An Algebraic Construction of Predicate Transformers

Paul Gardiner 1, Clare Martin 2 and Oege de Moor 1

1 Oxford University Computing Laboratory, Programming Research Group, Keble Road, Oxford OX1 3QD, UK
2 University of Buckingham, Buckingham MK18 1EG, UK

Abstract. In this paper we present an algebraic construction of monotonic predicate transformers, using a categorical construction which is similar to the algebraic construction of the integers from the natural numbers. When applied to the category of sets and total functions once, it yields a category isomorphic to the category of sets and relations; a second application yields a category isomorphic to the category of monotonic predicate transformers. This hierarchy cannot be extended further: the category of total functions is not itself an instance of the categorical construction, and can only be extended by it twice.

1 Introduction

Predicate transformers were introduced originally by Dijkstra [8] in order to provide an elegant semantics for his programming language. Their strength lies in the fact that they can be used to model non-deterministic and non-terminating behaviour in terms of total functions, rather than relations. Not all monotonic predicate transformers represent programs in Dijkstra's language, but the language has various extensions such as the refinement calculus [26, 1, 28] in which they do all represent either programs or specifications. Indeed, monotonic predicate transformers provide a uniform framework for the specification and derivation of non-deterministic programs, in the same way that relations do for the specification and derivation of efficient functional programs [4]. In this paper we explore the mathematical properties underlying this similarity.

As motivation, the paper begins with a discussion of some algebraic properties which are common to the categories of relations and predicate transformers. Then the categorical construction which unifies the two categories is introduced. It is a slight variation of the standard construction of the category of *spans* [22], where a span is simply a pair of arrows with joint source. The category of relations can be constructed by forming the class of spans of all total functions, and identifying those which are equivalent with respect to an appropriate equivalence relation. So the construction is similar to the standard algebraic construction of the integers as a set of equivalence classes of pairs of natural numbers. The advantage of using categories instead of sets is that it is not necessary to provide an additional definition of the associated ordering and composition: both are incorporated into the definition of the span category. We call the new construction which unifies the categories of relations and predicate transformers a *skew span* category. It differs from the standard construction by taking into consideration an ordering relation associated with the category from which it is constructed. When applied to the category of sets and total functions once, it yields a category isomorphic to the category of sets and relations; a second application yields a category isomorphic to the category of monotonic predicate transformers.

The view of predicate transformers taken in this paper has some potential applications to programming and specification languages which have not yet been fully explored. At present, languages for program derivation such as the refinement calculus are untyped. The skew span construction of relations and predicate transformers provides a mechanism whereby certain aspects of the rich type structure of the category of total functions can be promoted to predicate transformers. In particular, the promotion of functors and adjunctions to lax functors and local adjunctions in span categories respectively [17] is still valid for skew span categories. Furthermore, recent research [25] has shown that data types defined by initial algebras in the category of total functions are transformed to final coalgebras under the embedding into the category of monotonic predicate transformers. Such results could prove useful for the future incorporation of types into the refinement calculus, and will be discussed in Sect. 5. The application on which we will focus at the end of this paper was suggested by Hoare [16], who has interpreted the findings in the context of specifications involving Hoare logic. When translated to the refinement calculus this provides new single complete rules for refinement and composition.

An application which will not be discussed in this paper concerns a recent result on the completeness of data refinement. In [11], it is shown that one of the reasons why monotonic predicate transformers provide a logical setting for the derivation of non-deterministic programs is that there is a single complete rule for data refinement. The same is true of the relational setting for the derivation of functional programs. A unified proof of both these observations is given in [9], where the specific results of [11] are generalized to arbitrary skew span categories.

This paper is organised as follows. In Sect. 2 we discuss the factorization properties of relations and predicate transformers which facilitate their skew span representation. The properties are described in the broader context of order enriched categories. In Sect. 3 we introduce skew span categories, and in Sect. 4 we show how such categories may be identified by their factorization properties. In Sect. 5 we briefly discuss the relevance of this work to the future development of a typed version of the refinement calculus, and in Sect. 6 we show how it provides the existing calculus with two extra proof rules.

2 Order Enriched Categories

We assume familiarity with the basic definitions of category theory, but will begin by summarizing some other standard definitions. We use the notation $(p\,;q)$ to denote the composite of each pair of arrows $p : A \to B$, $q : B \to C$. Functional composition is denoted by juxtaposition, and associates to the right.

A *preorder* $(P, \sqsubseteq)$ is a special case of a category: its objects are the members of a set P, and arrows are exactly the pairs (p, q) for which $p \sqsubseteq q$, where $\sqsubseteq$ is a reflexive and transitive relation. There is an equivalence relation $(\equiv)$ associated with every preorder which is defined by:

$$p \equiv q \Leftrightarrow (p \sqsubseteq q \text{ and } q \sqsubseteq p) \ .$$

Any equivalence relation on a set X gives rise to a partition of X into disjoint equivalence classes of the form $(\!|a|\!) = \{x \in X | x \equiv a\}$. The set of all equivalence classes of $(P, \sqsubseteq)$ under the equivalence relation associated with $\sqsubseteq$ forms a *partial order* $((\!|P|\!), \leq)$; a partial order (or *poset*) is a preorder which is anti-symmetric. The partial order $\leq$ on $(\!|P|\!)$ is defined by

$$(\!|p|\!) \leq (\!|q|\!) \Leftrightarrow p \sqsubseteq q \ .$$

A *preorder enriched category* $(\mathcal{C}, \sqsubseteq)$ is a category $\mathcal{C}$ with a preorder $\sqsubseteq$ defined on homsets, with respect to which the categorical composition ; is monotonic:

$$(p \sqsubseteq q \text{ and } r \sqsubseteq s) \quad \Rightarrow \quad p\,;r \sqsubseteq q\,;s$$

for all $p, q : A \to B$ and $r, s : B \to C$. *Poset enriched* categories are defined in the same way, except that instead of a preorder, each homset is a poset. Both these kinds of order enriched category can be thought of as a '2-dimensional' category: the arrows of $\mathcal{C}$ represent the horizontal dimension and the those of the order relation the vertical dimension. This property is formalized by the notion of a 2-*category* [12], of which preorder enriched categories are a special case. Functors between such categories must preserve both the horizontal and vertical structure, which in this case means that functors must be monotonic. Therefore when we refer to a functor between order enriched categories, we will always mean a monotonic functor.

As most of the following examples illustrate, order enriched categories are well suited for modelling programming languages with types and scopes. The object structure matches the type structure of strictly typed programming languages, and composition represents sequential execution. Furthermore, the refinement relation on programs imposes a preorder enriched structure on such categories. One program p is refined by another q if for any purpose whatsoever and in any context of use q will perform at least as well, and perhaps better than p. The associated relation is clearly reflexive, and it is transitive since if the behaviour of the program p is in all respects as good as that of the program q which in turn is as good as r, then p too is as good as r.

Examples 1.

1. $(\mathcal{M}, \leq)$ **a commutative monoid**. A *monoid* is a triple $\langle \mathcal{M}, \oplus, e \rangle$ where $\mathcal{M}$ is a set, and $\oplus$ is an associative binary operator $\oplus : \mathcal{M} \times \mathcal{M} \to \mathcal{M}$ with identity e. A commutative monoid is a monoid where the operator $\oplus$ is also commutative. A monoid determines a category with one object, usually denoted $*$. The arrows of the category are the elements of $\mathcal{M}$, with $*$ as source and target; composition is the binary operator $\oplus$. Since this category has only one object it has just one homset; the preorder is defined for all $m, n \in \mathcal{M}$ by

$$m \leq n \quad \Leftrightarrow \quad (\exists k \in \mathcal{M} \ \bullet \ m \oplus k = n) \ .$$

The operator $\oplus$ is monotonic in both arguments with respect to this ordering, so $(\mathcal{M}, \leq)$ is a preorder enriched category.

2. **(Set,=), the category of total functions** whose class of objects is the class of all sets; whose homsets $\mathbf{Set}(A, B)$ are the sets of total functions from A to B; and whose composition is the usual composition of functions. Two total functions from A to B are comparable if and only if they are equal, so the partial order on each homset is the trivial equality relation.
   This category can be used to model typed functional programming languages in which all programs are terminating and deterministic. One such program refines another only if they compute the same function, so the refinement ordering is correctly represented by equality.
3. **(Rel, $\subseteq$), the category of relations** whose class of objects is the class of all sets; whose homsets $\mathbf{Rel}(A, B)$ are the sets of subsets of $A \times B$ regarded as relations from A to B; and whose composition is the usual composition of relations. Subset inclusion is a reflexive, transitive and anti-symmetric relation, so each homset $(\mathbf{Rel}(A, B), \subseteq)$ is a partial order. Relational composition is monotonic with respect to subset inclusion, so $(\mathbf{Rel}, \subseteq)$ is a poset enriched category.
   This category is the natural setting in which to embed models of imperative programming languages which exhibit (angelic) non-determinism. Programs are represented by relations, and sequential composition by relational composition.
4. **(Pow, $\sqsubseteq$), the category of monotonic predicate transformers** whose class of objects is the class of all powersets; whose homsets $\mathbf{Pow}(\mathbb{P}A, \mathbb{P}B)$ are the sets of total monotonic functions from $\mathbb{P}A$ to $\mathbb{P}B$; and whose composition is the usual composition of functions. The subset inclusion relation determines a partial order on each object of this category, which is promoted to the pointwise order on arrows: if $p, q : \mathbb{P}A \to \mathbb{P}B$, then

   $$p \sqsubseteq q \Leftrightarrow (p\ X \subseteq q\ X \text{ for all } X \subseteq A) \ .$$

   The resulting partial order on each homset $(\mathbf{Pow}(\mathbb{P}A, \mathbb{P}B), \sqsubseteq)$ is preserved by functional composition since all the functions are monotonic, so $(\mathbf{Pow}, \sqsubseteq)$ is a poset enriched category.
   The name of this category derives from the fact that each arrow in **Pow** is equivalent to a total function from predicates to predicates, usually called a predicate transformer [8]. For if $\mathcal{P}A$ denotes the set of predicates of classical logic with free variables drawn from the set A, then the poset $(\mathcal{P}A, \Rightarrow)$ can be identified with $(\mathbb{P}A, \subseteq)$ by interpreting each predicate in $\mathcal{P}A$ as the set of values in A satisfying it.
   This category can be used to model specification formalisms like the refinement calculus.

Future references to order enriched categories will not necessarily include the associated order relations. So for example, $(\mathbf{Rel}, \subseteq)$ will be referred to as **Rel**.

Maps

Order enriched categories can accommodate a far greater range of constructs than ordinary categories, since all the standard categorical definitions can be weakened by substituting inequalities for equalities. An example of such a definition is that of a *map* [7] which is a weak analogue of the standard concept of an isomorphism. It is introduced here in order to define the factorization property shared by relations and predicate transformers.

An arrow $p : A \to B$ of a preorder enriched category $(\mathcal{C}, \sqsubseteq)$ is called a map if there exists another arrow $p^* : B \to A$ such that

$$I_A \sqsubseteq p\,;p^* \tag{1}$$

$$p^*\,;p \sqsubseteq I_B \quad . \tag{2}$$

In the trivial case that the preorder $\sqsubseteq$ is the equality relation, this definition coincides with the definition of isomorphism, and in the same way that isomorphisms are preserved by functors, maps are preserved by monotonic functors. Clearly all identities are maps, and the composites of maps are maps, so the maps form a preorder enriched subcategory of $\mathcal{C}$, which is usually denoted by $Map(\mathcal{C})$. An arrow p^* that satisfies the conditions of the above definition will be referred to here as a *comap*. Each map p determines up to equivalence its comap p^*, in the sense that if there exists another arrow $q : B \to A$ which can be substituted for p^* in (1) and (2), then $p \equiv q$. This is immediate from the following property of maps: let $p : C \to D$, $q : A \to B$ be maps in $\mathcal{C}$, then for all $r : A \to C$, $s : B \to D$ in $\mathcal{C}$

$$r\,;p \sqsubseteq q\,;s \;\Leftrightarrow\; q^*\,;r \sqsubseteq s\,;p^* \quad . \tag{3}$$

The subcategory of maps in the first two categories of Examples 1 simply consist of the isomorphisms; the last two provide more interesting examples.

Examples 2.

1. $Map(\mathbf{Rel}) = \mathbf{Set}$
   The relational converse of a relation r is denoted by r°. All total functions $f : A \to B$ in **Rel** are maps, since

   $$\begin{array}{ll} I_A \subseteq f; f^\circ & \text{total} \\ f^\circ; f \subseteq I_B & \text{functional} \;\; . \end{array}$$

   Conversely, any relation which satisfies the first map axiom (1) must be total, and if it also satisfies (2) it must be functional.
2. $Map(\mathbf{Pow}) \cong \mathbf{Rel}$
   A well-known property of monotonic functions between complete lattices [18] implies that an arrow $p : \mathbb{P}A \to \mathbb{P}B$ in **Pow** is a map if and only if it preserves arbitrary unions, which is to say that for all $S \subseteq \mathbb{P}A$,

   $$p(\cup S) = \cup\{p\; X \mid X \in S\} \quad . \tag{4}$$

   It is easy to check that all maps in **Pow** have this property, and conversely any p satisfying (4) has a comap p^* defined for all $Y \in \mathbb{P}B$ by

   $$p^*\; Y \mathrel{\hat{=}} \cup\{X \in \mathbb{P}A \mid p\; X \subseteq Y\} \quad .$$

   There is an isomorphism between **Rel** and the subcategory of **Pow** containing all functions which preserve arbitrary unions, which is defined by the *existential* *image* functor $\langle\_\rangle : \mathbf{Rel} \to Map(\mathbf{Pow})$ where for all $r : A \to B$ in **Rel** and $X \in \mathbb{P}A$

$$\begin{aligned}\langle A\rangle &\mathrel{\widehat=} \mathbb{P}A\\ \langle r\rangle X &\mathrel{\widehat=} \{b \in B \mid \exists a \in X \bullet a(r)b\}\ .\end{aligned}$$

The comap of the map $\langle r\rangle$ corresponding to each relation r is called the *universal image* of r, and will be written here as $[r]$.

Map Factorization

We will now introduce a factorization property associated with maps and comaps which resembles the standard concept of epi/monic factorization. Although neither of the categories **Rel** and **Pow** has epi/monic factorization, both have the factorization property introduced in this section.

Let $(\mathcal{C}, \sqsubseteq)$ be a preorder enriched category. We say that $\mathcal{C}$ has *map factorization* if every arrow $p : A \to B$ has a factorization

$$A \xrightarrow{p} B = A \xrightarrow{n^*} E \xrightarrow{m} B$$

where n^* is a comap and m is a map.

Examples 3.

1. **Rel** has map factorization. For let $r : A \to B$ be a relation, and let R denote that subset of the cartesian product $A \times B$ determined by r. The associated projections are given by:

   $$\begin{array}{lcl}\pi : R \to A & \text{and} & \mu : R \to B\\ \pi(a,b) = a & & \mu(a,b) = b\ .\end{array}$$

   Since both projections are total functions, we have the map factorization

   $$r = \pi^\circ\, ; \mu.$$

2. **Pow** has map factorization: one such factorization is constructed using the union function $\cup_A : \mathbb{P}\mathbb{P}A \to \mathbb{P}A$ which sends each subset of $\mathbb{P}A$ to its union, together with the powerset function $\mathbb{P}_A : \mathbb{P}A \to \mathbb{P}\mathbb{P}A$ that sends each set in $\mathbb{P}A$ to the set of its subsets. Let $p : \mathbb{P}A \to \mathbb{P}B$ be an arrow in **Pow**. The effect of applying p to some subset X of A is equivalent to mapping X to its powerset $\mathbb{P}X$, applying p to each element of the powerset, and taking the union of the resulting set of sets. So the map factorization of p is given by

   $$p = \mathbb{P}_A\, ; \langle p\rangle\, ; \cup_B$$

   where $\langle\_\rangle$ is the existential image functor introduced in Examples 2. In Sect. 5, we will use the following version of this factorization, which is expressed in terms of the *epsiloff* [10] relation $\ni_A : \mathbb{P}A \to A$, the converse of the membership relation:

   $$\mathbb{P}A \xrightarrow{p} \mathbb{P}B = \mathbb{P}A \xrightarrow{[\ni_A]} \mathbb{P}\mathbb{P}A \xrightarrow{\langle p\, ; \ni_B\rangle} \mathbb{P}B$$

   where $[\_]$ is the universal image functor introduced in Examples 2. This example also appears in [14].

Uniqueness of Factorization

The map factorization in **Rel** and **Pow** has a uniqueness property which is similar to that normally associated with epi/monic factorization [13].

Let $(\mathcal{C}, \sqsubseteq)$ be a preorder enriched category with map factorization. We will say that the map factorization in $\mathcal{C}$ is *unique up to equivalence* if for all comap/map pairs $(n^* : N \to E, m : E \to M)$ and $(n'^* : N \to E', m' : E' \to M)$,

$$n^* ; m \sqsubseteq n'^* ; m' \quad \Leftrightarrow \quad \exists h \in Map(\mathcal{C}) \bullet (h ; n' \sqsubseteq n \text{ and } m \sqsubseteq h ; m') \ . \qquad (5)$$

If the preorder $\sqsubseteq$ is a partial order we will simply refer to this property as *unique map factorization*. The following lemma gives a restatement of the above definition which is useful for validating concrete examples.

Lemma 1. *If a preorder enriched category $(\mathcal{C}, \sqsubseteq)$ has map factorization, then the factorization is unique up to equivalence if and only if whenever $m : A \to C$ and $m' : B \to C$ are maps in $\mathcal{C}$, then for all $r : A \to B$*

$$m \sqsubseteq r ; m' \quad \Rightarrow \quad \exists k \in Map(\mathcal{C}) \ \bullet \ (k \sqsubseteq r \textit{ and } m \sqsubseteq k ; m') \ . \qquad (6)$$

Proof. First suppose that $(\mathcal{C}, \sqsubseteq)$ is a preorder enriched category with unique map factorization. Let $m : A \to C$, $m' : B \to C \in Map(\mathcal{C})$, and suppose that $m \sqsubseteq r ; m'$ for some $r : A \to B \in \mathcal{C}$. Let $p^* ; q$ be a map factorization of r, so

$$m \sqsubseteq p^* ; q ; m' \ .$$

By (5) and (3) there exists a map h such that

$$h \sqsubseteq p^* \text{ and } m \sqsubseteq h ; q ; m' \ .$$

So $k = h ; q$ is a map which establishes (6).

Conversely, suppose that (6) is satisfied. Let $(n^* : N \to E, m : E \to M)$ and $(n'^* : N \to E', m' : E' \to M)$ be comap/map pairs which satisfy

$$\begin{array}{rll}
n^* ; m \sqsubseteq n'^* ; m' & \Rightarrow & \{ (3) \} \\
m \sqsubseteq n ; n'^* ; m & \Rightarrow & \{ (6) \} \\
\exists k \in Map(\mathcal{C}) \ \bullet \ (k \sqsubseteq n ; n'^* \text{ and } m \sqsubseteq k ; m') & \Rightarrow & \{ (3) \} \\
\exists k \in Map(\mathcal{C}) \ \bullet \ (k ; n' \sqsubseteq n \text{ and } m \sqsubseteq k ; m') & &
\end{array}$$

which establishes the uniqueness of factorization 5.

□

Examples 4.

1. **Rel** has unique map factorization. For suppose that $m \subseteq r ; m'$ in **Rel** where $m : A \to C$ and $m' : B \to C$ are maps and $r : A \to B$. We require a map k such that
$$k \subseteq r \text{ and } m \subseteq k ; m' \ .$$

Since k is a map in **Rel** it must be a total function. It is constructed as follows: let $a \in A$, and take c such that $(a,c) \in m \subseteq r\,;m'$. By definition of composition there exists b such that

$$(a,b) \in r \text{ and } (b,c) \in m' \ .$$

Defining $k(a) = b$ establishes the result. Note that this construction of k requires the axiom of choice. Indeed, to say that **Rel** has unique map factorization is to assert the axiom of choice.

2. **Pow** has unique map factorization. For suppose that $m \sqsubseteq p\,;m'$ in **Pow**, where $m : \mathbb{P}A \to \mathbb{P}C$ and $m' : \mathbb{P}B \to \mathbb{P}C$ are maps and $p : \mathbb{P}A \to \mathbb{P}B$. We require an arrow $k : \mathbb{P}A \to \mathbb{P}B$ which satisfies the conditions of Lemma 1. Let

$$k\ X = \cup\{p\{x\} | x \in X\} \text{ for all } X \in \mathbb{P}A$$

then clearly k preserves arbitrary unions, so it is a map in **Pow**. Since p is monotonic, $k \sqsubseteq p$. So it remains to check that $m \sqsubseteq k\,;m'$. Let $X \in \mathbb{P}A$, then

$$\begin{array}{lcl} m\ X & = & \{\ m \in Map(\mathbf{Pow})\ \} \\ \cup\{m\{x\} | x \in X\} & \subseteq & \{\ m \sqsubseteq p\,;m'\ \} \\ \cup\{m'p\{x\} | x \in X\} & = & \{\ m' \in Map(\mathbf{Pow})\ \} \\ m'(\cup\{p\{x\} | x \in X\}) & = & \{\ \text{definition of } k\ \} \\ m'kX & & \end{array}$$

which establishes the result.

3 Skew Span Categories

We will now introduce the skew span category which will be shown to provide the uniform construction of the categories of relations and predicate transformers in Sect. 4.

There are two categorical techniques for constructing the category of relations from the category of total functions. One is known as the *category of relations* and the other is known as the locally posettal reflection of the *category of spans* [6]. Both of these constructions can only be carried out for categories with certain properties. The second construction is much simpler than the first one, to the point of being naive. It was this simplicity that first directed our attention to the categorical theory of relations [22]. Here we consider a generalization of that construction which can be applied to any preorder enriched category $(\mathcal{C}, \sqsubseteq)$ provided that $\mathcal{C}$ has an asymmetric kind of weak pullback. If the preorder $\sqsubseteq$ in $\mathcal{C}$ is the equality relation $=$, then the new construction is identical to that in [22].

Preordered Spans

Let $(\mathcal{C}, \sqsubseteq)$ be a preorder enriched category. A *span* in $\mathcal{C}$ is an ordered pair of arrows with a common source

$$(n : E \to N, m : E \to M) \ .$$

For example, let $(\mathbb{N}, \leq_{\mathbb{N}})$ be the preorder enriched category represented by the monoid of natural numbers under addition $(\mathbb{N}, +, 0)$. Since all arrows in a monoid have the same source, a span is just a pair of natural numbers (n, m).

We define a preorder on spans similar to that associated with the uniqueness of map factorization; it is defined in terms of *skew span morphisms*. Let $R = (n : E \to N, m : E \to M)$ and $R' = (n' : E' \to N, m' : E' \to M)$ be spans in $\mathcal{C}$. We define a skew span morphism to be an arrow $(h : E \to E')$ in $\mathcal{C}$ such that

$$m \sqsubseteq h; m' \quad \text{and} \quad h; n' \sqsubseteq n \ . \tag{7}$$

We will use the notation $(h : R \Rightarrow R')$ to denote a skew span morphism from R to R'. This definition is illustrated by the semi-commuting diagram below:

$$\begin{tikzcd}
 & E \arrow[dl, "n"'] \arrow[dr, "m"] \arrow[dd, "h"'] & \\
N \arrow[r, phantom, "\sqsupseteq"] & & M \arrow[l, phantom, "\sqsupseteq"'] \\
 & E' \arrow[ul, "n'"] \arrow[ur, "m'"'] &
\end{tikzcd}$$

The direction of the inequalities in this definition is chosen to attune with the definition of unique map factorization, as is highlighted by Examples 5 (2) below. The standard definition of a span morphism, as used in [22, 6] is a special case of this one, in which the preorder $\sqsubseteq$ is the equality relation. The preorder on spans is given by: $R \leq R'$ if and only if there is a skew span morphism from R to R'. We will write $(\!| n, m |\!)$ for the class of spans that are equivalent to (n, m) with respect to this preorder. The preorder on spans gives rise to a partial order on equivalence classes of spans in the usual way.

Examples 5.

1. The category of natural numbers $(\mathbb{N}, \leq_{\mathbb{N}})$ provides a simple illustration of the skew span preorder. Using ordinary subtraction of integers, we have

$$(n, m) \leq (n', m') \quad \Leftrightarrow \quad (m - n) \leq_{\mathbb{N}} (m' - n') \ .$$

   In contrast, the preorder on pairs of natural numbers induced by standard span morphisms implies that the corresponding integers are equal.
2. Consider any poset enriched category $(\mathcal{C}, \sqsubseteq)$ with map factorization. To say that the map factorization in $\mathcal{C}$ is unique is the same as saying that for all spans $(n : E \to N, m : E \to M)$, $(n' : E' \to N, m' : E' \to M)$ in $Map(\mathcal{C})$

$$(n, m) \leq (n', m') \quad \Leftrightarrow \quad n^* ; m \sqsubseteq n'^* ; m' \ .$$

Pullovers

One way that two spans $(n : E \to N, m : E \to M)$ and $(n' : D \to M, m' : D \to M')$ can be combined into a single span is by taking the pullback of the inner pair of arrows $(m : E \to M, n' : D \to M)$. The pullback is the usual composition operator on spans [3], but it cannot be used here because it is not necessarily monotonic with respect to the preorder defined by skew span morphisms. Instead we introduce the notion of a *pullover* in a preorder enriched category, which is a weak analogue of a pullback. Its definition is related to that of *subequalizers* which were introduced by Lambek [19]. If subequalizers are substituted for equalizers in the construction of pullbacks from products and equalizers, then the result is nearly the same as a pullover.

Let $(\mathcal{C}, \sqsubseteq)$ be a preorder enriched category. We define the pullover of a pair of arrows $m : E \to M$ and $n : D \to M$ in $\mathcal{C}$ to be a span (r, q) such that for all spans (r', q')

$$q';n \sqsubseteq r';m \quad \Leftrightarrow \quad (r', q') \leq (r, q) \ . \tag{8}$$

Diagramatically, a pullover is represented by

$$\begin{tikzcd}
P' \arrow[dr, "h"] \arrow[drr, "q'"] \arrow[ddr, "r'"'] \arrow[dr, phantom, shift left=3, "\sqsupseteq"] \arrow[dr, phantom, shift right=3, "\sqsupseteq"'] & & \\
& P \arrow[r, "q"] \arrow[d, "r"] \arrow[dr, phantom, "\sqsupseteq"] & D \arrow[d, "n"] \\
& E \arrow[r, "m"] & M
\end{tikzcd}$$

Whereas pullbacks are defined up to isomorphism, pullovers are only determined up to equivalence of spans. The following lemma is similar to a result about pullbacks in [7], and provides a simple method for finding concrete examples of pullovers.

Lemma 2. *Let $(\mathcal{C}, \sqsubseteq)$ be a category with unique map factorization. Then the semi-commuting diagram*

$$\begin{tikzcd}
P \arrow[r, "q"] \arrow[d, "r"'] \arrow[dr, phantom, "\sqsupseteq"] & D \arrow[d, "n"] \\
E \arrow[r, "m"] & M
\end{tikzcd}$$

is a pullover diagram in $Map(\mathcal{C})$ if and only if

$$r^* ;q = m ;n^* \ . \tag{9}$$

Proof. Suppose that m, n, q, r satisfy (9) and that $(r' : P' \to E, q' : P' \to D)$ is another span of maps, then

$$\begin{array}{lcl} q' \mathbin{;} n \sqsubseteq r' \mathbin{;} m & \Leftrightarrow & \{ (3) \} \\ r'^{*} \mathbin{;} q' \sqsubseteq m \mathbin{;} n^{*} & \Leftrightarrow & \{ (9) \} \\ r'^{*} \mathbin{;} q' \sqsubseteq r^{*} \mathbin{;} q & \Leftrightarrow & \{ \text{Examples 5(2)} \} \\ (r', q') \leq (r, q) \ . & & \end{array}$$

Hence (r, q) is the pullover of (m, n). Since any two pullovers of the same pair of arrows are equivalent with respect to the skew span preorder, by Examples 5 (2) all pullovers of (m, n) must satisfy (9).

□

The first two of the following examples can be verified using this lemma.

Examples 6.

1. In (**Set**,=) the pullover coincides with the pullback:

$$P_{\mathbf{Set}} = \{ \ (x, y) \ \mid \ x \in E \ \wedge \ y \in D \ \wedge \ n\,y = m\,x \ \} \ .$$

   The span (r, q) is the pair of left and right projections

$$(\pi : P_{\mathbf{Set}} \to E, \mu : P_{\mathbf{Set}} \to D) \ .$$

   Pullovers in categories of the form $(\mathcal{C}, =)$ are not necessarily the same as pullbacks: the substitution of equality for the inequality $\sqsubseteq$ in the definition of the pullover yields only the definition of a *weak* pullback [22]. A weak pullback is like a pullback except that the universal arrow is not required to be unique.
2. Although pullbacks do not exist in **Rel**, it does have pullovers: one form of the pullover in **Rel** is given by

$$P_{\mathbf{Rel}} = \{ \ (x, y) \ \mid \ x \in \mathbb{P}E \ \wedge \ y \in \mathbb{P}D \ \wedge \ \langle n \rangle\, y \subseteq \langle m \rangle\, x \ \}.$$

   Here the span (r, q) is the pair of projections

$$((\pi \mathbin{;} \ni) : P_{\mathbf{Rel}} \to E, (\mu \mathbin{;} \ni) : P_{\mathbf{Rel}} \to D) \ .$$

3. Pullovers do not exist in **Pow** [21].

The skew span category $\boldsymbol{Span}(\boldsymbol{\mathcal{C}})$

Pullovers provide a monotonic composition operator on spans, which can now be used to define a category of spans.

Let $(\mathcal{C}, \sqsubseteq)$ be a preorder enriched category with pullovers. The *skew span* category $(\mathit{Span}(\mathcal{C}), \leq)$ is the category whose class of objects consists of all the objects in $\mathcal{C}$, and whose homsets $(\mathit{Span}(\mathcal{C}))(N, M)$ consist of the equivalence classes of spans

$$(\!| n : E \to N, m : E \to M |\!)$$

for all such n, E, m in $\mathcal{C}$. Composition in $Span(\mathcal{C})$ is defined as follows: let

$$\llparenthesis n, m \rrparenthesis : N \to M \quad \text{and} \quad \llparenthesis n', m' \rrparenthesis : M \to K$$

be arrows in $Span(\mathcal{C})$. Their composition is

$$\llparenthesis n, m \rrparenthesis \,;\llparenthesis n', m' \rrparenthesis \quad = \quad \llparenthesis r; n, q; m' \rrparenthesis \tag{10}$$

where (r, q) is a pullover of (m, n'). In a diagram:

$$\begin{tikzcd}
& & P \arrow[dl, "r"'] \arrow[dr, "q"] & & \\
& E \arrow[dl, "n"'] \arrow[dr, "m"] \arrow[rr, phantom, "\sqsupseteq"] & & D \arrow[dl, "n'"'] \arrow[dr, "m'"] & \\
N & & M & & K
\end{tikzcd}$$

Composition is associative and monotonic and $\llparenthesis I_N, I_N \rrparenthesis$ is the identity arrow associated with each object N. Since skew span morphisms define a partial order on objects in $Span(\mathcal{C})$, the monotonicity of composition is sufficient to ensure that it is also well-defined. The ordering $\leq$ on homsets in $Span(\mathcal{C})$ is the partial order on equivalence classes associated with the preorder defined by skew span morphisms.

4 Characterization Theorem

We will now show how skew span categories, like ordinary span categories can be characterized in terms of their factorization properties alone. It is then immediate that the categories of relations and monotonic predicate transformers are isomorphic to skew span categories.

It is well-known [6] that the span category of any suitable category $\mathcal{C}$ has map factorization, and that its subcategory of maps is isomorphic to $\mathcal{C}$. We will begin by showing that the same is true of skew span categories. The functor which embeds a category into its span category is called the *graph* functor $\mathsf{G} : \mathcal{C} \to Span(\mathcal{C})$ where

$$\begin{gathered} \mathsf{G}M \mathrel{\widehat{=}} M \\ \mathsf{G}(m : E \to M) \mathrel{\widehat{=}} \llparenthesis I, m \rrparenthesis : E \to M . \end{gathered}$$

The graph functor has the property that for all arrows $m, m' : E \to M$ in $(\mathcal{C}, \sqsubseteq)$,

$$m \sqsubseteq m' \quad \Leftrightarrow \quad (\mathsf{G}m \leq \mathsf{G}m') \ . \tag{11}$$

This is indicative of the way that the skew span category is really a 2-categorical concept, since the graph functor preserves not only the arrows of $\mathcal{C}$ but also the corresponding order relation. It also shows that whenever the relation $\sqsubseteq$ is a partial order the graph functor is injective. For each arrow $m : E \to M$ in $\mathcal{C}$, $\mathsf{G}m$ is a map in $\mathit{Span}(\mathcal{C})$: its comap is given by

$$(\mathsf{G}m)^* = (\!|m, I|\!) : M \to E \ .$$

Furthermore, every arrow $(\!|n, m|\!)$ in $\mathit{Span}(\mathcal{C})$ can be teased apart into the following map factorization:

$$(\!|n, m|\!) = (\mathsf{G}n)^* \,; \mathsf{G}m \ . \tag{12}$$

The following theorem shows that all maps in $\mathit{Span}(\mathcal{C})$ can be expressed in the form $\mathsf{G}m$ for some $m \in \mathcal{C}$.

Theorem 3. *Let $(\mathcal{C}, \sqsubseteq)$ be a poset enriched category with pullovers. Then*

$$\mathit{Map}(\mathit{Span}(\mathcal{C})) \cong \mathcal{C} \ .$$

Proof. The isomorphism is defined by the functor which sends each arrow in $\mathcal{C}$ to its graph in $\mathit{Map}(\mathit{Span}(\mathcal{C}))$. We have already seen from (11) that the functor is monotonic and injective, and it is clearly surjective on objects. Therefore it remains to show that every map in $\mathit{Span}(\mathcal{C})$ can be written as the graph of an arrow in $\mathcal{C}$. Let

$$p = (\!|n, m|\!) \quad \text{and} \quad p^* = (\!|n', m'|\!) \ .$$

be a map and comap in $\mathit{Span}(\mathcal{C})$, and suppose that (q, r) is a pullover of (m, n'). By (1) together with the definition of span composition (10) and skew span morphism (7), there exists $h \in \mathcal{C}$ such that

$$I \sqsubseteq h \,; q \,; m' \tag{13}$$

$$h \,; r \,; n \sqsubseteq I \ . \tag{14}$$

We will now use mutual inclusion to show that p can be written in the form of a graph. Using these inequalities we can write p in the form of a graph. For it follows from (8) and (14) that

$$(h \,; r) : (I, h \,; q \,; n') \Rightarrow (n, m) \ .$$

Therefore

$$\mathsf{G}(h \,; q \,; n') \le p \ .$$

Conversely,

$$\begin{array}{lcl}
p & \le & \{ \text{(13) and monotonicity of } \mathsf{G} \text{ and } ; \} \\
\mathsf{G}(h \,; q \,; m') \,; p & = & \{ \mathsf{G} \text{ is a functor} \} \\
\mathsf{G}(h \,; q) \,; \mathsf{G}m' \,; p & \le & \{ \text{(1) and monotonicity of } ; \} \\
\mathsf{G}(h \,; q) \,; \mathsf{G}n' \,; (\mathsf{G}n')^* \,; \mathsf{G}m \,; p & = & \{ \mathsf{G} \text{ is a functor} \} \\
\mathsf{G}(h \,; q \,; n') \,; (\mathsf{G}n')^* \,; \mathsf{G}m' \,; p & = & \{ \text{(12)} \} \\
\mathsf{G}(h \,; q \,; n') \,; p^* \,; p & \le & \{ \text{(2) and monotonicity of } ; \} \\
\mathsf{G}(h \,; q \,; n') & &
\end{array}$$

and therefore $p = \mathsf{G}(h \,; q \,; n')$ as required.

□

It follows trivially from this theorem that if $\mathcal{C}$ is a category such that $\mathcal{C} \cong \mathit{Span}(\mathcal{D})$ for some $\mathcal{D}$, then $\mathcal{C} \cong \mathit{Span}(\mathit{Map}(\mathcal{C}))$. This suggests that under appropriate conditions the the *Span* and *Map* functions might be mutually inverse operators on categories. The following theorem shows that the condition under which this is the case is the familiar unique map factorization property.

Theorem 4 Characterization Theorem. *$(\mathcal{C}, \sqsubseteq)$ is a poset enriched category with unique map factorization if and only if*

$$\mathit{Span}(\mathit{Map}(\mathcal{C})) \cong \mathcal{C} \ .$$

Proof. Suppose that $\mathit{Span}(\mathit{Map}(\mathcal{C})) \cong \mathcal{C}$, then by (12) $\mathcal{C}$ clearly has map factorization. The uniqueness of this factorization follows from Theorem 3 and the definition of the span preorder.

Conversely, suppose that $\mathcal{C}$ is a poset enriched category with unique map factorization. Then by Lemma 2, $\mathit{Map}(\mathcal{C})$ has pullovers, so the category $\mathit{Span}(\mathit{Map}(\mathcal{C}))$ certainly exists. The functor which defines its isomorphism with $\mathcal{C}$ is obviously $F : \mathit{Span}(\mathit{Map}(\mathcal{C})) \to \mathcal{C}$, where

$$\begin{gathered} FN \mathrel{\widehat{=}} N \\ F(\llparenthesis n, m \rrparenthesis : N \to M) \mathrel{\widehat{=}} (n^* \mathbin{;} m : N \to M) \ . \end{gathered}$$

Since $\mathcal{C}$ has map factorization, F is clearly surjective. It is also monotonic, well-defined and bijective, since Examples 5 (2) showed that the preorder on spans is the same as that on map factorizations. It remains to show that it is a functor: suppose that $\llparenthesis n, m \rrparenthesis : N \to M$ and $\llparenthesis n', m' \rrparenthesis : M \to M'$ are classes of spans in $\mathit{Map}(\mathcal{C})$, and that (r, q) is a pullover of (m, n'), then

$$\begin{array}{ll} F(\llparenthesis n, m \rrparenthesis \mathbin{;} \llparenthesis n', m' \rrparenthesis) & = \quad \{ \ (10) \ \} \\ F\llparenthesis r \mathbin{;} n, q \mathbin{;} m' \rrparenthesis & = \quad \{ \text{ definition of } F \ \} \\ (r \mathbin{;} n)^* \mathbin{;} (q \mathbin{;} m') & = \quad \{ \text{ contravariance of } {}^* \ \} \\ n^* \mathbin{;} r^* \mathbin{;} q \mathbin{;} m' & = \quad \{ \text{ Lemma 2 } \} \\ n^* \mathbin{;} m \mathbin{;} n'^* \mathbin{;} m' & = \quad \{ \text{ definition of } F \ \} \\ F\llparenthesis n, m \rrparenthesis \mathbin{;} F\llparenthesis n', m' \rrparenthesis \ . & \end{array}$$

Therefore since F is monotonic and preserves identities, it is a functor, which establishes the result.

□

We have now established that skew span categories and categories with unique map factorization are essentially the same. The following corollary is now immediate, since the skew span construction preserves isomorphism.

Corollary 5.

$$\begin{gathered} \mathbf{Rel} \cong \mathit{Span}(\mathbf{Set}) \\ \mathbf{Pow} \cong \mathit{Span}(\mathbf{Rel}) \ . \end{gathered}$$

The category **Set** is not itself isomorphic to a skew span category, since its only maps are bijections. Moreover, it is not possible to construct the skew span category of predicate transformers, since **Pow** does not have pullovers. So this hierarchy of span categories cannot be extended further in either direction. The relationship between predicate transformers and relations can be generalized however [23]: if $\mathbf{Rel}_{\mathcal{C}}$ is the category of relations overs an arbitrary topos $\mathcal{C}$, then the skew span category of $\mathbf{Rel}_{\mathcal{C}}$ exists and is isomorphic to a category of predicate transformers.

5 Extending Functors

Associated with the construction of skew span categories is an extension of functors and adjunctions to lax functors and local adjunctions similar to that associated with ordinary span categories [17]. In this section we will first describe this extension of functors in the context of categories with unique map factorization, and then outline its application to programming languages.

Suppose that $(\mathcal{C}, \subseteq)$ and $(\mathcal{D}, \sqsubseteq)$ are poset enriched categories where $\mathcal{C}$ has unique map factorization. Let $F : Map(\mathcal{C}) \to Map(\mathcal{D})$ be a monotonic functor. Then F has a well-defined extension to $F' : \mathcal{C} \to \mathcal{D}$ where for all spans (n, m) in $Map(\mathcal{C})$,

$$F'(n^* ;m) \mathrel{\widehat{=}} (Fn)^* ;Fm \ .$$

This extension is a monotonic graph morphism, but it is not necessarily a functor, only a *lax* functor: a lax functor $H : C \to D$ is a monotonic graph morphism such that for all composable pairs of arrows p, q and objects A in $\mathcal{C}$,

$$\begin{aligned} HI_A &\sqsubseteq I_{HA} \\ H(p ;q) &\sqsubseteq Hp ;Hq \ . \end{aligned}$$

If F' is a functor, then it is the unique extension of F in the sense that if $G : \mathcal{C} \to \mathcal{D}$ is another monotonic functor that agrees with F on $Map(\mathcal{C})$, then $G = F'$. This is immediate from the fact that monotonic functors preserve maps and comaps: for all $p \in Map(\mathcal{C})$,

$$Gp^* = (Gp)^* \ .$$

The following theorem shows that even if F' is only a lax functor it is unique in the sense that it is the least extension of F.

Theorem 6. *Suppose that $(\mathcal{C}, \subseteq)$ and $(\mathcal{D}, \sqsubseteq)$ are poset enriched categories where $\mathcal{C}$ has unique map factorization. Let $F : Map(\mathcal{C}) \to Map(\mathcal{D})$ be a functor and $G : \mathcal{C} \to \mathcal{D}$ be a lax functor that agrees with F on $Map(\mathcal{C})$, that is $Fp = Gp$ for all $p \in Map(\mathcal{C})$. Then $F' \sqsubseteq G$.*

Proof. Suppose $(n^* ;m) \in \mathcal{C}$, then

$$\begin{array}{lcl}
F'(n^* ;m) & = & \{ \text{ definition of } F' \ \} \\
(Fn)^* ;Fm & = & \{ \ F \text{ and } G \text{ agree on } Map(\mathcal{C}) \ \} \\
(Fn)^* ;Gm & \sqsubseteq & \{ \ (2) \text{ and monotonicity of } G \ \} \\
(Fn)^* ;G(n ;n^* ;m) & \sqsubseteq & \{ \ G \text{ is a lax functor } \} \\
(Fn)^* ;Gn ;G(n^* ;m) & = & \{ \ F \text{ and } G \text{ agree on } Map(\mathcal{C}) \ \} \\
(Fn)^* ;Fn ;G(n^* ;m) & \sqsubseteq & \{ \ (1) \ \} \\
G(n^* ;m) \ . & &
\end{array}$$

□

These observations show that every endofunctor F in **Rel** is the unique extension of an endofunctor in **Set** and has a canonical extension to **Pow** which is isomorphic to the lax endofunctor $F^\star$ defined by

$$F^\star p \mathrel{\widehat{=}} [F \ni] \,;\langle F(p \,;\ni)\rangle$$

where this definition uses the map factorization of predicate transformers given in Examples 3.

The technique of extending functors has applications to the theory of typed non-deterministic programming languages. Whereas deterministic languages like the lambda calculus have a definitive type structure provided by the cartesian closed category of sets and total functions, that of non-deterministic languages with either a relational or a predicate transformer semantics is less clearly defined, since neither category is cartesian closed. Therefore, when seeking to endow such languages with a natural type structure, it is useful to know that there is only one way to extend functors, and hence type constructors, from total functions to relations and predicate transformers. This fact was exploited in [24], where a relational calculus for program derivation was developed as a canonical extension of an established calculus for the derivation of functional programs [5]. Using the resulting calculus it was possible to unify a varied collection of dynamic programming algorithms.

The data types used in the functional calculus of [5] and its extensions are defined in [20] as fixed points of endofunctors in **Set**. Each data type occurs as either the initial algebra or final coalgebra of a functor, and the familiar functional programming operators such as fold or reduce are defined as homomorphisms on these data types. The reason that the calculus can be generalized so effectively to relations is that initial algebras are preserved under the extension of functors from total functions to relations. It has now been shown [25] that the extension of functors from relations to predicate transformers simply dualizes the universal properties of initial algebras by transforming them to final coalgebras. This means that the theoretical foundations for the future development of a corresponding calculus of predicate transformers are fully established. Such a calculus would combine all the benefits of the refinement calculus with the additional capability of reasoning about types.

6 An Application

In [16], it is shown how the description of specifications in terms of Hoare logic [15] can clarify the intuition behind the skew span representation of predicate transformers. Specification statements are represented as spans of relations, and the inference rules of Hoare logic are used to show that the refinement ordering on specifications is equivalent to the preorder on spans. Similarly, the inference rule for composition is used to show that the composition operator for spans would also be a suitable operator for specifications. In this section we will discuss the implications of these observations on the laws of the refinement calculus of [27]. In particular, they provide single complete rules for the refinement and composition of certain commands.

Until now we have only discussed the mathematical model of the refinement calculus: the monotonic predicate transformers. The language itself consists of a notation and a set of refinement laws for deriving programs from their specifications. It takes its inspiration from the earlier methods of Hoare and Dijkstra, but is innovative in incorporating both programs and specifications within a single framework: each command of the refinement calculus represents either a guarded command from Dijkstra's programming language [8] or a non-executable specification. The transition from each specification to executable code is broken down into a sequence of development steps, each of which is justified by one of the refinement laws. Therefore it is important to know whether the set of laws is complete, which is to say that every valid refinement can be verified using only the given laws. We will now show how the span representation of predicate transformers provides two such completeness results.

One of the most general forms of specification statement in the refinement calculus is the following:

$$|[\mathbf{con}\ X : E \bullet [\mathit{pre}, \mathit{post}]]|\ . \tag{15}$$

We will call such statements *transition* statements. As we will see, it is possible to express any statement of the refinement calculus as a transition specification. Any program satisfying specification (15) will, when activated in a state satisfying *pre*, establish a final state satisfying *post*. The two predicates *pre* and *post* have free variables drawn from a set of program variables V and a set of logical variables E; the purpose of the logical variables is to relate initial and final values of the program variables. They are assumed to denote the same value when they occur in the precondition *pre* as they do in the postcondition *post*, and therefore must not be altered by programs in order to meet the specification. A simple example of a transition specification is that met by the program which increments the value of the program variable x by 1:

$$|[\mathbf{con}\ X : E \bullet [x = X, x = X + 1]]|\ .$$

The results of Sect. 4 show that the meaning of any command in the refinement calculus can be expressed not only as a monotonic predicate transformer, but also as an equivalence class of spans of relations. The predicates *pre* and *post* in (15) both describe a relationship between the two sets of variables E and V so they can be considered alternatively as relations from E to V, as it is observed in [16]. By doing so, and using the laws of the refinement calculus to calculate the meaning of (15), we find that it has the following simple expression as a class of spans of relations:

$$(\!|\mathit{post}, \mathit{pre}|\!)\ .$$

So every monotonic predicate transformer, and therefore any statement of the refinement calculus, corresponds to an equivalence class of transition specifications. The refinement ordering on specifications is defined to be the pointwise ordering on the corresponding predicate transformers, which is equivalent to the skew span preorder on spans of relations. Therefore, the latter can be translated into a single complete rule for the refinement of transition specifications. Let *pre* and *post* be a pair of logical formulae with free variables drawn from the set of the program variables V and the set of logical variables E. Similarly for the pair *pre*$'$ and *post*$'$ with respect to the sets V and E'. When it is necessary to make the set of free variables associated with a predicate explicit we will use subscripts to do so: for example *pre* would be written as pre_{EV}. The definition of the preorder on spans of relations translates to the following preorder relation between specifications:

$$
\begin{array}{l}
\textit{Span Preorder}: \\
\qquad |[\mathbf{con}\ X : E \bullet [pre, post]]| \sqsubseteq |[\mathbf{con}\ X' : E' \bullet [pre', post']]| \\
\mathrel{\widehat{=}} \\
\qquad \exists h_{EE'} \bullet (pre \Rightarrow (\exists X' : E' \bullet h \wedge pre')) \wedge ((\exists X' : E' \bullet h \wedge post') \Rightarrow post)\ .
\end{array}
$$

This rule can be broken down into the following three more familiar rules, which are together equivalent to it. For simplicity suppose now that unless indicated otherwise, all predicates share the same two sets of free variables E and V. Then the Span Preorder rule is equivalent to the following:

1. *Weaken Precondition*: If $pre \Rightarrow pre'$ then

$$|[\mathbf{con}\ X : E \bullet [pre, post]]| \sqsubseteq |[\mathbf{con}\ X : E \bullet [pre', post]]|\ .$$

2. *Strengthen Postcondition*: If $post' \Rightarrow post$ then

$$|[\mathbf{con}\ X : E \bullet [pre, post]]| \sqsubseteq |[\mathbf{con}\ X : E \bullet [pre, post']]|\ .$$

3. *Change logical variable*:

$$
\begin{array}{l}
\qquad |[\mathbf{con}\ X' : E' \bullet [(\exists X : E \bullet h_{EE'} \wedge pre), (\exists X : E \bullet h_{EE'} \wedge post)]]| \\
\sqsubseteq \\
\qquad\qquad |[\mathbf{con}\ X : E \bullet [pre, post]]|\ .
\end{array}
$$

Both the first two of these laws occur among the laws of the refinement calculus. Law (3) is not listed explicitly, but is derivable within the calculus, so this shows that the laws of the refinement calculus are complete for proving refinements of transition specifications.

Similar reasoning can be used to translate the skew span composition operator into a composition law for transition specifications. For simplicity we will just consider the case where the specifications being composed share the same set of logical variables and program variables E and V respectively. The composition becomes:

$$
\begin{array}{l}
\textit{Span Composition}: \\
\qquad |[\mathbf{con}\ X : E \bullet [pre, mid]]| \,;\, |[\mathbf{con}\ X : E \bullet [mid', post]]| \\
\mathrel{\widehat{=}} \\
\qquad |[\mathbf{con}\ Y : D \bullet [(\exists X : E \bullet h_{DE} \wedge pre), (\exists X : E \bullet h'_{DE} \wedge post)]]|
\end{array}
$$

where (h, h') is any pair of predicates such that for all predicates ψ such that Y does not occur free in ψ,

$$(\exists Y : D \bullet (h \wedge (\forall X : E \bullet h' \Rightarrow \psi)) = (\forall x : V \bullet mid \Rightarrow (\exists X : E \bullet mid' \wedge \psi))\ .$$

One suitable choice for h and h' is the following: let $\mathcal{P}E$ denote the set of predicates with free variables drawn from E, and let $D = \mathcal{P}E \times \mathcal{P}E$. Then for all $Y : D$ there exist predicates ϕ_E and ψ_E such that $Y = (\phi, \psi)$, so we can define h and h' as follows

$$\begin{aligned} mids_D &\mathrel{\widehat{=}} (\exists X : E \bullet mid \land \phi) \Rightarrow (\exists X : E \bullet mid' \land \psi) \\ h &\mathrel{\widehat{=}} mids \land \psi \\ h' &\mathrel{\widehat{=}} mids \land \phi \ . \end{aligned}$$

This definition of composition is rather unwieldy, and unlike the law for refinement, it cannot be proved from the laws of the refinement calculus, although it is possible to show that the expression on the right of the definition is refined by that on the left. Therefore the laws are complete for proving refinements of specifications into smaller pieces. The laws are usually only used in this way, since the process of derivation usually involves breaking a specification down into manageable parts, each of which is independently refined to code. However, it is conceivable that it might be useful sometimes to have a rule which does the opposite, and combines two specifications into a larger one. For example, it might be possible to substitute a library procedure which meets the composite of two specifications, but neither individually. Or, more probably, it might be useful once derivation is completed, to calculate the loosest specification satisfied by the composite of two programs whose separate specifications are known.

Acknowledgements

The authors would like to thank Tony Hoare and Carroll Morgan for their comments and encouragement. We gratefully acknowledge the financial support of the SERC and British Petroleum International, which made our cooperation possible.

References

1. Back, R.J.R.: A Calculus of Refinements for Program Derivation. Acta Informatica **25** (1988) 593-624
2. Barr, M.: Relational Algebras. In Reports of the Midwest Category Seminar IV, Springer-Verlag Lecture Notes in Mathematics **137** (1069) 435-477
3. Benabou, J.: Introduction to Bicategories. Springer-Verlag Lecture Notes in Mathematics **47** (1967)
4. Bird, R.S.,de Moor, O.: Solving Optimisation Problems with Catamorphisms. (1992) This volume.
5. Bird, R.S.: Lectures on Constructive Functional Programming. In M. Broy, editor. Constructive Methods in Computing Science, Springer-Verlag NATO ASI series F **55** (1989) 151-216
6. Carboni, A.,Kasangian, S.: Bicategories of Spans and Relations. Journal of Pure and Applied Algebra **33** (1984) 259-267
7. Carboni, A., Kelly, M., and Wood, R.: A 2-Categorical Approach to Geometric Morphisms I. Sydney Pure Mathematics Research Reports **89-19** (1989). Department of Pure Mathematics, University of Sydney, NSW 2006, Australia.
8. Dijkstra, E.W.: A Discipline of Programming. Prentice-Hall, Englewood Cliffs (1976)
9. Gardiner, P.H.B.: Data Refinement of Maps. Draft (1990)

10. Freyd, P.J. and Scedrov, A.: Categories, Allegories, North-Holland (1990)
11. Gardiner, P.H.B., Morgan, C.C.: A Single Complete Rule for Data Refinement. Technical Report PRG-TR-7-89, Programming Research Group, 11 Keble Road, Oxford OX1 3QD, 1989.
12. Gray, J.W.: Formal Category Theory: Adjointness for 2-categories. Springer-Verlag Lecture Notes in Mathematics **391** (1974)
13. Herrlich H., Strecker G.E.: Category Theory. Allyn and Bacon Inc., Boston (1973)
14. Hesselink, W.H.: Modalities of Nondeterminacy. In Beauty is Our Business: A Birthday Salute to Edsger W. Dijkstra. Springer-Verlag, (1990) 182-193
15. Hoare, C.A.R.: An Axiomatic Basis for Computer Programming. Communications of the ACM **12(10)** (1969) 576-580,583
16. Hoare, C.A.R. Handwritten note (1991)
17. Jay, C.B.: Local Adjunctions. Journal of Pure and Applied Algebra **53** (1988) 227-238
18. Johnstone, P.T.: Stone Spaces Cambridge University Press (1982)
19. Lambek, J.: Subequalizers. Bulletin Amer. Math. Soc. **13(3)** (1970) 337-349
20. Malcolm, G.: Data Structures and Program Transformation. Science of Computer Programming **14** (1990) 255-279
21. Martin, C.E.: Preordered Categories and Predicate Transformers. D. Phil Thesis. Programming Research Group, 11 Keble Road, Oxford OX1 3QD. (1991)
22. de Moor, O.: Categories, Relations and Dynamic Programming. PRG Technical Report PRG-TR-18-90, Programming Research Group, 11 Keble Road, Oxford OX1 3QD. (1990)
23. de Moor, O., Gardiner, P.H.B., and Martin, C.E.: Factorizing Predicate Transformers in a Topos. Draft (1991)
24. de Moor, O.: Categories, Relations and Dynamic Programming. D. Phil Thesis. PRG Technical Monograph PRG-98, Programming Research Group, 11 Keble Road, Oxford OX1 3QD. (1992)
25. de Moor, O.: Inductive Data Types for Predicate Transformers. Information Processing Letters (to appear)
26. Morgan, C.C.: Programming From Specifications. Prentice-Hall (1990)
27. Morgan, C.C., Robinson, K. and Gardiner, P.H.B.: On the Refinement Calculus. PRG Technical Monograph PRG-70. Programming Research Group, 11 Keble Road, Oxford OX1 3QD. (1988)
28. Morris, J.M.: Laws of Data Refinement. Acta Informatica **26** (1989) 287-308

This article was processed using the LaTeX macro package with LLNCS style