An implementation cannot be chosen unambiguously because of lack of information.

Erroneous code example:

```
trait Generator {
    fn create() -> u32;
}

struct Impl;

impl Generator for Impl {
    fn create() -> u32 { 1 }
}

struct AnotherImpl;

impl Generator for AnotherImpl {
    fn create() -> u32 { 2 }
}

fn main() {
    let cont: u32 = Generator::create();
    // error, impossible to choose one of Generator trait implementation
    // Should it be Impl or AnotherImpl, maybe something else?
}
```

This error can be solved by adding type annotations that provide the missing information to the compiler. In this case, the solution is to use a concrete type:

```
trait Generator {
    fn create() -> u32;
}

struct AnotherImpl;

impl Generator for AnotherImpl {
    fn create() -> u32 { 2 }
}

fn main() {
    let gen1 = AnotherImpl::create();

    // if there are multiple methods with same name (different traits)
    let gen2 = <AnotherImpl as Generator>::create();
}
```