

Batch Proving and Proof Scripting in PVS

César A. Muñoz
munoz@nianet.org

National Institute of Aerospace

AFM 2006



NIA @ NASA LaRC



The PVS Theorem Prover

- PVS is a powerful **interactive** theorem prover.
- For expert users:
PVS provides a powerful **batch** mode as well.
- Why do we need a batch mode ?



The PVS Theorem Prover

- PVS is a powerful **interactive** theorem prover.
- For expert users:
PVS provides a powerful **batch** mode as well.
- Why do we need a batch mode ?



The PVS Theorem Prover

- PVS is a powerful **interactive** theorem prover.
- For expert users:
PVS provides a powerful **batch** mode as well.
- Why do we need a batch mode ?



Scenario 1

After several weeks we have finished the development of an Interval library in PVS: 10 files, 322 lemmas.

- We want to double check that the status of all the lemmas.
- A new version of PVS is available. We want to recheck all the proofs.



Scenario 2

Consider the function

$$r(\phi) = \frac{a}{1 + (1 - f)^2 \tan^2 \phi},$$

where a and f are constants. For efficiency reasons, we want to approximate the function $r(\phi)$ by the polynomial

$$\hat{r}(\phi) = \frac{4439091}{4} + (\bar{\phi}^2 - \phi^2) \times \left(\frac{9023647}{4} + (\bar{\phi}^2 - \phi^2) \times \left(\frac{13868737}{64} + (\bar{\phi}^2 - \phi^2) \times \left(\frac{13233647}{2048} + (\bar{\phi}^2 - \phi^2) \times \left(\frac{-1898597}{16384} + (\bar{\phi}^2 - \phi^2) \times \frac{-6661427}{131072} \right) \right) \right) \right),$$

where $\bar{\phi} = \frac{715}{512}$ and $\phi \in [0, \bar{\phi}]$.



Problem

- We want to prove that

$$\left| \frac{e(\phi)}{r(\phi)} \right| \leq 1.36 \times 10^{-6},$$

where $e(\phi) = r(\phi) - \hat{r}(\phi)$.

- In PVS,

```
PHI : Interval = [| 0,715/512 |]
```

```
RI : LEMMA
```

```
FORALL (phi:real) :
```

```
  phi ## PHI IMPLIES
```

```
  |e(phi) / r(phi)| ## [| 0,136/1000000000 |]
```



Automatic Proof by Interval Splitting

- Strategy:
 - 1 Use Interval's `numerical` on $[0, \bar{\phi}]$.
 - 2 If step 1 doesn't work, split interval into $[0, \frac{\bar{\phi}}{2}]$ and $[\frac{\bar{\phi}}{2}, \bar{\phi}]$, and recursively go to step 1.
- Problem: Very inefficient approach when a large number of splittings are needed (in this case about 10.000).
- Solution: Compute the splitting outside the theorem prover and generate PVS files with lemmas and proofs (in this case 3 lemmas per splitting).



Automatic Proof by Interval Splitting

- Strategy:
 - ① Use Interval's `numerical` on $[0, \bar{\phi}]$.
 - ② If step 1 doesn't work, split interval into $[0, \frac{\bar{\phi}}{2}]$ and $[\frac{\bar{\phi}}{2}, \bar{\phi}]$, and recursively go to step 1.
- Problem: Very inefficient approach when a large number of splittings are needed (in this case about 10.000).
- Solution: Compute the splitting outside the theorem prover and generate PVS files with lemmas and proofs (in this case 3 lemmas per splitting).



Automatic Proof by Interval Splitting

- Strategy:
 - ① Use Interval's `numerical` on $[0, \bar{\phi}]$.
 - ② If step 1 doesn't work, split interval into $[0, \frac{\bar{\phi}}{2}]$ and $[\frac{\bar{\phi}}{2}, \bar{\phi}]$, and recursively go to step 1.
- Problem: Very inefficient approach when a large number of splittings are needed (in this case about 10.000).
- Solution: Compute the splitting outside the theorem prover and generate PVS files with lemmas and proofs (in this case 3 lemmas per splitting).



PVS in Batch Mode

For Expert Users

- PVS prover and Emacs interface in batch mode:

```
% pvs -batch
```

- Regression testing:

```
;; file.el  
(pvs-validate "file.log" "dir"  
  (let ((current-prefix-arg t))  
    (prove-pvs-file "file.pvs")))
```

```
% pvs -batch -l file.el
```

- PVS prover without Emacs interface:

```
% pvs -raw
```



NIA @ NASA LaRC



PVS in Batch Mode

For Expert Users

- PVS prover and Emacs interface in batch mode:

```
% pvs -batch
```

- Regression testing:

```
;; file.el  
(pvs-validate "file.log" "dir"  
  (let ((current-prefix-arg t))  
    (prove-pvs-file "file.pvs")))
```

```
% pvs -batch -l file.el
```

- PVS prover without Emacs interface:

```
% pvs -raw
```



NIA @ NASA LaRC



PVS in Batch Mode

For Regular Users (via ProofLite's proveit utility)

```
% proveit Interval/top.pvs
```



NIA @ NASA LaRC



PVS in Batch Mode

For Regular Users (via ProofLite's proveit utility)

```
% proveit -importchain Interval/top.pvs
```



NIA @ NASA LaRC



PVS in Batch Mode

For Regular Users (via ProofLite's proveit utility)

```
% proveit -importchain -clean Interval/top.pvs
```



NIA @ NASA LaRC



PVS in Batch Mode

For Regular Users (via ProofLite's proveit utility)

```
% proveit -importchain -clean -packages Field Interval/top.pvs
```



NIA @ NASA LaRC



PVS in Batch Mode

For Regular Users (via ProofLite's proveit utility)

```
% proveit -importchain -clean -packages Field Interval/top.pvs  
Processing Interval/top.pvs. Writing output to file Interval/top.out.
```

Proof summary for theory interval

```
IMP_sigma_TCC1.....proved - complete  
sharp_Proper.....proved - complete  
Proper_sharp.....proved - complete  
specialbrackets_TCC1.....proved - complete  
Lt_Ge.....proved - complete  
Le_Gt.....proved - complete  
Abs_TCC1.....proved - complete  
Abs_TCC2.....proved - complete
```

...

```
Theory totals: 156 formulas, 156 attempted, 156 succeeded (72.33 s)
```

...

```
Grand Totals: 322 proofs, 322 attempted, 322 succeeded (122.73 s)
```



NIA @ NASA LaRC



“The format is:

```
(<theory-id> (<decl-id> <default-proof-posn> (<id>  
<description> <create-date> <run-date> <script> <status>  
<refers-to> <real-time> <run-time> <interactive?>  
<decision-procedure-used>) ...) ...)
```

where <default-proof-posn> is the (0-based) position of the default proof in the list of proofs associated with the declaration. The <create-date> is the time that the proof was first saved, and the <run-date> is the time it was last rerun. The <real-time> and <run-time> are the time it took the last time it was run, and <interactive?> indicates whether that was an interactive run or not [...] Most of the rest of the fields should be self-explanatory ...”*



PVS Batch Proofs

For Regular Users (via ProofLite scripts)

```
PHIO : Interval = [| 0, 82225/51200000 |]
```

```
RpIO : LEMMA
```

```
  phi ## PHIO
```

```
  IMPLIES
```

```
    |ep(phi)/rp(phi)| ## [| 0, 136/10000000 |]
```

```
%|- RpIO : PROOF
```

```
%|- (instint :taylor "Ep0" :hints "Ep_deriv")
```

```
%|- QED
```



NIA @ NASA LaRC



The ProofLite Package

- Package for non-interactive proof scripting in PVS:
 - Utility for running the theorem prover in batch mode.
 - A proof scripting notation where proof scripts reside in `.pvs` files.
- Suitable for batch generation of specifications and **proof scripts**.
- Download: <http://research.nianet.org/~munoz/ProofLite>



The proveit Utility

Usage: `proveit [OPTION] FILE[@TH1,...,THn]*`

For each `FILE`, `proveit` runs PVS in batch mode and proves theories `TH1, ..., THn`, which are either imported or defined in `FILE.pvs`. If no theories are provided, `proveit` proves all theories in `FILE`.

- `-clean`: Removes bin files and `.pvscontext` before proving
- `-force`: Overrides current proofs with ProofLite scripts
- `-importchain`: Proves chain of imported theories
- `-packages P1, ..., Pn`: Loads packages `P1, ..., Pn`
- `-prooftraces`: Output proof traces.



- ProofLite scripts are written in PVS files using the special comment form:

```
l1: LEMMA a*a >= 0
```

```
%|- l1 : PROOF (grind) QED
```

- ProofLite scripts can extend to multiple lines:

```
l2: LEMMA (nza/2)*(2/nza) = 1
```

```
%|- l2 : PROOF
```

```
%|-   (then (skosimp)
```

```
%|-       (grind))
```

```
%|- QED
```



Sharing ProofLite Scripts

Several lemmas can share the same ProofLite script:

```
13: LEMMA a*a >= 0
```

```
14: LEMMA (nza/2)*(2/nza) = 1
```

```
%|- 13 : PROOF
```

```
%|- 14 : PROOF
```

```
%|- (grind)
```

```
%|- QED
```



- Name-matching lemmas can share the same ProofLite script.
- The symbol `*` stands for an arbitrary sequence of one or more characters, e.g.,

```
l3a: LEMMA a*a >= 0
```

```
l4a: LEMMA (nza/2)*(2/nza) = 1
```

```
%|- l*a : PROOF
```

```
%|- (grind)
```

```
%|- QED
```



- Name-matching lemmas can be used to create macro scripts.
- The symbol $\$0$ refers to the name of the lemma and the symbol $\$n$ refers to n -th matching string from left to right, e.g.,

```
1_5_6 : LEMMA EXISTS (a) : 5 < a AND a < 6
```

```
1_6_7 : LEMMA EXISTS (a) : 6 < a AND a < 7
```

```
%|- 1_*_* : PROOF
%|-   (then (skip-msg "Proving Lemma: $0")
%|-     (inst 1 "$1 + ($2 - $1)/2")
%|-     (grind))
%|- QED
```



Parametric Scripts

- Parametric scripts have the form:

```
%|- <script_name>[e1;...;en] : PROOF
%|-   <steps>
%|- QED
```

- The symbol $\#n$ is substituted by e_n , e.g.,

```
l_8 : LEMMA EXISTS (a,b) : a+b = 8
l_9 : LEMMA EXISTS (a,b) : a+b = 9
%|- l_8[2;6] : PROOF
%|- l_9[4;5] : PROOF
%|-   (then (skip-msg "Proving Lemma: $0")
%|-       (inst 1 "#1" "#2")
%|-       (grind))
%|- QED
```



Installing ProofLite Scripts

Interactively

- ProofLite scripts in the current theory.
 - Without overriding old proofs:
`M-x install-proof-lite-scripts-theory (C-c it).`
 - Overriding old proofs:
`M-x install-proof-lite-scripts-theory! (C-c !t).`
- ProofLite scripts at the cursor position.
 - Without overriding old proofs:
`M-x install-proof-lite-script (C-c ip).`
 - Overriding old proofs:
`M-x install-proof-lite-script! (C-c !p).`



Installing ProofLite Scripts

In batch mode

`proveit` automatically installs ProofLite scripts on untried formulas (and on tried formulas if the option `-force` is used).



NIA @ NASA LaRC



Creating ProofLite Scripts from Proofs

ProofLite scripts can be created from proofs in two ways:

- Place the cursor on the formula for which you want to create a ProofLite script and issue the Emacs command:
`M-x insert-proof-lite-script (C-c 2p)`. The ProofLite script is automatically inserted after the formula.
- Issue the command:
`M-x display-proof-lite-script (C-c dp)`
and enter the name of a formula. The ProofLite script of that formula is displayed in the buffer “ProofLite”.



Application: Verification of Numerical Bounds

30.000 lemmas (and their respective proof scripts) were generated and mechanically discharged in batch mode [1].



Application: Verification of ATM Concept

117 lemmas (and their respective proof scripts) were generated and mechanically discharged in batch mode [2].

```
%|- T_*: PROOF (st) QED
```

```
%|- Maz_*: PROOF (smaz) QED
```

```
T_0 : LEMMA
```

```
So+Lb(iaf(1)) <= D(1,T(2))+Lb(iaf(2)) AND T(2) <= t AND
```

```
...
```

```
Lb(iaf(3))+Lf <= D(3,T(1))
```

```
IMPLIES St <= S(3,1,t)
```

```
Maz_74 : LEMMA
```

```
So+Lb(iaf(3)) <= D(3,T(4))+Lb(iaf(4)) AND
```

```
...
```



```
T(1) <= t AND D(1,t) <= Lb(iaf(1))+Lf+Lm(mahf(1))
```

```
IMPLIES Smaz <= S(3,1,t)
```



- The basic capabilities provided by ProofLite are already available in proof assistants such as Coq, HOL, etc.
- The ProofLite scripting notation also supports several forms of proof sharing and proof reuse.
- Proof scripts vs. user defined strategies.



-  M. Dumas, G. Melquiond, and C. Muñoz.
Guaranteed proofs using interval arithmetic.
In Proceedings of the 17th IEEE Symposium on Computer Arithmetic, ARITH-17, Cape Cod, Massachusetts, 2005.
-  C. Muñoz and G. Dowek.
Hybrid verification of an air traffic operational concept.
In Proceedings of IEEE ISoLA Workshop on Leveraging Applications of Formal Methods, Verification, and Validation, Columbia, Maryland, 2005.

