Improve the handling of exceptions thrown from contract predicates

Gašper Ažman (gasper.azman@gmail.com)
Timur Doumler (papers@timur.audio)

Document #: P3417R0
Date: 2024-10-16
Project: Programming Language C++
Audience: SG21, LEWG

Abstract

This paper proposes an improvement to how exceptions thrown from the evaluation of a contract predicate are handled. In particular, we propose to handle such an exception separately from an exception that was being handled when the contract violation occurred; `std::current_exception()` should return a pointer only to the latter, never to the former; the former should only be accessible through the dedicated function proposed in [P3227R0].

1 The problem

The Contracts MVP [P2900R9] specifies that when the evaluation of a contract predicate exits via an exception, the contact-violation handler is called and acts as an exception handler for that exception. Therefore, the exception can be retrieved with the following incantation:

```
void handle_contract_violation (contract_violation& violation) {
  if (violation.detection_mode() == detection_mode::evaluation_exception)
    my::handle(std::current_exception());
}
```

In [P3227R0], we propose to add a member function `evaluation_exception()` to class contract_violation, which simplifies the above to the much more user-friendly:

```
void handle_contract_violation (contract_violation& violation) {
  if (auto ex = violation.evaluation_exception())
    my::handle(ex);
}
```

However, the library API additions proposed in [P3227R0] do not alter the [P2900R9] behaviour of `std::current_exception()`. The result is that, if the contract check exited via an exception, `std::current_exception()` and `predicate_ exception()` will always both point to that exception; if it did not, the former may be a null pointer or may point to an unrelated exception that was being handled when the contract violation occurred, while the latter will always be a null pointer.

This somewhat surprising behaviour is the consequence of treating the predicate exception as just another exception on the exception stack. Arguably, this design goes against the design principle that the evaluation of contract predicates should be "ghost code" that does not affect the state of the program. It also raises the question of what happens when we rethrow the current exception from the contract-violation handler:

```
void handle_contract_violation (contract_violation& violation) {
  if (auto ex = std::current_exception())
    std::rethrow_exception(ex);  // or just `throw;`
    // what happens now?
```

With the current specification in [P2900R9], this may either rethrow the predicate exception or, if the contract check did not throw an exception but occurred inside a `catch` clause in user code, rethrow the entirely unrelated exception that was being handled there. This behaviour confounds two entirely different use cases for throwing an exception from the contract-violation handler.

If a contract violation occurs inside a `catch` clause, we might decide that we cannot continue executing that catch clause correctly, and instead rethrow the currently handled exception from the contract-violation handler in order for it to be handled further up in user code. The context of this exception is unknown to the handler.

If, on the other hand, the contract check itself throws an exception, we might want to rethrow *that* exception to be handled in a specific way. The context of this exception is known to the handler: it is an exception that represents a failure to perform a contract check. These two types of exceptions are *of a different nature* and will require a different handling strategy. *Rethrowing each exception should therefore be performed with a different construct*.

Rethrowing the current exception from the contract-violation handler should rethrow whatever exception was being handled when the contract violation occurred, and *not* the exception thrown from the evaluation of the predicate. This prevents leaking any information about the contract check into user code.

On the other hand, rethrowing the predicate exception should be done with an explicit construct expressing the intent of the developer. Such a construct is being proposed in [P3227R0]:

```
  std::rethrow_exception(violation.predicate_exception());
```

2 The solution

We can specify the desired behaviour as follows. Instead of treating the contract-violation handler as the handler for an exception thrown during predicate evaluation, we can treat an exception thrown during predicate evaluation as being handled *before* the contract violation handler is called. We can express this as a modification of the pseudocode in [P2900R9], Section 3.5.11 that illustrates the compiler-generated contract-violation handling process:

P2900R8	**This paper**

`bool violation = false;`
`try {`
`  violation = !predicate;`
`}`
`catch (...) {`
`  // set detection_mode to`
`  // evaluation_exception`
`  handle_contract_violation(...);`
`}`
`if (violation) {`
`  // set detection_mode to`
`  // predicate_false`
`  handle_contract_violation(...);`
`}`
	`bool violation = false;`
	`try {`
	`  _violation = !predicate;`
	`}`
	`catch (...) {`
	`  // store currently handled exception`
	`  // in predicate_exception`
	`}`
	`__handle_contract_violation(...);`

Implementations can choose to implement these semantics as above, or alternatively, to treat an exception thrown during predicate evaluation on an entirely separate exception stack. Regardless of the implementation strategy, the observable behaviour is the same, and is consistent with treating predicate exceptions as separate from the remainder of the program, thus bringing the semantics in line with the design principles of [P2900R9].

This proposal is a change in language semantics. It is being proposed on top of proposal [P3227R0], which itself only modifies library API and is an improvement over [P2900R9] in its own right regardless of whether *this* proposal will be adopted.

3 Proposed wording

We propose the following changes to [P2900R9].

- Change [basic.contract.eval] as follows:

  If the contract violation occurred because the evaluation of the predicate exited via an exception, the contract-violation handler is invoked while that exception is the currently handled exception ([except.handle]). [*Note:* This allows the exception to be inspected within the contract-violation handler ([basic.contract.handler]) using std::current_exception ([except.special.general]). — *end note*]that exception is handled by the implementation before the contract-violation handler is invoked. [*Note:* The exception can be inspected within the contract-violation handler ([basic.contract.handler)] using std::contract_violation::predicate_exception ([support.contracts.violation]). — *end note*]

Acknowledgements

Many thanks to Eric Fiselier for valuable discussions regarding this paper.

References

[P2900R9] Joshua Berne, Timur Doumler, and Andrzej Krzemieński: "Contracts for C++". 2024-10-11

[P3227R0] Gašper Ažman and Timur Doumler: "Fixing the library API for contract violation handling". 2024-10-16