW(t)
=>
a : s <-> t
```

proving a : s <-> t is equivalent to proving dom(a) : POW(s) and ran(a) : POW(t)

- Some rules are axioms for the prover

axiom

```
InFINXY.127
p..q : FIN(INTEGER)
```

any interval p..q is a finite subset of INTEGER

```
EqualityXY.131
binhyp(B\/A = C)
=>
A\/B = C
```

A\/B = C is true if B\/A = C is an hypothesis

Proof System

Mathematical Rules

Simplify predicates (goal or hypotheses) **[type 2]**

SimplifyRelImaXY.27
(r|>u)[v]
==
r[v]/\u

(r|>u)[v] is rewritten in r[v]/\u

Proof System

≡ Mathematical Rules

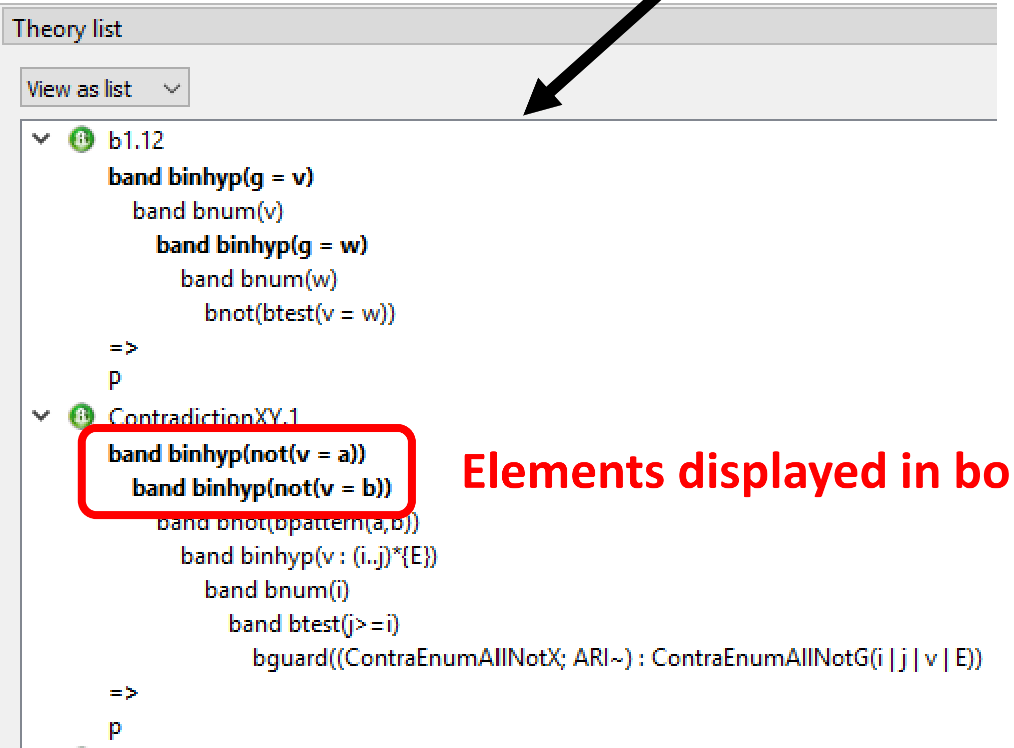
Generate new hypotheses by combining them **[type 3]**

GenEqualityX.1
a<=b
b<=a
=>
a = b

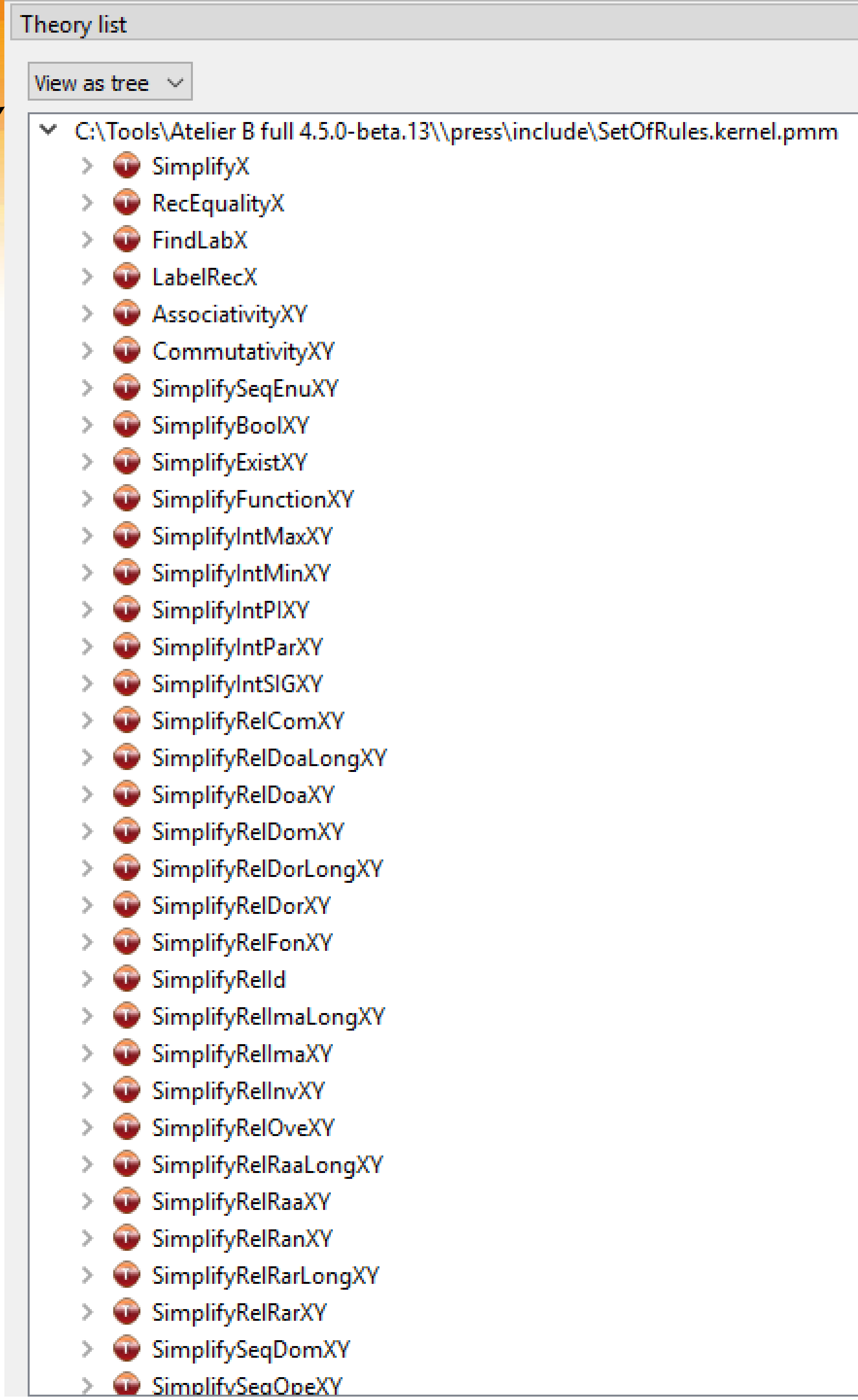
if a<=b is a new hypothesis and b<=a an existing hypothesis then generate hypothesis a = b

Proof System

- Rules available on the “Theory List” panel
- Grouped in packages
- View as a tree: all rules displayed
- View as a list: only the rules that can be triggered displayed



Elements displayed in bold hold



Proof System

Mathematical Rules

User rules should be used when everything else failed

- the number of rules added should be as small as possible
- Reasons:
  - symbol not (well) covered (ex: transitive closure)
  - simplify or generalize complex proofs

Rules are added in:

- Component file (<component.pmm>) – rules are only visible by the component
- PatchProver (bdp/PatchProver directory) – rules global to the project

Rules may be validated by the predicate prover (but again no guaranty that validrules are always demonstrated)

Proof System

Add component Rules project

≡ Mathematical Rules

- go to AssertionLemmas.3, type in dd to load hypotheses

```
10..20<|ff = ff
```

- add the rule (; is required to separate 2 rules)
  - `f: S +-> T => S<|f = f`
- compile and apply this rule: pc & ar(MyRules.1,Once)
  - displayed goal contains a one-letter identifier: non provable.

Wilcard instantiation (one-letter identifier) is only done within the goal

if a joker has to be instantiated by a hypothesis, binhyp should be used

binhyp is a guard. If the guard is evaluated as true, the rule is applied.

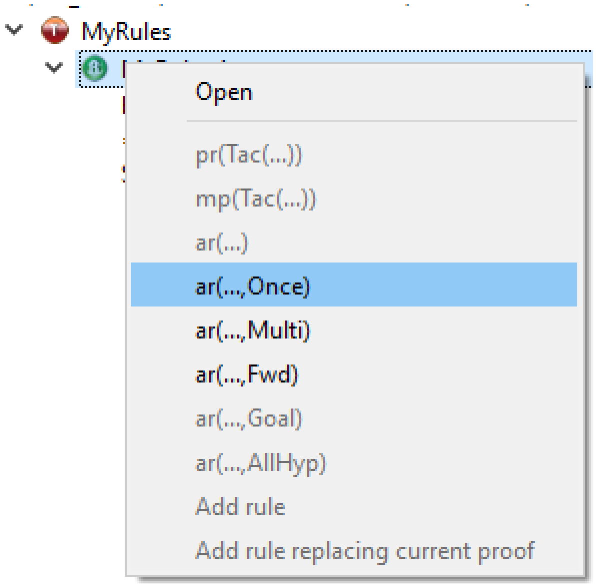
CLEARSY SYSTEMS ENGINEERING

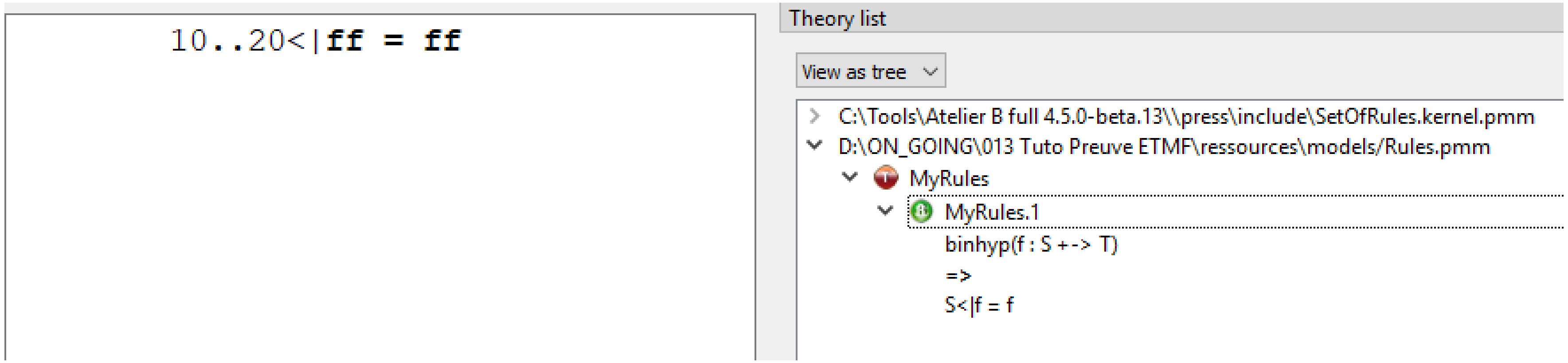
Proof System

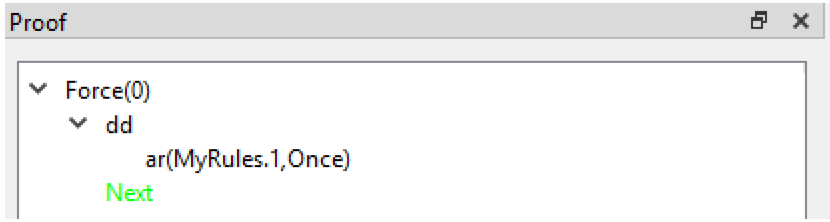
Add component Rules project

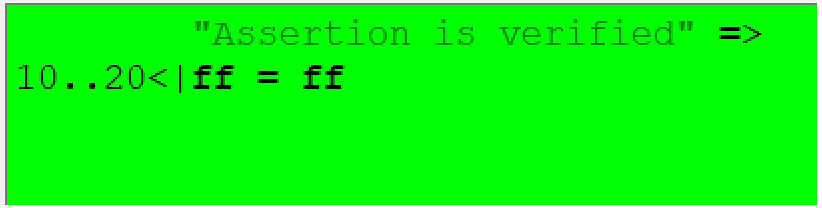
≡ Mathematical Rules

- correct the rule
  - `binhyp(f: S +-> T) => S<|f = f`
- retry: it works









Proof System

≡ Proof with Assertions

Assertions are predicates which are part of B models their only role is to ease proof

2 kinds of assertions :

- clause ASSERTIONS - global to a component
  it should be deduced from the invariant and the previous assertions (order is important)
  assertions become hypothesis of other PO

- substitutions ASSERT - 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

≡ Proof with Assertions: 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,
- all PO have the same complexity

```
ASSERTIONS
  ff~ : NATURAL +-> 0..100
```

ETMF_2018 (OK|OK|59|5|91%)

Classical view

Component	TypeChecked	POs Generated	Proof Obligations	Proved	Unproved
Assertions	OK	OK	12	11	1
Assertions0	OK	OK	16	12	4

advantage: assertions allows to factorise proof of operations of a component.

Proof System

≡ Proof with Assertions: substitution ASSERT

example: machine Assert and refinements Assert_1 et Assert_0 (with and without assertion)

- examine differences between these 2 refinements,

```
ASSERT
    (zz >= 1) <=> (xx >= 0)
```

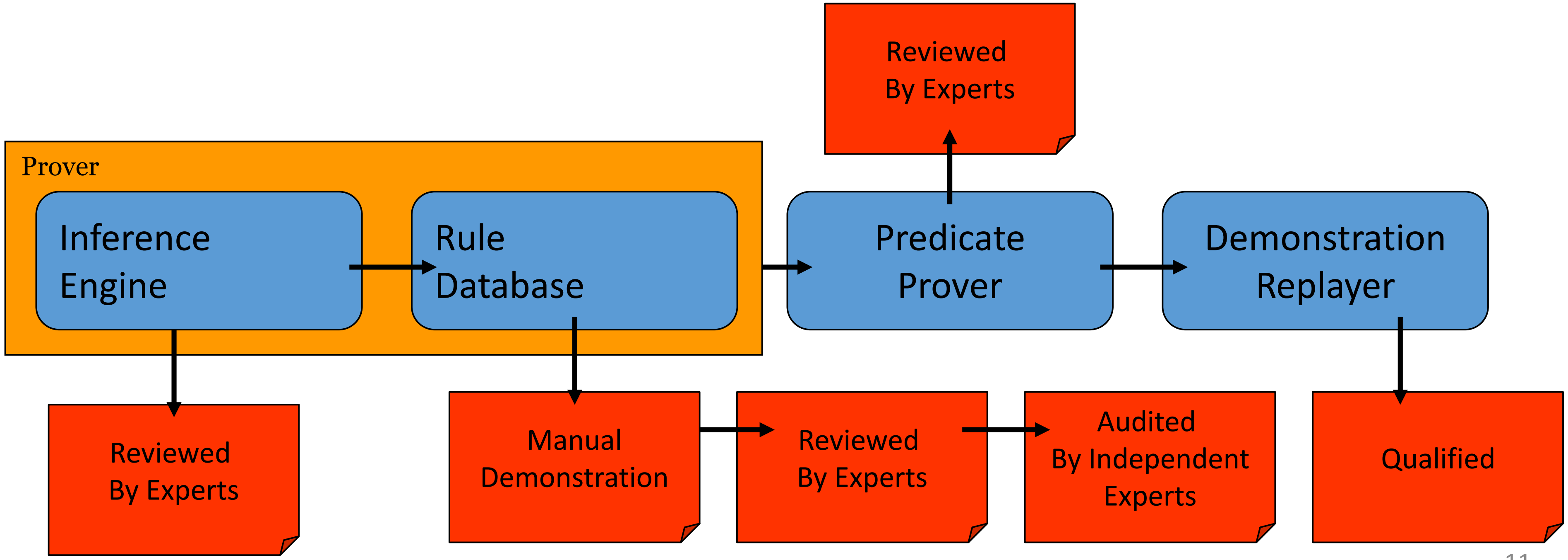
- assertion precise how the IF is refined, the case yy >= 1 corresponds with the case xx >= 0 of the specification,
- add these components and start proof with force 0,
- PO have the same complexity

Component	TypeChecked	POs Generated	Proof Obligations	Proved	Unproved
Assert	OK	OK	1	1	0
Assert_0	OK	OK	3	1	2
Assert_1	OK	OK	4	3	1

advantage: assertions allow to ease proof of an operation.

Complete the proof of the remaining POs (hint: with only one command)

Prover Qualification



Proof System

Proof Algorithm

considering only proof requiring more than one step

Have a look at the goal

Search for related hypotheses

Identify (nearly) applicable rules

Identify missing information

New hypothesis

New simplification / resolution rule

Add information

One step ahead: try to simplify/solve

Proof System

≡ Application DMS Sequencer

Event-B model of an inertia central SW sequencer

Used for SW validation

11 refinements

30% automatic proof only …

Project Status for SEQ

Component	TC	POG	nPO	nUN	%Pr
dms00	OK	OK	42	18	57
dms01	OK	OK	1	1	0
dms02	OK	OK	5	5	0
dms03	OK	OK	16	8	50
dms04	OK	OK	16	8	50
dms05	OK	OK	18	12	33
dms06	OK	OK	17	13	23
dms07	OK	OK	12	8	33
dms08	OK	OK	24	17	29
dms09	OK	OK	50	40	20
dms10	OK	OK	31	19	38
dms_valuation09	OK	OK	32	32	0
dms_valuation09_r	OK	OK	6	6	0

Proof System

- PO

Model: dms00

Proof obligation: Swap.21

```
        "`Local hypotheses'" &
        time: INTEGER &
        morrow: INTEGER &
        victor: PROCESSES &
        leftspan: INTEGER &
        clock0+1<=time &
        time+1<=morrow &
        not(clock0..morrow/\dom(Schedule) = {}) =>
clock0..morrow/\dom(Schedule) = {time} &
        elected0: Tasks => time<=clock0+term0(elected0) &
        elected0: Tasks => leftspan = term0(elected0)-(time-clock0) &
        not(term0[Schedule[{time}]] = {}) => term0[Schedule[{time}]] = {0} &
        victor: {Phantom}\/Schedule[{time}]\/term0~[NATURAL-{0}] &
        victor = Phantom => Schedule[{time}] = {} &
        victor = Phantom => term0~[NATURAL-{0}] = {elected0} &
        victor = elected0 => 1<=leftspan &
        Schedule[{time}] = {} => elected0: Tasks &
        Schedule[{time}] = {} => time = clock0+term0(elected0) &
        task: Tasks &
        "`Check that the invariant (!task.(task: Tasks => SIGMA(time).(time:
(0..clock0-1<|Schedule)~[{task}] | Deadline(task)) = SIGMA(time).(time:
(dom(spans0)<|log0)~[{task}] | spans0(time)-time)+term0(task))) is preserved by
the operation - ref 3.4'"
=>
        SIGMA(time$0).(time$0: (0..morrow-1<|Schedule)~[{task}] |
Deadline(task)) = SIGMA(time$0).(time$0: (dom(spans0\/{clock0|->time})<|
(log0\/{morrow|->victor}))~[{task}] | (spans0\/{clock0|->time})(time$0)-
time$0)+(term0<+({Phantom}<<|{elected0|->leftspan}\/(Schedule[{time}]<|
Deadline)))(task)
```

≡ Rules

Swap.21

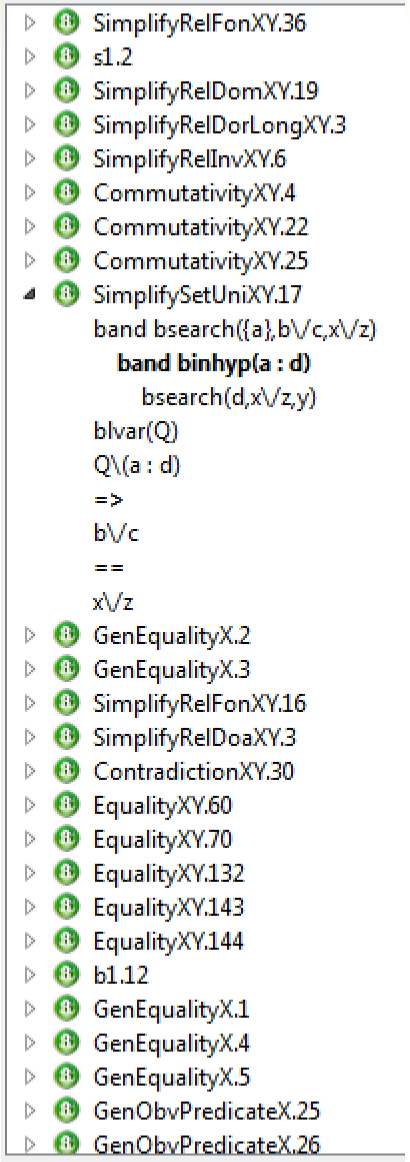
Demonstrate that $\sum_{t1} D(t1) = \sum_{t2} D'(t2)$

17 local hypotheses

39 hypotheses (16 for typing)

250 « related » mathematical rules

To help identifying missing bits, holding guards are bold



Proof System

Proof Algorithm

23 rules added to the whole project

- DMS_SIG
  - DMS_SIG.1
  - DMS_SIG.2
  - DMS_SIG.3
  - DMS_SIG.4
  - DMS_SIG.5
  - DMS_SIG.6
  - DMS_SIG.7
  - DMS_SIG.8
  - DMS_SIG.9
  - DMS_SIG.10
  - DMS_SIG.11
  - DMS_SIG.12
  - DMS_SIG.13
- DMS_DIV
  - DMS_DIV.1
  - DMS_DIV.2
  - DMS_DIV.3
  - DMS_DIV.4
- DMS_MOD
  - DMS_MOD.1
  - DMS_MOD.2
- DMS_MUL
  - DMS_MUL.1
  - DMS_MUL.2
- DMS_IND
  - DMS_IND.1
- DMS_FIN
  - DMS_FIN.1

```
/* DMS_SIG.5 */
  bmatch(x,P,Q,y) &
  bmatch(x,E,F,y) &
  x\(Q,F) &
  y\(P,E)
=>
  SIGMA(x).(P|E) = SIGMA(y).(Q|F)
```

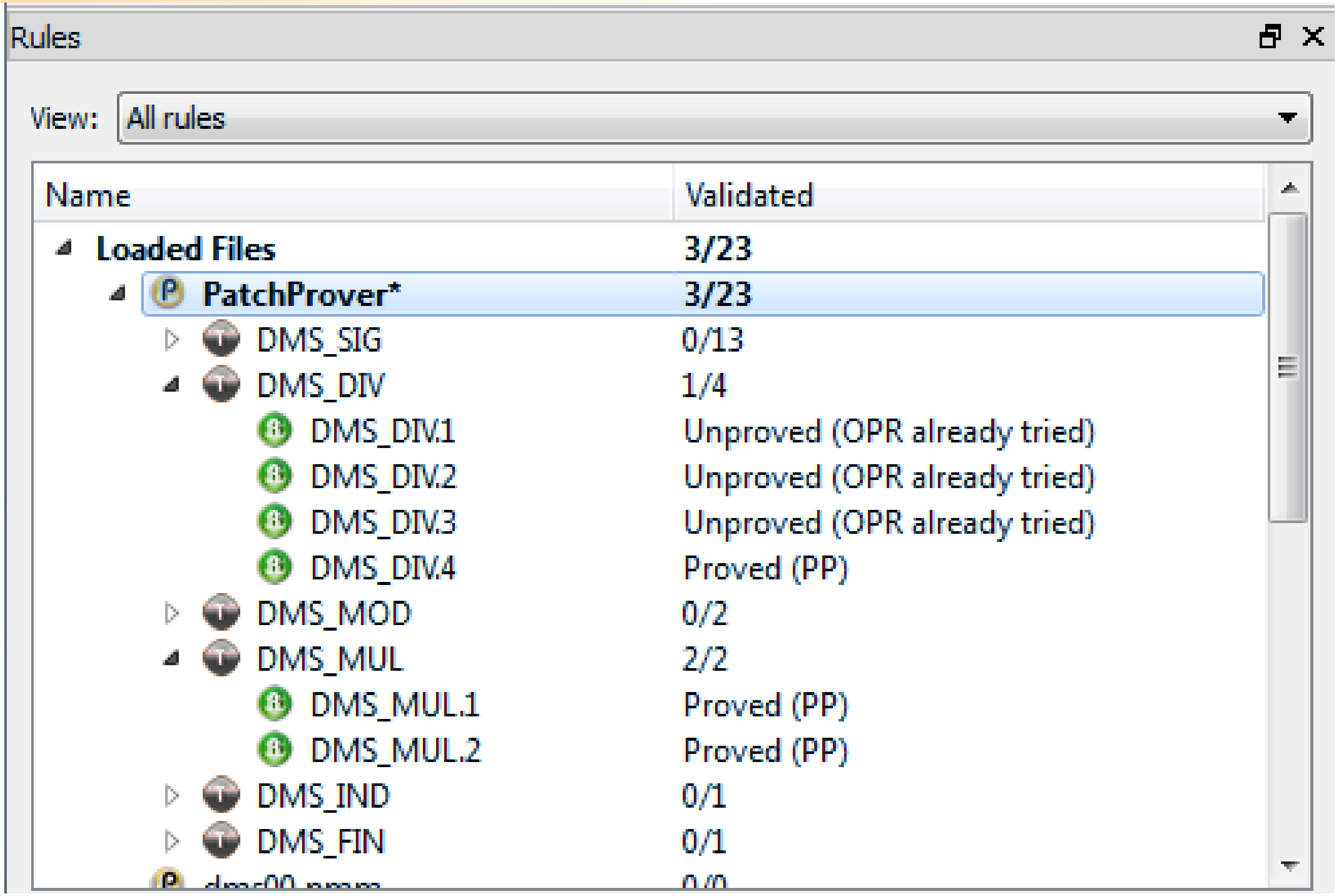
$\sum_x P(E) = \sum_y Q(F)$ if

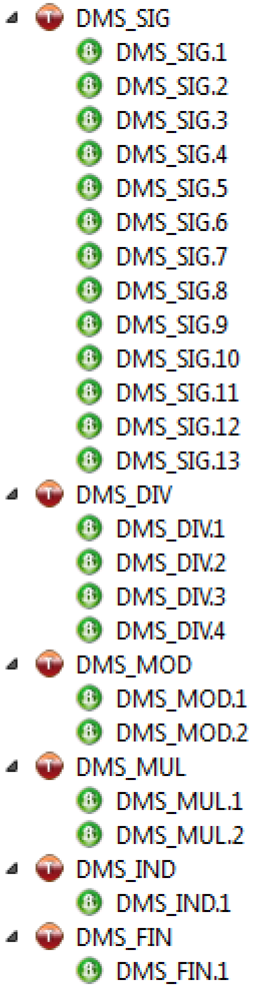
- $P(x)=Q(y)$ if x is replaced by y in $P(x)$
- $E(x)=F(y)$ if x is replaced by y in $E(x)$
- x is free in Q and F, y is free in P and E

CLEARSY
SYSTEMS ENGINEERING

Proof System

Proof Algorithm





CLEARSY
SYSTEMS ENGINEERING

Proof System

≡ Proof Algorithm

The resulting proof tree: 136 steps

```
Force(0)
  dd
    ah((0..morrow-1<|Schedule)~[{task}] = (0..clock0-1<|Schedule)~[{task}]\/({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(spans0\/{clock0|->time})<|(log0\/{morrow|->victor}))~[{task}] = (dom(spans0)<|log0)~[{task}]\/({clock0}<|log0)~[{task}])
              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(spans0\/{clock0|->time})<|(log0\/{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}] | (spans0\/{clock0|->time})(time$0)-time$0) = SIGMA(time$0).(time$0: (dom(spans0)<|log0)~[{task}] | spans0(time$0)...
                      ar(DMS_SIG.3,Once)
                        mp
                      dd
                        eh(SIGMA(time$0).(time$0: (dom(spans0)<|log0)~[{task}] | (spans0\/{clock0|->time})(time$0)-time$0),_h,Goal)
                          ah(SIGMA(time$0).(time$0: ({clock0}<|log0)~[{task}] | (spans0\/{clock0|->time})(time$0)-time$0) = SIGMA(time$0).(time$0: ({clock0}<|log0)~[{task}] | time-time$0))
                            ar(DMS_SIG.3,Once)
                              ah(dom(spans0) = dom(log0)-{clock0})
                                ah(spans0: INTEGER <-> INTEGER)
                                  pp(rt.0)
                            dd
                              eh(SIGMA(time$0).(time$0: ({clock0}<|log0)~[{task}] | (spans0\/{clock0|->time})(time$0)-time$0),_h,Goal)
                                ah(SIGMA(time$0).(time$0: ({clock0}<|log0)~[{task}] | time-time$0)+(term0<+({Phantom}<<|{elected0|->leftspan}\/(Schedule[{time}]<|Deadline)))(task) ...
                                  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}\/(Schedule[{time}]<|Deadline)))(task) = leftspan)
                                                ah(task: dom({Phantom}<<|{task|->leftspan}\/(Schedule[{time}]<|Deadline)))
                                                  mp
                                                  dd
                                                    ar(SimplifyRelFonXY.8,Goal)
                                                      ah(task: dom({Phantom}<<|{task|->leftspan}))
                                                        mp
                                                        dd
                                                          ar(SimplifyRelFonXY.14,Goal)
                                                            ah(not(task = Phantom))
                                                              mp
                                                              pp(rp.0)
                                                dd
                                                  eh((term0<+({Phantom}<<|{task|->leftspan}\/(Schedule[{time}]<|Deadline)))(task),_h,Goal)
                                                    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}]] = {}) => term0[Schedule[{time}]] = {0})
                                                          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)
```

```
dd
  dc(task: Schedule[{time}])
    dd
      ah(task: dom({Phantom}<<|{elected0|->leftspan}\/(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)
```

```
        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}\/(Schedule[{time}]...
          ph(task,!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
SYSTEMS ENGINEERING

Proof System

- Zoom on the PT

```
◢ dd
  ◢ dc(task: Schedule[{time}])                                        Split case
    ◢ dd
      ◢ ah(task: dom({Phantom}<<|{elected0|->leftspan}\/(Schedule[{time}]<|Deadline)))
          mp                                                           Prove
        ◢ dd
          ◢ ar(SimplifyRelFonXY.8,Goal)                                Rewrite
            ◢ ah(task: dom(Schedule[{time}]<|Deadline))
                mp                                                     Prove
              ◢ dd
                ◢ ar(SimplifyRelFonXY.15,Goal)                         Rewrite
                  ◢ ar(SimplifyRelFonXY.3,Goal)
                    ◢ ah(({time}<|Schedule)~[{task}] = {time})
                      ◢ ah(task: Schedule[{time}])
                          pp(rp.0)                                     Prove
                      ◢ dd
                        ◢ eh(({time}<|Schedule)~[{task}],{time},Goal)  Rewrite
                          ◢ ar(SimplifyIntSIGXY.12,Goal)
                            ◢ ar(DMS_SIG.1,Goal)                       Apply lemma
                              ◢ ah(not(task = elected0))
                                ◢ eh(elected0,_h,Goal)                 Rewrite
                                    mp                                 Prove
                              ◢ ss
                                ◢ ah(not(term0[Schedule[{time}]] = {}) => term0[Schedule[{time}]] = {0})
                                  ◢ ah(task: Schedule[{time}])
                                    ◢ ah(ran(Schedule) = Tasks)
                                      ◢ ah(dom(term0) = Tasks)
                                          pp(rp.0)                     Prove
```

Proof System

Application: ATP

Automatic metro pilot (Beijing metro)

Used for generating Ada software

127 components (model, refinement, implementation)

65 000 proof obligations

98 % automatically proved (1300 to prove)

CLEARSY
SYSTEMS ENGINEERING

Proof System

≡ PO

Model: uevol_loc_output_2_i

Proof obligation: iterateOnBlock.58

```
        "`Local hypotheses'" &
        l_ii_found$2: t_bool &
        l_nextBlockLentgh$2: t_distance &
        l_b1$2: t_bool &
        currentBlock$1|->currentDirection$1: dom(sidb_nextBlock) &
        p_out_block$1: t_block &
        p_out_dir$1: t_direction &
        p_out_block$1|->p_out_dir$1 = sidb_nextBlock(currentBlock$1|->currentDirection$1) &
        ii_translation$1<=0 &
        ii_computed$1 = FALSE => loc_ext1Abs$2 = {c_up|-
>sgd_blockLength(currentBlock$1)+ii_translation$1,c_down|-> -ii_translation$1}(currentDirection$1) &
loc_ext1Dir$2 = currentDirection$1 & loc_ext1Block$2 = currentBlock$1 & ii_computed$2 = TRUE &
        ii_computed$1 = TRUE => loc_ext1Abs$2 = loc_ext1Abs$1 & loc_ext1Dir$2 = loc_ext1Dir$1 &
loc_ext1Block$2 = loc_ext1Block$1 & ii_computed$2 = ii_computed$1 &
        jj_computed$1 = FALSE => loc_int2Abs$2 = {c_up|-
>sgd_blockLength(currentBlock$1)+jj_translation$1,c_down|-> -jj_translation$1}(currentDirection$1) &
(loc_int2Dir$2: {c_up,c_down} & not(loc_int2Dir$2 = currentDirection$1)) & loc_int2Block$2 = currentBlock$1
& jj_computed$2 = TRUE &
        jj_computed$1 = TRUE => loc_int2Abs$2 = loc_int2Abs$1 & loc_int2Dir$2 = loc_int2Dir$1 &
loc_int2Block$2 = loc_int2Block$1 & jj_computed$2 = jj_computed$1 &
        kk_computed$1 = FALSE => loc_int1Abs$2 = {c_up|-
>sgd_blockLength(currentBlock$1)+kk_translation$1,c_down|-> -kk_translation$1}(currentDirection$1) &
loc_int1Dir$2 = currentDirection$1 & loc_int1Block$2 = currentBlock$1 & kk_computed$2 = TRUE &
        kk_computed$1 = TRUE => loc_int1Abs$2 = loc_int1Abs$1 & loc_int1Dir$2 = loc_int1Dir$1 &
loc_int1Block$2 = loc_int1Block$1 & kk_computed$2 = kk_computed$1 &
        "`Check that the invariant (loc_trainLocated = loc_trainLocated$1) is preserved by the operation -
ref 4.4, 5.5'"
=>
        loc_ext1Abs$2: t_distance
```

CLEARSY
SYSTEMS ENGINEERING

≡ Proof Algorithm

iterateOnBlock.58

« Size does matter »

Demonstrate that locAbsExt$2 is implementable 32-bit integer

34 local hypotheses

1380 hypotheses

Anticipating thousands steps demonstration …

Proof System

Proof Algorithm

```
mp(Tac(simp))                                  Apply lemma
  dc(1<=jj_translation$1)                      Split case
      mp
      mp                                       Prove
        ah(jj_translation$1<=0)
          pr
            pr                                 Prove
  pr
     dc(1<=kk_translation$1)                   Split case
        pr
        pr                                     Prove
  pr
     ah(kk_translation$1<=0)
        pr                                     Prove
```

CLEARSY
SYSTEMS ENGINEERING

Proof System

Metrics

Up to 2500 hypotheses in the middle of the proof

1800 added rules

800 rules in the Patchprover (32%)

30 tactics and 200 demonstrations to demonstrate the whole projet

Proof System

Application: MPU

Event B model of a smart card electronic device

Used for VHDL generation

18 levels of refinement

40% automatic proof

Proof System

Model: mpu_017

Proof obligation: psi.1

≡ Metrics

```
        "`Local hypotheses'" &
        ee7$2 = {xe | xe: eb7$2 & sc7$1(xe): {c0$1,ClPb}} &
        m0$1 = 1 &
        ea7$2 = {xa | xa: SEs & sm10$1(xa) = TRUE & (ssh13$1><ssm13$1><ssl13$1;hmln)(xa)|-
>hash(a0$1): heq & hash(a0$1)|->(seh13$1><sem13$1><sel13$1;hmln)(xa): heq} &
        eb7$2 = {xb | xb: ea7$2 & t0$1: st7$1[{xb}]} &
        ec7$2 = sc7$1[eb7$2] &
        ed7$2 = {xd | xd: eb7$2 & a0$1: hate[{(ssh13$1><ssm13$1><ssl13$1;hmln)(xd)}]} &
        "`Check that the invariant (ea7 = ea7$1) is preserved by the operation - ref 4.4, 5.5'"
=>
        ea7$2 =
sm10$1~[{TRUE}]/\(ssh13$1><ssm13$1><ssl13$1;hmln;heq;hash~)~[{a0$1}]/\(seh13$1><sem13$1><sel13$1
;hmln;heq~;hash~)~[{a0$1}]
```

To demonstrate that ea7$2 …. hmmmm …. points to the correct memory cell

Proof System

≡ Proof Tree

- dd
  - eh(ea7$2)
    - eh(ssh13$1><ssm13$1><ssl13$1;hmln)
      - eh(seh13$1><sem13$1><sel13$1;hmln)
        - ah(dom(hash) = ADs)
          - ah(hash: ADs +-> NBs)
            - ah(a0$1: ADs)
              - ah(heq: NBs <-> NBs)
                - ah(ssn12: SEs +-> NBs)
                  - ah(dom(ssn12) = SEs)
                    - ah(sen12: SEs +-> NBs)
                      - ah(dom(sen12) = SEs)
                        - ah(sm10$1: SEs +-> BOOL)
                          - ah(dom(sm10$1) = SEs)
                            - p0

Rewrite

Prove

Proof System

≡ Metrics

20 tactics

No added rule !

1 000 proof obligations in total

Proof System

A Real Failure

ATP model including a constant representing clock ticks over time (function: $\mathbb{N} \rightarrow$ BOOL)

Specified by its properties:

C $\in$ {C $\in$ $\wedge$ C(m+118)=FALSE $\wedge$ C(m119)=TRUE $\wedge$

C(m+120)=FALSE $\wedge$ C(m+121)=TRUE $\wedge$ C(m+122)=TRUE $\wedge$

C(m+123)=FALSE $\wedge$ C(m+124)=TRUE $\wedge$ C(m+124)=FALSE$\wedge$

C(m+125)=FALSE $\wedge$ C(m+126)=TRUE $\wedge$ C(m+127)=TRUE $\wedge$

….}

Proof System

≡ Metrics

In B, constants needs to be non-miracle

E.g: values should be given in implementation and prove to comply with properties

For this infinite function, we decided to go for an admission rule and a paper demonstration

I wrote the paper demonstration, cross-read by 2 other « experts »

Proof System

Metrics

$$C \in \{C \in \wedge\ C(m+118)=FALSE \wedge C(m119)=TRUE \wedge$$
$$C(m+120)=FALSE \wedge C(m+121)=TRUE \wedge\ C(m+122)=TRUE \wedge$$
$$C(m+123)=FALSE \wedge C(m+124)=TRUE \wedge\ C(m+124)=FALSE \wedge$$
$$C(m+125)=FALSE \wedge C(m+126)=TRUE \wedge\ C(m+127)=TRUE \wedge$$
$$....\}$$

- Exploit:
  - add trivialhypothesis: C(m+124)= C(m+124)
  - Replace C(m+124) by its values: TRUE = FALSE
  - You can prove the project with this property

- Detected by independent assessor

The Atelier B Proof System and Its Improvements

- Intro to B method
- Proof System
- **Improvements**

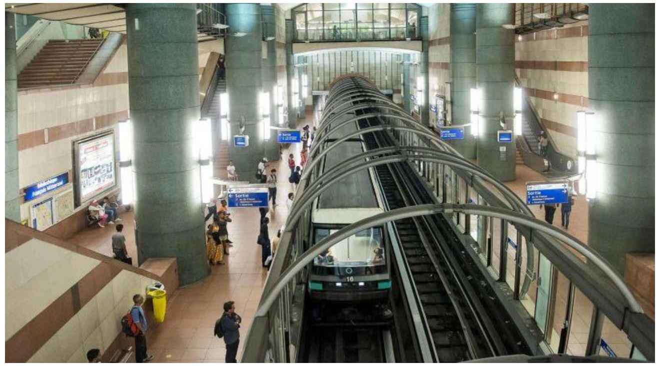
Improvements

Improvements

Stuck in 1998

- **Core Prover (mecanisms + rule) has stopped its evolution in 1998**

- **No proof regression on existing projects**
  - Safety-critical software need functional updates
  - Modifications in the Core => demonstrations failing to prove
  - 1 PO == 35 € (16 PO per day, 500 € per engineer day)
  - Noone is willing to pay thousands €



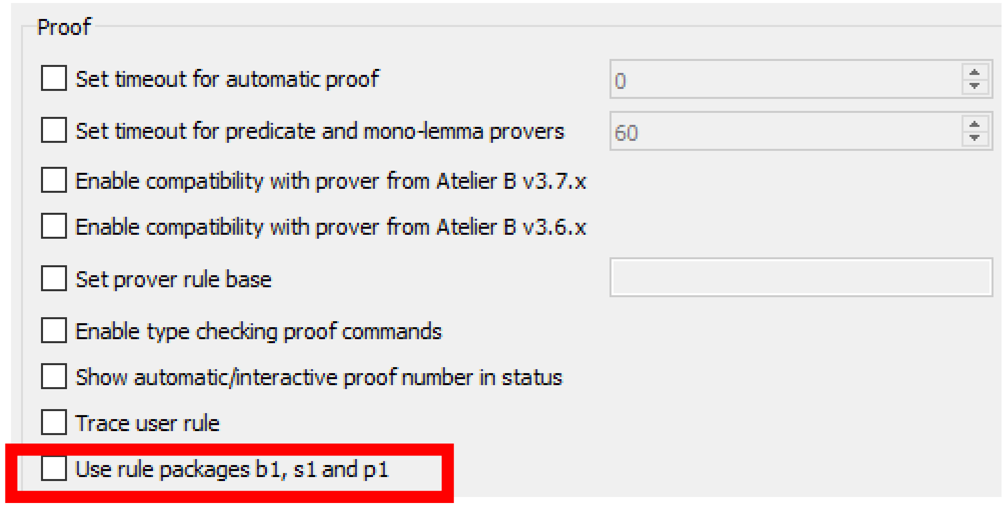
Meteor line 14 released in Dec 1998

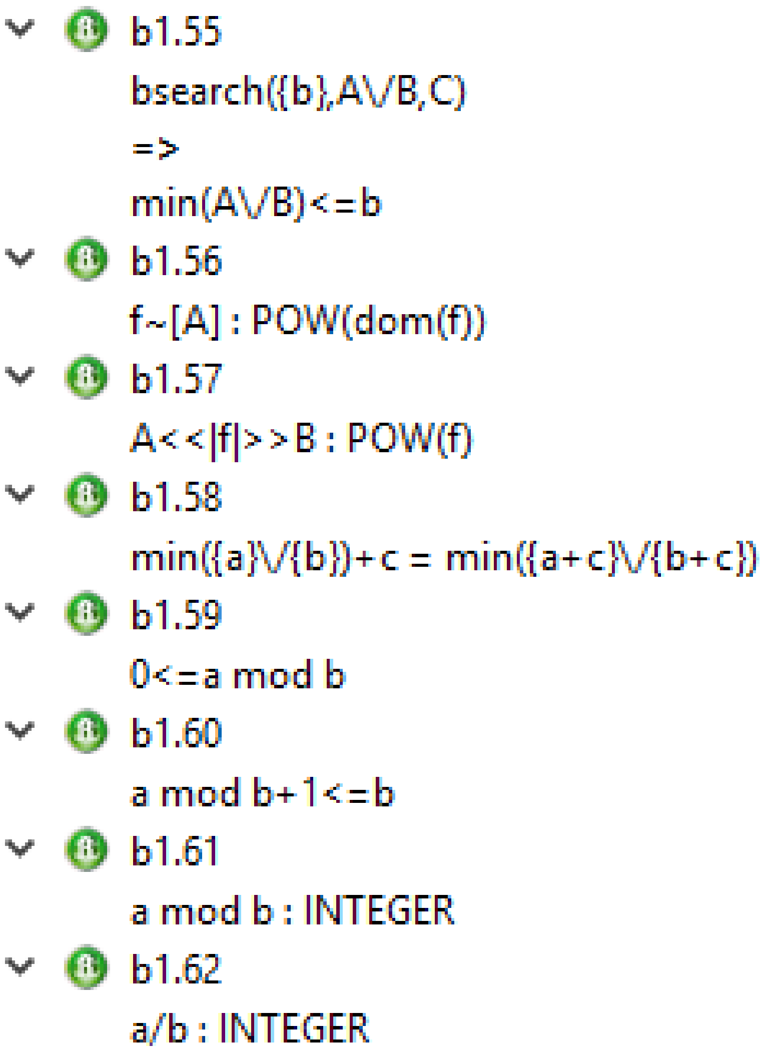
- **Peripheral evolutions**
  - New proof commands
  - New additional rules packages
  - Connecting other provers
  - Proof servers, maximizing cores usage

CLEARSY
SYSTEMS ENGINEERING

Improvements

Additional Rules Package

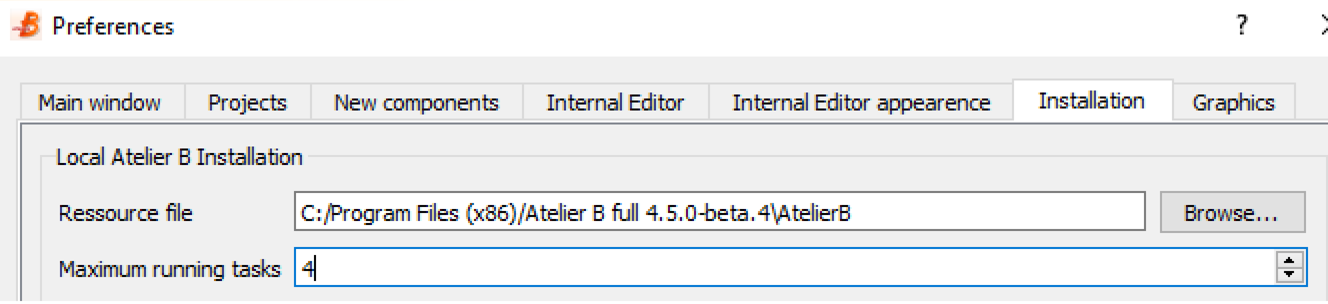




Improvements

Using Cores

Automatic proof is a quick process most of the time, especially as you can distribute automatic proof on all your cores



Component	Action	Status	Messages	Server
dcg_donnees_i		Running	clause refinement_of_get_v_ComM1__MessageRecu_Replica1_uint32 - Proved 37, Unproved 0, Tried 37/128,...	localhost-1
dcg_message_coeur_dis...		Running	clause PrepMessage_CoeurDistant_Traitement - Proved 5, Unproved 0, Tried 5/7, Estimated end at 13:41:23	localhost-3
dcg_operateurs		Running	clause WellDefinednessProperties - Proved 7, Unproved 0, Tried 7/18, Estimated end at 13:41:23	localhost-4
dcg_phases		Running	End of Proof	localhost-2
dcg_phases_i		Waiting		
dcg_phases_r		Waiting		
dcg_projet		Waiting		
dcg_public		Waiting		
dcg_public_i		Waiting		
dcg_registres		Waiting		
dcg_type		Waiting		
dcg_util		Waiting		
dcg_util_i		Waiting		
dcg_verif_ALU		Waiting		

CLEARSY
SYSTEMS ENGINEERING

More information

Lecture 15: Loops

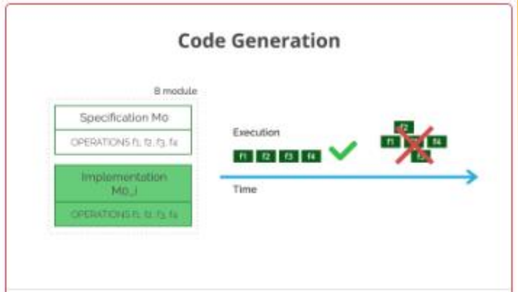
This video presents how the B-Method provides support to loops, an essential programming construct.

Level: Basic Video duration: 31:42

Lecture 16: Structuring

This video details the different ways of structuring a B project in order to lower the complexity of the modelling and to ease the

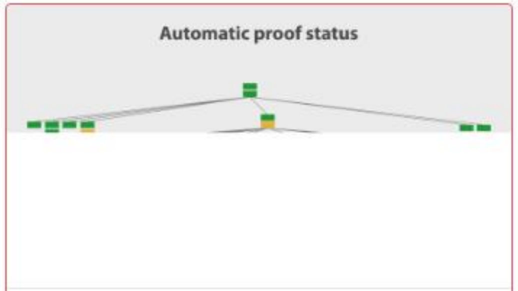
Level: Basic Video duration: 07:35



Lecture 17: Code Generation

This video show how a B model is transformed into C code and which constraints have to be met to be successful.

Level: Basic Video duration: 12:22



Lecture 18: Introduction to Proofs

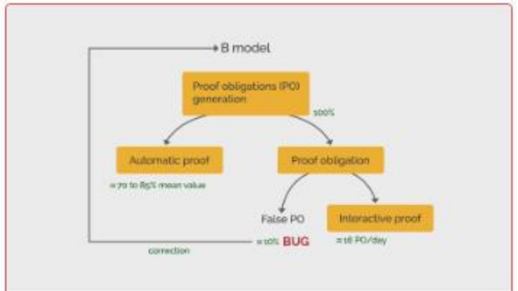
The video explains what proving a software against its specification means, what automatic proof is, and introduces interactive

Level: Basic Video duration: 13:28

Lecture 19: Proofs

This video explains how to improve automatic proof performances and provides some hints about the relation between modelling and

Level: Basic Video duration: 09:49



Lecture 20: Managing Projects

This video describes the B development cycle, provides metrics and explains how to reduce the complexity and to simplify the

Level: Basic Video duration: 07:55

MOOC
massive open online course

https://mooc.imd.ufrn.br/

UFRN UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

IMD INSTITUTO METRÓPOLE DIGITAL

CLEARSY SYSTEMS ENGINEERING

Thank you for your attention

Salvador , November 26th 2018

Thierry Lecomte
R&D Director, ClearSy

thierry.lecomte@clearsy.com

CLEARSY
SYSTEMS ENGINEERING