Disable pointers to contracted functions

Document #:	P3221R0
Date:	2024-04-15
Project:	Programming Language C++
Audience:	SG21 Contracts Working Group
Reply-to:	Jonas Persson <jonas.persson@iar.com>

Contents

1 Motivation **1**
1.1 EWG review in Tokyo . . . 1

2 Proposal **2**

3 Rationale **2**
3.1 Thunk call locations . . . 2
3.2 Unpredictable aborts . . . 2
3.3 Future breaking change . . . 2

4 Workaround **3**

5 Conclusion **3**

6 References **3**

1 Motivation

The contract MVP [P2900R5] does not specify contracts on function pointers. This may be something we want to add, probably post C++26. But until then we need make sure we are in a state that does not prevent its addition at a later time. The currently proposed implicit casting away contract specification needs to be reconsidered.

1.1 EWG review in Tokyo

In the EWG review of the contracts MVP proposal in the 2024 spring meeting in Tokyo there was a poll about requiring support for function pointers in the MVP. There was no clear consensus in any direction.

Poll: P2900r6: contracts should specify contracts on function pointers in its Minimal Viable Proposal.

SF	F	N	A	SA
6	6	14	15	8

It would have been useful with a poll to see if contracts on pointers were seen as vital to the full contract solution. But from the result above we can suspect that there are at least some expectations for that.

2 Proposal

The only realistic alternative to designing a full solution for contracted function pointers is to disallow it for now. That will give us a good starting point for specifying it later, without being constrained by backward compatibility issues.

Proposal: Taking the address of a function or virtual function with pre or post condition is ill formed.

3 Rationale

Why cant we just have pointer to function point to anything as in P2900?
An ordinary function pointer which points to a function with contracts is problematic. It forces an implementation with suboptimal properties, and using such a pointer is not as safe as it could be.

The more desirable solution would be to have contracts on function pointers, and require the pointer and the pointeen to behave in a typesafe manner. That includes making sure contracts is not lost when taking the address of the function.

Allowing the current unsafe behavior of implicit conversion to funciton pointers will obstruct future proposals for safe pointers to functions with contracts, as we will not be able to remove that unsafe implicit conversion once it is out.

3.1 Thunk call locations

A problem for implementation that check contracts on caller side is where to put the checks when the caller is decoupled. The most probable implementation of pointers to functions where a non-contracted pointer is bound to a contracted function, is to let the compiler wrap it in a thunk call. The problem with this is that it will have no location in the source, and unless a source_location is passed as a hidden parameter no information of who the caller was can be reported. This will make it hard to figure out what has gone wrong by only reading contract violation logs. The lack of location also applies to static analysis tools which want to have some place to point at where it can recommend adding some defensive code. Wrapping the checks also hinders eliding checks. Using contracted pointers can make these problems go away.

3.2 Unpredictable aborts

Allowing a non-contracted function pointer to point to a function with a precondition is dangerous. As it does not expose the contract of the pointee, a caller may call through it in good faith but get an unexpected contract violation. This is difficult to keep under control, especially if the pointer is passed to a different part of the code base, as it may need a global analysis to track the usage.

For this reason it is better to include the contract in the function pointer type contract and only allow assignment of “safe” functions, with equal or more generous contracts. Until such function pointers are specified it better to not encourage such practises.

3.3 Future breaking change

There is a good chance for future function pointers to have contracts in their type, or in some other way attached to the pointer instances. If the MVP allow functions to decay into ordinary function pointers, it will be a breaking change to later enforce matching contracts between pointer and function.
The alternative of not statically checking for compatible contracts when assigning function pointers is not very compelling and will make contracts on function pointers less useful.

The current MVP will make it really hard to get it into the language when the time comes.

P2900	Future	This
`void fun(A& a) pre (a.ok());`	`void fun(A& a) pre (a.ok());`	`void fun(A& a) pre (a.ok());`
`using F = void(A& a);`	`using F = void(A& a);`	`using F = void (A& a);`
	`using G = void(A& a) pre(a.ok()`	
`F* f = fun;  // Ok`	`F* f = fun; // Unsafe`	`F* f = fun;  // Error`
	`G* g = fun;  // Ok`	`F* g = +[](A&a){fun(a)}; // Ok`

4 Workaround

There are of course a lot of places that takes ordinary pointera and where is nessessary to pass in a function with contracts.

Even with the limitatations we introduce here it will always be possible to convert a contracted function to a function pointer by wrapping the function in a captureless lambda and let that convert to a pointer. We then avoid the future breakage and the hidden thunk call problems. But unfortunately not the unpredictable aborts. This has benefits even when when we dont have contracted pointers. It works like an explicit cast that clealy states in the code where the statically typed contracts are removed.

5 Conclusion

Taking all this into consideration the conclusion should be that, unless we never want contracts on function pointers, and perhaps even then, all uses of indirect calls to functions with contracts should be prohibited until a proper solution is put forward.

6 References

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