A reference has a longer lifetime than the data it references.

Erroneous code example:

```
struct Foo<'a> {
    x: fn(&'a i32),
}

trait Trait<'a, 'b> {
    type Out;
}

impl<'a, 'b> Trait<'a, 'b> for usize {
    type Out = &'a Foo<'b>; // error!
}
```

Here, the problem is that the compiler cannot be sure that the `'b` lifetime will live longer than `'a`, which should be mandatory in order to be sure that `Trait::Out` will always have a reference pointing to an existing type. So in this case, we just need to tell the compiler than `'b` must outlive `'a`:

```
struct Foo<'a> {
    x: fn(&'a i32),
}

trait Trait<'a, 'b> {
    type Out;
}

impl<'a, 'b: 'a> Trait<'a, 'b> for usize { // we added the lifetime enforcement
    type Out = &'a Foo<'b>; // it now works!
}
```