

# How to do something relevant for a change

CONOR MC BRIDE

*Mathematically Structured Programming Group,  
Department of Computer and Information Sciences, University of Strathclyde, G1 1XH  
(e-mail: conor.mcbride@strath.ac.uk)*

---

## Abstract

This paper explores a particular manner of dependently typed programming in which specifications stand in the foreground. Where possible, the program text concerns irrelevant proofs of propositions about relevant existential witnesses which are left implicit and inferred. In other words, the text is the explanation which provokes the mechanical construction of the part of the program that gets executed. The technique is illustrated, in the context of syntax with binding, by constructing the thin double category with *relevant* simultaneous substitutions in one dimension and variable selections in the other. The data live in the substitutions and selections, but we maintain the invariants by foregrounding the propositional squares which relate them. An Agda implementation accompanies all the work presented herein.

---

## 1 Introduction

The essence of what I call ‘Milner’s coincidence’ is that the program you write is the program the computer executes, with types conjured by the front end of the compiler to ensure that your program is safe, then exorcised by the back end of the compiler in pursuit of performance. I should like to demonstrate an alternative, in which the program the computer executes is, as far as possible, not *written* at all, but rather *inferred* by the very same machinery with which Milner conjures types. The things I prefer to make explicit are *explanations*, provocative of program inference.

My example here is simultaneous relevant substitution for what I call ‘codeBuijn’ syntax with binding—the types of terms are indexed over their *support* of nameless variables, with each such used at least once. Variables unused in some regions of a term are expunged from this support as near to the *root* of the term as possible, thus yielding a canonical representative of the term’s  $\alpha$ -equivalence class. (By contrast, de Bruijn’s indexed representation retains all the variables in scope down to the leaves where, by choosing one, you expunge the others. Correspondingly, the ‘de Bruijn shift’ which introduces a new unused variable demands a wearisome renaming action at all variable use sites.) Relevant substitutions have the property that every variable in the target support occurs in at least one image of a variable in the source support. As every source variable occurs at least once in the input term, every target variable occurs at least once in the output term. The invariant is maintained and support is respected on the nose.

The codeBrijn representation works by tightening the support of the subterms every time a term splits in two. The left and right subterms have left and right supports which make left and right selections from the support of the whole, and those selections must *cover*: any variable absent left and right should have been expunged already. These selections must then also act on substitutions, retaining the images only of selected variables from the source support, thus tightening the target variables in play to a selection from the target support.

Relevant substitutions are closed under identity and composition, and thus form a category. Selections, likewise, form a category. The squares which relate ‘vertical’ source and target selections to ‘horizontal’ super- and sub-substitutions paste compositionally in both dimensions, giving the structure of a double category, which is ‘thin’ in the sense that the data are in the points and the lines, with the enclosed area being merely propositional. The action of the source selection on the super-substitution is to complete such a square, and we may do so by giving the area—the correctness proof—explicitly, leaving the data in its boundary to be inferred by unification.

## 2 Relevant syntax

Let us begin at the beginning, which is to introduce a notion of multi-sorted syntax with binding, in the style of universal algebra. The whole setup is parametrised over some  $\text{Sort} : \mathbf{Set}$  of *term* sorts, to be defined by mutual induction. We shall need to explain how to build terms from collections of subterms. So let us introduce a notion of *adicity*,  $\mathbf{Ad} \text{Sort}$ , so that we can also have a parameter specifying the *constructors*  $\text{Ctor} : \text{Sort} \rightarrow \mathbf{Ad} \text{Sort} \rightarrow \mathbf{Set}$ , such that  $\text{Ctor } S T$  tells us which constructor tags make a  $T$ -adic term of sort  $S$ . All  $\mathbf{Ad}$  does is close  $\text{Sort}$  under unit and pairing, but also abstraction:

$$\frac{S : \text{Sort}}{S : \mathbf{Ad} \text{Sort}} \quad \frac{}{\mathbf{1} : \mathbf{Ad} \text{Sort}} \quad \frac{T, U : \mathbf{Ad} \text{Sort}}{T \times U : \mathbf{Ad} \text{Sort}} \quad \frac{S : \text{Sort} \quad T : \mathbf{Ad} \text{Sort}}{S \triangleright T : \mathbf{Ad} \text{Sort}}$$

For example, the untyped lambda calculus might be given by taking  $\text{Sort} = \mathbf{1}$  (the unit type, with  $\langle \rangle : \mathbf{1}$ ) and having

$$\text{lam} : \text{Ctor } \langle \rangle (\langle \rangle \triangleright \langle \rangle) \quad \text{app} : \text{Ctor } \langle \rangle (\langle \rangle \times \langle \rangle)$$

Scopes (and, more relevantly, supports) are snoc-lists

$$\frac{}{\varepsilon : X^*} \quad \frac{xz : X^* \quad x : X}{xz, x : X^*}$$

of sorts, and were we to introduce a *de Bruijn* syntax, we should have scoped terms

$$\frac{T : \mathbf{Ad} \text{Sort} \quad \Gamma : \text{Sort}^*}{T \dashv_{\mathbf{dB}} \Gamma : \mathbf{Set}}$$

with variables, constructors, unit, pairing and binding:

$$\frac{i : S \leftarrow \Gamma}{i : S \dashv_{\mathbf{dB}} \Gamma} \quad \frac{c : \text{Ctor } S T \quad t : T \dashv_{\mathbf{dB}} \Gamma}{c(t) : S \dashv_{\mathbf{dB}} \Gamma}$$

$$\frac{}{\varepsilon : \mathbf{1} \dashv_{\mathbf{dB}} \Gamma} \quad \frac{t : T \dashv_{\mathbf{dB}} \Gamma \quad v : V \dashv_{\mathbf{dB}} \Gamma}{t, v : T \times V \dashv_{\mathbf{dB}} \Gamma} \quad \frac{t : T \dashv_{\mathbf{dB}} \Gamma, S}{\lambda t : S \triangleright T \dashv_{\mathbf{dB}} \Gamma}$$

where  $S \leftarrow \Gamma$  is the type of deBruijn *indices*—locations of  $S$  in  $\Gamma$ , i.e., glorified numbers guaranteed to be below then length of  $\Gamma$  and to indicate the sort  $S$ .

However, my aim is to be relevant, for a change, indexing tightly over the *support* of the term, rather than loosely over the scope. We thus have *the* variable with singleton support, but constructors are as before:

$$\frac{}{\bullet : S \dashv \varepsilon, S} \quad \frac{c : \text{Ctor } S \ T \quad t : T \dashv \Gamma}{c(t) : S \dashv \Gamma}$$

Unit and pairing are sensitive to support. The empty tuple has empty support. For pairing, I shall detail  $\leq$  and  $\cup$  shortly, but they allow us to ensure that the left support  $\Gamma_l$  and the right support  $\Gamma_r$  collectively cover the whole support  $\Gamma$ .

$$\frac{}{\varepsilon : 1 \dashv \varepsilon} \quad \frac{t : T \dashv \Gamma_l \quad \theta_l : \Gamma_l \leq \Gamma \quad u : \theta_l \cup \theta_r \quad \theta_r : \Gamma_r \leq \Gamma \quad v : V \dashv \Gamma_l}{t \langle u \rangle v : T \times V \dashv \Gamma}$$

Binding comes in vacuous and relevant variants, so that an unused variable never comes *into* scope:

$$\frac{t : T \dashv \Gamma}{\kappa t : S \triangleright T \dashv \Gamma} \quad \frac{t : T \dashv \Gamma, S}{\lambda t : S \triangleright T \dashv \Gamma}$$

Note that every  $\Gamma : \text{Sort}^*$  induces an *adicity*  $\bar{\Gamma} : \mathbf{Ad} \text{ Sort}$  by tupling:

$$\bar{\varepsilon} = 1 \quad \overline{\Gamma, S} = \bar{\Gamma} \times S$$

Kindly let me suppress the  $\bar{\cdot}$  and just write supports in places that adicities are expected. We thus acquire  $\Gamma \dashv \Delta$  as the type of *relevant substitutions*, giving an image over some subscope of  $\Delta$  for each variable in  $\Gamma$ , so that all variables in  $\Delta$  get used. Our mission is thus to construct an action

$$\cdot \cdot \cdot : T \dashv \Gamma \rightarrow \Gamma \dashv \Delta \rightarrow T \dashv \Delta$$

which induces categorical structure. Note that as substitutions are made the same way as terms, this action of a substitution gives us both substitution for terms and substitution composition.

### 3 The category of thinnings, or dually, selections

Let me first, however, pick up on my promise to give the details of these ‘thinnings’, such as our  $\theta_l : \Gamma_l \leq \Gamma$ . These may be thought of as bit vectors which witness the embedding (dually, selection) of a sublist into (from) a list. The **1**s in the vector mark the places where the sublist elements go to (come from). They amount to binomial coefficients, generalised from numbers to lists and reified as types. Pascal’s triangle becomes a family of types!

$$\frac{}{\varepsilon : \varepsilon \leq \varepsilon} \quad \frac{\theta : \Gamma \leq \Delta}{\theta, 1 : \Gamma, S \leq \Delta, S} \quad \frac{\theta : \Gamma \leq \Delta}{\theta, 0 : \Gamma \leq \Delta, S}$$

We may compute the ‘all 0s’ and ‘all 1s’ thinnings

$$\bar{0} : \varepsilon \leq \Gamma \quad \bar{1} : \Gamma \leq \Gamma$$

by iteration over  $\Gamma$ , with  $\bar{1}$  giving us a suitable notion of ‘identity thinning’.

What about composition? I *could* define it as a *function*,

$$\cdot \circ \cdot : \Gamma \leq \Delta \rightarrow \Delta \leq \Theta \rightarrow \Gamma \leq \Theta$$

but then I would find myself referring to particular composite thinnings by but one of the compositions  $\theta \circ \phi$  which might deliver it. If I have a commuting square

$$\begin{array}{ccc} \Gamma & \theta_0 & \Delta_0 \\ \theta_1 & & \phi_0 \\ \Delta_1 & \phi_1 & \Theta \end{array}$$

then its diagonal is *both*  $\theta_0 \circ \phi_0$  and  $\theta_1 \circ \phi_1$ , an equation relating the *outputs* of two computations from which we will learn nothing without inspecting the *inputs*!

The way out of this trouble is to specify commuting triangles *inductively*,

$$\frac{\theta : \Gamma \leq \Delta \quad \phi : \Delta \leq \Theta \quad \psi : \Gamma \leq \Theta}{\theta \circ \phi \cong \psi : \mathbf{Set}} \quad \frac{}{\varepsilon : \varepsilon \circ \varepsilon \cong \varepsilon}$$

with a base case and three step cases,

$$\frac{v : \theta \circ \phi \cong \psi}{v, 0 : \theta \circ \phi, 0 \cong \phi, 0} \quad \frac{v : \theta \circ \phi \cong \psi}{v, 01 : \theta, 0 \circ \phi, 1 \cong \phi, 0} \quad \frac{v : \theta \circ \phi \cong \psi}{v, 1 : \theta, 1 \circ \phi, 1 \cong \phi, 1}$$

which is of course, the *graph* of the function we might otherwise have implemented. The difference is that dependent pattern matching will allow us to manipulate triangles directly, rather than via their edges or vertices.

Allow me to write  $\langle P \cdot \rangle$  to abbreviate (constructive) existential quantification ‘ $Px$  is inhabited for some  $x$ ’. We may thus write  $\langle \theta \circ \phi \cong \cdot \rangle$  as the *proposition* that  $\theta$  and  $\phi$  can be completed to a composition triangle. It is immediate that any two proofs of this proposition are equal, because the relation is the graph of a deterministic function. Moreover, we may readily define triangle completion (leaving the existential witnesses implicit).

$$\frac{\theta : \Gamma \leq \Delta \quad \phi : \Delta \leq \Theta}{\theta \triangle \phi : \langle \theta \circ \phi \cong \cdot \rangle} \quad \begin{array}{l} \theta \triangle \phi, 0 = \theta \triangle \phi, 0 \\ \theta, 0 \triangle \phi, 1 = \theta \triangle \phi, 01 \\ \theta, 1 \triangle \phi, 1 = \theta \triangle \phi, 1 \\ \varepsilon \triangle \varepsilon = \varepsilon \end{array}$$

That is, we give as explicit output an irrelevant proof object, with the relevant data it specifies (the composite thinning) kept invisible because inevitable.

It is straightforward to check the following

$$\theta \circ \bar{1} \cong \theta \quad \bar{1} \circ \phi \cong \phi \quad \bar{0} \circ \phi \cong \bar{0}$$

Meanwhile, the fact that thinnings represent *injective* maps makes

$$\langle \cdot \circ \phi \cong \psi \rangle$$

a proposition.

Just as composition is the completion of triangles, so its associativity is the completion of *tetrahedra*. If we have four points,  $\Gamma_0, \dots, \Gamma_3$  and six edges  $\theta_{ij} : \Gamma_i \leq \Gamma_j$  for  $i < j$ , then

we can fill in the  $\Gamma_0, \Gamma_2, \Gamma_3$  face, given the other three.

$$\frac{v_{012} : \theta_{01} \circ \theta_{12} \cong \theta_{02} \quad v_{013} : \theta_{01} \circ \theta_{13} \cong \theta_{03} \quad v_{123} : \theta_{12} \circ \theta_{23} \cong \theta_{13}}{\text{assoc } v_{012} v_{013} v_{012} : \theta_{02} \circ \theta_{23} \cong \theta_{03}}$$

The proof has a base case and four step cases — each element of  $\Gamma_3$  is either dropped by one of  $\theta_{23}, \theta_{12}$  or  $\theta_{01}$ , or else retained by all three.

$$\begin{array}{llll} \text{assoc } v_{012} & (v_{013}, 0) & (v_{123}, 0) & = \text{assoc } v_{012} v_{013} v_{123}, 0 & \text{dropped by } \theta_{23} \\ \text{assoc } (v_{012}, 0) & (v_{013}, 0) & (v_{123}, 01) & = \text{assoc } v_{012} v_{013} v_{123}, 01 & \text{dropped by } \theta_{12} \\ \text{assoc } (v_{012}, 01) & (v_{013}, 01) & (v_{123}, 1) & = \text{assoc } v_{012} v_{013} v_{123}, 01 & \text{dropped by } \theta_{01} \\ \text{assoc } (v_{012}, 1) & (v_{013}, 1) & (v_{123}, 1) & = \text{assoc } v_{012} v_{013} v_{123}, 1 & \text{retained} \\ \text{assoc } \varepsilon & \varepsilon & \varepsilon & = \varepsilon & \end{array}$$

In fact, the tetrahedron can be completed from the faces adjoining  $\theta_{12}, \theta_{02}$  or  $\theta_{13}$ , constructing  $\theta_{03}, \theta_{13}$  or  $\theta_{02}$ , respectively. These results are straightforward consequences of *assoc*, together with the existence and uniqueness of triangle completion.

Let us now define what it means for two thinnings to constitute a *cover*.

$$\frac{\theta_l : \Gamma_l \leq \Gamma \quad \theta_r : \Gamma_r \leq \Gamma}{\theta_l \cup \theta_r : \mathbf{Set}}$$

Covering amounts to a bitwise disjunction, and is entirely propositional. We must ensure that every element is retained either on the left, on the right, or on both.

$$\frac{u : \theta_l \cup \theta_r}{u, 10 : \theta_l, 1 \cup \theta_r, 0} \quad \frac{u : \theta_l \cup \theta_r}{u, 01 : \theta_l, 0 \cup \theta_r, 1} \quad \frac{u : \theta_l \cup \theta_r}{u, 1 : \theta_l, 1 \cup \theta_r, 1} \quad \frac{}{\varepsilon : \varepsilon \cup \varepsilon}$$

We should not be at all surprised by this outbreak of Boolean logic. The ‘in-arrows’  $\langle \cdot \leq \Delta \rangle$  for some  $\Delta$  give all the possible subsets of  $\Delta$ , paired with their embeddings into the whole. We should naturally expect these to be partially ordered by inclusion, and to admit finite unions. (Category theorists may recognize these as coproducts in the slices of  $\leq$ .) If we have  $\psi_l : \Gamma_l \leq \Delta$  and  $\psi_r : \Gamma_r \leq \Delta$ , we should be able to compute their union  $\phi : \Gamma \leq \Delta$ , together with  $\theta_l : \Gamma_l \leq \Gamma$  and  $\theta_r : \Gamma_r \leq \Gamma$  such that

$$v_l : \theta_l \circ \phi \cong \psi_l \quad u : \theta_l \cup \theta_r \quad v_r : \theta_r \circ \phi \cong \psi_r$$

Note that the thinnings  $\theta_l, \phi, \theta_r$  are bit-bearing, but the triangles and the covering are not. Let us thus explain the *latter* in such a way that the former are inferred. It is enough to tabulate the step changes in the outputs which result from the step changes in the inputs, as some sort of glorified truth table.

$\psi_l$	$\psi_r$	$v_l$	$u$	$v_r$
$\cdot, 0$	$\cdot, 0$	$\cdot, 0$	$\cdot$	$\cdot, 0$
$\cdot, 0$	$\cdot, 1$	$\cdot, 01$	$\cdot, 01$	$\cdot, 1$
$\cdot, 1$	$\cdot, 0$	$\cdot, 1$	$\cdot, 10$	$\cdot, 01$
$\cdot, 1$	$\cdot, 1$	$\cdot, 1$	$\cdot, 1$	$\cdot, 1$

Note that the triangles  $v_l$  and  $v_r$  present a *factoring* of  $\psi_l$  and  $\psi_r$  through some  $\phi$ . We can establish a universal property that the best (smallest  $\Gamma$ ) factoring is the one with the covering,  $u$ . That is, for any ‘primed pretenders’, we have a mediating thinning — the

existential witness, below

$$\theta'_l \circ \phi' \cong \psi_l \times \theta'_r \circ \phi' \cong \psi_r \rightarrow \langle \theta_l \circ \cdot \cong \theta'_l \times \cdot \circ \phi' \cong \phi \times \theta_r \circ \cdot \cong \theta'_r \rangle$$

Whenever an element is retained on either side, the pretender has no choice but to retain it in  $\phi'$ , and neither have we; whenever an element is dropped on both sides,  $\phi'$  may choose to retain it anyway, but  $\phi$  surely drops it.

## 1 Detritus

There is too much tell, here, and not enough show. Perhaps that's inevitable.

Dependently typed programming in the *intrinsic* style involves the imposition of key structural invariants *a priori* on data and the maintenance of those invariants by operations on the data. Sometimes, the poor benighted typechecker (which can do *arithmetic*, but very little *algebra*) cannot see for itself that the programmer has successfully (re)established an invariant: the peg fits the hole but will not go in without the ministrations of the mallet that is proof.

Reasoning about such programs is then complicated by reasoning about the proofs which hold them together, often requiring rather tight coordination between the proofs *in* the programs and the proofs *about* the programs, and so on, up the razor blade of meta. The ‘with’-abstraction feature of dependently typed languages, abstracting the value of an expression of interest from all types present and offering its value for inspection, provides a basic coordination mechanism, but sometimes the coupling is just too tight and the consequent game of ‘with-jenga’ too macho. Let us make the game easier by loosening the coupling, rather than increasing the cleverness with which we play it.

Let me briefly sketch the prospectus of this paper by way of furnishing some sort of example. We shall be working in a multisorted syntax where scope is explicitly managed. Scopes  $\Gamma, \Delta$  will be snoc-lists of the sorts  $S, T$  of the free variables. More particularly, *relevance* will be managed. There will be a notion of relevant *term*  $S \dashv \Gamma$  requiring that every variable in  $\Gamma$  occurs at least once. We shall further have a notion of relevant *substitution*  $\Gamma \Rightarrow \Delta$  such that every variable in  $\Delta$  occurs in the image of at least one variable from  $\Gamma$ . It should then be possible to construct an operation of type

$$S \dashv \Gamma \rightarrow \Gamma \Rightarrow \Delta \rightarrow S \dashv \Delta$$

as the images of the  $\Gamma$ -variables are all substituted at least once, and every  $\Delta$ -variable occurs in at least one image.

Wander we must, inward through the structure of the input, and we shall sometimes encounter *pairing* constructs: relevance requires that every variable in scope occurs on the left or on the right or both. That is we must make two selections,  $\theta_l : \Gamma_l \leq \Gamma$  and  $\theta_r : \Gamma_r \leq \Gamma$  which collectively form a *cover*. We say  $\theta_l \cup \theta_r$  if every variable in  $\Gamma$  is in  $\Gamma_l$  or  $\Gamma_r$  or both. Our *relevant* pair will thus consist of two subterms in  $S_l \dashv \Gamma_l$  and  $S_r \dashv \Gamma_r$ , respectively, together with a proof that  $\theta_l \cup \theta_r$ .

How are we to push substitution  $\sigma : \Gamma \Rightarrow \Delta$  into these subterms? If have only variables in  $\Gamma_l$ , then we need only the part  $\sigma_l : \Gamma_l \Rightarrow \Delta_l$  which gives the images of those variables, and there must be some  $\phi_l : \Delta_l \leq \Delta$ . Similarly, we must have  $\sigma_r : \Gamma_r \Rightarrow \Delta_r$  and  $\phi_r : \Delta_r \leq \Delta$ . To reestablish the relevance of the substitution images, we shall need to show that  $\phi_l \cup \phi_r$

whenever  $\theta_l \cup \theta_r$ . That is, if the two components covered the input variables, then their substitution images cover the output variables. Given such a proof, we may pair our  $S_l \dashv \Delta_l$  and  $S_r \dashv \Delta_r$ .

The data  $\sigma_l, \Delta_l, \phi_l$  are computed from the  $\Gamma_l, \theta_l, \Gamma, \sigma, \Delta$  and likewise on the right, and the proof that  $\phi_l \cup \phi_r$  is essential to the substitution operation. Proving properties of the substitution operation thus, apparently, necessitates negotiating the behaviour of these auxiliary computations and the key covering proof. How are we to show that the action of composed substitutions is the composition of the action of substitutions with a gnarly knot like this?

The key strategy to ameliorate these woes is *wishful thinking*. We can write relational specifications for our operations which *presuppose* all the coherence properties necessary to make the types fit together properly. Our specification of substitution will presume that the computed and proven data arrive as if by magic. We will say what ‘diagrams’ we need without specifying how they are to be computed. Our metatheorems will be entirely diagram-pasting. Meanwhile, our implementations will be proofs of diagram *completion*.

We shall, for example, express what it is to be a square  $x_l$

$$\begin{array}{ccc} \Gamma & \sigma & \Delta \\ \theta_l & x_l & \phi_l \\ \Gamma_l & \sigma_l & \Delta_l \end{array}$$

making a *double category* with vertical edges in  $\leq^{\text{op}}$  and horizontal edges in  $\Rightarrow$ . To substitute through a pair, it enough to have *somehow* acquired left and right squares  $x_l$  and  $x_r$ , together with coverings  $\theta_l \cup \theta_r$  and  $\phi_l \cup \phi_r$ . We can *reason* about relevant substitution assuming we have all of these pieces and that they all fit together.

To prove that we can *perform* substitution, we need to show that these squares can be completed from their top left boundaries, and that if the two left boundaries form a covering, then so do the two right boundaries. The proof  $x_l$  that the square is valid can be presented as data but is entirely *propositional* — there is at most one way such a square can be valid, and the code will be explicit about that explanation; the useful, bit-bearing data are the  $\sigma_l, \Delta_l, \phi_l$  which constitute the rest of the boundary, and these the program text will suppress.

We thus begin to subvert what I have sometimes called ‘Milner’s Coincidence’. In ML, the program text is the program that is executed, and the types are inferred then erased. Here, the program text is the explanation which can be erased at run time: its purpose is to provoke the inference of a correct program that is retained for execution. That is to say, by putting Milner’s idea of unification-based inference to work on the program as well as the types, we acquire a formalism for correct program calculation.