

`CoerceUnsized` was implemented on a struct which does not contain a field with an `unsized` type.

Example of erroneous code:

```
#![feature(coerce_unsized)]
use std::ops::CoerceUnsized;

struct Foo<T: ?Sized> {
    a: i32,
}

// error: Struct `Foo` has no unsized fields that need `CoerceUnsized`.
impl<T, U> CoerceUnsized<Foo<U>> for Foo<T>
    where T: CoerceUnsized<U> {}
```

An `unsized` type is any type where the compiler does not know the length or alignment of at compile time. Any struct containing an `unsized` type is also `unsized`.

`CoerceUnsized` is used to coerce one struct containing an `unsized` type into another struct containing a different `unsized` type. If the struct doesn't have any fields of `unsized` types then you don't need explicit coercion to get the types you want. To fix this you can either not try to implement `CoerceUnsized` or you can add a field that is `unsized` to the struct.

Example:

```
#![feature(coerce_unsized)]
use std::ops::CoerceUnsized;

// We don't need to impl `CoerceUnsized` here.
struct Foo {
    a: i32,
}

// We add the unsized type field to the struct.
struct Bar<T: ?Sized> {
    a: i32,
    b: T,
}

// The struct has an unsized field so we can implement
// `CoerceUnsized` for it.
impl<T, U> CoerceUnsized<Bar<U>> for Bar<T>
    where T: CoerceUnsized<U> {}
```

Note that `CoerceUnsized` is mainly used by smart pointers like `Box`, `Rc` and `Arc` to be able to mark that they can coerce `unsized` types that they are pointing at.