Document number: P4363R0
Date: 2026-09-07
Audience: EWG
Project: ISO Programming Language C++
Reply-to: Jonathan Grant jgrantonline@gmail.com

C++ Contracts compile-time evaluation semantic: static

Abstract

This paper introduces an additional evaluation semantic for C++ Contracts: **`"static"`**. This is an additional semantic which sits along side the existing: ignore, observe, enforce, quick-enforce evaluation semantics.

The `static` semantic differs from the existing checking semantics in that its contract assertion predicates are subject to analysis purely during translation and have no runtime effect. An implementation is encouraged to issue a diagnostic unless it can establish that the predicate is satisfied.

A contract assertion having the compile-time evaluation semantic has no runtime effect. Its predicate is not evaluated during program execution.

Quoting Sir Tony Hoare 1980 ACM Turing Award Lecture:

"I was eventually persuaded of the need to design programming notations so as to maximize the number of errors which cannot be made, or if made, can be reliably **detected at compile time**."

Motivation

For Functional Safety it's vital to detect issues during development, rather than when running on real machines. By detecting these issues during development it avoids businesses fixing issues in released products, reducing recalls and product updates.

Runtime contract checking necessarily introduces runtime evaluation of contract predicates. Some users instead desire contract assertions to express properties that implementations are encouraged to establish statically, without introducing runtime checking.

The `static` semantic provides this model while permitting implementations to differ in the sophistication of their static analysis.

The proposed semantic follows a prove-or-diagnose model: if an implementation cannot establish that a contract assertion predicate is satisfied, it is encouraged to issue a diagnostic.

A previous proposal, P4021 introduced `compile_assert()`, a single statement form that checks invariant constraints at compile time. This proposal adds the `static` feature as an evaluation semantic for C++ Contracts.

By making the checks at compile-time without observable side-effects they adhere to Hoare Logic {P} Command {Q}

An important design goal is that selecting the compile-time semantic does not alter the runtime behaviour of the program. One approach might be treating pre() and post() as `[[noreturn]]` and if an implementation evaluated expression constraints that are invariant, or an implementation could use only `constexpr`. Compile-time would not allow any exceptions be thrown from within a logical predicate.

If `pre()` and `post()` are allowed runtime effects it would not be Hoare Logic predicates (logical propositions), it would be three commands in a row:

{PreCommand}, Command, {PostCommand}.

The issue with allowing Pre or Post predicates to contain a Command, the Command may depend upon executing the Pre or Post contract itself, which is quite different from Hoare Logic. Pre predicates and Post Predicates are predicates over program state, not commands that may modify that state. Permitting contracts to have observable effects therefore changes the model away from Hoare Logic. Program behaviour should be reasoned about independently of executing additional program behaviour.

The `static` evaluation semantic applies to `contract_assert()` within the Command as well.

As Dave Banham stated:
"a program should be the same program (behaviour-wise) with and without the contract expressions compiled in."

Design

The ability of an implementation to establish that a predicate is satisfied is a matter of implementation quality. Failure to establish that a predicate is satisfied does not make the program ill-formed.

Outcomes

Analysis	Meaning	Outcome
Pass	Predicate is established to be satisfied	No diagnostic
Fail	Predicate is established not to be satisfied	Diagnostic encouraged
Unknown	Analysis cannot establish that the predicate is satisfied	Diagnostic encouraged

Example

```
void func(int x)
    pre (x > 0);

void g(int x)
{
    func(x);
}
```

Analysis cannot establish `(x>0)`, therefore the status is Unknown, and a diagnostic is encouraged.

```
int set_data_idx(container * a, int idx, long data)
    pre(idx < max_index)
    pre(idx >= 0)
{
    a->idx_buf[idx] = data;
}
```

If `set_data_idx(container, -1, 500)` were called a diagnostic is encouraged.

Further discussion

Considered if naming "`compile_time`" or "`verify`" was a better name for the evaluation semantic.

References

P4021 compile_assert() – control-flow enforced conditions at compile-time
https://wg21.link/P4021

P2900 R14 C++ Contracts
https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2025/p2900r14.pdf

Sir Tony Hoare: An axiomatic basis for computer programming. 1969
https://dl.acm.org/doi/epdf/10.1145/363235.363259

Static Analysis of Contracts P3386R1
https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2024/p3386r1.pdf