15 November 2024

Change Documentation for

Testwell CTC++

Version 10.2.0

Features and Changes

Merging for source code variants

Coverage results for variants due to preprocessing differences can now be merged to one result per source file.
For that purpose, **ctcreport** provides two new options:

- `-merge-variants`: Instead of file.c (1), file.c (2), …, file.c (n), one merged result is reported.
- `-merge-and-keep-variants`: The variants and the merged result are reported.

For every coverage measure, unified coverage ratios are calculated.

Files and Functions	MC/DC	Decision	Statement	Function
program.c	10 / 12	10 / 12	7 / 9	4 / 4
program.c (1)	7 / 8	7 / 8	5 / 7	3 / 3
program.c (2)	7 / 8	7 / 8	5 / 6	3 / 3
program.c (3)	4 / 6	4 / 6	3 / 4	2 / 3

Merged functions, files and directories are marked with symbol , if they contain preprocessing differences relevant for measuring coverage.
In the source code view of a merged function, instrumentation probes are marked with a grey bar if they do not match across variants.
Code lines are marked with a bar if they are both active and inactive across variants.

```
2                5  void foo(int x) {
                 6      if (x == MY_CONST) {
1        0               if (x == 1)
0        1               if (x == 2)
                 7          statement();
                 8      }
                 9      else {
                10          #if MY_CONST == 1
                11          statement();
                12          #endif
                13          statement();
                14      }
2               15  }
```

Remark: All merging information is given for convenience, suitable for small changes due to preprocessing. Typically, code variants must be tested independently.

Justifications for ternary-? operator

Decisions in a ternary-? operator can now be actively justified:

```
  6 True | Reviewed: hard to test, low risk.
```

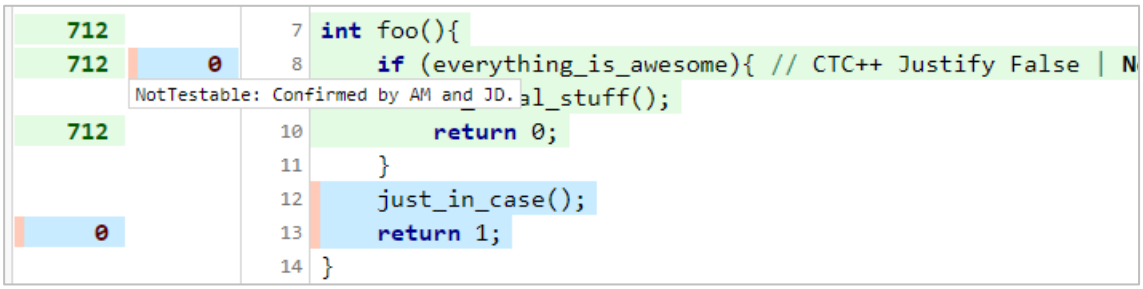
In this example, the true counter is justified in a companion file.

```
0        1       6      int a = x? 17: 42;
Reviewed: hard to test, low risk.  do_something();
```

Please note that justifications are not derived from or to these decisions, they work independently from other justifications

Justifications for code after not reachable branch ends

A branch with a `return` statement at its end could prevent later code from being justified by derivation. Now justifications are no longer blocked in cases like line 12 and 13 below. A justification of the false-counter in line 8 now leads to justifications of the two statements at the end.



Remark: This does not work if there is not reachable code between the return statement in line 10 and its branch end in line 11.

New and extended template variables for reporting

For functions, new information is available for reporting:

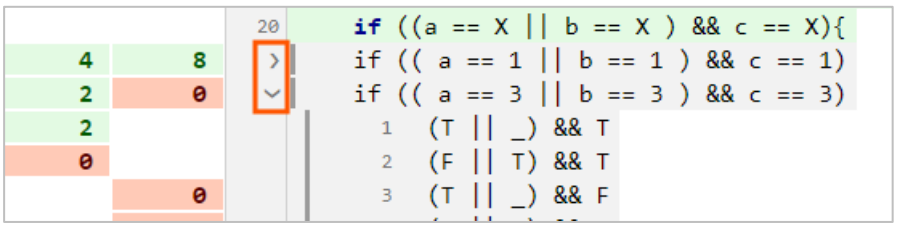
- $DefinedInLine$: Gives the line of function definition recorded by **ctc**, usable in single-file templates.
- $NumberOfCalls$: Usable in single-file templates and in source code view of structured templates.
- $FunctionParameters$: Now also usable in source code view of structured templates.

Structured reporting templates

While the content of template files **overview.html**, **detail.html** and **sourcecode.html** does not need to be HTML, the file extension .html was mandatory. With that release, the file extension can be chosen matching the content.

Collapsing and expanding

In HTML report, coverage details for condition, multicondition, and MC/DC can now be collapsed and expanded for each complex decision probe individually. This is especially useful for merged results, but not restricted to these cases.



Documented parameters of functions

Function parameters were recorded from the first declaration visible to **ctc**. This has changed, only function definition is recorded now.
Example: For function foo, `( int arg )` is recorded now instead of `( int x_arg ).`

```
1   int foo(int x_arg);
2
3   int foo(int arg){
4       return arg;
5   }
```

Reduction of **ctcpost**

ctcpost is no longer able to generate reports. Reporting is purely provided by **ctcreport**. **ctcpost** is now only responsible for

- Combining symbol or data files (option `-a`).
- Extracting instrumentation information from symbol and data files (options `-l` and `-L`).

Withdrawal of **ctcxmlmerge**

The tool previously used to merge XML reports from **ctcpost** into summary XML reports or text reports is no longer part of Testwell CTC++. Use the new merging feature of **ctcreport** for this purpose. For specific needs or questions, please get in contact with us in advance.

Withdrawal of Cygwin support

Cygwin support is no longer part of the Windows package.

Suppress `concept` and `requires` treatment (up to C++ 17)

In consequence of a bug fix mentioned below, **ctc** now processes `concept` and `requires` strictly as keywords according to newest C++ standards.
For C++ code using these terms for own identifiers, this treatment can lead to unwanted results. You can suppress it with a new configuration parameter:

```
SUPPRESS_CPP_CONCEPTS = ON
```

Bug Fixes

Parsing of C++ 20 concepts

For C++ 20 and beyond, a parsing issue could occur for concepts, requires clauses and requires expressions, often used in system headers. This issue could lead to completely uninstrumented source files.

Uninstrumented code after template definition

Template definitions with less-than operator inside parameters were not parsed correctly if followed by `using`. Code after the definition was left uninstrumented.
Example:

```
template<size_t _Len,
         size_t _Align = _Len < ... // Less-than
         ...
        >
using aligned_storage_t = typename aligned_storage<_Len, _Align>::type;
```

Miscalculated statements

For functions containing a switch-statement without braces, like

```
switch(a)
        case 1: statement();
```

statement count at the end of the function was misleading in some cases.

Change of execution path for instrumented program

It was possible that a composed Boolean expression containing a cast evaluated differently when instrumented for multicondition.
Example:

```
int a = 1, b = 65536;
if (a && (short)(b)){
   …
}
```

Necessary for the occurrence of this issue was that a C-style cast is applied to an expression in parentheses and changes its value regarding true / false evaluation.

False variants for nested includes

For an included instrumented header file, **ctcreport** falsely generated variants if the header includes another instrumented header file between its own functions:

```
void in_header_1(){}

#include "header_in_between.h"

void in_header_2(){}
```

In this case, its functions were not correctly assigned to these variants.

Code shown as inactive

In symbol files, information about active code lines was missing when upper- and lower-case drive letters were used in combination during build on Windows. This led to falsely inactive code in HTML report.

Wrong function assignment

In symbol files, the return from files included was not recorded correctly when upper- and lower-case drive letters were used in combination during build on Windows. This led to functions assigned to the wrong source file.

Wrong function names

Function names could be recorded with wrong namespace hierarchy, depending on source code from headers included. Typically, `std::` was inserted in between own namespaces.

Crash of **ctc**

A crash of **ctc** could occur when the compiler was invoked with more than one source file, depending on code elements like `typedef` present in the translation units.

ctclaunch on Linux

For Linux distributions like Ubuntu 24 and newer using glibc version 2.37, **ctclaunch** was not able to find its 64-bit shared object `liblaunch.so`.

Firefox presentation bug

For the HTML report, browser Firefox wrapped “MC/DC” in coverage ring charts. This is suppressed now.

Compiler error for [[fallthrough]] attribute

Since version 10.1, attributes `[[fallthrough]]` and `[[clang::fallthrough]]` could lead to compiler warnings or errors when instrumented.

Reading of outdated data by **ctcreport**

Symbol files overwritten by **ctc** are not truncated, but their new end is marked with END_OF_SYMBOLFILE. In some cases, **ctcreport** processed information after END_OF_SYMBOLFILE and falsely reported this information.

Braces in single-file templates

Loops directly inserted after an opening `{` were not parsed correctly by **ctcreport**.

```
  {{CTC_LOOP("Functions")} …
```