Formal Aspects of Computing (2003) 3: 1–11

Data Refinement, Call by Value, and Higher Order Programs

David A. Naumann[1]

Mathematics and Computer Science, Southwestern University, Georgetown, TX 78626 USA

Keywords: programming calculi, refinement calculus, data refinement, predicate transformers, higher types, lax exponent.

Abstract. Using 2-categorical laws of algorithmic refinement, we show soundness of data refinement for stored programs and hence for higher order procedures with value/result parameters. The refinement laws hold in a model that slightly generalizes the standard predicate transformer semantics for the usual imperative programming constructs including prescriptions.

1. Introduction

Data refinement, sometimes called coordinate transformation or change of data representation, has appeared in a number of formulations [Hoa72, CU89]. For data refinement to be sound means that it can be carried out piecewise, explicitly refining only those parts of the program that involve the transformed data type: each program construct should preserve data refinement in the sense that constructed programs are refined whenever their constituents are. This is a monotonicity property, just as crucial for program development as monotonicity with respect to algorithmic refinement.

Soundness of data refinement has been shown for the simple imperative language extended with local variables, recursion, and procedures with value and result parameters [GM91, Mor89]. Data refinement has also been shown to be sound for "prescriptions" [GM91]. Our contribution is to show that data refinement is sound for higher order programs.

Prescriptions represent pre/post specifications as abstract programs, which

[1] Partially supported by a Southwestern University Cullen Foundation grant.
Correspondence and offprint requests to: D.A. Naumann

gives rise to a calculus of programming based on the relation of (algorithmic) refinement: the refinement ordering represents satisfaction of specifications by programs as well as refinement of one program by another [Bac80, Mor94]. Predicate transformers provide the standard model of refinement calculus. For programs alone, predicate transformers are but one of several well known models. Plotkin and Smyth studied the close connection between predicate transformer and state transformer (functional and relational) models [Plo81, Smy83]. This "Stone duality" was explored in depth by Abramsky and others [Abr91, Vic89, BK93]. Aside from the methodological advantages touted by Dijkstra, predicate transformers have a mathematical advantage. By dropping all healthiness conditions except monotonicity, we obtain a model for prescriptions freely intermixed with programs — the refinement calculus. Although this leaves Stone duality behind, it has a mathematically elegant descriptions in terms of "span categories" [GMdM94] — which is beyond our scope, except insofar as "maps" play a role in the sequel.

Predicate transformer semantics has recently been given for higher order imperative programs, with procedure type variables and hence call-by-value parameters [Mar91, Nau92a, Nau94b]. That work uses program constructs corresponding to categorical "exponents": Currying and application. Alas, these constructs do not satisfy very strong laws in general; but a few conditional inequations are sufficient to determine them uniquely, and those laws also suffice to show preservation of data refinement.

Readers familiar with the categorical axiomatization of higher order functions (*e.g.* [Pie91]) should find the weak exponent and weak product at least vaguely familiar. The laws needed here are stated elementarily, so very little knowledge of category theory is needed to read this paper — provided the reader ignores occasional remarks made for the benefit of experts. A tutorial introduction to the weak exponent appears elsewhere [Nau92a].

Many program constructs can be shown to preserve data refinement thanks to just their simple algebraic properties, *i.e.* their refinement laws. There has been work on a categorical theory of preservation of data refinement [HH90, Nau92b]; it is actually a "2-categorical" theory because ordered categories are needed to cater for inequational refinement laws. However, the published results [Pow89] are not adequate for the refinement notions and programming constructs of interest, and none of the results known to me are adequate to deal with the very weak product of state spaces on which the exponent is based. The published proofs for specific constructs of refinement calculus are at the level of predicates, using predicate transformer semantics [GM91, Mor89]. It seems clear from the proofs in the sequel that ultimately it will be possible to extend the categorical treatment to deal with constructs like the very weak product. Recent work of Kinoshita and Power seems most promising [KP94].

For soundness of the constructs for higher order programs, we need that the exponent is functorial, *i.e.* preserves identities and distributes through composition. But in the standard model of predicate transformers as functions between powersets, the exponent fails to preserve identities [Nau92a]. (There are actually several weak exponent and weak product structures; we say "the" to refer to the specific ones which are operationally adequate — which means in part that they restrict to ordinary products and exponents for predicate transformers that correspond to total functions.) To obtain a functorial exponent, we use a slightly more general model where states are partially ordered and predicates are required to be monotonic (*i.e.* upward closed subsets). This model subsumes the

standard one (by taking the order to be equality) and all the other definitions and laws still apply, with one exception: at higher types, the logic of predicates is intuitionistic rather than Boolean (as in [Ten94, Abr91, Vic89]). Details of a denotational semantics for a higher order imperative language have appeared, using the generalized model validating all the laws in the sequel [Nau94b]. Category theorists may be interested to see that the generalized model arises via a span construction from posets [Nau94a] just as predicate transformers between powersets arise by span construction from sets.

Data refinement has been studied for higher order functional languages using "logical relations" [Mit90], but it is not trivial to extend that approach to higher order imperative programs, due in part to interaction between local variables and parameters [Ten94, OT93]. Recently, soundness of data refinement was shown for an Algol-like language with call-by-name procedure type parameters [Ten94]. That result, however, is for a restricted form of data refinement based on algorithmic *equivalence* rather than *refinement*. Tennant has observed that it may also be difficult to extend that work to a semantics of the full refinement calculus with prescriptions, and to a semantics of stored procedures (which are important in object-oriented programming and as the basis for call-by-value parameters).

Contents. Section 2 describes the ordered category of predicate transformers, and gives the refinement laws we need for the weak product and weak exponent constructs; it also defines data refinement. Section 3 shows that weak products preserve data refinement; although new, this result is primarily of interest as a basis for higher types. Section 4 shows that the basic exponent constructs —generalized Currying and application— preserve data refinement.

2. Prerequisites

We begin with a set-theoretic model of predicates, in spite of the fact that a major virtue of predicate-transformer semantics is its focus on the algebra of predicates, which is best used unencumbered by "points" and models. Indeed, our interest is in the algebra of programs, unencumbered by unnecessary predicates. But we want an algebra that is sound with respect to the behaviors of the machines we program, and the set-theoretic model has a clear connection with operational semantics.

Results in this section may be found in [Nau92a]; some of them, along with many related results, appear elsewhere [Mar91, Nau92b, DM92]. We work here in the opposite of the category used in the cited work, so "co-exponent" becomes "exponent" and $\oplus$ becomes $\otimes$.

Predicate transformers. The application of function f to argument x is written $f.x$, as in the expression $\mathbb{P}.A$ for the powerset of a set A. In section 4, $\mathbb{P}.A$ will be reinterpreted as the monotonic predicates (upward closed subsets) of poset A. We refrain from doing so at the outset in order to delineate the part of the results where it matters. A *predicate transformer* f is a function $f \in \mathbb{P}.A \to \mathbb{P}.B$ that is monotonic with respect to subset inclusion, *i.e.* $\alpha \subseteq \alpha' \;\Rightarrow\; f.\alpha \subseteq f.\alpha'$ for all subsets α and α' of A. The refinement order is defined pointwise: for predicate transformers $f, g \in \mathbb{P}.A \to \mathbb{P}.B$, define $f \sqsubseteq g$ iff $(\,\forall\, \alpha \;:\; \alpha \in \mathbb{P}.A \;:\; f.\alpha \subseteq g.\alpha\,)$. Variables f, g, h, k, p, q range over predicate transformers. The identity function on $\mathbb{P}.X$ is written id_X.

Predicate transformers $f \in \mathbb{P}.B \to \mathbb{P}.A$ model state-transformations from state space A to B. Typically predicate transformers are defined for a single fixed state space, but we need the types to vary in order to use categorical notions. At the risk of some confusion, we use notations and categorical terminology for the category **Prog**, which is simply the opposite of the category **Tran** of predicate transformers used in some work [Mar91, GMdM94].[2] For objects, **Prog** has all (small) sets. For any sets A, B, the homset $\mathbf{Prog}(A, B)$ is the set of predicate transformers $f \in \mathbb{P}.B \to \mathbb{P}.A$. Note the reversal of direction. The homsets of **Prog** are partially ordered by $\sqsubseteq$. The composition of $f \in \mathbf{Prog}(A, B)$ with $g \in \mathbf{Prog}(B, C)$ is written $(f\,;g)$, so $(f\,;g)$ is the usual composition $f \circ g$ of functions (recall Dijkstra's $wp.(S\,;Q) = wp.S \circ wp.Q$). Composition is monotonic with respect to $\sqsubseteq$, so **Prog** is an order-enriched category [Kel82] (but not complete-lattice enriched, although the homsets are complete lattices). Monotonicity of composition and transitivity of $\sqsubseteq$ are used without mention in the proofs below.

Comaps and maps. An arrow $c \in \mathbf{Prog}(A, B)$ is a *comap* just if it has a *map* (left adjoint) $c^\circ \in \mathbf{Prog}(B, A)$. That is, c and c° satisfy

$$id_A \sqsubseteq c^\circ\,;c \qquad \text{and} \qquad c\,;c^\circ \sqsubseteq id_B \quad . \tag{1}$$

A map and its comap determine each other uniquely. It is standard that, because c is a monotonic function between complete posets, it is a comap just if it is universally disjunctive; the same for maps and conjunctivity. (Keep in mind that **Prog** is the opposite of **Tran**: a map in **Prog** would be called a comap in **Tran**.) A useful property of maps and comaps is

$$c\,;f \sqsubseteq g \;\equiv\; f \sqsubseteq c^\circ\,;g \qquad \text{for all } f, g \quad . \tag{2}$$

Weak products. For any sets A and B, the disjoint union $A + B$ is a coproduct object in **Prog**, but our interest here is in Cartesian products.[3] It can be shown (*e.g.* [Mar91]) that **Prog** does not have categorical products. We use the operationally sensible Cartesian product of state spaces, which we call the weak product.

Define $A \otimes B = A \times B$, so that **Prog**-arrows with target $A \otimes B$ are predicate transformers with source $\mathbb{P}.(A \times B)$. For $f \in \mathbf{Prog}(D, A)$ and $g \in \mathbf{Prog}(D, B)$, the "pairing" $f \lhd g \in \mathbf{Prog}(D, A \otimes B)$ generalizes the pairing of functions. The left projection $\pi \in \mathbf{Prog}(A \otimes B, A)$ is the predicate transformer corresponding to the projection function.[4] The action of $\otimes$ on arrows is defined in terms of $\lhd$ and the projections: $f \otimes g = (\pi\,;f \lhd \pi'\,;g)$. We let $(;)$ bind more tightly than $\lhd$ and $\otimes$ (and later $\rightsquigarrow$). The following laws hold for all f, g, h, k for which the expressions are defined [Mar91, Nau92b].

$$h\,;(f \lhd g) \sqsubseteq h\,;f \lhd h\,;g \qquad \text{if } h \text{ is a comap} \tag{3}$$

$$f\,;h \lhd g\,;k \sqsubseteq (f \lhd g)\,;(h \otimes k) \tag{4}$$

$$(f \otimes g)\,;\pi \sqsubseteq \pi\,;f \qquad \text{if } f, g \text{ are comaps} \tag{5}$$

[2] A similar choice is made in Locale theory [Abr91, Vic89] and in some work on predicate transformers [Plo81, Nau94c].

[3] The coproduct in **Prog** does preserve data refinement, as can be shown using the categorical theory [HH90, Nau92b]. An elementary proof has recently appeared [BB94].

[4] For completeness, here are the definitions: $\pi_{A,B}.\alpha = \alpha \times B$, and for $\gamma \subseteq A \times B$ define $(f \lhd g).\gamma = (\cup \alpha, \beta : \alpha \times \beta \subseteq \gamma : f.\alpha \cap g.\beta)$.

$$f\,;h\otimes g\,;k \;\sqsubseteq\; (f\otimes g)\,;(h\otimes k) \tag{6}$$
$$f\,;h\otimes g\,;k \;=\; (f\otimes g)\,;(h\otimes k) \qquad \text{if } h,g, \text{ are maps} \tag{7}$$

By inequation (6), and the fact that $\otimes$ is monotonic and preserves identities, $\otimes$ is a "lax functor" [BP88].

Aside: In conventional denotational semantics, Cartesian products are used both for coordinates of the state space and for components of the environment. In predicate transformer semantics, commands denote transformers (so the weak laws pertain to the product of state spaces) but, as usual, environments are functions from identifiers to their values (in particular, the usual laws of categorical products are still valid for environments) [Nau94b].

Weak exponents. For any $f \in \mathbf{Prog}(B\otimes A, C)$, "Curried f" is the transformer $\mathsf{cur}.f \in \mathbf{Prog}(A, B \rightsquigarrow C)$, where —until further notice— $B \rightsquigarrow C$ may be any chosen subset of $\mathbf{Prog}(B,C)$ (provided that $f \in A \rightsquigarrow B$ and $g \in B \rightsquigarrow C$ implies $(f\,;g) \in A \rightsquigarrow C$ for all A,B,C,f,g). Various considerations [Nau92a] suggest different choices for $B \rightsquigarrow C$, but several of the laws we need are independent of the choice. There is an "application" $\mathsf{ap}_{B,C} \in \mathbf{Prog}(B\otimes(B\rightsquigarrow C), C)$.[5] The action of $\rightsquigarrow$ on arrows is defined in terms of ap and cur. For now we separate the arguments of $\rightsquigarrow$ and give its action on object X and arrows $g \in \mathbf{Prog}(B,A)$, $h \in \mathbf{Prog}(D,C)$ by

$$X \rightsquigarrow g = \mathsf{cur}.(\mathsf{ap}\,;g) \quad \text{and} \quad h \rightsquigarrow X = \mathsf{cur}.((h\otimes id_{D\rightsquigarrow X})\,;\mathsf{ap})$$

so $(X\rightsquigarrow g) \in \mathbf{Prog}((X\rightsquigarrow B),(X\rightsquigarrow A))$ and $(h\rightsquigarrow X) \in \mathbf{Prog}((C\rightsquigarrow X),(D\rightsquigarrow X))$. Here are most of the laws needed in the sequel [Nau92a].

$$\mathsf{cur}.(f\,;h) \;\sqsubseteq\; \mathsf{cur}.f\,;(B\rightsquigarrow h) \tag{8}$$
$$\mathsf{cur}((id_B\otimes g)\,;f) \;\sqsubseteq\; g\,;\mathsf{cur}.f \qquad \text{if } g \text{ is a map} \tag{9}$$
$$\mathsf{cur}.((f\otimes id_D)\,;k) \;\sqsubseteq\; \mathsf{cur}.k\,;(f\rightsquigarrow B) \tag{10}$$
$$X\rightsquigarrow f\,;g \;\sqsubseteq\; (X\rightsquigarrow f)\,;(X\rightsquigarrow g) \tag{11}$$
$$g\,;f\rightsquigarrow X \;\sqsubseteq\; (f\rightsquigarrow X)\,;(g\rightsquigarrow X) \tag{12}$$
$$id_A\rightsquigarrow X \;\sqsubseteq\; id_{A\rightsquigarrow X} \qquad \text{and} \qquad X\rightsquigarrow id_A \;\sqsubseteq\; id_{X\rightsquigarrow A} \tag{13}$$

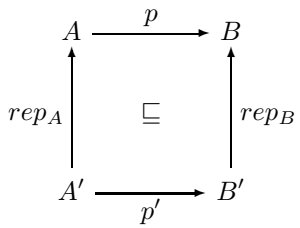
For any f,g,h, we have that $X\rightsquigarrow g$, $h\rightsquigarrow X$, and $\mathsf{cur}.f$ are maps. Also, cur is monotonic with respect to $\sqsubseteq$.

Preservation of data refinement. Among the various definitions of data refinement [CU89], the one most convenient for us is adapted from Gardiner and Morgan [GM91]. For $p \in \mathbf{Prog}(A,B)$ to be *data-refined* by *rep* to $p' \in \mathbf{Prog}(A',B')$ means: $rep_A \in \mathbf{Prog}(A',A)$ and $rep_B \in \mathbf{Prog}(B',B)$ are comaps, and

$$rep_A\,;p \;\sqsubseteq\; p'\,;rep_B \quad . \tag{14}$$

This inequality is depicted

[5] Here are the definitions, for transformers between powersets [Nau92a]. For $a \in A$, and $\delta \subseteq B\rightsquigarrow C$, define $\mathsf{cur}.f$ by $a \in \mathsf{cur}.f.\delta \;\equiv\; (\forall g : g \in B\rightsquigarrow C \wedge g \sqsupseteq \mathsf{cu}.f.a : g \in \delta)$, where $\mathsf{cu}.f.a \in \mathbf{Prog}(B,C)$ is defined by $b \in \mathsf{cu}.f.a.\gamma \;\equiv\; (b,a) \in f.\gamma$. And $(b,g) \in \mathsf{ap}_{B,C}.\gamma \;\equiv\; b \in g.\gamma$ for $\gamma \subseteq C$ and $g \in B\rightsquigarrow C$. For transformers between monotonic predicates over posets discussed after Theorem 3, cur reduces to the Stone dual of Currying, still using cu. That is, $a \in \mathsf{cur}.f.\delta \;\equiv\; \mathsf{cu}.f.a \in \delta$. Everything else stays the same.



One may say that *rep* translates predicates on the "concrete" data to predicates on the "abstract". Often, rep_A is the direct image function for a relation from A' to A. The conjunction of (14) and the comap requirement is sometimes called Galois simulation or total simulation.

For a construct $\mathcal{C}$ to *preserve data refinement* means that if $\mathcal{C}$ is applicable to p (hence to p') and (14) holds then

$$rep\,;\mathcal{C}[p] \;\sqsubseteq\; \mathcal{C}[p']\,;rep \quad . \tag{15}$$

In (15), the occurrences of *rep* need to be constructed using $\mathcal{C}$ if $\mathcal{C}$ is a type constructor like $\otimes$ and $\rightsquigarrow$, in which case each constructed *rep* needs to be a comap. No more formal definition of "preserving data refinement" is given here; what we mean is exactly what is proved for $\otimes$ in section 3 and for $\rightsquigarrow$ in section 4.

Having described a particular model and some of its laws, we proceed to use only the laws; the results to follow hold in a variety of models.

3. Weak product

This section shows that data refinement is preserved by the weak product. First it is shown that $\lhd$ preserves data refinement; with the definition of $rep_{A\otimes B}$ given in the proof, $\pi_{A,B}$ is then shown to be data-refined to $\pi_{A',B'}$ (when A', B' represent A, B). As a consequence, $\otimes$ preserves data refinement, since it is defined in terms of $\lhd, (;)$, and the projections.

Theorem 1. $\lhd$ preserves data refinement.

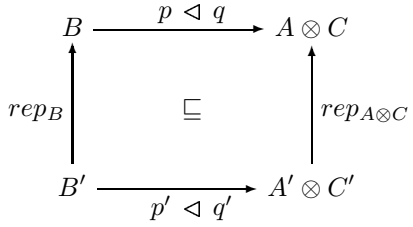
Proof. Given rep_A, rep_B, rep_C such that

$$rep_B\,;p \;\sqsubseteq\; p'\,;rep_A \qquad \text{and} \qquad rep_B\,;q \;\sqsubseteq\; q'\,;rep_C \tag{16}$$

for $p \in \mathbf{Prog}(B, A)$ and $q \in \mathbf{Prog}(B, C)$ (and primed counterparts), we need $rep_{A\otimes C} \in \mathbf{Prog}((A' \otimes C'), (A \otimes C))$ satisfying

$$rep_B\,;(p \lhd q) \;\sqsubseteq\; (p' \lhd q')\,;rep_{A\otimes C} \quad , \tag{17}$$

which is depicted



Define $rep_{A\otimes C} = rep_A \otimes rep_C$. It is a comap since $\otimes$ preserves comaps. The proof is concluded by calculating

$$\begin{array}{ll}
& (17) & \\
\equiv & & \ldots \text{def } rep_{A\otimes C} \\
& rep_B \,;\, (p \lhd q) \sqsubseteq (p' \lhd q') \,;\, (rep_A \otimes rep_C) & \ldots (3),\ rep_B \text{ comap} \\
\Leftarrow & & \ldots (4) \\
& rep_B \,;\, p \lhd rep_B \,;\, q \sqsubseteq (p' \lhd q') \,;\, (rep_A \otimes rep_C) & \ldots \lhd \text{ monotonic} \\
\Leftarrow & & \\
& (16) &
\end{array}$$

□

Theorem 2. Given comaps $rep_A \in \mathbf{Prog}(A', A)$ and $rep_B \in \mathbf{Prog}(B', B)$, $\pi_{A,B}$ is data refined by $rep_{A\otimes B}$ to $\pi_{A',B'}$.

Proof.

$$\begin{array}{ll}
& rep_{A\otimes B} \,;\, \pi_{A,B} \sqsubseteq \pi_{A',B'} \,;\, rep_A & \ldots \text{def } rep_{A\otimes B} \\
\equiv & & \\
& (rep_A \otimes rep_B) \,;\, \pi_{A,B} \sqsubseteq \pi_{A',B'} \,;\, rep_A & \ldots (5),\ rep_A \text{ and } rep_B \text{ comaps} \\
\Leftarrow & & \\
& \pi_{A',B'} \,;\, rep_A \sqsubseteq \pi_{A',B'} \,;\, rep_A &
\end{array}$$

□

4. Weak Exponent

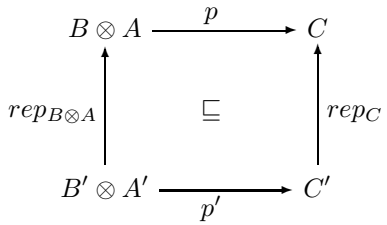
First we show that cur preserves data refinement. Given

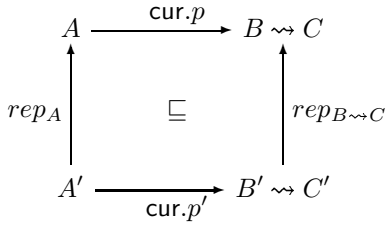
$$rep_{B\otimes A} \,;\, p \sqsubseteq p' \,;\, rep_C \quad , \tag{18}$$

we need to define $rep_{B\rightsquigarrow C} \in \mathbf{Prog}((B' \rightsquigarrow C'), (B \rightsquigarrow C))$ such that

$$rep_A \,;\, \mathsf{cur}.p \sqsubseteq \mathsf{cur}.p' \,;\, rep_{B\rightsquigarrow C} \quad . \tag{19}$$

Here are the pictures.





We use the well-known technique of applying a contravariant function to a map, defining $rep_{B\rightsquigarrow C}$ by

$$rep_{B\rightsquigarrow C} = (B' \rightsquigarrow rep_C) \,;\, (rep_B^\circ \rightsquigarrow C) \quad .$$

Observe that (19) follows from (18) because

$$
\begin{array}{ll}
& (19) & \\
\equiv & & \ldots \text{def } rep_{B \rightsquigarrow C} \\
& rep_A \,;\, \mathsf{cur}.p \sqsubseteq \mathsf{cur}.p' \,;\, (B' \rightsquigarrow rep_C) \,;\, (rep_B^\circ \rightsquigarrow C) & \\
\Leftarrow & & \ldots (8) \\
& rep_A \,;\, \mathsf{cur}.p \sqsubseteq \mathsf{cur}.(p' \,;\, rep_C) \,;\, (rep_B^\circ \rightsquigarrow C) & \\
\equiv & & \ldots (2),\ rep_A \text{ comap} \\
& \mathsf{cur}.p \sqsubseteq rep_A^\circ \,;\, \mathsf{cur}.(p' \,;\, rep_C) \,;\, (rep_B^\circ \rightsquigarrow C) & \\
\Leftarrow & & \ldots (9),\ rep_A^\circ \text{ map} \\
& \mathsf{cur}.p \sqsubseteq \mathsf{cur}.((id \otimes rep_A^\circ) \,;\, p' \,;\, rep_C) \,;\, (rep_B^\circ \rightsquigarrow C) & \\
\Leftarrow & & \ldots (10) \\
& \mathsf{cur}.p \sqsubseteq \mathsf{cur}.((rep_B^\circ \otimes id) \,;\, (id \otimes rep_A^\circ) \,;\, p' \,;\, rep_C) & \\
\equiv & & \ldots (7),\ id, rep^\circ \text{ maps} \\
& \mathsf{cur}.p \sqsubseteq \mathsf{cur}.((rep_B^\circ \otimes rep_A^\circ) \,;\, p' \,;\, rep_C) & \\
\Leftarrow & & \ldots \mathsf{cur} \text{ monotonic} \\
& p \sqsubseteq (rep_B^\circ \otimes rep_A^\circ) \,;\, p' \,;\, rep_C & \\
\equiv & & \ldots \otimes \text{ preserves comaps} \\
& p \sqsubseteq (rep_B \otimes rep_A)^\circ \,;\, p' \,;\, rep_C & \\
\equiv & & \ldots (2), \text{ def } rep_{B \otimes A} \\
& (18) &
\end{array}
$$

It remains to show that $rep_{B \rightsquigarrow C}$ is a comap. We shall show that $(rep_B \rightsquigarrow rep_C^\circ)$ is its map. Note that for brevity we have switched to treating $\rightsquigarrow$ as a function of two arguments; in particular $rep_{B \rightsquigarrow C} = rep_B^\circ \rightsquigarrow rep_C$. For one conjunct of the comap property (1), we need $\rightsquigarrow$ to preserve identities; *i.e.* (13) needs to be strengthened to

$$id_B \rightsquigarrow id_X \;=\; id_{B \rightsquigarrow X} \quad . \tag{20}$$

This law holds in the model discussed below. For the first conjunct of the comap property we have

$$
\begin{array}{ll}
= & (rep_B \rightsquigarrow rep_C^\circ) \,;\, rep_{B \rightsquigarrow C} \qquad \ldots \text{def } rep_{B \rightsquigarrow C} \\
\sqsupseteq & (rep_B \rightsquigarrow rep_C^\circ) \,;\, (rep_B^\circ \rightsquigarrow rep_C) \qquad \ldots \ (11),(12) \\
\sqsupseteq & rep_B^\circ \,;\, rep_B \rightsquigarrow rep_C^\circ \,;\, rep_C \qquad \ldots rep \text{ comap}, \rightsquigarrow \text{ monotonic} \\
= & id_B \rightsquigarrow id_C \qquad \ldots \ (20) \\
& id_{B \rightsquigarrow C}
\end{array}
$$

Below we justify strengthening (11) and (12) to

$$f \,;\, g \rightsquigarrow h \,;\, k \;=\; (g \rightsquigarrow h) \,;\, (f \rightsquigarrow k) \quad , \tag{21}$$

which makes it possible to prove the other conjunct of the comap property:

$$
\begin{array}{ll}
= & rep_{B \rightsquigarrow C} \,;\, (rep_B \rightsquigarrow rep_C^\circ) \qquad \ldots \text{def } rep_{B \rightsquigarrow C} \\
= & (rep_B^\circ \rightsquigarrow rep_C) \,;\, (rep_B \rightsquigarrow rep_C^\circ) \qquad \ldots (21) \\
\sqsubseteq & rep_B \,;\, rep_B^\circ \rightsquigarrow rep_C \,;\, rep_C^\circ \qquad \ldots rep \text{ comap}, \rightsquigarrow \text{ monotonic} \\
\sqsubseteq & id \rightsquigarrow id \qquad \ldots (13) \\
& id
\end{array}
$$

Given (20) and (21), we have proved the following.

Theorem 3. cur preserves data refinement. □

Law (20) poses a significant difficulty; it does not hold —without excessively restricting $A \rightsquigarrow B$ (to include only deterministic, everywhere terminating

programs)— in the powerset model [Nau92a]. It does hold in a slightly richer model. Redefine **Prog** as follows. Let the objects be all partially ordered sets A, and let $\mathbb{P}.A$ denote the set of upward closed subsets.[6] Everything else remains the same. This category contains the original one as a full subcategory, because if A is ordered by equality then $\mathbb{P}.A$ is its powerset. But the data type $A \rightsquigarrow B$ can be ordered by $\sqsubseteq$, which permits the exponent to have slightly better properties (even though the definitions remain the same, see footnote 5). Predicates at higher types must be monotonic with respect to refinement of program type variables [Nau94b]; this is no restriction in practice, because the predicates of interest are pre/post specifications. The logic of monotonic predicates is intuitionistic (*e.g.* [Vic89]), but that does not obtrude in practice because negations of specifications are not needed in program derivations. All of the standard definitions and laws of refinement calculus carry over unchanged to the new setting [Nau94b]. It is shown elsewhere [Nau94a] that all the laws in the present paper hold, along with (20) and (21).

As for (21), it holds even in the powerset model provided that each hom-object $X \rightsquigarrow A$ has "enough" elements. One alternative is to take $X \rightsquigarrow A$ to be $\mathbf{Prog}(X, A)$ for each X, A. With this definition, (21) holds, as do two additional laws that are needed for the next theorem.

$$\mathsf{ap} \,;\, g \;=\; (id \otimes (B \rightsquigarrow g)) \,;\, \mathsf{ap} \tag{22}$$

$$(g \otimes id_{A \rightsquigarrow C}) \,;\, \mathsf{ap}_{A,C} \;=\; (id_B \otimes (g \rightsquigarrow C)) \,;\, \mathsf{ap}_{B,C} \tag{23}$$

(Both (22) and (23) hold as $lhs \sqsubseteq rhs$ even if $X \rightsquigarrow A$ does not have enough elements [Nau92a].)

Theorem 4. Given comaps $rep_B \in \mathbf{Prog}(B', B)$ and $rep_C \in \mathbf{Prog}(C', C)$, $\mathsf{ap}_{B,C}$ is data-refined by $rep_{B \otimes (B \rightsquigarrow C)}$ to $\mathsf{ap}_{B',C'}$.

Proof. Here $rep_{B \otimes (B \rightsquigarrow C)}$ is defined from rep_B and rep_C as in Theorems 1 and 3. We calculate:

[6] A subsets α of A is upward closed if $x \in \alpha$ and $x \leq y$ imply $y \in \alpha$.

$$
\begin{array}{ll}
 & rep_{B\otimes(B\rightsquigarrow C)}\,;\,\mathsf{ap}_{B,C} \\
= & \quad \ldots \text{def } rep_{B\otimes(B\rightsquigarrow C)} \\
 & (rep_B \otimes (B' \rightsquigarrow rep_C)\,;\,(rep_B^\circ \rightsquigarrow C))\,;\,\mathsf{ap} \\
\sqsubseteq & \quad \ldots \text{unit law, (6)} \\
 & (rep_B \otimes id)\,;\,(id \otimes (B' \rightsquigarrow rep_C))\,;\,(id \otimes (rep_B^\circ \rightsquigarrow C))\,;\,\mathsf{ap} \\
= & \quad \ldots (23) \\
 & (rep_B \otimes id)\,;\,(id \otimes (B' \rightsquigarrow rep_C))\,;\,(rep_B^\circ \otimes id)\,;\,\mathsf{ap} \\
= & \quad \ldots (7),\ rep_B^\circ \text{ and } id \text{ are maps} \\
 & (rep_B \otimes id)\,;\,(rep_B^\circ \otimes (B' \rightsquigarrow rep_C))\,;\,\mathsf{ap} \\
\sqsubseteq & \quad \ldots \text{unit law, (6)} \\
 & (rep_B \otimes id)\,;\,(rep_B^\circ \otimes id)\,;\,(id \otimes (B' \rightsquigarrow rep_C))\,;\,\mathsf{ap} \\
= & \quad \ldots (22) \\
 & (rep_B \otimes id)\,;\,(rep_B^\circ \otimes id)\,;\,\mathsf{ap}\,;\,rep_C \\
= & \quad \ldots id = id^\circ \\
 & (rep_B \otimes id)\,;\,(rep_B^\circ \otimes id^\circ)\,;\,\mathsf{ap}\,;\,rep_C \\
= & \quad \ldots \otimes \text{ preserves comaps} \\
 & (rep_B \otimes id)\,;\,(rep_B \otimes id)^\circ\,;\,\mathsf{ap}\,;\,rep_C \\
\sqsubseteq & \quad \ldots (1), \text{ unit law} \\
 & \mathsf{ap}\,;\,rep_C
\end{array}
$$

□

Because its action is defined in terms of cur, (;), and ap, the construct $\rightsquigarrow$ also preserves data refinement.

Acknowledgements

Thanks to Bob Tennant for inspiring me to write this paper, to the referees for suggestions that improved the presentation, and to Carroll Morgan for encouraging me to use **Prog** rather than **Tran**. Discussions with John Power have been very helpful. Paul Taylor's macros set the diagrams.

References

[Abr91] Samson Abramsky. Domain theory in logical form. *Annals of Pure and Applied Logic*, 51:1–77, 1991.

[Bac80] R.J.R. Back. Correctness preserving program refinements: Proof theory and applications. Technical Report Tract 131, CWI, 1980.

[BB94] R.J.R. Back and M.J. Butler. Exploring summation and product operators in the refinement calculus. Technical Report on Computer Science & Mathematics, Ser. A, No. 152, Abo Akademi, September 1994. Available from ftp://ftp.abo.fi/pub/cs/techreports/reports/A94-152.ps.Z.

[BK93] Marcello Bonsangue and Joost N. Kok. Isomorphisms between predicate and state transformers. In *MFCS*, pages 301–310. Springer LNCS 711, 1993.

[BP88] Renato Betti and A. John Power. On local adjointness of distributive bicategories. *Bolletino U.M.I.*, 7:931–947, 1988.

[CU89] Wei Chen and Jan Tijmen Udding. Towards a calculus of data refinement. In *Proceedings, Mathematics of Program Construction*, pages 197–218, 1989. LNCS 375.

[DM92] O. De Moor. Inductive data types for predicate transformers. *Information Processing Letters*, 43(3):113–118, 1992.

[GM91] Paul Gardiner and Carroll Morgan. Data refinement of predicate transformers. *Theoretical Computer Science*, 87:143–162, 1991.

[GMdM94] Paul H.B. Gardiner, Clare E. Martin, and Oege de Moor. An algebraic construction of predicate transformers. *Science of Computer Programming*, 22:21–44, 1994.

[HH90] C.A.R. Hoare and Jifeng He. Data refinement in a categorical setting. Technical Monograph PRG-PRG-90, November 1990.

[Hoa72] C.A.R. Hoare. Proofs of correctness of data representations. *Acta Informatica*, 1:271–281, 1972.

[Kel82] G.M. Kelly. *Basic Concepts of Enriched Category Theory*, volume 64 of *London Mathematical Society Lecture Notes*. Cambridge, 1982.

[KP94] Yoshiki Kinoshita and John Power. Enriched categories and data refinement. Technical Report 94-23, University of Edinburgh Department of Computer Science, 1994.

[Mar91] Clare E. Martin. Preordered categories and predicate transformers. Dissertation, Oxford University, 1991.

[Mit90] J.C. Mitchell. Type systems for programming languages. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 365–458. MIT Press/Elsevier, 1990.

[Mor89] Joseph M. Morris. Laws of data refinement. *Acta Informatica*, 26:287–308, 1989.

[Mor94] Carroll Morgan. *Programming from Specifications, second edition*. Prentice Hall, 1994.

[Nau92a] David A. Naumann. Predicate transformers and higher order programs. *Theoretical Computer Science*, 1992. To appear.

[Nau92b] David A. Naumann. Two-categories and program structure: Data types, refinement calculi, and predicate transformers. Dissertation, University of Texas at Austin, 1992.

[Nau94a] David A. Naumann. Categorical models of higher order imperative programming. Typescript, 1994.

[Nau94b] David A. Naumann. Predicate transformer semantics of an Oberon-like language. In Ernst-Rüdiger Olderog, editor, *Programming Concepts, Methods and Calculi*, IFIP Transactions A-56. Elsevier, 1994.

[Nau94c] David A. Naumann. A recursion theorem for predicate transformers on inductive data types. *Information Processing Letters*, 50:329–336, 1994.

[OT93] P.W. O'Hearn and R.D. Tennant. Relational parametricity and local variables (preliminary report). In *Proceedings, Twentieth POPL*, pages 171–184, 1993.

[Pie91] Benjamin C. Pierce. *Basic Category Theory for Computer Scientists*. MIT, 1991.

[Plo81] Gordon Plotkin. Post-graduate lecture notes in advanced domain theory. Typescript, 1981.

[Pow89] A.J. Power. An algebraic formulation for data refinement. In *Proceedings, 5th International Conf. on Mathematical Foundations of Programming Semantics*, pages 390–401, 1989. LNCS 442.

[Smy83] M.B. Smyth. Power domains and predicate transformers: A topological view. In *ICALP*, 1983. LNCS 154.

[Ten94] R.D. Tennant. Correctness of data representations in Algol-like languages. In A.W. Roscoe, editor, *A Classical Mind: Essays Dedicated to C.A.R. Hoare*. Prentice-Hall, 1994.

[Vic89] Steven Vickers. *Topology Via Logic*. Cambridge, 1989.