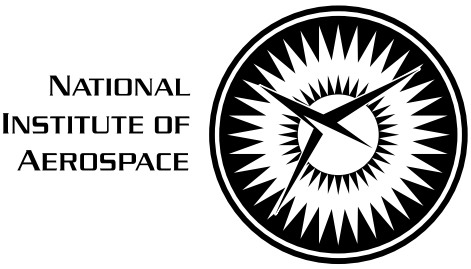
NASA/CR-2007-214546
NIA Report No. 2007-03



Batch Proving and Proof Scripting in PVS

Cesar A. Munoz
National Institute of Aerospace (NIA), Hampton, Virginia

February 2007

The NASA STI Program Office . . . in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA Scientific and Technical Information (STI) Program Office plays a key part in helping NASA maintain this important role.

The NASA STI Program Office is operated by Langley Research Center, the lead center for NASA's scientific and technical information. The NASA STI Program Office provides access to the NASA STI Database, the largest collection of aeronautical and space science STI in the world. The Program Office is also NASA's institutional mechanism for disseminating the results of its research and development activities. These results are published by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counterpart of peer-reviewed formal professional papers, but having less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services that complement the STI Program Office's diverse offerings include creating custom thesauri, building customized databases, organizing and publishing research results ... even providing videos.

For more information about the NASA STI Program Office, see the following:

- Access the NASA STI Program Home Page at ***http://www.sti.nasa.gov***

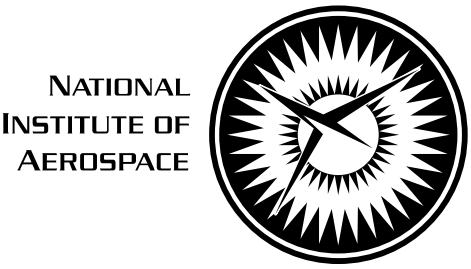
- E-mail your question via the Internet to help@sti.nasa.gov

- Fax your question to the NASA STI Help Desk at (301) 621-0134

- Phone the NASA STI Help Desk at (301) 621-0390

- Write to:
  NASA STI Help Desk
  NASA Center for AeroSpace Information
  7115 Standard Drive
  Hanover, MD 21076-1320

NASA/CR-2007-214546
NIA Report No. 2007-03



Batch Proving and Proof Scripting in PVS

Cesar A. Munoz
National Institute of Aerospace (NIA), Hampton, Virginia

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

Prepared for Langley Research Center
under Cooperative Agreement
NCC1-02043

February 2007

Available from:

NASA Center for AeroSpace Information (CASI)
7115 Standard Drive
Hanover, MD 21076-1320
(301) 621-0390

National Technical Information Service (NTIS)
5285 Port Royal Road
Springfield, VA 22161-2171
(703) 605-6000

BATCH PROVING AND PROOF SCRIPTING IN PVS*

César A. Muñoz†

ABSTRACT

The batch execution modes of PVS are powerful, but highly technical, features of the system that are mostly accessible to expert users. This paper presents a PVS tool, called ProofLite, that extends the theorem prover interface with a batch proving utility and a proof scripting notation. ProofLite enables a semi-literate proving style where specification and proof scripts reside in the same file. The goal of ProofLite is to provide batch proving and proof scripting capabilities to regular, non-expert, users of PVS.

1 INTRODUCTION

The Prototype Verification System (PVS) [9] is a higher order logic theorem prover developed and maintained by SRI International.[1] PVS has been applied to verification problems in a variety of areas, including safety critical industrial applications.

PVS is well known for its expressive specification language and its impressive theorem prover. The specification language is based on a rich type system that supports predicate subtyping and dependent records [12]. The theorem prover has been optimized for large proofs, for example basic numerical types are built-in and propositional simplification uses Binary Decision Diagrams. Furthermore, like most theorem provers, PVS can be conservatively extended with user-defined inference rules [10], called *strategies*, that tailor the deductive power of the system to specific domains [1, 15].

Less known features of PVS are the batch execution modes. Although these modes are quite powerful, their correct use requires a good knowledge of the PVS programming interface. Therefore, they are mostly accessible to PVS expert users.

Another limitation of the PVS interface is that, in contrast to most theorem provers, it does not explicitly support a proof scripting notation where proofs are written in a non-interactive way. In PVS, proofs are interactively constructed via proof commands through a read-and-evaluate loop. The proof commands are automatically saved by the system in text files using an internal format. Those files are not intended to be directly edited by the user.

These two capabilities, batch proving and proof scripting, become important when PVS is integrated into other verification tools. Assume for example that a static checker of a programming language wants to generate proof obligations for PVS along with specialized proof commands for each obligation. The formulas can be written into a `.pvs` file. The proofs commands, on the other hand, have to be written into a `.prf` file using the internal proof fomat. Finally, a PVS batch execution mode has to be used to check whether the proof obligations are discharged or not.

*This work was supported by the National Aeronautics and Space Administration, Langley Research Center under the Research Cooperative Agreement No. NCC-1-02043.

†National Institute of Aerospace (NIA), 100 Exploration Way, Hampton VA, 23666. Email: `munoz@nianet.org`.

[1]PVS is electronically available from `http://pvs.csl.sri.com`.

This paper describes a PVS tool, called ProofLite, that provides a user-friendly interface to a PVS batch execution mode. ProofLite also supports a proof scripting notation where formulas and proofs may reside in the same text file. The rest of this paper is structured as follows. Section 2 gives an overview of the PVS batch modes. Section 3 briefly presents different proof formats used by PVS. Sections 4 and 5 describe the tool and its applications. The last section concludes this work.

2 PVS BATCH MODES

Typically, users interact with PVS through its customized Emacs interface. Even mechanical tasks that do not involve editing, such as, rerunning all the proofs of a fully developed theory, normally require an interaction with the PVS Emacs interface.

Curious PVS users may have noticed that the PVS command line accepts the option "`-batch`", which runs the system in batch mode [11]. This option is generally used with the option `-l` that loads and executes an Emacs Lisp file. This facility is extremely powerful as arbitrarily complex Emacs Lisp can be executed this way. In particular, any PVS command can be invoked. Unfortunately, many PVS commands are context-dependent and only make sense when they are invoked interactively. Therefore, the correct use of this mechanism requires a good knowledge of the PVS programming interface.

One of the main uses of the PVS batch mode is regression testing. For instance, the following Emacs Lisp code will change the context to `<dir>`, rerun all the proofs of `<file.pvs>`, and collect the output into `<file.log>`. It will then compare the output against the last run and report whether there is nothing to compare, there are no significant changes, or some difference were found since the last run.

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

If this code is saved in the file `<file.el>`, the validation run can be performed in batch mode with the command line:

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

When a difference is reported, the Emacs command
`M-x pvs-compare-validation-window` will place the cursor at the position where the output files differs, if the two log files are in a split window.

For real PVS hackers, a more obscure execution mode is available through the option `-raw`. In this mode, the PVS Common Lisp runtime engine is executed without the Emacs interface. Common Lisp expressions, and in particular PVS Common Lisp commands, can be executed in batch mode via the command line option `-e`.

3 PVS PROOF FORMATS

In PVS, specifications and proofs reside in different types of files. Specifications are written in `.pvs` text files. Proofs are interactively constructed via proof commands and automatically

saved by the system in `.prf` files. Although proof files are also text files, they are not intended for user manipulation. The format of the `.prf` file is described by Sam Owre, one the main developers of the system, in a message to the PVS mailing list on June 2003 as follows: "... 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. These may not really reflect the last run, because the prove-theory, etc. commands do not write out a new `.prf` file. Most of the rest of the fields should be self-explanatory ..."

Furthermore, existing PVS proofs can be edited using the PVS Emacs interface. When a proof is edited by the user, it is presented in the Emacs buffer `Proof` as a sequence of commands in a proof tree. For instance, a possible proof of lemma `th2:`

```
th2 : LEMMA a <= b IMPLIES a*abs(a) <= b*abs(b)
```

is displayed in the buffer `Proof` as follows:

```
(""
 (skeep)
 (case "a >= 0")
 (("1" (grind :theories "real_props"))
  ("2"
   (grind :theories "real_props")
   (mult-ineq -1 -1 :signs (- -))
   (assert))))
```

Note that, in this format, any control structure provided by a proof strategy such as `try`, `if`, `branch`, etc., is lost.

The buffer `Proof` is typically used for global editing operations, such as replacing an identifier, for copying a proof from one formula to another, and for stepping through a proof via the interactive theorem prover. However, given the lack of control structure information, the proof format displayed in the buffer `Proof` is not suitable for proof scripting.

4 PROOFLITE

ProofLite is a *PVS package*.[2] PVS packages, which are also called *prelude extensions*, are the mechanism offered by PVS to modularly and conservatively extend the system with user-defined Emacs Lisp code, Common Lisp code, proof strategies, and PVS theories. In particular, the ProofLite package consists of Emacs Lisp and Common Lisp functions that implement:

- a command line utility, called `proveit`,
- a proof scripting notation, and
- a set of Emacs commands for management of proof scripts.

4.1 The `proveit` Utility

ProofLite includes the command line utility `proveit` that executes the theorem prover in batch mode on a `.pvs` file and reruns all its proofs.

For instance, assume that all the formulas in `thms.pvs` have been proved.

```
thms : THEORY
BEGIN
  a,b : VAR real
  nza : VAR nzreal

  th1    : LEMMA a*a >= 0
  th2    : LEMMA a <= b IMPLIES a*abs(a) <= b*abs(b)
  th3    : LEMMA a*a >= 0
  th4    : LEMMA (nza/2)*(2/nza) = 1
  th3a   : LEMMA a*a >= 0
  th4a   : LEMMA (nza/2)*(2/nza) = 1
  th_5_6 : LEMMA EXISTS (a) :  5 < a AND a < 6
  th_6_7 : LEMMA EXISTS (a) :  6 < a AND a < 7
  th_8   : LEMMA EXISTS (a,b) :  a+b = 8
  th_9   : LEMMA EXISTS (a,b) :  a+b = 9
END thms
```

The invocation

```
% proveit thms
```

[2] ProofLite is freely available from `http://research.nianet.org/~munoz/ProofLite`.

reruns all the proofs in `thms.pvs`, writes the output into `thms.out`, and prints the following summary information:

```
Processing thms.pvs. Writing output to file thms.out.

 Proof summary for theory thms
    th1.................................proved - complete
    th2.................................proved - complete
    th3.................................proved - complete
    th4.................................proved - complete
    th3a................................proved - complete
    th4a................................proved - complete
    th_5_6..............................proved - complete
    th_6_7..............................proved - complete
    th_8................................proved - complete
    th_9................................proved - complete
    Theory totals: 10 formulas, 10 attempted, 10 succeeded (2.63 s)

Grand Totals: 10 proofs, 10 attempted, 10 succeeded (2.63 s)
```

The utility `proveit` supports several options, e.g,

- The option `-clean` removes `.pvscontext` and other binary files. This option is useful when the system has died abruptly and the context is left in an inconsistent state.
- The option `-importchain` reruns the proofs of all imported theories as well.
- The option `-prooftraces` outputs the proof traces, which are needed for regression testing. Unfortunately, this option does not provide all the functionality of `pvs-validate` yet. This extension is planned for a future release.
- The option `-package` loads a PVS strategy package such as Manip [16], Field [8], PVSio [5] or Interval [7]. For instance, if the proofs in `thms.pvs` use strategies defined in Field, the invocation has the form:

  ```
  % proveit -package Field thms
  ```

- The option `-help` prints the complete set of options supported by the utility.

4.2 ProofLite Scripts

ProofLite scripts are proof scripts written in specially formatted comments that reside in regular `.pvs` files. The simplest type of ProofLite script has the form

```
%|- <id> : PROOF <step> QED
```

where `<id>` is the name of an existing formula and `<step>` is a proof command supported by the PVS strategy language [13].

For instance, the proof of `th1` can be written in the file `thms.pvs` using the ProofLite script:

```
%|- th1 : PROOF (grind) QED
```

ProofLite scripts can extend to multiple lines. In this case, each line is preceded by the special comment "`%|-`". For instance, the proof of lemma `th2` can be written:

```
%|- th2 : PROOF
%|- (then
%|-  (skeep)
%|-  (spread (case "a >= 0")
%|-   ((grind :theories "real_props")
%|-    (then (grind :theories "real_props")
%|-          (mult-ineq -1 -1 :signs (- -))
%|-          (assert)))))
%|- QED
```

Normally, ProofLite scripts are just comments to the PVS system. Indeed, unless explicitly requested by the user, ProofLite scripts are not installed as proofs. The ProofLite utility `proveit` automatically installs proof scripts into their respective formulas when processing a `.pvs` file. To prevent accidental overriding of proofs, by default, `proveit` does not install proof scripts in formulas that have an existing proof. To override existing proofs, the `proveit` option `-force` must be used. Installation of ProofLite scripts can also be done through the interactive PVS Emacs interface as described in Section 4.3.

Proof script sharing is supported by ProofLite. For instance, the following ProofLite script associates the same proof script to lemmas `th3` and `th4`:

```
%|- th3 : PROOF
%|- th4 : PROOF
%|-   (grind)
%|- QED
```

The proof sharing mechanism is generalized to name-matching formulas, where the character "`*`" in the script identifier stands for an arbitrary sequence of one or more characters. In the following example, all formulas in `thms.pvs` whose names match the string "`th*a`", e.g., `th3a` and `th4a`, share the same proof command:

```
%|- th*a : PROOF (then (skeep) (grind-reals)) QED
```

Proof scripts are not restricted to user-defined formulas. The following ProofLite script associates the same proof command to all Type Correctness Conditions (TCCs) in a theory:

```
%|- *_TCC* : PROOF <step> QED
```

Name-matching lemmas can be used to create proof macros. In a ProofLite script `%|- <id> : PROOF <step> QED`, the proof command `<step>` may contain the special symbols `$n`, where `n` ≥ 0. The symbol `$0` refers to the name of the lemma that matches `<id>`. The symbol `$n`, where `n` ≥ 1, refers to n-th matching string, from left to right, in the lemma's name. Consider the ProofLite script

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

The string `th_*_*` matches the name `th_5_6`. Therefore, the symbols `$0`, `$1`, and `$2` refer to `th_5_6`, `5`, and `6`, respectively. In this case, the proof command associated with lemma `th_5_6` is

```
(then (skip-msg "Proving Lemma: th_5_6")
      (inst 1 "5 + (6 - 5)/2")
      (grind))
```

Moreover, the string `th_*_*` matches the name `th_6_7`. Therefore, the proof command associated with lemma `th_6_7` is

```
(then (skip-msg "Proving Lemma: th_6_7")
      (inst 1 "6 + (7 - 6)/2")
      (grind))
```

Proof macros are particularly useful when PVS specifications are automatically generated and proof lemmas follow a particular naming convention. However, the parameters enabled by this mechanism are limited to substrings of valid identifiers. ProofLite supports a more general parameterization mechanism. Parametric ProofLite scripts have the form:

```
%|- <id>[e1;...;en]: PROOF
%|-   <step>
%|- QED
```

In `<step>`, the symbol `#`n is substituted by e_n. Consider the ProofLite script

```
%|- th_8[2;6] : PROOF
%|- th_9[4;5] : PROOF
%|-   (then (skip-msg "Proving Lemma: $0")
%|-         (inst 1 "#1" "#2")
%|-         (grind))
%|- QED
```

In this case, the proof command associated with lemma `th_8` is

```
(then (skip-msg "Proving Lemma: th_8")
      (inst 1 "2" "6")
      (grind))
```

Moreover, the proof command associated with lemma `th_9` is

```
(then (skip-msg "Proving Lemma: th_9")
      (inst 1 "4" "5")
      (grind))
```

4.3 Proof Script Management Through the PVS Emacs Interface

In general, a PVS package is loaded into the interactive PVS Emacs interface through the Emacs command `M-x load-prelude-library`, which will prompt the user for a package name, e.g., `ProofLite`. This has to be done only the first time that the package is used in a working context or after the `.pvscontext` file has been removed.

Once ProofLite has been loaded into the PVS Emacs interface, a ProofLite script can be installed as the default proof of a formula by placing the cursor on the script and issuing the Emacs command `M-x install-prooflite-script`. If the ProofLite is shared by several formulas, all proofs are simultaneously installed. However, this command does not install a proof in formulas that already have a default proof. The Emacs commands `M-x install-prooflite-script!` forces the installation of a proof script regardless of the existence of a previous proof.

All ProofLite scripts in theory can be installed for the first time through the Emacs command `M-x install-prooflite-scripts-theory`. Alternatively, the Emacs command `M-x install-prooflite-scripts-theory!` installs all ProofLite scripts and overwrites any default proof.

The default proof of a formula can be converted into a ProofLite script by placing the cursor on the formula and issuing the Emacs command `M-x insert-prooflite-script`. The script is automatically inserted in the `.pvs` file after the formula. The Emacs command `M-x display-prooflite-script` prompts the user for a formula name and, then, puts the ProofLite script of the formula's default proof in the Emacs buffer `ProofLite`. Afterward, the script can be modified and manually inserted anywhere in the `.pvs` file.

Key abbreviations for all these commands are listed in the following table.

Emacs Command	**Key Abbreviation**
`M-x install-prooflite-script`	`C-c ip`
`M-x install-prooflite-script!`	`C-c !p`
`M-x install-prooflite-scripts-theory`	`C-c it`
`M-x install-prooflite-scripts-theory!`	`C-c !t`
`M-x insert-prooflite-script`	`C-c 2p`
`M-x display-prooflite-script`	`C-c dp`

5 APPLICATIONS

ProofLite has been extensively and successfully used in verification projects at the National Institute of Aerospace and NASA Langley.

Reference [2] presents a tool for mechanical verification of numerical bounds using interval arithmetic. The formal verification is performed in PVS. However, all the technical burden of proving properties in a proof assistant system is hidden from the user. In this case, a C++ module computes bounds of numerical expressions and, then, generates proof obligations, in the form of PVS formulas, along with ProofLite scripts that discharge the obligations. Formulas and proof scripts are written in a series of `.pvs` files that are processed in batch mode via the command line utility `proveit`. The tool was used to formally check that a polynomial approximation, taken from a critical aeronautical application, is close to about one unit in the last place of the exact transcendental function, i.e., the relative error is

bounded by 1.36 × 10-6. The C++ module generated about 30000 PVS lemmas, with their respective proof scripts, that were mechanically checked on a high performance cluster.

Reference [6] reports on the formal verification of an operational concept for air traffic management in a self controlled airspace. The operational concept is modeled as a hybrid non-deterministic asynchronous state transition system. A tool, implemented in PVSio [5] and formally verified in PVS, explicitly computes the set of reachable states of the system. From this set, PVS lemmas, and their respective ProofLite scripts, are generated. All together, the lemmas guarantee that under nominal operations the minimum separation between two aircraft is higher than a given safety threshold. In total, 117 lemmas were generated and mechanically verified in batch mode via the ProofLite utility `proveit`.

6 CONCLUSION

ProofLite is a PVS package for batch proving and proof scripting that can be used by regular PVS users. The basic capabilities provided by the package are commonly found in comparable theorem provers such as Coq [14], HOL [4], and ACL2 [3].

The ProofLite scripting notation supports several forms of proof sharing and proof reuse. Modern theorem provers provide mechanisms to conservatively extend the proof search and automation power of their systems via user-defined strategies. Proof scripting provides a higher level of abstraction that is appropriate for certain kind of problems and domains.

Future versions of ProofLite will fully support regression testing and will continue to explore new ways of sharing and reusing proofs.

Acknowledgment

During a summer visit to NIA in August 2005, Florent Kirchner rewrote most of the new (and faster) code of the `proveit` utility.

REFERENCES

[1] M. Archer. TAME: Using PVS strategies for special-purpose theorem proving. *Ann. Math. Artif. Intell*, 29(1-4):139–181, 2000.

[2] 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.

[3] Matt Kaufmann and J Strother Moore. ACL2: An industrial strength version of nqthm. In *Compass'96: Eleventh Annual Conference on Computer Assurance*, page 23, Gaithersburg, Maryland, 1996. National Institute of Standards and Technology.

[4] M. J. C. Gordon and T. F. Melham. *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*. Cambridge University Press, 1993.

[5] C. Muñoz. Rapid prototyping in PVS. Report NIA Report No. 2003-03, NASA/CR-2003-212418, NIA-NASA Langley, National Institute of Aerospace, Hampton, VA, May 2003.

[6] 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.

[7] C. Muñoz and D. Lester. Real number calculations and theorem proving. In J. Hurd and T. Melham, editors, *Proceedings of the 18th International Conference on Theorem Proving in Higher Order Logics, TPHOLs 2005*, volume 3603 of *Lecture Notes in Computer Science*, pages 195–210, Oxford, UK, 2005. Springer-Verlag.

[8] C. Muñoz and M. Mayero. Real automation in the field. Technical Report NASA/CR-2001-211271 Interim ICASE Report No. 39, ICASE-NASA Langley, ICASE Mail Stop 132C, NASA Langley Research Center, Hampton VA 23681-2199, USA, December 2001.

[9] S. Owre, J. M. Rushby, and N. Shankar. PVS: A prototype verification system. In Deepak Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752, Saratoga, NY, June 1992. Springer-Verlag.

[10] S. Owre and N. Shankar. Writing PVS proof strategies. In Myla Archer, Ben Di Vito, and César Muñoz, editors, *Design and Application of Strategies/Tactics in Higher Order Logics (STRATA 2003)*, number CP-2003-212448 in NASA Conference Publication, pages 1–15, Hampton, VA, September 2003. NASA Langley Research Center. The complete proccedings are available at `http://research.nianet.org/fm-at-nia/STRATA2003/`.

[11] S. Owre, N. Shankar, J. M. Rushby, and D. W. J. Stringer-Calvert. *PVS System Guide*. Computer Science Laboratory, SRI International, Menlo Park, CA, September 1999.

[12] Sam Owre and Natarajan Shankar. The formal semantics of PVS. Technical Report SRI-CSL-97-2, Computer Science Laboratory, SRI International, Menlo Park, CA, August 1997.

[13] N. Shankar, S. Owre, J. M. Rushby, and D. W. J. Stringer-Calvert. *PVS Prover Guide*. Computer Science Laboratory, SRI International, Menlo Park, CA, September 1999.

[14] The Coq Team. The Coq proof assistant: Reference manual: Version 7.2. Technical Report RT-0255, INRIA, Rocquencourt, France, February 2002. Available at `http://coq.inria.fr/doc/main.html`.

[15] B. Di Vito. High-automation proofs for properties of requirements models. *STTT*, 3(1):20–31, 2000.

[16] B. Di Vito. A PVS prover strategy package for common manipulations. Report NASA/TM-2002-211647, NASA Langley Research Center, Hampton, VA 23681-0001, 2002.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.
PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
01/02/2007	Contractor Report	

4. TITLE AND SUBTITLE

Batch Proving and Proof Scripting in PVS

5a. CONTRACT NUMBER

5b. GRANT NUMBER
NCC1-02043

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Munoz, Cesar A.

5d. PROJECT NUMBER
2204

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

NASA Langley Research Center
Hampton, VA 23681-2199

National Institute of Aerospace (NIA)
100 Exploration Way
Hampton, VA 23666

8. PERFORMING ORGANIZATION REPORT NUMBER
NIA Report No. 2007-03

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSORING/MONITOR'S ACRONYM(S)
NASA

11. SPONSORING/MONITORING REPORT NUMBER
NASA/CR-2007-214546

12. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified - Unlimited
Subject Category 61
Availability: NASA CASI (301) 621-0390

13. SUPPLEMENTARY NOTES

Langley Technical Monitor: Ricky W. Butler

14. ABSTRACT

The batch execution modes of PVS are powerful, but highly technical, features of the system that are mostly accessible to expert users. This paper presents a PVS tool, called ProofLite, that extends the theorem prover interface with a batch proving utility and a proof scripting notation. ProofLite enables a semi-literate proving style where specification and proof scripts reside in the same file. The goal of ProofLite is to provide batch proving and proof scripting capabilities to regular, non-expert, users of PVS.

15. SUBJECT TERMS

Theorem Proving; Proof Scripting

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19b. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			STI Help Desk (email: help@sti.nasa.gov)
U	U	U	UU	15	19b. TELEPHONE NUMBER *(Include area code)* (301) 621-0390