Borrow rules violation with disjoint borrows

help

ytausky March 4, 2019, 12:27pm 1

I have the following code:

```
struct A {
   field: Option<i32>,
}

impl A {
   fn foo(&mut self) -> &i32 {
       if let Some(field) = self.field.as_ref() {
           return field;
       }
       self.field = Some(42);
       self.field.as_ref().unwrap()
   }
}
```

(Playground link)
I'm getting the following error:

```
error[E0506]: cannot assign to `self.field` because it is borrowed
 --> src/lib.rs:10:9
  |
6  |     fn foo(&mut self) -> &i32 {
  |            - let's call the lifetime of this reference `'1`
7  |         if let Some(field) = self.field.as_ref() {
  |                              ---------- borrow of `self.field` occurs here
8  |             return field;
  |                    ----- returning this value requires that `self.field` is borrowed for `'1`
9  |         }
10 |         self.field = Some(42);
  |         ^^^^^^^^^^^^^^^^^^^^^ assignment to borrowed `self.field` occurs here
```

On the one hand, I understand what's going on in terms of lifetimes, but on the other hand, it seems like there should be a way to make this work, since the assignment to `self.field` is mutually exclusive with the return statement. How can I solve this?

dodomorandi March 4, 2019, 12:37pm 2

This is exactly what you need for this simple case:

```
fn foo(&mut self) -> &i32 {
   self.field.get_or_insert(42)
}
```

But I am not sure for a more general case.

ytausky March 4, 2019, 12:41pm 3

This is just a simplified version of my real code, but `Option::get_or_insert`'s implementation hints at the solution. I'd still be interested in a nicer way to avoid this problem, though...

jonh March 4, 2019, 4:05pm 4

Without function style you typically have to change code around.

```
        if self.field.is_none() {
           self.field = Some(42);
       }
       self.field.as_ref().unwrap()
```

Yandros March 4, 2019, 10:07pm 5

Hmm, if we change `if let Some(field) = self.field.as_ref()` to `if let Some(ref field) = self.field` the code compiles o_O

```
struct A {
   field: Option<i32>,
}

impl A {
   fn foo(&mut self) -> &i32 {
       if let Some(ref field) = self.field {
           return field;
       }
       self.field = Some(42);
       self.field.as_ref().unwrap()
   }
}
```

I think this is easier to see with an unsugared match

```
impl A {
   fn foo(&mut self) -> &i32 {
       match self.field.as_ref() { // preemptive borrow starts here ...
           Some(field) => {
               field
           },
           _ => { self.field = Some(42); self.field.as_ref().unwrap() },
       } // ... and only ends at the end of the whole match
       // hence the error
   }
}
```

```
impl A {
   fn foo(&mut self) -> &i32 {
       match self.field {
           Some(ref field) => { // borrow starts here ...
               field
           }, // ... and ends here
           _ => { self.field = Some(42); self.field.as_ref().unwrap() },
       }
       // hence valid
   }
}
```

2 Likes

harmic March 5, 2019, 1:03am 6

I would have thought this:

```
    if let Some(field) = self.field.as_ref() {
       return field;
   }
```

would desugar into this:

```
    match self.field.as_ref() {
       Some(field) => {
           field
       },
       _ => (),
   }
```

in which case the scope of the borrow should end at the end of the scope of the `if let`?

I found this similar issue but it is talking about what happens in the `else` arm of an `if let`.

vitalyd March 5, 2019, 2:21am 7

I don't think what @Yandros wrote is the actual desugaring. The issue really is what Polonius will hopefully address one day.

Yandros March 5, 2019, 8:40am 8

`return` statements make the analysis more complicated (your error message mentions that); I think that a decent "approximation" in this case to how the compiler thinks is to rewrite your function without it.

```
impl A {
   fn foo(&mut self) -> &i32 {
       if let Some(field) = self.field.as_ref() {
           field
       } else {
           self.field = Some(42);
           self.field.as_ref().unwrap()
       }
   }
}
```

And then my desugaring does apply.

Aside: imperative `return`

If we really wanted to see how non-polonius thinks about return values, I'm pretty sure it is something along these lines:

```
impl A {
   fn foo(&mut self) -> &i32 {
       let ret_value: &i32; // always the first local
       if let Some(field) = self.field.as_ref() {
           ret_value = field;
       } else {
           self.field = Some(42);
           ret_value = self.field.as_ref().unwrap();
       }
       ret_value
   }
}
```

Which is why, when a value is returned, its lexical scope does not end before the very bottom of the function (and this is the counter-intuitive part about it, that polonius should address)

1 Like

Related topics

Topic		Replies	Views	Activity
Returning mutable refererences to optional self fields help	3	425	November 7, 2023	
Borrow checker false positive? help	5	1024	May 16, 2020	
E0502 when borrowing in if let help	6	152	December 28, 2024	
Modifying and returning reference in match statement help	9	179	October 20, 2025	
How to return a reference to a `Vec` help	4	855	August 6, 2022	