A type mismatched an associated type of a trait.

Erroneous code example:

```
trait Trait { type AssociatedType; }

fn foo<T>(t: T) where T: Trait<AssociatedType=u32> {
//                    ~~~~~~~~ ~~~~~~~~~~~~~~~~~~
//                        |            |
//         This says `foo` can         |
//           only be used with         |
//              some type that         |
//         implements `Trait`.         |
//                                     |
//                             This says not only must
//                             `T` be an impl of `Trait`
//                             but also that the impl
//                             must assign the type `u32`
//                             to the associated type.
    println!("in foo");
}

impl Trait for i8 { type AssociatedType = &'static str; }
//~~~~~~~~~~~~~~~   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//      |                             |
// `i8` does have                     |
// implementation                     |
// of `Trait`...                      |
//                     ... but it is an implementation
//                     that assigns `&'static str` to
//                     the associated type.

foo(3_i8);
// Here, we invoke `foo` with an `i8`, which does not satisfy
// the constraint `<i8 as Trait>::AssociatedType=u32`, and
// therefore the type-checker complains with this error code.
```

The issue can be resolved by changing the associated type:

1. in the `foo` implementation:

```
trait Trait { type AssociatedType; }

fn foo<T>(t: T) where T: Trait<AssociatedType = &'static str> {
    println!("in foo");
}

impl Trait for i8 { type AssociatedType = &'static str; }

foo(3_i8);
```

2. in the `Trait` implementation for `i8`:

```rust
trait Trait { type AssociatedType; }

fn foo<T>(t: T) where T: Trait<AssociatedType = u32> {
    println!("in foo");
}

impl Trait for i8 { type AssociatedType = u32; }

foo(3_i8);
```