The `self` parameter in a method has an invalid "receiver type".

Erroneous code example:

```
struct Foo;
struct Bar;

trait Trait {
    fn foo(&self);
}

impl Trait for Foo {
    fn foo(self: &Bar) {}
}
```

Methods take a special first parameter, of which there are three variants: `self`, `&self`, and `&mut self`. These are syntactic sugar for `self: Self`, `self: &Self`, and `self: &mut Self` respectively.

```
# struct Foo;
trait Trait {
    fn foo(&self);
//         ^^^^^ `self` here is a reference to the receiver object
}

impl Trait for Foo {
    fn foo(&self) {}
//         ^^^^^ the receiver type is `&Foo`
}
```

The type `Self` acts as an alias to the type of the current trait implementer, or "receiver type". Besides the already mentioned `Self`, `&Self` and `&mut Self` valid receiver types, the following are also valid: `self: Box<Self>`, `self: Rc<Self>`, `self: Arc<Self>`, and `self: Pin<P>` (where P is one of the previous types except `Self`). Note that `Self` can also be the underlying implementing type, like `Foo` in the following example:

```
# struct Foo;
# trait Trait {
#     fn foo(&self);
# }
impl Trait for Foo {
    fn foo(self: &Foo) {}
}
```

This error will be emitted by the compiler when using an invalid receiver type, like in the following example:

```
# struct Foo;
# struct Bar;
# trait Trait {
#     fn foo(&self);
# }
impl Trait for Foo {
```

```
    fn foo(self: &Bar) {}
}
```

The nightly feature Arbitrary self types extends the accepted set of receiver types to also include any type that can dereference to `Self`:

```
#![feature(arbitrary_self_types)]

struct Foo;
struct Bar;

// Because you can dereference `Bar` into `Foo`...
impl std::ops::Deref for Bar {
    type Target = Foo;

    fn deref(&self) -> &Foo {
        &Foo
    }
}

impl Foo {
    fn foo(self: Bar) {}
//         ^^^^^^^^^ ...it can be used as the receiver type
}
```