Primitive Recursion for Rank-2 Inductive Types

Andreas Abel* and Ralph Matthes**

Department of Computer Science
University of Munich

Recently, higher-rank datatypes have drawn interest in the functional programming community [Oka99,Oka96,Hin01]. Rank-2 non-regular types, so-called *nested datatypes*, have been investigated in the context of Haskell. To define total functions which traverse nested datastructures, Bird et al. [BP99] have developed *generalized folds* which implement an iteration scheme and are strong enough to encode most of the known algorithms for nested datatypes. In this note, we investigate a scheme to overcome some limitations of iteration which we expound in the following.

Since the work of Böhm *et al.* [BB85] it is well-known that iteration for rank-1 datatypes can be simulated in typed lambda-calculi. The easiest examples are iterative definitions of addition and multiplication for Church numerals. The iterative definition of the predecessor, however, is inefficient: It traverses the whole numeral in order to remove one constructor. Surely, taking the predecessor should run in constant time.

Primitive recursion is the combination of iteration and efficient predecessor. A typical example for a prim. rec. algorithm is the natural definition of the factorial function. It is common belief that prim. rec. cannot be reduced to iteration in a computationally faithful manner. This is because no encoding of natural numbers in the polymorphic lambda-calculus (System F) seems possible which supports a constant-time predecessor operation (see Spławski and Urzyczyn [SU99]). Mendler extended System F by a scheme of prim. rec. for rank-1 datatypes and proved strong normalization [Men87]. Mendler's formulation does not follow the usual category-theoretic approach with initial recursive algebras (see Geuvers [Geu92]).

For rank-2 datatypes there are also examples of functions which can most naturally be implemented with prim. rec. One is *redecoration for triangular matrices* which is presented below. These examples are not instances of generalized folds à la Bird *et al.*, which remain within the realm of iteration but hardwire Kan extensions into the recursion scheme. Rank-2 prim. rec., which we propose in this work, seeks to extend rank-2 iteration in the same way that prim. rec. extends rank-1 iteration. We achieve this by lifting Mendler's scheme of prim. rec. to rank 2. The decision for Mendler-style and against the traditional way roots in the following observation: Experiments with formulations according to the traditional style showed unnecessary but unavoidable traversals of the whole data structures in our examples. Mendler's style, however, yielded precisely the

* The first author is supported by the *Graduiertenkolleg "Logik in der Informatik"* of the *Deutsche Forschungsgemeinschaft.*

** Both authors acknowledge financial support by ETAPS 2003.

desired efficient reduction behavior. This was crucial since the only reason to incorporate prim. rec. is operational efficiency as opposed to denotational expressiveness.

We work within the framework System F^ω of higher-order parametric polymorphism formulated in Curry-style, i.e., as a type assignment system for the pure lambda-calculus. For type transformers $X, Y : * \to *$ we abbreviate the type of natural transformations $\forall A.\, X A \to Y A$ from X to Y by $X \subseteq Y$. Let $\mathsf{id} = \lambda x.x$ denote the identity function.

We extend the framework by a new constructor constant μ and two term constants in and MRec and a new reduction rule as follows.

Formation.	$\mu \quad : ((* \to *) \to * \to *) \to * \to *$
Introduction.	$\mathsf{in} \quad : \forall F^{(* \to *) \to * \to *}.\, F\,(\mu F) \subseteq \mu F$
Elimination.	$\mathsf{MRec} : \forall F^{(* \to *) \to * \to *} \forall G^{* \to *}.\, (\forall X^{* \to *}.\, X \subseteq \mu F \to X \subseteq G \to F\,X \subseteq G) \to \mu F \subseteq G$
Reduction.	$\mathsf{MRec}\, s\, (\mathsf{in}\, t) \longrightarrow_\beta s\ \mathsf{id}\, (\mathsf{MRec}\, s)\, t$

The type transfomer $\mu F : * \to *$ is the least fixed-point of the constructor $F : (* \to *) \to * \to *$ and denotes a simultaneously defined family of types of well-founded trees, their shape depending on F. For instance, using $F = \lambda X \lambda A.\, 1 + A \times X\, A$ the well-known type of polymorphic lists is recovered. The term in is the general constructor, which, in case of lists, codes together nil and cons. The term MRec establishes a scheme of primitive recursion in the style of Mendler. Typical for this style is the universally quantified constructor variable X in the type of the step term s which ensures termination without any positivity restrictions on F. During reduction, X is instantiated by μF, and the first parameter, $i : X \subseteq \mu F$, by id. The presence of a transformation i from the blank type X back into the fixed-point μF is what distinguishes Mendler-style prim. rec. from Mendler-style iteration.

$$\begin{array}{cccccc} A & E & E & E & \dots & E \\ & A & E & E & \dots & E \\ & & A & E & \dots & E \\ & & & A & \dots & E \\ & & & & \ddots & E \\ & & & & & A \end{array}$$

An example of a non-regular datatype is $\mathsf{Tri}\, A = (\mu\, \mathsf{TriF})\, A$ with $\mathsf{TriF} = \lambda X \lambda A. A \times (1 + X (E \times A))$, the type of triangular matrices over a given entry type E but with type A on the diagonal. For these matrices, we define a redecoration operation

$$\mathsf{redec} : \forall A \forall B.\ \mathsf{Tri}\, A \to (\mathsf{Tri}\, A \to B) \to \mathsf{Tri}\, B.$$

The call $\mathsf{redec}\ t\, f$ replaces each diagonal element a of t with the result of applying f to the sub-triangle whose upper-left corner is a. Redecoration is a natural example for primitive recursion and is no instance of a generalized fold.

System F^ω, extended by Mendler-style primitive recursion, is still confluent and strongly normalizing. A dual construction can be carried out to obtain coinductive families with primitive corecursion.

Acknowledgement. We thank Tarmo Uustalu for communicating the example of triangular matrices to us.

References

[BB85] Corrado Böhm and Alessandro Berarducci. Automatic synthesis of typed λ-programs on term algebras. *Theoretical Computer Science*, 39:135–154, 1985.

[BP99] Richard Bird and Ross Paterson. Generalised folds for nested datatypes. *Formal Aspects of Computing*, 11(2):200–222, 1999.

[Geu92] Herman Geuvers. Inductive and coinductive types with iteration and recursion. In Bengt Nordström, Kent Pettersson, and Gordon Plotkin, editors, *Proceedings of the 1992 Workshop on Types for Proofs and Programs, Båstad, Sweden, June 1992*, pages 193–217, 1992. Electronically available via `ftp://ftp.cs.chalmers.se/pub/cs-reports/baastad.92/proc.dvi.Z`.

[Hin01] Ralf Hinze. Manufacturing datatypes. *Journal of Functional Programming*, 11(5):493–524, 2001.

[Men87] Nax P. Mendler. Recursive types and type constraints in second-order lambda calculus. In *Proceedings of the Second Annual IEEE Symposium on Logic in Computer Science, Ithaca, N.Y.*, pages 30–36. IEEE Computer Society Press, 1987.

[Oka96] Chris Okasaki. *Purely Functional Data Structures.* PhD thesis, Carnegie Mellon University, 1996.

[Oka99] Chris Okasaki. From Fast Exponentiation to Square Matrices: An Adventure in Types. In *International Conference on Functional Programming*, pages 28–35, September 1999.

[SU99] Zdzisław Spławski and Paweł Urzyczyn. Type fixpoints: Iteration vs. recursion. *SIGPLAN Notices*, 34(9):102–113, 1999. Proceedings of the 1999 International Conference on Functional Programming (ICFP), Paris, France.