Library Management for PVS

Miguel Romero, Camilo Rocha

***Abstract*—The Prototype Verification System (PVS) is a specification language integrated with support tools and a theorem prover. One mechanism for extending and improving PVS is through the development of libraries, i.e., collections of mechanized theories in the language of PVS. This paper presents the pvslm tool: (i) a description language for annotating PVS libraries, with a small footprint but expressive enough for describing complex dependencies among library theories; and (ii) an implementation for managing PVS libraries annotated with this language, offering support for several library sources. The usefulness of the tool is illustrated in this paper with a detailed step-by-step guide on how to manage the current version of the NASA PVS Library with both pvslm's annotation language and implementation.**

***Index Terms*—**

I. Introduction

The Prototype Verification System [1] (PVS) is a verification system comprising a specification language integrated with support tools and a theorem prover; basically, PVS is a mechanization of classical typed higher-order logic with specifications organized into parameterized theories in this logic. The PVS system has been used in state-of-the-art formal methods projects as a productive environment for constructing and maintaining collections of theories, both in industry and in research. These novel developments often result in large collections of PVS definitions, theorems, and proofs, grouped into libraries, with many dependencies among them. This is the case, for instance, of the NASA PVS Library [2] maintained by the NASA Langley Formal Methods Team: a large collection of freely-available formal developments in PVS comprising more than 40 theories, 24000+ theorems, and 136000+ and 3623000+ lines of, respectively, specification and proofs, all of theses ranging from trigonometry to graphs and topology. In general, given the increasing size of mathematical developments in the form of libraries and the growing need for and adoption of mechanized proof environments [3], [4] such as PVS, it is important to have tools for managing such libraries.

This paper presents pvslm [5], an utility for assisting in the management of PVS libraries, comprising: (i) a description language for annotating PVS libraries, with a small footprint but expressive enough for describing complex dependencies among library theories; and (ii) an implementation for managing PVS libraries annotated with this language, offering support for several library sources. In pvslm, the description language is designed to represent a PVS library as a collection of packages, with a package consisting of a collection of PVS theories. In this sense, the annotations can help in documenting each library with the information of the theories it offers and their dependencies, so it can be shared among PVS users with the help of the pvslm implementation. The pvslm implementation is a command line tool offering support for adding, updating, deleting, and re-installing the contents of a library source, and also managing several library sources at the same time.

The pvslm description language is presented in this paper with the help of BNF-like notation. The pvslm implementation is written in the Python programming language and it can manage *any* (annotated) PVS library that is publicly available from a pvslm server through the internet: once available, such a library source can be configured in pvslm, be automatically downloaded from the internet, and set up in the host system. 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 user interaction.

The current distribution of `pvslm` is freely available for download [5] under the GNU General Public License GPLv3 and it automatically installs the latest version of the NASA PVS Library, which is currently annotated with the pvslm description language.

a) Paper outline.: Section II presents the pvslm description language. Sections III and IV present, respectively, instructions for obtaining and installing the pvslm implementation, and a step-by-step guide on the configuration and installation with pvslm of the NASA PVS Library. Some concluding remarks are presented in Section V.

II. The pvslm Description Language

This section presents a formal definition of the pvslm description language in the form of BNF-like notation. It also presents the main conventions and assumptions used by the pvslm tool for managing PVS libraries.

a) 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. A *library* is at the top level of aggregation, comprising a collection of packages. In summary, a library is a collection of packages and a package is a collection of theories. The pvslm tool can manage several libraries each with several packages.

b) Package configuration.: A package is defined in a folder at the root of the library source, with the folder name defining the name of the package. Each folder contains the `pvs` and `prf` files for its theories, and a folder named `pvsbin`: this is a special folder used by the pvslm implementation for accessing the *package metadata* from a file named `top.dep`.

c) Package metadata.: The metadata of a package is defined in its `top.dep` file, located inside folder `pvsbin`. Table I presents the syntax of pvslm description language, in BNF-like notation, used to populate this metadata file.

⟨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 I: Syntax of the `top.dep` metadata file in NASALib.

The topmost symbol in the description language 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 depended upon. 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 its package.

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`.

III. Obtaining the PVSLM Implementation

The pvslm tool can be installed automatically from the terminal in *nix systems 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 script is executed as a Python 2 program. During the installation process, the user can select the value for global variable `PVS_PATH`, i.e., 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 procedure and the installation script have been tested both on Linux and Mac OS X boxes.

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

IV. Case study: Managing NASALib with PVSLM

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.

a) 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 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 because it installs the NASALib library source by default. However, this command is included in this section for the sake of a complete installation example. Also note that it is not possible to have two or more library sources with the same name.

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

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

This command internally uses *git*'s clone command to obtain a local copy of the entire library source repository by using the URL configured with the library source.

```
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.

c) 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 local copy of the entire library via *git*'s pull command by using the URL associated to the library source.

d) 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
```

e) 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.

f) 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.

V. 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 PVS library sources with the help of 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, each configured to work on different versions of PVS installed in the host system. This will require an extension of the current pvslm description language for annotating library sources.

Acknowledgment

The authors would like to thank C. Muñoz in 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] S. Owre, J. M. Rushby, and N. Shankar, "PVS: A prototype verification system," in *11th International Conference on Automated Deduction (CADE)*, ser. Lecture Notes in Artificial Intelligence, D. Kapur, Ed., vol. 607. Saratoga, NY: Springer-Verlag, jun 1992, pp. 748–752. [Online]. Available: http://www.csl.sri.com/papers/cade92-pvs/

[2] 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]. [Online]. Available: http://shemesh.larc.nasa.gov/fm/ftp/larc/PVS-library/

[3] J. Avigad and J. Harrison, "Formally verified mathematics," *Communications of the ACM*, vol. 57, no. 4, pp. 66–75, Apr. 2014. [Online]. Available: http://doi.acm.org/10.1145/2591012

[4] T. C. Hales, "Developments in formal proofs," *CoRR*, vol. abs/1408.6474, 2014. [Online]. Available: http://arxiv.org/abs/1408.6474

[5] M. Romero and C. Rocha, “A PVS library manager,” http://migueleci.github.io/pvslm/, June 2015, [Online; accessed 14-Feb-2016]. [Online]. Available: http://migueleci.github.io/pvslm/