```
{-# 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 Data.Nat

open import nakata.Traces
open import nakata.Language
open import nakata.BigRel hiding (execDeterministic)

open exloop hiding (exloopincrementing; incrementingFrom; incrementingtrace; increasing; incrementingAlwa
open Trace

module latex.ProofProgram where





incrementingFrom : State → Trace
incrementingFrom st = tcons st ( tcons st ( (incrementingFrom (next st))))

incrementingtrace : Trace
incrementingtrace = incrementingFrom startState

exloopincrementing : exec (Swhile ( _ → 1) (Sassign 0 add1)) startState incrementingtrace
exloopincrementing = t startState
  where
    t : (st : State) → exec (Swhile ( _ → 1) (Sassign 0 add1)) st (incrementingFrom st)
    t st = execWhileLoop
        (tcons st ( (tcons st ( (tnil (update 0 (add1 st) st))))))
        _
        __.refl
        (execseqCons st _ _ ( execseqNil (execAssign (tcons ( tnil)))))
        (execseqCons st _ _ ( (execseqCons st _ _ ( (execseqNil (t (next st)))))))


postulate
  trace : Trace
  program : Stmt
  fromState : State
  proof : exec program fromState trace


data increasing : Id → Val → Trace → Set where
  increasingCons : {id : Id} {v : Val} {st : State} {tr tr : Trace}
    → st id  v
    → tr  tcons st ( (tcons st ( tr)))
    →  (increasing id (suc v) tr)
    → increasing id v tr

incrementingAlwaysIncrements : increasing 0 0 incrementingtrace
incrementingAlwaysIncrements = forever refl
  where
    open Setoid setoid using () renaming (refl to refl)
    open import Relation.Binary.PropositionalEquality
    open -Reasoning

    lem : {x : } → x + 1  suc x
    lem {zero} = refl
    lem {suc x} = begin
      suc (x + 1)
      ⟨⟩
      suc x + suc zero
      ⟨ cong suc (lem) ⟩
      suc (suc x)


    lem : {v : Val} → (st : State) → (st 0  v) → next st 0  suc v
    lem {v} st x = begin
      next st 0
      ⟨⟩
      st 0 + 1
      ⟨ cong (_+ 1) x ⟩
      v + 1
      ⟨ lem ⟩
      suc v
    forever : {st : State} {v : Val} → (st 0  v) → increasing 0 v (incrementingFrom st)
    forever {st} x = increasingCons x (tcons ( (tcons ( refl)))) ( forever (lem st x))


postulate
  execDeterministic : {s : Stmt} {st : State} {tr tr : Trace}
    → exec s st tr
    → exec s st tr
    → tr  tr
```