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 pvslm, 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 pvslm 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 pvslm for the configuration of the current version of the NASA PVS Library, available from GitHub.

1 Introduction

The Prototype Verification System [**?**] (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 [**?**,**?**], it is important to have tools for managing these libraries. This paper presents pvslm [**?**], an open source tool for managing PVS libraries that is freely available for download [**?**] under the GNU General Public License GPLv3.

The pvslm 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 pvslm 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 pvslm 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 pvslm, and be automatically downloaded from the internet and set up in the host system. For internet downloads, the pvslm tool depends on *curl*, a command line tool commonly found in Linux and Mac OS X installations. The pvslm tool features support for updating, deleting, and re-installing the contents of a library source. Several library sources can be configured in pvslm.

The current distribution of pvslm automatically installs the latest version of the NASA PVS Library [**?**], a collection of formal PVS developments maintained by the NASA Langley Formal Methods Team. As a case study for the use of the pvslm 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 **??** presents the annotation language for configuring a PVS library and introduces some assumptions on the internal structure of the library. Section **??** presents an overview of the tool's installation and its distributed architecture. Section **??** includes the list of commands available from pvslm. Section **??** presents a step-by-step guide on the installation and management of the NASA PVS Library that is available from GitHub. Finally, Section **??** presents some concluding remarks.

2 Library Configuration

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

Terminology. The pvslm tool distinguishes three levels of aggregation for PVS sources. A PVS theory is the building block of a library managed by pvslm. 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 libraries. The pvslm 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`. Figure **??** depicts the structure of package `analysis` from the NASA PVS Library [**?**] (NASALib).

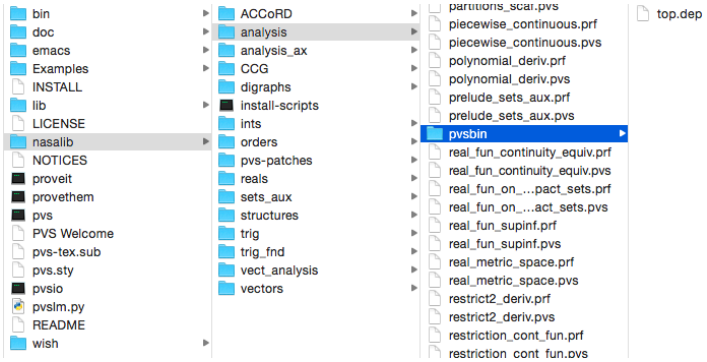

Fig. 1: The structure of a package `analysis` in NASALib.

Package metadata. The metadata of a package is defined in its `top.dep` file located inside folder `pvsbin`. Table **??** 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.

⟨`metadata`⟩	::=	⟨`header`⟩ ⟨`body`⟩
⟨`header`⟩	::=	'/' ⟨`theorylist`⟩
⟨`theorylist`⟩	::=	⟨`theory`⟩ | ⟨`theory`⟩ ',' ⟨`theorylist`⟩
⟨`body`⟩	::=	(⟨`packagedep`⟩ | ⟨`theorydep`⟩)*
⟨`packagedep`⟩	::=	⟨`package`⟩ '/' ⟨`theorylist`⟩
⟨`theorydep`⟩	::=	⟨`theory`⟩ ':' ⟨`qualtheorylist`⟩?
⟨`qualtheorylist`⟩	::=	⟨`qualtheory`⟩ | ⟨`qualtheory`⟩ ',' ⟨`qualtheorylist`⟩
⟨`qualtheory`⟩	::=	(⟨`package`⟩ '@')? ⟨`theory`⟩

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

Figure ?? 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 7 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`.

```
/top,trig_doc,trig,trig_basic,trig_values,trig_ineq,trig_full,trig_extra
structures/for_iterate
reals/real_fun_preds,factorial,binomial,abs_lems,sign,sqrt_exists
analysis_ax/continuous_functions_props,derivatives,sqrt_derivative
finite_sets/finite_sets_minmax,finite_sets_inductions
ints/factorial
top:trig_doc,trig,trig_full,trig_basic,trig_values,trig_ineq
trig_doc:
trig:trig_basic,reals@sqrt,trig_values,trig_ineq
trig_basic:reals@sqrt
trig_values:trig_ineq
trig_ineq:trig_basic
```

Fig. 2: Overview of metadata file for package `trig` in NASALib.

3 Installation and Architecture

This section presents an overview of the installation procedure and the distributed architecture of pvslm.

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

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

This command uses the *curl* utility to download the pvslm 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. Figure ?? depicts a successful installation procedure of pvslm in a Ubuntu Linux box.

Upon its successful installation, pvslm 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.

```
miguel@miguel-VM:~/PVS$ curl http://migueleci.github.io/pvslm/downloads/pvslm-conf.py -o pvslm-insta
ll && chmod +x pvslm-install && python ./pvslm-install
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  4007  100  4007    0     0  23167      0 --:--:-- --:--:-- --:--:-- 23296
Enter the library manager installation path (default=/home/miguel/PVS):
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 15165  100 15165    0     0  61370      0 --:--:-- --:--:-- --:--:-- 61396
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   290  100   290    0     0   1423      0 --:--:-- --:--:-- --:--:--  1428
Cloning into '/home/miguel/PVS/.pvslm/repos/nasalib'...
remote: Counting objects: 4349, done.
remote: Total 4349 (delta 0), reused 0 (delta 0), pack-reused 4349
Receiving objects: 100% (4349/4349), 17.35 MiB | 496.00 KiB/s, done.
Resolving deltas: 100% (2425/2425), done.
Checking connectivity... done.
Checking out files: 100% (3024/3024), done.
Setting PVSPATH in the "pvs", "pvsio", "proveit", and "provethem"
shell scripts to /home/miguel/PVS
If this is not the right path, edit the scripts before using them.
PVS Library Manager has been successfully configured. Thanks!
miguel@miguel-VM:~/PVS$
```

Fig. 3: A successful installation procedure of pvslm in a Ubuntu Linux box.

The pvslm tool is desiged with a distributed architecture. It can connect to library sources over the internet. Each time a source is configured, pvslm can download the library into the host system as a local *git* repository. Further updates of the library are carried out internally by pvslm by using *git*'s pull command. Figure **??** depicts the architecture of pvslm. It is important to note that although GitHub is used as *git* server of reference throughout this paper, it is also possible to use other publicly available servers such as BitBucket, for instance, containing an annotated library.

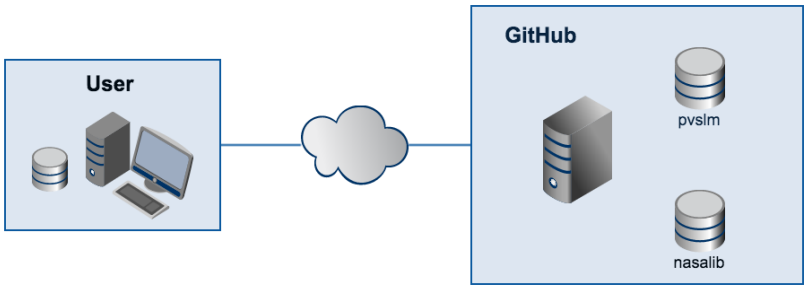

Fig. 4: Distributed architecture of pvslm.

4 Available Commands

This section presents the list commands available from the pvslm tool. Section ?? presents examples on the use of some of these commands.

The pvslm tool provides commands at two levels. First, it provides commands at the level of library sources for managing library *git* repositories. Second, it provides commands at the level of packages for managing the contents of a library. Table ?? lists all commands available from pvslm.

Level	Command	Parameters	Description
src	-a	name desc URL	Add a new library source with the given name, description, and URL.
	-d	name	Delete the given library source.
	-c	name	Clone the given library source.
	-u	name	Update the given library source.
	-r	name	Remove the clone of the given library source.
pkg	-i	library@package	Install and update the given package, and all its dependencies.
	-u	library@package	Update a package and all its dependencies.
	-d	library@package	Delete a package and all ones depending on it.
	-l		List the installed libraries.
	-l	library	List the available packages of the given library.
	-l	library@package	List all the dependencies of the given package.

Table 2: pvslm command list.

At the level of library sources, the tool provides 5 different commands, all identified with the special token `src`. They include commands for:

- Adding a library source (i.e., command `-a`) with a name, a short description, and a *git* URL.
- Deleting a library source (i.e., command `-d`) with the given name.
- Cloning a library source (i.e., command `-c`) with the given name.
- Updating a library source (i.e., command `-u`) with the given name.
- Removing the clone of a library source (i.e., command `-r`) with the given name.

It is important to note that none of the commands at the library source level modify the user's PVS installation. These commands exclusively modify the internals of the pvslm configuration. Also, note that a library source is realized exactly by one *git* repository.

At the level of packages, the tool provides 4 different commands, all identified with the special token `pkg`. They include commands for:

- Installing and updating a given package (i.e., command `-i`) from a given library source, including *all* its dependencies.
- Updating a given package (i.e., command `-u`) from a given library source, including *all* its dependencies.
- Deleting a given package (i.e., command `-d`) from a given library source (local copy), including all packages that depend on it.
- Listing the contents (i.e., command `-l`) from a given library source.

Note that listing command has three variants. In the first case, all libraries available to the system are listed. In the second case, all packages of a given library are listed. In the third case, all dependencies of a given package and library are listed. Internally, the pvslm uses a topological sorting algorithm, based upon each package's metadata, for computing the set of dependencies among theories and packages.

5 Case study: NASALib Configuration

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

Library source configuration. The first step is to issue a commnad 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 following library source configuration file in the destination folder chosen during the installation process of pvslm:

```
Name = nasalib
Description = The NASA PVS Library ...
URL = https://github.com/nasa/pvslib.git
Enable = Yes
```

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 the package to be removed has dependencies, the 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 libaries 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. The tool 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.