# "use of moved value" when matching while merging two vectors

Asked 5 years, 8 months ago

Active 5 years, 8 months ago

Viewed 788 times

0

I am writing a merge function for vectors of tags with counts, but am getting borrowing errors.

```
fn merge(mut l1: Vec<(String, u32)>, mut l2: Vec<(String, u32)>) -> Vec<(String, u32)> {
    let mut d1 = l1.drain(..);
    let mut d2 = l2.drain(..);
    let mut result = Vec::new();
    let mut v1 = d1.next();
    let mut v2 = d2.next();
    loop {
        match (v1, v2) {
            (None, None) => return result,
            (None, Some(x)) => {
                result.push(x.clone());
                v2 = d2.next()
            }
            (Some(x), None) => {
                result.push(x.clone());
                v1 = d1.next()
            }
            (Some(p1), Some(p2)) => {
                let (ref s1, t1) = p1;
                let (ref s2, t2) = p2;
                if s1 == s2 {
                    result.push((s1.clone(), t1 + t2));
                    v1 = d1.next();
                    v2 = d2.next();
                } else if s1 < s2 {
                    result.push(p1.clone());
                    v1 = d1.next();
                } else {
                    result.push(p2.clone());
                    v2 = d2.next();
                }
            }
        }
    }
}
```

gives the error:

```
error: use of moved value: `v1` [E0382]
         match (v1,v2) {
                ^~
help: run `rustc --explain E0382` to see a detailed explanation
note: `v1` was previously moved here because it has type `core::option::Option<(collections::string::String, u32)>`, which is non-copyable
```

and a similar error for `v2`. It usually shows the problem location and the previous move that causes the problem, but not here.

I've tried many permutations, and with the following change I've gotten it to compile, but I'm not happy about all the cloning and recreating tuples and recreating `Option`s.

```
match (v1, v2) {
    (None, None) => return result,
    (None, Some(x)) => {
        result.push(x.clone());
        v1 = None;
        v2 = d2.next();
    }
    (Some(x), None) => {
        result.push(x.clone());
        v1 = d1.next();
        v2 = None;
    }
    (Some(p1), Some(p2)) => {
        let (ref s1, t1) = p1;
        let (ref s2, t2) = p2;
        if s1 == s2 {
            result.push((s1.clone(), t1 + t2));
            v1 = d1.next();
            v2 = d2.next();
        } else if s1 < s2 {
            result.push(p1.clone());
            v1 = d1.next();
            v2 = Some((s2.clone(), t2));
        } else {
            result.push(p2.clone());
            v1 = Some((s1.clone(), t1));
            v2 = d2.next();
        }
    }
}
```

Adding what I'd really like to write, for reference, in case someone is looking for a challenge for the borrow checker:

```
fn merge(mut l1: Vec<(String, u32)>, mut l2: Vec<(String, u32)>) -> Vec<(String, u32)> {
    let mut d1 = l1.drain(..);
    let mut d2 = l2.drain(..);
    let mut result = Vec::new();
    let mut v1 = d1.next();
    let mut v2 = d2.next();
    loop {
        match (v1, v2) {
            (None, None) => return result,
            (None, Some(p2)) => {
                result.push(p2);
                v1 = None;
                v2 = d2.next()
            }
            (Some(p1), None) => {
                result.push(p1);
                v1 = d1.next();
                v2 = None
            }
            (Some(p1 @ (s1, _)), o2 @ Some((s2, _))) if s1 < s2 => {
                result.push(p1);
                v1 = d1.next();
                v2 = o2
            }
            (o1 @ Some((s1, _)), Some(p2 @ (s2, _))) if s1 > s2 => {
                result.push(p2);
                v1 = o1;
                v2 = d2.next()
            }
            (Some((s1, t1)), Some((_, t2))) => {
                result.push((s1, t1 + t2));
                v1 = d1.next();
                v2 = d2.next()
            }
```

Note that the match on `(v1, v2)` should move the values so that each path is enforced to set `v1` and `v2`. Still not as clean as Haskell, but closer.

rust borrowing

Share
Improve this question
Follow

edited Apr 18 '16 at 12:50
Shepmaster
**305k** ● 59 ● 824 ● 1083

asked Apr 17 '16 at 3:05
Dave Mason
**427** ● 3 ● 9

---

Did you mean to paste two different examples of your match block? They look the same to me.
– Jimmy
Apr 17 '16 at 7:47

1

the second one assigns to both v1 and v2 on every branch, but have excessive clone operations
– Dave Mason
Apr 17 '16 at 13:28

Add a comment

## 1 Answer

Active Oldest Votes

1

Variables `v1` and `v2` move out when creating a tuple in the `match` expression. You need to modify these variables inside the `match`, so you can't borrow them.

With `Option<T>` you can use take() method:

```
fn merge(mut l1: Vec<(String, u32)>, mut l2: Vec<(String, u32)>) -> Vec<(String, u32)> {
    let mut d1 = l1.drain(..);
    let mut d2 = l2.drain(..);
    let mut result = Vec::new();
    let mut v1 = d1.next();
    let mut v2 = d2.next();
    loop {
        match (v1.take(), v2.take()) {//Takes the value out of the option, leaving a None in its place.
            (None, None) => return result,
            (None, Some(x)) => {
                result.push(x);
                v2 = d2.next()
            }//v1 is None
            (Some(x), None) => {
                result.push(x);
                v1 = d1.next()
            }//v2 is None
            (Some(p1), Some(p2)) => {
                use std::cmp::Ordering::{Equal, Less, Greater};
                match p1.0.cmp(&p2.0) {
                    Equal => {
                        result.push((p1.0, p1.1 + p2.1));
                        v1 = d1.next();
                        v2 = d2.next();
                    }
                    Less => {
                        result.push(p1);
                        v1 = d1.next();
                        v2 = Some(p2);
                    }//restore v2
                    Greater => {
                        result.push(p2);
                        v1 = Some(p1); //restore v1
                        v2 = d2.next();
```

I have altered the code of the last branch to avoid unnecessary borrowing.

Disadvantage of this approach is that you may forget to assign a new value to a variable. I would recommend to return the values from the `match` expression:

```
fn merge(mut l1: Vec<(String, u32)>, mut l2: Vec<(String, u32)>) -> Vec<(String, u32)> {
    let mut d1 = l1.drain(..);
    let mut d2 = l2.drain(..);
    let mut result = Vec::new();
    let mut v = (d1.next(), d2.next());
    loop {
        v = match (v.0.take(), v.1.take()) {
            (None, None) => return result,
            (None, Some(x)) => {
                result.push(x);
                (None, d2.next())
            }
            (Some(x), None) => {
                result.push(x);
                (d1.next(), None)
            }
            (Some(p1), Some(p2)) => {
                use std::cmp::Ordering::{Equal, Less, Greater};
                match p1.0.cmp(&p2.0) {
                    Equal => {
                        result.push((p1.0, p1.1 + p2.1));
                        (d1.next(), d2.next())
                    }
                    Less => {
                        result.push(p1);
                        (d1.next(), Some(p2))
                    }
                    Greater => {
                        result.push(p2);
                        (Some(p1), d2.next())
                    }
                }
            }
        };
```

Removed unnecessary `clone` s as mentioned by @mcarton

Share
Improve this answer
Follow

edited Apr 18 '16 at 12:50

Shepmaster
**305k** ● 59 ● 824 ● 1083

answered Apr 17 '16 at 11:01

aSpex
**3,906** ● 12 ● 20

---

1

There is also no need for all those `clone` s thanks to the `drain` ed iterators.
– mcarton
Apr 17 '16 at 11:19

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 borrowing 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

Related

0
How do I include ':Send' in the type signature?

5
FnOnce inside Enum: cannot move out of borrowed content

Returning a borrow from an owned resource

Why doesn't the lifetime of a mutable borrow end when the function call is complete?

How to write a proper generic function signature when borrowing data across multiple traits

How to get Result's Ok value if the Result is in a Vector?

Why I cannot borrow something I borrowed and why I can move a method's self?

Rust: Cannot reference local variable in return value - but the "local variable" is passed to the caller

Hot Network Questions

Schrödinger's cat program

Why are nerves blocked even though potassium chanels are not blocked?

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?

Applied mathematics or Computer science PhD?

On what basis do countries repay international loans?

more hot questions

Question feed