LFCS

Laboratory for Foundations of Computer Science
Department of Computer Science - The University of Edinburgh

Relational Parametricity and Local Variables
(Preliminary Report)

by

P W O'Hearn
R D Tennent

LFCS Report Series **ECS-LFCS-92-223**

LFCS **July 1992**
Department of Computer Science
University of Edinburgh
The King's Buildings
Edinburgh EH9 3JZ

Relational Parametricity and Local Variables

(Preliminary Report)

P. W. O'Hearn*
School of Computer and Information Science
Syracuse University
Syracuse, New York, U.S.A. 13244

R. D. Tennent†
Laboratory for Foundations of Computer Science
Department of Computer Science
University of Edinburgh
Edinburgh, Scotland, U.K. EH9 3JZ

July 16, 1992

Abstract

J. C. Reynolds has argued that Strachey's intuitive concept of "parametric" (i.e., uniform) polymorphism has essentially to do with *representation independence* in the programming of data representations, and demonstrated that logical relations could be used to formalize this principle in languages with type variables and user-defined types.

Here, we use relational parametricity to address long-standing problems with the semantics of local-variable declarations in Algol-like languages. The new model is based on a cartesian closed category of "relation-preserving" functors and natural transformations which is induced by a suitable category of "possible worlds" with relations assigned to its objects and morphisms. The semantic interpretation supports straightforward validations of all the test equivalences that have been proposed in the literature; however, it is not known whether it is fully abstract.

*This author was supported by NSF grant CCR-92110829.

†On leave from Queen's University, Kingston, Canada. This author was supported by an operating grant from the Natural Sciences and Engineering Research Council of Canada.

1 Introduction

One of the first things most programmers learn is how to "declare" a new assignable variable, as in the following Algol 60 block:

$$\begin{array}{l} \textbf{begin} \\ \quad \textbf{integer}\ z; z := 0; \\ \quad \cdots z := z + 1; \cdots \\ \textbf{end} \end{array}$$

It might be thought that a satisfactory semantic interpretation for such a fundamental and apparently elementary mechanism would be well established by now. But existing models have serious problems when free identifiers of higher-order type can appear within the bodies of such blocks [MS88; OT92]. Intuitively, the difficulty is in defining precisely the sense in which non-local entities are "independent" of locally-declared variables.

For example, consider the following block [MS88]:

$$\begin{array}{l} \textbf{begin} \\ \quad \textbf{integer}\ z; \\ \quad \textbf{procedure}\ inc; \\ \quad\quad z := z + 1; \\ \quad P(inc) \\ \textbf{end} \end{array}$$

Although the unknown non-local procedure P can use its argument to *change* the value of z, this value can never be *read*, and so the block should be equivalent (for all possible meanings of P) to $P(skip)$, where *skip* does nothing. But this equivalence fails in every published denotational model of local variables!

The reader's reaction to this example might be that it is contrived, and that this failure of full abstraction has no practical significance; after all, who would ever write such a program? But consider the following example, which is only slightly more complicated:

$$\begin{array}{l} \textbf{begin} \\ \quad \textbf{integer}\ z; \\ \quad \textbf{procedure}\ inc; \\ \quad\quad z := z + 1; \\ \quad \textbf{integer procedure}\ val; \\ \quad\quad val := z; \\ \quad z := 0; \\ \quad P(inc, val) \\ \textbf{end} \end{array}$$

The local variable, the two procedure declarations, and the initialization can be considered as constituting the concrete representation of an abstract "counter" object. Procedure P, the "client," is passed only the capabilities for incrementing and evaluating the counter, and cannot access the counter representation in any other way. In a "sugared" syntax, one might write

```
module counter(exports inc, val);
begin
  integer z;
  invariant z ≥ 0;
  procedure inc;
    z := z + 1;
  integer procedure val;
    val := z;
  z := 0
end counter;
... counter.inc; ... counter.val ...
```

but the unsugared form shows that the combination of local variables and procedures in Algol-like languages (without any additional features) is already sufficient to support a form of *representational abstraction*, which is one of the main themes of modern programming methodology. (In fact, the same example is used in the Appendix of [Rey78] to make the same point.)

To a certain extent, the relevance of representational abstraction to the semantics of local variables has already been exploited. The models described in [MS88, OT91] were designed to allow validation of *invariance* principles often used [Hoa72] for reasoning about data representations. For example, these models validate the following equivalence:

```
begin
  integer z;
  procedure inc;
    z := z + 1;
  integer procedure val;
    val := z;                    ≡     diverge
  z := 0;
  P(inc, val);
  if z ≥ 0 then diverge
end
```

Because P can be *any* procedure (of the appropriate type), the equivalence demonstrates that $z \geq 0$ is an invariant of the counter representation; i.e., $z \geq 0$ is true before and after every call of *inc* from P.

But there is more to representational abstraction than preservation of representation invariants. Consider the following block, which uses a "non-standard" representation of a counter:

```
begin
  integer z;
  procedure inc;
    z := z - 1;
  integer procedure val;
    val := -z;
  z := 0;
  P(inc, val)
end
```

This block should be equivalent to the block that uses the "standard" representation. The equivalence illustrates the principle of *representation independence*: one concrete representation of a data abstraction should be replaceable by another, provided the relevant *abstract* properties are preserved. It is clearly important to be able to validate this kind of equivalence (or other putative formalizations of representation independence), but existing semantic models of local variables fail on such equivalences. . .

The traditional approach to formalizing representation independence semantically is to use the algebraic concept of homomorphic function; however, as pointed out in [Rey83], this approach fails to deal with higher-order operations. To address this problem, Reynolds used the technique of "logical" (families of) relations for simply-typed lambda calculi [Plo73, Plo80, Sta85, Mit90], and showed how to generalize the technique to languages that support data abstraction using programmer-defined types; see [Rey75, Coo91] for comparisons of linguistic approaches to representational abstraction.

We can illustrate the representation-independence property provable using logical relations as follows:

$$\begin{array}{ccc} W & \quad & \llbracket\pi\rrbracket W \xrightarrow{\llbracket P\rrbracket W} \llbracket\theta\rrbracket W \\ R\updownarrow & & \llbracket\pi\rrbracket R \updownarrow \qquad\qquad \updownarrow \llbracket\theta\rrbracket R \\ W' & & \llbracket\pi\rrbracket W' \xrightarrow{\llbracket P\rrbracket W'} \llbracket\theta\rrbracket W' \end{array}$$

Here

- θ is a type expression with (say) one free type variable, and π is a typing context, i.e., a finite list of types over the same type variable;
- W and W' are sets, regarded as alternative "representations" of the type variable;
- $\llbracket\theta\rrbracket W$ is the set of meanings of type θ when W is assigned as the meaning of the type variable, and similarly for $\llbracket\theta\rrbracket W'$;
- $\llbracket\pi\rrbracket W$ is the set of π-compatible environments when W is assigned as the meaning of the type variable, and similarly for $\llbracket\pi\rrbracket W'$;
- $R \subseteq W \times W'$ is any relation on W and W';
- $\llbracket\theta\rrbracket R \subseteq \llbracket\theta\rrbracket W \times \llbracket\theta\rrbracket W'$ is the relation on θ-meanings "logically" induced by R, and similarly for $\llbracket\pi\rrbracket R \subseteq \llbracket\pi\rrbracket W \times \llbracket\pi\rrbracket W'$;
- P is a phrase of type θ in context π;
- $\llbracket P\rrbracket W$ is the meaning of P when W is assigned as the meaning of the type variable, and similarly for $\llbracket P\rrbracket W'$.

Then the diagram asserts that $\llbracket P\rrbracket W$ and $\llbracket P\rrbracket W'$ preserve the relevant relations; i.e., for all $u \in \llbracket\pi\rrbracket W$ and $u' \in \llbracket\pi\rrbracket W'$,

$$\text{if } u\big[\llbracket\pi\rrbracket R\big]u' \text{ then } \llbracket P\rrbracket W u\big[\llbracket\theta\rrbracket R\big]\llbracket P\rrbracket W' u'.$$

We will refer to this kind of property as an instance of *relational uniformity*.

In this work, we show that relational uniformity is also useful when representational abstraction is achieved using local variables and procedures in an Algol-like language, and, in particular, leads to an improved semantics of local variables.

We first review the functor-category approach to local variables pioneered by Reynolds and Oles [Rey81b, Ole82]. The general framework is as follows:

- types are interpreted as *functors* from a suitable category of "possible worlds" to a category of domains and continuous functions;
- phrases are interpreted as *natural transformations* of these functors; and
- the meaning of a declaration block at one world is defined in terms of the meaning of the body of the block at an "expanded" world where states have an additional component to hold the value of the locally-declared variable.

The naturality condition on phrase meanings is illustrated by the following commutative diagram:

$$\begin{tikzcd} W \arrow[d, "f"'] \\ W' \end{tikzcd} \qquad \begin{tikzcd} \llbracket\pi\rrbracket W \arrow[r, "\llbracket P\rrbracket W"] \arrow[d, "\llbracket\pi\rrbracket f"'] & \llbracket\theta\rrbracket W \arrow[d, "\llbracket\theta\rrbracket f"] \\ \llbracket\pi\rrbracket W' \arrow[r, "\llbracket P\rrbracket W'"] & \llbracket\theta\rrbracket W' \end{tikzcd}$$

where

- θ, π, and P are as before, but without any type variables;
- worlds W and W' are sets of states;
- $f\colon W \to W'$ is the "expansion" morphism from W to $W' = W \times Z$, for Z the set of integers;
- $\llbracket\theta\rrbracket W$ and $\llbracket\theta\rrbracket W'$ are the domains of θ-meanings appropriate to worlds W and W', respectively, and similarly for environment domains $\llbracket\pi\rrbracket W$ and $\llbracket\pi\rrbracket W'$;
- $\llbracket\pi\rrbracket f$ is the change of environment induced by f, and similarly for $\llbracket\theta\rrbracket f$; and
- $\llbracket P\rrbracket W$ and $\llbracket P\rrbracket W'$ are functions that interpret phrase P in worlds W and W', respectively.

In many respects, this is similar to the relational-uniformity picture discussed earlier. The key point is that $\llbracket P\rrbracket$ is, in a certain sense, polymorphic, and so it is possible to require relational uniformity *in addition to* naturality; this stronger uniformity requirement yields a significantly improved model of local variables.

To achieve this combination of relational uniformity and naturality, we will define suitable cartesian closed categories of "relation-preserving" functors and natural transformations. The key technical notion needed for this construction is that of a category equipped with assignments of (abstract) "relations" to its objects and morphisms; this is discussed in Section 2. Categories of relation-preserving functors and cartesian closure are treated in Section 3. Semantics of local variables and equivalences are discussed in Sections 4 and 5, respectively. In Section 6, a location-oriented semantics of variables is discussed.

2 Relations and Reflexive Graphs

It will suffice for our applications to work exclusively with *binary* relations. The definitions and results (though not some of the notation) generalize straightforwardly to n-ary relations for any n.

We begin with some preliminary definitions and notational conventions. If X and Y are sets and $R \subseteq X \times Y$, we write

- $x[R]y$ to mean $(x, y) \in R$.

If X is any set,

- $\delta_X \subseteq X \times X$ is the diagonal on X; i.e., $x[\delta_X]x' \iff x = x'$.

If W, W', X, and X' are sets and $R \subseteq W \times W'$ and $S \subseteq X \times X'$,

- $R \times S \subseteq (W \times X) \times (W' \times X')$ is defined by

$$(w, x)[R \times S](w', x') \iff w[R]w' \text{ and } x[S]x'$$

- $R \to S \subseteq (W \to X) \times (W' \to X')$ is defined by

$$f[R \to S]f' \iff \text{for all } w \in W,\ w' \in W', \text{ if } w[R]w' \text{ then } f(w)[S]f'(w')\,.$$

If D and E are partially-ordered sets and $R \subseteq D \times E$,

- $R_\perp \subseteq D_\perp \times E_\perp$ is defined by

$$d[R_\perp]e \iff d = e = \perp \text{ or } d[R]e$$

  where $D_\perp$ is obtained from D by adding a new least element $\perp$.

In the following, we will need functor-like maps that preserve a certain kind of relational structure. There is a serious difficulty, however. We do not want to insist on relations being *composable*, and so the structure that must be preserved is not really "categorical." One reason for not requiring composability is that, as is well known, composition is not preserved by logical relations at higher types. Another is that we want to be able to generalize to n-ary relations for $n > 2$, and then there is no evident notion of composition.

We propose that the appropriate way to describe the relational structure that *is* needed is to use a notion well-known to category-theorists, that of a *reflexive graph*. For our purposes, a reflexive graph is a (possibly large) collection of *vertices* with (directed) *edges* between them. We use the notations $R \in \mathcal{E}(w, x)$ and $R: w \leftrightarrow x$ to mean that R is an edge from vertex w to vertex x. Furthermore, for every vertex w, there is a distinguished edge $I_w: w \leftrightarrow w$ termed the *identity on* w. Notice that a reflexive graph is more structured than a set or class (because there are edges as well as vertices), but less structured than most categories (because edges are not necessarily composable).

We can now define a category $\mathbf{R}$ having all reflexive graphs as objects. (For reasons of size, this must be thought of as a "meta-category" [ML71] or "quasi-category" [HS79].) A morphism $F: G \to G'$ maps vertices to vertices and edges to edges such that

$F(R): F(w) \leftrightarrow F(x)$ in G' whenever $R: w \leftrightarrow x$ in G, and $F(I_x) = I_{F(x)}$. Composition and identities are obvious.

The structures in which we are interested can now be described succinctly as *internal* categories of $\mathbf{R}$ [BW90, AL91]. To assist readers not familiar with the concept of internal category, we will actually use an "external" description; an internal category of $\mathbf{R}$ can be thought of as consisting of an arbitrary conventional category $\mathbf{C}$, together with the following additional data to specify the reflexive-graph structure:

- for every pair w, x of $\mathbf{C}$-objects, a collection $\mathcal{E}(w,x)$, termed *edges from w to x*, and, for every $\mathbf{C}$-object w, a distinguished edge $I_w: w \leftrightarrow w$, termed the *identity on w*, (i.e., a reflexive graph whose vertices are the objects of $\mathbf{C}$), and
- for every pair $f: w \to x$, $f': w' \to x'$ of $\mathbf{C}$-morphisms, a family $\mathcal{E}(f, f')$ of pairs (R, S) of edges $R: w \leftrightarrow w'$ and $S: x \leftrightarrow x'$, satisfying certain conditions, as follows, where we use the relational-uniformity diagram $\begin{array}{ccc} w & \xrightarrow{f} & x \\ R\updownarrow & & \updownarrow S \\ w' & \xrightarrow[f']{} & x' \end{array}$ to mean that $(R, S) \in \mathcal{E}(f, f')$:

  - for every $\mathbf{C}$-morphism $f: w \to x$, $\begin{array}{ccc} w & \xrightarrow{f} & x \\ I_w\updownarrow & & \updownarrow I_x \\ w & \xrightarrow[f]{} & x \end{array}$;

  - if $R: w \leftrightarrow x$ then $\begin{array}{ccc} w & \xrightarrow{\mathrm{id}_w} & w \\ R\updownarrow & & \updownarrow R \\ x & \xrightarrow[\mathrm{id}_x]{} & x \end{array}$, where id_w is the identity morphism on w;

  and

  - if $\begin{array}{ccc} w & \xrightarrow{f} & x \\ R\updownarrow & & \updownarrow S \\ w' & \xrightarrow[f']{} & x' \end{array}$ and $\begin{array}{ccc} x & \xrightarrow{g} & y \\ S\updownarrow & & \updownarrow T \\ x' & \xrightarrow[g']{} & y' \end{array}$ then $\begin{array}{ccc} w & \xrightarrow{f\,;\,g} & y \\ R\updownarrow & & \updownarrow T \\ w' & \xrightarrow[f'\,;\,g']{} & y' \end{array}$, where $f\,;g$ is the composite of f and g in diagrammatic order.

Notice that relational-uniformity diagrams $\begin{array}{ccc} \cdot & \longrightarrow & \cdot \\ \updownarrow & & \updownarrow \\ \cdot & \longrightarrow & \cdot \end{array}$ are required by these conditions to have categorical structure in the horizontal dimension, but only reflexive-graph structure in the vertical dimension; cf. [KS74].

From now on, we refer to categories augmented with reflexive-graph structure as above simply as *internal categories* (leaving "of $\mathbf{R}$" implicit). As our first example of an internal category, we can add reflexive-graph structure to the usual category of sets and functions as follows.

- For every pair X, Y of sets, $\mathcal{E}(X, Y)$ is the set of all subsets of $X \times Y$.

- For every set X, $I_X = \delta_X$.
- For every pair $f: X \longrightarrow Y$, $f': X' \longrightarrow Y'$ of functions, $\mathcal{E}(f, f')$ is the set of pairs of relations $(R: X \leftrightarrow X', S: Y \leftrightarrow Y')$ such that $f[R \rightarrow S]f'$.

This internal category will be called **S**. Other internal categories are obtained by setting I_X to the full relation $X \times X$, or to the empty relation $\emptyset$; however, the diagonal leads to stronger uniformity conditions than the full relation, and the empty relation would not be preserved at higher types.

Similarly, we can add reflexive-graph structure to the category of pre-domains (i.e., directed-complete partially-ordered sets and continuous functions) as follows.

- For every pair D, E of pre-domains, $\mathcal{E}(D, E)$ is the set of all complete subsets of $D \times E$; i.e., $R \in \mathcal{E}(D, E)$ iff, for all directed subsets D_0 of D and E_0 of E, $d[R]e$ for all $d \in D_0$ and $e \in E_0$ implies $\bigsqcup D_0\, [R] \bigsqcup E_0$.
- For every pre-domain D, $I_D = \delta_D$, the diagonal on (the set underlying) D.
- For every pair $f: D \longrightarrow E$, $f': D' \longrightarrow E'$ of continuous functions, $\mathcal{E}(f, f')$ is the set of pairs of relations $(R: D \leftrightarrow D', S: E \leftrightarrow E')$ such that $f[R \rightarrow S]f'$.

This internal category will be called **D**. Another internal category is obtained by using the partial order on D as I_D.

Finally, we consider the category of "store shapes" described in [Ole82].

- The objects are (certain) sets, including desired data types, such as $\{\mathit{true}, \mathit{false}\}$ and $\{0, 1, 2, \ldots\}$, and all finite (set) products of these.
- The morphisms from W to X are all pairs (ϕ, ρ) such that
  - ϕ is a function from X to W;
  - ρ is a function from $W \times X$ to X, where the $\times$ here (and throughout this example) is the set-theoretic Cartesian product;
  - for all $x \in X$, $\rho(\phi(x), x) = x$;
  - for all $x \in X$ and $w \in W$, $\phi(\rho(w, x)) = w$;
  - for all $x, x' \in X$ and $w \in W$, $\rho(w, \rho(w', x)) = \rho(w, x)$.

  For example, W can be "expanded" to $X = W \times Z$ for some data type Z by morphism (ϕ, ρ) such that $\phi(w, z) = w$ and $\rho(\overline{w}, (w, z)) = (\overline{w}, z)$; i.e., ϕ "projects" a large stack into the small stack it contains, and ρ "replaces" the small stack contained in a large stack by a new small stack, leaving unchanged local variables on the larger stack. In fact, Oles shows that any $(\phi, \rho): W \longrightarrow X$ induces a set isomorphism $X \cong W \times Z$ for some non-empty set Z; that is, up to isomorphism, *every* morphism is an expansion.
- The composite of $(\phi, \rho): W \longrightarrow X$ and $(\phi', \rho'): X \longrightarrow Y$ is $(\phi'', \rho''): W \longrightarrow Y$ such that $\phi'' = \phi' \,;\, \phi$ and $\rho''(w, y) = \rho'(\rho(w, \phi'(y)), y)$.
- The identity morphism on W is (ϕ, ρ) such that $\phi(w) = w$ and $\rho(w, w') = w$.

We can add reflexive-graph structure appropriate to our application as follows.

- For every pair W, X of objects, $\mathcal{E}(W, X)$ is the set of all subsets of $W \times X$.
- For every object W, $I_W = \delta_W$.
- For every pair $(\phi, \rho): W \to X$ and $(\phi', \rho'): W' \to X'$ of morphisms, $\mathcal{E}\left((\phi, \rho), (\phi', \rho')\right)$ is the set of all edge pairs $(R: W \leftrightarrow W', S: X \leftrightarrow X')$ such that $\phi[S \to R]\phi'$ and $\rho[R \times S \to S]\rho'$.

This internal category will be called $\mathbf{W}$. The condition on edge pairs (R, S) is noteworthy:
$$\begin{tikzcd} W \arrow[r, "{(\phi,\rho)}"] \arrow[d, leftrightarrow, "R"'] & X \arrow[d, leftrightarrow, "S"] \\ W' \arrow[r, "{(\phi',\rho')}"'] & X' \end{tikzcd}$$
holds in $\mathbf{W}$ if and only if both
$$\begin{tikzcd} X \arrow[r, "\phi"] \arrow[d, leftrightarrow, "S"'] & W \arrow[d, leftrightarrow, "R"] \\ X' \arrow[r, "\phi'"'] & W' \end{tikzcd}$$
and
$$\begin{tikzcd} W \times X \arrow[r, "\rho"] \arrow[d, leftrightarrow, "R \times S"'] & X \arrow[d, leftrightarrow, "S"] \\ W' \times X' \arrow[r, "\rho'"'] & X' \end{tikzcd}$$
in $\mathbf{S}$. This definition ensures that appropriate relations will be preserved by variable de-allocation (using the "projections" ϕ) and by state changes in larger worlds induced by changes at smaller ones (using the "replacements" ρ). The methods described in [MS91, MR92] for assigning relational structure to categories do not seem to allow the flexibility available using internal categories in $\mathbf{R}$.

3 Internal Functors and Transformations

In this section, we describe "relation-preserving" functors and natural transformations for internal categories in $\mathbf{R}$, and show that certain analogues of functor categories are cartesian closed.

Let $\mathbf{X}$ and $\mathbf{Y}$ be internal categories; an *internal functor* from $\mathbf{X}$ to $\mathbf{Y}$ consists of

- a mapping F_0 from $\mathbf{X}$-objects to $\mathbf{Y}$-objects;
- a mapping F_1 from $\mathbf{X}$-morphisms to $\mathbf{Y}$-morphisms; and
- a mapping F_2 from edges on $\mathbf{X}$-objects to edges on $\mathbf{Y}$-objects

such that

- if $f: x \longrightarrow x'$ in $\mathbf{X}$ then $F_1(f): F_0(x) \longrightarrow F_0(x')$ in $\mathbf{Y}$;
- $F_1(\mathrm{id}_x) = \mathrm{id}_{F_0(x)}$ for every $\mathbf{X}$-object x;
- $F_1(f\,;g) = F_1(f)\,;F_1(g)$ for all composable $\mathbf{X}$-morphisms f and g;
- if $R: w \leftrightarrow x$ in $\mathbf{X}$, then $F_2(R): F_0(w) \longleftrightarrow F_0(x)$ in $\mathbf{Y}$;
- $F_2(I_x) = I_{F_0(x)}$, for every $\mathbf{X}$-object x; and
- if $\begin{tikzcd} w \arrow[r, "f"] \arrow[d, leftrightarrow, "R"'] & x \arrow[d, leftrightarrow, "S"] \\ w' \arrow[r, "f'"'] & x' \end{tikzcd}$ in $\mathbf{X}$, then $\begin{tikzcd} F_0(w) \arrow[r, "F_1(f)"] \arrow[d, leftrightarrow, "F_2(R)"'] & F_0(x) \arrow[d, leftrightarrow, "F_2(S)"] \\ F_0(w') \arrow[r, "F_1(f')"'] & F_0(x') \end{tikzcd}$ in $\mathbf{Y}$.

The first three conditions say that F_0 and F_1 constitute a conventional functor; the next two conditions say that F_0 and F_2 constitute a "relator" [MS91, AJ91]; and the last condition ensures that the morphism part is uniform relative to the relation part. We will use the notation $F: \mathbf{X} \longrightarrow \mathbf{Y}$ to mean that F is an internal functor from $\mathbf{X}$ to $\mathbf{Y}$, and adopt the usual notational abuse of using the symbol F to denote all three mappings.

The above is, of course, an external description of the standard notion of internal functor in $\mathbf{R}$. Similarly, the following is an external description of another standard notion of internal category theory (for internal categories of $\mathbf{R}$): for $F, G: \mathbf{X} \longrightarrow \mathbf{Y}$, η is an *internal natural transformation* from F to G (notation $\eta: F \dot{\longrightarrow} G$) if it maps $\mathbf{X}$-objects to $\mathbf{Y}$-morphisms such that

- for every $\mathbf{X}$-object x, $\eta(x): F(x) \longrightarrow G(x)$;
- for every $\mathbf{X}$-morphism $f: w \longrightarrow x$, $\begin{tikzcd} F(w) \arrow[r, "\eta(w)"] \arrow[d, "F(f)"'] & G(w) \arrow[d, "G(f)"] \\ F(x) \arrow[r, "\eta(x)"'] & G(x) \end{tikzcd}$ commutes; and

- for every $R: w \leftrightarrow x$ in $\mathbf{X}$, $$\begin{tikzcd} F(w) \arrow[r, "\eta(w)"] \arrow[d, leftrightarrow, "F(R)"'] & G(w) \arrow[d, leftrightarrow, "G(R)"] \\ F(x) \arrow[r, "\eta(x)"'] & G(x) \end{tikzcd}$$ in $\mathbf{Y}$.

The first two conditions say that η is a conventional natural transformation (of the relevant components of F and G), and the last condition is a relational-uniformity requirement.

Internal natural transformations compose (in the morphism direction only) in the obvious point-wise way, with identities $\mathrm{id}_F(x) = \mathrm{id}_{F(x)}$ for all $x \in \mathbf{X}$. If $\mathbf{X}$ is any small internal category (the collection of morphisms is small), and $\mathbf{Y}$ is any internal category, we define $\mathbf{X} \Rightarrow \mathbf{Y}$ as the following internal category:

- the objects are all internal functors from $\mathbf{X}$ to $\mathbf{Y}$;
- the morphisms are all internal natural transformations;
- for every $F, G: \mathbf{X} \longrightarrow \mathbf{Y}$, $\mathcal{E}(F, G) = \prod_{x \in \mathbf{X}} \mathcal{E}(F(x), G(x))$, with $I_F(x) = I_{F(x)}$; and
- for every $\eta: F \stackrel{\cdot}{\longrightarrow} G$ and $\eta': F' \stackrel{\cdot}{\longrightarrow} G'$, $\mathcal{E}(\eta, \eta')$ is the set of edge pairs $(R: F \leftrightarrow F', S: G \leftrightarrow G')$ such that, for all $x \in \mathbf{X}$, $$\begin{tikzcd} F(x) \arrow[r, "\eta(x)"] \arrow[d, leftrightarrow, "R(x)"'] & G(x) \arrow[d, leftrightarrow, "S(x)"] \\ F'(x) \arrow[r, "\eta'(x)"'] & G'(x) \end{tikzcd}$$.

We are particularly interested in categories of this form when $\mathbf{Y}$ is $\mathbf{S}$ or $\mathbf{D}$, the previously-defined internal categories of sets and pre-domains, respectively.

Proposition 1 *For $\boldsymbol{X}$ any small internal category, the categories underlying $\boldsymbol{X} \Rightarrow \boldsymbol{S}$ and $\boldsymbol{X} \Rightarrow \boldsymbol{D}$ are cartesian closed.*

Products can be defined pointwise:

$$\begin{aligned} (F \times G)(w) &= F(w) \times G(w) \\ (F \times G)(f) &= F(f) \times G(f) \\ (F \times G)(R) &= F(R) \times G(R) \end{aligned}$$

with the obvious (relationally-uniform) projections. For the exponential, $(F \longrightarrow G)(w)$ is the set (or pointwise-ordered pre-domain) of all families $p \in \prod_{f: w \to x} F(x) \longrightarrow G(x)$ of functions satisfying both the usual *naturality* condition that

$$\begin{tikzcd} F(x) \arrow[r, "p(f)"] \arrow[d, "F(g)"'] & G(x) \arrow[d, "G(g)"] \\ F(y) \arrow[r, "p(f\,;g)"] & G(y) \end{tikzcd}$$

commutes for all $f\colon w \longrightarrow x$ and $g\colon x \longrightarrow y$, and also the following *relational-uniformity* condition:

$$\text{if } \begin{tikzcd} w \arrow[r, "f"] \arrow[d, leftrightarrow, "I_w"'] & x \arrow[d, leftrightarrow, "R"] \\ w \arrow[r, "g"'] & y \end{tikzcd} \text{ in } \mathbf{X} \text{ then } \begin{tikzcd} F(x) \arrow[r, "p(f)"] \arrow[d, leftrightarrow, "F(R)"'] & G(x) \arrow[d, leftrightarrow, "G(R)"] \\ F(y) \arrow[r, "p(g)"'] & G(y) \end{tikzcd} \text{ in } \mathbf{S} \text{ or in } \mathbf{D}.$$

The morphism part of $F \longrightarrow G$ is defined by

$$(F \longrightarrow G)(f)(p)(g) = p(f\,;g)$$

and the relation part is as follows: $p\big[(F \longrightarrow G)(R\colon w \leftrightarrow w')\big]p'$ iff, for all $f\colon w \longrightarrow x$, $f'\colon w' \longrightarrow x'$, and $S\colon x \leftrightarrow x'$,

$$\text{if } \begin{tikzcd} w \arrow[r, "f"] \arrow[d, leftrightarrow, "R"'] & x \arrow[d, leftrightarrow, "S"] \\ w' \arrow[r, "f'"'] & x' \end{tikzcd} \text{ then } p(f)\big[F(S) \rightarrow G(S)\big]p'(f').$$

Application and currying are defined as in conventional pre-sheaf categories; the necessary relational-uniformity properties are easily proved. □

It is noteworthy that a relational-uniformity constraint arises in the definition of the exponential. The explanation for this "internalization" is that even non-definable procedure meanings must be constrained if procedure applications in phrases are to satisfy the appropriate relational-uniformity condition. This is strongly reminiscent of the use of a parametricity condition to constrain values of $\forall$ types in [Rey83]. In fact, the identity edge I_w in the condition

$$\text{if } \begin{tikzcd} w \arrow[r, "f"] \arrow[d, leftrightarrow, "I_w"'] & x \arrow[d, leftrightarrow, "R"] \\ w \arrow[r, "g"'] & y \end{tikzcd} \text{ in } \mathbf{X} \text{ then } \begin{tikzcd} F(x) \arrow[r, "p(f)"] \arrow[d, leftrightarrow, "F(R)"'] & G(x) \arrow[d, leftrightarrow, "G(R)"] \\ F(y) \arrow[r, "p(g)"'] & G(y) \end{tikzcd} \text{ in } \mathbf{S} \text{ or in } \mathbf{D}.$$

plays a role similar to the identity relations there. (Of course, the foundational difficulties described in [Rey83, Rey84, RP90] do not arise here, because the source category $\mathbf{X}$, over which indexing is done, will always be small.)

The use of squares of the form $\begin{tikzcd} w \arrow[r, "f"] \arrow[d, leftrightarrow, "I_w"'] & x \arrow[d, leftrightarrow, "R"] \\ w \arrow[r, "g"'] & y \end{tikzcd}$ to constrain the R's that can be used when applying the relational-uniformity property is very important in practice. For example, when $\mathbf{X}$ is the internal category $\mathbf{W}$ of store shapes, and f and g are both the expansion from set of states W to $W \times Z$, this condition requires R to be a relation of the form $\delta_W \times Q$, for some $Q \subseteq Z \times Z$. If we were able to use arbitrary

R's, the requirement
$$\begin{tikzcd} F(x) \arrow[r, "p(f)"] \arrow[d, leftrightarrow, "F(R)"'] & G(x) \arrow[d, leftrightarrow, "G(R)"] \\ F(y) \arrow[r, "p(g)"'] & G(y) \end{tikzcd}$$
would be *too* restrictive, leading to nonsensical requirements such as, e.g., that procedure calls of the form $P(\mathbf{skip})$ never change the values of any variables.

4 Semantics

As basic phrase types, we take **exp** (expressions), **comm** (commands), and **var** (storage variables); for simplicity, all storable values will be integers. We interpret types and phrases as objects and morphisms, respectively, of $\mathbf{W} \Rightarrow \mathbf{D}$, where $\mathbf{W}$ is the internal category of store shapes.

For simplicity, the objects of expression and command meanings can be defined pointwise, as in [Ole82]. For expressions:

- for every $\mathbf{W}$-object W,

  $$[\![\mathbf{exp}]\!]W = W \rightarrow Z_{\perp} ,$$

  where Z is the set of integers;

- for every $\mathbf{W}$-morphism $(\phi, \rho): W \rightarrow X$ and $e \in [\![\mathbf{exp}]\!]W$,

  $$[\![\mathbf{exp}]\!](\phi, \rho)\; e = \phi\,; e ,$$

  and

- for every $R: W \leftrightarrow W'$,

  $$[\![\mathbf{exp}]\!]R = R \rightarrow (\delta_Z)_{\perp} .$$

For commands:

- for every $\mathbf{W}$-object W,

  $$[\![\mathbf{comm}]\!]W = W \rightarrow W_{\perp} ;$$

- for every $\mathbf{W}$-morphism $(\phi, \rho): W \rightarrow X$, $c \in [\![\mathbf{comm}]\!]W$ and $x \in X$,

  $$[\![\mathbf{comm}]\!](\phi, \rho)\; c\; x = \begin{cases} \rho(c(\phi(x)), x), & \text{if } c(\phi(x)) \neq \perp \\ \perp, & \text{if } c(\phi(x)) = \perp \end{cases}$$

  and

- for every $R: W \leftrightarrow W'$,

  $$[\![\mathbf{comm}]\!]R = R \rightarrow R_{\perp} .$$

The relational-uniformity conditions are easily verified. It is noteworthy that these pointwise definitions are actually isomorphic to what is obtained by introducing the obvious contravariant "states" functor S and defining

$$[\![\mathbf{exp}]\!] = S \rightarrow \Delta Z_{\perp}$$

$$[\![\mathbf{comm}]\!] = S \rightarrow S_{\perp}$$

using an *internal* version of "contra-exponentiation" [OT92], where ΔD is the constant internal functor whose object, morphism, and edge parts always yield D, id_D, and δ_D, respectively. This is already an indication of the effectiveness of the relational-uniformity constraints.

For storage variables, we can use the exponentiation and product in $\mathbf{W} \Rightarrow \mathbf{D}$ as follows:

$$[\![\mathbf{var}]\!] = \left(\Delta Z \longrightarrow [\![\mathbf{comm}]\!]\right) \times [\![\mathbf{exp}]\!]$$

The two factors allow for, respectively, updating and accessing the current value of a variable. This "object-oriented" approach to storage variables is discussed in [Rey81b].

Finally, for procedures,

$$[\![\theta \longrightarrow \theta']\!] = [\![\theta]\!] \longrightarrow [\![\theta']\!] ,$$

where the $\longrightarrow$ on the right-hand side is the exponentiation in $\mathbf{W} \Rightarrow \mathbf{D}$.

The interpretation of phrases as internal natural transformations is mostly straightforward. For example, the usual assignment command can be interpreted by

$$\mathit{assign}\colon [\![\mathbf{var}]\!] \times [\![\mathbf{exp}]\!] \stackrel{\cdot}{\longrightarrow} [\![\mathbf{comm}]\!]$$

defined as follows:

$$\mathit{assign}\, W\Big((a, e), e'\Big)(s) = \begin{cases} a(\mathrm{id}_W)(z)(s), & \text{if } e'(s) = z \\ \bot, & \text{if } e'(s) = \bot \end{cases}$$

for any world W, $(a, e) \in [\![\mathbf{var}]\!]W$, $e' \in [\![\mathbf{exp}]\!]W$, and $s \in W$. See [Ole82, Ten90, Ten91, OT92] for other examples.

We will define

$$\mathit{new}\colon [\![\mathbf{var} \longrightarrow \mathbf{comm}]\!] \stackrel{\cdot}{\longrightarrow} [\![\mathbf{comm}]\!]$$

to indicate how the variable-declaration block can be treated. For $\mathbf{W}$-object W, $p \in [\![\mathbf{var} \longrightarrow \mathbf{comm}]\!]W$ and $w \in W$,

$$\mathit{new}\ W\ p\ w = \begin{cases} w', & \text{if } p(f)(a, e)(w, z_0) = (w', z') \\ \bot, & \text{if } p(f)(a, e)(w, z_0) = \bot \end{cases}$$

where $f\colon W \longrightarrow W \times Z$ is an "expansion" morphism in $\mathbf{W}$, $z_0 \in Z$ is the standard initial value of new variables, and $(a, e) \in [\![\mathbf{var}]\!](W \times Z)$ is the new variable, defined as follows: $a(\mathrm{id}_{W \times Z})(\overline{z})(w, z) = (w, \overline{z})$ and $e(w, z) = z$. (Only the "identity" component of a needs to be specified; the other components are determined by naturality.) The relational uniformity of $a \in (\Delta Z \longrightarrow [\![\mathbf{comm}]\!])(W \times Z)$ follows directly from the relational uniformity of $[\![\mathbf{comm}]\!]$.

To see the relational uniformity of new, consider any worlds W and W', relation $R\colon W \leftrightarrow W'$, functions $p \in [\![\mathbf{var} \longrightarrow \mathbf{comm}]\!]W$ and $p' \in [\![\mathbf{var} \longrightarrow \mathbf{comm}]\!]W'$ such that $p\big[[\![\mathbf{var} \longrightarrow \mathbf{comm}]\!]R\big]p'$, and states $w \in W$ and $w' \in W'$ such that $w[R]w'$. We must show that $\mathit{new}\, W\, p\, w\big[R_\bot\big]\mathit{new}\, W'\, p'\, w'$. It can be verified that

$$\begin{tikzcd} W \arrow[r, "f"] \arrow[d, leftrightarrow, "R"'] & W \times Z \arrow[d, leftrightarrow, "R \times \delta_Z"] \\ W' \arrow[r, "f'"'] & W' \times Z \end{tikzcd}$$

where $f\colon W \longrightarrow W \times Z$ and $f'\colon W' \longrightarrow W' \times Z$ are expansions in $\mathbf{W}$; hence,

$$p(f)\big[[\![\mathbf{var}]\!](R \times \delta_Z) \longrightarrow [\![\mathbf{comm}]\!](R \times \delta_Z)\big]p'(f') .$$

Furthermore, it can be verified that $(a,e)\Big[[\![\mathbf{var}]\!](R \times \delta_Z)\Big](a',e')$ for the new variables $(a,e) \in [\![\mathbf{var}]\!](W \times Z)$ and $(a',e') \in [\![\mathbf{var}]\!](W' \times Z)$, and so

$$p(f)(a,e)(w,z_0)\Big[(R \times \delta_Z)_\perp\Big]p'(f')(a',e')(w',z_0)\ ,$$

and this ensures that $new\, W\ p\ w\Big[R_\perp\Big]new\, W'\ p'\ w'$.

5 Equivalences

In this section we demonstrate the validity of the equivalences discussed in the Introduction, and others that have appeared in the literature [MS88, OT92].

We begin by describing a class of relations that can be used in several examples. Suppose W is any world and $E \subseteq Z$, where, as before, Z is the set of integers; we can then define $R_E : W \longleftrightarrow W \times Z$ by

$$w[R_E](w', z) \iff w = w' \text{ and } z \in E \, .$$

It can be verified that

$$\begin{tikzcd} W \arrow[r, "\mathrm{id}_W"] \arrow[d, leftrightarrow, "I_W"'] & W \arrow[d, leftrightarrow, "R_E"] \\ W \arrow[r, "f"'] & W \times Z \end{tikzcd}$$

where $f\colon W \to W \times Z$ is an expansion and id_W is an identity morphism in $\mathbf{W}$. It may be recalled that this kind of property is a necessary prerequisite to applying the relational-uniformity condition on procedures.

Suppose $c \in [\![\mathbf{comm}]\!](W \times Z)$ is such that $\mathit{skip}\big[[\![\mathbf{comm}]\!]R_E\big]c$, where $\mathit{skip} \in [\![\mathbf{comm}]\!]W$ is defined by $\mathit{skip}(w) = w$. Then, for any $p \in [\![\mathbf{comm} \to \mathbf{comm}]\!]W$, relational uniformity implies that

$$(*) \qquad p(\mathrm{id}_W)(\mathit{skip})\big[[\![\mathbf{comm}]\!]R_E\big]p(f)(c)$$

We can use this condition whenever we have a command c that does not change the values of non-local variables and preserves property E of the local variable.

For example, consider the first equivalence discussed in the Introduction. To validate this, we can use the relation R_Z; i.e.,

$$w[R_Z](w', z) \iff w = w' \, .$$

Intuitively, entities will be R_Z-related if they "work the same way" on the W part of the stack. This is a property of $x := x + 1$ and **skip**; more precisely, if we define $\mathit{inc} \in [\![\mathbf{comm}]\!](W \times Z)$ by $\mathit{inc}(w, z) = (w, z + 1)$, then $\mathit{skip}\big[[\![\mathbf{comm}]\!]R_Z\big]\mathit{inc}$. Then we can use the property $(*)$ to conclude

$$p(\mathrm{id}_W)(\mathit{skip})w\big[(R_Z)_\perp\big]p(f)(\mathit{inc})(w, z) \, .$$

This means that either both of these procedure calls must yield $\perp$, or both must terminate with the first component of $p(f)(\mathit{inc})(w, z)$ equal to $p(\mathrm{id}_W)(\mathit{skip})w$. Clearly, then, the semantics of variable declarations ensures the desired equivalence:

$$\begin{array}{lcl} \mathbf{begin} & & \\ \quad \mathbf{integer}\ z; & & \\ \quad \mathbf{procedure}\ \mathit{inc};\ z := z + 1; & \equiv & P(\mathbf{skip}) \\ \quad P(\mathit{inc}) & & \\ \mathbf{end} & & \end{array}$$

Our second example demonstrates that the invariant-preserving properties of the models described in [MS88, OT91] are encompassed by relational uniformity. If $Z^{\oplus}$ is the set of *nonnegative* integers, we again get $skip\left[[\![\mathbf{comm}]\!]R_{Z^{\oplus}}\right]inc$. The property (*) now ensures that z is non-negative when $p(f)(inc)(w,0) = (w', z)$. This can be used to verify that the value of local variable z is still nonnegative on termination of the procedure call in

begin
  integer z;
  procedure inc; $z : z + 1$;
  $z := 0$;
  $P(inc)$;
  ...
end

Our last example using relations of the form R_E is

$$\begin{array}{lcl} \textbf{begin} & & \\ \quad \textbf{integer}\ z; z := 0; & \equiv & P(0) \\ \quad P(z) & & \\ \textbf{end} & & \end{array}$$

where $P\colon \mathbf{exp} \longrightarrow \mathbf{comm}$; we have assumed a de-referencing coercion from **var** to **exp** in the argument of the call. The intuition here is that the value of z will be 0 each time it is used during execution of the call $P(z)$, because P cannot *write to* z. Therefore, this should be equivalent to simply supplying 0 as an argument instead of z.

To validate this we can use $R_{\{0\}}$; we note that (the denotations of) 0 and z are related by $[\![\mathbf{exp}]\!]R_{\{0\}}$, and then use the relational uniformity of P, as in the other examples. (The denotation of 0 is the constantly 0 function in $[\![\mathbf{exp}]\!](W \times Z)$, and the denotation of z, as an expression, is the projection in $W \times Z \rightarrow Z$.)

Finally, we consider a relation that does not fit into the R_E pattern: $R\colon (W \times Z) \longleftrightarrow (W \times Z)$, defined by

$$(w,z)[R](w',z') \iff w = w' \text{ and } z \geq 0 \text{ and } z' = -z.$$

This can be used to validate the equivalence discussed in the Introduction between blocks that use non-negative and non-positive implementations of a counter. The representations are directly related and then relational uniformity of procedures is used, as in the previous examples. This technique is essentially similar to the the method for showing correctness of data representations described in [Hoa72]; see also Chapter 5 of [Rey81a].

6 Locations

The traditional approach to storage variables in denotational semantics is to use "locations." For a location-oriented semantics of variables, an appropriate category of worlds is as follows: the objects are natural numbers (the number of variables on the run-time stack) and the morphisms from n to m are the injective functions from $\{0,1,2,\ldots,n-1\}$ to $\{0,1,2,\ldots,m-1\}$.

As many authors have suggested [Hoa75, Don77, Rey81b, Bro85], locations provide a "low-level" semantics for variables. It is therefore not surprising that adding appropriate reflexive-graph structure to this category is rather complicated. Technically, it would be possible to consider the edges between worlds n and m simply as (certain) subsets of $\{0,1,2,\ldots,n-1\} \times \{0,1,2,\ldots,m-1\}$; but the resulting relational-uniformity conditions would be rather weak for a language with states. We allow for relations on states as well as on locations, and these will be suitably constrained to ensure the relational uniformity of the relevant primitives. The reflexive-graph structure is as follows.

- For every pair n, m of natural numbers, $\mathcal{E}(n,m)$ is the set of all pairs (R_0, R_1) such that
  - $R_0 \subseteq L_n \times L_m$, where, for every i, $L_i = \{0,1,2,\ldots,i-1\}$ is the set of locations in world i;
  - $R_1 \subseteq S_n \times S_m$, where, for every i, $S_i = L_i \rightarrow Z$ is the set of storage states in world i;
  - for all $i, \bar{\imath} \in L_n$ and $j, \bar{\jmath} \in L_m$,
  $$\text{if } i[R_0]j \text{ and } \bar{\imath}[R_0]\bar{\jmath} \text{ then } i = \bar{\imath} \iff j = \bar{\jmath};$$
  - for all $w \in S_n$, $x \in S_m$, $i \in L_n$, $j \in L_m$ and $z \in Z$,
  $$\text{if } w[R_1]x \text{ and } i[R_0]j \text{ then } (w \mid i \mapsto z)[R_1](x \mid j \mapsto z),$$
  where $(w \mid i \mapsto z)$ is the state like w except that it maps i to z; and
  - for all $w \in S_n$, $x \in S_m$, $i \in L_n$ and $j \in L_m$,
  $$\text{if } w[R_1]x \text{ and } i[R_0]j \text{ then } w(i) = x(j).$$
- For every object n, $I_n = (\delta_{L_n}, \delta_{S_n})$.
- For every pair $f: n \rightarrow m$ and $f': n' \rightarrow m'$ of morphisms (injective functions), $\mathcal{E}(f, f')$ is the set of all edge pairs
  $$\big((R_0, R_1): n \longleftrightarrow n', (T_0, T_1): m \longleftrightarrow m'\big)$$
  such that
  - $f[R_0 \rightarrow T_0]f'$;
  - for all $j \in L_m$ and $j' \in L_{m'}$
  $$\text{if } j[T_0]j' \text{ then } j \in f(L_n) \iff j' \in f'(L_{n'});$$

- $\phi_f[T_1 \rightarrow R_1]\phi'_f$; and
- $\rho_f[R_1 \times T_1 \rightarrow T_1]\rho'_f$,

where $\phi_f: S_m \rightarrow S_n$ and $\rho_f: S_n \times S_m \rightarrow S_m$ are defined from f by

$$\phi_f(x) = f\,;x$$

and

$$\rho_f(w,x)(i) = \begin{cases} w(j), & \text{if } i = f(j) \\ x(i), & \text{otherwise,} \end{cases}$$

and $\phi'_f: S'_m \rightarrow S'_n$ and $\rho'_f: S'_n \times S'_m \rightarrow S'_m$ are similarly defined from f'.

The third condition on (R_0, R_1) says that R_0 must be single-valued and injective, i.e., given by a span $n \leftarrow k \rightarrow m$ of morphisms between worlds. This is needed to ensure the relational uniformity of an operation for location-equality testing. The last two conditions ensure the relational uniformity of operations for updating and accessing of state components.

The first, third, and fourth conditions on $\mathcal{E}(f, f')$ say, respectively, that f and f' map related locations to related locations, and that the induced "projection" and "replacement" maps are relation-preserving, as in the internal state-set category of worlds. The second condition is perhaps less expected. It says that newly allocated locations, i.e., those not in the image of f or f', cannot be related to old ones. If we were to omit this condition, the relational-uniformity constraints would be *too* restrictive.

We now interpret the basic types as internal functors from this internal category to **D** as follows.

$$\begin{aligned}
[\![\mathbf{var}]\!]n &= L_n \\
[\![\mathbf{var}]\!]f &= f \\
[\![\mathbf{var}]\!](R_0, R_1) &= R_0 \\
[\![\mathbf{exp}]\!]n &= S_n \rightarrow Z_\perp \\
[\![\mathbf{exp}]\!]f\ e &= \phi_f\,; e \\
[\![\mathbf{exp}]\!](R_0, R_1) &= R_1 \rightarrow (\delta_Z)_\perp \\
[\![\mathbf{comm}]\!]n &= S_n \rightarrow (S_n)_\perp \\
[\![\mathbf{comm}]\!]f\ c\ x &= \begin{cases} \rho_f(c(\phi_f(x)), x), & \text{if } c(\phi_f(x)) \neq \perp \\ \perp, & \text{if } c(\phi_f(x)) = \perp \end{cases} \\
[\![\mathbf{comm}]\!](R_0, R_1) &= R_1 \rightarrow (R_1)_\perp
\end{aligned}$$

where L_n, S_n, ϕ_f and ρ_f are defined as discussed previously.

We now define internal natural transformations

$$\mathit{assign}: [\![\mathbf{var}]\!] \times [\![\mathbf{exp}]\!] \stackrel{\cdot}{\rightarrow} [\![\mathbf{comm}]\!]$$

$$\mathit{deref}: [\![\mathbf{var}]\!] \stackrel{\cdot}{\rightarrow} [\![\mathbf{exp}]\!]$$

$$new\colon [\![\mathbf{var} \rightarrow \mathbf{comm}]\!] \stackrel{\cdot}{\rightarrow} [\![\mathbf{comm}]\!]$$

$$eqloc\colon [\![\mathbf{var}]\!] \stackrel{\cdot}{\times} [\![\mathbf{var}]\!] \stackrel{\cdot}{\rightarrow} [\![\mathbf{exp}]\!]$$

as follows: for every natural number n,

- $assign(n)(i)(e)(w) \;=\; \begin{cases} (w \mid i \mapsto z), & \text{if } e(w) = z \\ \bot, & \text{if } e(w) = \bot \end{cases}$

where $i \in L_n$, $e \in [\![\mathbf{exp}]\!]n$ and $w \in S_n$;

- $deref(n)(i)(w) \;=\; w(i)$,

where $i \in L_n$ and $w \in S_n$;

- $new(n)(p)(w) \;=\; \begin{cases} f\,;\overline{w}, & \text{if } p(f)(n)(w \mid n \mapsto z_0) = \overline{w} \\ \bot, & \text{if } p(f)(n)(w \mid n \mapsto z_0) = \bot \end{cases}$

where $p \in [\![\mathbf{var} \rightarrow \mathbf{comm}]\!]n$, $w \in S_n$ and f is the inclusion of L_n in L_{n+1};.

- $eqloc(n)(i)(j)(w) \;=\; \begin{cases} 0, & \text{if } i \neq j \\ 1, & \text{if } i = j \end{cases}$

where $i, j \in L_n$ and $w \in S_n$. These can be used to interpret assignment, dereferencing, allocation, and location-equality testing, respectively. With these interpretations, and the relational-uniformity constraints, it is possible to validate equivalences like Example 6 in [MS88].

7 Discussion

The interpretation of Algol presented here is the best denotational model of local variables currently available. It supports validation of *all* the test equivalences proposed in the literature. Although we do not know whether a full-abstraction result can be proved, we are not aware of any true equivalence that it fails to validate.

A noteworthy aspect of the approach is that the validations are remarkably simple and natural. This encourages us to hope that practical program-verification techniques based on relational parametricity can be developed; see [Wad89, Mai91, Ma92] for work on the proof theory of relational parametricity.

Meyer and Sieber [MS88] have also developed denotational models that combine functors and logical relations. It is not clear how their approach relates to ours, but theirs seems tightly tied to a location-oriented view of variables. See also [MT92] for an approach to these issues based on operational, rather than denotational, semantics.

Other directions in which the work described here might be developed or applied include

- application to the semantic description of fully dynamic storage allocation and other dynamic-allocation regimes (cf. [Mog90]);
- strengthening the Proposition to a result of the following form: for $\mathbf{X}$ any small internal category and $\mathbf{Y}$ any *complete* cartesian closed internal category, $\mathbf{X} \Rightarrow \mathbf{Y}$ is a complete cartesian closed internal category (cf. [Nel81]); and
- application to more expressive languages, such as a language that allows both user-defined types and local variables.

Finally, we speculate that focusing on reflexive-graph (or similar) structure might prove useful in other contexts involving relational parametricity; for instance, in determining minimal hypotheses for the Abstraction Theorem [Rey83, MR92], or in clarifying the mathematical significance of using diagonal relations as "identities."

Acknowledgement

We are grateful to Andy Pitts for pointing out to us that a category we are using is the "well-known" category of reflexive graphs.

References

[AJ91] S. Abramsky and T. P. Jensen. A relational approach to strictness analysis for higher-order polymorphic functions. In *Conf. Record 18th ACM Symp. on Principles of Programming Languages*, pages 49–54, Orlando, Florida, 1991. ACM, New York.

[AL91] A. Asperti and G. Longo. *Categories, Types, and Structures.* The MIT Press, Cambridge, Mass., 1991.

[Bro85] S. D. Brookes. A fully abstract semantics and a proof system for an ALGOL-like language with sharing. In A. Melton, editor, *Mathematical Foundations of Programming Semantics*, volume 239 of *Lecture Notes in Computer Science*, pages 59–100, Manhattan, Kansas, 1985. Springer-Verlag, Berlin.

[BW90] M. Barr and C. Wells. *Category Theory for Computing Science.* Prentice-Hall International, London, 1990.

[Coo91] W. Cook. Object-oriented programming versus abstract data types. In J. W. de Bakker et al., editors, *Foundations of Object-Oriented Languages*, volume 489 of *Lecture Notes in Computer Science*, pages 151–178. Springer-Verlag, 1991.

[Don77] J. Donahue. Locations considered unnecessary. *Acta Informatica*, 8:221–242, 1977.

[Hoa72] C. A. R. Hoare. Correctness of data representations. *Acta Informatica*, 1:271–281, 1972. Reprinted in D. Gries, editor, *Programming Methodology*, pages 269-281. Springer-Verlag, Berlin, 1978.

[Hoa75] C. A. R. Hoare. Recursive data structures. *Int. J. of Computer and Information Sciences*, 4:105–132, 1975.

[HS79] H. Herrlich and G. E. Strecker. *Category Theory.* Heldermann Verlag, Berlin, second edition, 1979.

[KS74] G. M. Kelly and R. H. Street. Review of the basic elements of 2-categories. In G. M. Kelly, editor, *Category Seminar: Proceedings Sydney Category Theory Seminar, 1972/3*, volume 420 of *Lecture Notes in Mathematics.* Springer-Verlag, Berlin, 1974.

[Ma92] QingMing Ma. Parametricity as subtyping. In *Conf. Record 19th ACM Symp. on Principles of Programming Languages*, pages 281–292. ACM, New York, 1992.

[Mai91] H. G. Mairson. Outline of a proof theory of parametricity. In J. Hughes, editor, *Proceedings of Fifth International Conference on Functional Programming and Computer Architecture*, volume 523 of *Lecture Notes in Computer Science*, pages 313–327. Springer-Verlag, Berlin, 1991.

[Mit90] J. C. Mitchell. Type systems for programming languages. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 8, pages 365–458. Elsevier, Amsterdam, and The MIT Press, Cambridge, Mass., 1990.

[ML71] S. Mac Lane. *Categories for the Working Mathematician.* Springer-Verlag, New York, 1971.

[Mog90] E. Moggi. An abstract view of programming languages. Technical Report ECS-LFCS-90-113, Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh, 1990.

[MR92] QingMing Ma and J. C. Reynolds. Types, abstraction, and parametric polymorphism, part 2. In S. Brookes et al., editors, *Mathematical Foundations of Programming Semantics*, volume 598 of *Lecture Notes in Computer Science*, pages 1–40. Springer-Verlag, Berlin, 1992. Proceedings of the 1991 Conference.

[MS88] A. R. Meyer and K. Sieber. Towards fully abstract semantics for local variables: preliminary report. In *Conf. Record 15th ACM Symp. on Principles of Programming Languages*, pages 191–203, San Diego, California, 1988. ACM, New York.

[MS91] J. C. Mitchell and A. Scedrov. Sconing, relators, and parametricity. Unpublished draft, 1991.

[MT92] I. A. Mason and C. L. Talcott. References, local variables, and operational reasoning. In *Proceedings, 7th Annual IEEE Symposium on Logic in Computer Science*, Santa Cruz, California, 1992. IEEE Computer Society Press, Los Alamitos, California.

[Nel81] E. Nelson. On exponentiating exponentiation. *J. of Pure and Applied Algebra*, 20:79–91, 1981.

[Ole82] F. J. Oles. *A Category-Theoretic Approach to the Semantics of Programming Languages.* Ph.D. thesis, Syracuse University, Syracuse, N.Y., 1982.

[OT91] P. W. O'Hearn and R. D. Tennent. Semantical analysis of specification logic, part 2. Technical Report 91-304, Department of Computing and Information Science, Queen's University, Kingston, Canada, 1991. To appear in revised form in *Information and Computation.*

[OT92] P. W. O'Hearn and R. D. Tennent. Semantics of local variables. In M. P. Fourman, P. T. Johnstone, and A. M. Pitts, editors, *Applications of Categories in Computer Science*, volume 177 of *London Mathematical Society Lecture Note Series*, pages 217–238. Cambridge University Press, Cambridge, 1992.

[Plo73] G. D. Plotkin. Lambda-definability and logical relations. Memorandum SAI-RM-4, School of Artificial Intelligence, University of Edinburgh, October 1973.

[Plo80] G. D. Plotkin. Lambda-definability in the full type hierarchy. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays in Combinatory Logic, Lambda Calculus and Formalism*, pages 363–373. Academic Press, 1980.

[Rey75] J. C. Reynolds. User-defined types and procedural data structures as complementary approaches to data abstraction. In S. A. Schuman, editor, *New Advances in Algorithmic Languages 1975*, pages 157–168. Inst. de Reserche d'Informatique et d'Automatique, 1975. Reprinted in D. Gries, editor, *Programming Methodology*, pages 309-317. Springer-Verlag, Berlin, 1978.

[Rey78] J. C. Reynolds. Syntactic control of interference. In *Conf. Record 5th ACM Symp. on Principles of Programming Languages*, pages 39–46, Tucson, Arizona, 1978. ACM, New York.

[Rey81a] J. C. Reynolds. *The Craft of Programming.* Prentice-Hall International, London, 1981.

[Rey81b] J. C. Reynolds. The essence of Algol. In J. W. de Bakker and J. C. van Vliet, editors, *Algorithmic Languages*, pages 345–372. North-Holland, Amsterdam, 1981.

[Rey83] J. C. Reynolds. Types, abstraction and parametric polymorphism. In R. E. A. Mason, editor, *Information Processing 83*, pages 513–523. North Holland, Amsterdam, 1983.

[Rey84] J. C. Reynolds. Polymorphism is not set-theoretic. In G. Kahn, D. B. MacQueen, and G. D. Plotkin, editors, *Semantics of Data Types*, volume 173 of *Lecture Notes in Computer Science*, pages 145–156, Sophia-Antipolis, France, 1984. Springer-Verlag, Berlin.

[RP90] J. C. Reynolds and G. D. Plotkin. On functors expressible in the polymorphic typed lambda calculus. Technical Report CMU-CS-90-147, Carnegie Mellon University, School of Computer Science, 1990. To appear in *Information and Computation.*

[Sta85] R. Statman. Logical relations and the typed λ-calculus. *Information and Computation*, 65:85–97, 1985.

[Ten90] R. D. Tennent. Semantical analysis of specification logic. *Information and Computation*, 85(2):135–162, 1990.

[Ten91] R. D. Tennent. *Semantics of Programming Languages.* Prentice-Hall International, 1991.

[Wad89] P. Wadler. Theorems for free! In *Functional Programming Languages and Computer Architecture*, pages 347–359, 4th International Symposium, Imperial College, London, September 1989. ACM, New York.