Restrict lifetime parameter to scope of parameters of a function

Asked 5 years, 11 months ago
Active 3 years, 6 months ago
Viewed 162 times

4

1

Consider the following example

```
trait MyTrait<'a> {
    type N: 'a;

    fn func(&'a self) -> Self::N;
}

fn myfunc<'a, T: 'a + MyTrait<'a>>(g: T) {
    g.func();
}

fn main() {}
```

Compiling this small program fails with:

```
error[E0597]: `g` does not live long enough
 --> src/main.rs:8:5
  |
8 |     g.func();
  |     ^ borrowed value does not live long enough
9 | }
  | - borrowed value only lives until here
  |
note: borrowed value must be valid for the lifetime 'a as defined on the function body at 7:1...
 --> src/main.rs:7:1
  |
7 | fn myfunc<'a, T: 'a + MyTrait<'a>>(g: T) {
  | ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

As far as I understand, the lifetime parameter `'a` is not restricted and could be arbitrary. However, `g` is a parameter and its lifetime is only the function scope, therefore it does not satisfy the condition of lifetime `'a` in the definition of method `func`.

What I really want is that the associated type `N` is always restricted to the lifetime of `self` in `MyTrait`. That's why I came up with the explicit lifetime parameter `'a` of `MyTrait`. I want function `myfunc` to work, i.e. `'a` should somehow be restricted to the lifetime of of the parameter `g`.

What is the "correct" way to solve this problem?

A very simple example is

```
struct MyPtr<'a> {
    x: &'a usize,
}

struct MyStruct {
    data: Vec<usize>,
}

impl<'a> MyTrait<'a> for MyStruct {
    type N = MyPtr<'a>;

    fn func(&'a self) -> Self::N {
        MyPtr { x: &self.data[0] }
    }
}
```

Note that this is extremely simplified, of course. The idea is that `N` always contains a reference to something contained in `MyTrait` and should therefore never outlive `MyTrait`.

rust lifetime

Share
Improve this question
Follow

edited Jun 28 '18 at 17:14
Shepmaster
305k • 59 • 824 • 1083

asked Jan 27 '16 at 13:17
fifr
138 • 5

Why do you want to constrain `N` to the lifetime of `self`? Could you show us an example implementation of `MyTrait`? Maybe you just need to change `&'a self` to `&self`.
– Francis Gagné
Jan 27 '16 at 13:34

“What I really want is that the associated type N is always restricted to the lifetime of self .” I’d like to point out that the : in the context of lifetimes means “outlives”, not the other way around. So N: 'a does not really restrict N , it just says that it lives at least as long as self , which should be true anyway for the struct type to be well-formed.
– kirelagin
Jan 27 '16 at 14:03

Nevertheless, you still might need to put it down explicitly, e.g. for your example to work. I just wanted to make it clear, that this “restriction” works the other way around, you do not restrict it to the lifetime of self , you demand that it lives longer than self .
– kirelagin
Jan 27 '16 at 14:11

Add a comment

1 Answer

Active Oldest Votes

3

What you want is not to bind a generic lifetime, but to allow "any" lifetime:

```
fn myfunc<T: for<'a> MyTrait<'a>>(g: T) {
    g.func();
}
```

Fully working example in the playground.

The best source for an explanation is How does for<> syntax differ from a regular lifetime bound?.

Share
Improve this answer
Follow

edited Jun 28 '18 at 17:15
Shepmaster
305k ● 59 ● 824 ● 1083

answered Jan 27 '16 at 13:56
oli_obk
24.5k ● 2 ● 72 ● 88

1

Thanks a lot. Where can I find something about this strange construct (I've never seen that before)?
– fifr
Jan 27 '16 at 14:27

1

@fifr I’m afraid, RFC 1214 is the only official source that I’m aware of that kind of tries to explain for<…> , which is, well, unfortunate.
– kirelagin
Jan 27 '16 at 14:36

The best source for an explanation for the for<...> syntax is now stackoverflow.com/a/35595491/1103681
– oli_obk
Feb 25 '16 at 10:33

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

Linked

How does for<> syntax differ from a regular lifetime bound?

Related

What is the lifetime of a static variable in a C++ function?

Expected bound lifetime parameter, found concrete lifetime [E0271]

Struct needs a lifetime because?

Return struct with lifetime for FFI

How can this instance seemingly outlive its own parameter lifetime?

Problems with Tuple's lifetime in rust.

How does one restrict a lifetime to a closure environment in rust?

Hot Network Questions

How do I get the http endpoint to work for cardano-wallet?

FEM for vector valued problems: reference request

Without passport stamps, can my country's authorities still know what countries I've visited?

Are they already planning a successor to the JWST?

What's the largest REG_SZ value that Regedit can edit?

more hot questions

Question feed