Homomorphisms, Factorisation and Promotion

Nico Verwer

RUU-CS-90-5
February 1990

Utrecht University

Department of Computer Science

Padualaan 14, P.O. Box 80.089,
3508 TB Utrecht, The Netherlands,
Tel. : ... + 31 - 30 - 531454

Homomorphisms, Factorisation and Promotion

Nico Verwer [*]

Rijksuniversiteit Utrecht

February [illegible] 1990

1 Introduction

The category theoretical description of abstract data types provides a general and powerful description of abstract data types. One of the first people to note this was Hagino [3], who shows that his type functors are a generalisation of [illegible] and (co)algebras.

Recently, category theory has been used to derive laws within the *Bird-Meertens formalism* [illegible]. In two papers [4, 5], Grant Malcolm has [illegible] general theorems of the Bird-Meertens formalism with the use of [illegible] types.

In my view, the notation that Malcolm uses is a bit obscure. In this paper I use a more concise notation to show the essence of the factorisation theorem more clearly. I found that proving the factorisation theorem is now only a simple diagram chasing exercise. I also found that I had to generalise the notion of reductions, which [illegible] define for snoc lists and factorability. It appears that factorability can be seen as a property of homomorphisms, not of type-functors.

Together with Johan Jeuring, I found that the new notation can also be used to express the promotion theorem more clearly. We also made a link with the [illegible] promotion law which is used in squiggol.

A remark on the presentation: Some people like to draw diagrams instead of writing down formulas, whereas others think that pictures give a false sense of understanding and can not give an exact description. Personally I like diagrams as an illustration of formulas, and I think they can help in understanding formulas. Therefore, I shall use them throughout this note, but I shall not rely on them for proofs. The diagrams were typeset with Francis Borceux's diagram package [2], which saved me a lot of time and work.

[*] e-mail: [illegible]

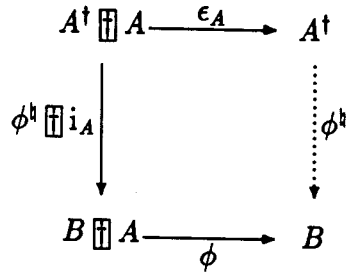
ISSN:0924-3275

which is a *homomorphism* (respects the structure):

$$\phi^\natural \circ \epsilon_A = \phi \circ (\phi^\natural \boxplus \mathrm{i}_A).$$

□

This is shown in the following commuting diagram.



The dotted arrow indicates that it is the *unique* function which makes the diagram commute.

The constructor ϵ is an isomorphism, with inverse $(\epsilon \boxplus \mathrm{i})^\natural$ (see for instance [7]). Because of this isomorphism, $A^\dagger$ can be seen as a fixed point of $(\boxplus A)$:

$$A^\dagger \boxplus A \cong A^\dagger.$$

For this reason, some people write $A^\dagger$ as $(\mu X \,.\, X \boxplus A)$. The function

$$\epsilon_A^{-1} = (\epsilon_A \boxplus \mathrm{i}_A)^\natural : A^\dagger \to A^\dagger \boxplus A$$

splits a $A^\dagger$-term in its components. It is the 'pattern-matching function', which is used implicitly in functional programming languages to do case-analysis on the construction of a term. This provides a recursive definition of $\phi^\natural$ (which is easily established from the commuting diagram):

$$\phi^\natural = \phi \circ (\phi^\natural \boxplus \mathrm{i}_A) \circ (\epsilon_A \boxplus \mathrm{i}_A)^\natural.$$

Malcolm [4, 5] would write $(\!|\phi|\!)$ instead of $\phi^\natural$, and $(\otimes_0, \ldots, \otimes_n)$ for a components-functor $\boxplus = \lambda X \lambda A \,.\, (X \otimes_0 A + \ldots + X \otimes_n A)$. He also sometimes writes $(F_1, \ldots, F_n)$ for $(\boxplus A)$. Although in many cases the components-type is indeed a disjoint union of types, we think that it is not necessary to indicate this, and we rather have one components-functor.

Example 3. An example of a type-functor is the cons-list constructor $*$, with

$$B \boxtimes A = 1 + (A \times B) \quad , \quad g \boxtimes f = \mathrm{i}_1 + (f \times g)$$

$$\epsilon_A = [\Box, >\!+] : A^* \boxtimes A \to A^*.$$

In this case, the commuting diagram says that for all types A, B and functions $[c, \oplus] : \mathbf{1} + (A \times B) \to B$:

$$[c, \oplus]^\natural \circ [\Box, \succ\!+] = [c, \oplus] \circ (\mathrm{i}_1 + (\mathrm{i}_A \times [c, \oplus]^\natural))$$

or, equivalently:

$$[c, \oplus]^\natural \circ \Box = c \quad , \quad [c, \oplus]^\natural \circ (a \succ\!+ x) = a \oplus ([c, \oplus]^\natural x).$$

This function replaces $\Box$ by c and $\succ\!+$ by $\oplus$.

The inverse of $[\Box, \succ\!+]$ is $([\Box, \succ\!+] \boxplus \mathrm{i}_A)^\natural$ which splits up a list in its head and tail:

$$([\Box, \succ\!+] \boxplus \mathrm{i}_A)^\natural \circ [\Box, \succ\!+] = \mathrm{i}_1 + (\mathrm{i}_A \times \mathrm{i}_{A^*})$$

or

$$([\Box, \succ\!+] \boxplus \mathrm{i}_A)^\natural \circ \Box = \mathrm{i}_1 \quad , \quad ([\Box, \succ\!+] \boxplus \mathrm{i}_A)^\natural \circ (a \succ\!+ x) = (a, x).$$

□

Example 4. Another type-functor is the non-empty join-list constructor $*$, with

$$B \boxplus A = A + (B \times B) \quad , \quad g \boxplus f = f + (g \times g)$$

$$\epsilon_A = [[\cdot], +\!\!+].$$

The reader is encouraged to draw the corresponding diagram, and investigate its meaning. (We have not required $+\!\!+$ to be associative, so we really have specified binary trees.)

□

3 Maps

A map is the part of a type-functor that works on functions. In general we have, for a type-functor $\dagger$ and a function $f : A \to B$, a $\dagger$-mapped function:

$$f^\dagger : A^\dagger \to B^\dagger.$$

Functors preserve identity and composition:

$$(\mathrm{i}_A)^\dagger = \mathrm{i}_{A^\dagger} \quad , \quad (g \circ f)^\dagger = g^\dagger \circ f^\dagger.$$

The idea is that a mapped function only works on the A-elements of a $A^\dagger$-term, leaving the structure unchanged.

We can define maps as homomorphisms. In order to do so, we try to find a function

$$\phi : B^\dagger \boxplus A \to B^\dagger$$

such that

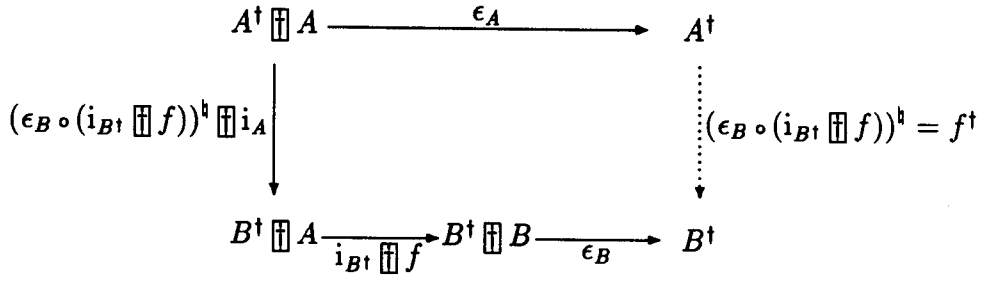
$$\phi^\natural = f^\dagger : A^\dagger \to B^\dagger.$$

We can do this by first applying f to the A-elements of the $(B^\dagger \boxplus A)$-term, giving a term in $B^\dagger \boxplus B$. Then we embed this in a $B^\dagger$-structure by applying the constructors ϵ_B.

Definition 5. The map corresponding to a type-functor $\dagger$ is defined for all functions $f : A \to B$ as

$$f^\dagger = (\epsilon_B \circ (\mathrm{i}_{B^\dagger} \boxplus f))^\natural.$$

This is illustrated in the following commuting diagram.



□

This definition is exactly the same as the one in Malcolm's paper [4], who also proves that maps indeed preserve identity and composition.

Proposition 6.

$$f^\dagger \circ \epsilon_A = \epsilon_B \circ (f^\dagger \boxplus f).$$

Proof. This corresponds exactly to the commuting diagram. Crucial steps are the fact that (bi)functors preserve composition,

$$(h \boxplus k) \circ (p \boxplus q) = (h \circ p) \boxplus (k \circ q)$$

the identity laws and the definition of $f^\dagger$:

$$(f^\dagger \boxplus \mathrm{i}_A) \circ (\mathrm{i}_{B^\dagger} \boxplus f) = f^\dagger \boxplus f.$$

□

This proposition shows how our definition of maps corresponds to the usual definition in the Bird-Meertens formalism. For example, in the case of lists it becomes

$$f^\dagger \circ \Box = \Box \quad , \quad f^\dagger \circ \succ\!+ = \succ\!+ \circ (f \times f^\dagger).$$

4 Reductions

On cons-lists, we define reductions $\oplus\not\leftarrow_e : A^* \rightarrow A$ as

$$(\oplus\not\leftarrow_e)\square = e \quad , \quad (\oplus\not\leftarrow_e)(a \mathrel{>\!\!+} x) = a \oplus ((\oplus\not\leftarrow_e)x)$$

for $\oplus : A \times A \rightarrow A$ and $e : A$. This is slighty different from the usual definition, where $\oplus : A \times B \rightarrow B$ (see the note below). A reduction is primarily a function on the structure, not on the elements of a $A^\dagger$-term. (One might argue that the function $+\not\leftarrow_0 : N^* \rightarrow N$ does affect the integer elements in the list, but this is really a consequence of equations that hold in the integer domain.)

We can define reductions as homomorphisms, just as we did for maps.

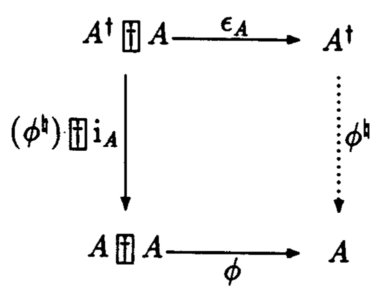
Definition 7. A $A^\dagger$-reduction is defined for functions

$$\phi : A \boxed{\dagger} A \rightarrow A$$

as

$$\phi^\natural : A^\dagger \rightarrow A.$$

This is illustrated by the diagram below.



$\square$

Reductions are usually written as $\oplus\not\leftarrow_e$ for $\phi = [e, \oplus]$ (for cons-lists), or another notation considered appropriate.

Example 8. On non-empty join-lists, reductions are defined for functions

$$[f, \oplus] : B + (B \times B) \rightarrow B.$$

In the special case that $f = \mathrm{i}_B$, we write $\oplus/$ for the reduction $[\mathrm{i}_B, \oplus]^\natural$. We shall use this notation for other data types to emphasize the fact that some homomorphism is a reduction. The above diagram gives the recursive definition of $\oplus/$. $\square$

Note that in the above definition we do not require $\dagger$ to be *factorable*, like Malcolm [4] does. Thus reductions over cons- or snoc-lists can be defined in the usual way. For instance, the reduction which is normally written as $\oplus\not\leftarrow_e$ is exactly

the same as $[e, \oplus]^\natural$ In this case, the commuting diagram from the definition above amounts to the definition of $\oplus \not\leftarrow_e$ given earlier.

In the literature on constructive functional programming, reductions on cons-lists are often defined for operators $\oplus : A \times B \to B$. Although this is more general, we chose not to do so, because then reductions would be exactly the same as homomorphisms. We feel that reductions like they are defined here can be very useful, because on the elements of a structure $A^\dagger$ they act as a function from A to A. In Malcolm's paper, reductions act as the identity function on elements. We had to mention this property explicitly in the join-list example above. Reductions in the sense of Malcolm [4] are also reductions according to our definition.

5 Factorisation

It is well known [1] that homomorphisms on lists can be factored into a map followed by a reduction. In his paper, Malcolm [4] shows that homomorphisms on *factorable* type-functors can be factored this way. His definition of factorable requires $\boxplus$ to have a special form, as in the following definition.

Definition 9. (Malcolm) A type functor $\dagger$ is *factorable* if the corresponding components-functor has the form

$$X \boxplus A = A + X^F$$

where A does not occur in X^F. Its constructor functions must have the form

$$[f, g] : A + A^{\dagger F} \to A^\dagger.$$

□

This means, for instance, that join-lists are factorable, but cons-lists are not.

In the previous section we defined reductions for *all* type functors. Still it is not possible to factor every homomorphism we can think of into a map followed by a reduction. Therefore, we need a more subtle definition of factorability. We define the factorability of homomorphisms as a property of the homomorphisms themselves, not of the type functor for which they are defined.

Definition 10. A homomorphism $\phi^\natural : A^\dagger \to C$ is *factorable* if the function $\phi : C \boxplus A \to C$ can be written as

$$\phi = \oplus \circ (\mathrm{i}_C \boxplus f)$$

for some

$$\oplus : C \boxplus B \to C \quad , \quad f : A \to B.$$

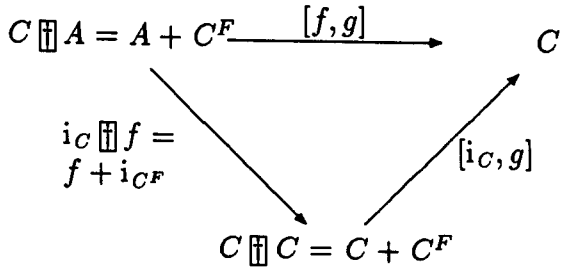
□

Proposition 11. For a type functor which is factorable in the sense of definition 9, every homomorphism is factorable in the sense of defintion 10.

Proof. Consider a homomorphism $\phi^\natural : A^\dagger \to C$, where $\dagger$ is factorable (according to definition 9), i.e. $X \boxplus A = A + X^F$. Then $\phi = [f, g] : A + C^F \to C$. Now because

$$\begin{aligned} [f,g] &= [\mathrm{i}_C, g] \circ (f + \mathrm{i}_{C^F}) \\ &= [\mathrm{i}_C, g] \circ (\mathrm{i}_C \boxplus f) \end{aligned}$$

ϕ is factorable according to our definition, as shown in the diagram.



□

We can now formulate the factorisation theorem, which says that factorable homomorphisms can be factored into a map followed by a homomorphism:

Proposition 12. If $\phi^\natural$ is factorable and $\phi = \oplus \circ (\mathrm{i}_C \boxplus f)$, where $\oplus : C \boxplus B \to C$ and $f : A \to B$, then

$$\phi^\natural = \oplus^\natural \circ f^\dagger.$$

Proof. We first prove:

$$\begin{array}{lcl} \oplus^\natural \circ f^\dagger \circ \epsilon_A & = & \text{(map diagram)} \\ \oplus^\natural \circ \epsilon_B \circ (\mathrm{i}_{B\dagger} \boxplus f) \circ (f^\dagger \boxplus \mathrm{i}_A) & = & \text{(reduction diagram)} \\ \oplus \circ (\oplus^\natural \boxplus \mathrm{i}_B) \circ (\mathrm{i}_{B\dagger} \boxplus f) \circ (f^\dagger \boxplus \mathrm{i}_A) & = & \text{(functors preserve composition, identity laws)} \\ \oplus \circ (\mathrm{i}_C \boxplus f) \circ (\oplus^\natural \boxplus \mathrm{i}_A) \circ (f^\dagger \boxplus \mathrm{i}_A) & = & \text{(functors preserve composition, definition of } \phi) \\ \phi \circ ((\oplus^\natural \circ f^\dagger) \boxplus \mathrm{i}_A). & & \end{array}$$

By definition, we also know that $\phi^\natural = (\oplus \circ (\mathrm{i}_C \boxplus f))^\natural$ is the *unique* function which satisfies

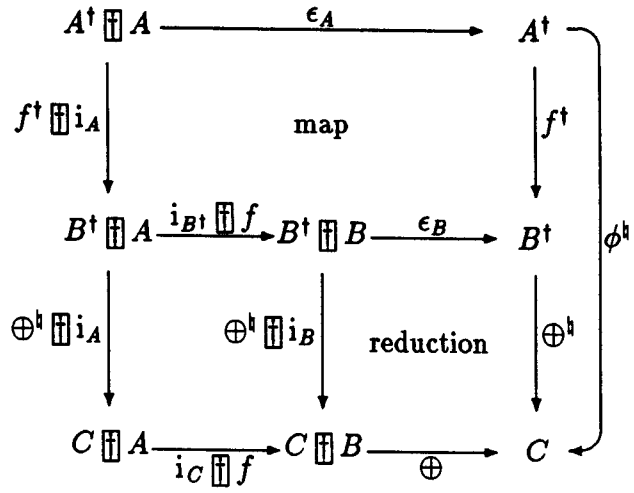
$$\phi^\natural \circ \epsilon_A = \phi \circ (\phi^\natural \boxplus \mathrm{i}_A).$$

Since $\oplus^\natural \circ f^\dagger$ has the same property, we conclude that

$$\phi^\natural = \oplus^\natural \circ f^\dagger.$$

□

The proof is illustrated in the following diagram.



From the proof of proposition 11, it is easy to see that in the case of a factorable type functor, the homomorphism can be factored into a map followed by a *reduction* (then the types B and C are equal).

6 Promotion

A very important theorem is the promotion theorem given by Malcolm in [5]. In our notation it reads:

Proposition 13. Let $\phi : B \amalg A \to B$, $\psi : C \amalg A \to C$ and $f : B \to C$. If

$$f \circ \phi = \psi \circ (f \amalg \mathrm{i}_A)$$

(f is $\phi \to \psi$-promotable), then

$$\psi^\natural = f \circ \phi^\natural.$$

Proof.

$$\begin{array}{lcl} (f \circ \phi^\natural) \circ \epsilon_A & = & \text{(def. of } \phi^\natural) \\ f \circ \phi \circ (\phi^\natural \amalg \mathrm{i}_A) & = & \text{(promotability-assumption)} \\ \psi \circ (f \amalg \mathrm{i}_A) \circ (\phi^\natural \amalg \mathrm{i}_A) & = & \text{(functors preserve composition)} \\ \psi \circ ((f \circ \phi^\natural) \amalg \mathrm{i}_A) & & \end{array}$$

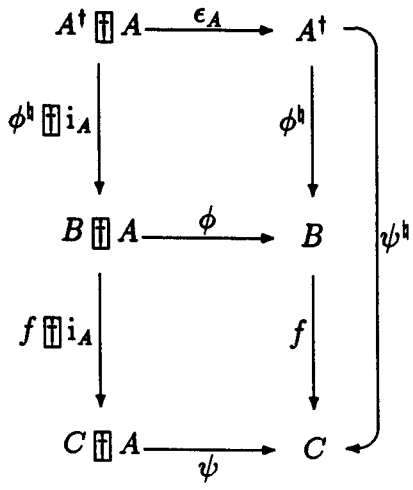
Since $\psi^\natural$ is the unique function with the property

$$\psi^\natural \circ \epsilon_A = \psi \circ (\psi^\natural \amalg \mathrm{i}_A)$$

we conclude that

$$\psi^\natural = f \circ \phi^\natural.$$

□



A special case arises when $\phi = \oplus : A \amalg A \to A$, (then $\phi^\natural$ is a reduction), and $\psi^\natural$ is factorable as

$$\psi = \otimes \circ (\mathrm{i}_C \amalg f).$$

Then the promotion theorem becomes:

Proposition 14. If

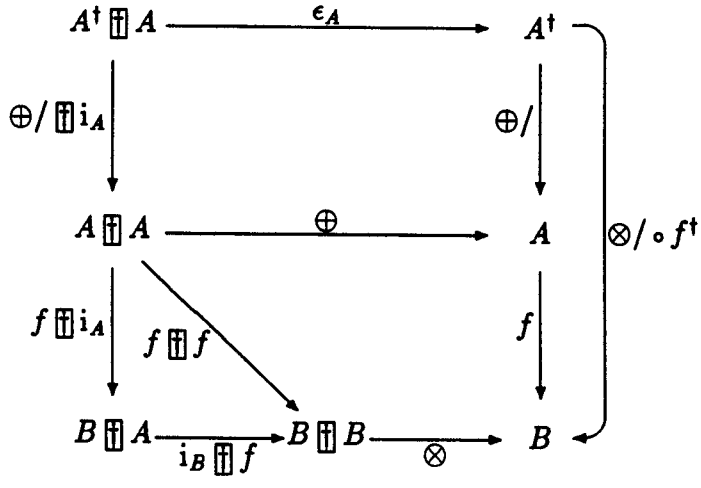
$$f \circ \oplus = \otimes \circ (f \amalg f)$$

(f is $\oplus \to \otimes$-promotable), then

$$\otimes/ \circ f^\dagger = f \circ \oplus/.$$

Proof. Because of the simplifying assumptions, $\phi^\natural$ may be written as $\oplus/$, and by the factorisation theorem $\psi^\natural = \otimes/ \circ f^\dagger$. Substituting this in the general promotion theorem then gives the special one. □

This is illustrated in the figure below.



Example 15. In the case of non-empty join-lists, the last proposition is the well-known law for list-promotion. If we substitute $[\mathrm{i}_A, \oplus]$ for $\oplus$, and $[\mathrm{i}_B, \otimes]$ for $\otimes$, the promotability-condition becomes

$$f \circ \oplus = \otimes \circ (f \times f)$$

and we then have

$$f \circ \oplus / = \otimes / \circ f^*$$

where $\oplus/$, $\otimes/$ are defined as in the earlier example. □

References

[1] Richard S. Bird, *Lectures on constructive functional programming*, in *Constructive Methods in Computing Science*, NATO ASI F55, Springer-Verlag, 1989.

[2] Francis Borceux, *User's guide for the diagram macro's*, UCL, Louvain-la-Neuve, Belgium. (this macro package can be obtained via FTP from `praxis.cs.ruu.nl`, 131.211.80.6.)

[3] Tatsuya Hagino, *Category Theoretic Approach to Data Types*, Thesis, University of Edinburgh, 1987.

[4] Grant Malcolm, *Factoring Homomorphisms*, Technical Report Computing Science Notes CS8909, Dept. of Mathematics and Computing Science, University of Groningen, 1989.

[5] Grant Malcolm, *Homomorphisms and Promotability*, in *Mathematics of Program Construction*, LNCS 375, Springer-Verlag 1989.

[6] Lambert Meertens, *Algorithmics — towards programming as a mathematical activity*, in *Proceedings CWI symposium on Mathematics and Computer Science*, CWI Monographs vol. 1, North-Holland, 1986.

[7] G.C. Wraith, *A note on categorical data types*, in *Category theory and computer science 1989*, LNCS 389, Springer-Verlag 1989.