# An Exact Interpretation of **while**

Paul Taylor
Department of Computing,
Imperial College,
London SW7 2BZ
+44 71 589 5111 *ext.* 5057
<pt@doc.ic.ac.uk>

August 12, 2003

**Abstract**

The behaviour and interaction of finite limits (products, pullbacks and equalisers) and colimits (coproducts and coequalisers) in the category of sets is illustrated in a "hands on" way by giving the interpretation of a simple imperative language in terms of partial functions between sets of states. We show that the interpretation is a least fixed point and satisfies the usual proof rule for loop invariants.

This paper is to appear in *Theory and Formal Methods 1993: Proceedings of the First Imperial College, Department of Computing, Workshop on Theory and Formal Methods* editted by G.L. Burn and S.J. Gay and M.D. Ryan, to be published by Springer Verlag (Workshops in Computer Science).

## 1 The language

We shall interpret a language with the following ⟨command⟩s in a category:

**skip**
| ⟨command⟩ **;** ⟨command⟩
| **let** ⟨variable⟩ = ⟨expression⟩ : ⟨type⟩
| **discard** ⟨variable⟩
| ⟨variable⟩ := ⟨expression⟩
| **if** ⟨Boolean expression⟩ **then** ⟨command⟩ **else** ⟨command⟩ **fi**
| **while** ⟨Boolean expression⟩ **do** ⟨command⟩ **od**

Clearly **skip** and sequencing (**;**) correspond to identity and composition, so each command is a *morphism*, but what are the *objects* which serve as its source and target?

Each command of a program has a "compile-time" as well as its run-time effect: here just the declaration (**let**) and un-declaration (**discard**) of variables. Before the command (in the text) there is a certain collection $\Gamma$ of variables which are **in scope**, so in particular they may be used in forming expressions. At the end the scope, $\Delta$, consists of $\Gamma$, *plus* the newly declared variable(s), *minus* those which have been discarded. $\Gamma$ and $\Delta$ are the source and target respectively of the morphism representing the command; by construction the source of one command is the target of the previous one, as it must be for composition to be defined.

We do not require scopes to be nested, instead adopting the rule that if a variable name is in scope then it may not be the subject of a definition. This apparently weak convention is convenient because it allows assignment to be defined in terms of the other commands:

$$x := a \qquad \text{is the same as} \qquad \textbf{let } x' = a \textbf{ ; discard } x \textbf{ ; let } x = x' \textbf{ ; discard } x'$$

where $x'$ is a new variable, *i.e.* not in scope. In programming **discard** is, apart from this, redundant (provably so in a certain sense in our formal language) but it has been added for an exact match with the categorical concept (products) which is being used to interpret the language.

Abstractly we have already said enough about the objects: they are lists of variables. (Similarly we have also given the morphisms as programs, but we shall see that we must impose an equivalence relation on them.) Variables are **typed**: we need the **Boolean type 2** (with its elements $\mathsf{yes}$ and $\mathsf{no}$), but shall not introduce any (other) type constructors.

Concretely, a type is the **set** of values which may be taken by a variable to which it is attributed. The **state** of the machine is determined by the **tuple** of current values of the variables in scope, and so the object $[\![\Gamma]\!]$ representing the scope $\Gamma$ is the **product** of (the sets interpreting) the types. In particular,

$[\![\textbf{discard } x]\!]$ is the **product projection** which omits $[\![x : X]\!]$.

and we write $\hat{x} : [\![\Gamma]\!] \times X \longrightarrow\!\!\triangleright\, [\![\Gamma]\!]$ for it.

To say what **expressions** are, we need a language of algebraic operations.

$$\begin{array}{rcl} \langle \text{expression} \rangle & ::= & \langle \text{variable} \rangle \\ & | & \langle \text{constant} \rangle \\ & | & \langle \text{operator} \rangle (\langle \text{expression} \rangle, ..., \langle \text{expression} \rangle) \end{array}$$

The categorical interpretation of algebra is very well known. Each $\langle \text{operator} \rangle$ $\mathsf{f}$ and $\langle \text{constant} \rangle$ $\mathsf{c}$ has an intended meaning as a morphism $f : Y_1 \times \cdots \times Y_m \to Z$ or $c : \mathbf{1} \to Z$ in the category, $\vec{Y}$ and $Z$ being the argument and result types.

In the scope $\Gamma \equiv [x_1 : X_1, \ldots, x_n : X_n]$, expressions are interpreted as

$$\begin{array}{lll} [\![x_i]\!] & : [\![\Gamma]\!] = X_1 \times \cdots \times X_n \xrightarrow{\pi_i}\!\!\triangleright\, X_i \\ [\![\mathsf{c}]\!] & : [\![\Gamma]\!] \xrightarrow{!} \mathbf{1} \xrightarrow{c} Y \\ [\![\mathsf{f}(e_1, \ldots, e_m)]\!] & : [\![\Gamma]\!] \xrightarrow{\langle [\![e_1]\!], \ldots, [\![e_m]\!] \rangle} Y_1 \times \cdots \times Y_m \xrightarrow{f} Y \end{array}$$

by structural induction, where the $[\![e_i]\!] : [\![\Gamma]\!] \to Y_i$ interpret the subexpressions.

$$\begin{array}{lll} [\![\textbf{let } y = e : Y]\!] & : [\![\Gamma]\!] \xrightarrow{\langle \mathsf{id}, [\![e]\!] \rangle} [\![\Gamma]\!] \times Y \\ [\![\textbf{discard } x_i]\!] & : [\![\Gamma]\!] \xrightarrow{\widehat{x_i}}\!\!\triangleright\, X_1 \times \cdots \times X_{i-1} \times X_{i+1} \times \cdots \times X_n \\ [\![x_i := e]\!] & : [\![\Gamma]\!] \xrightarrow{\langle \pi_1, \ldots, \pi_{i-1}, [\![e]\!], \pi_{i+1}, \ldots \pi_n \rangle} [\![\Gamma]\!] \end{array}$$

similarly interpret the declaration and un-declaration commands.

The following equations may be proved by structural induction

$$\begin{array}{lcl} [\![\textbf{let } x = a \textbf{ ; discard } x]\!] & = & \mathsf{id} \\ [\![\textbf{let } x = a \textbf{ ; let } y = c]\!] & = & [\![\textbf{let } y = c^{[x:=a]} \textbf{ ; let } x = a]\!] \\ [\![\textbf{discard } x \textbf{ ; let } y = b]\!] & = & [\![\textbf{let } y = b \textbf{ ; discard } x]\!] \\ [\![\textbf{discard } x \textbf{ ; discard } y]\!] & = & [\![\textbf{discard } y \textbf{ ; discard } x]\!] \end{array}$$

where $x$ does not occur in $b$ (it may in $c$), nor $y$ in $a$.

The meta-notation $c^{[x:=a]}$ means the substitution of $a$ for $x$ in $c$. It is defined in the usual way, but is in fact much simpler than in the $\lambda$-calculus because there is no variable binding or renaming. The equation which uses this may be read in either direction: from left to right, to give the fully substituted version or closed form, or from right to left, to reduce each expression to a sequence of single operations or subroutine calls.

The foregoing description may be read directly as an interpretation of the language in a category (such as **Set**) with products. However it may also be taken as an *abstract* definition of a category out of the language: the objects are lists of typed variables and the morphisms are programs subject to the equations given. This category itself has products, and the interpretation is a product-preserving functor from it to the desired semantic category.

# 2 Interpretation on subsets

So far we have only allowed the objects in the semantic category to be *products* of sets, but it is convenient to extend the interpretation to subsets. In an imperative understanding the notation

$$\{\phi\}\, P\, \{\psi\}$$

means "if the program $P$ is run from an initial state satisfying $\phi$ then when (if) it terminates $\psi$ will hold." In the case of the whole program, $\phi$ and $\psi$ constitute the **specification**. Of course we may always take $\phi = \bot$ and $\psi = \top$, but this vacuous specification says that the program is good for nothing.

The predicates $\phi$ and $\psi$ do not need to be computable: even when they are it would often be more complicated to compute them than the program itself, and sometimes they involve universal quantification over infinite sets.

We can express the property categorically by saying that in the square

$$\begin{array}{ccc} \{\vec{x} : \phi(\vec{x})\} & \dashrightarrow & \{\vec{y} : \psi(\vec{y})\} \\ \downarrow & & \downarrow \\ \left[\!\left[\vec{x} : \vec{X}\right]\!\right] & \xrightarrow{[\![P]\!]} & \left[\!\left[\vec{y} : \vec{Y}\right]\!\right] \end{array}$$

there is some map at the top making it commute; it is unique since the right hand map is an inclusion.

There are corresponding logical rules:

$$\{\phi(\vec{x})\}\, \mathbf{discard}\ y\, \{\phi(\vec{x})\} \qquad \{\psi\}\, \mathbf{skip}\, \{\psi\} \qquad \{\psi(\vec{x}, a)\}\, \mathbf{let}\ y = a\, \{\psi(\vec{x}, y)\}$$

where $y$ does not occur (freely) in $\phi$, and

$$\frac{\{\phi\}\, P\, \{\psi\} \qquad \{\psi\}\, Q\, \{\chi\}}{\{\phi\}\, P\, ;\, Q\, \{\chi\}} \qquad\qquad \frac{\phi' \vdash \phi \qquad \{\phi\}\, P\, \{\psi\} \qquad \psi \vdash \psi'}{\{\phi'\}\, P\, \{\psi'\}}$$

It is more idiomatic to prove a program by inserting **mid-conditions** between the lines than by this repetitive sequent-style notation. It is then a **natural deduction** with definitions but no temporary hypotheses.

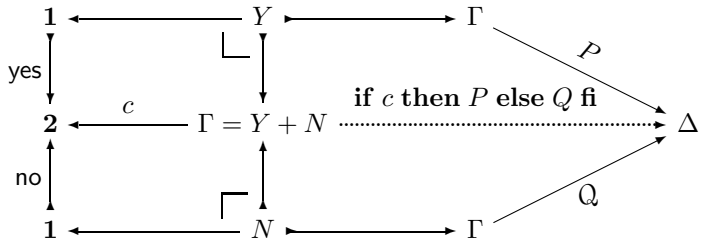
Each command $P$ and post-condition $\psi$ have a **weakest precondition** $P^*\psi$ for which the property is valid, indeed we have given $\{P^*\psi\}\, P\, \{\psi\}$ above. Categorically, the square is then a **pullback**, satisfying a universal property similar to that for a product. We shall only need this special case, called an **inverse image**, in which two of the sides are inclusions.

# 3 Conditionals

Now we shall add the **if then else fi** construction to the language. The use of the word "**if**" is misleading: The test is not of a general predicate as in the last section, but a computable function with two possible terminating values, called yes and no to make the distinction from logic clear. Nor does **if** mean implication: in fact it is related to disjunction.

For correct scoping, the two branches must end up with the same variables; in particular any that only one declares must be discarded.

To interpret the conditional we first evaluate the test, and restrict the interpretations of the two branches to the subsets on which it succeeds and fails.




First consider $\Gamma$, $c$, $Y$ and $N$ fixed, the rest variable. The dotted map always exists and is to be determined uniquely by the pair $Y \to \Delta \leftarrow N$, so $\Gamma$ is the **coproduct** $Y + N$. By definition this satisfies the universal property we have just given: the same as that for the product but with the arrows reversed.

The **boolean type** $\mathbf{2} = \mathbf{1} + \mathbf{1}$ is, as we shall see, the primitive case. The two structural maps from the terminal object are the constants yes and no.

However many categories have coproducts which are quite unsuitable for interpreting conditionals. For example, they coincide with products in the category of finite dimensional vector spaces, where a cancellation law also holds; this means that the **distributive law**

$$A \times (Y + N) \cong (A \times Y) + (A \times N)$$

is valid only when one of the summands on the right vanishes. This equation is needed so that the interpretation of a conditional using $Y + N$ is unaffected by the presence of "passive" variables $A$. In fact we require a stronger property.

The condition $c$ gives a partition $\Gamma = Y + N$, but it may be recovered from it by considering the command

$$\textbf{if } c \textbf{ then let } x = \mathsf{yes} \textbf{ else let } x = \mathsf{no} \textbf{ fi ; discard } \Gamma$$

Hence the commutation of the two squares on the left must define a bijection

- between morphisms $X \to \mathbf{2}$, where $\mathbf{2}$ is the coproduct $\mathbf{1} + \mathbf{1}$
- and coproduct diagrams $Y \to X \leftarrow N$.

More precisely, these provide the objects of two categories (whose morphisms are appropriate commutative triangles) which are to be equivalent [3].

By choosing appropriate collections of squares it is not difficult to show that

- the two squares are pullbacks, indeed inverse images,
- the structure maps $Y \to Y + N$ and $N \to Y + N$ are (regular) monos,
- (**stable**) the pullback of any coproduct diagram is another such,
- in particular the distributive law holds,

- there is a **strict initial object 0** (the empty set): *i.e.* any map $X \to \mathbf{0}$ is an isomorphism; equivalently $X \times \mathbf{0} \cong \mathbf{0}$ for any object $X$, and
- the components of the coproduct are **disjoint**, *i.e.* the pullback (intersection) of the structure maps is **0**.

Specifying **stable disjoint coproducts** and the strict initial object is usual; they imply our (more useful) form of the definition, but this is trickier to show.

**Set** enjoys these properties by of the **tag** construction of the disjoint union:

$$Y + N \cong (Y \times \{\mathsf{yes}\}) \cup (N \times \{\mathsf{no}\})$$

In the interpretation of programs, if a partition $[\![\Gamma]\!] = Y + N$ is classified by a computable function $[\![c]\!] : [\![\Gamma]\!] \to \mathbf{2}$ then this is unique. However it may not exist: there are partitions which are not **decidable**. Such coproducts are no good for defining conditional commands, because "complementary" programs can't be pasted together: we need to know which way to go *first*. The Boolean type $\mathbf{2} = \mathbf{1} + \mathbf{1}$ is primitive in the sense that it does allow this pasting, and its pullbacks also allow it by first reducing to this case.

Finally, the conditional satisfies the proof rule

$$\frac{\{\phi \wedge c\}\, P\, \{\psi\} \qquad \{\phi \wedge \neg c\}\, Q\, \{\psi\}}{\{\phi\}\, \mathbf{if}\ b\ \mathbf{then}\ P\ \mathbf{else}\ Q\ \mathbf{fi}\, \{\psi\}}$$

because of stability: $\{\phi\} = \{\phi \wedge c\} + \{\phi \wedge \neg c\}$.

# 4 Partial functions

Programs in the fragment of the language which we have defined so far always terminate, and so can be interpreted as total functions between sets. This is no longer possible when we add loops: since the Halting Problem is insoluble there is no restriction which we can place on programs in order to guarantee termination without destroying the expressive power of the language.

A **partial function** with **source** $X$, **target** $Y$ and **support** $U$ in a category $\mathcal{S}$ is a diagram of the form

$$\begin{array}{ccc} U & \xrightarrow{f} & Y \\ {\scriptstyle i}\downarrow & & \\ X & & \end{array}$$

where $U \hookrightarrow X$ is mono — a subset. If there is an isomorphism $e : V \cong U$ we regard the pair $\langle e\,;\, i,\ e\,;\, f\rangle$ as the same partial function as $\langle i, f\rangle$. More generally the existence of any map (necessarily a unique mono) making the two triangles commute is taken as an instance of an order relation between partial maps.

Every **total** function (in particular the identity) is also partial: we just put $i = \mathsf{id}_X : X = U \hookrightarrow X$.

Composition is defined by pullback (inverse image):

$$\begin{array}{ccccc} U\,;\,V & \longrightarrow & V & \xrightarrow{g} & Z \\ \downarrow & \lrcorner & {\scriptstyle j}\downarrow & & \\ U & \xrightarrow{f} & Y & & \\ {\scriptstyle i}\downarrow & & & & \\ X & & & & \end{array}$$

and is easily shown to be associative.

In the interpretation using *total* functions, discarding a variable immediately after it is declared with a value has no effect. Now

$$\mathsf{supp}(e) = [\![\textbf{let } x = e \textbf{ ; discard } x]\!]$$

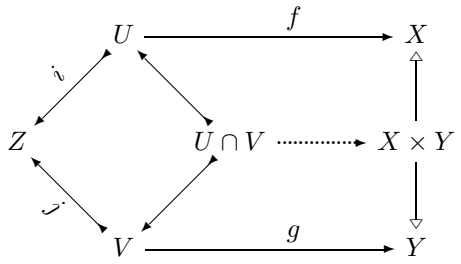
is a partial endofunction $\langle i, i\rangle : [\![\Gamma]\!] \rightharpoonup [\![\Gamma]\!]$ where $\langle i, f\rangle = [\![e]\!]$. It is sometimes convenient to regard this as a subset of $[\![\Gamma]\!]$ instead.

Instead of allowing *all* monos to be the supports of partial maps, one may restrict to a class $\mathcal{M}$ of them, satisfying the following conditions:

- $\mathcal{M}$ contains all isomorphisms,
- $\mathcal{M}$ is closed under pullback against arbitrary maps, and
- $\mathcal{M}$ is closed under composition.

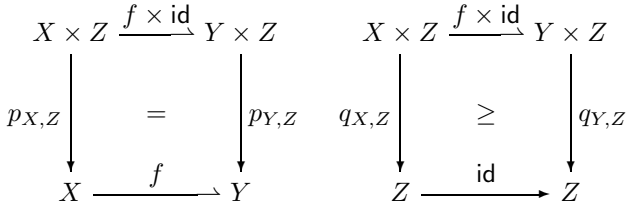
Write $\mathsf{P}(\mathcal{S}, \mathcal{M})$ for the category of partial maps whose supports belong to $\mathcal{M}$ in the category $\mathcal{S}$.

Given partial maps $\langle i, f\rangle : Z \rightharpoonup X$ and $\langle j, g\rangle : Z \rightharpoonup Y$, we must take the intersection of their supports in order to define a partial map into a **product**:




This means that the pairing is **strict**, *i.e.* only defined when both components are. In general, therefore, composition with the projections does not recover the components, so this is not the product in the category of partial functions.

It nevertheless defines a **tensor product**: a functor of two arguments which is coherently associative and commutative. The behaviour of the projections is less natural here than amongst total functions:




The diagonal, $d_X : X \to X \times X$, remains natural, and they all satisfy

$$\begin{array}{lcllcl}
d_X \,;\, p_{X,X} & = & \mathsf{id}_X & = & d_X \,;\, q_{X,X} \\
(\mathsf{id}_X \times p_{Y,Z}) \,;\, p_{X,Y} & = & p_{X,Y\times Z} & = & (\mathsf{id}_X \times q_{Y,Z}) \,;\, p_{X,Z} \\
(p_{X,Y} \times \mathsf{id}_Z) \,;\, q_{X,Z} & = & q_{X\times Y,Z} & = & (q_{X,Y} \times \mathsf{id}_Z) \,;\, q_{Y,Z} \\
d_{X\times Y} \,;\, (p_{X,Y} \times q_{X,Y}) & = & \mathsf{id}_{X\times Y} & &
\end{array}$$

Abstractly, a category $\mathcal{P}$ together with a functor $\otimes : \mathcal{P} \times \mathcal{P} \to \mathcal{P}$ and natural transformations $d : \mathsf{id} \to (- \times -)$, $p_{-,X} : - \times X \to \mathsf{id}$ and $q_{X,-} : X \times - \to \mathsf{id}$ satisfying these equations is called a **p-category**. Giuseppe Rosolini [6, 7] showed that any p-category $(\mathcal{P}, \otimes)$ has a full embedding $\mathcal{P} \hookrightarrow \mathsf{P}(\mathcal{S}, \mathcal{M})$ such that $\otimes$ restricts to the categorical product in $\mathcal{S}$.

We do not have an equivalence, because the supports of $\mathcal{P}$-maps may be missing. For example $\mathcal{P}$ may be defined from a programming language which has only the type $\mathbb{N}$ of natural numbers but is adequate for arithmetic; then $\mathcal{S}$ and $\mathcal{M}$ consist of the **recursively enumerable** sets of numbers.

The coproduct in $\mathcal{S}$ also extends to a functor on $\mathsf{P}(\mathcal{S}, \mathcal{M})$, where it remains the coproduct. It is not stable disjoint there, but we do not want it to be.

# 5 Relational algebra

Partial functions are a special case of **binary relations**, $R \hookrightarrow X \times Y$, in a category, which we shall use as a tool to study **while** in the next section. The following results are familiar for **Set**, but it would be instructive to show by diagram chasing that they follow from the given categorical definitions.

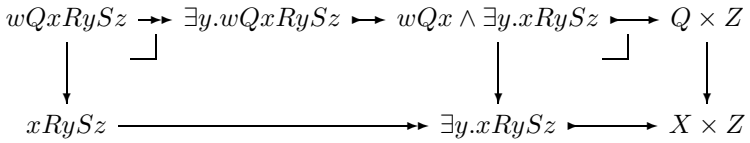
- For an **endo-relation** $U \subset X \times X$, the **equaliser** $E \hookrightarrow U \rightrightarrows X$ is isomorphic to $U \cap \Delta$, where $\Delta$ denotes the diagonal or equality relation on $X$.

Consider **composition**, for which $\Delta$ is the identity. The pullback gives a subobject, abbreviated below as $xRySz \subset X \times Y \times Z$, of the three-fold product, but not necessarily a mono into $X \times Z$. We use **stable image factorisation** to get one: any function may be expressed as a surjection followed by a mono in a way which is unique up to isomorphism and stable under pullback.

The logical form of relational composition uses an **existential quantifier**, which obeys **Frobenius' law**

$$\exists x.\phi \wedge \psi(x) \iff \phi \wedge \exists x.\psi(x)$$

where $x$ does not occur in $\phi$. The quantifier is represented categorically by factori-




sation. To say that this is *stable* means that the mono in the middle of the top row is invertible, *i.e.* Frobenius' law holds, whence

- relational composition is associative.

Although the coproduct of algebras is not stable disjoint, their homomorphisms *do* have stable image factorisation: they form **regular categories**.

A relation $R$ is **diamond** (or **confluent**, or satisfies the **Church-Rosser** property) if $R^{\mathsf{op}} \,;\, R \subset R \,;\, R^{\mathsf{op}}$. Then

- powers of a diamond relation are diamond;
- if $T$ is transitive and diamond then $T \,;\, T^{\mathsf{op}}$ is transitive;
- if $T$ is also reflexive then $T \,;\, T^{\mathsf{op}}$ is its equivalence closure.

To make full use of relational algebra we need unions: it will come as no surprise to the reader now that these must also be stable under pullback. With just finite stable unions we can now interpret disjunction as well as conjunction and existential quantification; this fragment is known as **coherent logic**.

- Relational composition distributes over stable unions.
- If $U$ is a partial function then $R = U \cup \Delta$, its reflexive closure, is diamond.

Extending to stable **countable unions**,

- the **transitive closure** of $U$ is $T = \bigcup_{n\in\mathbb{N}} U^n = \bigcup^{\uparrow}_{n\in\mathbb{N}} R^n$,

from which one can show by brute force that

- $T = \Delta \cup (U \,;\, T)$,
- $E \,;\, T = E$,
- if the relation $A$ satisfies $A \,;\, U \subset U \,;\, A$ then $A \,;\, T \subset T \,;\, A$, and
- for any partial function $V : X \rightharpoonup Z$, if $V = U \,;\, V$ then $V = T \,;\, V$.

Alternatively, if all unions exist and are preserved by the monotone functions $(-)\,;A$ and $A \,;\, (-)$ then these have right adjoints, $(-)/A$ and $A\backslash(-)$, which both reduce to **Heyting implication** when $A \subset \Delta$. The foregoing results may then be proved in a finitary way using the universal properties. We show that

$$E\backslash E \qquad \Delta \cup (U \,;\, T) \qquad A\backslash T \,;\, A \qquad V\backslash V$$

are transitive whenever $T$ is, and hence contain the transitive closure.

By putting first $R$ and then $R^{\mathsf{op}}$ for $A$ we have

- if $R$ is diamond then so is $T$

We now have quite a lot of information about the **equivalence closure** of a relation, especially if it is a partial function. Only in the case of equivalence relations do we have a direct construction of the **coequaliser** in **Set**: it is the set of equivalence classes. Here and in categories of algebras every equivalence relation (**congruence**) is the kernel pair (pullback) of its **quotient** (coequaliser); Michael Barr called such a (regular) category **exact**, though the word has also been used in other senses. An exact category with stable disjoint sums is called a **pretopos**; stable binary unions may be constructed as the quotient of the coproduct by the equivalence relation induced by the intersection.

For a general parallel pair we have first to take its image as a relation, and then the equivalence closure. This involves stable directed unions, which are a feature of finitary algebraic theories but not part of the definition of a pretopos.

# 6 Loop programs

We are now ready to interpret the program

$$\textbf{while } c \textbf{ do } U \textbf{ od}$$

which, without loss of generality, satisfies the following simplifying properties:

- The condition $c$ is guaranteed to terminate without side-effect. This can be ensured by inserting **let** $x = c$ before the loop and $x := c$ at the end of its body. The test is then just that of a (Boolean) *variable.*
- If the body $U$ were executed in a state where $c = \mathsf{no}$, then it would terminate with no effect. Replacing $U$ by **if** $c$ **then** $U$ **else skip fi** achieves this.

Now we write

- $X = [\![\Gamma]\!]$ for the type (set of states) at the beginning and end of the loop.
- $Y, N \subset X$ for the subsets on which the condition $c$ succeeds and fails, respectively; then $X = Y + N$ by the first assumption.

- $i, f : U \rightrightarrows X$, with $i$ injective, for the two maps which describe the partial endomorphism of $X$ interpreting the body $U$ of the loop.
- $E \hookrightarrow U \rightrightarrows X$ for their equaliser, so $E \cong U \cap \Delta$. By the second assumption, $N \subset E$, and it is convenient to regard these as binary subrelations of $\Delta$.
- $R = U \cup \Delta$, $T$ and $K$ for the reflexive, transitive-reflexive and equivalence closures of $U$.
- $Q$ for the coequaliser of $i, f : U \rightrightarrows X$.

The loop will be interpreted as the relation $W = K \,;\, N \subset X \times X$: we must show that this is functional. For any functional relation, *i.e.* for a parallel pair one of which is mono,

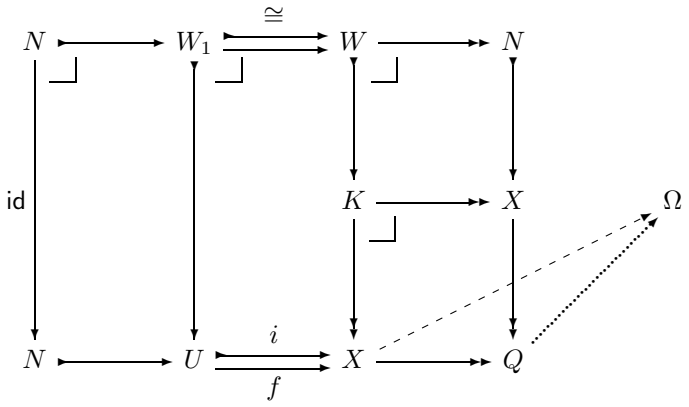
$$E \rightarrowtail U \underset{f}{\overset{i}{\rightrightarrows}} X \twoheadrightarrow Q$$

I claim that the composite from the equaliser to the coequaliser is also mono. The reflexive closure $R$ is diamond, as then is the transitive-reflexive closure $T$, so $K = T \,;\, T^{\mathsf{op}}$ is the equivalence closure. Two elements $x, y \in E$ of the equaliser become identified in the coequaliser $Q \cong X/K$ iff $\langle x, y \rangle \in K$, by exactness. Then

$$\langle x, y \rangle \in E \,;\, K \,;\, E^{\mathsf{op}} = E \,;\, T \,;\, T^{\mathsf{op}} \,;\, E^{\mathsf{op}} = E \,;\, E^{\mathsf{op}} = E \subset \Delta$$

since $E = E \,;\, T$, but then $x = y$ as claimed. This result depends on stability of unions: it fails in the category of groups.

Now consider the kernel pair of the composite $N \hookrightarrow Q$.




We showed that the vertical map $W \hookrightarrow K \twoheadrightarrow X$ is mono. This is the support of the interpretation of the loop, and $W \twoheadrightarrow N \hookrightarrow X$ is the effect.

The partial endofunction $W$ satisfies the equation

$$W \equiv \textbf{if } c \textbf{ then } U \,;\, W \textbf{ else skip fi}$$

and the condition is $\mathsf{no}$ on exit. Since $N = N^{\mathsf{op}} \subset E = E \,;\, T$,

$$W = K \,;\, N = T \,;\, T^{\mathsf{op}} \,;\, N = T \,;\, N$$

By construction $N \subset E \subset U \subset T$, so $N = N^3 \subset U \,;\, T \,;\, N$. Then

$$W = T \,;\, N = (\Delta \cup U \,;\, T) \,;\, N = N \cup U \,;\, T \,;\, N = U \,;\, T \,;\, N = U \,;\, W$$

Also $N \subset W_1 = W$ and $W = K \,;\, N = K \,;\, N \,;\, N = W \,;\, N$. In terms of the program equation, these mean respectively that $W$ behaves like $U \,;\, W$ when $c = \mathsf{yes}$ and like **skip** when $c = \mathsf{no}$, and that afterwards $c = \mathsf{no}$.

This is not yet enough to show that the interpretation is correct: we want to show that it is the *least* fixed point. We must show that $W \subset V$ for any functional relation $V$ satisfying $N \subset V = U \,;\, V$. But the two cases for $c$ give $N \subset V$ and $V = U \,;\, V$. Then $V = T \,;\, V$, so $W = T \,;\, N \subset T \,;\, V = V$.

Any map $\phi : X \to Z$ with $i \,;\, \phi = f \,;\, \phi$ is *invariant* in the strict sense that its value is restored after each iteration, so it is the same when the loop terminates (if it does) as at the beginning. Such a map factors through the coequaliser. However, according to standard usage, a **loop invariant** is a predicate (so $Z = \Omega$) which, *if* it holds before execution of the body *then* it holds afterwards. In other words $\phi$ may *become* valid when it had not been before, and the converse implication is not relevant. The correctness of **while** loops is always shown by finding an appropriate invariant, using the following proof rule:

$$\frac{\{c \wedge \phi\}\ U\ \{\phi\}}{\{\phi\}\ \mathbf{while}\ c\ \mathbf{do}\ U\ \mathbf{od}\ \{\neg c \wedge \phi\}}$$

for any predicate $\phi$. The premise of the rule means

$$\forall u \in U.\ i(u) \in Y \wedge \phi(i(u)) \Rightarrow \phi(f(u))$$

but the second assumption makes the proviso $i(u) \in Y$ unnecessary. Putting $A = \{\langle x, y\rangle : (x = y) \wedge \phi(x)\}$, it becomes $A \,;\, U \subset U \,;\, A$ with equality iff it is a strict invariant. The conclusion similarly means

$$\forall x \in W. \phi(j(x)) \Rightarrow \phi(g(x)) \wedge g(x) \in N$$

*i.e.* $A \,;\, T \,;\, N \equiv A \,;\, W \subset W \,;\, (A \cap N) \equiv T \,;\, A \,;\, N$ as relations. But we showed that $A \,;\, U \subset U \,;\, A \Rightarrow A \,;\, T \subset T \,;\, A$. Hence our interpretation $W$ is correct with respect to this rule. Equality holds for a strict invariant.

# 7 Discussion

Although we have used properties of **Set**, in particular the transitive closure, to *prove* correctness of the interpretation, it can be *stated* using finite limits and finite colimits alone. This means that any (**exact**) functor which preserves finite limits and colimits preserves the interpretation.

Moreover correctness is *reflected* if the functor $F : \mathcal{C} \hookrightarrow \mathcal{S}$ is also full and faithful. For suppose that both categories have the limits and colimits needed to draw the diagrams in the previous section (so $F$ makes these agree) and that in $\mathcal{S}$ we have shown that $FW \rightrightarrows FX$ is a functional relation satisfying

- $FN \subset FW = FU \,;\, FW$,
- $\forall V.\ FN \subset V = FU \,;\, V \Rightarrow FW \subset V$, and
- $\forall A.\ A \,;\, FU \subset FU \,;\, A \Rightarrow A \,;\, FW \subset FW \,;\, (A \cap FN)$.

Then $F$ preserves composition and intersection of relations, and reflects their containment, so these properties restrict to $\mathcal{C}$.

When can a category without infinite unions be embedded in one with them, thereby generalising the interpretation?

Stability of the coequaliser is clearly necessary, but unfortunately seems not to be sufficient. If the coequaliser of $U$ and its kernel $K$ exist then $K$ is always the union of powers of $U$ and $U^{\mathsf{op}}$, even though we have not asked for a general infinitary union operation. Then a pretopos $\mathcal{C}$ can be embedded in a topos of sheaves on $\mathcal{C}$

preserving (finite limits, coproducts, quotients of equivalence relations and) the coequaliser of $U$ iff *this union is stable under pullbacks in* $\mathcal{C}$.

Using this condition there is a simpler way to extend the proof: we only need unions of relations, not the extra objects in this sheaf topos. Define an **ideal relation** $\mathfrak{A} : X \to Y$ to be a *set* of relations $A \subset X \times Y$ which is closed downwards and under whatever *stable* unions already exist. For example

$$(A) = \{B : \ B \subset A\} \qquad \{B : \ B \,;\, E \subset E\} \qquad \{B : \ V \,;\, B \subset V\}$$

Composition and the lattice operations extend to ideal relations, indeed we have an example of an **allegory**. The last two examples are transitive, as is

$$\{B : \ (A \,;\, B) \subset \mathfrak{T} \,;\, A\}$$

for any transitive ideal relation $\mathfrak{T}$. The same argument goes through as before, the crucial point being

$$(K) = \mathfrak{T} \,;\, \mathfrak{T}^{\mathsf{op}}$$

where we return from the ideal transitive closure $\mathfrak{T}$ to the real equivalence closure $K$. For this we need that the latter be a *stable* union. To sum up,

> *The language is correctly interpreted in any pretopos such that each functional relation has an equivalence closure which is* stably *the union of powers. Any function which preserves finite limits and colimits also preserves the interpretation.*

Consider the following special case, with $x : \mathbb{N}$, which always terminates:

$$\textbf{while } x > 0 \textbf{ do } x := x - 1 \textbf{ od}$$

On exit $x = 0$, so the coequaliser is $Q \cong N = \{0\}$, whilst $Y = \{n : n \geq 1\}$, so $\mathsf{succ} : \mathbb{N} \cong Y$. The condition $X = Y + N$ and the interpretation of the program then reduce to the coproduct and coequaliser diagrams

$$\mathbf{1} \xrightarrow{0} \mathbb{N} \xleftarrow{\mathsf{succ}} \mathbb{N} \qquad \mathbb{N} \underset{\mathsf{id}}{\overset{\mathsf{succ}}{\rightrightarrows}} \mathbb{N} \longrightarrow\!\!\!\!\!\rightarrow \mathbf{1}$$

This characterisation of $\mathbb{N}$ shows [4] that exact functors (in particular inverse images of geometric morphisms) between toposes preserve $\mathbb{N}$.

The word "exact" in the title refers to the properties of **Set** relating limits and colimits which are technically known as exactness. The pun which suggests logical **completeness** is a cheat, because the full power of the coequaliser is not exploited. Analogously to the distinctions between non-normalising terms in the untyped $\lambda$-calculus, it makes a subtle identification amongst non-terminating states. For further research it would be interesting to ask whether by varying the loop condition $c$ a **full abstraction** result can be proved.

I was a Ph.D. student [8] when I found the interpretation of **while** as a coequaliser; I wrote [9], discussing p-categories, when I was a Research Assistant on *Foundational Structures in Computer Science,* and now I am an Advanced Research Fellow, all funded by the Science and Engineering Research Council.

I would like to thank Samson Abramsky, Peter Freyd, Martin Hyland, Barry Jay, Leopoldo Román, Giuseppe Rosolini, Mark Ryan, Art Stone and Bob Walters for their comments.

# References


[1] Michael Barr and Charles Wells. *Toposes, Triples and Theories.* Springer Verlag, 1984.

[2] Michael Barr and Charles Wells. *Category Theory for Computing Science.* Prentice-Hall International Series in Computer Science. Prentice-Hall, 1990.

[3] Aurelio Carboni, Stephen Lack, and Bob Walters. Introduction to extensive and distributive categories. *Journal of Pure and Applied Algebra*, 84:145–158, 1993.

[4] Peter Freyd. Aspects of topoi. *Bull. Austral. Math. Soc.*, 7:1–76, 1972.

[5] Peter Freyd and André Scedrov. *Categories, Allegories.* North-Holland, 1990.

[6] Edmund Robinson and Guiseppe Rosolini. Categories of partial maps. *Inform. and Comput.*, 79:95–130, 1988.

[7] Guiseppe Rosolini. *Continuity and Effectiveness in Topoi.* PhD thesis, Carnegie-Mellon University, 1986.

[8] Paul Taylor. *Iterated partial maps in a regular category.* Manuscript, DPMMS, Cambridge, 1984.

[9] Paul Taylor. *An exact interpretation of while, I and II.* Dept. of Computing, Imperial College, 1987. Available by anonymous FTP from `theory.doc.ic.ac.uk` in the directory `/theory/papers/Taylor`; also appeared in the draft version of these proceedings.

[10] Bob Walters. *Categories and Computer Science.* Carslaw Publications (1991), and Cambridge University Press (1992).