Programming with nested datatypes through dependent types

Ralph Matthes

Institut de Recherche en Informatique de Toulouse (IRIT), CNRS
Équipe ACADIE
(Assistance à la Certification d'Applications Distribuées et Embarquées)

Workshop Dependently Typed Programming 2008
NCSL, Jubilee Campus
University of Nottingham, U. K.
February 19, 2008
with small changes on pages 2, 7, 14 (February 21)

The Tentative Original Abstract

Nested datatypes are families of datatypes that are indexed over all (small) types. They can be represented in intensional type theory, and terminating recursion schemes can be developed in type theory, with laws of program verification in the same system. Following suggestions by several colleagues, I would like to take dependent types more seriously already for programs that work on nested datatypes. This can be done by indexing the nested datatypes additionally over the natural numbers as with sized nested datatypes (Andreas Abel's PhD thesis) where the size corresponds to the number of iterations of the datatype "functor" over the constantly empty family.

Abstract, Cont'd

But one can also try to define functions directly for all powers of the nested datatype (suggested to me by Nils Anders Danielsson) or even define all powers of it simultaneously (suggested by an anonymous referee of a paper).
The author has presented preliminary results at the Seminar on Dependently Typed Programming at Dagstuhl in 2004 and at the TYPES 2004 meeting about yet another approach where the indices are finite trees that branch according to the different arguments that appear in the recursive equation for the nested datatype (based on ideas by Anton Setzer and Peter Aczel). The talk will try to shed a light on all these possibilities, by way of examples that have been carried out in the Coq theorem prover.

Bushes (Bird & Meertens 1998)

A family of types $Bush\,A$ for any (small) type A, with recursive equation

$$Bush\,A = 1 + A \times Bush(Bush\,A)$$

The datatype constructors are

$$\begin{aligned} bnil &: \forall A.\, Bush\,A \\ bcons &: \forall A.\, A \to Bush(Bush\,A) \to Bush\,A \end{aligned}$$

A truly nested datatype: An inductive family which has at least one datatype constructor for which one of the argument types has a nested call to the family name, i. e., the family name appears somewhere inside the type argument of the family name occurrence in the argument type of that datatype constructor.

Bushes into Lists

$$BushF := \lambda X \lambda A.\, 1 + A \times X(X\,A)$$

Mendler-style iteration (Mendler 1987) can be extended to nested datatypes (joint work with Andreas Abel and Tarmo Uustalu, 2005).
We use the abbreviation $X \subseteq G := \forall A.\, XA \to GA$.

Define a function $BtL : Bush \subseteq List$ by

$$\begin{aligned} BtL := MIt\ List\ \big(\lambda X \lambda it^{X \subseteq List} \lambda A \lambda t^{BushF\ X\ A}.\, \mathrm{match}\ t\ \mathrm{with}\ inl\ \_ \mapsto [] \\ \mid inr(a^A, b^{X(X\,A)}) \mapsto a :: flat\_map\,(X\,A)\,A\,(it\,A)(it\,(XA)\,b)\big) \end{aligned}$$

$$\begin{aligned} BtL\,A\,(bnil\,A) &\longrightarrow^+ [] \\ BtL\,A\,(bcons\,A\,a\,b) &\longrightarrow^+ a :: flat\_map\,(BtL\,A)(BtL\,(Bush\,A)\,b) \end{aligned}$$

ACADIE

Sized Nested Datatypes (Andreas Abel)

Bushes with height bounded by k:

$$\begin{array}{lcl} \mathit{bnilS} & : & \forall k \forall A.\, \mathit{Bush}_{Sk}\, A \\ \mathit{bconsS} & : & \forall k \forall A.\, A \to \mathit{Bush}_k(\mathit{Bush}_k\, A) \to \mathit{Bush}_{Sk}\, A \end{array}$$

The definition looks like a truly nested datatype but is only a family of types that can be explicitly defined by recursion on natural numbers k (with base case $\mathit{Bush}_0\, A := \bot$).

A striking benefit is precise typing, such as for the map term

$$\mathit{bushS} : \forall k \forall A \forall B.\, (A \to B) \to \mathit{Bush}_k\, A \to \mathit{Bush}_k\, B,$$

to be defined by recursion on k.

Andreas' original system has subtyping in order to pass from $\mathit{Bush}_k\, A$ to $\mathit{Bush}_{Sk}\, A$. We can do this by a function, defined by recursion over k and using bushS.

IRIT ACADIE

Bushes with Size into Lists

By recursion on k, one can define

$$BStL : \forall k \forall A.\, Bush_k\, A \longrightarrow List\, A$$

such that

$$\begin{array}{lcl} BStL_{Sk}\, A\, (bnilS\, k\, A) & = & [] \\ BStL_{Sk}\, A\, (bconsS\, k\, a\, b) & = & a :: flat\_map(BStL_k\, A)(BStL_k\, (Bush_k\, A)\, b) \end{array}$$

It is an exercise to prove naturality of $BStL_k$ as transformation from $(Bush_k, bushS\, k)$ to $(List, map)$.

Bush from $Bush_k$?

Define

$$Bush\, A := \{k : nat \,\&\, Bush_k\, A\}$$

with the Coq notation for a strong Σ-type where the second component is also computationally relevant.
$bnil : \forall A.\, Bush\, A$ can be defined by taking 1 for k and using $bnilS\, 0\, A$.

How to obtain $bcons : \forall A.\, A \to Bush(Bush\, A) \to Bush\, A$?
Assume $a : A$ and $b : Bush(Bush\, A)$. Thus, there is a k and $b_0 : Bush_k(Bush\, A)$. In order to apply $bconsS$, we need an element b' of $Bush_{k'}(Bush_{k'}\, A)$ for some k'.
We may go through the list $BStL\, k\, b_0$ and dermine all the indices and finally set k' to the maximum of all those indices and k. Then, we have to transform b_0 into the desired b'.
A complicated and non-generic procedure for a constructor!

More Perspicuous Behaviour Through All Powers of *Bush*

This approach follows a suggestion by Nils Anders Danielsson. We are in an easier situation since we already have $BtL : Bush \subseteq List$. Define $BtL_k : Bush^k \subseteq List$ by recursion on k such that

$$\begin{array}{lcl} BtL_0\, A\, a & = & [a] \\ BtL_{Sk}\, A\, t & = & flat\_map\,(BtL_k\, A)\,(BtL\,(Bush^k\, A)\, t) \end{array}$$

Using properties of *flat_map*, one easily obtains

$$\begin{array}{lcl} BtL_{Sk}\, A\,(bnil\,(Bush^k\, A)) & = & [] \\ BtL_{Sk}\, A\,(bcons\, a\, b) & = & BtL_k\, A\, a + BtL_{S(Sk)}\, A\, b \end{array}$$

Here, $+$ stands for list concatenation (append).

Starting with the final equations as a definition constitutes the proposal "by Chalmers".

How This Looks Like for Non-canonical Terms

My system *LNMIt* allows to prove naturality of the original definition of *BtL*. This works by passing to bigger types $Bush\,A$. Their datatype constructors are

$$bnil\_ : \forall A \forall X \forall ef^{\mathcal{E}\,X} \forall j^{X \subseteq Bush}.\, j \in \mathcal{N}(m\,ef,\, bush) \to BushA$$

and

$$bcons\_ : \forall A \forall X \forall ef^{\mathcal{E}\,X} \forall j^{X \subseteq Bush}.\, j \in \mathcal{N}(m\,ef,\, bush) \to A \to X(XA) \to BushA$$

Here, $\mathcal{E}\,X$ states that X has a map term that satisfied the functor laws and only depends on the extension of its function argument. The only requirement in *LNMIt* for a datatype "functor" F is that it preserves extensional functors in that sense. No monotonicity of F in its type transformer argument X is needed!

ACADIE

BtL_k for Non-canonical Bushes

Also for non-canonical empty bushes, the result is the empty list. The interesting case is

$$\forall k \forall A \forall X \forall ef^{\mathcal{E}\, X} \forall j^{X \subseteq Bush} \forall n^{j \in \mathcal{N}(m\, ef,\, bush)} \forall a^{Bush^k\, A} \forall b^{X(X(Bush^k\, A))}.$$
$$BtL_{Sk}\, A\, (bcons\_\, ef\, j\, n\, a\, b) =$$
$$BtL_k\, A\, a + BtL_{S(Sk)}\, A\, (bush\, (j_{Bush^k\, A})(j_{X(Bush^k\, A)}\, b))$$

It is proven by naturality of BtL.

Who knows how to argue about termination of such a recursive equation (if it were used to define BtL_k)?

Introducing All Powers Simultaneously

One might also directly give an inductive definition of all powers of *Bush*:

$$\begin{array}{lcl} \mathit{bnilP} & : & \forall k \forall A.\, \mathit{Bush}^{Sk}\ A \\ \mathit{bconsP} & : & \forall k \forall A.\, \mathit{Bush}^{k}\ A \to \mathit{Bush}^{S(Sk)}\ A \to \mathit{Bush}^{Sk}\ A \\ \mathit{bstartP} & : & \forall A.\, A \to \mathit{Bush}^{0}\ A \end{array}$$

In Coq with impredicative Set, this is supported directly (after changing k and A as parameters). The "Chalmers definition" of *BtL* is then nearly dictated by this representation.
We want to get back $\mathit{Bush}\,A$ as $\mathit{Bush}^1\,A$. Clearly, *bnil* can be defined from $\mathit{bnilP}\,0$, but for *bcons*, we need $\mathit{Bush}^1(\mathit{Bush}^1\,A)$ instead of $\mathit{Bush}^2\,A$ that is the second argument to $\mathit{bconsP}\,0$.
The solution: Define "Aczel application" of type $\forall k_1 \forall k_2 \forall A.\, \mathit{Bush}^{k_1}(\mathit{Bush}^{k_2}\,A) \to \mathit{Bush}^{k_1+k_2}\,A$.

Background For This Approach

This all is just an instance of the more general observation by Anton Setzer (March 2003) made for lambda terms with explicit flattening: One should index the nested datatype by a finite tree that indicates how A has been transformed in order to get the type argument to the nested datatype. For bushes, there is only one operation on these type arguments A, namely taking $Bush\,A$ in order to have the type argument in $Bush\,(Bush\,A)$. Therefore, natural numbers suffice to trace the transformation.
For lambda terms with explicit flattening, a mathematical development has been done by Peter Aczel, and algorithms programmed for the Dagstuhl seminar on Dependently Typed Programming in September 2004 by myself.

How can these loose ends be put together?

- System *LNMIt* has the power of Mendler iteration, is generic in the nested datatype, can prove naturality of iteratively defined functions, but bothers the user with non-canonical elements. (One can relativize to hereditarily canonical elements, but this is not fully generic.)
- Sized nested datatypes (in the form presented here without the limit $Bush_\infty$) have a simple ontology, give precise typings for map terms but do not easily allow to translate the results back into the original formulation without sizes. It is not clear what should be verified about that "absolute formulation", since there does not seem to be a useful induction principle (unless one reintroduces non-canonical terms).
- Going straight to powers can clarify certain relations but is technically more demanding and also suffers from verification problems (and needs properties that capture all powers).

Thank you for listening.