```
{-# OPTIONS --guardedness #-}

open import Codata.Musical.Notation
open import Data.Nat using (; suc; zero)
open import Relation.Binary.Core using (Rel)
open import Relation.Binary.Bundles using (Setoid)
open import Relation.Binary.Definitions using (Reflexive; Symmetric; Transitive)
open import Relation.Binary.PropositionalEquality using (__; subst; subst) renaming (sym to eqSym; trans to
import Level using (zero)
open import Data.Maybe using (Maybe; nothing; just)
open import Data.Maybe.Properties
open import Data.Bool using (Bool; true; false)
open import Data.Product
open import Data.Sum
open import Function.Base using (case_of_)
open import Relation.Nullary using (contradiction)

module Trace3 where

Id : Set
Id =

Val : Set
Val =

State : Set
State = Id → Val



record Trace : Set where
  coinductive
  constructor mkTr
  field
    hd : State
    tl : Maybe Trace
open Trace



record __ (tr tr : Trace) : Set where
  coinductive
  field
    hd : hd tr  hd tr
    tl : (tl tr  nothing × tl tr  nothing)
```

```
 {A = (Trace × Trace)}  x → (
tl tr  just (proj x)
×
tl tr  just (proj x)
×
(proj x)  (proj x))
```

```
postulate
  test :  {tr tr tr : Trace} →
    tr  tr → tr  tr → tr  tr
```