P3079R0: Should ignore and observe exist for constant evaluation of contracts?

Oliver J.Rosten

Timur's paper: https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2023/p2894r1.pdf
SG21 Reflector discussion
BSI discussion

Agenda

- Thought experiment:

  What if constant evaluation had zero cost?

- Is the cost of constant evaluation sufficient to justify ignore?

Imagine constant evaluation has zero cost

What are the arguments for nevertheless having an ignore semantic at compile time?

- Regularity
- Predicates which can’t be constant expressions
- Predicates which aren’t constant expressions
- Maturity of checks
- The multiverse: anything goes

Regularity

> “Compile time evaluation and runtime evaluation should not be egregiously different if they do not need to be.” - JB

Q. Are the problem spaces sufficiently similar that ignore for one and not the other is egregious?

There is a big difference between

- A live program running a nuclear power plant
- Writing a program to run a nuclear power plant

A. Immediacy of severe consequences suggests that the problem spaces are actually rather different. Therefore, it does not necessarily follow that allowing ignore at runtime but not compile time is egregious. However, that doesn’t imply that there isn’t some degree of irregularity.

Bugs

Ignore at runtime for systems that must keep running:

- I have deployed my software
- If there is a runtime bug or an incorrect contract failure (bug in the contract), my client requires that my program keeps going

Ignore at compile time:

- I’ve shipped a library
- There’s a mistake in a contract
- Clients want to… ignore this or maybe they just want it fixed?

Regularity of handling bugs visible at compile time

Suppose I have shipped a contract with a bug in it

Q. Is this different from any other bug I might have shipped?

A.

- It is similar to bugs in static assertions
- It is different to other kinds of bugs in that a contract should have no effect for correct code

The danger of ignore

My code doesn’t compile; I’ve found a compiler bug!

My code doesn’t compile. The contract must be wrong so I’m going to ignore it.

> “Never try to discourage thinking, for you are sure to succeed.” - Bertrand Russell

What’s wrong with following the ‘standard’ workflow when you encounter a bug in code written by others?

- Report the bug and/or
- Create a workaround and/or
- Fix it locally and/or
- Raise a PR and/or
- Realize while doing this that the bug is actually in your code ;)

Predicates which can’t be constant expressions

From Timur’s paper:

```
constexpr int do_something()
    pre (hardware_thingy_available()); // not constexpr; intrinsically runtime


constexpr bool can_do_something() {
    if consteval { return true; }
    else         { return hardware_thingy_available(); }
}
```

Predicates which aren’t constant expressions (but could be)

Adapted from Timur’s paper

```
bool pred(); // predicate not constexpr

constexpr int f() pre (pred()) { return 42; }


int main() {

    constexpr auto x = f();

}
```

Options

- If the predicate cannot possibly be constexpr, see earlier
- Otherwise
  - Make it constexpr or
  - Use the ignore semantic

Is the latter really what we want?

- Compiler differences may mean something which is ignorable on one platform is necessarily ill-formed on another.
- The path of least resistance could mean people using ignore rather than making things constexpr which arguably should be.

Maturity

I’ve written a new contract check but I might have made a mistake and I don’t want to break everyone’s build

⇒ Use ignore or observe

Really?!

- Do it in a branch
- Make sure everything compiles
- Add tests
- Raise a PR

The multiverse

There is a balance to be struck:

- Allowing people to do anything they want (within reason)
- Facilitating fragmentation (different dialects)
- Facilitating bad practice
  - ignore may be the path of least resistance

We want people to use contracts.

But are we in danger of saying that the only way we can get people to use contracts is if they can ignore them?!

Cost of constant evaluation

What is the intersection of expensive contract checks and things people *actually* do at compile time?

We don’t know.

Options:

- Assume that this may be a problem for some and provide ignore
- Wait until the community requests that this is standardized

Summary

I think more justification for having ignore and observe at compile time is needed

Expense of contract evaluation at compile time *might* be a strong enough reason but I think we need more evidence.

Observation

We can already emulate purely compile-time contracts:

```
constexpr float my_sqrt(float x)
{
    if consteval {
        if(x < 0) throw 1;
    }

    return std::sqrt(x);
}
```

Actually, gcc doesn’t allow sqrt of negative numbers at compile time, anyhow!