Correctness analysis

Preliminaries

- True positive (TP): An influential change is correctly classified as influential
  - `x in EIS && x in AIS`
  - Occurs if a **FAILURE** verification result from CBMC is considered sound based on manual inspection.
- True negative (TN): An equivalent change is correctly classified as equivalent
  - `x not in EIS && x not in AIS`
  - Occurs if a **SUCCESS** verification result from CBMC is considered sound based on manual inspection.
- False positive (FP): An equivalent change is incorrectly classified as influential
  - `x in EIS && x not in AIS`
  - Occurs if a **FAILURE** verification result from CBMC is considered unsound based on manual inspection.
- False negative (FN): An influential change is correctly classified as equivalent
  - `x not in EIS && x in AIS`
  - Occurs if a **SUCCESS** verification result from CBMC is considered unsound based on manual inspection.

Note that the CBMC traces do not show states in order of occurrence, the counterexample trace may also differ between repeated invocations since there could exist several counterexamples.

The standard requirement for manually verifying that a verification result is sound is to device an equivalent counterpart to the **FAILURE** example which highlights that the failed verification indeed contains a new execution flow. All of these crafted examples were performed with `--unwind 1` (just like the actual trials).

Note, the harnesses created by EUF are always created inside of a `.harnesses` directory in the old version of the repository. This means that all headers that are included correspond to headers from the old version of the dependency. Any changes to the struct definitions in these headers which may have occurred in an update, are thus not taken into account when declaring the non-deterministic input variables.

Each library will however still have their own canonically defined version of each struct separate from the one imported in the harness. I.e. invocations of a new function will use the struct definition relevant to their own context but the input may have been initialised using a different template.

If not otherwise stated, screenshots with four splits show the influential update to the *left* and the equivalent update to the *right*.

Unless otherwise stated, each function analysis focuses on a specific *SUCCESS* and *FAILURE* result. This is preferable since multiple occurrences of the same result for a function often relate to the same underlying change.

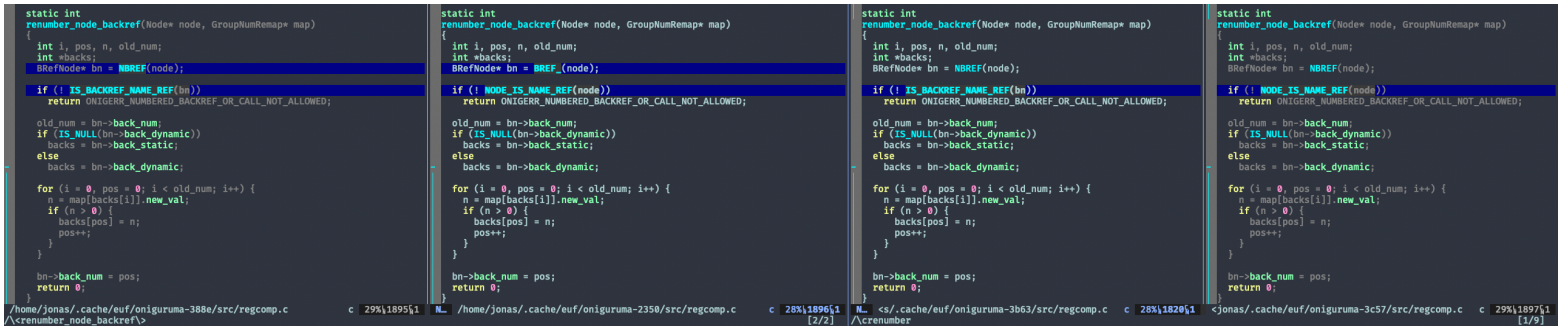
libonig

`renumber_node_backref()`

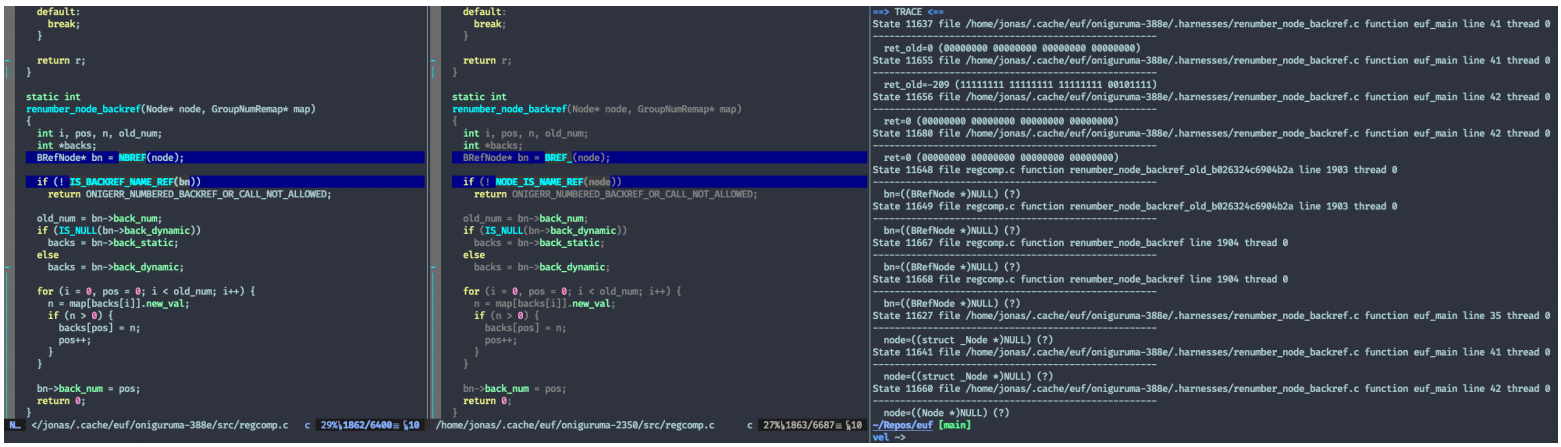
SUCCESS_UNWIND_FAIL (1)
FAILURE_UNWIND_FAIL (2)

```
./scripts/analyze_function.sh onig renumber_node_backref 388e 2350
git difftool --no-index ~/.cache/euf/oniguruma-388e/src/regcomp.c \
  ~/.cache/euf/oniguruma-2350/src/regcomp.c
# 1899:1 <-> 1900:1
# Wed Jun 7 14:20:18 2017 -> Tue Jun 20 16:30:21 2017
```

```
# => ~/.cache/euf/oniguruma-388e/.harnesses/renumber_node_backref.c (FAILURE_UNWIND_FAIL)

./scripts/analyze_function.sh onig renumber_node_backref 3b63 3c57
git difftool --no-index ~/.cache/euf/oniguruma-3b63/src/regcomp.c \
  ~/.cache/euf/oniguruma-3c57/src/regcomp.c
# 1824:1 <-> 1901:1
# Wed May 24 13:43:25 2017 -> Thu Jun 8 13:24:35 2017
# => ~/.cache/euf/oniguruma-3b63/.harnesses/renumber_node_backref.c (SUCCESS_UNWIND_FAIL)
```



The influential and equivalent update both originate from textually identical versions of `renumber_node_backref`. Both updates modify the `if` condition that determines if `ONIGERR_NUMBERED_BACKREF_OR_CALL_NOT_ALLOWED` (-209) is returned. CBMC has found a counterexample where the *2350* revision evaluates the condition as true while the old version evaluates it as false for the same input.



Expanding macros shows that the change from `NBREF -> BREF_` in the influential case has no effect.

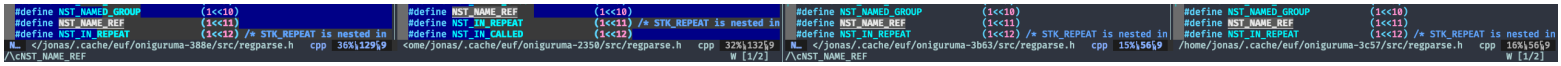
```
// Same across all versions
#define ONIGERR_NUMBERED_BACKREF_OR_CALL_NOT_ALLOWED         -209

// 388e, 3b63 (old versions) and 3c57 (new equivalent version)
#define NBREF(node)                 (&((node)->u.bref))

// 2350 (new influential version)
```

```
#define BREF_(node)                 (&((node)->u.bref))
```

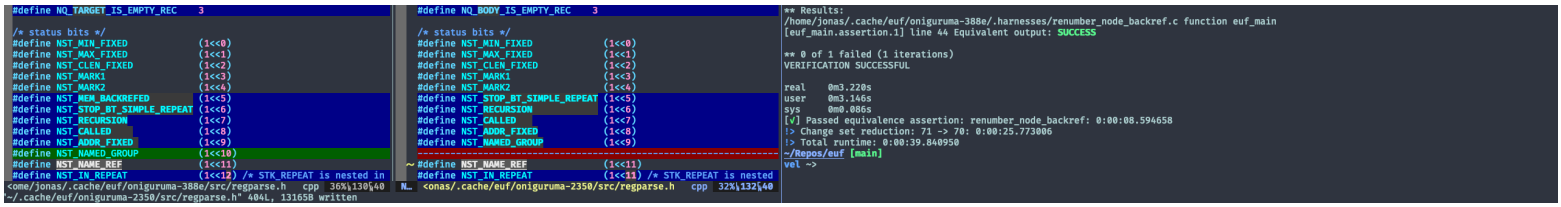
The change from `IS_BACKREF_NAME_REF -> NODE_IS_NAME_REF` looks equivalent in both cases on first glance but expanding all macros shows a notable difference between *2350* and the other three revisions.

```
// 388e and 3b63 (old versions)
#define IS_BACKREF_NAME_REF(bn)    (((bn)->state & NST_NAME_REF)    != 0)

// 3c57 (new equivalent)
#define NODE_IS_NAME_REF(node)     (((node)->u.base.status & NST_NAME_REF)     != 0)

// 2350 (new influential)
#define NODE_IS_NAME_REF(node)     ((NODE_STATUS(node) & NST_NAME_REF) != 0)
#define NODE_STATUS(node)          (((Node* )node)->u.base.status)
```

The difference which triggers the counterexample lies in the definition of `NST_NAME_REF`.



Patching `regparse.h` in revision *2350* to have the same definition of `NST_NAME_REF` gives a successful equivalence verification. The reason for CBMC's labeling has thus been pin-pointed and both classifications are thus considered sound.



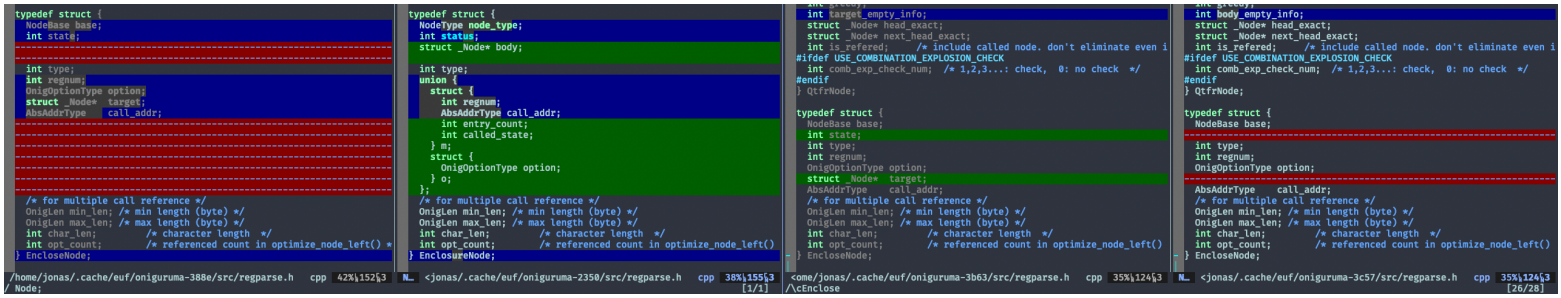
Before noticing the difference in `NST_NAME_REF`, differences in the composition of the `Node` struct were investigated.

Notice how neither of the old versions define the `status` field in `NodeBase`. Both of the new versions define the `status` field (revision *3c57* is textually different by defining the `NodeBase` struct nested within the `Node` struct but this does not affect the compiled code) and rely on it during their invocations of `NODE_IS_NAME_REF`.

Recall that EUF initialises all structs inside of a harness using definitions from the old version of the dependency. This means that the `Node` object passed to the new versions should lack `status` and `body` fields. However, the equivalent result in *3b63 -> 3c57* (and the revised version of *388e -> 2350*) indicate that this did not cause issues for verification.

A partial explantation for this lies in the fact that the `Node` struct uses a `union` for all its fields. Since memory will be allocated based on the largest member in the `union`, the `Node` struct would have valid memory in the locations of `status` and `body` despite not having them explicitly defined.

This line of thinking implies that invalid memory accesses would occur if a field exceeding the maximum size of the original `Node` struct would be defined and used in the new version. At a glance, the change from `EncloseNode` to `EnclosureNode` seems like a possible candidate for this type of scenario.

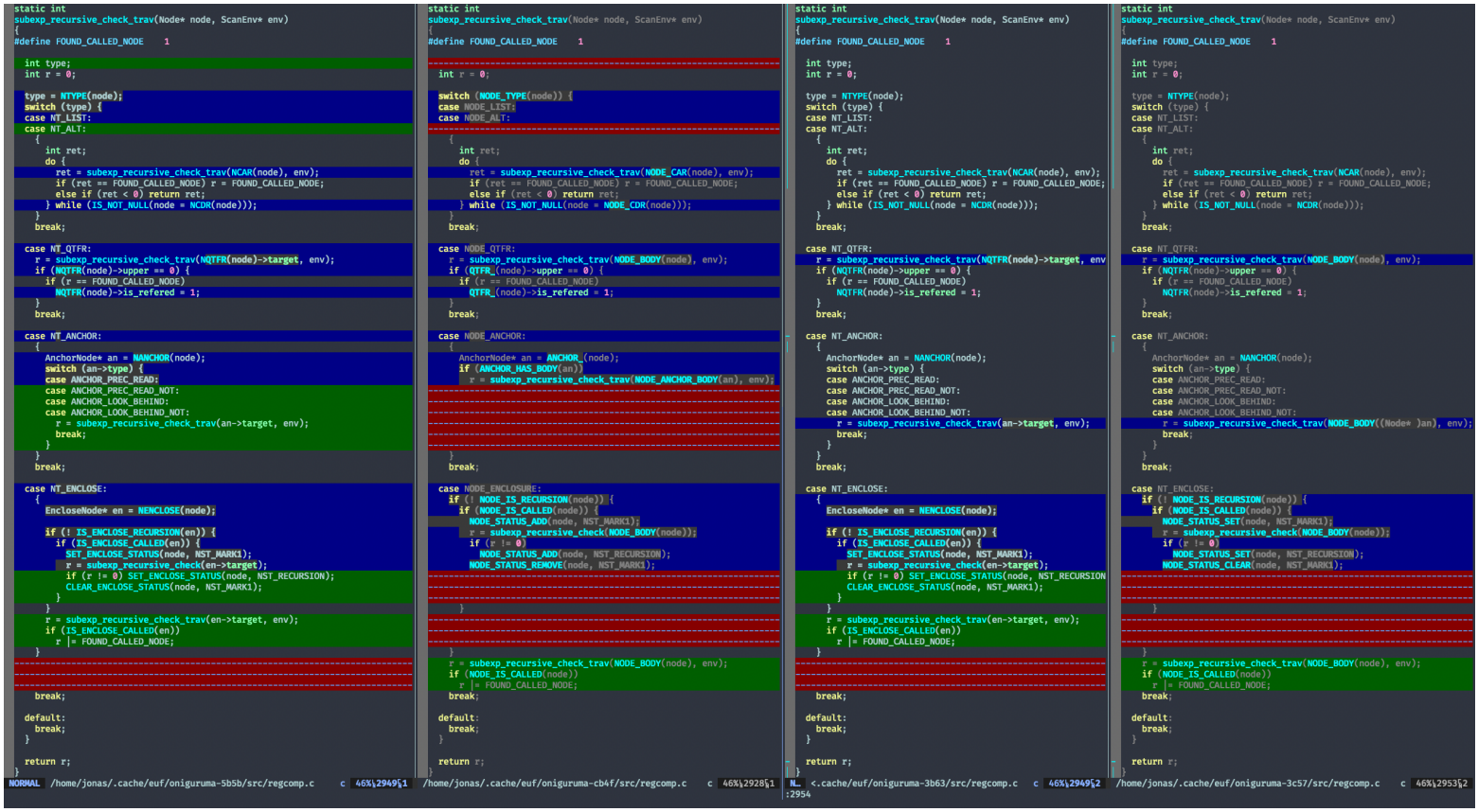


- **Equivalent verdict**: Sound
- **Influential verdict**: Sound

`subexp_recursive_check_trav()`

SUCCESS_UNWIND_FAIL (1)
FAILURE_UNWIND_FAIL (1)

```
./scripts/analyze_function.sh onig subexp_recursive_check_trav 5b5b cb4f
git difftool --no-index ~/.cache/euf/oniguruma-5b5b/src/regcomp.c \
  ~/.cache/euf/oniguruma-cb4f/src/regcomp.c
# 2950:1 <-> 2929:1
# Thu Jun 1 13:16:09 2017 -> Mon Jun 12 11:30:51 2017
# => ~/.cache/euf/oniguruma-5b5b/.harnesses/subexp_recursive_check_trav.c (FAILURE_UNWIND_FAIL)
./scripts/analyze_function.sh onig subexp_recursive_check_trav 3b63 3c57
git difftool --no-index ~/.cache/euf/oniguruma-3b63/src/regcomp.c \
  ~/.cache/euf/oniguruma-3c57/src/regcomp.c
# 2950:1 <-> 2954:1
# Wed May 24 13:43:25 2017 -> Thu Jun 8 13:24:35 2017
# => ~/.cache/euf/oniguruma-3b63/.harnesses/subexp_recursive_check_trav.c (SUCCESS_UNWIND_FAIL)
```



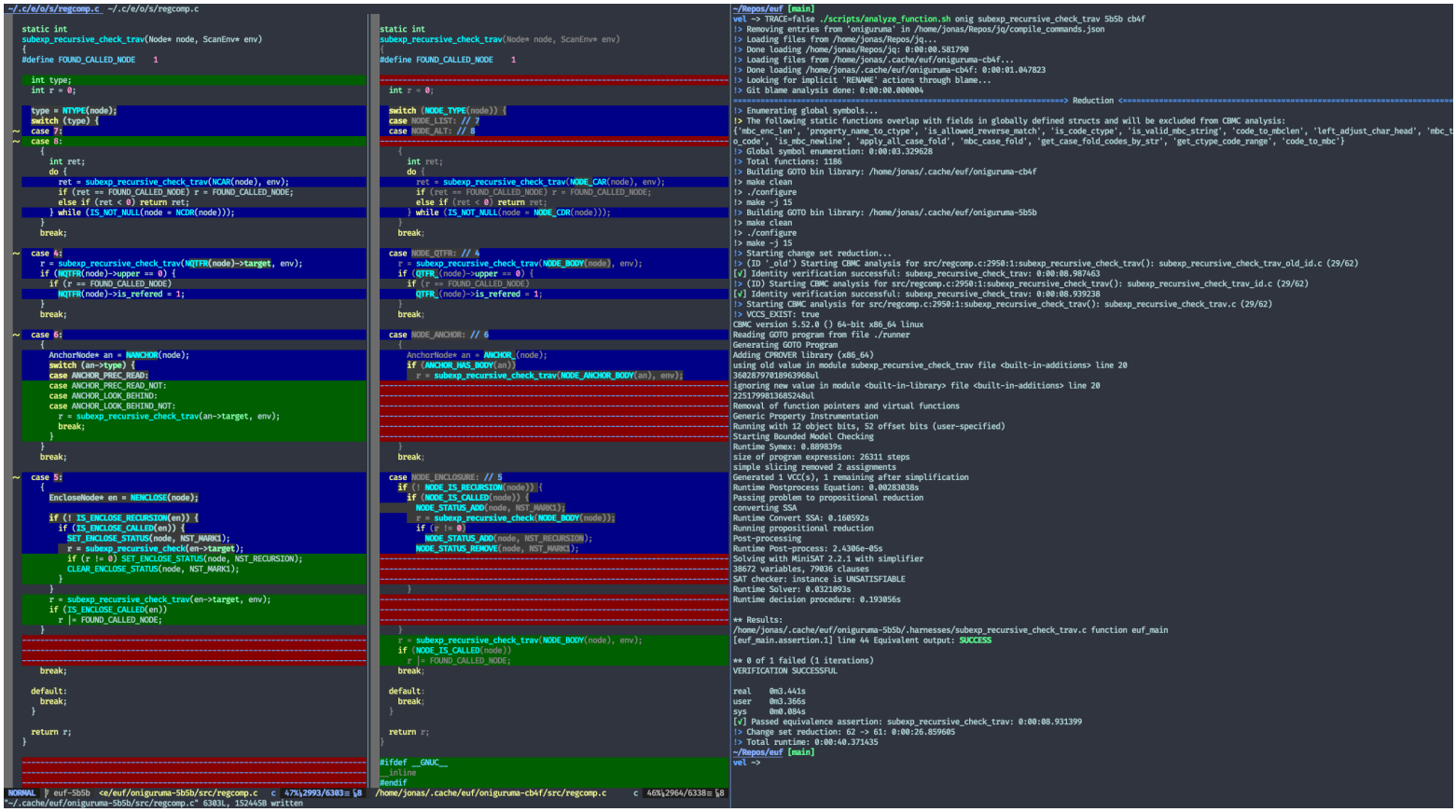
Both changes originate from the same base version. The influential case presents a counterexample where the old version returns `r=0` while the new version returns `r=1`.

The `r` variable is assigned with recursive output from the function and with 1 (through logical OR with `FOUND_CALL_NODE=1`). Both updates modify the two lines preceding `r |= FOUND_CALLED_NODE;` in the same manner.

Note that revision *cb4f* is the only one that modifies the names of constants in each `case`. One would assume that `NT_ENCLOSE` and `NODE_ENCLOSURE` have the same value but the *cb4f* update has actually set `NODE_ENCLOSURE = 5`, violating this assumption since `NT_ENCLOSE=4`.

With this in mind, it is fairly clear why the verification fails. Given the same `Node` object, a type of `5` will pick the `ENCLOSURE` case in the new version and instead of `NT_ENCLOSE` the old version will pick the `NT_QTFR = 5` branch. The old branch does not contain the `r |= FOUND_CALLED_NODE;` statement and a failed verification is attained.

Modifying the old `case` statements to be inline with the new values for each symbol yields a successful verification, highlighting this as the root cause for the failed verification.



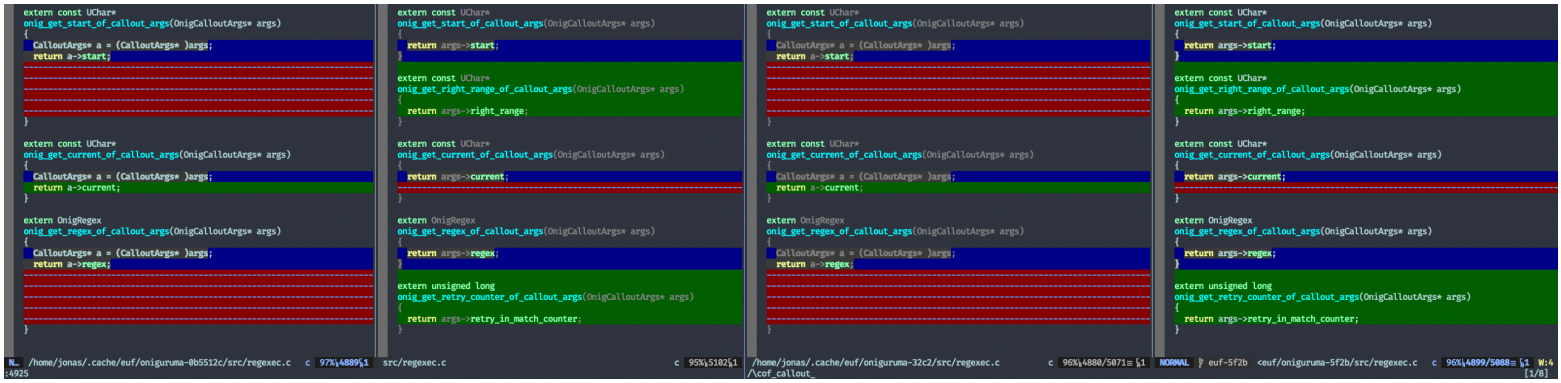
- **Equivalent verdict**: Sound
- **Influential verdict**: Sound

`onig_get_start_of_callout_args()`

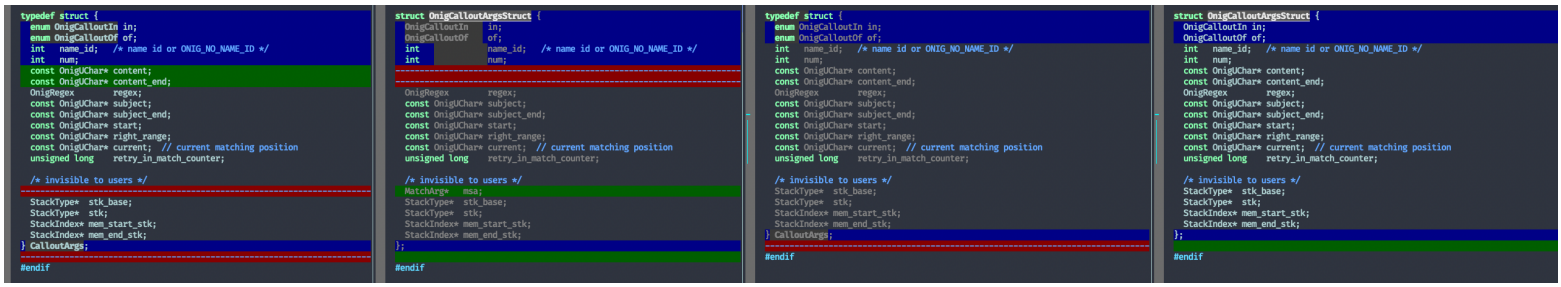
FAILURE (4)
SUCCESS (1)

```
./scripts/analyze_function.sh onig onig_get_start_of_callout_args 0b5512c 23b2
git difftool --no-index ~/.cache/euf/oniguruma-0b5512c/src/regexec.c \
  ~/.cache/euf/oniguruma-23b2/src/regexec.c
# 4891:1 <-> 5104:1
# Tue Feb 13 13:26:46 2018 -> Mon Feb 26 11:04:07 2018
# => ~/.cache/euf/oniguruma-0b55/.harnesses/onig_get_start_of_callout_args.c (FAILURE)

./scripts/analyze_function.sh onig onig_get_start_of_callout_args 32c2 5f2b
git difftool --no-index ~/.cache/euf/oniguruma-32c2/src/regexec.c \
  ~/.cache/euf/oniguruma-5f2b/src/regexec.c
# 4925:1 <-> 4939:1
```

```
# Tue Feb 13 14:02:35 2018 -> Wed Feb 21 17:36:04 2018
# => ~/.cache/euf/oniguruma-32c2/.harnesses/onig_get_start_of_callout_args.c (SUCCESS)


# Comparing the influential (_right_ as it is newer) and equivalent new states
./scripts/analyze_function.sh onig onig_get_start_of_callout_args 5f2b 23b2
git difftool --no-index  ~/.cache/euf/oniguruma-5f2b/src/regexec.c \
  ~/.cache/euf/oniguruma-23b2/src/regexec.c
```



The influential and equivalent cases are textually identical in this scenario and thus also originate from the same base state. The only change is the removal of a pointer cast from `OnigCalloutArgs` to `CalloutArgs`. In both of the new revisions, the definition of `CalloutArgs` has been replaced with a struct definition for `OnigCalloutArgsStruct` (which `OnigCalloutArgs` is a `typedef` for). The influential update differs in that it has removed the `content` and `content_end` fields while adding a new field, `match` in this struct.



Since the `start` field is placed after the new content fields, it will be accessed at a different offset in *0b55* compared to *23b2*. The internal format of `OnigCalloutArgs` (i.e. the format in each version of `libonig`) will dictate what offsets are used to retrieve data. The inconsistent return value thus arises from the new version attempting to use the indices of the new version of `OnigCalloutArgs` upon an object initialised using the old format. The fact that a successful verification is attained when patching the new version with the old fields supports this line of reasoning.

While the reason behind the FAILURE case can be soundly explained, it constitutes a FP from a practical point of view since the function itself has not changed in an influential way. Had the new version been given a `OnigCalloutArgs` struct matching its own internal representation, the behaviour would remain the same. The current design of EUF does not have an interface which supports several definitions of the same struct while also keeping all overlapping fields equal (through assumptions), something akin to this would be required to resolve FPs of this type.

- **Equivalent verdict**: Sound
- **Influential verdict**: Unsound

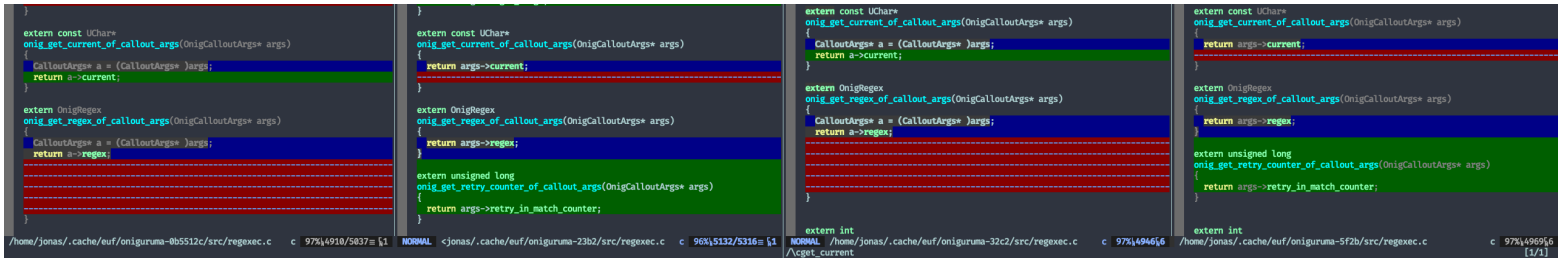
`onig_get_current_of_callout_args()` and `onig_get_regex_of_callout_args()`

FAILURE (4+4)
SUCCESS (1+1)

```
./scripts/analyze_function.sh onig onig_get_current_of_callout_args 0b5512c 23b2
./scripts/analyze_function.sh onig_get_regex_of_callout_args 0b5512c 23b2
git difftool --no-index ~/.cache/euf/oniguruma-0b55/src/regexec.c ~/.cache/euf/oniguruma-23b2/src/regexec.c
# 4898:1 <-> 5116:1
# Tue Feb 13 13:26:46 2018 -> Mon Feb 26 11:04:07 2018
# => ~/.cache/euf/oniguruma-0b55/.harnesses/onig_get_current_of_callout_args.c (FAILURE)
```

```
# => ~/.cache/euf/oniguruma-0b55/.harnesses/onig_get_regex_of_callout_args.c (FAILURE)

./scripts/analyze_function.sh onig onig_get_current_of_callout_args 32c2 5f2b
./scripts/analyze_function.sh onig_get_regex_of_callout_args 32c2 5f2b
git difftool --no-index ~/.cache/euf/oniguruma-32c2/src/regexec.c ~/.cache/euf/oniguruma-5f2b/src/regexec.c
# 4932:1 <-> 4951:1
# Tue Feb 13 14:02:35 2018 -> Wed Feb 21 17:36:04 2018
# => ~/.cache/euf/oniguruma-32c2/.harnesses/onig_get_current_of_callout_args.c (SUCCESS)
# => ~/.cache/euf/oniguruma-32c2/.harnesses/onig_get_regex_of_callout_args.c (SUCCESS)
```

The same analysis given for `onig_get_start_of_callout_args()` applies for both updates to these functions.

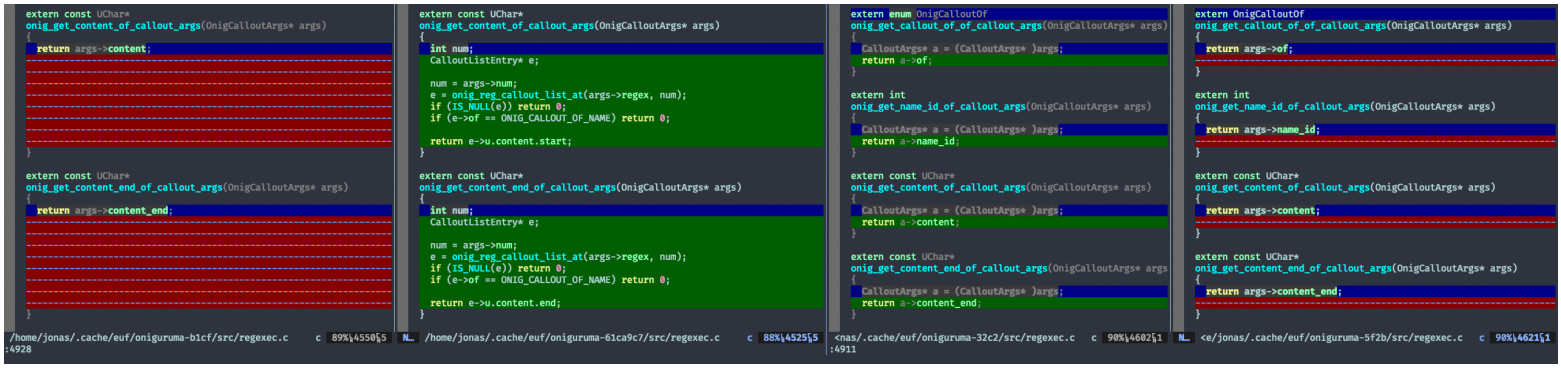


- **Equivalent verdict**: Sound
- **Influential verdict**: Unsound

`onig_get_content_of_callout_args()` and `onig_get_content_end_of_callout_args()`

FAILURE (1+1)
SUCCESS (1+1)

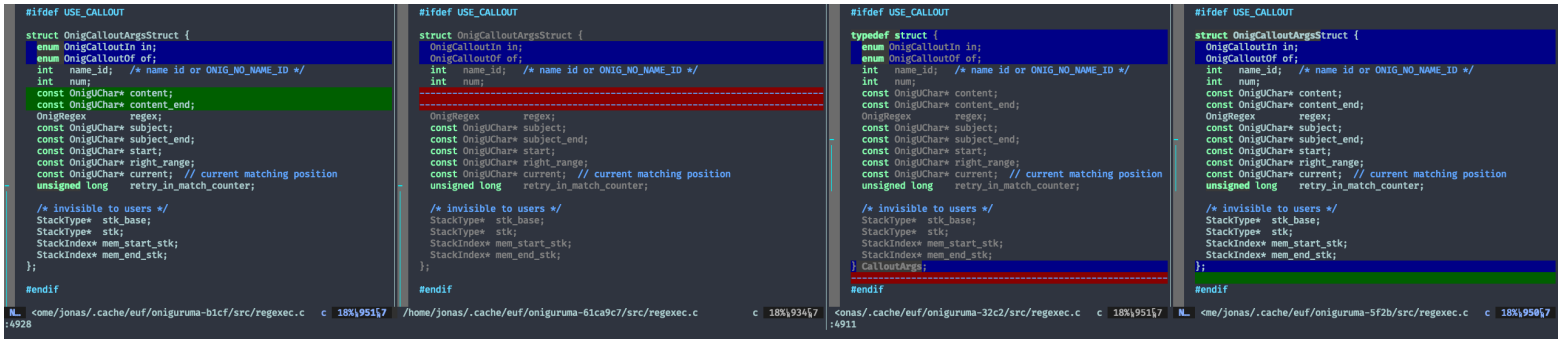
```
./scripts/analyze_function.sh onig onig_get_content_of_callout_args b1cf 61ca9c7
./scripts/analyze_function.sh onig onig_get_content_end_of_callout_args b1cf 61ca9c7
git difftool --no-index ~/.cache/euf/oniguruma-b1cf/src/regexec.c \
  ~/.cache/euf/oniguruma-61ca9c7/src/regexec.c
# 4928:1 <-> 4903:1
# Thu Feb 15 16:34:51 2018 -> Thu Feb 22 13:57:46 2018
# => ~/.cache/euf/oniguruma-b1cf/.harnesses/onig_get_content_of_callout_args.c (FAILURE)

./scripts/analyze_function.sh onig onig_get_content_of_callout_args 32c2 5f2b
./scripts/analyze_function.sh onig onig_get_content_end_of_callout_args 32c2 5f2b
git difftool --no-index ~/.cache/euf/oniguruma-32c2/src/regexec.c \
  ~/.cache/euf/oniguruma-5f2b/src/regexec.c
# 4911:1 <-> 4927:1
# Tue Feb 13 14:02:35 2018 -> Wed Feb 21 17:36:04 2018
# => ~/.cache/euf/oniguruma-32c2/.harnesses/onig_get_content_of_callout_args.c (SUCCESS)
```

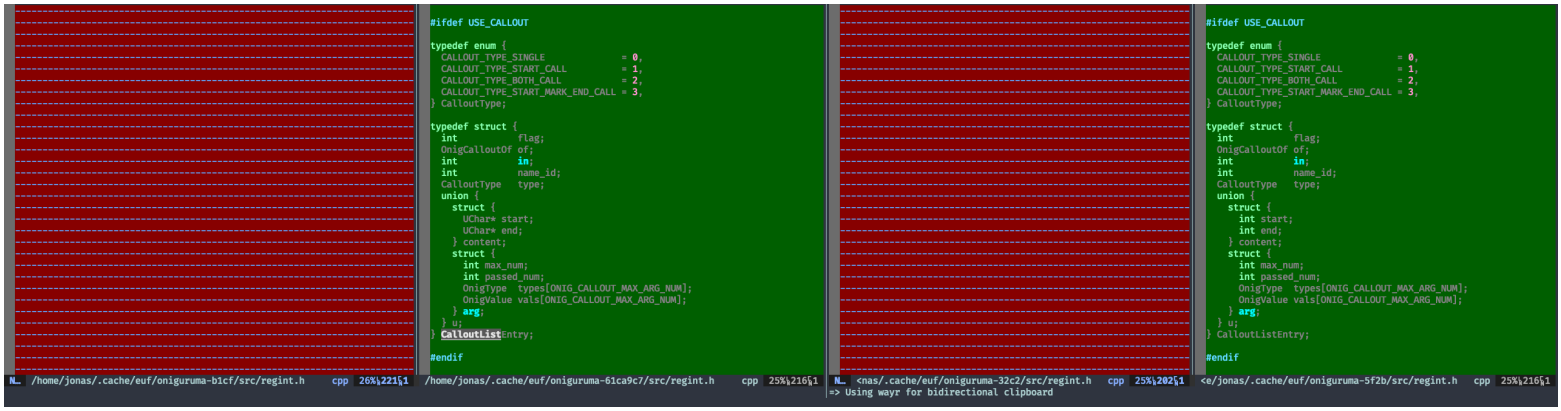


The equivalent updates for both functions follow the same format as that described for `onig_get_start_of_callout_args()`. The new equivalent revision, *5f2b* and the old influential revision, *b1cf*, share the same structure of `OnigCalloutArgsStruct` and the functions in question.

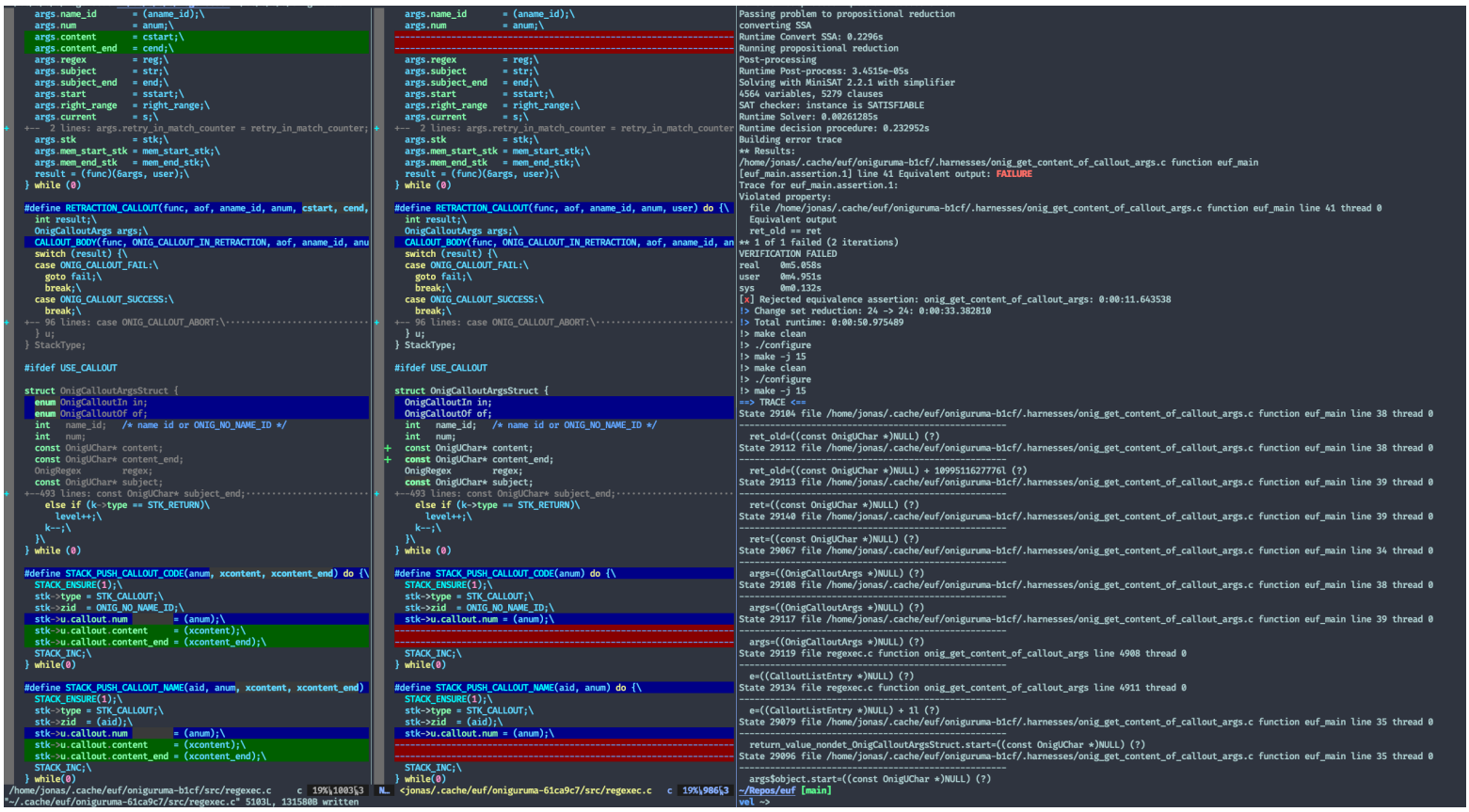
Both influential changes return a field with a similar name and presumably the same internal purpose, either `content.start` or `content.end`, from a `CalloutListEntry` object which is derived from the passed argument (`OnigCalloutArgs`). The `CalloutListEntry` is identically defined in both new revisons but is only utilised in *61ca*.

The influential update has removed the `content` and `content_end` fields from its `OnigCalloutArgs` struct and while input/output behaviour could feasibly have been maintained from an architectural point of view, proving this with CBMC would as described in the analysis of `onig_get_start_of_callout_args()` require initialisation of two different types of `OnigCalloutArgs` variables with an assumed equality in every shared field.





Restoring `OnigCalloutArgsStruct` to its original state (as done for `onig_get_start_of_callout_args()`) does not yield a succesful verification in this scenario since the return value is also dependent on the new struct, `CalloutListEntry`.



- **Equivalent verdict**: Sound
- **Influential verdict**: Unsound

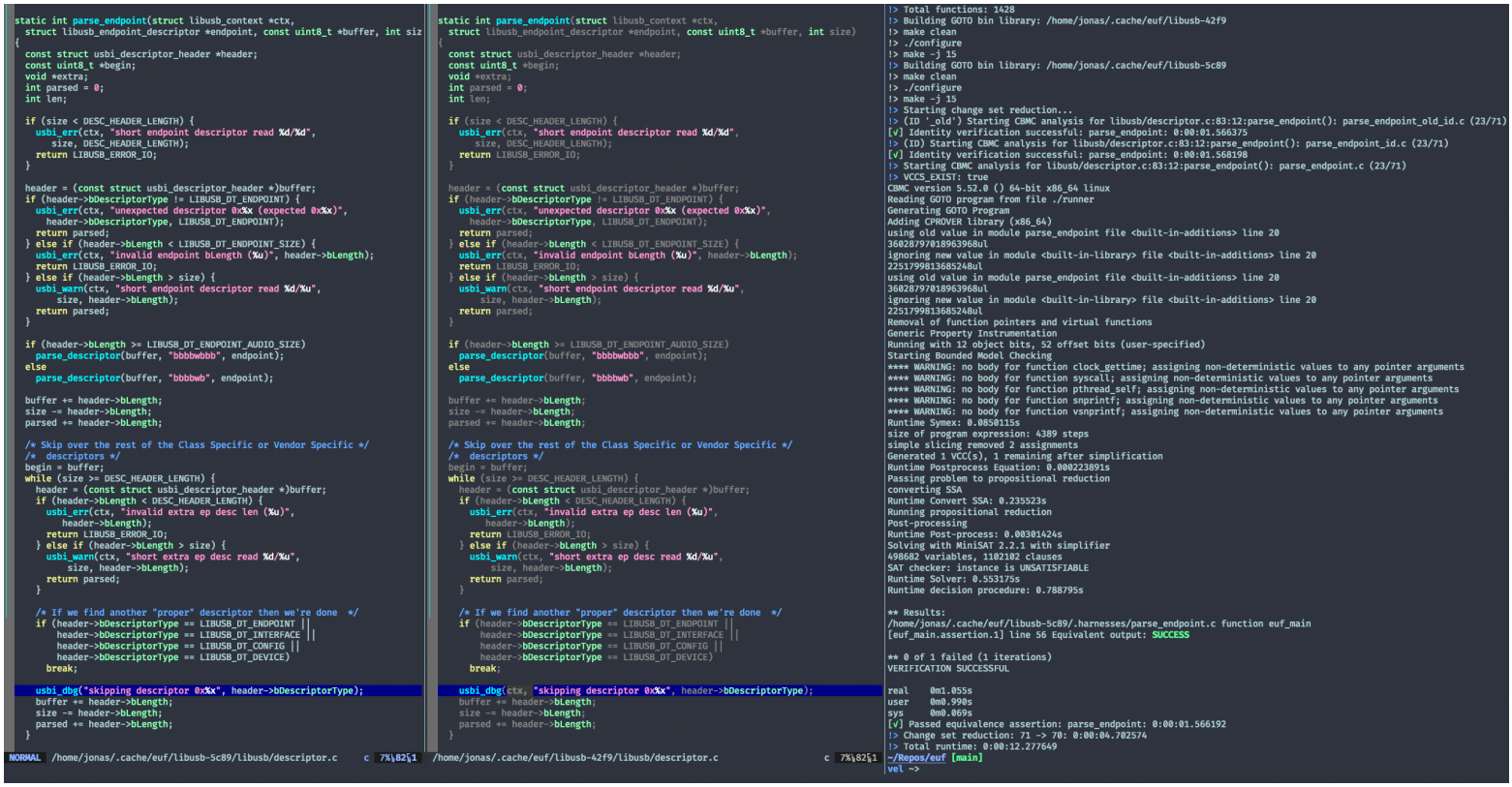
libusb

`parse_endpoint()`

SUCCESS_UNWIND_FAIL (4)
FAILURE_UNWIND_FAIL (13)

```
./scripts/analyze_function.sh usb parse_endpoint 5c89 42f9
git difftool --no-index ~/.cache/euf/libusb-5c89/libusb/descriptor.c \
  ~/.cache/euf/libusb-42f9/libusb/descriptor.c
# 83:12 <-> 83:12
# Tue Jun 15 22:33:45 2021 -> Wed Jul 21 11:47:25 2021
# => ~/.cache/euf/libusb-5c89/.harnesses/parse_endpoint.c (SUCCESS_UNWIND_FAIL)

./scripts/analyze_function.sh usb parse_endpoint ba6b d424
git difftool --no-index ~/.cache/euf/libusb-ba6b/libusb/descriptor.c \
  ~/.cache/euf/libusb-d424/libusb/descriptor.c
# 83:12 <-> 101:12
# Mon Aug 10 19:19:21 2020 -> Mon Aug 17 14:15:10 2020
# => ~/.cache/euf/libusb-ba6b/.harnesses/parse_endpoint.c (FAILURE_UNWIND_FAIL)
```

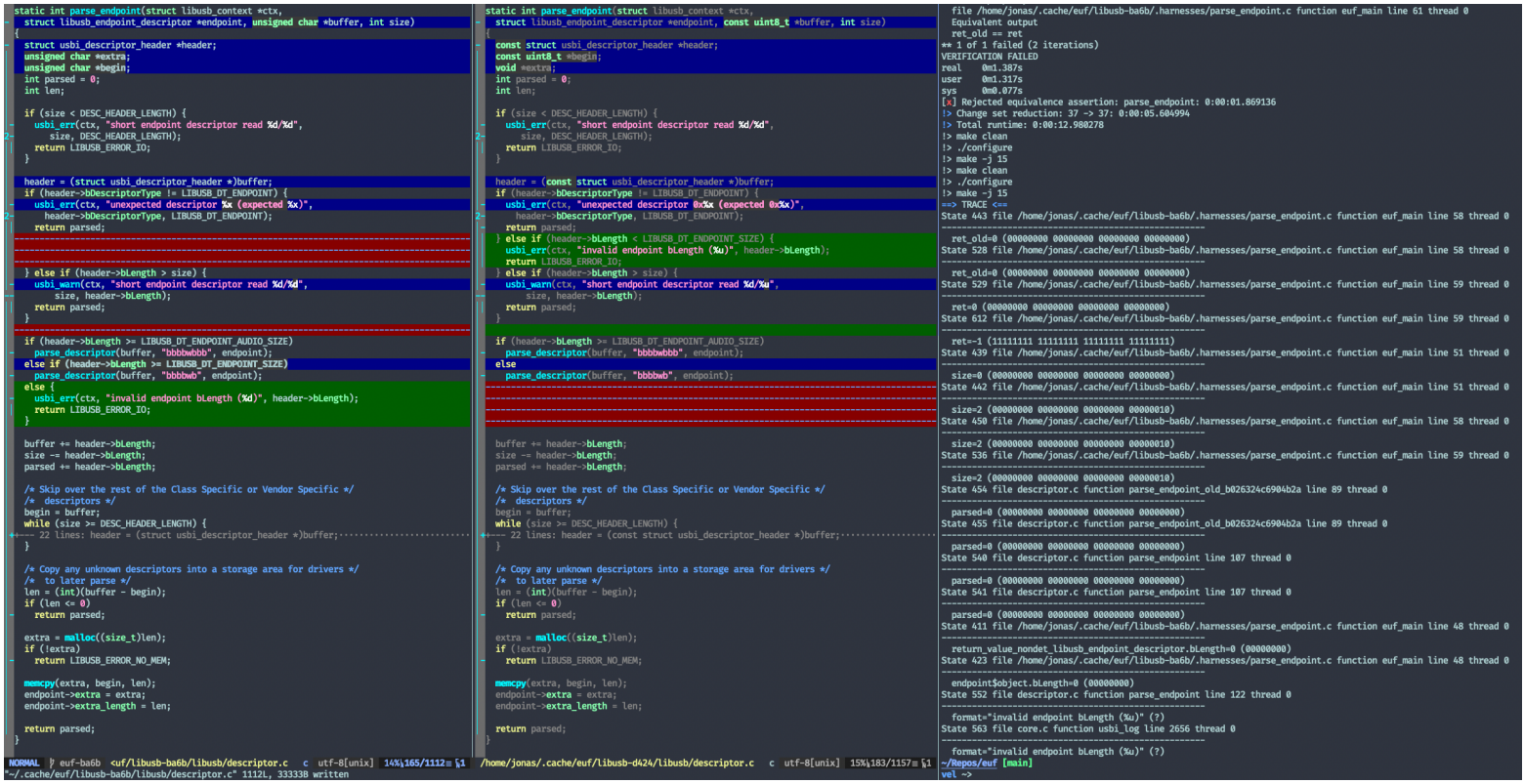
The equivalent update originates from a modification to the `usbi_dbg` prototype and does not affect the return value in any way, the classification is thus considered sound.



The influential update has several modifications but notably moves the `else` block that is evaluated for `bLength < LIBUSB_DT_ENDPOINT_SIZ` (`bLength < 7`) higher up in the function and converts it to an `else if` statement.

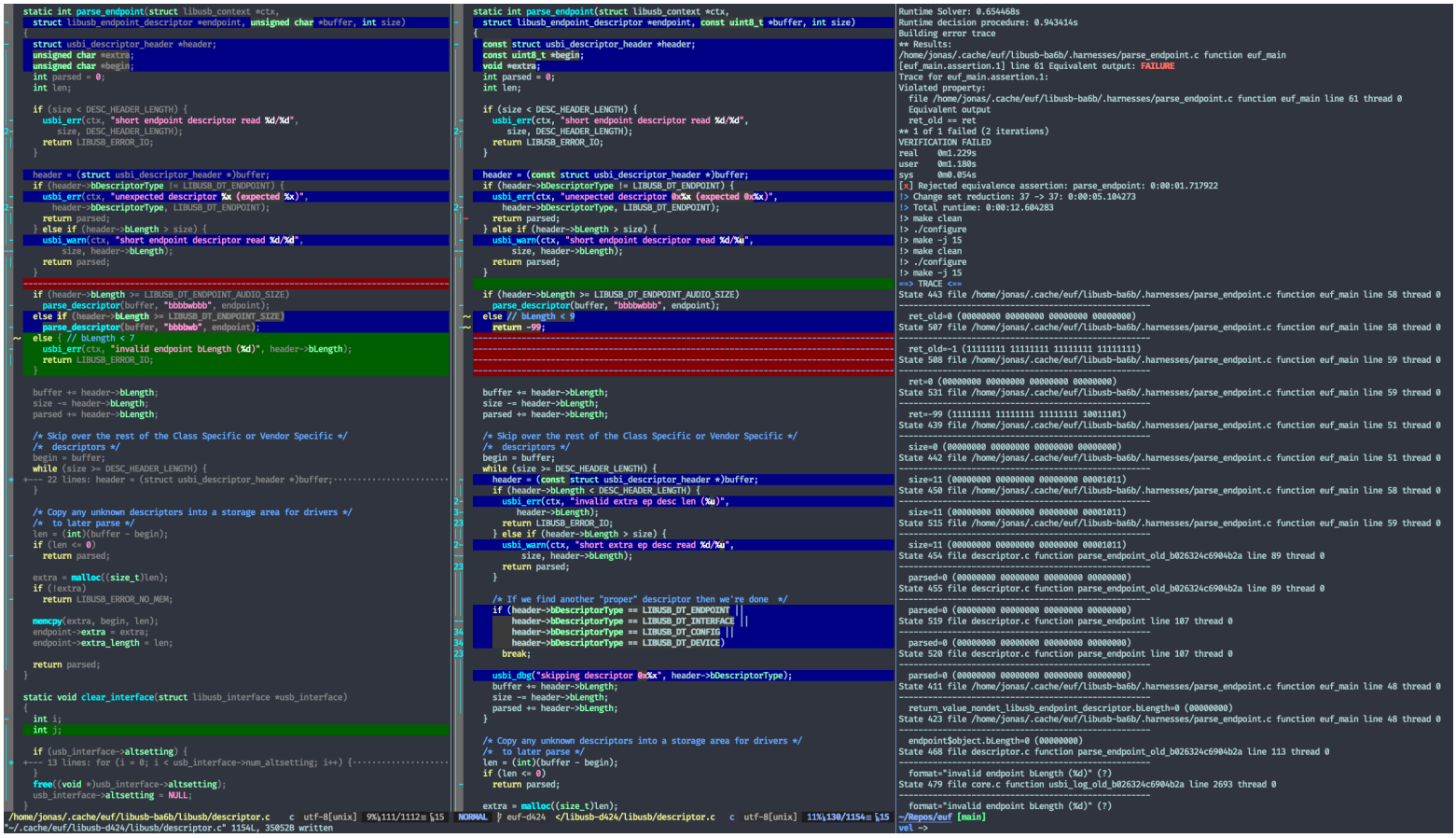
The counterexample trace shows that for `size=2` (a size not below `DESC_HEADER_LENGTH`) the old version takes the `else if (header->bLength > size)` branch, returning the value of `parsed`, 0, while the new version takes the `else if (header->bLength < LIBUSB_DT_ENDPOINT_SIZE)` branch, returning `LIBUSB_ERROR_IO` (-1).

CBMC has thus found a sound counter-example where a value of `header->bLength` in the range `[size,LIBUSB_DT_ENDPOINT_SIZE]` will trigger different behaviour across the two versions. A complete list of the states for `bLength` was not attainable from CBMC but the trace should theoretically show a value in this prescribed range.

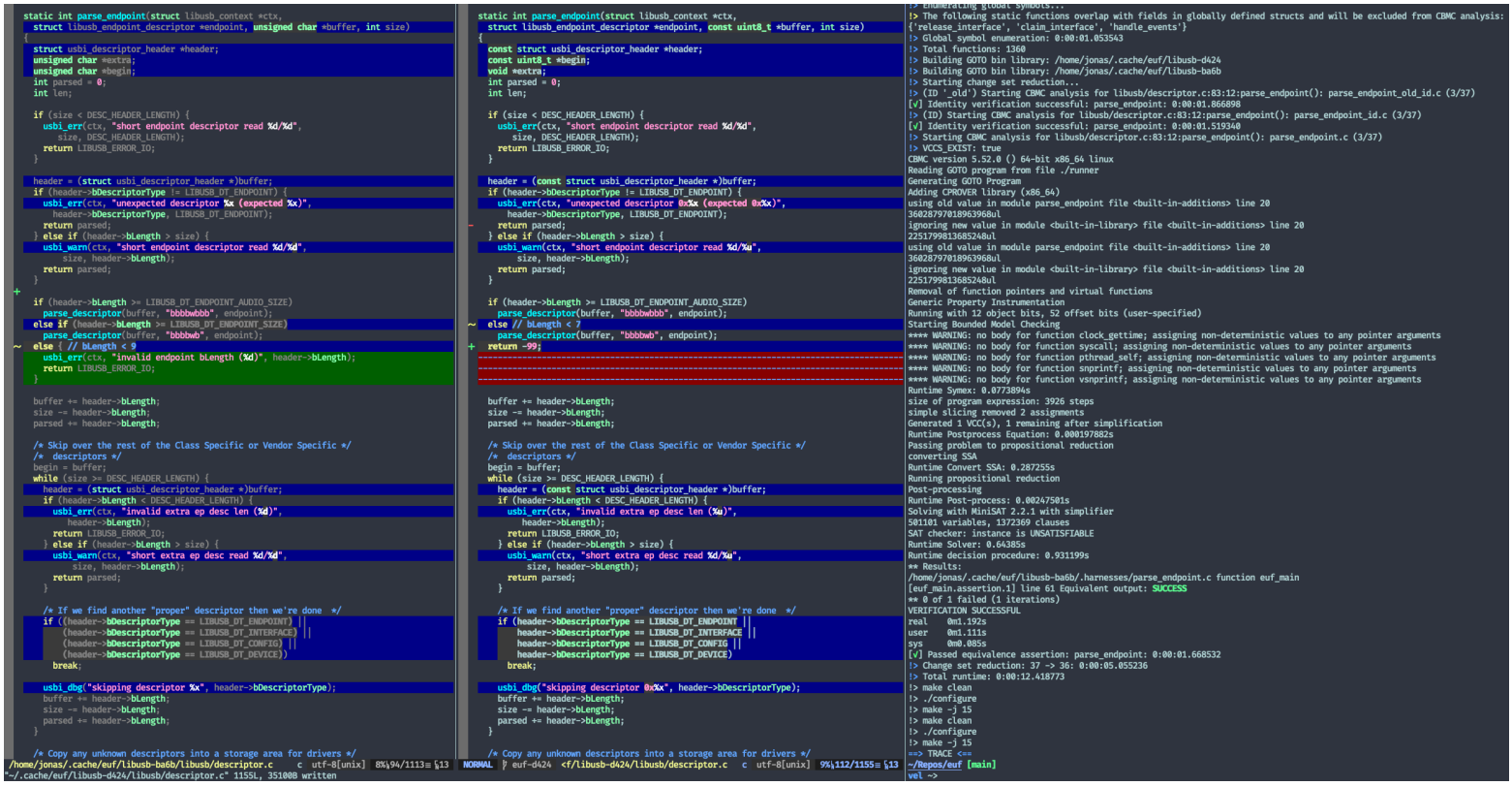


Removing the `(header->bLength < LIBUSB_DT_ENDPOINT_SIZE)` branch from the new version yields a successful verification.

This seems like a FN since if `bLength < 7` were to be true for the old version, it would return -1 while the new version would take the `bLength < 9` branch and continue execution. Replacing the `parse_descriptor(buffer, "bbbbwb", endpoint)` call with `return -99` expectedly creates a failed verification, highlighting that the `bLength < 9` block in the new version is indeed evaluated together with the old `bLength < 7` block for at least one trace.

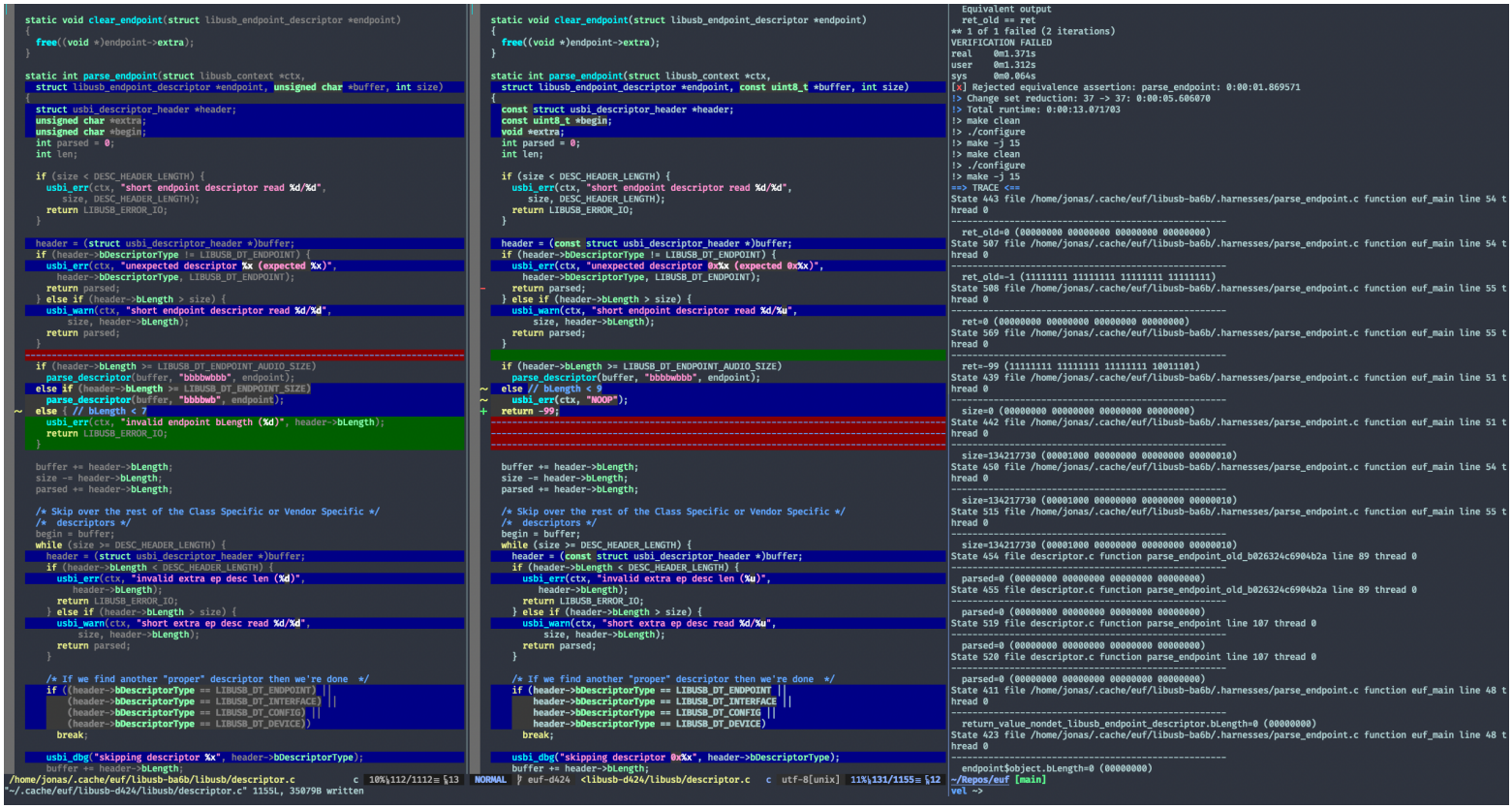


However, placing the `return -99` statement outside of the `else` block yields an equivalent result. For this classification to be valid would infer that no trace in the new version actually reaches the inserted return statement, which should not be possible unless one of the three earlier `return` statements are taken.



The `parse_descriptor` function does not contain any obvious behaviour that could prevent `return -99` from still being executed but replacing the call with a `usbi_err` call (essentially a NOOP) triggers the expected counter example.

Note how the counter example trace has a large value set for `size`, this indirectly implies that `bLength` has a value exceeding `size` since the third return statement in the new version was not taken. This in turn means that the `usbi_err` call should not be evaluated, making the reason as to why the modification triggers a counter example even less clear.



As previously shown, the *ba6b -> d424* update contains a valid counter-example and thereby does not constitute a FP per say. However, modifying the update shows cracks in the coverage of CBMC for this function, highlighting the fact that several forms of similar influential updates could be incorrectly labeled and create FPs.

- **Equivalent verdict**: Inconclusive
- **Influential verdict**: Sound