A `where` clause contains a nested quantification over lifetimes.

Erroneous code example:

```
trait Tr<'a, 'b> {}

fn foo<T>(t: T)
where
    for<'a> &'a T: for<'b> Tr<'a, 'b>, // error: nested quantification
{
}
```

Rust syntax allows lifetime quantifications in two places within `where` clauses: Quantifying over the trait bound only (as in `Ty: for<'l> Trait<'l>`) and quantifying over the whole clause (as in `for<'l> &'l Ty: Trait<'l>`). Using both in the same clause leads to a nested lifetime quantification, which is not supported.

The following example compiles, because the clause with the nested quantification has been rewritten to use only one `for<>`:

```
trait Tr<'a, 'b> {}

fn foo<T>(t: T)
where
    for<'a, 'b> &'a T: Tr<'a, 'b>, // ok
{
}
```