



Products

# Why can Rust not infer the proper lifetime in simple closures, or infers they are conflicting?

Login or sign up

Ask Question

Asked 5 years, 10 months ago

Active 5 years, 9 months ago

Viewed 398 times



7



1



I have not found any rules in the Rust documentation that would explain how lifetime elision applies to closures. Let's take a simple example:

```
fn foo(s: &str) {  
    let id = |x: &str| x;  
    println!("{}", id(s));  
}  
  
fn main() {  
    foo("string");  
}
```

I thought that the closure in the `foo` function would work similar to the following code:

```
fn foo(s: &str) {  
    struct Id; // A helper structure for closure  
    impl Id {  
        fn id(self: Self, x: &str) -> &str { &x }  
    }  
    let id = Id; // Creating a closure  
    println!("{}", id.id(s));  
}
```

The latter works fine, but the former fails to compile and produces a long error message about conflicting lifetime requirements:

```
t3.rs:2:24: 2:25 error: cannot infer an appropriate lifetime due to conflicting requirements [E0495]  
t3.rs:2   let id = |x: &str| x;  
           ^  
  
t3.rs:2:24: 2:25 note: first, the lifetime cannot outlive the anonymous lifetime #1 defined on the block at 2:23...  
t3.rs:2   let id = |x: &str| x;  
           ^  
  
t3.rs:2:24: 2:25 note: ...so that expression is assignable (expected `&str`, found `&str`)  
t3.rs:2   let id = |x: &str| x;  
           ^  
  
<std macros>:3:11: 3:36 note: but, the lifetime must be valid for the expression at 3:10...  
<std macros>:3 print ! ( concat ! ( $ fmt , "n" ) , $ ( $ arg ) * ) );  
           ^  
  
<std macros>:2:25: 2:56 note: in this expansion of format_args!  
<std macros>:3:1: 3:54 note: in this expansion of print! (defined in <std macros>)  
t3.rs:3:5: 3:27 note: in this expansion of println! (defined in <std macros>)  
<std macros>:3:11: 3:36 note: ...so type `(&&str,)` of expression is valid during the expression  
<std macros>:3 print ! ( concat ! ( $ fmt , "n" ) , $ ( $ arg ) * ) );  
           ^  
  
<std macros>:2:25: 2:56 note: in this expansion of format_args!  
<std macros>:3:1: 3:54 note: in this expansion of print! (defined in <std macros>)  
t3.rs:3:5: 3:27 note: in this expansion of println! (defined in <std macros>)  
error: aborting due to previous error
```

I wonder why Rust cannot infer the proper lifetime in simple closures like the one that I wrote above. Moreover, why does the compiler think that there are *conflicting* requirements for lifetime.

[rust](#) [lifetime](#)

Share

Improve this

question

Follow

edited Feb 29 '16 at 14:59



Shepmaster

305k • 59 • 824 • 1083

asked Feb 29 '16 at 13:10



svat

136 • 6

Remove the `&str` and it works. The `&str` there doesn't mean what you think it means. I have no time to explain now, however, as I should be in bed.

– Chris Morgan

Feb 29 '16 at 13:31

## 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.

Actually, I meant `&str`, because it is necessary in a bit more complex case. In any case, my question was not how to get this trivial example to work, but what are the rules here? Why does the compiler finds the conflicting requirements here?

[Accept all cookies](#) [Customize settings](#)

Feb 29 '16 at 13:39

@ChrisMorgan: If you have some time, it would be great if you could explain what's going on. I have a feeling it could be due to an inferred `for<a>` but it's not quite clear... and the lack of answer after 20 hours seems to mean I am not the only one unsure of what's going on ^^

– Matthieu M.  
Mar 1 '16 at 9:52

[Add a comment](#)

## 2 Answers

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



3



When you specify the type of a parameter or the return type in a closure, and that type is a reference, the compiler sets wrong expectations on the implicit lifetime parameter, and there's no way to define the lifetime parameter explicitly on a closure. [This is a known issue](#). The workaround is to omit the parameter's type or the return type and let the compiler infer everything.

```
fn foo(s: &str) {
    let id = |x| {
        println!("{}", id(s));
    };
}

fn main() {
    foo("string");
}
```

If you still need to give a type hint, you can do so with a `let` binding inside the closure:

```
fn foo(s: &str) {
    let id = |x| { let y: &str = x; y };
    println!("{}", id(s));
}

fn main() {
    foo("string");
}
```

[Share](#)

[Improve this answer](#)

Follow

answered Mar 14 '16 at 4:40



[Francis Cagné](#)

**50.1k** • 3 • 138 • 127

[Add a comment](#)



0



The closure can't infer the lifetime from the method signature. Effectively you're defining a lifetime in the method signature let's say it's implicitly `'a`, and another lifetime, implicitly `'b` in the closure.

Match up the lifetimes and it will compile.

```
fn foo<'a>(s: &'a str) {
    let id = |x: &'a str| x;
    println!("{}", id(s));
}

fn main() {
    foo("string");
}
```

[Share](#)

[Improve this answer](#)

Follow

edited Mar 1 '16 at 15:42



[Shepmaster](#)

**305k** • 59 • 824 • 1083

answered Mar 1 '16 at 15:29



[master\\_of\\_rust](#)

1

---

I am sorry if I worded my question badly, but I was interested in the *rules* for lifetime inference in closures, not how to get the code to compile. Also I tried a version with a static string: `id("string")`, and Rust still cannot infer the lifetime correctly and produce the same error. If Rust does not support lifetime inference in closures, I have no problem to specify it explicitly, but the error message said about *conflicting\_requirements*, which I find very confusing.

– svat

Mar 2 '16 at 5:15

[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](#)

Not the answer you're looking for? Browse other questions tagged [rust](#) [lifetime](#) or ask your own question.

#### 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

[Visit chat](#)

#### Related

[How do I create an array of unboxed functions / closures?](#)

[Allowing reference lifetime to outlive a closure](#)

[How do I specify a lifetime that is dependent on the borrowed binding of a closure in a separate type?](#)

[Rust error E0495 using split\\_at\\_mut in a closure](#)






[3 Why do I get the error "cannot infer an appropriate lifetime for lifetime parameter in generic type" when using nested mutable references?](#)

[Recursive closure as function parameter "cannot infer an appropriate lifetime due to conflicting requirements"](#)

[Can I coerce a lifetime parameter to a shorter lifetime \(soundly\) even in the presence of '&mut T'?](#)

[Specify Rust closures lifetime](#)

#### Hot Network Questions

-  [What do I do when my boss is sabotaging interviews?](#)
-  [Given many questions as to whether Jesus was born on 25 December or not, I ask if the ambiguity in scripture is meant to teach us something?](#)
-  [Could mass timber construction techniques have been used in the past?](#)
-  [How to force Mathematica to do infinite-precision calculations?](#)
-  [What's the social meaning of "He was a student of..."?](#)

[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