



Products

# Zero width reference for borrow checking only?

[Log in](#) [sign up](#)

[Ask Question](#)

Asked 2 years, 4 months ago

Active 2 years, 4 months ago

Viewed 54 times



1



My goal is to enforce an invariant in my code using the borrow checker at zero-cost. However, to make it zero-cost, it seems like I'd need to have a zero-width reference. Here's the context:

I want to make make a factory for some objects, with a way to invalidate all the objects at once.

```
/// Example struct.
/// In my real code, there's stuff in here,
/// but its unnecessary for the question I'masking.
struct Obj;

/// A struct that lets you build new `Obj`s
struct ObjFactory;

impl ObjFactory {
    /// makes a new object
    fn make(&self) -> Obj {
        Obj
    }

    /// enforce that there are no objects from this ObjFactory
    fn recall(&self) {
        // ?????
    }
}
```

The rules I want are the same as the rules that the borrow checker enforces, so I added some dummy references that make the borrow checker follow the invariant at compile time.

```
/// new placeholder struct that should take up no space
struct Dummy;

/// Example struct.
/// Now has a dummy reference so the compiler knows when to get mad
struct Obj<a>(&a Dummy);

/// A struct that lets you build new `Obj`s
struct ObjFactory {dummy: Dummy}

impl ObjFactory {

    /// makes a new object
    fn make(&self) -> Obj {
        // let the `Obj` immutably borrow the dummy
        Obj(&self.dummy)
    }

    /// enforce that there are no objects from this ObjFactory
    fn recall(&mut self) {
        // mutably borrow `dummy`, which means that Obj's can't borrow it anymore.
        let _borrow = &mut self.dummy;
    }
}
```

Now, the compiler can detect when the invariant is broken:

```
fn main() {

    // make factory
    let mut obj_factory = ObjFactory {dummy: Dummy};

    // create an object
    let obj = obj_factory.make();

    // uh oh! no objects are allowed!
    obj_factory.recall();

    // obj lasts until the end of the function
    core::mem::drop(obj);
}
```

The compiler correctly detects that there's an object that exists when `recall` happens.

## Your privacy

By clicking "Accept all cookies", you agree Stack Exchange can store cookies on your device and disclose information in accordance with our [Cookie Policy](#).

[Accept all cookies](#)

[Customize settings](#)

```

error[E0502]: cannot borrow `obj_factory` as mutable because it is also borrowed as immutable
--> src/main.rs:4:5
|
| 3 | let obj = obj_factory.make();
|   |           ^^^^^^^^^ immutable borrow occurs here
| 4 | obj_factory.recall();
|   | ~~~~~ mutable borrow occurs here
| 5 | core::mem::drop(obj);
|   |   ^ immutable borrow later used here

```

This works exactly the way I want; however, it isn't zero cost. Even though each `Obj`'s dummy is zero width, it still takes up space in the `Obj` :

```
print!("{}", std::mem::size_of:<Obj>()); // 8
```

How should I go about getting the space usage of `Obj` down to 0, while also making the borrow-checker enforce the `ObjFactory` invariants?

[rust](#) [borrow-checker](#)

Share  
 Improve this  
 question  
 Follow

asked Aug 5 '19 at 4:57

   
 users5770298

Possible duplicate of [In Rust, how do you explicitly tie the lifetimes of two objects together, without referencing eachother?](#)

– [trent](#) formerly el

Aug 5 '19 at 9:13

@trentel The way I see it, I'm not just trying to tie the lifetimes together, I'm also trying to take advantage of the mutable / immutable borrow shennanigans from the borrow checker.

– users5770298

Aug 5 '19 at 16:23

[Add a comment](#)

1 Answer

[Active](#) [Oldest](#) [Votes](#)

▲

1

▼



You can use `PhantomData` to have an empty type with a lifetime:

```

use core::marker::PhantomData;

/// Example struct.
struct Obj<'a>{PhantomData<&'a ()>};

/// A struct that lets you build new `Obj`s
struct ObjFactory;

impl ObjFactory {
    /// makes a new object
    fn make(&self) -> Obj<'_> {
        Obj(PhantomData) // the PhantomData has the same lifetime as `self`
    }

    /// enforce that there are no objects from this ObjFactory
    fn recall(&mut self) {}
}

fn main() {
    // make factory
    let mut obj_factory = ObjFactory;

    // create an object
    let obj = obj_factory.make();


    // uh oh! no objects are allowed!
    obj_factory.recall();

    // obj lasts until the end of the function
    core::mem::drop(obj);
}

```

Share  
 Improve this answer  
 Follow

answered Aug 5 '19 at 7:02

   
 mcarton  
 22.7k • 5 • 63 • 75



[Add a comment](#)

Your Answer



Post Your Answer

By clicking "Post Your Answer", you agree to our [terms of service](#), [privacy policy](#) and [cookie policy](#)

#### The Overflow Blog

-  Sequencing your DNA with a USB dongle and open source code
-  Don't push that button: Exploring the software that flies SpaceX rockets and...

#### Featured on Meta

-  Providing a JavaScript API for userscripts
-  Congratulations to the 59 sites that just left Beta

#### Linked

In Rust, how do you explicitly tie the lifetimes of two objects together, without referencing eachother?

#### Related

6

cannot move out of borrowed content when unwrapping a member variable in a &mut self method

Struct with immutable reference to other struct

Returning a reference from a HashMap or Vec causes a borrow to last beyond the scope it's in?

Is it possible to borrow parts of a struct as mutable and other parts as immutable?

2






Can't borrow reference to structure in captured tree because it doesn't live long enough

How to deal with rust borrow checker

1

Prevent cannot borrow '\*self' as immutable because it is also borrowed as mutable when accessing disjoint fields in struct?

#### Hot Network Questions

-  FEM for vector valued problems: reference request
  -  Problems that are polynomially "hard" to compute but "easy" to verify
  -  How to salvage bitter homemade mustard?
  -  How to plot molecules with angles and bond lengths
  -  What edges are not in a Gabriel graph, yet in a Delauney graph?
- [more hot questions](#)

 Question feed

#### STACK OVERFLOW

[Questions](#)  
[Jobs](#)  
[Developer Jobs Directory](#)  
[Salary Calculator](#)  
[Help](#)  
[Mobile](#)

#### PRODUCTS

[Teams](#)  
[Talent](#)  
[Advertising](#)  
[Enterprise](#)

#### COMPANY

[About](#)  
[Press](#)  
[Work Here](#)  
[Legal](#)  
[Privacy Policy](#)  
[Terms of Service](#)  
[Contact Us](#)

[Cookie Settings](#)  
[Cookie Policy](#)

#### STACK EXCHANGE NETWORK

[Technology](#)  
[Culture & recreation](#)  
[Life & arts](#)  
[Science](#)  
[Professional](#)  
[Business](#)  
[API](#)  
[Data](#)

[Blog](#)  
[Facebook](#)  
[Twitter](#)  
[LinkedIn](#)  
[Instagram](#)

site design / logo © 2021 Stack Exchange Inc; user contributions licensed under [cc by-sa](#). rev 2021.12.22.41046