Handling `#ifdef` Expressions in CPPSTATS

Claus Hunsen
hunsen@fim.uni-passau.de

September 2014

CPPSTATS was initially developed by Jörg Liebig at University of Passau for a set of studies [LAL+10, LKA11]. In 2013, Claus Hunsen from the same university has taken over development. The most current version of CPPSTATS is available at `https://github.com/clhunsen/cppstats/`. Further information can be found at `http://fosd.net/cppstats`.

1 Introduction

CPPSTATS is a tool for analyzing software systems regarding their variability. Therefore, we focus on software systems written in C using the capabilities of the CPP (the C preprocessor) to express variability. CPPSTATS handles the expressions of the CPP inclusion-guards (`#if`, `#elif`, `#endif`, etc.) in a special way, which is why we provide the used procedure in this document.[1]

There is handling of `#ifdef` expressions within CPPSTATS during three different parts of the whole program: *1*) light adaption of the expressions during *preparation* part of CPPSTATS, before generating SRCML files from the source code; *2*) collecting the expressions from the SRCML files and rewriting them by making implicit tangling explicit; and *3*) building a global expression pool for the analyzed software-project.

2 Example

As `#ifdef`s are explained best by means of an example, the three common pattern of CPP usage are shown in Figure 1. Each part of the example is present in a different file. Part (a) shows nesting of `#ifdef`s, while Part (b) and (c) show the use of `#else` and `#elif` branches in an `#ifdef` cascade.

During this document, the reader is referenced to these small examples to illustrate all matters.

[1] We mainly describe the mechanism that are performed in the GENERAL analysis (`analyses/general.py`), but the mechanisms relate to the other analyses, too.

```
1 #ifdef A
2   #ifdef B
3   #endif
4 #endif
```

(a) Nested `#ifdef` in file `X.c`.

```
5 #if defined(A) \
6      && defined(B)
7 #else
8 #endif
```

(b) `#else` branch in file `Y.c`.

```
8  #ifndef Z_H
9  #define Z_H
10 #ifdef C
11 #elif defined(D)
12 #endif
13 #endif // Z_H
```

(c) `#elif` branch and include guard in file `Z.h`.

Figure 1: Short examples of the patterns that occur while using CPP and that are treated by CPPSTATS. Each example and their rewriting rules are explained in this very document.

3 Source-Code Preparation to SRCML files

The first part of a CPPSTATS run is the source-code preparation that transforms the plain-text source code to SRCML[2] files. SRCML is an XML format for source code that preserves line numbers and preprocessor information.

Before transforming the source code to SRCML, there are several code-normalization steps, which heavily depend on the code analysis to be performed. For the different preparation types, please refer to the file `preparation.py` of CPPSTATS.

There are up to three possible steps in the preparation that can have effect on the `#ifdef` expressions, though all have only cosmetic effect: *1*) the handling of multi-line expressions; *2*) the rewriting of the shortcut expressions `#ifdef` and `#ifndef`; and *3*) the removal of include guards.

After all steps of the preparation, CPPSTATS has generated proper SRCML files for each source file. Based on these files, CPPSTATS performs the analyses.

3.1 Multi-Line `#ifdef` Expressions (`preparations/rewriteMultilineMacros.py`)

A multi-line `#ifdef` expression is shown in Fig. 1b on Lines 5 and 6. This expressions is rewritten as a single-line expression, so that CPPSTATS does not have to handle line breaks in `#ifdef` expressions later. The line numbers are preserved during this step.

The output, after applying this change to Fig. 2a, is shown in Fig. 2b.

3.2 Rewriting of `#ifdef` and `#ifndef` (`preparations/rewriteIfdefs.py`)

As another step, the `#ifdef` expressions are more streamlined by rewriting the conditionals `#ifdef` and `#ifndef` to their long forms. This yields `#if defined(X)` for the conditional `#ifdef X`, and `#if !defined(Y)` for `#ifndef Y`.

[2] http://www.srcml.org/

```
5 #if defined(A) \
6     && defined(B)
7 #else
8 #endif
```

(a) Repetition of Fig. 1b, containing a multi-line expression on Lines 5 and 6.

```
5 #if defined(A) && defined(B)
6
7 #else
8 #endif
```

(b) The same code with single-line expressions.

Figure 2: An `#else` branch in file `Y`, (a) before and (b) after rewriting multi-line expressions into single-line fashion.

3.3 Removal of Include Guards (`preparations/deleteIncludeGuards.py`)

The last preparation step that affects `#ifdef` expressions is the removal of include guards. Include guards are used to prevent the multiple inclusion of header files and would bias the results of CPPSTATS, because they are not a mechanism to implement variability.

An inclusion guard, as shown in Fig. 1c consists of three parts: *1*) a conditional-inclusion guard checking if the file was already included (Line 8); *2*) the definition of the file guard for that is checked in Line 8 (Line 9); and *3*) the closing `#endif` for the `#ifndef` (Line 13).

When a header file containing this construct is included again, the conditional in Line 8 will be false, because `Z_H` is already defined. The preprocessor will skip over the entire contents of the file, and the compiler will not see it twice.

As a result of this preparation step, the source-code lines corresponding to the include guard (Lines 8, 9, and 13) are removed from the file and exchanged with blank lines, so that line numbers are preserved.

4 Processing of Expressions During File Analysis

CPPSTATS performs its analyses based on the SRCML files that are generated at the end of the preparation step. The `#ifdef` expressions are collected while traversing the XML tree of the input files, the main functionality is implemented in the function `_getFeatures` (`analyses/general.py`, Line 565ff.). During this action, the nesting hierarchy of the conditionals is remembered and used to rewrite the `#ifdef` expressions accordingly in the function `_getFeatureSignature` (Lines 479ff.). Implicit tangling of `#ifdef` expressions are made explicit in this step, while preserving all nesting-depth information.

For example, CPPSTATS rewrites all nested `#ifdef`s, each with a condition that conjoins their own conditional expression with the enclosing ones, because the inner expression depends on the evaluation of its own condition and, additionally, on the evaluation of the surrounding `#ifdef` expression. This is done also for `#else` and `#elif` expressions.

The result of these mechanisms is shown in Fig. 3.

```
#if defined(A)
  #if defined(A) && defined(B)
  #endif
#endif
```

(a) Nested `#ifdef` in file `X.c`.

```
#if defined(A) && defined(B)
#else
#endif
```

(b) `#else` branch in file `Y.c`.

```
#if defined(C)
#elif !defined(C) && defined(D)
#endif
```

(c) `#elif` branch and include guard in file `Z.h`.

Figure 3: Rewritten `#ifdef` expressions from Fig. 1, also including all changes from Section 3.
(Note that the files are formatted as SRCML actually, but, for better understanding, the source-code representations are shown here.)

5 Global Expression Pool

The processed `#ifdef` expressions from Section 4

References

[LAL+10] Jörg Liebig, Sven Apel, Christian Lengauer, Christian Kästner, and Michael Schulze. An Analysis of the Variability in Forty Preprocessor-Based Software Product Lines. In *Proc. Int. Conf. Software Engineering (ICSE)*, pages 105–114. ACM, 2010.

[LKA11] Jörg Liebig, Christian Kästner, and Sven Apel. Analyzing the Discipline of Preprocessor Annotations in 30 Million Lines of C Code. In *Proc. Int. Conf. Aspect-Oriented Software Development (AOSD)*, pages 191–202. ACM, 2011.