A Verified Implementation of Sequences

Research Project by Shashank Anand
Supervised by Jesper Cockx and Lucas Escot

TU Delft

Introduction

1. Haskell is a purely functional programming language, which makes it easy to reason about the correctness of programs
2. However, proofs largely "on paper", hence difficult to verify.
3. In a dependently typed language like Agda, proofs can be written in the language and verified by the type checker
4. Agda2Hs is a recent initiative to translate Agda to Haskell, enabling verification in Agda, with code in Haskell

Coming Soon

1. Verification of all identified properties

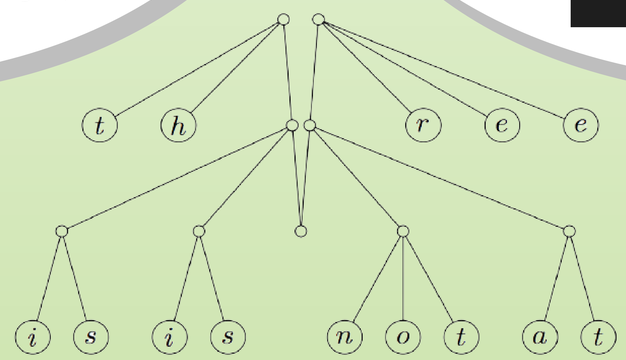
```
length xs ≡ lengthT (toList xs)
```

2. Port to Haskell using agda2hs

```
data Seq (a : Set) : Set where
  Sequence : FingerTree (Elem a) -> Seq a
{-# COMPILE AGDA2HS Seq #-}
```

↓

```
data Seq a = Sequence (FingerTree (Elem a))
```



Sequences

1. Sequences are an alternative implementation of lists.
2. Implemented using finger trees (Hinze and Patterson 2006)
3. Constant time access to ends (fingers), and logarithmic time concatenation
4. Provided in the Data.Sequence library in Haskell

	type	[a]	Seq a
head	:: Seq a -> a	O(1)	O(1)
tail	:: Seq a -> Seq a	O(1)	O(log n)
cons	:: a -> Seq a -> Seq a	O(1)	O(log n)
last	:: Seq a -> a	O(n)	O(1)
init	:: Seq a -> Seq a	O(n)	O(log n)
snoc	:: Seq a -> a -> Seq a	O(n)	O(log n)
(++)	:: Seq a -> Seq a -> Seq a	O(n)	O(log n)
(!!)	:: Seq a -> Int -> a	O(n)	O(log n)

Research Question

Can agda2hs be used to produce a verified implementation of the Sequence library?

Progress

1. Implementation of Sequence in Agda complete

```
data Seq (a : Set) : Set where
  Sequence : FingerTree (Elem a) -> Seq a
{-# COMPILE AGDA2HS Seq #-}
```

All Done

2. Properties identified