```
{-# 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)

open import nakata.Traces hiding (module Trace)
open import nakata.Language

module latex.Trace1 where




data Trace : Set where
  tnil : State → Trace
  tcons : State →  Trace → Trace




data __ : Rel Trace Level.zero where
  tnil :  {st} → (tnil st)  (tnil st)
  tcons :  {st tr tr}
    → ( tr   tr)
    → (tcons st tr)  (tcons st tr)



mutual

  data exec : Stmt → State → Trace → Set where
    execWhileLoop :
      {c : Expr} {b : Stmt}
      {st : State} (tr tr : Trace)
  → (isTrue (c st))  true
  → execseq b (tcons st ( tnil st)) tr
  → execseq (Swhile c b) tr tr
  → exec (Swhile c b) st tr

data execseq : Stmt → Trace → Trace → Set where
 execseqNil : {st : State} {s : Stmt} {tr : Trace}
  → exec s st tr
  → execseq s (tnil st) tr

 execseqCons : {s : Stmt}
  (st : State) (tr tr :   Trace)
  →  (execseq s ( tr) ( tr))
  → execseq s (tcons st tr) (tcons st tr)
```