Document Number: P4011R0
Date: 2026-02-12
Reply-to: Matthias Kretz <m.kretz@gsi.de>
Audience: EWG

Redefining narrow contract

ABSTRACT

This paper provides a simple revised definition of narrow and wide contracts.

CONTENTS

1 CHANGELOG

(placeholder)

2 STRAW POLLS

(placeholder)

3 THE PROBLEM

Meredith et al. [N3248] defined the terms *narrow* and *wide contract* in terms of *undefined behavior*:

> **Wide Contracts**
>
> A wide contract for a function or operation does not specify any undefined behavior. Such a contract has no preconditions: A function with a wide contract places no additional constraints on its arguments, on any object state, nor on any external global state. Examples of functions having wide contracts would be `vector<T>::begin()` and `vector<T>::at(size_type)`. Examples of functions not having a wide contract would be `vector<T>::front()` and `vector<T>::operator[](size_type)`.
>
> **Narrow Contracts**
>
> A narrow contract is a contract which is not wide. Narrow contracts for a functions or operations result in undefined behavior when called in a manner that violates the documented contract. Such a contract specifies at least one precondition involving its arguments, object state, or some external global state, such as the initialization of a static object. Good examples of standard functions with narrow contracts are `vector<T>::front()` and `vector<T>::operator[](size_type)`.

Let's consider a trivial function `f` that results in UB when its argument is negative:

```
void f(int x) {
  if (x < 0) std::unreachable();
}
```

Clearly, according to the definition above, this function has a precondition: `x >= 0`. If we now encode that precondition using P2900 syntax we get:

```
void f(int x) pre(x >= 0) {
  if (x < 0) std::unreachable();
}
```

The trouble only begins when we introduce syntax that not only encodes the precondition, but at the same time unconditionally removes the undefined behavior:

```
void f(int x) pre!(x >= 0) {
  if (x < 0) std::unreachable();
}
```

Since calling `f` can never lead to UB, the function now has a *wide contract* according to the above definition. This is the reason why some say an unconditionally enforced precondition check is not a contract check anymore.

I believe we need to revise the definition of narrow and wide contract, and thus the implied definition of a *contract check*.

4 A NEW DEFINITION

Definition: A function has a *narrow contract* if there exists at least one input (this includes indirect/global state that affects the function) for which the call is erroneous. Otherwise, the function has a *wide contract*.

Consequence: A human — and potentially also a tool — can tell whether a specific call to a function is a bug in the program or not.

Here "erroneous" does not have to imply erroneous behavior (but see below). This definition aims to decouple the discussion from *behavior* of the function. For some of us this new definition makes no difference, for some it hopefully does. The goal is to help us speak a common language.

5 A CONNECTION TO ERRONEOUS BEHAVIOR?

Consider:

```
int g(int x) pre(x >= 0) {
  if (x < 0) return 0;
  else return x + INT_MIN;
}
```

Here, the function has a precondition but also a certain behavior in case the precondition is not met. This is what we expect of EB: it is still detectable as a bug in the program, but the program can continue without invoking UB.

A BIBLIOGRAPHY

[N3248] Alisdair Meredith and John Lakos. *N3248: `noexcept` Prevents Library Validation*. ISO/IEC C++ Standards Committee Paper. 2011. URL: `https://wg21.link/n3248`.