Appendix D

SPOTS System Guide

D.1 Installation and User Guide

The SPOTS/PVS framework is based on the PVS system. In our setup, we have used PVS v3.2 which is available at [5]. PVS uses GNU Emacs an interface. PVS Lisp runs as an inferior lisp to Emacs Lisp. In the implementation of the verification conditions generator (VCGEN), we use interprocess communication between Emacs Lisp and PVS Lisp. In our setup, we have used GNU Emacs v21.2.1 which is available at [4]. The source of the SPOTS system is available at [1]. Below we outline the installation of the system.

Installation of the SPOTS/PVS system

(1) Download and install PVS v3.2 (or later).

(2) Download and untar/unzip `spotspvs.tgz` in a suitable directory. It contains `TTL`, `OVerification`, and `specifications` directories. `TTL` and `OVerification` directories contain PVS theories described in Section 9.1. `specifications` directory contains specifications of several optimizations viz. common subexpression elimination (`cse`), dead code elimination (`dce`), partial dead code elimination (`pdce`), lazy code motion (`lcm`), loop invariant code motion (`licm`), optimal code placement (`ocp`). It also contains a directory (`testsuite`) containing some test cases specified as PVS theories for validation of optimization specifications.

(3) Invoke PVS in `TTL` directory. Parse and typecheck all the theories in the directory.

(4) Change the context to `OVerification` directory using a PVS command `change-context`. The theories in `OVerification` directory use concepts defined in `TTL`

directory. We therefore import the theories in `TTL` directory by a PVS command `load-prelude-library`. The details about all PVS commands are available at [64].

(5) Parse and typecheck all theories in `OVerification` directory.

(6) Perform the following steps for each of the sub-directories in `specifications` directory: Change the context to a directory containing an optimization specification. Import the theories in `OVerification` directory using PVS command `load-prelude-library`. Parse and typecheck the theories in the directory.

(7) Change the context to `testsuite` directory. Import the theories in each of the optimization specifications. Parse and typecheck the theories in the directory.

Using the SPOTS/PVS system

The verification conditions for an optimization specification can be generated by invoking `vcgen` from the PVS Emacs interface. However, as mentioned in Section 9.2, the present implementation of VCGEN does not handle all the PVS language features. Proofs of the verification conditions can be derived using the PVS proof checker. For some specifications, supporting lemmas (`<opt>_lemmas.pvs`) and proofs are also provided with the system which can simply be checked in PVS.

An optimization specification can be validated on tests encoded as PVS theories in `testsuite` directory as follows:

(1) Parse and typecheck a test theory.

(2) Invoke the PVS ground evaluator in the context of the theory.

(3) Evaluate analysis or transformation functions defined as part of the specification on the program with appropriate arguments in the ground evaluator to check their results.

(4) Check verification conditions for the specification in the ground evaluator. The verification conditions should evaluate to `true`.

Installation of the SPOTS/GCC system

(1) Install the SPOTS/PVS system as described earlier.

(2) Download and untar/unzip GCC v4.1.0 source from [3]. Let `srcdir` denote the base directory of GCC source.

(3) Download and untar/unzip spotsgcc.tgz from [1]. Let `spotsgcc` denote the base directory. It contains `gcc-4.1.0`, `scripts`, and `tests` directories. `gcc-4.1.0` directory contains instrumented GCC source files. `scripts` directory contains AWK scripts for processing traces generated by the instrumented GCC as described in Chapter 10. `tests` directory contains test cases for validation of GCC.

(4) Copy `Makefile.in` and `print-spots-rtl.c` files from `spotsgcc/gcc-4.1.0` to `srcdir` directory.

(5) Create an installation directory for GCC, say `builddir`. Go to `builddir` directory and configure GCC installation as `srcdir/configure --enable-languages=c`. Make the GCC source by invoking command `make`. The resulting GCC compiler is a compiler for the C language and is not instrumented.

(6) Copy the instrumented GCC source files from `spotsgcc/gcc-4.1.0` directory to `srcdir`.

(7) Invoke `make` in `builddir`. The resulting compiler is the required instrumented version of GCC.

Using the SPOTS/GCC system

(1) Go to `spotsgcc/tests` directory.

(2) Compile a test C program using the instrumented GCC as follows:
```
builddir/gcc/cc1 -O1 test.c
```
It generates a `test.c.spots` file. In place of `O1`, optimization flags viz. `O2`, `O3`, and `Os` can also be used.

(3) Generate a PVS file for the SPOTS/GCC file as follows:
```
awk -f spotsgcc/scripts/spots-heuristics.awk
     -f spotsgcc/scripts/spots-pvs.awk test.c.spots > test.c.pvs
```

(4) Generate a DOT file for the SPOTS/GCC file as follows:

```
awk -f spotsgcc/scripts/spots-heuristics.awk
     -f spotsgcc/scripts/spots-dot.awk test.c.spots > test.c.dot
```

Generate a PS file from the DOT file for visualization:

```
dot -Tps test.c.dot > test.c.ps
```

(5) Invoke PVS and import the PVS theories from `OVerification` directory using PVS command `load-prelude-library`.

(6) Parse and typecheck `test.c.pvs`. Use the PVS ground evaluator to validate the GCC optimization. In order to check whether loops detected by GCC in loop optimizations are correct (Section 10.5), import `specifications/licm` directory which contains a formal definition of a loop.