Practical Verification of Infinite Structure using AGDA2HS

Author: Remco Schrijver, r.schrijver-1@student.tudelft.nl Supervisors: Jesper Cockx & Lucas Escot

TU Delft Delft University of Technology

Research Question and Aim

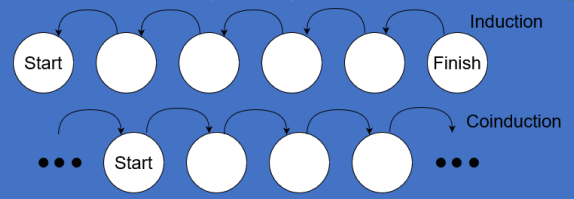
1

- Is it possible for AGDA2HS to translate infinite and cyclic structures that rely on co-induction in Agda to the concept of infinite structures used in Haskell, and if not can AGDA2HS be extended to be able to allow for this translation?

Introduction to infinite structures

2

- Agda uses coinduction for infinite structures



Translating InfiniteList to Haskell[1]

3

```
record InfiniteList (a : Set) : Set where
    coinductive
    field
        hd : a
        tl : InfiniteList a
{-# COMPILE AGDA2HS InfiniteList #-}

open InfiniteList public

fibonacci : Nat → Nat → InfiniteList Nat
hd (fibonacci n1 n2) = n1
tl (fibonacci n1 n2) = (fibonacci (n2) (n1 +++ n2))
{-# COMPILE AGDA2HS fibonacci #-}

_!!!_ : {a : Set} → InfiniteList a → Nat → a
list !!! Zero = hdInf list
list !!! Suc n = (tlInf list) !!! n
{-# COMPILE AGDA2HS _!!!_ #-}
```

```
data InfiniteList a = InfiniteList{hd :: a, tl :: InfiniteList a}

fibonacci :: Nat -> Nat -> InfiniteList Nat
fibonacci n1 n2 Data.InfiniteList.InfiniteList.hd = n1
fibonacci n1 n2 Data.InfiniteList.InfiniteList.tl
  = fibonacci n2 (n1 +++ n2)

(!!!) :: InfiniteList a -> Nat -> a
list !!! Zero = hdInf list
list !!! Suc n = tlInf list !!! n
```

Sized types might fix failing copattern translation[1]

4

```
data CoList (a : Set) (@0 i : Size) : Set where
  Nil  : CoList a i
  _:::_ : a → Thunk (CoList a) i → CoList a i
{-# COMPILE AGDA2HS CoList #-}

fibonacciCoList : {@0 i : Size} → Nat → Nat → CoList Nat i
fibonacciCoList n1 n2 = n1 ::: (λ where .force →
  (n2 ::: λ where .force → (fibonacciCoList n2 (n1 +++ n2))))
{-# COMPILE AGDA2HS fibonacciCoList #-}

_!!!_ : {a : Set} {@0 i : Size} → CoList a ∞ → Nat → Maybe a
Nil !!! _ = Nothing
(x ::: xs) !!! Zero = Just x
(x ::: xs) !!! Suc n = (xs .force) !!! n
{-# COMPILE AGDA2HS _!!!_ #-}
```

```
data CoList a = Nil
              | (:::) a (Thunk (CoList a))

fibonacciCoList :: Nat -> Nat -> CoList Nat
fibonacciCoList n1 n2
  = n1 :::
      \case
        Data.Thunk.Thunk.force -> n2 :::
        \case
          Data.Thunk.Thunk.force -> fibonacciCoList n2 (n1 +++ n2)

(!!!) :: CoList a -> Nat -> Maybe a
Nil !!! _ = Nothing
(x ::: xs) !!! Zero = Just x
(x ::: xs) !!! Suc n = force xs !!! n
```

Results

5

- Coinductive structures and sized types translation fine, but can be enhanced.
- In both cases functions that create of modify infinite structures do not translate, copatterns are the main problem.

Future research

6

- Cyclic structures can be infinitely traversed, but are not strictly infinite so these might behave differently in translation?
- Creating methods in Agda to mimic list generation and comprehension.

[1]: Examples are fully available on GitHub: https://github.com/RemcoSchrijver/verification-of-infinite-structures/tree/paper_reference