A type parameter which references `Self` in its default value was not specified.

Erroneous code example:

```
trait A<T=Self> {}

fn together_we_will_rule_the_galaxy(son: &A) {}
// error: the type parameter `T` must be explicitly specified in an
//        object type because its default value `Self` references the
//        type `Self`
```

A trait object is defined over a single, fully-defined trait. With a regular default parameter, this parameter can just be substituted in. However, if the default parameter is `Self`, the trait changes for each concrete type; i.e. `i32` will be expected to implement `A<i32>`, `bool` will be expected to implement `A<bool>`, etc... These types will not share an implementation of a fully-defined trait; instead they share implementations of a trait with different parameters substituted in for each implementation. This is irreconcilable with what we need to make a trait object work, and is thus disallowed. Making the trait concrete by explicitly specifying the value of the defaulted parameter will fix this issue. Fixed example:

```
trait A<T=Self> {}

fn together_we_will_rule_the_galaxy(son: &A<i32>) {} // Ok!
```