Martin Hofmann's case for non-strictly positive data types

Ralph Matthes

CNRS, Institut de Recherche en Informatique de Toulouse (IRIT),
University of Toulouse

Types Conference 2018, University of Minho,
Braga, Portugal

June 20, 2018

with a postscript added on June 26

Abstract

This talk is part of the special session of TYPES 2018 to honour Martin Hofmann whose untimely death in January is a great loss to the TYPES community.

In a nutshell, I'll describe the breadth-first traversal algorithm by Martin Hofmann, how it can be verified, what is needed to do a verification in an intensional setting (system F without parametric equality) and what else could be programmed in this spirit.
Time permitting, I'll allow myself some remarks on Martin Hofmann, as I have perceived him (as assistant in his research group).

IRIT Institut de Recherche en Informatique de Toulouse

ACADIE

Outline

Outline

1. Obtaining fancy breadth-first traversal
2. Analyzing fancy breadth-first traversal
3. Other non-strictly positive datatypes in use

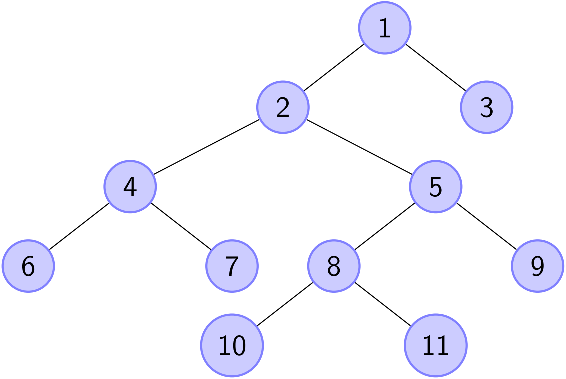
Breadth-first traversal

Binary trees with leaf labels and node labels in $\mathbb{N}$. Call this data type Tree, with constructors $\mathit{Leaf} : \mathbb{N} \to \mathit{Tree}$ and $\mathit{Node} : \mathit{Tree} \to \mathbb{N} \to \mathit{Tree} \to \mathit{Tree}$.

The type of homogeneous lists with elements from type A is called $\mathit{List}\,A$.

Go through $t : \mathit{Tree}$ in breadth-first order and collect the labels in $\mathit{breadthfirst}\,t : \mathit{List}\,\mathbb{N}$.

Illustration: result $[1, 2, \ldots, 11]$



The function is not compositional!

Less intuitive specification

Create the list of labels for every line. This function is called $niveaux : Tree \to List^2\mathbb{N}$.
Result for our example: $[[1], [2, 3], [4, 5], [6, 7, 8, 9], [10, 11]]$
$niveaux$ can be obtained by iteration over $Tree$: in the $Node$ case, zip the results for the sub-trees with list concatenation, vulgo $append$.
Define $flatten : List^2\mathbb{N} \to List\,\mathbb{N}$ as concatenation of all those lists (the monad multiplication of the list monad).
$breadthfirst\,t$ has to evaluate to the result of $flatten(niveaux\,t)$. The latter is not the algorithm but an executable specification.

This is for functional programmers. Imperative programming would suggest to use a queue of binary trees. We type theoreticians want language-based termination guarantees.

Martin Hofmann's 1993 proposal

A post to the TYPES mailing list, which is still in the TYPES archives.
Assumes a data type Cor with constructors

$$\begin{array}{lcl} Over & : & Cor \\ Next & : & ((Cor \to List\,\mathbb{N}) \to List\,\mathbb{N}) \to Cor \end{array}$$

Martin viewed the elements as continuations, but I learned from Olivier Danvy in 2002 that they are rather coroutines, hence the name Cor chosen here ($Over$ and $Next$ suggested by Danvy). $Over$: nothing more to be done; $Next$: its argument f takes a "continuation" argument $k : Cor \to List\,\mathbb{N}$ and computes a list.

Working with coroutines

Specify $apply : Cor \to (Cor \to List\,\mathbb{N}) \to List\,\mathbb{N}$ by distinguishing the two cases:

$$\begin{array}{lcl} apply\ Over\ k & \simeq & k\ Over \\ apply\ (Next\ f)\ k & \simeq & f\,k \end{array}$$

Relation $\simeq$ is used for definitional equality, i. e., convertibility. It does not by itself constitute a definition.

Martin Hofmann recasts the breadth-first traversal as a transformation on coroutines controlled by the input tree:

$breadth : Tree \to Cor \to Cor$, with

$$\begin{array}{lcl} breadth(Leaf\ n) & \simeq & \lambda c^{Cor}.\, Next(\lambda k.n :: apply\, c\, k) \\ breadth(Node\, l\, n\, r) & \simeq & \lambda c^{Cor}.\, Next\Big(\lambda k.n :: apply\, c \\ & & \quad (\lambda c_1^{Cor}.\, k(breadth\, l\, (breadth\, r\, c_1)))\Big) \end{array}$$

Extraction of the final result

$breadth\, t\, Over$ is a coroutine, and we want $breadthfirst\, t$ to be the list extracted from it by the function $ex : Cor \to List\, \mathbb{N}$, specified as

$$\begin{array}{lcl} ex\; Over & \simeq & [] \\ ex(Next\, f) & \simeq & f\, ex \end{array}$$

No problem with subject reduction—recall $f : (Cor \to List\, \mathbb{N}) \to List\, \mathbb{N}$. In our view, ex is a “continuation”, and so the argument f to $Next$ can be naturally applied to it.

Why is this recursion scheme safe, i. e., why does it not present the risk of non-termination?

Outline

Termination of ex

For Martin Hofmann, this was the main motivation. One can see Cor as least fixed point of the "functor" $CorF$ that a Haskell programmer could define as

```
data CorF cor = Over | Next ((cor -> List nat) -> List nat)
```

The variable `cor` for the datatype to be defined is twice to the left of `->`, hence at a positive position, even if not at a strictly positive position.

Martin argues that the specification of ex can be ensured by the usual Church encoding of data types in system F; in categorical terms, ex can be obtained as catamorphism for a certain $CorF$-algebra. However, only weak initiality is obtained, unless one uses parametric equality. In more computational terms, this means that ex is defined by pure iteration.

Pitfall concerning termination

The argument on ex is valid, even if Mendler-style iteration would more directly allow to program ex precisely according to the specification, and likewise with termination guarantee (as instance of Mendler-style iteration).
However, the function $apply : Cor \to (Cor \to List\,\mathbb{N}) \to List\,\mathbb{N}$ has to be defined with the same ontology for Cor. To recall:

$$\begin{array}{lcl} apply\; Over\; k & \simeq & k\; Over \\ apply\; (Next\; f)\; k & \simeq & f\, k \end{array}$$

No recursion but the patterns are distinguished. This is not compatible with weakly initial algebras, as obtained with the Church encoding. Martin was satisfied with parametric equality theory, but $apply$ should have constant execution time, if the proposed algorithm should have advantages over the executable specification.

Solution

Use primitive recursion in Mendler's style. In fact, the only addition to system F that is really needed to preserve termination is positive fixed-points μF with a retraction between μF and $F(\mu F)$—the sequence from $F(\mu F)$ via μF back to $F(\mu F)$ has to be pointwise definitionally equal to the identity. Termination of more complex schemes can be obtained by simulation of reductions.

functional correctness

Given that the termination question is already solved, how can one see that the algorithm $\mathit{breadthfirst}\,t := \mathit{ex}(\mathit{breadth}\,t\;\mathit{Over})$ meets the specification, i. e., computes the right list?
In 1995, Ulrich Berger (then in Munich) gave an exciting proof that uses a non-strictly positive inductive predicate when coroutines "represent" lists of lists.
Martin Hofmann provided in 1995 a proof by simple induction on Tree, by help of a function $\gamma : \mathit{List}^2\mathbb{N} \to \mathit{Cor} \to \mathit{Cor}$ for which (A) $\mathit{ex}(\gamma\,l\;\mathit{Over}) = \mathit{flatten}\,l$, (B) composition of γ for two lists of lists is γ for the zipping with append, which allows to prove that (C) $\mathit{breadth}\,t$ does the same as $\gamma(\mathit{niveaux}\,t)$. (A) and (C) then give correctness.
Also in 1995, Anton Setzer (then in Munich) showed in two steps that only specific forms of the coroutines ever appear in the algorithm.

Setzer-style verification by successive refinements

First step: $\mathit{breadth} : \mathit{Tree} \to \mathit{Cor} \to \mathit{Cor}$ can be shrunk down to a structurally recursive definition of a function $\mathit{breadth}_p : \mathit{Tree} \to \mathit{Cor}' \to \mathit{Cor}'$ with $\mathit{Cor}' := \mathit{List}(\mathit{List}\,\mathbb{N} \to \mathit{List}\,\mathbb{N})$. The crucial definition is a function $\phi : \mathit{Cor}' \to \mathit{Cor}$, for which one tries to obtain

$$\mathit{breadth}\, t\, (\phi\, l) = \phi(\mathit{breadth}_p\, t\, l)$$

This guides the definition process for $\mathit{breadth}_p$.

Second step: $\mathit{breadth}_p : \mathit{Tree} \to \mathit{Cor}' \to \mathit{Cor}'$ can be shrunk down to a structurally recursive definition of a function $\mathit{breadth}'_p : \mathit{Tree} \to \mathit{List}^2\mathbb{N} \to \mathit{List}^2\mathbb{N}$. The crucial definition is a function $\psi : \mathit{List}\,\mathbb{N} \to (\mathit{List}\,\mathbb{N} \to \mathit{List}\,\mathbb{N})$, so that for its mapping over lists, $\tilde{\psi} : \mathit{List}^2\mathbb{N} \to \mathit{Cor}'$, one can obtain

$$\mathit{breadth}_p\, t\, (\tilde{\psi}\, l) = \tilde{\psi}(\mathit{breadth}'_p\, t\, l)$$

Setzer-style verification—the end

The function $breadth'_p : Tree \to List^2\mathbb{N} \to List^2\mathbb{N}$ thus obtained is easy to grasp in terms of list operations:

$$breadth'_p\, t\, l = append\,(niveaux\, t)\, l$$

And $ex(\phi(\tilde{\psi}\, l)) = flatten\, l$.

Modulo the exciting presentation in terms of the non-strictly positive data type of coroutines, the outcome of the analysis was that $breadth$ adopted an "accumulation trick" for computing the levels, and that the extraction process took care of flattening.

Outline

Institut de Recherche en Informatique de Toulouse

non-strictly positive to understand classical logic

For a given type A, the type $\sharp A := \mu X.A + \neg\neg X$ is "a bit bigger" than $\neg\neg A$: the second constructor ensures

$$\neg\neg\sharp A \to \sharp A$$

but also $\neg\neg A$ has double negation elimination, however, $\sharp A$ is freely constructed with this property—called the "stabilization" of A. Being "bigger" (as target of an embedding) is better since several proofs of strong normalization of variants of $\lambda\mu$-calculus suffered from erasure problems. Second-order $\lambda\mu$-calculus can be simulated inside system F with these types $\sharp A$ and their iteration principle, see my TLCA'01 paper and subsequent work.

Another case for Setzer-style verification?

In 2002, Danvy communicated to me a coroutine solution to the same fringe problem.

Example

They have the same fringe. For this problem, inner nodes are unlabeled.

Same fringe with coroutines

Let Cor now have the constructors $Over : Cor$ and $Next : \mathbb{N} \to ((Cor \to \mathbb{B}) \to \mathbb{B}) \to Cor$. Variable convention: $k : Cor \to \mathbb{B}$ "continuations", and $f : (Cor \to \mathbb{B}) \to \mathbb{B}$.

The critical function that needs elimination principles for Cor is $skim : Cor \to Cor \to \mathbb{B}$ with $skim\ Over\ Over \simeq \mathsf{t}$, result f for two arguments with different constructor and

$$skim(Next\, n_1\ f_1)(Next\, n_2\ f_2) \simeq if\, n_1 \neq n_2\, then\, \mathsf{f}\, else\, f_1(\lambda c^{Cor}.f_2(skim\ c))$$

This is an instance of Mendler-style iteration, but needs the same addition we needed before for $apply$. It also nicely type-checks with sized types, as developed in the PhD thesis of Andreas Abel.

the same fringe program

Define $walk : Tree \to (Cor \to \mathbb{B}) \to ((Cor \to \mathbb{B}) \to \mathbb{B}) \to \mathbb{B}$ by

$$\begin{array}{lcl} walk\,(Leaf\ n)\,k\,f & \simeq & k(Next\,n\,f) \\ walk\,(Node\,l\,r)\,k\,f & \simeq & walk\,l\,k\,(\lambda k_1.\,walk\,r\,k_1\,f) \end{array}$$

Define $canf := \lambda k.k\,Over$ and $init : Tree \to (Cor \to \mathbb{B}) \to \mathbb{B}$ by $init\,t\,k := walk\,t\,k\,canf$. Finally, $smf : Tree \to Tree \to \mathbb{B}$ is defined by

$$smf\,t_1\,t_2 := init\,t_1\,(\lambda c_1.\,init\,t_2(skim\,c_1))$$

Is there a Setzer-style verification to demystify these operations?

Postscript June 26, 2018 – Conclusions come afterwards!

In a message to the Coq club—`https://sympa.inria.fr/sympa/arc/coq-club/2018-06/msg00096.html`—on the day following this talk, Simon Boulier, who attended the conference, announced the availability of a Coq plugin to deactivate the checks for strict positivity. He had already tested it with Martin Hofmann's program on the day of this talk.

Using his plugin, I formalized the functional correctness in the three styles described or mentioned in this talk. This is available as case study for the plugin at `https://github.com/SimonBoulier/TypingFlags/blob/master/theories/BreadthFirst.v`, as part of Boulier's GitHub repository.

Conclusion

From the published abstract:
And this talk should remind the audience how much Martin's scientific insights were able to fascinate other researchers, even if they were not considered as ready to be published by Martin. Sadly, we have to live with these memories without further opportunities to get new notes from Martin or to work with him. May he rest in peace.