Verified Data Structures and Algorithms in Agda
Progress Report

Razvan Kusztos
Supervisor: Dr. Timothy Griffin

University of Cambridge

Part II

February 12, 2017

Motivation

- Dependent typing enforces correctness of programs.
- Nested types maintain invariants of data structure.
- Together they present a challenge both for programming and verifying correctness
- The dissertation's purpose was:
-     Implementing the Finger Tree in agda
-     Proving the correctness of its methods.

Regular vs Nested Types

List is an example of a **regular** data type.

One can also declare **irregular (nested)** data types [0], where the recursive call to the type constructor takes as argument a different type (examples to follow)

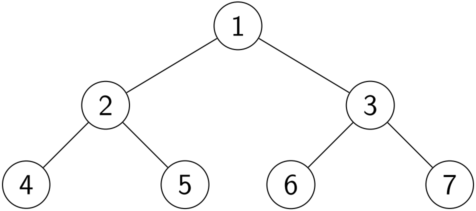
In older versions of Haskell and ML, although the declarations are valid, one couldn't write functions to operate on them. [0] (Hindley-Milner typing).

They turn out to be very useful in practice, as they can keep strong invariants on the data.

Example 1: Full binary tree

$$\textbf{data}\ \mathit{Nest}\ (A : \mathit{Set}) : \mathit{Set}\ \textbf{where}$$
$$\quad \mathit{Nil} : \mathit{Nest}\ A$$
$$\quad \mathit{Cons} : A \to \mathit{Nest}\ (A \times A) \to \mathit{Nest}\ A$$

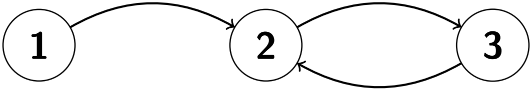
$$ex : \mathit{Nest}\ \mathbb{N}$$
$$ex = \mathit{Cons}\ 1\ (\mathit{Cons}\ (2, 3)\ (\mathit{Cons}\ ((4, 5), (6, 7))\ \mathit{Nil}))$$



Example 2: Cyclic list [5]

data *Clist* $(A : Set) : Set$ **where**
$\quad Var : A \rightarrow Clist\ A$
$\quad Nil : Clist\ A$
$\quad RCons : \mathbb{N} \rightarrow Clist\ (Maybe\ A) \rightarrow Clist\ A$

$ex : CList\ ()$
$clist2 = RCons\ 1\ (RCons2\ (RCons\ 3\ (Var\ (just\ nothing))))$



Example 3: Finger Trees

$$
\begin{array}{l}
\textbf{data}\ \mathit{FingerTree}\ \{a\}\ (A : \mathit{Set}\ a) : \mathit{Set}\ a\ \textbf{where} \\
\quad \mathit{Empty} : \mathit{FingerTree}\ A \\
\quad \mathit{Single} : A \rightarrow \mathit{FingerTree}\ A \\
\quad \mathit{Deep}\ \ : \mathit{Digit}\ A \rightarrow \mathit{FingerTree}\ (\mathit{Node}\ A) \rightarrow \mathit{Digit}\ A \rightarrow \\
\qquad\qquad \mathit{FingerTree}\ A
\end{array}
$$

Motivation

The original paper suggests implementing Views [6] in order to make implementations more straight-forward, while keeping the same complexity.

$$\begin{array}{l} \textbf{data}\ \mathit{ViewL}\ \{a\}\ (A : \mathit{Set}\ a) : \mathit{Set}\ a\ \textbf{where} \\ \quad \mathit{nilL} : \mathit{ViewL}\ A \\ \quad \mathit{consL} : A \to \mathit{FingerTree}\ A \to \mathit{ViewL}\ A \end{array}$$

And a function to transform between the views.

$$\begin{array}{l} \mathit{viewL} : \forall \{a\}\ \{A : \mathit{Set}\ a\} \to \mathit{FingerTree}\ A \to \mathit{ViewL}\ A \\ \mathit{viewL}\ \mathit{ft} = \{!!\} \end{array}$$

Motivation

However, implementing functions stops working because of termination checker.

$$
\begin{aligned}
&\mathit{reverse} : \forall \{a\}\ \{A : \mathit{Set}\ a\} \rightarrow \mathit{FingerTree}\ A \rightarrow \mathit{FingerTree}\ A \\
&\mathit{reverse}\ \mathit{ft}\ \mathit{with}\ \mathit{viewL}\ \mathit{ft} \\
&\mathit{reverse}\ \mathit{ft} \mid \mathit{nilL} = \mathit{ft} \\
&\mathit{reverse}\ \mathit{ft} \mid \mathit{consL}\ x\ \mathit{xs} = (\mathit{reverse}\ \mathit{xs}) \triangleright x
\end{aligned}
$$

Progress Made - Implementation

- Implemented the FingerTree in 3 iterations with operations like reduce, view, append etc.
- 1. The plain, non-dependent one (the declaration of which was above)
- 2. A modification that includes the concept of a measurement [3]
- 3. A dependent version of the second one.

Progress Made - Verification

- The dependent implementation contains internal verification of the axioms measurement should entail
- Wrote proofs of correctness for the cons operator.
- Specialized the data structure through measurements (e.g. Random Access Sequence).
- Implemented a 4th version of the FingerTree, following an Isabelle implementation [4], that has a very different approach

Progress Made - Overcoming Difficulties

- Using the principle of WF induction, I could write an implementation of the reverse function.
- I have presented an argument that Sized types cannot be used in this context.

Further work

- Produce more correctness proofs:
- Evaluate by seeing whether the dependent version hinders amortized performance (which is amortized O(1) for most operations [2])
- Explore the performance difference between the original and the isabelle implementation.
- Provide a more convincing argument that Sizes cannot be employed here.
- Attempt to generalize for other nested data structures [1].

References

Richard Bird and Lambert Meertens
Nested Datatypes

Ralf Hinze
Numerical Representations as Higher Order Nested Datatypes.

Chris Okasaki (1996)
Purely Functional Data Structures

Ralf Hinze and Ross Paterson (2006)
Finger Trees: A Simple General-purpose Data Structure
Journal of Functional Programming 16(2):197–217.

Benedikt Nordho, Stefan Korner, Peter Lammich (2014)
FingerTrees

Neil Ghani, Makoto Hamana, Tarmo Uustalu, Varmo Vene (2006)
Representing Cyclic Structures as Nested Datatypes

Phil Wadler (1986)
Views: A way for pattern matching to cohabit with data abstraction

Thank you!