This error indicates a violation of one of Rust's orphan rules for trait implementations. The rule concerns the use of type parameters in an implementation of a foreign trait (a trait defined in another crate), and states that type parameters must be "covered" by a local type.

When implementing a foreign trait for a foreign type, the trait must have one or more type parameters. A type local to your crate must appear before any use of any type parameters.

To understand what this means, it is perhaps easier to consider a few examples.

If `ForeignTrait` is a trait defined in some external crate `foo`, then the following trait `impl` is an error:

```
# #[cfg(for_demonstration_only)]
extern crate foo;
# #[cfg(for_demonstration_only)]
use foo::ForeignTrait;
# use std::panic::UnwindSafe as ForeignTrait;

impl<T> ForeignTrait for T { } // error
# fn main() {}
```

To work around this, it can be covered with a local type, `MyType`:

```
# use std::panic::UnwindSafe as ForeignTrait;
struct MyType<T>(T);
impl<T> ForeignTrait for MyType<T> { } // Ok
```

Please note that a type alias is not sufficient.

For another example of an error, suppose there's another trait defined in `foo` named `ForeignTrait2` that takes two type parameters. Then this `impl` results in the same rule violation:

```
ignore (cannot-doctest-multicrate-project) struct MyType2; impl<T>
ForeignTrait2<T, MyType<T>> for MyType2 { } // error
```

The reason for this is that there are two appearances of type parameter `T` in the `impl` header, both as parameters for `ForeignTrait2`. The first appearance is uncovered, and so runs afoul of the orphan rule.

Consider one more example:

```
ignore (cannot-doctest-multicrate-project) impl<T> ForeignTrait2<MyType<T>,
T> for MyType2 { } // Ok
```

This only differs from the previous `impl` in that the parameters `T` and `MyType<T>` for `ForeignTrait2` have been swapped. This example does *not* violate the orphan rule; it is permitted.

To see why that last example was allowed, you need to understand the general rule. Unfortunately this rule is a bit tricky to state. Consider an `impl`:

```
ignore (only-for-syntax-highlight) impl<P1, ..., Pm> ForeignTrait<T1,
..., Tn> for T0 { ... }
```

where `P1, ..., Pm` are the type parameters of the `impl` and `T0, ..., Tn` are types. One of the types `T0, ..., Tn` must be a local type (this is another orphan rule, see the explanation for E0117).

Both of the following must be true: 1. At least one of the types `T0..=Tn` must be a local type. Let `Ti` be the first such type. 2. No uncovered type parameters `P1..=Pm` may appear in `T0..Ti` (excluding `Ti`).

For information on the design of the orphan rules, see RFC 2451 and RFC 1023.

For information on the design of the orphan rules, see RFC 1023.