Document #: P3237R0
Date: 2024-04-15
Project: Programming Language C++
Audience: SG21
Reply-to:
Andrei Zissu <andrziss@gmail.com>

Matrix Representation of Contract Semantics

Contents

Introduction

Contract semantics, as proposed in [P2900R5], already comprise a set of 4 semantics (*ignore, enforce, observe,* and the as yet unnamed “Louis semantic”). Several new semantics are quite likely to be proposed in the foreseeable future. As can be currently seen with the “Louis semantic”, deciding upon proper naming of such semantics incurs considerable difficulties. This is spurred by the justified fear of an increasing number of contract semantics posing ever more challenges to the intuitive grasp of their meaning by the C++ community and even by WG21 members.

We therefore propose a different way of defining contract semantics, which would address the issues described above and even the design of new contract semantics.

Summary of Proposed Changes

We are proposing that contract semantics (as proposed in [P2900R5]) be defined in terms of separate traits, and that they be represented in a matrix comprising said traits. Those traits are not expected to be orthogonal - as seen below, some dependencies are expected to exist between them, thus reducing the number of eligible combinations.

Optionally, we also propose redefining the `contract_semantic` field of class `contract_violation` to incorporate the new traits-based representation.

Motivation

The contracts status quo represented by [P2900R5] currently includes 4 semantics: *ignore, enforce, observe* and "Louis". An *assume* semantic is already viewed as a likely post-MVP proposal. Additional semantics may be proposed, such as the tentative *terminate* semantic in [P3205R0].

Describing such a multitude of semantics and comparing them is quickly becoming a challenge. [P3205R0] tackles that in a way that seems natural and called for – a matrix (this example is from an early version of that paper, which for demonstration purposes is immaterial):

semantic	checks	calls handler	assumed after	terminates	proposed
assume	no	no	yes	no	no
ignore	no	no	no	no	[P2900R5]
"Louis"	yes	no	yes	`trap`-ish	TODO
terminate	yes	no	yes	`std::terminate`	here
observe	yes	yes	no	no	[P2900R5]
ensure	yes	yes	yes	`std::abort`-ish	[P2900R5]

(Note: *enforce* in the above table is erroneously referred to as *ensure.*)

Describing contract semantics in this manner affords us a bird's eye view of all their salient properties, as well as allowing us to spot missing traits which should be properly specified for each new proposed semantic.

In addition to allowing easier reasoning about the differences between various semantics, such a description would also allow us to consider the need for new semantics. This could be done by first determining dependencies between some matrix columns, and from there new semantics allowed by those dependencies might fall out.

For example, let's first determine some dependencies:
!checks implies *!calls_handler* – Not checking the predicate implies the violation handler will never be called. Similarly: *!checks* implies *!terminates*

terminates implies *assumed_after* – A terminating semantic (with whatever termination means) allows the compiler to optimize function code following the contract assertion under the assumption that it will only be

executed in-contract. This assumption holds even in semantics (such as *enforce*) which allow termination to be bypassed, e.g. by throwing an exception.

Such dependencies help us reduce the combinatorial explosion of legal matrix combinations and therefore of possible and sensible semantics. Whatever remains after eliminating illegal combinations may inspire proposals for new semantics.

This paper does not propose any particular set of matrix columns, i.e. contract semantic traits. If this proposal is adopted, we will have a principle by which such separate traits may be proposed (and possibly extended later).

Proposal

We propose that henceforth contract semantics be described as a set of separate traits (unspecified as to its content in this proposal). The full list of available semantics will be visualized as a matrix, with each column representing a semantic trait and each row a contract semantic. The possible values in most columns will span a 1-bit boolean range (*true* or *false*), but in some cases, they may span a larger range, or not be describable in numeric terms at all (a.k.a a "comment" column).

Going back to the earlier matrix example taken from [P3205R0]: *checks, calls handler* and *assumed after* would be 1-bit Boolean columns. *Terminate* already lists 4 possible states, so it would require at least 2 bits (we may want to reserve more bits for future expansion). *Proposed* is a comments column.

As an optional addition, we propose encoding the actionable matrix columns (those which are not comment columns) as a bit field (the names and sizes here should be viewed only as an example). This paper can also be adopted without this addition, in which case it would serve only as procedural terminological guidance.

```
struct contract_semantic {
   bool checks : 1;
   bool calls_handler : 1;
   bool assumed_after : 1;
   int terminates : 2;
};
```

This would in effect redefine the `contract_semantic` field of class `contract_violation` relative to its current definition in [P2900R5].

Were this optional addition to be adopted, we also propose specifying a basic set of semantic aliases (which can be extended later) which would make the most common contract semantics also available via a known name (as per the current [P2900R5] status quo) which would be described via a constexpr definition. Presumably this could also facilitate command line usage, as compiler flags would not necessarily have to be provided individually for each separate contract semantic trait. We would currently propose the following, as public members of `class contract_violation`:

```
constexpr contract_semantic enforce{.checks=true, .calls_handler=true,
.assumed_after=true, .terminates=0x10[1]};

constexpr contract_semantic observe{.checks=true, .calls_handler=true,
.assumed_after=false, .terminates=0};
```

Impact of the Changes

- No impact on current code, as contracts are not yet part of C++.
- Possible changes in the *contract_semantic* enum as proposed in [P2900R5].

Q&A

Wouldn't this create a huge matrix and only complicate things?

Not if we reign it in, by means of carefully defining inter-column dependencies as described in this paper. Thus we would never populate the matrix with semantics made up of trait combinations not permissible as per the defined dependencies.

Wording

To be added later, if needed.

References

[P2900R5] Joshua Berne, Timur Doumler, Andrzej Krzemieński. 2024-02-15. *Contracts for C++*. https://wg21.link/p2900r5

[P3205R0] Gašper Ažman, Jeff Snyder, Andrei Zissu. 2024-04-15. *Throwing from a noexcept function should be a contract violation*. https://isocpp.org/files/papers/P3205R0.pdf

[1] This assumes for now 2 bits for *terminates* - LSB representing yes/no termination via std::terminate() (0/1 respectively) and one additional spare bit.