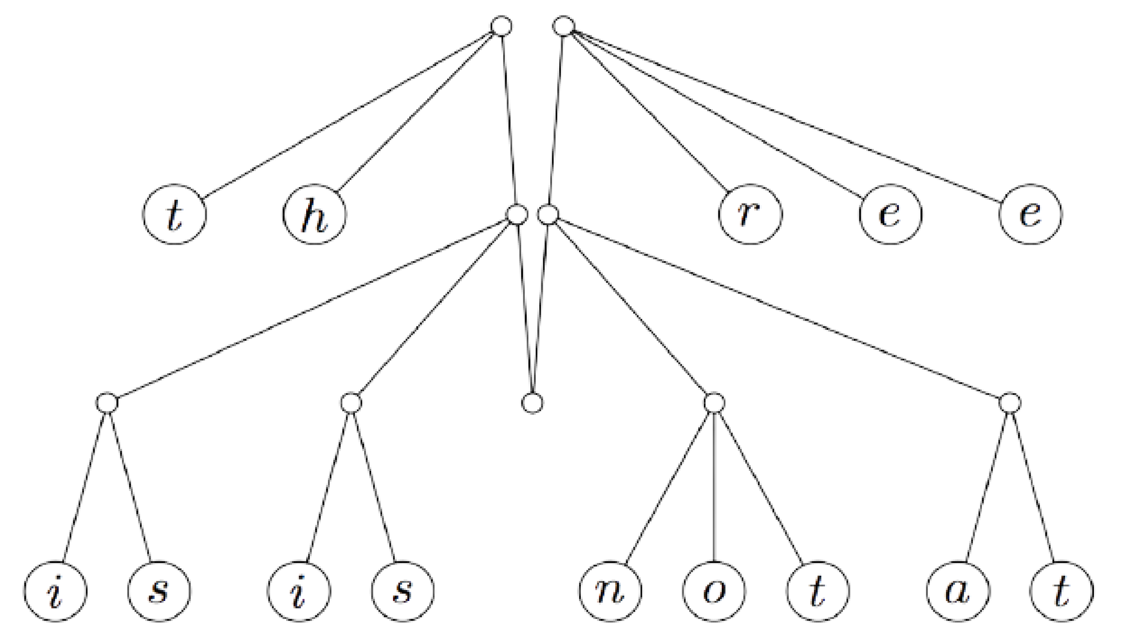
Producing a Verified Implementation of Sequences in Agda2Hs

CSE3000 Research Project
Shashank Anand – S.Anand-1@student.tudelft.nl
Jesper Cockx – J.G.H.Cockx@tudelft.nl, Lucas Escot – L.F.B.Escot@tudelft.nl
TU Delft

1. Sequences

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



Equivalent string of this finger tree is "thisisnotatree"

	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)

A comparison of Sequence and List operations

2. Testing vs Verification

Take two functions ***f*** and ***g***

To show f and g are equal, we show that

$$f(x) = g(x)\, \forall x$$

Using traditional testing methods, it is impossible to provide a guarantee that this is true, especially if the input is an infinite set. However using **dependently typed** languages such as **Agda**, we may **prove and verify** the property. The proof is validated by the type checker automatically.

Verification provides a stronger guarantee than testing.

3. Agda2Hs

Agda2Hs is a project that identifies a **common subset** of Agda and Haskell, and **translates** Agda code to Haskell code. We can produce **verified Haskell code** by verifying the Agda implementation and then translating it using Agda2Hs.

```
id : {a : Set} -> a -> a
id x = x
```

⬇

```
id :: a -> a
id x = x
```

Example : translating id function from Agda to Haskell

4. Research Question/ Objectives

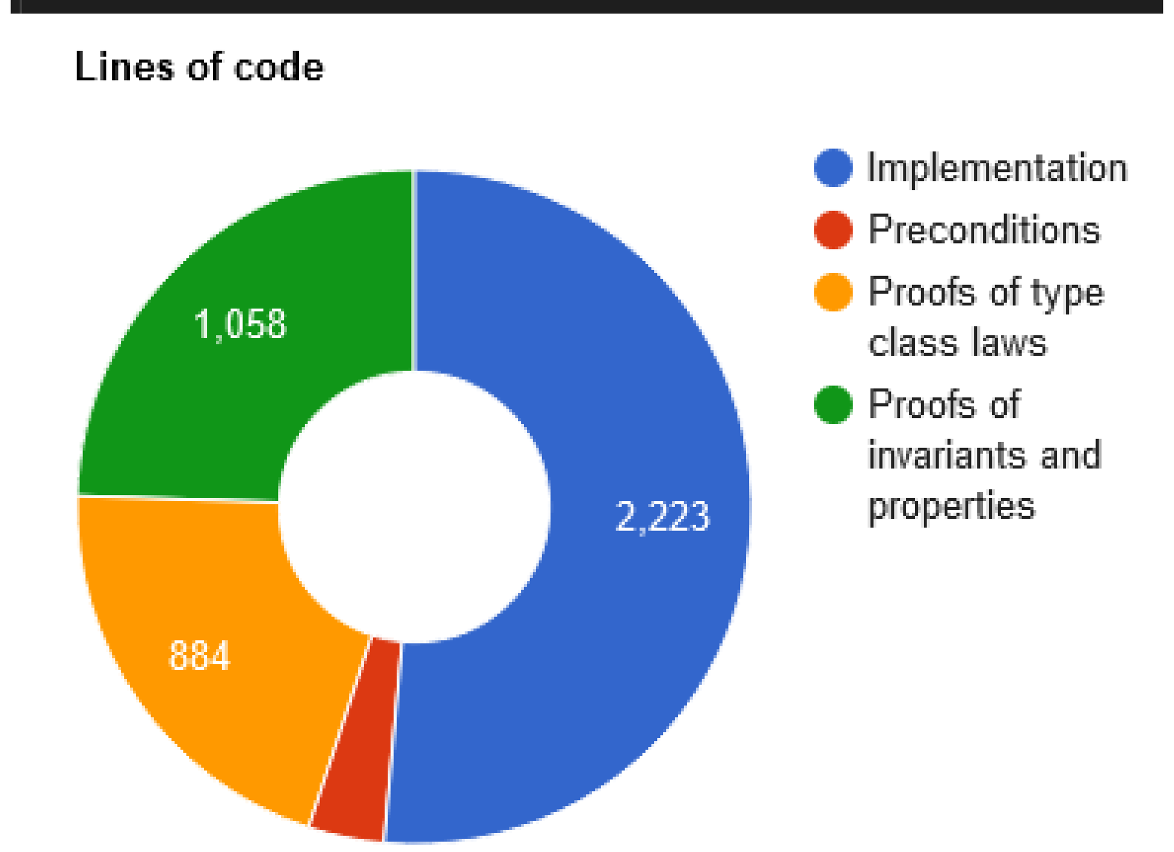
Can Agda2Hs be used to formally verify Haskell programs?

- Can agda2hs be used to implement the Sequence library?
- What are the invariants and properties guaranteed by the library?
- Is it possible to formally state the properties and prove the correctness of the library?

5. Implementation

As agda is a **total** language, partial functions in the library had to be made total using **preconditions. Non-empty** precondition restricts functions to take non-empty sequences only. In addition, the **termination checker** of agda must also be convinced. As **errors** cannot be thrown in a total language, a special error function provided by agda2hs must be used. This error function can only be used for **impossible cases.**

```
lookup : {a : Set} -> Int -> (ss : Seq a)
         -> {IsTrue (isNotEmptySequence ss)} -> Maybe a
```



6. Verification

- **Laws** of **type classes** implemented by sequence were proven.

```
functorSeqId : {a : Set} -> (xs : Seq a) -> fmap id xs ≡ id xs
```

- It is guaranteed that the size field (int) contains a valid size. We prove that functions that modify sequences return **valid sequences.**

```
sizeUnchangedToList : {a : Set} -> (xs : Seq a) -> {IsTrue (isValidSeq xs)}
                      -> size xs ≡ lengthList (toList xs)
```

7. Optimizations

- **Postulate** trivial properties and properties of imported types
- Prove the **innermost properties** first. Combine smaller properties to simplify proofs.
- Always **case-split** to prevent ambiguity/overlapping patterns
- Try to use the auto feature of Agda.

8. Conclusion

- The Data.Sequence library was **implemented** in Agda in the common subset of Agda and Haskell identified by Agda2Hs. **No extensions** to Agda2Hs were required.
- Invariants were identified and some properties were **proven successfully.** The technique identified may be applied to **extend** the set of verified functions.