Inductive Data Types for Predicate Transformers

Oege de Moor

September 14, 1995

Keywords: programming calculi, refinement calculus, predicate transformers, data types, catamorphisms

0 Introduction

Modern functional programming languages [5, 6] and specification formalisms [3] are built around the notion of inductive data types and homomorphisms on these data types. Such homomorphisms, which correspond to the familiar *fold* or *reduce* operators in functional programming, are called *catamorphisms*. In this note, it is shown how catamorphisms can be generalised from functions to relations, and from relations to predicate transformers. The first step of this generalisation (from functions to relations) was already achieved in a slightly different setting by Backhouse *et al.* [2]; the generalisation to predicate transformers is new.

In practical terms, the main result presented here says that a calculus based on predicate transformers (like the refinement calculus studied by Back, Morgan and others [1, 14]) can be enriched with program constructors for iterating over inductive data types. The refinement calculus did already allow a notion of inductive data types, but until now it lacked the program constructors that are associated with such data types, namely catamorphisms.

It will be assumed that the reader is acquainted with the categorical treatment of data types as algebras for an endofunctor. A readily accessible introduction to this subject, with many applications to program derivation, is Malcolm's paper [12]. It is also useful if the reader has some familiarity with the connection between relations and predicate transformers; see *e.g.* [9].

0.0 Notation Composition is written in diagrammatic order, and is denoted by a semi–colon. As usual, id_B stands for the identity arrow on B, and so one might write, for example,

$$\phi\,;id_B\,;\psi = \phi\,;\psi$$

where $\phi : A \to B$ and $\psi : B \to C$. Furthermore, function application is denoted by an infix dot, and it associates to the right.

1 Order–enriched Categories

An *order–enriched* category is a category where each homset comes equipped with a partial order $\subseteq$, and composition is monotonic. Expressed as a formula, monotonicity of composition reads as follows:

$$(\phi_0 \subseteq \phi_1) \wedge (\psi_0 \subseteq \psi_1) \quad \Rightarrow \quad (\phi_0\,;\psi_0) \subseteq (\phi_1\,;\psi_1)\ .$$

It is common practice to require that functors between order–enriched categories be monotonic, but we shall not do so here.

1.0 Functions The order–enriched category of sets and total functions is denoted by *Fun*. Functions are written with lower case identifiers: f, g, h, k. The partial order on arrows is just equality of functions. One could think of *Fun* as a model of a specification formalism where all specifications are total, deterministic mappings from input to output. An example of such a specification formalism is the calculus of functions developed by Bird and Meertens [3, 13].

1.1 Relations Let A and B be sets. A *relation* from A to B is a subset of the cartesian product $A \times B$. Relations are denoted by upper case identifiers: R, S, T. In the sequel, we shall use the notation

$$a(R)b$$

as shorthand for $(a, b) \in R$. The category *Rel* of sets and relations is defined as follows. Its objects are sets and its arrows are relations, and composition is defined in the usual manner:

$$a(R\,;S)c \quad = \quad \exists\, b : (\, a(R)b \wedge b(S)c\,)\ .$$

The order on arrows in *Rel* is inclusion of relations. Relations are a model of a specification formalism where specifications are partial, nondeterministic mappings from input to output. They give rise to a much richer calculus than functions; in particular, we have the *converse* operator, which reverses a relation:

$$R^\circ \quad = \quad \{\, (b, a) \mid (a, b) \in R \,\}\ ,$$

In the sequel, we shall make frequent use of the division operator $(/)$ on relations, which can be characterised by the equivalence:

$$(T\,;S \subseteq R) \;\;\equiv\;\; (T \subseteq R/S)\ ,$$

for $R : A \to B$, $S : C \to B$ and $T : A \to C$. This characterisation is the starting point of a paper by Hoare and He, who have named $(/)$ the *weakest prespecification* [10]. The above equivalence explains that terminology: R/S is the weakest specification of a program T such that $(T\,;\,S)$ implements R. The pointwise definition of the weakest prespecification is

$$a(R/S)c \;\;\equiv\;\; \forall\, b : (c(S)b \Rightarrow a(R)b)\ .$$

The operator $(/)$ binds more tightly than composition, so we write for example

$$f\,;R/S \;\;=\;\; (f\,;R)/S\ . \tag{0}$$

Note that this equation is only valid when f is a function.

1.2 Monotonic Predicate Transformers Slightly abusing terminology, we shall take the view that a *predicate* on A is a subset of A. The set of all subsets of A is called the *powerset* of A, and is denoted by $\mathrm{P}.A$. The category *Tran* of sets and predicate transformers is defined as follows. Its objects are sets, and its arrows are monotonic functions between powersets. That is, whenever

$$\forall\, x, y \;:\; (\,x \subseteq y \;\;\Rightarrow\;\; h.x \subseteq h.y\,)$$

for $h : \mathrm{P}.A \to \mathrm{P}.B$ in *Fun* we have an arrow

$$h^\flat : A \to B$$

in *Tran*. The partial order on arrows is given by pointwise inclusion:

$$h^\flat \subseteq k^\flat \;\;\equiv\;\; \forall\, x : (h.x \subseteq k.x)\ .$$

Note that *Tran* is *not* a subcategory of *Fun* because the order is different. The category *Tran* is a model of a specification formalism like the refinement calculus [1, 14], where specifications are given as pairs of pre– and post–conditions [8]. Note, however, that we do not restrict ourselves to a single universe of states: we allow heterogeneous predicate transformers of type $A \to B$ with $A \neq B$. Most expositions of the refinement calculus only consider homogeneous predicate transformers of type $A \to A$ for some fixed universe A.

2 Relations and Set–valued Functions: An Isomorphism

In set theory, a relation $R : A \to B$ is usually represented as a subset of the cartesian product $A \times B$, and we have adopted this viewpoint here. One could also view a relation $R : A \to B$ as a set–valued function $f : A \to \mathrm{P}.B$. Indeed, there is an isomorphism between relations and set–valued functions, as expressed by the equivalence below:

$$(f \,;\ni_B \;=\; R) \quad\equiv\quad (f = \Lambda.R) \;, \tag{1}$$

where $\ni_B : \mathrm{P}.B \to B$ is the converse of the membership relation on subsets of B. Freyd and Ščedrov have named $\ni$ *epsiloff* [7]. The function $\Lambda.R$ is called the *power transpose* of R. Power transpose can also be written using a set comprehension:

$$(\Lambda.R).a \;=\; \{\, b \mid a(R)b \,\} \;.$$

From the defining equivalence of power transpose, one may derive a number of useful identities, for example (again taking f to be a function)

$$\Lambda.(f \,;\, R) \;=\; f \,;\Lambda.R \;. \tag{2}$$

The *existential image* functor $\mathrm{E} : \mathit{Rel} \to \mathit{Fun}$ can be defined in terms of epsiloff and power transpose:

$$\mathrm{E}.R \;=\; \Lambda.(\ni_A ; R)$$

where $R : A \to B$. A more conventional way of defining the existential image functor is the following:

$$(\mathrm{E}.R).x \;=\; \{\, b \mid\, \exists\, a \in x : a(R)b \,\} \;.$$

To appreciate the definition in terms of power transpose, the reader might wish to prove that

$$\Lambda.(R \,; S) \;=\; \Lambda.R \,; \mathrm{E}.S \;. \tag{3}$$

The first definition of existential image suggests a dual concept, which is obtained by replacing the composition $(\ni_A ; \text{-})$ by a weakest prespecification:

$$\mathrm{A}.R \;=\; \Lambda.(\ni_B / R) \tag{4}$$

for $R : A \to B$. This gives a contravariant functor $\mathit{Rel} \to \mathit{Set}^{op}$, which is known as the *universal image* functor. When applied to functions, universal image may be expressed in terms of existential image:

$$\mathrm{A}.f \;=\; \mathrm{E}.f^{\circ} \;. \tag{5}$$

Note that for any relation R both its existential image $\mathrm{E}.R$ and its universal image $\mathrm{A}.R$ are monotonic functions between powersets. In other words, we have functors

$$\mathrm{E}^\flat : Rel \to Tran \quad \text{and} \quad \mathrm{A}^\flat : Rel \to Tran^{op} \ ,$$

which are defined by $\mathrm{E}^\flat.R = (\mathrm{E}.R)^\flat$ and $\mathrm{A}^\flat.R = (\mathrm{A}.R)^\flat$, and these functors are monotonic. The functor $\mathrm{A}^\flat$ reverses both composition and the partial order on arrows; $\mathrm{A}^\flat$ is the usual mapping of relations to conjunctive predicate transformers [9].

3 Extending Functors

3.0 From Functions to Relations Consider a functor $F : Fun \to Fun$ on functions, and a functor $G : Rel \to Rel$ on relations. These two functors *agree on functions* if for every function h, $G.h \ = \ F.h$. It is not always possible to construct a functor $Rel \to Rel$ that agrees with F on functions, but we can characterise those functors for which extension is possible. Say that a functor $F : Fun \to Fun$ is a *relator* if the following implication holds in *Rel*:

$$h^\circ \, ; k = f \, ; g^\circ \ \Rightarrow \ (F.h)^\circ \, ; F.k = F.f \, ; (F.g)^\circ \ .$$

Theorem 0 (see *e.g.* Carboni, Kelly and Wood [4]) *Let $F : Fun \to Fun$ be a functor. There exists at most one monotonic functor $Rel \to Rel$ that agrees with F on functions, say $F^\star$. This unique functor $F^\star$ exists if and only if F is a relator. Furthermore, every monotonic functor on relations arises in this way.*

Backhouse *et al.* [2] have proposed a subtly different definition of *relator*; they say that the extension $F^\star$ is a relator, and they do not consider F separately. Their definition is consistent with ours, because all their relators are of the form $F^\star$. Most functors that are used in defining the types of functional programming (in particular so–called *polynomial* functors) are relators.

3.1 From relations to predicate transformers The extension of functors from relations to predicate transformers is not as simple as the extension from functions to relations. In particular, there are many functors that do not have an extension in the strict sense of the word. It is necessary, therefore, to consider a weakening of the definition of functor. Let $\mathcal{D}$ and $\mathcal{E}$ be order–enriched categories. An *upfunctor* $H : \mathcal{D} \subseteq \mathcal{E}$ is a monotonic morphism of graphs such that

$$\begin{aligned} H.id_A &= id_{H.A} \\ H.(\phi \, ; \psi) &\subseteq H.\phi \, ; H.\psi \end{aligned}$$

for $\phi : A \to B$ and $\psi : B \to C$ in $\mathcal{D}$. Note that identities are preserved as usual, while distribution through composition is truly weakened to an inequation.

The partial order on the arrows of the category $\mathcal{E}$ may be extended to upfunctors $\mathcal{D} \subseteq \mathcal{E}$ as follows. We write $H \subseteq K$ if for all arrows ϕ in $\mathcal{D}$, $H.\phi \subseteq K.\phi$.

Let $G : \mathit{Rel} \to \mathit{Rel}$ be a functor, and let $H : \mathit{Tran} \subseteq \mathit{Tran}$ be an upfunctor. We say that G and H *agree on relations* if

$$\mathrm{A}.G.R \;=\; H.\mathrm{A}.R$$

for all relations R.

Theorem 1 (Gardiner, Martin and De Moor [8]) *Let $G : \mathit{Rel} \to \mathit{Rel}$ be a monotonic functor. Then there exists a minimum upfunctor $G^\star : \mathit{Tran} \subseteq \mathit{Tran}$ that agrees with G on relations. This upfunctor is given by*

$$G^\star.h^\flat \;=\; (\mathrm{A}.G.\ni_A \,;\, \mathrm{E}.G.\Lambda^{-1}.h)^\flat \;.$$

where $h^\flat : A \to B$ in Tran.

4 Extending functors preserves initial algebras

Let $\mathcal{D}$ be an order–enriched category, and let F be an upfunctor from $\mathcal{D}$ to itself. An *F–algebra* is an arrow of type $F.A \to A$ for some A. An *initial* F–algebra is an arrow $\tau : F.C \to C$ such that for each F–algebra $\phi : F.A \to A$ there exists an arrow $(\!|\phi|\!) : C \to A$ that satisfies the equivalence

$$(\tau \,;\, \psi = F.\psi \,;\, \phi) \;\;\equiv\;\; (\psi = (\!|\phi|\!)) \;.$$

Arrows of the form $(\!|\phi|\!)$ are called *catamorphisms*. For example, consider the functor $F : \mathit{Set} \to \mathit{Set}$ defined by

$$F.h \;=\; id_\top + (id_A \times h) \;,$$

where $\top$ is the terminal object of Set. The initial F–algebra is the data type of finite lists with elements from A, and the catamorphisms are instances of the well–known *fold–right* operator of functional programming.

The above definitions can be dualised: an initial F^{op}–algebra in $\mathcal{D}^{op}$ is called a *final F–coalgebra* in $\mathcal{D}$. If $(\!|\phi|\!)$ is a catamorphism in $\mathcal{D}^{op}$, we say it is an *anamorphism* in $\mathcal{D}$, and it is denoted by $[\![\phi]\!]$.

Theorem 2 *Let F be a relator that has initial algebra τ. Then τ is also an initial $F^\star$–algebra, and*

$$(\![R]\!) \;=\; \Lambda^{-1}.(\![\Lambda.(F^\star.\ni_A \;;\; R)]\!)$$

for all $R : F.A \to A$ in Rel.

Proof

$$\begin{array}{lcl}
\tau \,;\, (\![R]\!) = F^\star.(\![R]\!) \,;\, R & \equiv & \{ \text{ equivalence (1) } \} \\
\tau \,;\, (\![R]\!) = F^\star.(\Lambda.(\![R]\!) \,;\, \ni_A) \;;\; R & \equiv & \{ \; F^\star \text{ functor, } \Lambda.(\![R]\!) \text{ function (2) } \} \\
\tau \,;\, (\![R]\!) = F.\Lambda.(\![R]\!) \,;\, F^\star.\ni_A \,;\, R & \equiv & \{ \text{ equivalence (1), } \tau \text{ function } \} \\
\tau \,;\, \Lambda.(\![R]\!) = F.\Lambda.(\![R]\!) \;;\; \Lambda.(F^\star.\ni_A \,;\, R) & \equiv & \{ \; \tau \text{ initial } F\text{–algebra in } \mathit{Fun} \; \} \\
\Lambda.(\![R]\!) = (\![\Lambda.(F^\star.\ni_A \;;\; R)]\!) & \equiv & \{ \; \Lambda \text{ isomorphism } \} \\
(\![R]\!) = \Lambda^{-1}.(\![\Lambda.(F^\star.\ni_A \;;\; R)]\!) & &
\end{array}$$

Lemma 3 (Lambek [11]) *Let $\mathcal{D}$ be a category, and let $F : \mathcal{D} \to \mathcal{D}$ be a functor with initial algebra τ. Then τ is an isomorphism.*

Lemma 4 (Backhouse *et al.* [2]) *Let $F : \mathit{Fun} \to \mathit{Fun}$ be a relator with initial algebra τ. Then for all R and T in Rel,*

$$(T = (\![R]\!)^\circ) \;\;\equiv\;\; (T = R^\circ \,;\, F^\star.T \,;\, \tau)\ .$$

We are now in a position to prove the main result of this note. It says that an initial F–algebra in *Fun* lifts to a final $F^{\star\star}$–coalgebra in *Tran*, when F is a relator. The relevant anamorphism is expressed in terms of a relational catamorphism. Theorem 2 shows how such relational catamorphisms can be expressed in terms of functional catamorphisms.

Theorem 5 *Let $F : \mathit{Fun} \to \mathit{Fun}$ be a relator with initial algebra τ. Then $\mathrm{A}^\flat.\tau$ is a final coalgebra of $F^{\star\star} : \mathit{Tran} \subseteq \mathit{Tran}$, and*

$$[\![h^\flat]\!] \;=\; (\Lambda.(\![((\Lambda^{-1}.h)/(F^\star.\ni_A))^\circ]\!)^\circ)^\flat$$

for all $h^\flat : A \to F.A$ in Tran.

Proof Let C be the target of the initial algebra $\tau : F.C \to C$, and let $k^\flat : A \to C$ in *Tran*. Define $g : \mathrm{P}.F.A \to \mathrm{P}.F.C$ in *Fun* by

$$g^\flat \;=\; F^{\star\star}.k^\flat \;.$$

First observe that

$$\begin{array}{lcl}
\Lambda^{-1}.(h\,;g) & = & \{\text{ equation (2) }\} \\
h\;;\Lambda^{-1}.g & = & \{\text{ definition } g\text{, theorem 1 }\} \\
h\;;\Lambda^{-1}.(\mathrm{A}.F^\star.\ni_A\,;\mathrm{E}.F^\star.\Lambda^{-1}.k) & = & \{\text{ definition A (4), equation (3) }\} \\
h\;;\ni_{F.A}/(F^\star.\ni_A)\,;F^\star.\Lambda^{-1}.k & = & \{\text{ equations (0), (1) }\} \\
(\Lambda^{-1}.h)/(F^\star.\ni_A)\;;\;F^\star.\Lambda^{-1}.k & &
\end{array}$$

Furthermore, note that

$$\begin{array}{lcl}
\Lambda^{-1}.(k\,;\mathrm{A}.\tau) & = & \{\text{ equation (5) }\} \\
\Lambda^{-1}.(k\,;\mathrm{E}.\tau^\circ) & = & \{\text{ equation (3) }\} \\
\Lambda^{-1}.k\;;\;\tau^\circ & &
\end{array}$$

Using these two results, one may establish the desired equivalence as follows:

$$\begin{array}{lcl}
k^\flat\;;\mathrm{A}^\flat.\tau \;=\; h^\flat\;;\;F^{\star\star}.k^\flat & \equiv & \{\text{ definitions } g \text{ and } \mathit{Tran}\ \} \\
k\;;\mathrm{A}.\tau \;=\; h\,;g & \equiv & \{\ \Lambda\text{ isomorphism }\} \\
\Lambda^{-1}.(k\;;\mathrm{A}.\tau) \;=\; \Lambda^{-1}.(h\,;g) & \equiv & \{\text{ previous two calculations }\} \\
\Lambda^{-1}.k\;;\tau^\circ \;=\; (\Lambda^{-1}.h)/(F^\star.\ni_A)\;;\;F^\star.\Lambda^{-1}.k & \equiv & \{\text{ lemma 3 }\} \\
\Lambda^{-1}.k \;=\; (\Lambda^{-1}.h)/(F^\star.\ni_A)\;;\;F^\star.\Lambda^{-1}.k\;;\;\tau & \equiv & \{\text{ lemma 4 }\} \\
\Lambda^{-1}.k \;=\; (\!|((\Lambda^{-1}.h)/(F^\star.\ni_A))^\circ|\!)^\circ & \equiv & \{\ \Lambda\text{ isomorphism }\} \\
k \;=\; \Lambda.(\!|((\Lambda^{-1}.h)/(F^\star.\ni_A))^\circ|\!)^\circ & &
\end{array}$$

It remains to show that $\Lambda.(\!|((\Lambda^{-1}.h)/(F^\star.\ni_A))^\circ|\!)^\circ$ is indeed a monotonic function on the subset ordering. Since $\mathrm{A}^\flat.\tau$ is an isomorphism (because τ is an isomorphism and functors preserve isomorphisms), we have

$$\Lambda.(\!|((\Lambda^{-1}.h)/(F^\star.\ni_A))^\circ|\!)^\circ \;=\; h\;;\;g\;;\;\mathrm{A}.\tau^\circ \;.$$

The right–hand side of this equation is the composite of three monotonic functions.
End of proof

References

[1] R. J. R. Back. A calculus of refinements for program derivation. *Acta Informatica*, 25:593–624, 1988.

[2] R. C. Backhouse, P. De Bruin, G. Malcolm, T. S. Voermans, and J. C. S. P. Van der Woude. Relational catamorphisms. In B. Möller, editor, *Proceedings of the IFIP TC2/WG2.1 Working Conference on Constructing Programs*, pages 287–318. Elsevier Science Publishers B.V., 1991.

[3] R. S. Bird. Lectures on constructive functional programming. In M. Broy, editor, *Constructive Methods in Computing Science*, volume 55 of *NATO ASI Series F*, pages 151–216. Springer–Verlag, 1989.

[4] A. Carboni, G.M. Kelly, and R.J. Wood. A 2–categorical approach to geometric morphisms I. *Cahiers de Topologie et Geometrie Differentielle Categoriques*, 32(1):47–95, 1991.

[5] R. Cockett and T. Fukushima. Draft: About charity. Dept. of Computer Science, University of Calgary, Calgary, Alberta, Canada. Available via anonymous ftp from cpsc.ucalgary.ca., 1991.

[6] R. Cockett and D. Spencer. Strong categorical datatypes I. In *Category Theory 1991*, pages 141–169. Canadian Mathematical Society Proceedings, 1992.

[7] P. J. Freyd and A. Ščedrov. *Categories, Allegories*, volume 39 of *Mathematical Library*. North–Holland, 1990.

[8] P. H. B. Gardiner, C. E. Martin, and O. de Moor. An algebraic construction of predicate transformers. *Science of Computer Programming*, 22(1-2):21–44, 1994.

[9] W. H. Hesselink. Modalities of nondeterminacy. In *Beauty is our Business — A birthday salute to Edsger W. Dijkstra*, chapter 21, pages 182–192. Springer–Verlag, 1990.

[10] C. A. R. Hoare and J. He. The weakest prespecification. Technical Monograph PRG-44, Programming Research Group, 11 Keble Road, Oxford OX1 3QD, England, June 1985. Abridged version appeared in: *Information Processing Letters*, 24:127–132, 1987.

[11] J. Lambek. A fixpoint theorem for complete categories. *Mathematische Zeitschrift*, 103:151–161, 1968.

[12] G. Malcolm. Data structures and program transformation. *Science of Computer Programming*, 14:255–279, 1990.

[13] L. Meertens. Algorithmics — towards programming as a mathematical activity. In J. W. De Bakker, M. Hazewinkel, and J. K. Lenstra, editors, *Mathematics and Computer Science*, volume 1 of *CWI Monographs*, pages 3–42. North–Holland, 1987.

[14] C. C. Morgan. *Programming from Specifications*. 2nd edition, Prentice–Hall, 1994.