An attempt was made to constrain an associated type.

Erroneous code example:

```
pub trait Vehicle {
    type Color;
}

pub trait Box {
    type Color;
}

pub trait BoxCar : Box + Vehicle {}

fn dent_object<COLOR>(c: dyn BoxCar<Color=COLOR>) {} // Invalid constraint
```

In this example, <code>BoxCar</code> has two supertraits: <code>Vehicle</code> and <code>Box</code>. Both of these traits define an associated type <code>Color</code>. <code>BoxCar</code> inherits two types with that name from both supertraits. Because of this, we need to use the fully qualified path syntax to refer to the appropriate <code>Color</code> associated type, either <code><BoxCar</code> as <code>Vehicle>::Color</code> or <code><BoxCar</code> as <code>Box>::Color</code>, but this syntax is not allowed to be used in a function signature.

In order to encode this kind of constraint, a where clause and a new type parameter are needed:

```
pub trait Vehicle {
   type Color;
pub trait Box {
  type Color;
pub trait BoxCar : Box + Vehicle {}
// Introduce a new `CAR` type parameter
fn foo<CAR, COLOR>(
   c: CAR,
) where
    // Bind the type parameter `CAR` to the trait `BoxCar`
   CAR: BoxCar,
   // Further restrict `<BoxCar as Vehicle>::Color` to be the same as the
    // type parameter `COLOR`
   CAR: Vehicle<Color = COLOR>,
   // We can also simultaneously restrict the other trait's associated type
   CAR: Box<Color = COLOR>
{ }
```