

A Library Manager for the Prototype Verification System

Miguel Romero and Camilo Rocha

Escuela Colombiana de Ingeniería
AK 45 No. 205-59 (Autopista Norte)
Bogotá, Colombia.

Abstract. The Prototype Verification System (PVS) is a successful specification language with support tools and an automated theorem prover that is being used globally by the formal methods community. This paper presents `pvsln`, a tool for managing PVS libraries featuring support for different library sources, libraries with several theories, and dependencies among theories. The tool, freely available for download, is a command line application written in the Python programming language and depends, mainly, on the distributed revision control system *git*. This paper presents the main features of `pvsln` and a description language for annotating libraries so that they can be shared with the help of the tool. This paper also includes a detailed step-by-step guide on how to use `pvsln` for the configuration of the current version of the NASA PVS Library, available from GitHub.

1 Introduction

The Prototype Verification System [4] (PVS) is a mechanized environment for formal specification and verification. PVS consists of a specification language, a number of predefined theories, a type checker, an interactive theorem prover that supports the use of several decision procedures and a symbolic model checker. Nowadays, PVS is being used as a productive environment for constructing and maintaining large formalizations and proofs. PVS is extensible via libraries, collections of theories comprising formal developments that include definitions, theorems, and proofs in the PVS language. Given the increasing size of mathematical developments in the form of libraries, and the growing need for and adoption of mechanized proof environments [1,2], it is important to have tools for managing these libraries. This paper presents `pvsln` [5], an open source tool for managing PVS libraries that is freely available for download [5] under the GNU General Public License GPLv3.

The `pvsln` tool features support for different library sources, libraries with several theories, and dependencies among theories within the same library. It uses a small footprint description language to annotate libraries so they can be shared among PVS users with the help of the `pvsln` tool. This language is intended to document each library with the information of theories it offers and

the dependencies among them. This paper presents the description language in the form of BNF-like notation; its use is illustrated with actual running examples.

The `pvsln` tool is a command line tool written in the Python programming language. It can manage *any* (annotated) library that is publicly available from a *git* server through the internet. Once available, such a library source can be configured in `pvsln`, be automatically downloaded from the internet and set up in the host system. For internet downloads, the `pvsln` tool depends on *curl*, a command line tool commonly found in Linux and Mac OS X installations. The `pvsln` tool features support for updating, deleting, and re-installing the contents of a library source. Several library sources can be configured in `pvsln`.

The current distribution of `pvsln` automatically installs the latest version of the NASA PVS Library [3], a collection of formal PVS developments maintained by the NASA Langley Formal Methods Team. As a case study for the use of the `pvsln` tool, this paper presents a step-by-step guide on how to manually configure the NASA PVS Library comprising the command line instructions and snapshots of the interaction.

Paper outline. Section 2 presents the relevance of the tool for PVS users. Section 3 presents the annotation language for configuring a PVS library and introduces some assumptions on the internal structure of the library. Section 4 presents an overview of the tool's installation. Section 5 presents a step-by-step guide on the installation and management of the NASA PVS Library that is available from GitHub. Finally, Section 6 presents some concluding remarks.

2 `pvsln` Practical Value

This section describe the advantages of using `pvsln`, and its relevance for the PVS users.

Currently PVS users have to download the libraries and configure them manually, i.e. they have to get a local copy of the entire repository (*git*) and then move the sources to the `PVS_PATH`, every time the need to use a new library or update an old one. `pvsln` gives the advantage of managing several libraries (installing, updating or deleting) through the command line.

The ease of installing and keeping the packages up-to-date is the practical value of the tool. Since, once a library is configured, just a single command is needed to both update and configure the package and all their dependencies.

3 Library Configuration

This section describes the conventions and assumptions made by `pvsln` for managing PVS library sources.

Terminology. The `pvsln` tool distinguishes three levels of aggregation for PVS sources. A PVS theory is the building block of a library managed by `pvsln`. A *package* is a collection of theories. At the top level of aggregation is a *library*,

which comprises a collection of packages. In summary, a library is a collection of packages and a package is a collection of theories. The `pvs` tool can manage several libraries each with several packages.

Package configuration. Each package of a library is defined in a folder at the root of the library source. The folder's name defines the name of the package. Each folder contains the corresponding `pvs` and `prf` files for its theories, and the folder `pvsbin`, a special folder with the package's metadata included in a file named `top.dep`.

Package metadata. The metadata of a package is defined in its `top.dep` file located inside folder `pvsbin`. Table 1 presents the syntax of the metadata file in BNF-like notation. The topmost symbol is `<metadata>`, while `<theory>` and `<package>` are terminals representing, respectively, theory and package names. The metadata comprises two parts, namely, a header and a body. The header corresponds to a single line with an `'/'` symbol followed by a comma-separated list of theory names; these names correspond to the names of the theories included in the package (in any order). A body comprises any number of lines, each with either a package dependency or a theory dependency. A package dependency describes a dependency from another package and the list of theories from that package that are being used. A theory dependency describes, for each one of the theories listed in the header of the package, the list of its theory dependencies. In the case such a theory depends on a theory from other package, the name of that dependency must be qualified by the name of the corresponding package.

<code><metadata></code>	<code>::= <header> <body></code>
<code><header></code>	<code>::= '/' <theorylist></code>
<code><theorylist></code>	<code>::= <theory> <theory> ',' <theorylist></code>
<code><body></code>	<code>::= (<packagedep> <theorydep>)*</code>
<code><packagedep></code>	<code>::= <package> '/' <theorylist></code>
<code><theorydep></code>	<code>::= <theory> ':' <qualtheorylist>?</code>
<code><qualtheorylist></code>	<code>::= <qualtheory> <qualtheory> ',' <qualtheorylist></code>
<code><qualtheory></code>	<code>::= (<package> '@')? <theory></code>

Table 1: Syntax of the `top.dep` metadata file in NASALib.

Figure 1 presents an overview of the configuration file for package `trig` in NASALib. According to its header description, package `trig` defines theories `top`, `trig_doc`, `trig`, `trig_values`, etc. It contains 5 package dependencies and 6 theory dependencies. For instance, package `trig` depends on theory `for_iterate` in package `structures`, and on theories `finite_sets_minmax` and `finite_sets_inductions` in package `finite_sets`. On the other hand,

theory `trig_doc` has no dependencies, while theory `trig` depends on theories `trig_basic`, `sqrt`, `trig_values`, and `trig_ineq`. In the case of theory `sqrt`, it is explicitly stated that such a theory is in package `reals`.

```

1 /top,trig_doc,trig,trig_basic,trig_values,trig_ineq,trig_full,trig_extra
2 structures/for_iterate
3 reals/real_fun_preds,factorial,binomial,abs_lems,sign,sqrt_exists
4 analysis_ax/continuous_functions_props,derivatives,sqrt_derivative
5 finite_sets/finite_sets_minmax,finite_sets_inductions
6 ints/factorial
7 top:trig_doc,trig,trig_full,trig_basic,trig_values,trig_ineq
8 trig_doc:
9 trig:trig_basic,reals@sqrt,trig_values,trig_ineq
10 trig_basic:reals@sqrt
11 trig_values:trig_ineq
12 trig_ineq:trig_basic

```

Fig. 1: Overview of metadata file for package `trig` in NASALib.

4 Installation

This section presents an overview of the installation procedure.

The `pvslnm` tool can be installed automatically from the command line by issuing the following command:

```

curl http://migueleci.github.io/pvslnm/downloads/pvslnm-conf.py \
-o pvslnm-install && chmod +x pvslnm-install && \
python ./pvslnm-install

```

This command uses the `curl` utility to download the `pvslnm` installation sources from GitHub. Once these sources are downloaded and some file permissions adjusted, the installation process is executed as a Python 2 script. During the installation process, the user can select the location in which the tool is to be installed, including where the configuration files for the library sources and the local copy of the libraries are to be placed. This script has been tested both on Linux and Mac OS X boxes.

Upon its successful installation, `pvslnm` automatically configures the NASALib library sources and makes a local copy of them by using Git's clone command, so they are available for installation in PVS.

Further information of `pvslnm` is available on [5], e.g. a detailed list of the tool commands and some examples of the configuration files.

5 Case study: NASALib Configuration

This section presents a step-by-step guide on the configuration and installation with `pvslnm` of the NASA PVS Library (NASALib) that is available from GitHub.

Library source configuration. The first step is to issue a command for configuring the library sources as follows:

```
$ pvslm.py src -a \\  
    nasalib \\  
    'The NASA PVS Library is a collection of formal PVS developments \\  
    maintained by the NASA Langley Formal Methods Team.' \\  
    https://github.com/nasa/pvslib.git
```

Internally, this command generates the library source configuration file in the destination folder chosen during the installation process of `pvslm`.

It is important to note that this step is unnecessary with the current distribution of `pvslm` since it includes this source by default. However, this command is included in this section for the completeness of the example. Also note that it is not possible to have two or more library sources with the same name.

Cloning a library. After a library source is configured, it is possible to issue a command for cloning the source as a local *git* repository. The following command clones NASALib:

```
$ pvslm.py src -c nasalib
```

The effect of this command is to obtain an internal copy of the entire repository via *git*'s clone command by using the URL associated to the library source.

Updating a library. When necessary, it is possible to update the contents of a local copy of a library source. For the effect of updating the local copy of NASALib, a user can issue the following command:

```
$ pvslm.py src -u nasalib
```

The effect of this command is to update the (internal) local copy of the entire repository via *git*'s pull command by using the URL associated to the library source.

Listing the packages in a library. By having a local copy of a library, it is possible to list all its packages. Also, it is possible to list the dependencies of a given package. The following command lists the package dependencies of package `complex` from NASALib:

```
$ pvslm.py pkg -l nasalib@complex
```

This command generates the following output:

```
Package complex depends on:  
algebra  
analysis_ax  
ints  
lnexp  
reals  
structures  
trig
```

Installing a package. The following command installs package `complex` in PVS.

```
$ pvslm.py pkg -i nasalib@complex
```

Since package `complex` depends on other packages, the `pvslm` tool also asks for permission to install its dependencies. The following is the output to the user for this installation command.

Package `complex` depends on:

...

Would you like to install the package(s) (y/N):

By convention, `pvslm` performs *all* installations in the folder specified by the `PVS_PATH` global variable.

Deleting a package. Finally, a user can issue the following command to delete package `ints` from the (internal) local copy of NASALib:

```
$ pvslm.py pkg -d nasalib@ints
```

If there are packages that depend on the package to be removed, the `pvslm` tool lists all of them and asks the user for authorization to remove them too.

6 Concluding Remarks

This paper presented the `pvslm` tool for managing libraries for the Prototype Verification System (PVS). This tool features support for different library sources, libraries with several theories, and dependencies among the theories (within the same library source). The tool is freely available for download and it is distributed under GNU's GPLv3 license. It uses a small footprint language for annotating libraries, which is described in full detail in BNF-like notation in this paper. This paper also includes all commands available from the tool, its architecture, and an overview of its installation process. A detailed step-by-step case study is included for illustrating the main features of the tool.

As usual, much work remains to be done. First, it is important to make available other library sources through the `pvslm` tool. Also, it is important to test the tool against different servers and in more operating systems. Finally, it would be highly desirable for the tool to manage different versions of a library source, which can depend on the installed versions of PVS in the host system. This will likely require an extension of the current metadata language for annotating library sources.

Acknowledgments. The authors would like to thank C. Muñoz from the NASA Langley Formal Methods Team for his encouragement, ideas, and suggestions, specially for the help with the definition of the metadata description language for packages.

References

1. J. Avigad and J. Harrison. Formally verified mathematics. *Communications of the ACM*, 57(4):66–75, Apr. 2014.
2. T. C. Hales. Developments in formal proofs. *CoRR*, abs/1408.6474, 2014.
3. NASA Langley Formal Methods Team. NASA PVS library. <http://shemesh.larc.nasa.gov/fm/ftp/larc/PVS-library/>, June 2015. [Online; accessed 14-June-2015].
4. S. Owre, J. M. Rushby, , and N. Shankar. PVS: A prototype verification system. In D. Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752, Saratoga, NY, jun 1992. Springer-Verlag.
5. M. Romero and C. Rocha. A PVS library manager. <http://migueleci.github.io/pvslm/>, June 2015. [Online; accessed 14-June-2015].