```
newvar x:=e1 in (newvar y:=e2 in c ) ? (?)
newvar y:=e2 in (newvar x:=e1 in c)
```

No son equivalentes. Se advierte que podría haber problemas si x ocurrirera libre en e2 y/o la var. y ocurriera libre en e1.

```
? newvar x:=e1 in (newvar y:=e2 in c ?? =

(\? . [? | x : ?x ])? ( ?newvar y:=e2 in c?[ ? | x : ?e1? ? ] ) =

(\? . [? | x : ?x ])?
   ( (\? . [? | y : ?y ])? ( ?c?[ [ ? | x : ?e1? ? ]
                                   | y : ?e2? [ ? | x : ?e1? ? ] ] ) =

? newvar y:=e2 in (newvar x:=e1 in c ?? =

(\? . [? | y : ?y ])? ( ?newvar x:=e1 in c?[ ? | y : ?e2? ? ] ) =

(\? . [? | y : ?y ])?
   ( (\? . [? | x : ?x ])? ( ?c?[ [ ? | y : ?e2? ? ]
                                   | x : ?e1? [ ? | y : ?e2? ? ] ] ) =
```

Para refutar la equivalencia buscamos en qué casos

```
?c?[ [ ? | x : ?e1? ? ] | y : ?e2? [ ? | x : ?e1? ? ] ] )
?c?[ [ ? | y : ?e2? ? ] | x : ?e1? [ ? | y : ?e2? ? ] ] )
```

modifiquen de distinta manera una variable que no sea x ni y. Esto es porque estas variables son restauradas en su valor original, así que no podremos apreciar ninguna diferencia entre

```
? newvar x:=e1 in (newvar y:=e2 in c ??  y
? newvar y:=e2 in (newvar x:=e1 in c ??
```

mirando esas variables.

Por ejemplo tomemos

```
c = (z := y)
e1 = 1
e2 = x + 1

?z:=y?[ [ ? | x : 1 ] | y : ?x+1? [ ? | x : 1 ] ]  =
?z:=y?[ [ ? | x : 1 ] | y : 2 ]  =
[ ? | x : 1 ] | y : 2 ] | z : 2]

?z:=y?[ [ ? | y : ?x+1?? ] | x : 1 ]  =
?z:=y?[ [ ? | y : ? x+1 ] | x : 1 ]  =
[ ? | x : 1 ] | y : 2 ] | z : ?x+1 ]
```

Luego

```
? newvar x:=e1 in (newvar y:=e2 in c ?? z es distinto de
? newvar y:=e2 in (newvar x:=e1 in c ?? z
```