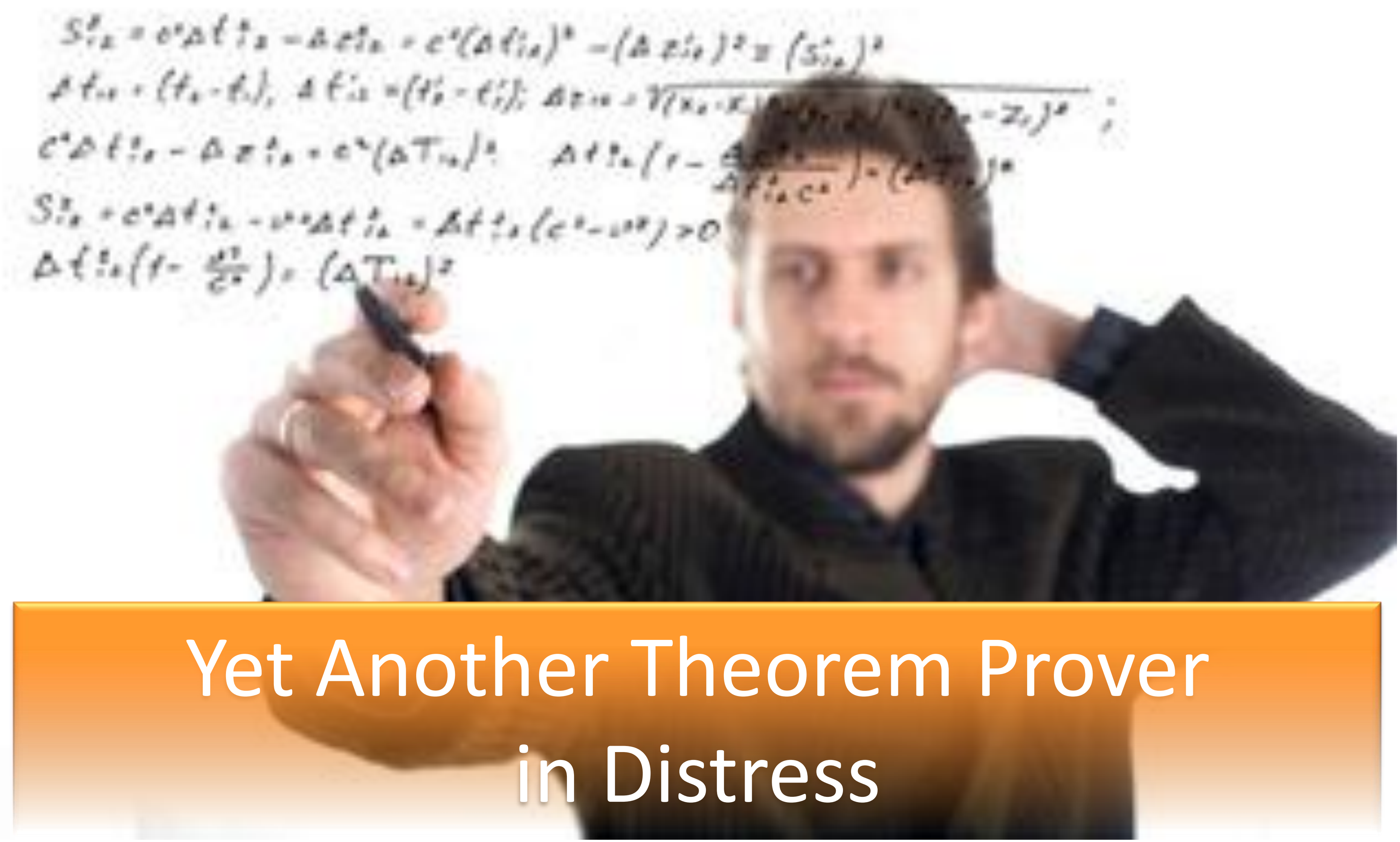


Yet Another Theorem Prover in Distress

Thierry Lecomte
ClearSy
thierry.lecomte@clearsy.com

Statements

- Modelling and proof problems are tightly linked
- In my talk, only proof is questionnable
- Some experiences are reported:
  - Proof of (large) B models
  - Using several theorem provers

Proof work

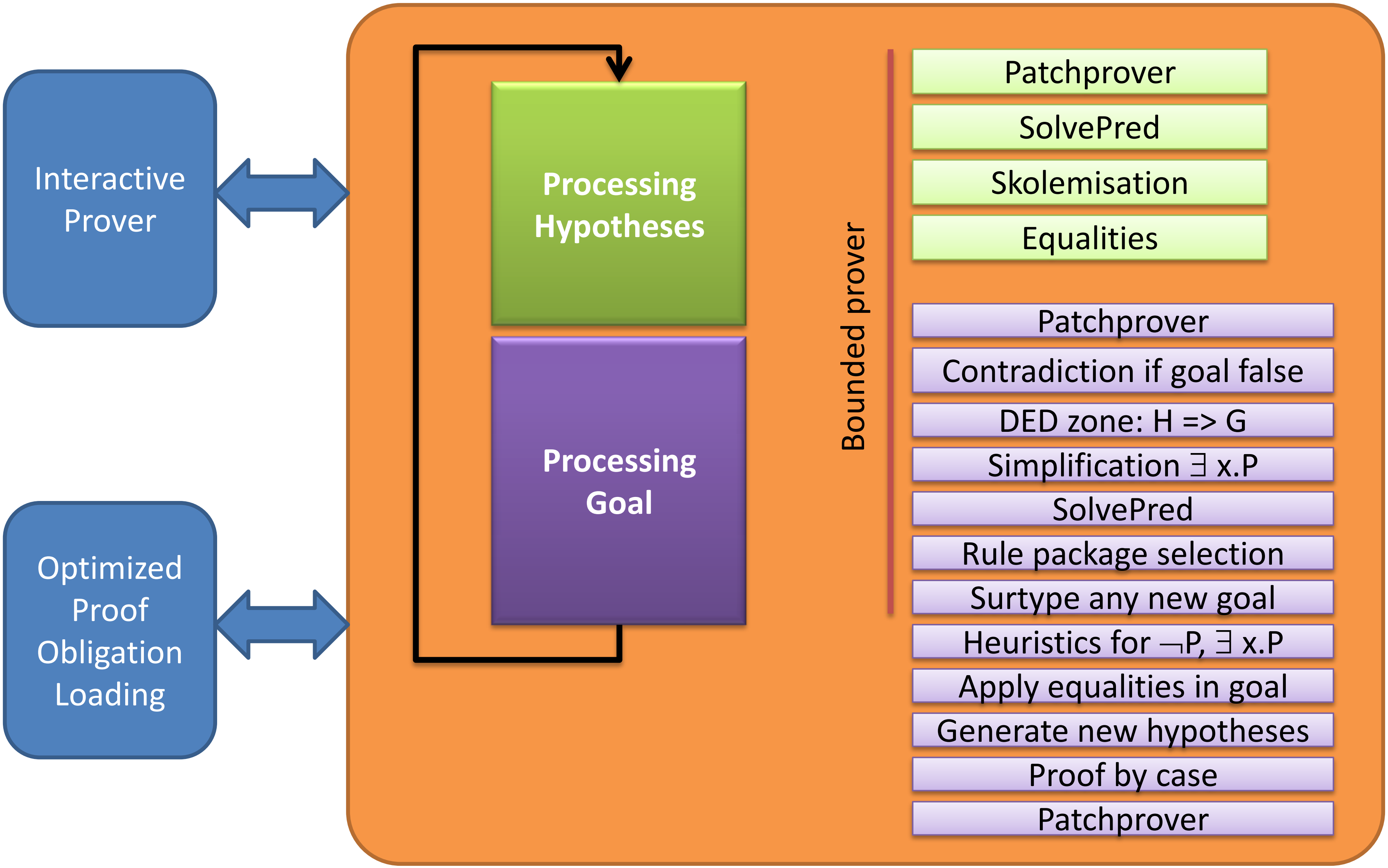
- Several kinds of modellings (Event-B/B)
- Several tools with different application domains (the « how »)
- Several ways of adding proof information:
  - Assertions (model)
  - Mathematical rules
  - (Generic) demonstrations

- Successful proof: choreography

Provers / solvers

- Main prover
  - Top-down: applications of simplification mechanisms and mathematical rules, triggered by hypotheses
  - Bottom-up: generation of new hypotheses (combination) in relation with the goal or with hypotheses in relation with the goal ….

Main prover architecture



Main prover architecture

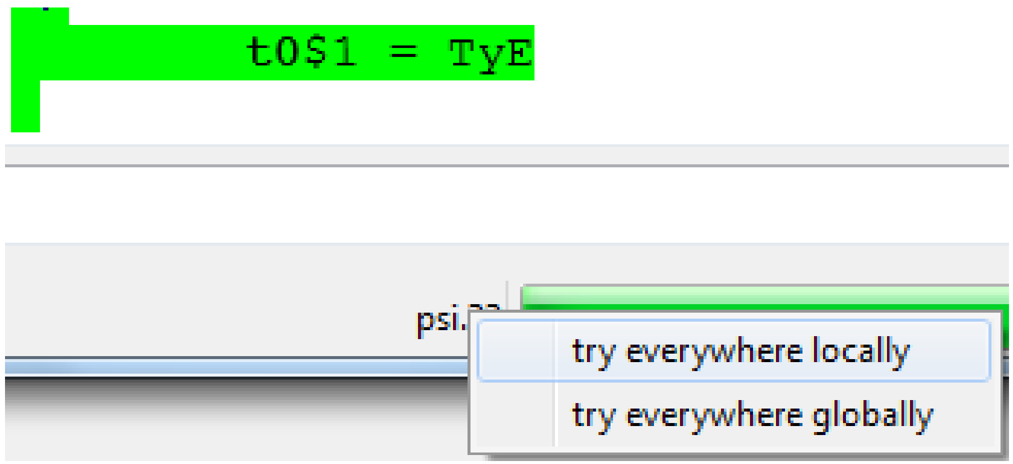
Rules and tactics

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

Operation filter | Goal filter | Interactive commands

ff(0) & ah(Mhyp(P => Sgoal(H => G | G))) & p0

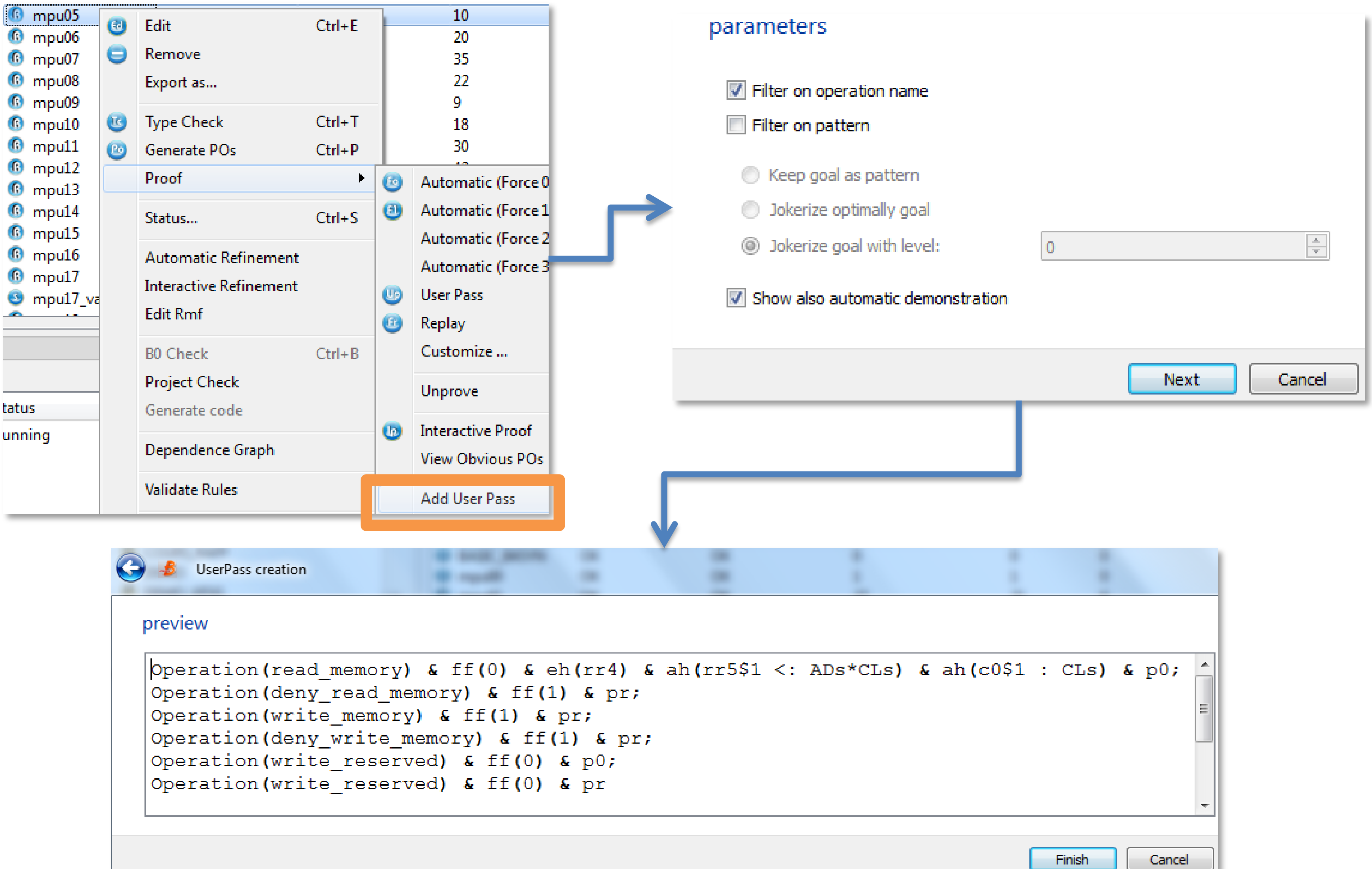
How come an interactive demonstration becomes a tactic ?



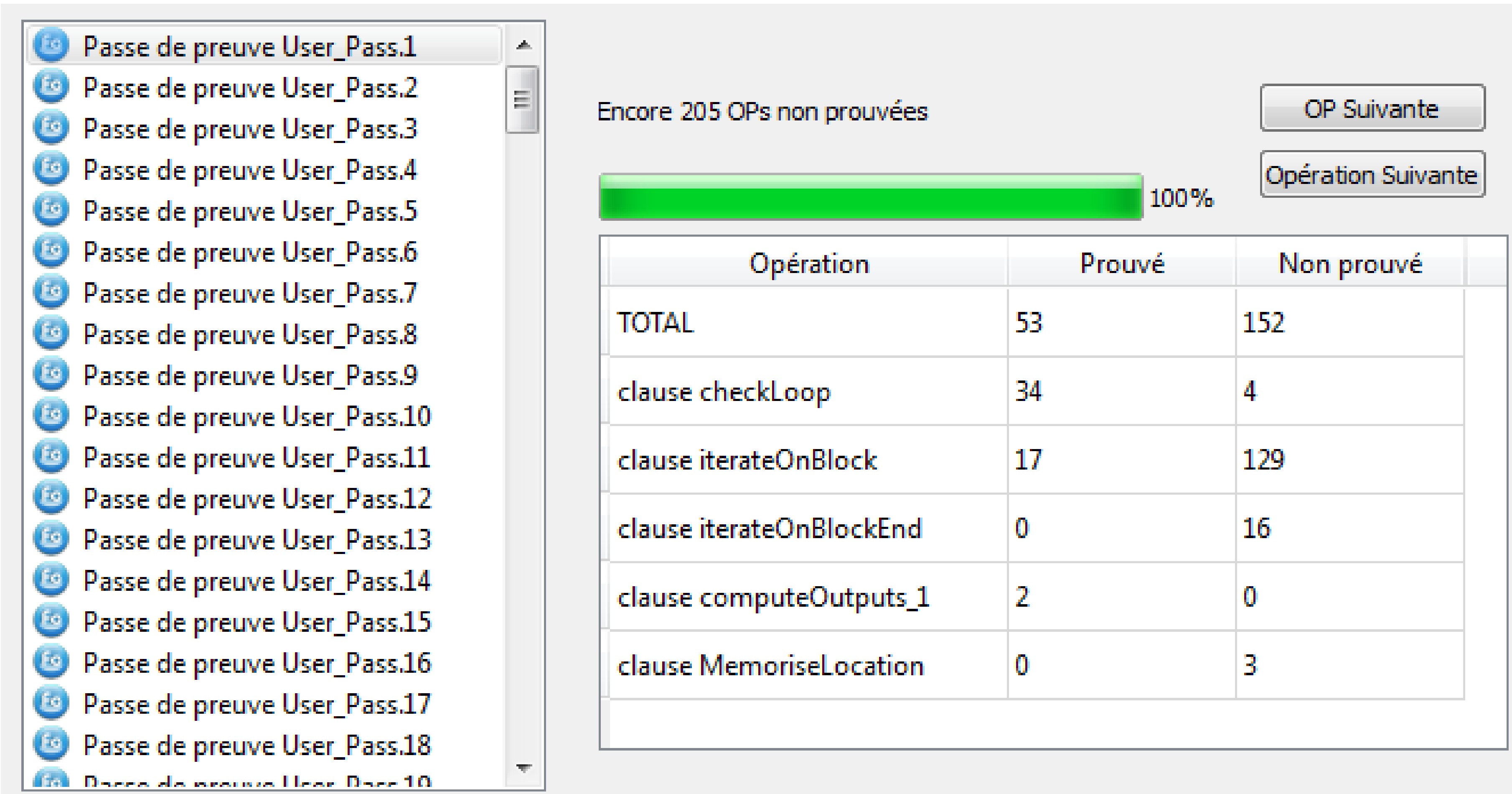
- Replayable as it is
- Or without minor modification
- Or by abstracting parameters

- Bottom-up: generic (abstract) demo is not search first

From demo to tactic



How efficient are tactics ?

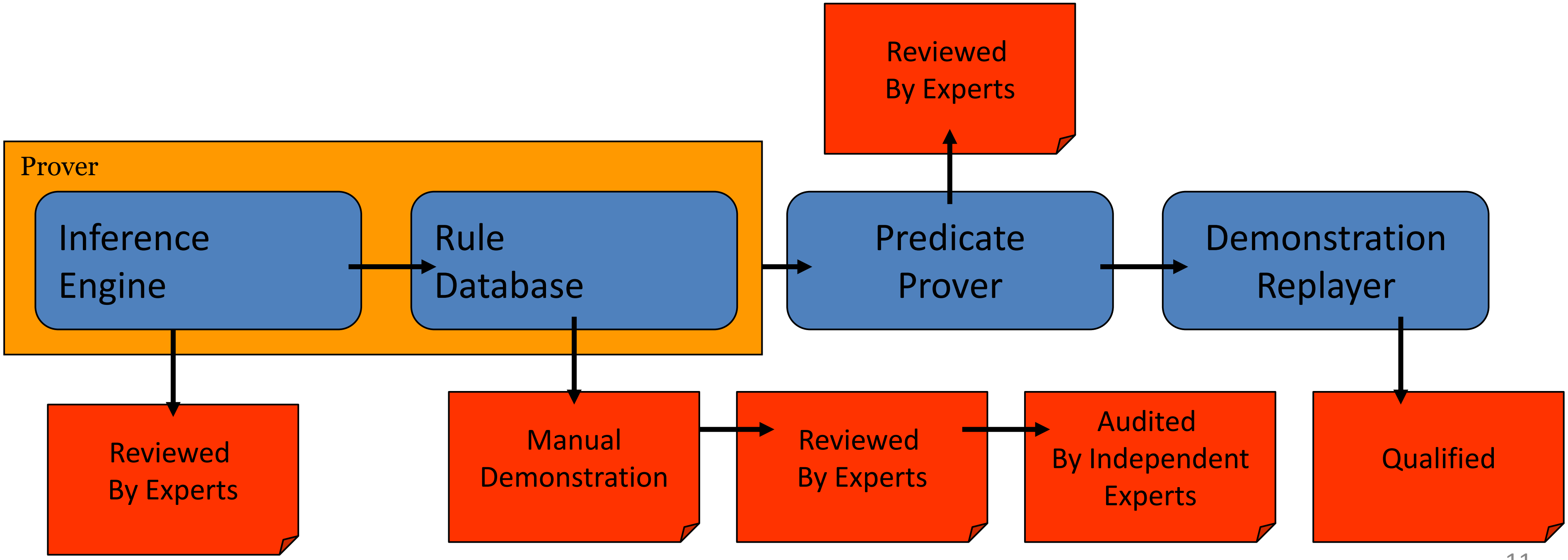


Opération	Prouvé	Non prouvé
TOTAL	53	152
clause checkLoop	34	4
clause iterateOnBlock	17	129
clause iterateOnBlockEnd	0	16
clause computeOutputs_1	2	0
clause MemoriseLocation	0	3

Provers / solvers

- Main prover
  - Top-down: applications of simplification mechanisms and mathematical rules, triggered by hypotheses
  - Bottom-up: generation of new hypotheses (combination) in relation with the goal or with hypotheses in relation with the goal ….
- Predicate prover
  - tableaux method
  - Initialy developed to validate main prover math. rules

About prover qualification



Provers / solvers

- Main prover
  - Top-down: applications of simplification mechanisms and mathematical rules, triggered by hypotheses
  - Bottom-up: generation of new hypotheses (combination) in relation with the goal or with hypotheses in relation with the goal ….
- Predicate prover
  - tableaux method
  - Initialy developed to validate main prover math. rules
- Arithmetic prover
  - linear equations
- Set solver

Application domains

the « how » alongside the « why »

- Main prover:
  - The more versatile
  - Rule database (2600 rules) developed through domain specific projects
- Predicate prover:
  - Powerful when applied to propositional logic
  - Requires few hypotheses to work efficiently
- Arithmetic prover
  - Only with pure linear equations
- Set solver
  - Efficient to simplify set based expressions

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

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

Model: dms00

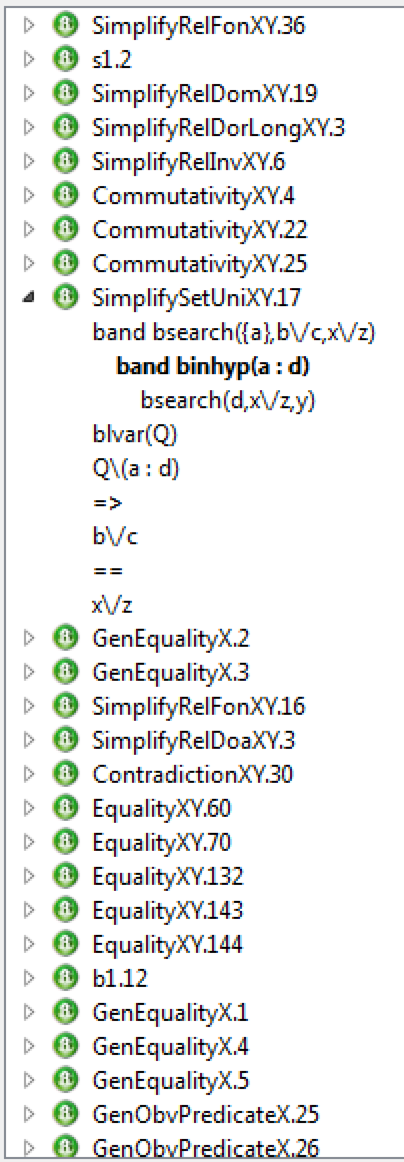
Proof obligation: Swap.21

```
        "`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)
```

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



Additions

- 23 rules added to the whole project

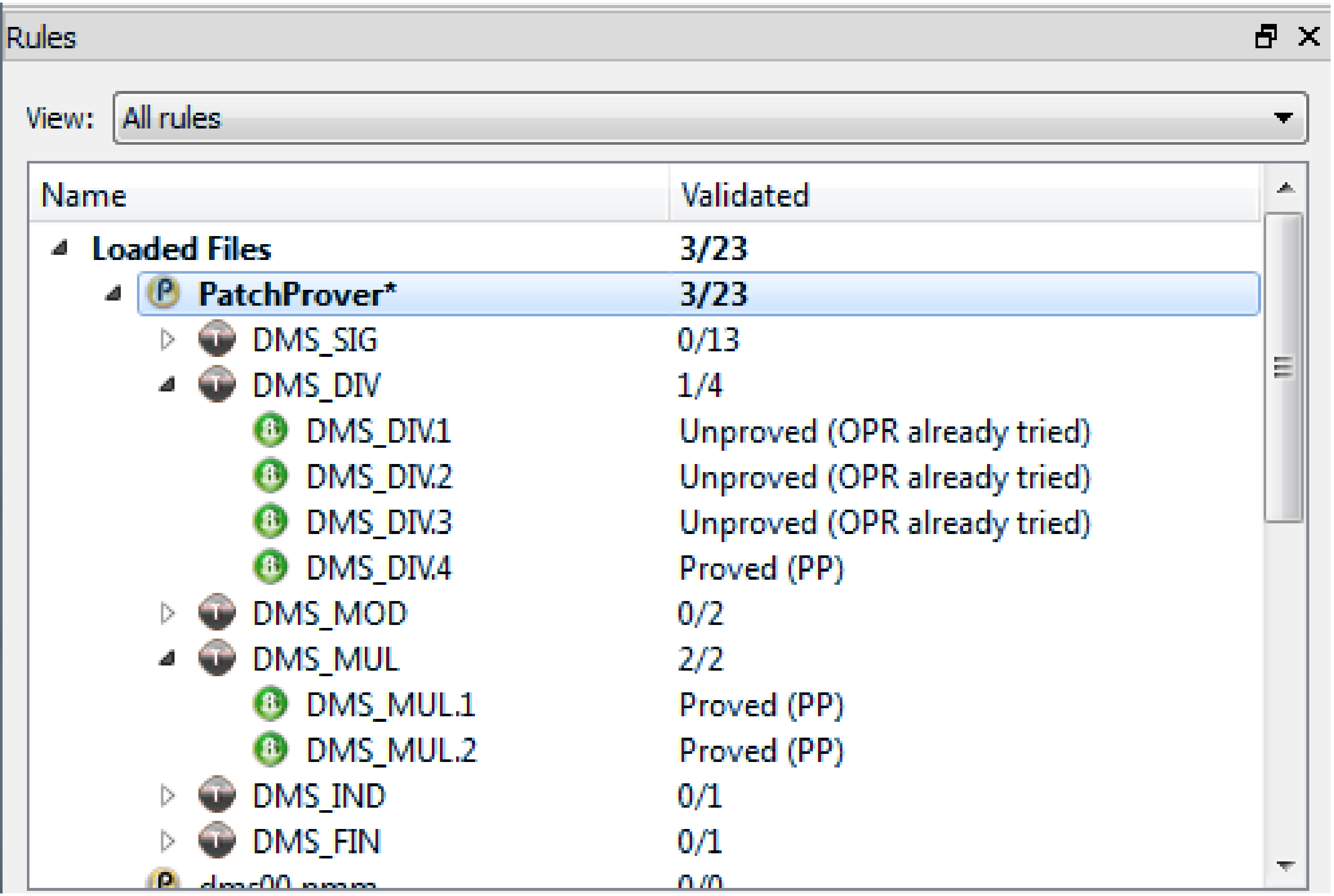
- DMS_SIG
  - DMS_SIG.1
  - DMS_SIG.2
  - DMS_SIG.3
  - DMS_SIG.4
  - DMS_SIG.5
  - DMS_SIG.6
  - DMS_SIG.7
  - DMS_SIG.8
  - DMS_SIG.9
  - DMS_SIG.10
  - DMS_SIG.11
  - DMS_SIG.12
  - DMS_SIG.13
- DMS_DIV
  - DMS_DIV.1
  - DMS_DIV.2
  - DMS_DIV.3
  - DMS_DIV.4
- DMS_MOD
  - DMS_MOD.1
  - DMS_MOD.2
-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

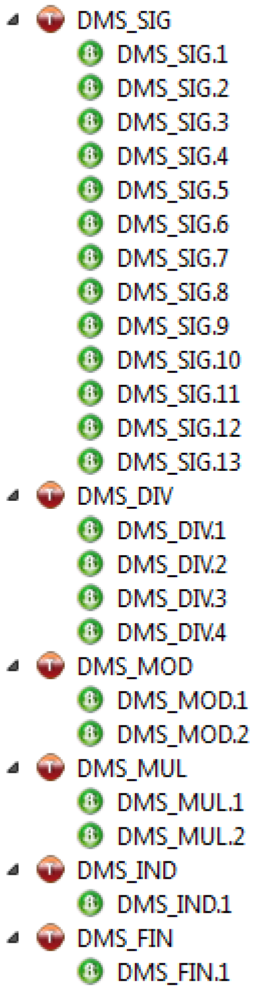
```
/* DMS_SIG.5 */
   bmatch(x,P,Q,y) &
   bmatch(x,E,F,y) &
   x\(Q,F) &
   y\(P,E)
=>
   SIGMA(x).(P|E) = SIGMA(y).(Q|F)
```

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

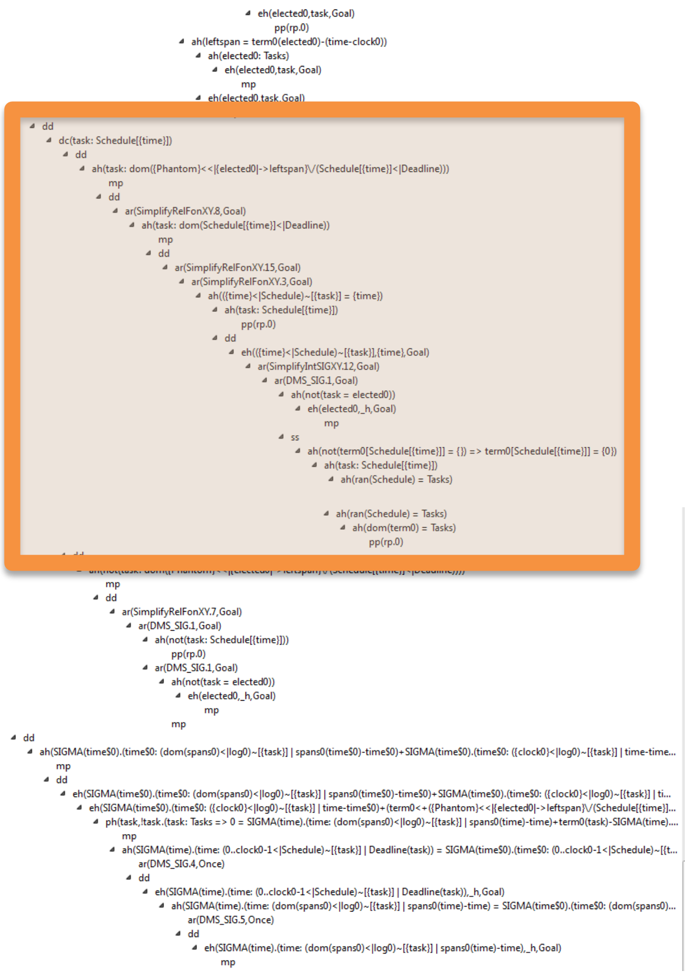
- $P(x)=Q(y)$ if x is replaced by y in $P(x)$
- $E(x)=F(y)$ if x is replaced by y in $E(x)$
- x is free in Q and F, y is free in P and E

Validating rules for new domains





The resulting proof tree: 136 steps



Zoom on the proof tree

```
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
```

Some metrics

Split case	2
Apply lemma	16

Rewrite	20
Prove	27
Sanitize	71

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)

Model: uevol_loc_output_2_i
Proof obligation: iterateOnBlock.58

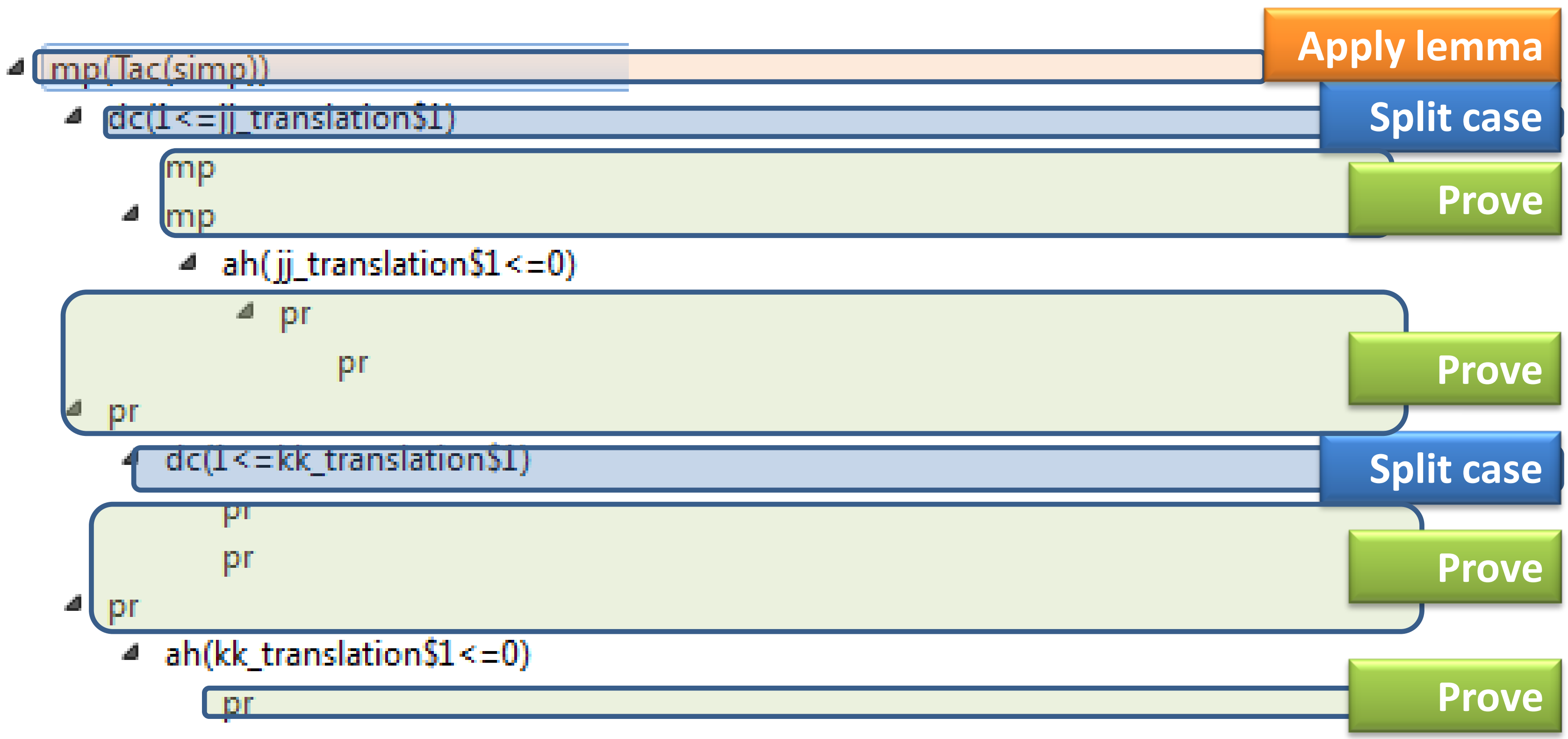
```
        "`Local hypotheses'" &
        l_ii_found$2: t_bool &
        l_nextBlockLentgh$2: t_distance &
        l_b1$2: t_bool &
        currentBlock$1|->currentDirection$1: dom(sidb_nextBlock) &
        p_out_block$1: t_block &
        p_out_dir$1: t_direction &
        p_out_block$1|->p_out_dir$1 = 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
```

iterateOnBlock.58
« Size does matter »

- Demonstrate that locAbsExt$2 is implementable 32-bit integer
- 34 local hypotheses
- 1380 hypotheses
- Anticipating thousands steps demonstration …

The proof tree

No need for zoom



Some 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

Application: MPU

- Event B model of a smart card electronic device
- Used for VHDL generation
- 18 levels of refinement

- 40% automatic proof

Model: mpu_017
Proof obligation: psi.1

```
        "`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 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

Some metrics

- 20 tactics
- No added rule !
- 1 000 proof obligations in total

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 $\wedge$

  ….}

A real failure … (cntd)

- 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 »

A real failure … (the end)

$C \in \{C \in \wedge$ 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

Conclusion & remarks

- « why » is deeply related to « how »
- Guru/experts required for first shot at new (kind of) modelling
- Transforming demos into tactics enables to save proof work
- When automatic proof fails, interactive proof almost requires to disengage all proof mechanisms and use the prover as a smart “calculette”

- Proof maintenance could be a nightmare
- Provers are almost stuck because of potential proof regressions