pvslm: A Library Manager for the Prototype Verification System

Miguel Romero[1], Camilo Rocha[1], and César Muñoz[2]

Escuela Colombiana de Ingeniería, Bogotá, Colombia
NASA Langley Research Center, Hampton VA, USA

Abstract.

1 Introduction

2 PVSLM

This section describes the design and architecture of PVS Library Manager.

Before to describe and analyze the tool, is necessary to define the structure used. The smallest part are the theories, which are organized in packages. Those packages are the main element for the tool. Finally there are sets of packages, called libraries. Furthermore, it is necessary to specify the connections or relations within a library, these are the dependencies between its packages. So every package must have a dependencies list defined as the union of the dependencies of its theories.

PVSLM has been developed to manage the installation and update of packages from a PVS library. It has been divided into two sections, the first one manage the source and repository for each library, and the second one manage the packages.

The source section is divided into two parts: the configuration file and the repository. The configuration file contains the information about the library as the name, description and the URL for the git repository where it can be download. It is important to recall that the library must be available on a git server as GitHub or BitBucket. The repository is a clone of a git, that can be updated or deleted. This section is completely dedicated to the configuration of the libraries.

The other section manage the packages, their uninstall, update, list and installation, based on the already configured libraries. PVSLM is based on the dependencies graph of the library, therefore all the modifications on a package will affect the adjacent ones.

As the tool has been developed for the PVS NASA library (nasalib), this one is configured by default. All the commands available on PVSLM are listed along with their description on Table 1.

PVSLM works locally, but it needs Internet connection for its installation and the communication with the git repositories. Basically it is downloaded and

Section	Command	Parameters	Description
src	-a	name description URL	Create a new .list file with the library information.
	-d	name	Delete the library .list file.
	-c	name	Create the library repository based on the source with the same name. The creation is a clone of the git repository.
	-u	name	Update the library using a pull command.
	-r	name	Remove the repository.
pkg	-i	library@package	Install and update the package and all its dependencies.
	-u	library@package	Update an installed package and all its dependencies.
	-d	library@package	Delete an installed package with all ones that depends on it.
	-l		List all the installed libraries.
	-l	library	List all the available packages of the library.
	-l	library@package	List all the dependencies of the package.

Table 1: PVS Library Manager command list

then configured, once it is installed there are several commands that needs to connect to the git repository either to clone or pull it. For example, when a package is going to be installed, it is updated along with all its dependencies. The Figure 1 shows how the tool works.

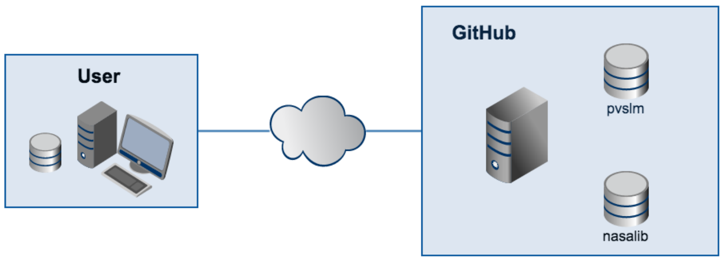


Fig. 1: PVS Library Manager architecture

In the Figure 2 the architecture of PVSLM is explained. In 1 the client sent a request to the Git repository where the tool is available. Then in 2 the Git server allows the download of the files required for the installation on step 3. In the installation, nasalib is requested in 4 to the Git server, and it is downloaded in 5. Finally in 6 the library is configured locally in the client machine. The installation and configuration of a library are reflected on steps 4 to 6.

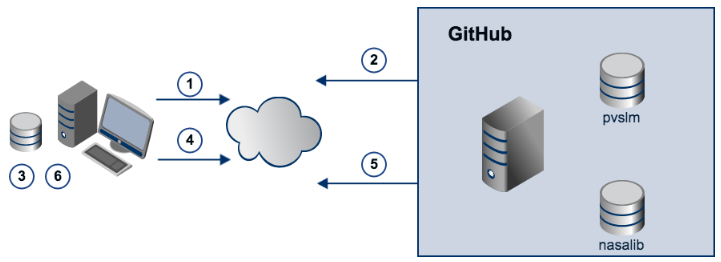


Fig. 2: PVS Library Manager architecture process

3 Library Configuration

The requirements of PVSLM are: curl PVS

Every package must have a dependencies file, where the relations between its theories and other packages are specified. The file has to be named "top.dep" and be located within the package in a folder named "pvsbin".

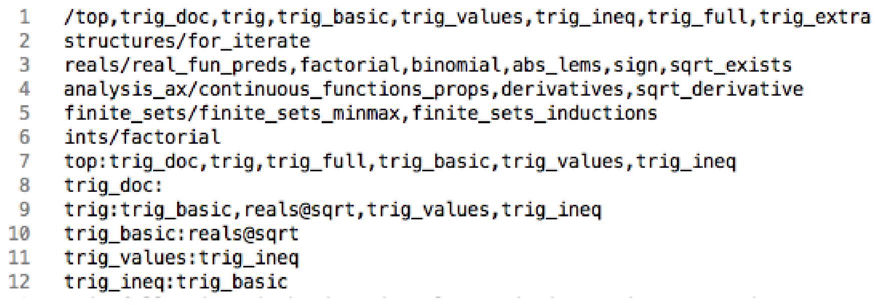
The syntax for the dependencies file is described below on Table **??**.

BNF: Dependencies syntax PVS Library

<part_one>	::=	"/" <theories_list>
<theories_list>	::=	<theory> | <theory> "," <theories_list>
<part_two>	::=	<package_dep> "/" <theories_dep>
<theories_dep>	::=	<theory_dep> | <theory_dep> "," <theories_dep>
<part_three>	::=	<theory> ":" <dependencies_pkg>
<dependencies_pkg>	::=	<package_dep> "@" <theory_dep> | <package_dep> "@" <theory_dep> "," <dependencies_pkg>

There are three parts in the syntax. The first one lists the theories of the package, this line must begin with a backslash '\' followed by the theories separated by commas. The second part is the list of the package and theories with which there are dependency relations, one package per line. Each line begins with the package name, then a backslash '\' and the theories separated by commas. This is the most important section of the file for PVSLM, because of its meaning.

Finally there is the list of relations between the theories, no matter what package are them from. Each line begins with the name of the theory followed by colon ':' and the list of theories with which there is relation, if the the theory is from another package it is necessary to include the name of the package followed by the symbol '@' before the name of the theory, otherwise the name of the theory is enough.



```
/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. 3: Dependencies file syntax example

In the Figure 3 there are examples for each of the cases explained above. The example is taken from a file of nasalib.

4 Case study: Nasalib configuration and installation

This section shows an example step by step of the configuration and installation of a library in PVSLM. Specifically, NASA PVS Library (nasalib) is going to be used in the case study.

1. The first step is the creation of the configuration file for the library. To reach it, it is necessary to assign a codename to the library that will be used afterward. Along with the codename, the description of the library and its URL are necessary. The URL belongs to an active git repository. The next command will create the file with all the attributes into it.

   ```
   $ pvslm.py src -a nasalib 'description' url
   ```

2. Then it is necessary to create the repository for the library. From the already created file, the URL is used to create a clone of the git repository. Besides,

within the PVS path is created a folder with the codename for the library.

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

3. The library is already configured with the commands described above. Now it can be used; namely, it is possible to manage all its packages. The first step is looking up the available packages and its dependencies. There are two commands to accomplish it, described below. The first one will list all the available packages within the library in lexicographic order; and the second one will list all the dependencies of the package, also in lexicographic order. Finally there is a third command that list the configured and available libraries.

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

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

   ```
   $ pvslm.py pkg -l
   ```

4. When installing a package there are two activities executed at the same time. When the command is executed a list with all the dependencies of the package will be shown, along with a confirmation message. Once the user accept to install the package with its dependencies, all of them are going to be updated before copying to the PVS path. As mentioned before, the installation is a copy of the packages from its repository (clone) to the PVS path.

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

5. To update a library without executing any extra activity, the following command will proceed a pull from the git repository specified in the configuration file.

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

6. Finally, to delete a package it is necessary to keep in mind the dependencies, i.e. if a package its deleted, it is necessary to delete every package that uses or depends on it. As the installation command, before the deletion is process a list with all the dependencies is going to be displayed along with a confirmation message. And example of the command is showed below.

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

5 Concluding Remarks