User's Manual for Psd

for version 1.0

Pertti Kellomäki, pk@cs.tut.fi
Tampere University of Technology
Software Systems Lab
Finland

July 9, 1992

1 Introduction

This is the user's manual for psd, the Portabe Scheme Debugger. Psd is a source level debugger for the Scheme language, based on istrumenting the original source code. Psd is placed under the GNU General Public License ("copyleft"), so you are free to use, distribute and modify it as long as you let other people do it, too. See the file `COPYING` in the psd distribution for more details.

2 What You Need

In order to use psd you need GNU Emacs, a (preferably R4RS [1] compliant) Scheme interpreter, the psd package and the cmuscheme package. GNU Emacs and the cmuscheme package can be found at many major ftp sites, for example nic.funet.fi. These ftp sites also carry Scheme interpreters. A good place to look for Scheme related material is the Scheme Repository at nexus.yorku.ca, maintained by Ozan Yigit. Psd is available at cs.tut.fi using anonymous ftp (`/pub/src/languages/schemes/psd.tar.Z`).

3 Supported Systems

Psd is known to work with Aubrey Jaffer's scm and Oliver Laumann's elk. It is known not to work with with sci, the interpreter in the Scheme to C compiler system. It should be easy to modify psd to work with sci, though, as the main problem is the default case for symbols.

It may be necessary to do some initializations in order to make psd work with a given Scheme implememtentation. For example, for elk it is necessary to set the variables `print-length` and `print-depth` to a negative value. In order

to accomplish this, the psd startup code looks for a file called `psd-`*name*`.scm`, where *name* is the value of the Emacs variable `scheme-program-name`. If that file exists, it is loaded, otherwise the file `psd.scm` is loaded.

4 Preparing Programs for Debugging

The easiest way to use psd is to use the GNU Emacs interface. The interface requires the cmuscheme package, which you should probably have even if you don't use psd. You can check if you are using cmuscheme by starting up an inferior Scheme session and giving the command `M-x eval-expression RET (featurep 'cmuscheme) RET`. If the result is `t` in the minibuffer, you are using cmuscheme. To load the cmuscheme package, use the command `M-x load-library RET cmuscheme`. You can automagically load cmuscheme every time you start Emacs by putting the line

```
(require 'cmuscheme)
```

in your `.emacs` file.

To use the interface, start up a Scheme session with `M-x run-scheme`. Then give the command `M-x psd-mode` in the Scheme buffer. If this does not work, you have to load the file `psd.el` with `M-x load-file`. The necessary Scheme code is now loaded into your Scheme session. If you want psd to be loaded when you start up a Scheme session, put the line

```
(setq inferior-scheme-mode-hook '(lambda () (psd-mode 1)))
```

in your `.emacs` file.

To debug all procedures in a Scheme source file, give the command `C-c d` or `M-x psd-debug-file` either in the inferior Scheme buffer, or in a buffer containing Scheme source. If everything goes well, psd will first produce an instrumented version of your file, and then load it. You can also go to a Scheme buffer and use the commands `ESC C-x`, `C-c e` and `C-c C-e`. If given a prefix argument, they will instrument the definition the cursor is on and load it into Scheme. For example, the command `C-u C-e` will instrument a definition and send it to the Scheme process.

If you have code in your file that is evaluated in load time, for example variable initializations, you may end up being in the psd prompt after one of these commands. Just step thru the initializations, and you will end up at the top level.

5 Running the Debugged Program

After the instrumentation is complete, you have in your Scheme environment top level definitions equivalent to those that you would have gotten if you had loaded

the original source file. However, all the procedures have been instrumented so that you can step thru the evaluation process, and see in another Emacs window where in the source code you are.

The debugged procedures can be invoked either from top level or from other procedures. In order to stop the execution to a specific line, go to the line and give the command `M-x psd-set-breakpoint` or `C-x SPC`. This will set a breakpoint to the specified line.

6 Debugger Commands

The current debugger top level loop is not very sophisticated, but undoubtedly it will have more commands in the future. If you have suggestions of what would be useful, feel free to contact me. The commands are:

val *sym* gives the value of *sym* in the current scope.

set! *sym val* sets the value of *sym* to *val* in the current scope.

g continues evaluation until the next breakpoint

w shows the current context as file name and a list of procedure names. For example, `"/tmp/killme.scm:(encode encode-symbol)"` means, that the you are in file `/tmp/killme.scm`, inside a procedure called `encode-symbol`, which is inside the procedure `encode`.

s steps one step in the evaluation process. Each time an expression is about to be evaluated, psd displays it and waits for a command. When an expression has been evaluated, psd displays the result and waits for a command.

n continues evaluation until evaluation reaches a different line

r *expr* evaluates expr and returns its value as the return value of the current expression

A list is taken to be a procedure call that is to be evaluated. All the essential procedures in R4RS are visible to the evaluator. Any other command displays a list of available commands.

If the debugger does not seem to be doing the right things, try the Emacs command `M-x psd-reset`, which will clear all the breakpoints and reset the runtime system.

7 Catching Run Time Errors

Psd tries to detect any run time errors before they are encountered. Before every call to an essential procedure in R4RS, it checks the number of arguments and their types. If a run time error would occur, psd stops execution and lets the user inspect and modify the environment. In order to continue, use the debugger command `r` and give a value that is to be returned as the result of the call. This works of course only if the code where the call is made from is being debugged.

8 Limitations of the Current Implementation

The current version handles all syntactic forms except `=>`, `delay` and unquoting. Unquoting is supported in the sense that procedures containing quasiquote and unquotations can de debugged, but it is not possible to step thru an unquotation, or set a breakpoint within a quasiquotation.

Macros are not supported at all. It would require that the instrumentation code would have to keep track of all macro definitions and be able to expand macros in their full glory.

The reader understands symbols, boolean values, strings, vector, characters, integers, simple floats and lists. Fancier numbers like complex numbers etc. are not supported. They are not very hard to implement, they are just not on top of the priority list for me. Hex, octal and binary numbers do work, though, thanks to Edward Briggs.

One thing that psd is not guaranteed to preserve is the order of evaluation. Because of the additional code that psd adds to the program, it is possible that the instrumented version of a procedure call is evaluated in a different order than the original. If the Scheme implementation used evaluates all arguments from left to right or right to left, there is no problem. If, however, the order of evaluation is something more exotic, the order of evaluation may change. In practice this is probably not a problem.

Because the instrumented programs and the runtime support for the debugger live in the same name space, there are some names that can not be used in the debugged programs. In psd, all the globally visible procedures start with the prefix `psd-`, and variables with the prefix `*psd-`. Do not use these prefixes in your programs.

9 Command Summary

This is a short list of the available commands. The Emacs commands are:

`C-c d`	psd-debug-file
`C-c C-e`	scheme-or-psd-send-definition-and-go
`C-c e`	scheme-or-psd-send-definition
`C-x SPC`	psd-set-breakpoint
`ESC C-x`	scheme-or-psd-send-definition
`M-x psd-reset`	clear all breakpoints and reset the psd runtime

The debugger commands are:

`val` *sym*	give the value of *sym*
`set!` *sym val*	set the value of *sym* to *val*
`g`	run until the next breakpoint
`w`	give the current context
`s`	step one step in the evaluation process
`n`	run until evaluation reaches another line
`r expr`	return expr as the value of current expression

A list is taken to be a procedure call to be evaluated. It can also be a `set!` form.

10 Acknowledgements

Thanks go to Edward Briggs and Aubrey Jaffer for their suggestions and modifications to psd. Also, special thanks go to Tatu Männistö for lively discussions about psd and for keeping up the Scheme spirit in the department.

References

[1] Jonathan A. Rees and William Clinger, editors. The revised^4 report on the algorithmic language Scheme. *LISP Pointers*, IV(3):1–55, July–September 1992.