`negative_impls`

The tracking issue for this feature is #68318.

With the feature gate `negative_impls`, you can write negative impls as well as positive ones:

```
#![feature(negative_impls)]
trait DerefMut { }
impl<T: ?Sized> !DerefMut for &T { }
```

Negative impls indicate a semver guarantee that the given trait will not be implemented for the given types. Negative impls play an additional purpose for auto traits, described below.

Negative impls have the following characteristics:

- They do not have any items.
- They must obey the orphan rules as if they were a positive impl.
- They cannot "overlap" with any positive impls.

Semver interaction

It is a breaking change to remove a negative impl. Negative impls are a commitment not to implement the given trait for the named types.

Orphan and overlap rules

Negative impls must obey the same orphan rules as a positive impl. This implies you cannot add a negative impl for types defined in upstream crates and so forth.

Similarly, negative impls cannot overlap with positive impls, again using the same "overlap" check that we ordinarily use to determine if two impls overlap. (Note that positive impls typically cannot overlap with one another either, except as permitted by specialization.)

Interaction with auto traits

Declaring a negative impl `impl !SomeAutoTrait for SomeType` for an auto-trait serves two purposes:

- as with any trait, it declares that `SomeType` will never implement `SomeAutoTrait`;
- it disables the automatic `SomeType: SomeAutoTrait` impl that would otherwise have been generated.

Note that, at present, there is no way to indicate that a given type does not implement an auto trait *but that it may do so in the future*. For ordinary types, this is done by simply not declaring any impl at all, but that is not an option for auto traits. A workaround is that one could embed a marker type as one of the fields, where the marker type is `!AutoTrait`.

Immediate uses

Negative impls are used to declare that `&T: !DerefMut` and `&mut T: !Clone`, as required to fix the soundness of `Pin` described in #66544.

This serves two purposes:

- For proving the correctness of unsafe code, we can use that impl as evidence that no `DerefMut` or `Clone` impl exists.
- It prevents downstream crates from creating such impls.