

$$\begin{aligned}
S_{12}^2 &= c^2 \Delta t_{12}^2 - \Delta x_{12}^2 = c^2 (\Delta t_{12})^2 - (\Delta x_{12})^2 \equiv (S_{12}^0)^2 \\
\Delta t_{12} &= (t_2 - t_1), \Delta t'_{12} = (t'_2 - t'_1); \Delta x_{12} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}; \\
c^2 \Delta t_{12}^2 - \Delta x_{12}^2 &= c^2 (\Delta T_{12})^2; \Delta t'_{12} \left( 1 - \frac{v^2}{c^2} \right) = (\Delta T_{12})^2 \\
S_{12}^2 &= c^2 \Delta t'_{12}^2 - v^2 \Delta t'_{12}^2 = \Delta t'_{12}^2 (c^2 - v^2) > 0 \\
\Delta t'_{12} \left( 1 - \frac{v^2}{c^2} \right) &= (\Delta T_{12})^2
\end{aligned}$$

# 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 questionable
- Some experiences are reported:
  - Proof of (large) B models
  - Using several theorem provers

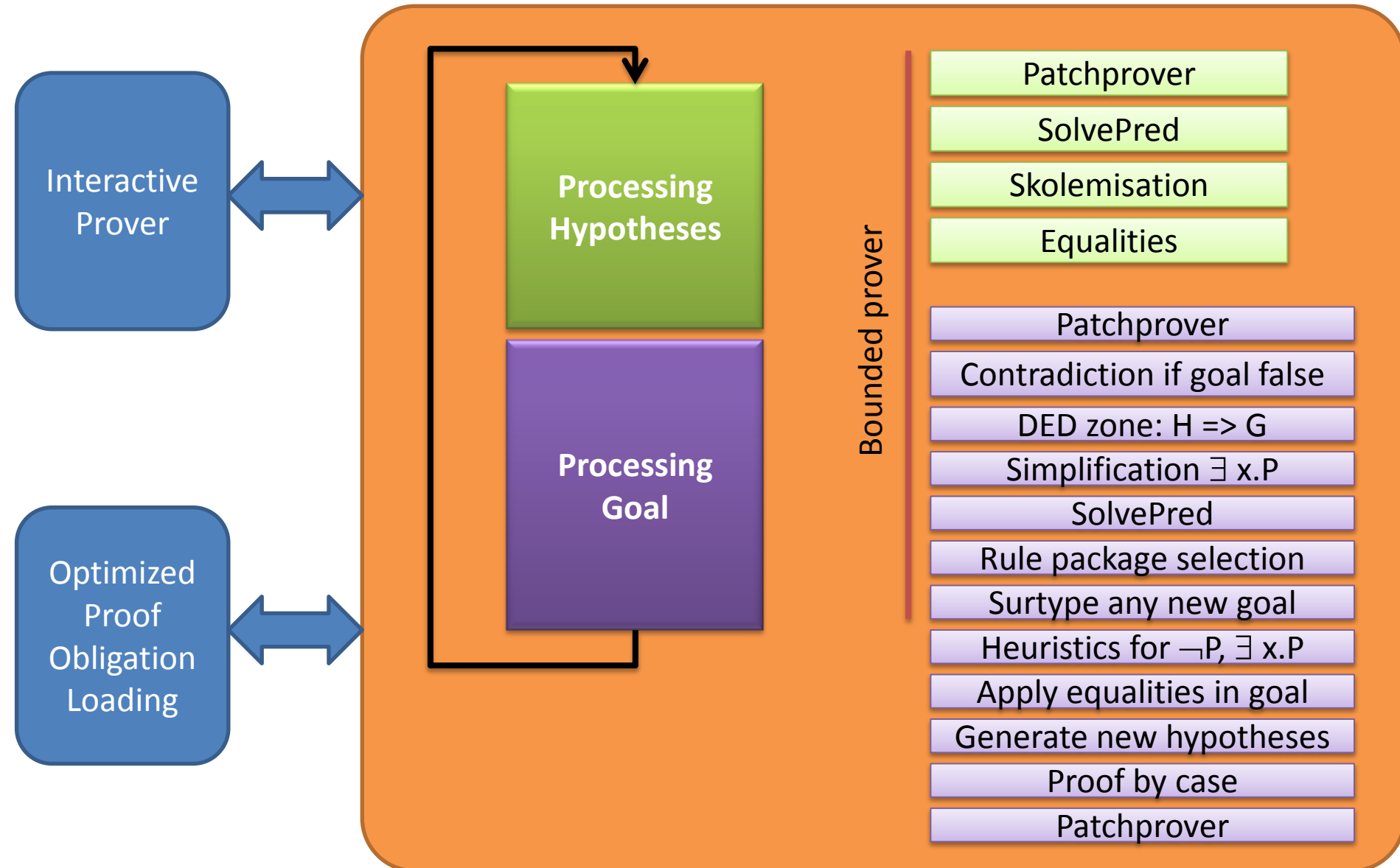
# Proof work

- 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

---

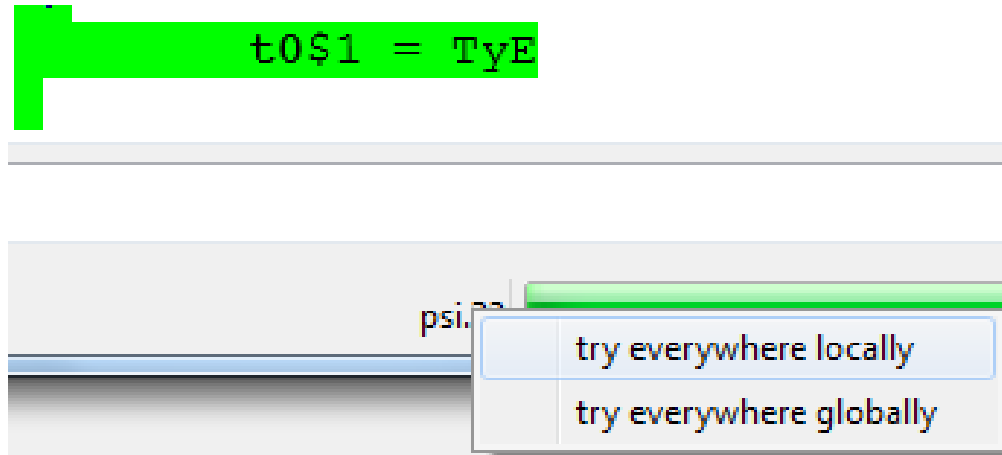
Goal filter

---

Interactive commands

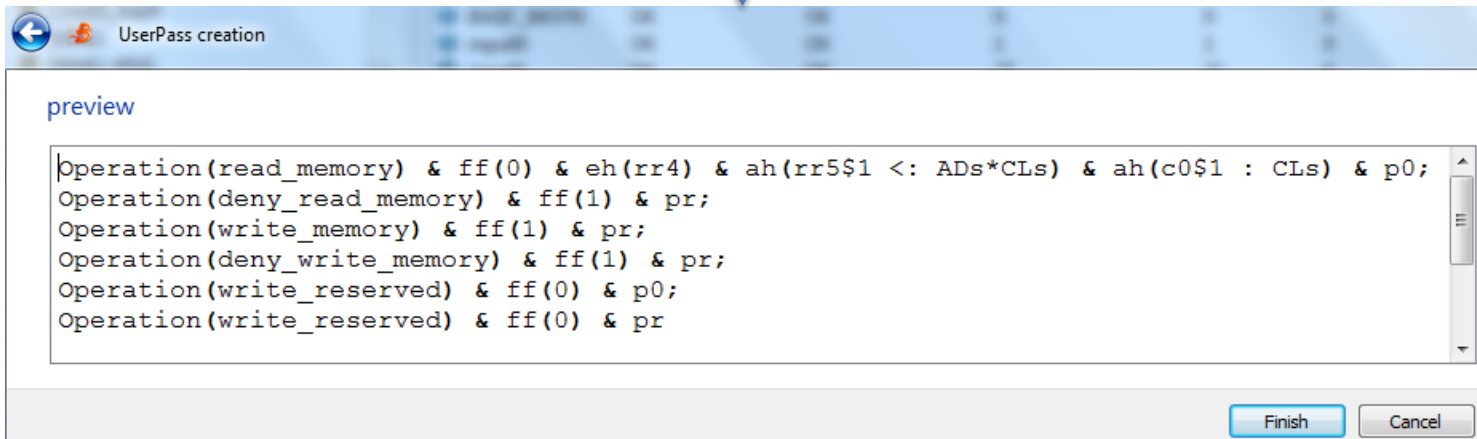
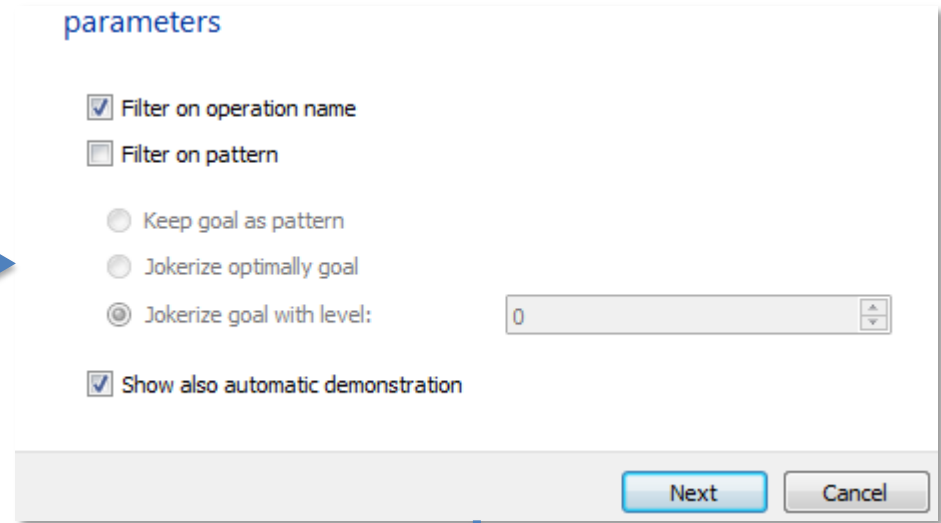
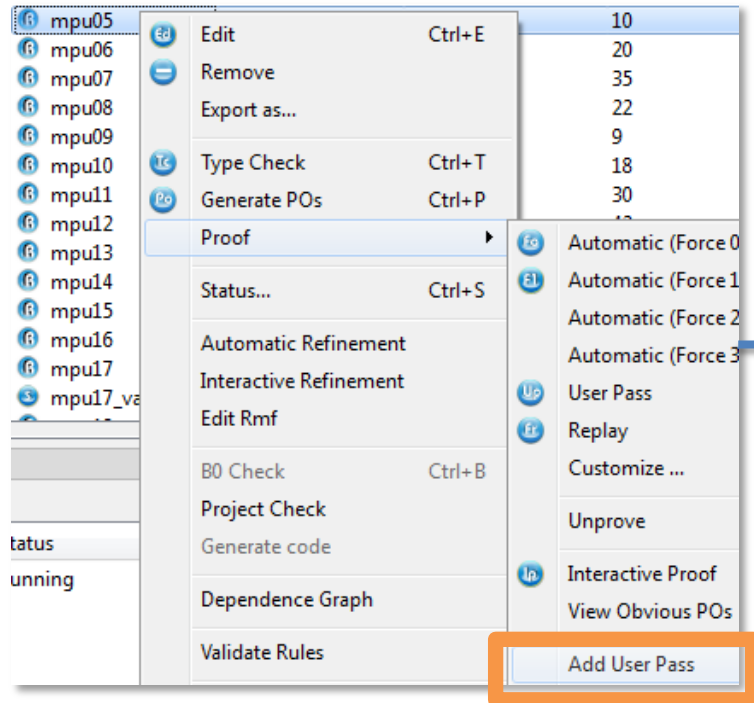
`ff(0) & ah(Mhyp(P => Sgoal(H => G | G))) & p0`

# 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 ?

Passe de preuve User\_Pass.1

Passe de preuve User\_Pass.2

Passe de preuve User\_Pass.3

Passe de preuve User\_Pass.4

Passe de preuve User\_Pass.5

Passe de preuve User\_Pass.6

Passe de preuve User\_Pass.7

Passe de preuve User\_Pass.8

Passe de preuve User\_Pass.9

Passe de preuve User\_Pass.10

Passe de preuve User\_Pass.11

Passe de preuve User\_Pass.12

Passe de preuve User\_Pass.13

Passe de preuve User\_Pass.14

Passe de preuve User\_Pass.15

Passe de preuve User\_Pass.16

Passe de preuve User\_Pass.17

Passe de preuve User\_Pass.18

Passe de preuve User\_Pass.19

Encore 205 OPs non prouvées

100%

OP Suivante

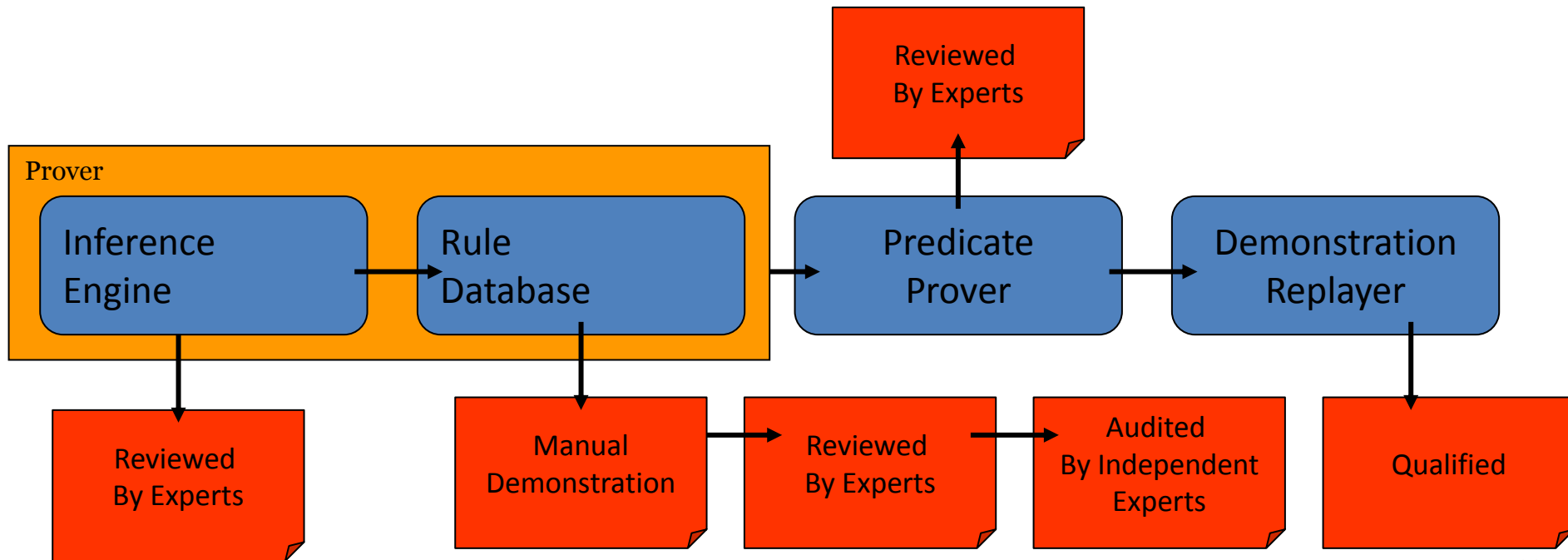
Opération Suivante

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
  - Initially 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
  - Initially 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

```

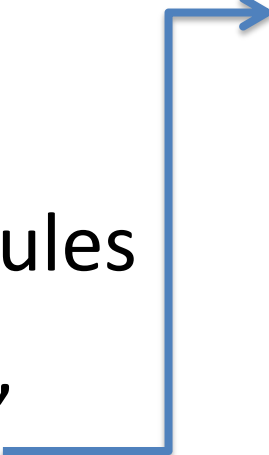
"Local hypotheses'" &
time: INTEGER &
morrow: INTEGER &
victor: PROCESSES &
leftspan: INTEGER &
clock0+1<=time &
time+1<=morrow &
not(clock0..morrow/\dom(Schedule) = {}) =>
clock0..morrow/\dom(Schedule) = {time} &
elected0: Tasks => time<=clock0+term0(elected0) &
elected0: Tasks => leftspan = term0(elected0)-(time-clock0) &
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



```

> SimplifyRelFonXY.36
> s1.2
> SimplifyRelDomXY.19
> SimplifyRelDorLongXY.3
> SimplifyRelInvXY.6
> CommutativityXY.4
> CommutativityXY.22
> CommutativityXY.25
> SimplifySetUniXY.17
  band bsearch({a},bVc,xVz)
    band binhyp(a : d)
      bsearch(d,xVz,y)
    blvar(Q)
    Q\ (a : d)
    =>
    bVc
    ==
    xVz
> GenEqualityX.2
> GenEqualityX.3
> SimplifyRelFonXY.16
> SimplifyRelDoaXY.3
> ContradictionXY.30
> EqualityXY.60
> EqualityXY.70
> EqualityXY.132
> EqualityXY.143
> EqualityXY.144
> b1.12
> GenEqualityX.1
> GenEqualityX.4
> GenEqualityX.5
> GenObvPredicateX.25
> GenObvPredicateX.26

```

# Additions

- 23 rules added to the whole project →

```
/* 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)
```

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

- $P(x)=Q(y)$  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  $E$

▲	⬆	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

# Validating rules for new domains

Rules

View: All rules

Name	Validated
Loaded Files	3/23
PatchProver*	3/23
DMS_SIG	0/13
DMS_DIV	1/4
DMS_DIV.1	Unproved (OPR already tried)
DMS_DIV.2	Unproved (OPR already tried)
DMS_DIV.3	Unproved (OPR already tried)
DMS_DIV.4	Proved (PP)
DMS_MOD	0/2
DMS_MUL	2/2
DMS_MUL.1	Proved (PP)
DMS_MUL.2	Proved (PP)
DMS_IND	0/1
DMS_FIN	0/1
class00.rules	0/0

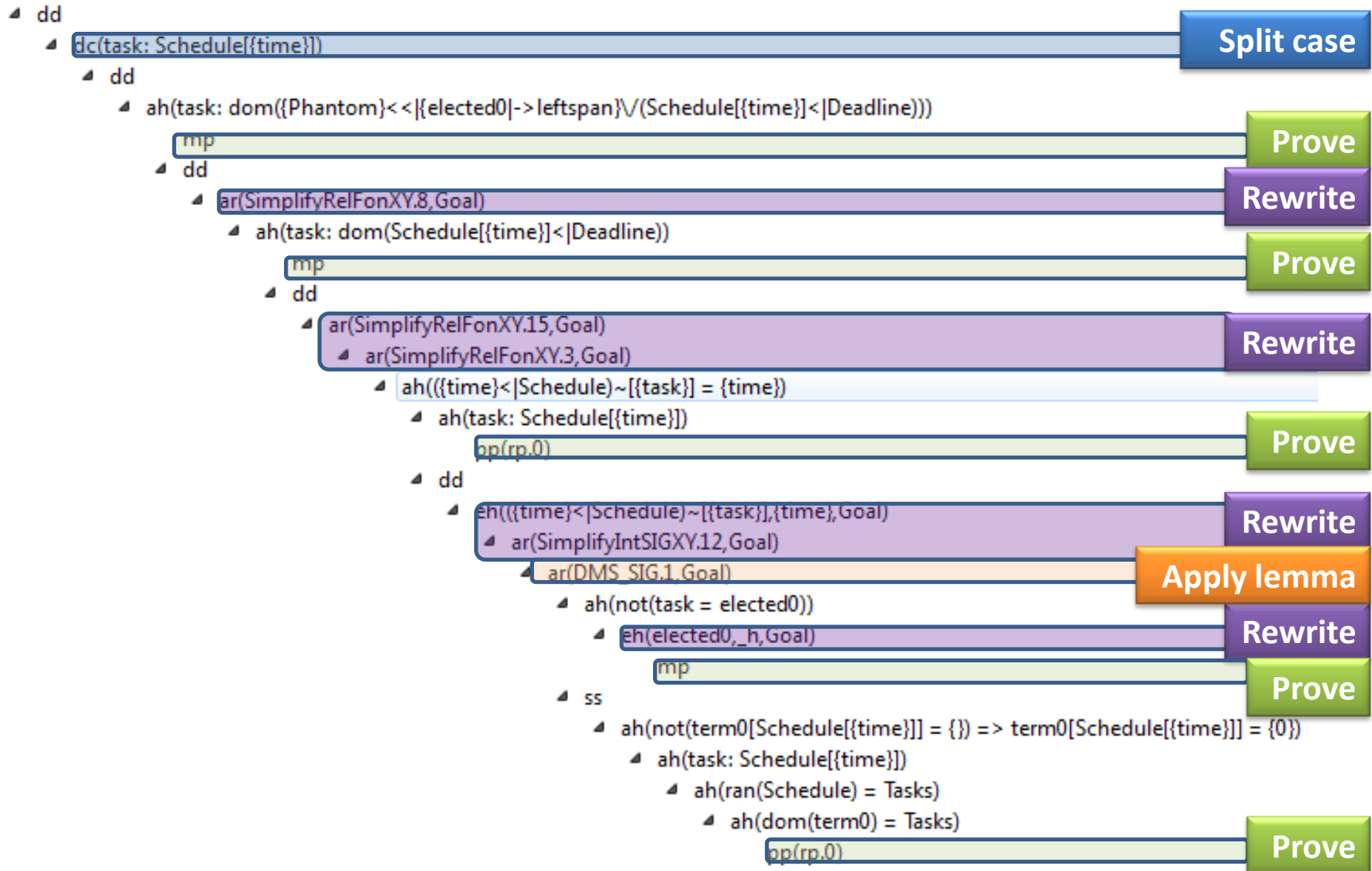
- DMS\_SIG
  - DMS\_SIG.1
  - DMS\_SIG.2
  - DMS\_SIG.3
  - DMS\_SIG.4
  - DMS\_SIG.5
  - DMS\_SIG.6
  - DMS\_SIG.7
  - DMS\_SIG.8
  - DMS\_SIG.9
  - DMS\_SIG.10
  - DMS\_SIG.11
  - DMS\_SIG.12
  - DMS\_SIG.13
- DMS\_DIV
  - DMS\_DIV.1
  - DMS\_DIV.2
  - DMS\_DIV.3
  - DMS\_DIV.4
- DMS\_MOD
  - DMS\_MOD.1
  - DMS\_MOD.2
- DMS\_MUL
  - DMS\_MUL.1
  - DMS\_MUL.2
- DMS\_IND
  - DMS\_IND.1
- DMS\_FIN
  - DMS\_FIN.1

# 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(span0V/clock0)->time)<[(log0V/morrow->victor)]~[task]) = (dom(span0<[(log0)-[task]]/[(clock0)-[log0]]~[task]))
  ah(time-1<=morrow)
  ah(clock0+1<=time)
  ah(dom(span0) = dom(log0)-(clock0))
  ah(dom(log0) < 0..clock0)
  pp(rp.0)
dd
  eh((dom(span0V/clock0)->time)<[(log0V/morrow->victor)]~[task]]_h.Goal)
  ar(DMS_SIG_2.Goal)
  ah(dom(span0) = dom(log0)-(clock0))
  ah(SIGMA(time$0).(time$0: (dom(span0)<[(log0)-[task]] | (span0V/clock0)->time))(time$0-time$0) = SIGMA(time$0).(time$0: (dom(span0)<[(log0)-[task]] | span0(time$0)...
  ar(DMS_SIG_3.Once)
  mp
  dd
    eh(SIGMA(time$0).(time$0: (dom(span0)<[(log0)-[task]] | (span0V/clock0)->time))(time$0-time$0)_h.Goal)
    ah(SIGMA(time$0).(time$0: ((clock0<[(log0)-[task]] | (span0V/clock0)->time))(time$0-time$0) = SIGMA(time$0).(time$0: ((clock0<[(log0)-[task]] | time-time$0))
    ar(DMS_SIG_3.Once)
    ah(dom(span0) = dom(log0)-(clock0))
    ah(span0: INTEGER <-> INTEGER)
    pp(rp.0)
  dd
    eh(SIGMA(time$0).(time$0: ((clock0<[(log0)-[task]] | (span0V/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(SimplifyRelFonXY12.Goal)
    ah((term0 <-> ((Phantom)<[(task->leftspan]/[Schedule][time]<[Deadline]](task) = leftspan)
    ah(task: dom((Phantom)<[(task->leftspan]/[Schedule][time]<[Deadline]]))
    mp
    dd
      ar(SimplifyRelFonXY8.Goal)
      ah(task: dom((Phantom)<[(task->leftspan)])
      mp
      dd
        ar(SimplifyRelFonXY14.Goal)
        ah(not(task = Phantom))
        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(SimplifyRelFonXY8.Goal)
      ah(task: dom(Schedule[time]<[Deadline])
      mp
      dd
        ar(SimplifyRelFonXY15.Goal)
        ar(SimplifyRelFonXY3.Goal)
        ah(((time)<[Schedule]~[task]) = (time))
        ah(task: Schedule[time])
        pp(rp.0)
      dd
        eh(((time)<[Schedule]~[task])(time).Goal)
        ar(SimplifyIntSIGXY12.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) = Task)
        pp(rp.0)
    mp
  mp
  ar(SimplifyRelFonXY7.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(span0)<[(log0)-[task]] | span0(time$0-time$0) - SIGMA(time$0).(time$0: ((clock0<[(log0)-[task]] | time-time...
  mp
  dd
    eh(SIGMA(time$0).(time$0: (dom(span0)<[(log0)-[task]] | span0(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: Tasks => 0 = SIGMA(time).(time: (dom(span0)<[(log0)-[task]] | span0(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(span0)<[(log0)-[task]] | span0(time-time) = SIGMA(time$0).(time$0: (dom(span0)...
      ar(DMS_SIG_5.Once)
      dd
        eh(SIGMA(time).(time: (dom(span0)<[(log0)-[task]] | span0(time-time)_h.Goal)
        mp
```

# Zoom on the proof tree



# 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
|
```



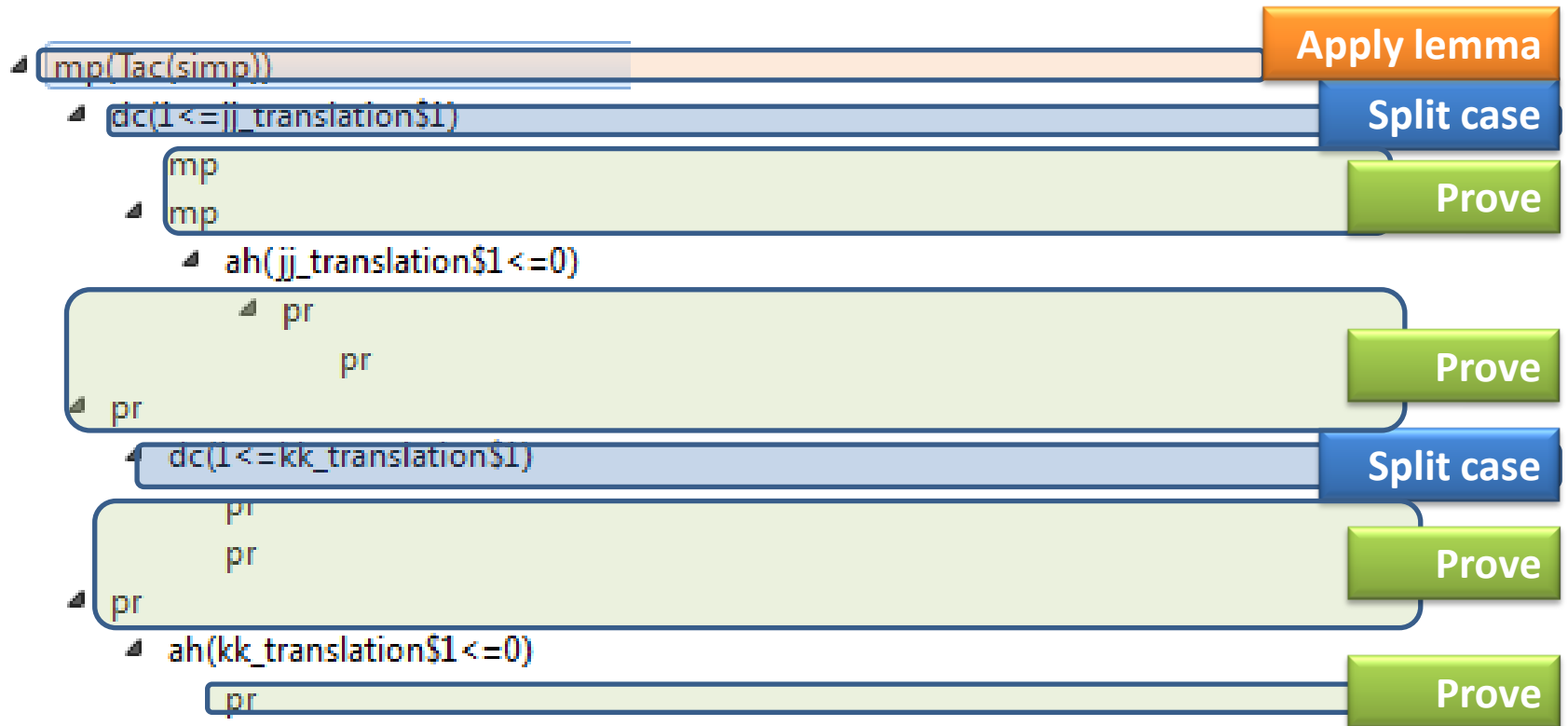
# iterateOnBlock.58

« 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,C1Pb}} &
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:  $\mathbb{N} \rightarrow \text{BOOL}$ )
- Specified by its properties:  
$$C \in \{C \in \wedge C(m+118)=\text{FALSE} \wedge C(m+119)=\text{TRUE} \wedge$$
$$C(m+120)=\text{FALSE} \wedge C(m+121)=\text{TRUE} \wedge C(m+122)=\text{TRUE} \wedge$$
$$C(m+123)=\text{FALSE} \wedge C(m+124)=\text{TRUE} \wedge C(m+124)=\text{FALSE} \wedge$$
$$C(m+125)=\text{FALSE} \wedge C(m+126)=\text{TRUE} \wedge C(m+127)=\text{TRUE} \wedge$$
$$\dots\}$$



# 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)

$C \in \{C \in \wedge C(m+118)=\text{FALSE} \wedge C(m+119)=\text{TRUE} \wedge$   
 $C(m+120)=\text{FALSE} \wedge C(m+121)=\text{TRUE} \wedge C(m+122)=\text{TRUE} \wedge$   
 $C(m+123)=\text{FALSE} \wedge C(m+124)=\text{TRUE} \wedge C(m+124)=\text{FALSE} \wedge$   
 $C(m+125)=\text{FALSE} \wedge C(m+126)=\text{TRUE} \wedge C(m+127)=\text{TRUE} \wedge$   
 $\dots\}$

- Exploit:
  - add trivial hypothesis:  $C(m+124) = C(m+124)$
  - Replace  $C(m+124)$  by its values:  $\text{TRUE} = \text{FALSE}$
  - 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