A Recursion Theorem for Predicate Transformers on Inductive Data Types

David A. Naumann
Mathematics and Computer Science
Southwestern University, Georgetown, TX 78626 U.S.A.

1994

[**Note: converted Feb. 2003 from obsolete latex file.**]

Keywords: programming calculi, refinement calculus, inductive data type, catamorphism, predicate transformer, higher types, lax exponent.

1 Introduction

The calculus of predicate transformers has proved fruitful for the derivation of state-transformation programs in conventional languages with general recursion and demonic nondeterminacy [8, 10, 11]. Derivations involving direct manipulation of predicates can be unwieldy, so it is hoped that the algebraic, program-level style [5] that has proved so successful in the simpler calculus of functions can be extended to predicate transformers. In the calculus of functions, structural recursion on inductive data types —homomorphisms from initial algebras— is used to great effect. De Moor showed that inductive data types can be lifted from the category **Fun** of functions to the ordered category of predicate transformers [7]. This paper takes another step towards extending the function calculus to predicate transformers, proving a generalization of Lawvere's recursion theorem (*e.g*, [9]) for predicate transformers. This gives a form of recursion more general than the defining property for inductive types but with similar algebraic properties. It applies to both programs and specifications, *i.e*, the full "refinement calculus" (*e.g*, [1, 14]).

We introduce the theorem by giving a special case, in the calculus of functions; first at the data level, then at the program (*i.e*, function) level. Then section 2 gives prerequisites and section 3 proves the theorem.

As an inductive type, the natural numbers $(\mathbb{N}, zero, succ)$ are axiomatized by the following principle of definition:

> (a) Given a set A, an element $a \in A$, and a function $h \in A \to A$, there is a unique function $f \in \mathbb{N} \to A$ such that
>
> $$f.0 = a \quad \text{and} \quad f.(n+1) = h.(f.n) \quad .$$

(Function application is written with a dot, which binds more tightly than other binary operators.) The function f is called a *catamorphism*, and is sometimes denoted by $(\!|a, h|\!)$. At the program level, the defining equations become $(zero\,;f) = a$ and $(succ\,;f) = (f\,;h)$, where " ; " is composition of functions.

Lawvere's theorem gives the following more general principle:

> (b) Given A, B and functions $g \in A \to B$ and $h \in B \to B$, there is a unique function $f \in A \times \mathbb{N} \to B$ such that, for all $a \in A$,
>
> $$f.(a, 0) = g.a \quad \text{and} \quad f.(a, n+1) = h.(f.(a, n)) \quad .$$

At the program level, the defining equations are

$$\langle a\,,\, zero\rangle\,;f = a\,;g \quad \text{and} \quad (a \times succ)\,;f = (a \times id)\,;f\,;h$$

(where $\langle ?, ?? \rangle$ denotes pairing). A corollary of the theorem is the following principle:

> (c) Given A, B, functions $g \in A \to B$ and $h \in A \times \mathbb{N} \times B \to B$, there is a unique $f \in A \times \mathbb{N} \to B$ such that
>
> $$f.(a, 0) = g.a \quad \text{and} \quad f.(a, n+1) = h.(a, n, (f.(a, n))) \quad .$$

Of course this is the well-known notion of primitive recursion on the naturals. Lawvere's theorem is that (b) (hence (c)) follows from just the algebraic axiom (a) and the algebraic laws for higher order functions (exponents and products). Thus it holds not just in **Fun** but in many important categories.

Lawvere's theorem is easily generalized to inductive types besides $\mathbb{N}$. Inductive types can be described as initial algebras for functors (*e.g*, [3]); then a catamorphism is precisely a unique homomorphism given by initiality. The functor for the naturals is given by the mapping $x \mapsto \{*\} + x$, where $+$ is disjoint union, *i.e*, the coproduct in **Fun**. Lawvere's theorem may be generalized to arbitrary functors on **Fun** of the form $x \mapsto K + F.x$, where K is some fixed set and $F : \mathbf{Fun} \to \mathbf{Fun}$ is a functor. This generalization

involves an extra parameter: a natural transformation ϕ from $F.? \times id$ to $F.(? \times id)$ where id is the identity functor.

From de Moor's result (Theorem 2 below) it follows that initial algebras in **Fun** are also initial in our category **Prog** of predicate transformers. Lawvere's theorem for **Prog** does not immediately follow, as **Prog** lacks categorical products and exponents. Indeed, the appropriate generalization of (b) is that for any predicate transformers g, h there are least and greatest solutions f, rather than a unique one. Since homsets of **Prog** are complete lattices, it is possible to prove such a result using Knaster-Tarski — *i.e*, by reducing primitive recursion to general recursion. Our result is that a closed form solution f in (b) can be constructed just as in the proof of Lawvere's theorem, using the "weak exponent" of **Prog** without recourse to fixed-point theory. Moreover, this is the greatest solution, although the proof of that fact uses an induction principle (Lemma 7) for initial algebras which is proved using fixed-point theory. We omit the similar generalization of corollary (c).

For familiarity, we give background definitions and facts in terms of a specific model, namely predicate transformers between sets. However, our recursion theorem holds in any ordered category equipped with weak products and weak exponents as described in section 2. This helps justify the hope that a small number of algebraic laws will suffice to axiomatize the refinement calculus.

Backhouse *et al.* [2] investigate catamorphisms in a setting similar to **Fun** and the category **Rel** of relations. It would be interesting to see whether a version of Lawvere's theorem can be proved in their setting, without recourse to Knaster-Tarski.

2 Background

We use predicate transformers on higher types, axiomatized as a kind of weak exponent which is based on a weak product. We deviate from previous categorical treatments of predicate transformers [7, 16, 13] and work in the opposite **Prog** of the category **Tran** of predicate transformers. A *predicate transformer* is a monotonic function from predicates to predicates; a predicate is a subset (of some set of states or data values). The objects of **Prog** are those of **Fun**: all sets. For any sets A, B, define $\mathbf{Prog}(A, B)$ to be the set of predicate transformers from B to A. Thus g is in $\mathbf{Prog}(A, B)$ iff g is a function sending each subset β of B to $g.\beta \subseteq A$ and g is monotonic with respect to inclusion. If g distributes over all intersections —including the empty intersection, *i.e*, $g.B = A$ — it is called a *map* (it is a comap in the opposite category **Tran**). A map that also distributes over all unions

is here called a *bimap*. Bimaps model deterministic, everywhere-terminating programs; **Fun** is embedded in **Prog** as its subcategory of bimaps.

The composite $(g\,;h)$ of g with $h \in \mathbf{Prog}(B,C)$ is the element of $\mathbf{Prog}(A,C)$ defined by $(g\,;h).\gamma = g.(h.\gamma)$ for all $\gamma \subseteq C$. The refinement relation $\sqsubseteq$ is defined pointwise: $g \sqsubseteq g'$ iff $(\,\forall\,\beta\,::\,g.\beta \subseteq g'.\beta\,)$. In the sequel, transitivity of $\sqsubseteq$ and monotonicity of ";" are used without mention; d, e, f, g, h, x, y denote predicate transformers.

Weak exponents. For each A, B, the data type $A \rightsquigarrow B$ of programs is defined by $A \rightsquigarrow B = \mathbf{Prog}(A,B)$.[1] We also need the type $A \otimes B$ which is the Cartesian product of sets, although the action of $\otimes$ on arrows satisfies rather weak laws in general [16, 13], so $A \otimes B$ is not a categorical product in **Prog**.[2] (Note that " ; " binds tighter than $\otimes$.) There is also an "application" program $\mathsf{ap} \in \mathbf{Prog}((A \rightsquigarrow B) \otimes A, B)$, and for each $g \in \mathbf{Prog}(C \otimes A, B)$ there is a "Curried program" $\mathsf{cur}.g \in \mathbf{Prog}(C, A \rightsquigarrow B)$.[3] Only a few of the laws of $\otimes$ and cur are needed here (for other laws, and for proofs, see [16]).

Theorem 1 *For all d, e, f, g, h for which the composites are defined:*

$$(\mathsf{cur}.g \otimes id)\,;\mathsf{ap} = g \tag{1}$$

$$g\,;\mathsf{cur}.h \sqsupseteq \mathsf{cur}.((g \otimes id)\,;h) \quad \textit{if } g \textit{ is a map} \tag{2}$$

$$g\,;\mathsf{cur}.h = \mathsf{cur}.((g \otimes id)\,;h) \quad \textit{if } g \textit{ is a bimap} \tag{3}$$

$$\mathsf{cur}.g \quad \textit{is a map, for all } g \tag{4}$$

$$(d \otimes e)\,;(f \otimes g) = d\,;f \otimes e\,;g \quad \textit{if } d, e, f, g \textit{ are maps} \tag{5}$$

Catamorphisms. Functors $F : \mathbf{Fun} \to \mathbf{Fun}$ that are "relators" lift to monotonic functors on **Rel** and to lax functors $\hat{F} : \mathbf{Prog} \to \mathbf{Prog}$ ($\hat{F}$ need not preserve composition). A relator is a functor that preserves surjectivity and preserves the factorization of relations into functions; constant functors, Cartesian product, and disjoint union are all relators. For the precise definition of "relator" and more details on the results in the rest of this section,

[1] This is one of several alternatives discussed in [16]; we need this particular alternative so that h in Theorem 8 satisfies (1) as an equality.

[2] It is these laws that characterize the particular kind of weak product we are interested in . In **Prog** , $\otimes$ is not even a local product in the sense of [12, 4].

[3] For completeness, here are the definitions of $\otimes, \mathsf{cur}, \mathsf{ap}$. For $\gamma \subseteq A \times B$, define $(f \otimes g).\gamma = (\,\cup\,\alpha, \beta\,:\,\alpha \times \beta \subseteq \gamma\,:\,f.\alpha \times g.\beta\,)$. For $c \in C$ and $\delta \subseteq A \rightsquigarrow B$, define

$$c \in \mathsf{cur}.g.\delta \;\equiv\; (\,\forall\, h\,:\,h \in A \rightsquigarrow B \,\wedge\, h \sqsupseteq \mathsf{cu}.g.c\,:\,h \in \delta\,)$$

where $\mathsf{cu}.g.c \in \mathbf{Prog}(A,B)$ is defined by $a \in \mathsf{cu}.g.c.\beta \;\equiv\; (c,a) \in g.\beta$. Finally, define $(g,a) \in \mathsf{ap}.\beta \;\equiv\; a \in g.\beta$ for $\beta \subseteq B$ and $g \in A \rightsquigarrow B$.

see de Moor [7] (also [13, 6]). We only need a few properties of $\hat{F}$, which are given below following the main theorem on inductive types. One other ingredient is needed: the *universal image functor* $\mathbb{A} : \mathbf{Rel} \to \mathbf{Prog}$ that sends each binary relation to a map, defined by $b \in \mathbb{A}.R.\gamma \;\equiv\; (\forall\, c : bRc : c \in \gamma)$. The embedding of **Fun** in **Prog** is given by $\mathbb{A}$.

Let $F : \mathbf{Fun} \to \mathbf{Fun}$ be a relator and K a set. Then there is a relator $G : \mathbf{Fun} \to \mathbf{Fun}$ defined by $G.A = K + F.A$ and $G.h = id_K + F.h$.

Theorem 2 *If G has an initial algebra $\tau : G.N \to N$ (in* **Fun***) then $\hat{G}$ has an initial algebra $\hat{\tau}$ (in* **Prog***).*

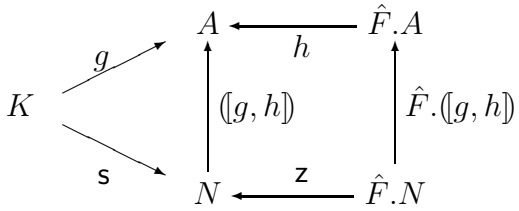
Theorem 2 is due to de Moor [7], who stated it for final coalgebras in the opposite category **Tran** instead of **Prog**. The algebra $\hat{\tau}$ is $\mathbb{A}.\tau$.

The form of definition of G, and the fact that the coproduct $+$ in **Fun** lifts to a coproduct $\oplus$ in **Prog** [13, 15] implies that $\hat{\tau}$ has type $\hat{\tau} \in \mathbf{Prog}(K \oplus \hat{F}.N, N)$. The coproduct property of $\oplus$ in **Prog** implies that there are $\mathsf{s} \in \mathbf{Prog}(K, N)$ and $\mathsf{z} \in \mathbf{Prog}(\hat{F}.N, N)$ such that $\hat{\tau} = [\mathsf{s}, \mathsf{z}]$ (where $[?, ??]$ is the copairing (case) construct). Moreover, the uniqueness property of catamorphisms can be expressed in the following way, which allows us to dispense with G and explicit coproducts in favor of s, z.

Corollary 3 *For each A, $g \in \mathbf{Prog}(K, A)$, and $h \in \mathbf{Prog}(\hat{F}.A, A)$, there is $(\!|g, h|\!) \in \mathbf{Prog}(N, A)$ such that for all x*

$$x = (\!|g, h|\!) \quad\equiv\quad \mathsf{s} \,;\, x = g \;\;\wedge\;\; \mathsf{z} \,;\, x = \hat{F}.x \,;\, h \tag{6}$$

Instantiating (6) with $x := (\!|g, h|\!)$ gives the following commuting diagram.



Lemma 4 s *and* z *are bimaps.*

Proof: s and z may be constructed by applying $\mathbb{A}$ to the initial algebra in **Fun**, and $\mathbb{A}$ sends functions to bimaps. ∎

Lemma 5 *If g, h (in corollary 3) are maps then so is $(\!|g, h|\!)$.*

Proof: If g and h are maps, then they correspond to relations, by the inverse of $\mathbb{A}$. The initiality property of G in **Rel** gives a relational catamorphism for g, h, which lifts by $\mathbb{A}$ to a map which is a catamorphism. Since catamorphisms are uniquely determined, that map is $(\!|g, h|\!)$. ∎

Lemma 6 *For any map* g , $\hat{F}.g$ *is a map and* $\hat{F}.(g\,;h) = \hat{F}.g\,;\hat{F}.h$ *for any* h .

Proof: Property of the lifting construction [13, 6]. ∎

Lemma 7 $\mathsf{s};x \sqsubseteq g \;\wedge\; \mathsf{z};x \sqsubseteq \hat{F}.x;h \;\Rrightarrow\; x \sqsubseteq (\!|g,h|\!)$, *for any* x,g,h .

Proof: The homsets of **Prog** are complete lattices, and $(\!|g,h|\!)$ can be expressed as a unique fixed point. For details see the references [7, 2]. ∎

3 The Theorem

In the rest of the paper, **Prog** is any ordered category with the structure described by the Theorems and Lemmata of section 2, and **Fun** is its subcategory of bimaps. In this regard, it is well known that the map and bimap properties can be axiomatized by inequations.

Henceforth we assume that $\hat{F} : \mathbf{Prog} \to \mathbf{Prog}$ is a graph morphism that restricts to a relator on **Fun** , and we assume that $x \mapsto K \oplus \hat{F}.x$ has an initial algebra $[\mathsf{s},\mathsf{z}]$ as in Corollary 3 and lemmata 4–7. Moreover, we assume that there is a transformation $\phi : \hat{F}.? \otimes id \to \hat{F}.(? \otimes id)$ that is natural on maps.[4] That is, for each W there is a component ϕ_W ,

$$\phi_W \in \mathbf{Prog}(\hat{F}.W \otimes A\ ,\ \hat{F}.(W \otimes A)) \quad ,$$

and for any map g in $\mathbf{Prog}(X,W)$ we have

$$(\hat{F}.g \otimes id_A)\,;\phi_W = \phi_X\,;\hat{F}.(g \otimes id_A) \quad .$$

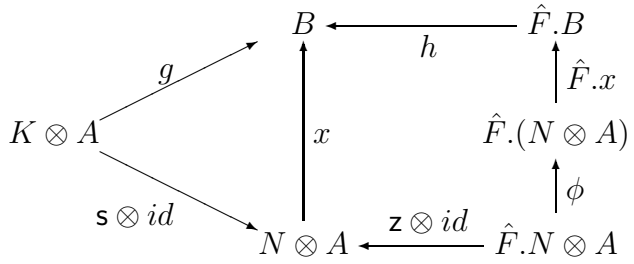
Applied to component arrows, $\mathbb{A}$ lifts natural transformations in **Fun** and **Rel** to transformations in **Prog** that are natural on maps.

Theorem 8 *If* A,B *are sets,* $g \in \mathbf{Prog}(K \otimes A, B)$, *and* $h \in \mathbf{Prog}(\hat{F}.B, B)$, *then there is a greatest solution for* x *in the conjunction of*

$$(\mathsf{s} \otimes id_A)\,;x = g \tag{7}$$
$$(\mathsf{z} \otimes id_A)\,;x = \phi_N\,;\hat{F}.x\,;h \tag{8}$$

[4] This follows (by fixing one argument) from the more general condition that ϕ is a natural transformation from $\hat{F}.? \otimes ??$ to $\hat{F}.(? \otimes ??)$ (sometimes called a "strength"). We do not need to allow the second argument $??$ to vary here, but it is needed to generalize (c) in section 1.

The theorem may be depicted as follows.



Initial algebras are isomorphisms (*e.g*, [3, 9]), so (7) and (8) can be rewritten as the single equation

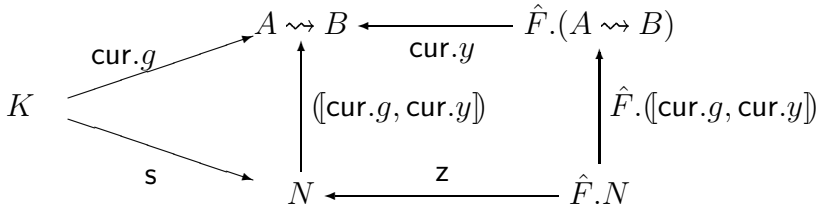
$$x = [(\mathsf{s} \otimes id)\,,\ (\mathsf{z} \otimes id)]^{-1}\ ;\ [g\,,\ (\phi\ ;\ \hat{F}.x\ ;\ h)] \quad .$$

Since the right side is monotonic in x, and homsets of **Prog** are complete lattices, there is a complete lattice of solutions (by Knaster-Tarski). However, we construct a solution without using lattice-theoretic properties; we show it to be maximal using Lemma 7. Lawvere's proof (*e.g*, [9, 3]) cannot be used directly, since cur is not a bijection, but we use the same idea: exponentiate to construct from the given g, h an instance of Corollary 3, from which a solution can be constructed. This is first shown to solve (7) and (8); then it is shown to be the greatest solution.

Proof: Define y by $y = (\phi_{A \rightsquigarrow B}\ ;\ \hat{F}.\mathsf{ap}\ ;\ h)$ as in

$$\hat{F}.(A \rightsquigarrow B) \otimes A \xrightarrow{\ \phi\ } \hat{F}.((A \rightsquigarrow B) \otimes A) \xrightarrow{\ \hat{F}.\mathsf{ap}\ } \hat{F}.B \xrightarrow{\ h\ } B \quad .$$

Now we have a configuration to which Corollary 3 applies:



Aiming for (7) and (8) with $g, h := \mathsf{cur}.g, \mathsf{cur}.y$, Corollary 3 with $g, h := \mathsf{cur}.g, \mathsf{cur}.y$ gives

$$\mathsf{s}\ ;\ (\!|\mathsf{cur}.g, \mathsf{cur}.y|\!) = \mathsf{cur}.g \tag{9}$$

$$\mathsf{z}\ ;\ (\!|\mathsf{cur}.g, \mathsf{cur}.y|\!) = \hat{F}.(\!|\mathsf{cur}.g, \mathsf{cur}.y|\!)\ ;\ \mathsf{cur}.y \tag{10}$$

From these we will derive a definition of f satisfying (7) and (8). For (7) we have

$$\begin{array}{ll} & (9) \\ \Rrightarrow & (\mathsf{s}\,;(\llbracket \mathsf{cur}.g, \mathsf{cur}.y \rrbracket \otimes id_A)\,;\mathsf{ap} = (\mathsf{cur}.g \otimes id_A)\,;\mathsf{ap} & \ldots \text{Leibniz} \\ \equiv & (\mathsf{s}\,;(\llbracket \mathsf{cur}.g, \mathsf{cur}.y \rrbracket \otimes id_A)\,;\mathsf{ap} = g & \ldots (1) \\ \equiv & (\mathsf{s} \otimes id_A)\,;((\llbracket \mathsf{cur}.g, \mathsf{cur}.y \rrbracket \otimes id_A)\,;\mathsf{ap} = g & \ldots (4),\ (5),\ \text{Lem. } 4{,}5 \\ \equiv & (\mathsf{s} \otimes id_A)\,;f = g & \ldots \text{defn. } f \text{ below} \end{array}$$

which suggests that f can be defined by $f = ((\llbracket \mathsf{cur}.g, \mathsf{cur}.y \rrbracket \otimes id_A)\,;\mathsf{ap}$. For (8), observe

$$\begin{array}{ll} = & (\mathsf{z} \otimes id_A)\,;f & \ldots \text{defn. } f \\ = & (\mathsf{z} \otimes id_A)\,;((\llbracket \mathsf{cur}.g, \mathsf{cur}.y \rrbracket \otimes id_A)\,;\mathsf{ap} & \ldots (4),\ (5),\ \text{Lemmata } 4{,}5 \\ = & (\mathsf{z}\,;(\llbracket \mathsf{cur}.g, \mathsf{cur}.y \rrbracket \otimes id_A)\,;\mathsf{ap} & \ldots (10) \\ = & (\hat{F}.(\llbracket \mathsf{cur}.g, \mathsf{cur}.y \rrbracket)\,;\mathsf{cur}.y \otimes id_A)\,;\mathsf{ap} & \ldots (4),\ (5),\ \text{Lemmata } 6{,}4{,}5 \\ = & (\hat{F}.(\llbracket \mathsf{cur}.g, \mathsf{cur}.y \rrbracket) \otimes id_A)\,;(\mathsf{cur}.y \otimes id_A)\,;\mathsf{ap} & \ldots (1) \\ = & (\hat{F}.(\llbracket \mathsf{cur}.g, \mathsf{cur}.y \rrbracket) \otimes id_A)\,;y & \ldots \text{defn. } y \\ = & (\hat{F}.(\llbracket \mathsf{cur}.g, \mathsf{cur}.y \rrbracket) \otimes id_A)\,;\phi_{A \rightsquigarrow B}\,;\hat{F}.\mathsf{ap}\,;h & \ldots \phi \text{ natural} \\ = & \phi_N\,;\hat{F}.((\llbracket \mathsf{cur}.g, \mathsf{cur}.y \rrbracket) \otimes id_A)\,;\hat{F}.\mathsf{ap}\,;h & \ldots id \text{ map, Lem. } 6, \text{ defn. } f \\ = & \phi_N\,;\hat{F}.f\,;h & \end{array}$$

So f solves (7) and (8). It remains to show that if x is a solution then $f \sqsupseteq x$, to which end we observe

$$\begin{array}{ll} & x \text{ satisfies (7) and (8)} & \\ \Rrightarrow & (\llbracket \mathsf{cur}.g, \mathsf{cur}.y \rrbracket) \sqsupseteq \mathsf{cur}.x & \ldots \text{see below} \\ \Rrightarrow & ((\llbracket \mathsf{cur}.g, \mathsf{cur}.y \rrbracket) \otimes id_A)\,;\mathsf{ap} \sqsupseteq (\mathsf{cur}.x \otimes id_A)\,;\mathsf{ap} & \ldots \otimes \text{ and } ; \text{ mono.} \\ \equiv & f \sqsupseteq x & \ldots (1), \text{ defn. } f \end{array}$$

The first step follows by Lemma 7 from

$$\begin{array}{ll} & (7) & \\ \Rrightarrow & \mathsf{cur}.((\mathsf{s} \otimes id_A)\,;x) = \mathsf{cur}.g & \ldots \text{Leibniz} \\ \Rrightarrow & \mathsf{s}\,;\mathsf{cur}.x = \mathsf{cur}.g & \ldots (3),\ \text{Lemma } 4 \end{array}$$

and

$$
\begin{array}{ll}
 & (8) \\
\Rrightarrow & \\
 & \mathsf{cur}.((\mathsf{z} \otimes id_A)\,;x) = \mathsf{cur}.(\phi_N\,;\hat{F}.x\,;h) \qquad \ldots \text{Leibniz} \\
\equiv & \\
 & \mathsf{z}\,;\mathsf{cur}.x = \mathsf{cur}.(\phi_N\,;\hat{F}.x\,;h) \qquad \ldots \ (3), \text{Lemma 4} \\
\Rrightarrow & \\
 & \mathsf{z}\,;\mathsf{cur}.x \ \sqsubseteq\ \hat{F}.(\mathsf{cur}.x)\,;\mathsf{cur}.y \qquad \ldots \text{below}
\end{array}
$$

In the step just above we used

$$
\begin{array}{ll}
 & \hat{F}.(\mathsf{cur}.x)\,;\mathsf{cur}.y \\
= & \qquad \ldots \text{defn. } y \\
 & \hat{F}.(\mathsf{cur}.x)\,;\mathsf{cur}.(\phi_{A\rightsquigarrow B}\,;\hat{F}.\mathsf{ap}\,;h) \\
\sqsupseteq & \qquad \ldots \ (2), (4), \text{Lemma 6} \\
 & \mathsf{cur}.((\hat{F}.(\mathsf{cur}.x) \otimes id)\,;\phi_{A\rightsquigarrow B}\,;\hat{F}.\mathsf{ap}\,;h) \\
= & \qquad \ldots \phi \text{ nat. on } \mathsf{cur}.x \\
 & \mathsf{cur}.(\phi_N\,;\hat{F}.(\mathsf{cur}.x \otimes id)\,;\hat{F}.\mathsf{ap}\,;h) \\
= & \qquad \ldots \text{Lem. 6, (4), } \otimes \text{ pres. maps} \\
 & \mathsf{cur}.(\phi_N\,;\hat{F}.((\mathsf{cur}.x \otimes id)\,;\mathsf{ap})\,;h) \\
= & \qquad \ldots (1) \\
 & \mathsf{cur}.(\phi_N\,;\hat{F}.x\,;h)
\end{array}
$$

∎

In the special case that $\hat{F}$ is the identity functor (and ϕ the identity transformation), it is easily shown that $\hat{F}.(\mathsf{cur}.x)\,;\mathsf{cur}.y = \mathsf{cur}.(\phi_N\,;\hat{F}.x\,;h)$ whence it follows —using Corollary 3 instead of Lemma 7— that f is the unique solution of (7) and (8).

Acknowledgements

Oege de Moor suggested the use of the weak exponent to obtain some generalization of Lawvere's theorem. The referees' suggestions led to significant improvements in presentation. Thanks to Paul Taylor for Latex diagram macros. The research was partly supported by a Cullen Foundation grant from Southwestern University for travel to Oxford.

References

[1] R. Back. Correctness preserving program refinements: Proof theory and applications. Technical Report Tract 131, CWI, 1980.

[2] R. Backhouse, P. de Bruin, G. Malcolm, T. Voermans, and J. van der Woude. Relational catamorphisms. In *IFIP TC2/WG2.1 Working Conference on Constructing Programs*, pages 287–318. Elsevier, 1991.

[3] M. Barr and C. Wells. *Category Theory for Computing Science*. Prentice Hall, 1990.

[4] R. Betti and A. J. Power. On local adjointness of distributive bicategories. *Bolletino U.M.I.*, 7:931–947, 1988.

[5] R. S. Bird. Lectures on constructive functional programming. In M. Broy, editor, *Constructive Methods in Computing Science*, pages 151–216. Springer-Verlag, 1989.

[6] A. Carboni, G. M. Kelly, and R. J. Wood. A 2-categorical approach to geometric morphisms, I. Technical Report 89-19, ISSN 1033–2359, Department of Pure Mathematics, The University of Sydney, NSW 2006, Australia, 1989.

[7] O. de Moor. Inductive data types for predicate transformers. *Inf. Process. Lett.*, 43(3):113–118, 1992.

[8] E. W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.

[9] P. J. Freyd and A. Scedrov. *Categories, Allegories*. North-Holland, 1990.

[10] W. Hesselink. *Programs, Recursion, and Unbounded Choice*. Cambridge, 1993.

[11] A. Kaldewaij. *Programming: the Derivation of Algorithms*. Prentice-Hall, 1990.

[12] C. Martin, C. A. R. Hoare, and J. He. Pre-adjunctions in order enriched categories. *Mathematical Structures in Computer Science*, 1:141–158, 1991.

[13] C. E. Martin. Preordered categories and predicate transformers. Dissertation, Oxford University, 1991.

[14] C. Morgan. *Programming from Specifications, second edition*. Prentice Hall, 1994.

[15] D. A. Naumann. Two-categories and program structure: Data types, refinement calculi, and predicate transformers. Dissertation, University of Texas at Austin, 1992.

[16] D. A. Naumann. Predicate transformers and higher order programs. *Theoretical Comput. Sci.*, 150:111–159, 1995.