Coccinelle Usage

December 31, 2008

1 Introduction

This document describes the options provided by Coccinelle. The options have an impact on various phases of the semantic patch application process. These are:

1. Selecting and parsing the semantic patch.
2. Selecting and parsing the C code.
3. Application of the semantic patch to the C code.
4. Transformation.
5. Generation of the result.

One can either initiate the complete process from step 1, or to perform step 1 or step 2 individually.

Coccinelle has quite a lot of options. The most common usages are as follows, for a semantic match `foo.cocci`, a C file `foo.c`, and a directory `foodir`:

- `spatch -parse_cocci foo.cocci`: Check that the semantic patch is syntactically correct.
- `spatch -parse_c foo.c`: Check that the C file is syntactically correct. The Coccinelle C parser tries to recover during the parsing process, so if one function does not parse, it will start up again with the next one. Thus, a parse error is often not a cause for concern, unless it occurs in a function that is relevant to the semantic patch.
- `spatch -sp_file foo.cocci foo.c`: Apply the semantic patch `foo.c` to the file `foo.c` and print out any transformations as a diff.
- `spatch -sp_file foo.cocci foo.c -debug`: The same as the previous case, but print out some information about the matching process.
- `spatch -sp_file foo.cocci -dir foodir`: Apply the semantic patch `foo.cocci` to all of the C files in the directory `foodir`.
- `spatch -sp_file foo.cocci -dir foodir -include_headers`: Apply the semantic patch `foo.cocci` to all of the C files and header files in the directory `foodir`.

In the rest of this document, the options are annotated as follows:

- ◆: a basic option, that is most likely of interest to all users.
- ⬙: an option that is frequently used, often for better understanding the effect of a semantic patch.
- ◇: an option that is likely to be rarely used, but whose effect is still comprehensible to a user.
- An option with no annotation is likely of interest only to developers.

2 Selecting and parsing the semantic patch

2.1 Standalone options

⬙ **-parse_cocci** ⟨**file**⟩ Parse a semantic patch file and print out some information about it.

2.2 The semantic patch

◆ **-c** ⟨**file**⟩**, -sp_file** ⟨**file**⟩**, -cocci_file** ⟨**file**⟩ Specify the name of the file containing the semantic patch. The file name should end in `.cocci`.

2.3 Isomorphisms

◇ **-iso, -iso_file** Specify a file containing isomorphisms to be used in place of the standard one. Normally one should use the `using` construct within a semantic patch to specify isomorphisms to be used *in addition to* the standard ones.

-track_iso Gather information about isomorphism usage.

-profile_iso Gather information about the time required for isomorphism expansion.

2.4 Display options

◇ **-show_cocci** Show the semantic patch that is being processed before expanding isomorphisms.

◇ **-show_SP** Show the semantic patch that is being processed after expanding isomorphisms.

◇ **-show_ctl_text** Show the representation of the semantic patch in CTL.

◇ **-ctl_inline_let** Sometimes `let` is used to name intermediate terms CTL representation. This option causes the let-bound terms to be inlined at the point of their reference. This option implicitly sets **-show_ctl_text**.

◇ **-ctl_show_mcodekind** Show transformation information within the CTL representation of the semantic patch. This option implicitly sets **-show_ctl_text**.

◇ **-show_ctl_tex** Create a LaTeX files showing the representation of the semantic patch in CTL.

3 Selecting and parsing the C files

3.1 Standalone options

⬙ **-parse_c** ⟨**file/dir**⟩ Parse a `.c` file or all of the `.c` files in a directory. This generates information about any parse errors encountered.

⬙ **-parse_h** ⟨**file/dir**⟩ Parse a `.h` file or all of the `.h` files in a directory. This generates information about any parse errors encountered.

⬙ **-parse_ch** ⟨**file/dir**⟩ Parse a `.c` or `.h` file or all of the `.c` or `.h` files in a directory. This generates information about any parse errors encountered.

⬙ **-control_flow** ⟨**file**⟩**, -control_flow** ⟨**file**⟩**:**⟨**function**⟩ Print a control-flow graph for all of the functions in a file or for a specific function in a file. This requires `dot` and `gv`.

◇ **-type_c** ⟨**file**⟩ Parse a C file and pretty-print a version including type information.

-tokens_c ⟨**file**⟩ Prints the tokens in a C file.

-parse_unparse ⟨**file**⟩ Parse and then reconstruct a C file.

-compare_c ⟨**file**⟩ ⟨**file**⟩**, -compare_c_hardcoded** Compares one C file to another, or compare the file tests/compare1.c to the file tests/compare2.c.

-test_cfg_ifdef ⟨**file**⟩ Do some special processing of #ifdef and display the resulting control-flow graph. This requires `dot` and `gv`.

-test_attributes ⟨**file**⟩**, -test_cpp** ⟨**file**⟩ Test the parsing of cpp code and attributes, respectively.

3.2 Selecting C files

An argument that ends in `.c` is assumed to be a C file to process. A directory can be specified with the option **-dir**, described below. Normally, only one C file or one directory is specified. If multiple files are specified, they are treated in parallel, *i.e.*, the first semantic patch rule is applied to all functions in all files, then the second semantic patch rule is applied to all functions in all files, etc. If a directory is specified then no files may be specified and only the rightmost directory specified is used.

◆ **-dir** Specify a directory containing C files to process.

⬙ **-include_headers** This option causes header files to be processed independently. This option only makes sense if a directory is specified using **-dir**.

⬙ **-use_glimpse** Use a glimpse index to select the files to which a semantic patch may be relevant. This option requires that a directory is specified. The index may be created using the script `coccinelle/scripts/glimpseindex_cocci.sh`. Glimpse is available at http://webglimpse.net/. In conjunction with the option **-patch_cocci** this option prints the regular expression that will be passed to glimpse.

-kbuild_info ⟨**file**⟩ The specified file contains information about which sets of files should be considered in parallel.

-disable_worth_trying_opt Normally, a C file is only processed if it contains some keywords that have been determined to be essential for the semantic patch to match somewhere in the file. This option disables this optimization and tries the semantic patch on all files.

-test ⟨**file**⟩ A shortcut for running Coccinelle on the semantic patch "file`.cocci`" and the C file "file`.c`".

-testall A shortcut for running Coccinelle on all files in a subdirectory `tests` such that there are all of a `.cocci` file, a `.c` file, and a `.res` file, where the `.res` contains the expected result.

-test_okfailed, -test_regression_okfailed Other options for keeping track of tests that have succeeded and failed.

-compare_with_expected Compare the result of applying Coccinelle to file.c to the file file.res representing the expected result.

3.3 Parsing C files

◇ **-show_c** Show the C code that is being processed.

◇ **-parse_error_msg** Show parsing errors in the C file.

◇ **-type_error_msg** Show information about where the C type checker was not able to determine the type of an expression.

-use_cache Use preparsed versions of the C files that are stored in a cache.

-debug_cpp, -debug_lexer, -debug_etdt, -debug_typedef Various options for debugging the C parser.

-filter_msg, -filter_define_error, -filter_passed_level Various options for debugging the C parser.

-only_return_is_error_exit In matching "..." in a semantic patch or when forall is specified, a rule must match all control-flow paths starting from a node matching the beginning of the rule. This is relaxed, however, for error handling code. Normally, error handling code is considered to be a conditional with only a then branch that ends in goto, break, continue, or return. If this option is set, then only a then branch ending in a return is considered to be error handling code. Usually a better strategy is to use `when strict` in the semantic patch, and then match explicitly the case where there is a conditional whose then branch ends in a return.

Macros and other preprocessor code

◆ **-D ⟨file⟩, -macro_file ⟨file⟩** Extra macro definitions to be taken into account when parsing the C files.

◇ **-ifdef_to_if** This option constructs represents an #ifdef in the source code as a conditional in the control-flow graph. This option is unsafe, as it makes the source code unparsable when #ifdef is used in a way that does not directly convert to a conditional. Nevertheless, it can be useful when #ifdef is used in a well-structured way.

◇ **-use_if0_code** Normally code under #if 0 is ignored. If this option is set then the code is considered, just like the code under any other #ifdef.

-noadd_typedef_root This seems to reduce the scope of a typedef declaration found in the C code.

Include files

◆ **-all_includes, -local_includes, -no_includes** These options control which include files mentioned in a C file are taken into account. **-all_includes** indicates that all included files will be processed. **-local_includes** indicates that only included files in the current directory will be processed. **-no_includes** indicates that no included files will be processed. If the semantic patch contains type specifications on expression metavariables, then the default is **-local_includes**. Otherwise the default is **-no_includes**. At most one of these options can be specified.

◆ **-I ⟨path⟩** This option specifies the directory in which to find non-local include files. This option should be used only once, as each use will overwrite the preceding one.

◇ **-relax_include_path** This option causes the search for local include files to consider the directory specified using **-I** if the included file is not found in the current directory.

4 Application of the semantic patch to the C code

4.1 Feedback at the rule level during the application of the semantic patch

◆ **-show_bindings** Show the environments with respect to which each rule is applied and the bindings that result from each such application.

◆ **-show_dependencies** Show the status (matched or unmatched) of the rules on which a given rule depends. **-show_dependencies** implicitly sets **-show_bindings**, as the values of the dependencies are environment-specific.

◆ **-show_trying** Show the name of each program element to which each rule is applied.

◆ **-show_transinfo** Show information about each transformation that is performed. The node numbers that are referenced are the number of the nodes in the control-flow graph, which can be seen using the option **-control_flow** (the initial control-flow graph only) or the option **-show_flow** (the control-flow graph before and after each rule application).

◆ **-show_misc** Show some miscellaneous information.

◇ **-show_flow** ⟨**file**⟩**, -show_flow** ⟨**file**⟩**:**⟨**function**⟩ Show the control-flow graph before and after the application of each rule.

-show_before_fixed_flow This is similar to **-show_flow**, but shows a preliminary version of the control-flow graph.

4.2 Feedback at the CTL level during the application of the semantic patch

◆ **-verbose_engine** Show a trace of the matching of atomic terms to C code.

◇ **-verbose_ctl_engine** Show a trace of the CTL matching process. This is unfortunately rather voluminous and not so helpful for someone who is not familiar with CTL in general and the translation of SmPL into CTL specifically. This option implicitly sets the option **-show_ctl_text**.

◇ **-graphical_trace** Create a pdf file containing the control flow graph annotated with the various nodes matched during the CTL matching process. Unfortunately, except for the most simple examples, the output is voluminous, and so the option is not really practical for most examples.

◇ **-gt_without_label** The same as **-graphical_trace**, but the PDF file does not contain the CTL code.

◇ **-partial_match** Report partial matches of the semantic patch on the C file. This can be substantially slower than normal matching.

◇ **-verbose_match** Report on when CTL matching is not applied to a function or other program unit because it does not contain some required atomic pattern. This can be viewed as a simpler, more efficient, but less informative version of **-partial_match**.

4.3 Actions during the application of the semantic patch

◇ **-allow_inconsistent_paths** Normally, a term that is transformed should only be accessible from other terms that are matched by the semantic patch. This option removes this constraint. Doing so, is unsafe, however, because the properties that hold along the matched path might not hold at all along the unmatched path.

◇ **-disallow_nested_exps** In an expression that contains repeated nested subterms, *e.g.* of the form `f(f(x))`, a pattern can match a single expression in multiple ways, some nested inside others. This option causes the matching process to stop immediately at the outermost match. Thus, in the example `f(f(x))`, the possibility that the pattern `f(E)`, with metavariable `E`, matches with `E` as `x` will not be considered.

◇ **-pyoutput coccilib.output.Gtk, -pyoutput coccilib.output.Console** This controls whether Python output is sent to Gtk or to the console. **-pyoutput coccilib.output.Console** is the default. The Gtk option is currently not well supported.

-loop When there is "`...`" in the semantic patch, the CTL operator AU is used if the current function does not contain a loop, and AW may be used if it does. This option causes AW always to be used.

-steps ⟨**int**⟩ This limits the number of steps performed by the CTL engine to the specified number. This option is unsafe as it might cause a rule to fail due to running out of steps rather than due to not matching.

-bench ⟨**int**⟩ This collects various information about the operations performed during the CTL matching process.

-popl, -popl_mark_all, -popl_keep_all_wits These options use a simplified version of the SmPL language. **-popl_mark_all** and **-popl_keep_all_wits** implicitly set **-popl**.

5 Generation of the result

Normally, the only output is a diff printed to standard output.

◇ **-o** ⟨**file**⟩ The output file.

◇ **-inplace** Modify the input file.

◇ **-outplace** Store modifications in a .cocci_res file.

◇ **-no_show_diff** Normally, a diff between the original and transformed code is printed on the standard output. This option causes this not to be done.

◇ **-U** Set number of diff context lines.

◇ **-save_tmp_files** Coccinelle creates some temporary files in `/tmp` that it deletes after use. This option causes these files to be saved.

-debug_unparsing Show some debugging information about the generation of the transformed code. This has the side-effect of deleting the transformed code.

-patch Deprecated option.

6 Other options

6.1 Version information

- **-version** The version of Coccinelle. No other options are allowed.

- **-date** The date of the current version of Coccinelle. No other options are allowed.

6.2 Help

- **-h, -shorthelp** The most useful commands.

- **-help, –help, -longhelp** A complete listing of the available commands.

6.3 Controlling the execution of Coccinelle

- **-timeout** ⟨**int**⟩ The maximum time in seconds for processing a single file.

- **-max** ⟨**int**⟩ This option informs Coccinelle of the number of instances of Coccinelle that will be run concurrently. This option requires **-index**. It is usually used with **-dir**.

- **-index** ⟨**int**⟩ This option informs Coccinelle of which of the concurrent instances is the current one. This option requires **-max**.

- **-mod_distrib** When multiple instances of Coccinelle are run in parallel, normally the first instance processes the first n files, the second instance the second n files, etc. With this option, the files are distributed among the instances in a round-robin fashion.

-debugger Option for running Coccinelle from within the OCaml debugger.

-profile Gather timing information about the main Coccinelle functions.

-disable_once Print various warning messages every time some condition occurs, rather than only once.

6.4 Miscellaneous

- **-quiet** Suppress most output. This is the default.

-pad, -hrule, -xxx, -l1