Observable Checkpoints During Contract Evaluation

Document #:	P3328R0
Date:	2024-06-14
Project:	Programming Language C++
Audience:	SG21 (Contracts)

Reply-to: Joshua Berne <jberne4@bloomberg.net>

Abstract

If the approach to bounding the impact of undefined behavior introduced in [P1494R3] is adopted, contract assertions as proposed by [P2900R7] can immediately benefit. This paper discusses how the other two proposals should interact and proposes the appropriate changes to the Contracts MVP to benefit from [P1494R3].

Contents

Revision History

Revision 0

- Original version of the paper

1 Introduction

A frequent concern for software engineers is undefined behavior and the wide reach of its impact. In particular, *time travel* optimizations can be of great concern when introducing any new piece of code that might have *any* undefined behavior in it, which, in practice, means basically any not completely trivial piece of code.

The Contracts study group, SG21, has put forth a proposal — [P2900R7] — for introducing a contract-checking facility into C++, and this Contracts facility works within the bounds of the current C++ abstract machine and evaluation model. This design leads to two notable ways in which undefined behavior can interact with contract assertions.

1. When a contract assertion is evaluated with a *checking* semantic, it is evaluated in the same fashion as any other C++ expression. Any *undefined behavior* that occurs when evaluating the predicate of a contract assertion, which is a regular C++ expression, can potentially *time travel* backward and alter the behavior of earlier evaluations:

   ```
   int i = 0;
   void f(int *p)
   {
     if (p != nullptr)  // #1
     {
       ++i;
     }
     contract_assert( *p >= 0 );  // undefined behavior if p == nullptr
   }
   ```

   When the contract assertion in the above code is evaluated with a *checking* semantic, the check at `#1` might be elided, possibly resulting in observing the increment of `i` even when `f(nullptr)` is invoked and the contract assertion is *enforced*.[1]

2. When a contract assertion is observed, similar time travel could potentially elide the contract check itself:

   ```
   void g(int *p)
   {
     contract_assert( p != nullptr );  // #2
     ++*p;  // undefined behavior if p == nullptr
   }
   ```

[1] An *enforced* contract assertion is one that checks the truth of its predicate and invokes the contract-violation handler if a violation is detected, terminating the program if the violation handler returns normally.

When the contract assertion in the above code is evaluated with the *observe* semantic[2] and the compiler has sufficient information about the contract-violation handling process to know that the violation handler will always return normally, the check itself may be elided completely.

Note that for *checking* semantics that do not allow continuation (i.e., *enforce* and *quick_enforce*), the undefined behavior cannot be reached and time travel cannot remove the check itself.

By design, the concept of an *observable checkpoint* introduced in [P1494R3] can be applied to both the above situations to prevent undefined behavior, in or after contract assertions, from time traveling back to alter earlier code. To achieve this, two simple changes need to be made to [P2900R7].

1. First, to prevent undefined behavior in the contract predicate itself from time traveling back to code before the predicate, we must make the beginning of the evaluation of a contract assertion with a *checking* semantic be an *observable checkpoint*.
2. Second, to prevent undefined behavior *after* a predicate from eliding the predicate or the contract-violation handler invocation, we must make returning normally from the contract-violation handler in an *observed* contract assertion into an *observable checkpoint*.

Taken together, the above two changes will severely limit the scope of time travel from undefined behavior in relation to contract assertions.

2 Wording Changes

The proposed wording is relative to [P2900R7] and [P1494R3].

Modify the [intro.abstract] paragraph introduced by [P1494R3] before paragraph 5:

> Certain events in the execution of a program are termed *observable checkpoints*. Program termination is one such. [*Note:* A call to std::observable ([support.start.term]) is also an observable checkpoint~~.~~, as are certain parts of the evaluation of contract assertions ([basic.contract]). — *end note*]

Modify [basic.contract.eval] paragraph 4:

> The evaluation of a contract assertion *A* with a checking semantic (observe, enforce, or quick_enforce) is also called a *contract check*. If the value *B* of the predicate can be determined without evaluating the predicate, that value may be used; otherwise, the predicate is evaluated and *B* is the result of that evaluation. There is an observable checkpoint *C* which happens before *A* such that any other operation *O* that happens before *A* also happens before *C*.

Modify [basic.contract.eval] paragraph 11:

> If the contract-violation handler returns normally and the evaluation semantic is observe, control flow continues normally after the point of evaluation of the contract asser-

[2] An *observed* contract assertion is one that checks the truth of its predicate and invokes the contract-violation handler if a violation is detected, then continues program execution normally if the violation handler returns normally.

tion. There is an observable checkpoint C which happens after the contract-violation handler returns normally such that any other operation O that happens after the contract-violation handler returns also happens after C.

3 Conclusion

To summarize, once [P1494R3] is adopted, the following change should be introduced into the Contracts MVP, [P2900R7]:

Proposal 1: Observable Checkpoints in Contract Assertion Evaluation

The following events in the execution of a program are *observable checkpoints*.

— The beginning of evaluation of a contract predicate when evaluating a contract assertion

— The contract-violation handler returning normally when it is invoked from a contract assertion having the *observe* semantic

Acknowledgements

Thanks to S. Davis Herring for producing [P1494R3], the paper that made this improvement to the Contracts facility in C++ possible, as well as providing feedback on the proposal and wording.

Thanks to Timur Doumler for reviewing this paper, and Lori Hughes for helping it approximate something that is in the English language.

Bibliography

[P1494R3] S. Davis Herring, "Partial program correctness", 2024
http://wg21.link/P1494R3

[P2900R7] Joshua Berne, Timur Doumler, and Andrzej Krzemieński, "Contracts for C++", 2024
http://wg21.link/P2900R7