An "or" pattern was used where the variable bindings are not consistently bound across patterns.

Erroneous code example:

```
match x {
    Some(y) | None => { /* use y */ } // error: variable `y` from pattern #1 is
                                      //        not bound in pattern #2
    _ => ()
}
```

Here, `y` is bound to the contents of the `Some` and can be used within the block corresponding to the match arm. However, in case `x` is `None`, we have not specified what `y` is, and the block will use a nonexistent variable.

To fix this error, either split into multiple match arms:

```
let x = Some(1);
match x {
    Some(y) => { /* use y */ }
    None => { /* ... */ }
}
```

or, bind the variable to a field of the same type in all sub-patterns of the or pattern:

```
let x = (0, 2);
match x {
    (0, y) | (y, 0) => { /* use y */}
    _ => {}
}
```

In this example, if `x` matches the pattern `(0, _)`, the second field is set to `y`. If it matches `(_, 0)`, the first field is set to `y`; so in all cases `y` is set to some value.