Document #: P3237R2
Date: 2024-10-23
Project: Programming Language C++
Audience: SG21
Reply-to:
Andrei Zissu <andrziss@gmail.com>

Matrix Representation of Contract Semantics

Contents

Revision History

Revision 0 (April 2024 Mailing)

- Initial revision.

Revision 1 (October 2024 Mailing)

- Properties as bitmasks rather than bit fields.
- Updated some sections.
- Added a code example (link to godbolt).
- Semantic properties are now always represented as a single boolean value.
- Added “questions for SG21” section.
- Added a Q&A paragraph on unsuccessful prior art.

Revision 2 (presented to SG21 after the October 2024 Mailing)

- Properties can now have more than 2 states.
- Code example updated accordingly.
- Various fixes and updates.

Introduction

Contract semantics, as proposed in [P2900R10], already comprise a set of 4 semantics - *ignore, enforce, observe,* and *quick_enforce* (the former "Louis semantic"). Several new semantics are quite likely to be proposed in the foreseeable future. As can be currently seen with the "Louis semantic" which took a while plus some considerable discussion to finally be named *quick_enforce* (see [P3191R0] and [P3226R0]), deciding upon proper naming of such semantics often 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. The outcome of this process may often be a semantic name which does not necessarily induce easy intuition of the properties of said semantic.

We therefore propose a different way of defining contract semantics, which would address the issue of semantic properties intuition and possibly also aid in the discovery of new contract semantics.

Summary of Proposed Changes

We are proposing that contract semantics (as proposed in [P2900R10]]) be defined in terms of separate properties, and that they be represented as a matrix comprising said properties. While technically orthogonal - as seen below, some dependencies are expected to exist between various properties, thus reducing the number of eligible combinations i.e. semantics. In most cases these properties will have 2 boolean states (true/false), but properties with more states (e.g. `terminates`) can also be supported.

We also propose that the only allowed semantics will be those listed in. [P2900R10]], changing only the way they are defined. Thus only the method of defining semantics will change, and not the semantics themselves (nor their names).

Optionally, we also propose redefining the permissible values of the `evaluation_semantic` enum so as to incorporate the new properties-based representation via combinations of power-of-2 values rather than the current sequential enum. Properties with more than 2 states will be represented by an appropriate >2 number of bits. Were this to be accepted, we also propose (library addition) defining a set of constexpr semantic aliases, each of which would define one of the standard semantics. Vendors would also be able to add their own non-standard semantics, thus also being able to experiment with new semantics.

Motivation

The contracts status quo represented by [P2900R10] currently includes 4 semantics: *ignore, enforce, observe* and *quick_enforce*. 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 properties 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. (Note: One could imagine future terminating semantics which may challenge this assumption, e.g. conditional or delayed termination.)

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 properties. If this proposal is adopted, we will have a principle by which such separate properties may be proposed (and possibly extended later). Thus, this paper can be viewed as mainly a policy proposal.

Proposal

We propose that henceforth contract semantics be described in terms of separate properties (unspecified as to their actual content in this proposal). The full list of available semantics will be visualized as a matrix, with each column representing a semantic property and each row a contract semantic. 2-state properties will be described in terms of a single boolean flag with `true` and `false` values. Other properties will be represented by log2(N) bits, where N is the number of states for said property, including the off state. Properties consisting of more than 2 states will be split into the appropriate number of separate single-state properties.

Going back to the earlier matrix example taken from [P3205R0]: *checks, calls handler* and *assumed after* would each be represented as a single property. *Terminate* lists 4 possible states (including the off state), which would be represented by 2 bits, therefore it would need to be defined as 2 properties. *Proposed* is a comments column, not represented as a property.

We further propose that named semantics would still be listed in the standard, but be described in terms of combinations of properties. (Presumably this could also facilitate command line usage, as compiler flags would not necessarily have to be provided individually for each separate contract semantic property.)

In terms of library facilities, we *optionally* propose mandating that the `evaluation_semantic` enum (returned by the member function `evaluation_semantic()` of class `contract_violation)` will contain only power-of-2 values, making it bitmask-friendly . This would enable easy inspection/specification of multiple properties en masse. This would require standardizing the underlying type of `evaluation_semantic` - we believe a 32 or 64 bit integer type would more than suffice for any future expansion, given that new contract semantic properties are expected to be few and far between (unlike new contract semantics, which are combinations thereof). With a 64-bit underlying type, we could reserve the upper 32 bits for vendor extensions - this may only prove unsatisfactory if different vendors may require interactions, thus needing unique universal property values.

With this optional library addition in place, the contract semantics listed in the [P2900R10] status quo) could be described via constexpr definitions. We would currently informally propose the following, semantic aliases:

```
CONTRACT_SEMANTIC(ignore);
CONTRACT_SEMANTIC(observe,      evaluates_predicate);
CONTRACT_SEMANTIC(enforce,      evaluates_predicate,
calls_violation_handler, enforces) + CONTRACT_PROPERTY(terminates,
std_terminate);
CONTRACT_SEMANTIC(quick_enforce, evaluates_predicate, enforces)
+ CONTRACT_PROPERTY(terminates, trapish);
```

See a more elaborate code example, representing the current [P2900R10] status quo, here:
https://compiler-explorer.com/z/exbrv845h

Impact of the Changes

- No impact on current code, as contracts are not yet part of C++.
- Possible changes in the permitted values of the *evaluation_semantic* enum as proposed in [P2900R10].

Q&A

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

Not if we rein 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 property combinations not permissible as per the defined dependencies.

Didn't similar past proposals fail?

Well, yes and no. Several proposals mentioned in [P3227R0] indeed failed to gain traction. However, they proposed contract semantic properties as first class entities (which is also the main objection raised in [P3227R0]), whereas we propose them only as building blocks of contract semantics. Whereas we have decided for now to present semantic aliases (the code part of this proposal) as an optional addition, the fact is that having them as first class entities and precluding any other property combinations as non-standard (though free to experiment with outside the standard) would most likely solve the main stumbling block of previous proposals by limiting the number of standard semantics to the same number presented by the current [P2900R10] status quo, or any other future status quo.

Wouldn't the proposed aliases bring us back to the naming conundrums described as motivation for this proposal?

Well, yes. However, having control over the number of legitimate property combinations (a.k.a semantics) is a stronger motivation. Having said that, having each alias/semantic directly defined as its set of properties will greatly aid in understanding their meaning, writing code referring to individual properties rather than full semantics, and possibly experimentation with new semantics.

Questions for SG21

- Do we agree with the general direction presented in this paper?

- Are we interested in redefining `evaluation_semantic` in terms of a bitmask?
- Are we interested in expressing contract semantics in code in terms of semantic aliases?
- Which underlying type would we like to choose, and how do we divide it between standard and vendor-specific properties? Do the latter need to be universally unique?

Wording

To be added later, depending on SG21 guidance.

Acknowledgements

I wish to thank Timur Doumler and Gašper Ažman for their valuable input and fruitful discussions surrounding this paper and its possible interaction with their [P3227R0] paper. Some of the changes of direction across different revisions of this paper are a direct result of this collaboration.

References

[P2900R10] Joshua Berne, Timur Doumler, Andrzej Krzemieński et al. 2024-10-12. *Contracts for C++*.
https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2024/p2900r10.pdf

[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

[P3227R0] Gašper Ažman, Timur Doumler. 2024-10-16. *Fixing the library API for contract violation handling*.
https://isocpp.org/files/papers/P3227R0.pdf

[P3191R0] Louis Dionne, Yeoul Na, Konstantin Varlamov. 2024-03-19. *Feedback on the scalability of contract violation handlers in P2900.*
https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2024/p3191r0.pdf

[P3226R0] Timur Doumler. 2024-04-12. *Contracts for C++: Naming the "Louis Semantic".*
https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2024/p3226r0.pdf