

Borrowing Trouble: The Difficulties Of A C++ Borrow-Checker



Authors: danakj@chromium.org, lukasza@chromium.org, palmer@chromium.org
Publication Date: 10th September 2021

Introduction

A common question raised when comparing C++ and Rust is whether the Rust borrow checker is really unique to Rust, or if it can be implemented in C++ too. C++ is a very flexible language, so it seems like it should be possible. In this article we'll explore if it is possible to do borrow checking at compile time in C++.

Some background on C++ efforts

Many folks are working on [improving C++](#), including improving its memory safety. [Clang](#) has [experimental -Wlifetime warnings](#) to help catch a class of use-after-free bugs. The cases it catches are typically [dangling references to temporaries](#), which makes them a valuable set of warnings to enable when it is available. But the cases it would solve do not seem to intersect with the set of cases [MiraclePtr](#) is attempting to protect against, which is an effort to frustrate

Merging state and references breaks ownership

If we accept that we can modify the language to make `HasMut<T>` and `HasRef<T>` non-destructible, and to enforce they are not used after a move, then we might consider to go a step further and do away with these troublesome types.

We might try to instead make the reference types `MutRef<T>` and `Ref<T>` not-publicly-destructible but also movable with a destructive move. Then we can eliminate the `HasMut` and `HasRef` types, and encode those states by the existence of the reference types.

However, that allows a method to steal ownership from a reference. By constructing a `Uniq<T>` from a `MutRef<T>`, ownership is taken without being passed a `Uniq<T>` explicitly. Thus we actually need the states representing `HasMut` and `HasRef` to remain in the original scope of the `Uniq<T>` they are transitioned from in order to return ownership back to the same scope (though not the same variable).

Conclusion

We attempted to represent ownership and borrowing through the C++ type system, however the language does not lend itself to this. Thus memory safety in C++ would need to be achieved through runtime checks.