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 ║σ =

(\σ . [σ | x : σx ])╨ ( ║newvar y:=e2 in c║[ σ | x : ║e1║ σ ] ) =

(\σ . [σ | x : σx ])╨

( (\σ . [σ | y : σy ])╨ ( ║c║[ [ σ | x : ║e1║ σ ]

| y : ║e2║ [ σ | x : ║e1║ σ ] ] ) =

║ newvar y:=e2 in (newvar x:=e1 in c ║σ =

(\σ . [σ | y : σy ])╨ ( ║newvar x:=e1 in c║[ σ | y : ║e2║ σ ] ) =

(\σ . [σ | y : σy ])╨

( (\σ . [σ | x : σx ])╨ ( ║c║[ [ σ | y : ║e2║ σ ]

| x : ║e1║ [ σ | y : ║e2║ σ ] ] ) =

Para refutar la equivalencia buscamos en qué casos

║c║[ [ σ | x : ║e1║ σ ] | y : ║e2║ [ σ | x : ║e1║ σ ] ] )

║c║[ [ σ | y : ║e2║ σ ] | x : ║e1║ [ σ | y : ║e2║ σ ] ] )

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 ║σ y

║ newvar y:=e2 in (newvar x:=e1 in c ║σ

mirando esas variables.

Por ejemplo tomemos

c = (z := y)

e1 = 1

e2 = x + 1

║z:=y║[ [ σ | x : 1 ] | y : ║x+1║ [ σ | x : 1 ] ] =

║z:=y║[ [ σ | x : 1 ] | y : 2 ] =

[ σ | x : 1 ] | y : 2 ] | z : 2]

║z:=y║[ [ σ | y : ║x+1║σ ] | x : 1 ] =

║z:=y║[ [ σ | y : σ x+1 ] | x : 1 ] =

[ σ | x : 1 ] | y : 2 ] | z : σx+1 ]

Luego

║ newvar x:=e1 in (newvar y:=e2 in c ║σ z es distinto de

║ newvar y:=e2 in (newvar x:=e1 in c ║σ z