Relational Parametricity and Local Variables

(Preliminary Report)

P. W. O'Hearn*
Computer and Information Science
Syracuse University
Syracuse, New York, U.S.A. 13244
ohearn@top.cis.syr.edu

R. D. Tennent†
Laboratory for Foundations of Computer Science
Department of Computer Science
University of Edinburgh
Edinburgh, Scotland, U.K. EH9 3JZ
rdt@dcs.ed.ac.uk

Abstract

J. C. Reynolds suggested that Strachey's intuitive concept of "parametric" (i.e., uniform) polymorphism is closely linked to *representation independence*, and used logical relations 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, by showing that interactions between local and non-local entities satisfy certain relational criteria.

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, and encompasses standard methods of reasoning about data representations; however, it is not known whether it is fully abstract.

1 Introduction

One of the first things most programmers learn is how to "declare" a new assignable local variable, and facilities to support this have been available in programming languages for over thirty years [NB+63]. It might be thought that there would by now be a satisfactory semantic interpretation for so fundamental and apparently elementary a mechanism. But existing models are *not* completely satisfactory [MS88, OT92]. The problems arise when block bodies can contain calls of non-local procedures, and the difficulty is in defining precisely the sense in which non-local entities are "independent" of a locally-declared variable.

For example, consider the following (Algol 60) block [MS88]:

> **begin**
>   **integer** z;
>   **procedure** $inc; z := z + 1$;
> $P(inc)$
> **end**

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 to $P(skip)$, where *skip* does nothing, for *every* possible meaning of P. But this equivalence fails in all published denotational models 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 slightly more complicated example:

> **begin**
>   **integer** z;
>   **procedure** $inc; z := z + 1$;
>   **integer procedure** $val; val := z$;
> $z := 0$;
> $P(inc, val)$
> **end**

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. A more modern language would provide a "sugared"

*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.

syntax, and one could write something like

$$
\begin{array}{l}
\textbf{module } \mathit{counter}(\textbf{exports } \mathit{inc}, \mathit{val});\\
\textbf{begin}\\
\quad \textbf{integer } z;\\
\quad \textbf{invariant } z \geq 0;\\
\quad \textbf{procedure } \mathit{inc}; z := z + 1;\\
\quad \textbf{integer procedure } \mathit{val}; \mathit{val} := z;\\
\quad z := 0\\
\textbf{end } \mathit{counter};\\
\ldots \mathit{counter}.\, \mathit{inc}; \ldots \mathit{counter}.\, \mathit{val} \ldots
\end{array}
$$

but the unsugared form shows that, even without additional features, the combination of local variables and procedures in Algol-like languages supports 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.) See [Rey81b, THM83, Ten91] for discussion of Algol-like languages, and [Rey75, Coo91] for comparisons of linguistic approaches to representational abstraction.

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 support validation of *invariance* principles often used for reasoning about data representations, as in [Hoa72]. For example, these models validate the following equivalence:

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

where **diverge** is a statement whose execution (in any state) never terminates. 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 this kind of representation invariant. Consider the following block, which uses a "non-standard" representation of a counter:

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

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; see, for example, [Mit86]. It is clearly important to be able to validate changes of representation; but existing semantic models of local variables almost always fail on such equivalences!

This failure is especially surprising because standard *informal* methods for demonstrating correctness of data representations [Hoa72][Rey81a, Chapter 5] can easily be adapted to proving such equivalences. For our example, consider the relation R between states for the two implementations such that, if z_0 and z_1 are the values of the variable z in the standard and non-standard implementations, respectively, R holds if and only if $-z_1 = z_0 \geq 0$ and all other variables have the *same* values. It can be shown that

- R is initially established by executing the two initializations (with identical non-local states);
- executions of (the two implementations of) *inc* preserve R; and
- evaluations of (the two implementations of) *val* in R-related states yield the same result.

The conclusion is that R holds after execution of the calls to P, and so the blocks have been "proved" to have equivalent effects on non-local variables. But, although there is no reason to think these methods are invalid, they have never been rigorously verified for a language with local-variable declarations!

In [Rey83], this relational approach to representation independence is adapted to (and made precise for) simply-typed lambda calculi and validated (for extensional models) using the technique of "logical" (families of) relations [Plo73, Plo80, Mit90], and then further generalized to languages that support data abstraction using programmer-defined types.

We can describe the representation-independence property provable using logical relations as follows. Suppose

- 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_0 and W_1 are domains, regarded as alternative "representations" of the type variable;
- $[\![\theta]\!]W_0$ is the domain of meanings of type θ when W_0 is assigned as the meaning of the type variable, and similarly for $[\![\theta]\!]W_1$;
- $[\![\pi]\!]W_0$ is the domain of π-compatible environments when W_0 is assigned as the meaning of the type variable, and similarly for $[\![\pi]\!]W_1$;

- $R \subseteq W_0 \times W_1$ is any complete relation on W_0 and W_1, regarded as relating representations of abstract values;

- $[\![\theta]\!] R \subseteq [\![\theta]\!] W_0 \times [\![\theta]\!] W_1$ is the complete relation on θ-meanings "logically" induced by R, and similarly for $[\![\pi]\!] R \subseteq [\![\pi]\!] W_0 \times [\![\pi]\!] W_1$;

- P is any phrase of type θ in context π;

- $[\![P]\!] W_0$ is a continuous function which is the meaning of P when W_0 is assigned as the meaning of the type variable, and similarly for $[\![P]\!] W_1$.

Then it can be proved that $[\![P]\!] W_0$ and $[\![P]\!] W_1$ is a *relation-preserving* pair of functions; i.e., for all $u_0 \in [\![\pi]\!] W_0$ and $u_1 \in [\![\pi]\!] W_1$,

$$\text{if } \; u_0 \big[[\![\pi]\!] R\big]\, u_1 \; \text{ then } \; [\![P]\!] W_0 u_0 \; \big[[\![\theta]\!] R\big] \; [\![P]\!] W_1 u_1.$$

Intuitively, this says that relations between different representations of a type variable are respected by programs that use it. We will refer to this kind of uniformity as *relational parametricity*, after [Str67] and [Rey83], and portray it diagrammatically as follows:

$$\begin{tikzcd} W_0 \arrow[d, leftrightarrow, "R"'] & [\![\pi]\!] W_0 \arrow[r, "{[\![P]\!] W_0}"] \arrow[d, leftrightarrow, "{[\![\pi]\!] R}"'] & [\![\theta]\!] W_0 \arrow[d, leftrightarrow, "{[\![\theta]\!] R}"] \\ W_1 & [\![\pi]\!] W_1 \arrow[r, "{[\![P]\!] W_1}"'] & [\![\theta]\!] W_1 \end{tikzcd}$$

Notice that double-headed arrows $\longleftrightarrow$ are used here for (binary) relations, and that this is not a conventional commutative diagram.

In this work, we show that relational parametricity is also relevant when representational abstraction is achieved using local variables and procedures, and, in particular, leads to an improved semantics of local variables in which examples such as those discussed above are correctly treated.

Before making this connection, we must review the possible-world approach to local variables pioneered by J. C. Reynolds [Rey81b] and F. J. Oles [Ole82, Ole85]. The essential idea is to *parameterize* interpretations by semantic entities, traditionally termed *possible worlds* [Kri65], that determine certain "local" aspects of meanings; for example, to treat local-variable declarations, each possible world must determine the set of storage states needed to represent the values of currently available variables. The meaning of a declaration block at one world can then be 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. Furthermore, each such change of world must induce suitable changes of meaning, so that non-local entities can be mapped into meanings appropriate to local contexts.

This semantic set-up can be elegantly described using basic concepts of category theory [BW90, Pie91].

- Possible worlds and changes of world form a *category*.

- Types are interpreted as *functors* from the category of possible worlds to a category of domains and continuous functions.

- Phrases are interpreted as *natural transformations* of these functors.

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

$$\begin{tikzcd} W_0 \arrow[d, "f"'] & [\![\pi]\!] W_0 \arrow[r, "{[\![P]\!] W_0}"] \arrow[d, "{[\![\pi]\!] f}"'] & [\![\theta]\!] W_0 \arrow[d, "{[\![\theta]\!] f}"] \\ W_1 & [\![\pi]\!] W_1 \arrow[r, "{[\![P]\!] W_1}"'] & [\![\theta]\!] W_1 \end{tikzcd}$$

where

- θ, π, and P are as before, but without any type variables;

- W_0 and W_1 are possible worlds;

- $[\![\theta]\!] W_0$ and $[\![\theta]\!] W_1$ are the domains of θ-meanings appropriate to worlds W_0 and W_1, respectively, and similarly for environment domains $[\![\pi]\!] W_0$ and $[\![\pi]\!] W_1$;

- $f: W_0 \to W_1$ is any change of world;

- $[\![\theta]\!] f$ is the change of meaning induced by f, and similarly for $[\![\pi]\!] f$; and

- $[\![P]\!] W_0$ and $[\![P]\!] W_1$ are continuous functions that interpret phrase P in worlds W_0 and W_1, respectively.

Notice that, in many respects, this is similar to the relational-parametricity picture discussed earlier.

The key point is that $[\![P]\!]$ is, in a certain sense, polymorphic, and so it is conceivable to require its components to satisfy a parametricity constraint. In general, this will be *different* from naturality, with neither being stronger than the other. Reynolds and Oles have shown that local-variable semantics is natural with respect to the structure in possible-world categories, and so we certainly do not want to give this up; but the examples discussed above show that naturality alone is not enough. In this work, we show how to *combine* these two uniformity criteria, and demonstrate that requiring *both* naturality *and* parametricity yields a significantly improved model of local variables.

To achieve this combination of relational parametricity and naturality, we will define suitable cartesian closed categories of "relation-preserving" functors and natural transformations. The key technical notion underlying this construction is that of a *reflexive graph*, which is essentially an arbitrary 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

We will regard a binary relation R as a triple (W_0, W_1, S) where W_0 and W_1 are sets (the *domains* of R) and $S \subseteq W_0 \times W_1$ (the *graph* of R). Although it will suffice for our applications to work exclusively with *binary* relations, our definitions and results (though not all of our notation) generalize straightforwardly to n-ary relations for any n. We will use the notations $R\colon W_0 \leftrightarrow W_1$ and $\begin{array}{c} W_0 \\ R\,\Big\updownarrow \\ W_1 \end{array}$ to mean that R is a binary relation with domains W_0 and W_1, and $w_0[R]w_1$ to mean $(w_0, w_1) \in \text{graph}\, R$.

If W is any set,

- $\Delta_W\colon W \leftrightarrow W$ is the diagonal relation on W; i.e., $w[\Delta_W]w' \iff w = w'$.

If W_0, W_1, X_0, and X_1 are sets and $R\colon W_0 \leftrightarrow W_1$ and $S\colon X_0 \leftrightarrow X_1$,

- $R \times S\colon\ W_0 \times X_0 \longleftrightarrow W_1 \times X_1$ is defined by $(w_0, x_0)[R \times S](w_1, x_1) \iff w_0[R]w_1$ and $x_0[S]x_1$;
- $R \rightarrow S\colon\ W_0 \rightarrow X_0 \longleftrightarrow W_1 \rightarrow X_1$ is defined by $f_0[R \rightarrow S]f_1 \iff$ for all $w_0 \in W_0$, $w_1 \in W_1$, if $w_0[R]w_1$ then $f_0(w_0)[S]f_1(w_1)$.

If D and E are partially ordered sets and $R\colon D \leftrightarrow E$ (i.e., R is a relation on the underlying sets),

- $R_\perp\colon D_\perp \longleftrightarrow E_\perp$ is defined by $d[R_\perp]e \iff d = e = \perp$ 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 fundamental 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 the notion, familiar to category theorists, of a *reflexive graph* [BW90, Law89, Joh89]. A reflexive graph is conventionally a set of *vertices* with (oriented) *edges* between them; furthermore, for any vertex v, there is a distinguished edge from v to itself, the *identity* on v. Notice that a reflexive graph is more structured than a set (because there are edges as well as vertices), but less structured than most categories (because edges need not be composable).

We will actually consider a generalization where *categories* of vertices and edges are allowed; the conventional notion of reflexive graph becomes the special case in which the vertex and edge categories are small and discrete. In some examples, the edge objects will be relations over pairs of vertex objects, and the edge morphisms will be relation-preserving pairs of vertex morphisms; however, in general, edges are not required to be any of the usual categorical forms of relation [MR92, MS91], and this flexibility will prove to be essential for our applications.

Here is a precise definition: a *reflexive graph* $\mathcal{G}$ consists of two categories, $\mathcal{G}_v$ (vertices) and $\mathcal{G}_e$ (edges), and three functors between them as follows:

$$\begin{tikzcd} \mathcal{G}_v \arrow[r, "\mathcal{G}_I"] & \mathcal{G}_e \arrow[l, bend right=50, "\mathcal{G}_{\delta_0}"'] \arrow[l, bend left=50, "\mathcal{G}_{\delta_1}"] \end{tikzcd}$$

such that $\mathcal{G}_I ; \mathcal{G}_{\delta_i} = 1_{\mathcal{G}_v}$ for $i = 0, 1$, where ; denotes composition in diagrammatic order and $1_{\mathcal{G}_v}$ is the identity functor on $\mathcal{G}_v$. Intuitively, $\mathcal{G}_{\delta_i}$ specifies the i'th domain for each edge and edge morphism, and $\mathcal{G}_I$ specifies the identity edge for each vertex and vertex morphism.

An equivalent and more elegant presentation is as follows: a reflexive graph is a functor $\mathcal{G}\colon \mathbf{G} \rightarrow \mathbf{CAT}$, where $\mathbf{CAT}$ is the meta-category of all categories and functors between them [ML71], and $\mathbf{G}$ is the two-object category whose (non-identity) morphisms are as follows:

$$\begin{tikzcd} v \arrow[r, "I"] & e \arrow[l, bend right=50, "\delta_0"'] \arrow[l, bend left=50, "\delta_1"] \end{tikzcd}$$

with $I ; \delta_i = \mathrm{id}_v$ for $i = 0, 1$, where id_v is the identity morphism on v. (More generally, reflexive graphs with n-ary edges would be generated by the two-object category having non-identity morphisms $I\colon v \rightarrow e$ and $\delta_i\colon e \rightarrow v$ for $i = 0, 1, \ldots, n-1$, with a similar commutativity requirement.)

As our first example, we define a reflexive graph $\mathcal{S}$ (sets) as follows.

- The "vertex" category, $\mathcal{S}_v$, is the usual category of sets and functions.
- The "edge" category, $\mathcal{S}_e$, has binary relations on sets as objects and relation-preserving pairs of functions as morphisms; i.e., a morphism with domain $R: W_0 \leftrightarrow W_1$ and co-domain $S: X_0 \leftrightarrow X_1$ is a quadruple (R, f_0, f_1, S) such that $f_0[R \to S]f_1$. We will use the relational-parametricity diagram $\begin{tikzcd} W_0 \arrow[r, "f_0"] \arrow[d, leftrightarrow, "R"'] & X_0 \arrow[d, leftrightarrow, "S"] \\ W_1 \arrow[r, "f_1"'] & X_1 \end{tikzcd}$ to depict such a morphism. The composite of $\begin{tikzcd} W_0 \arrow[r, "f_0"] \arrow[d, leftrightarrow, "R"'] & X_0 \arrow[d, leftrightarrow, "S"] \\ W_1 \arrow[r, "f_1"'] & X_1 \end{tikzcd}$ and $\begin{tikzcd} X_0 \arrow[r, "g_0"] \arrow[d, leftrightarrow, "S"'] & Y_0 \arrow[d, leftrightarrow, "T"] \\ X_1 \arrow[r, "g_1"'] & Y_1 \end{tikzcd}$ is then defined as $\begin{tikzcd} W_0 \arrow[r, "f_0;g_0"] \arrow[d, leftrightarrow, "R"'] & Y_0 \arrow[d, leftrightarrow, "T"] \\ W_1 \arrow[r, "f_1;g_1"'] & Y_1 \end{tikzcd}$, and the identity morphism on a relation $R: W_0 \leftrightarrow W_1$ is $\begin{tikzcd} W_0 \arrow[r, "\mathrm{id}_{W_0}"] \arrow[d, leftrightarrow, "R"'] & W_0 \arrow[d, leftrightarrow, "R"] \\ W_1 \arrow[r, "\mathrm{id}_{W_1}"'] & W_1 \end{tikzcd}$.
- Functors $\mathcal{S}_{\delta_i}: \mathcal{S}_e \to \mathcal{S}_v$ for $i = 0, 1$ are defined by $\mathcal{S}_{\delta_i}(R: W_0 \leftrightarrow W_1) = W_i$ and

$$\mathcal{S}_{\delta_i}\left(\begin{tikzcd} W_0 \arrow[r, "f_0"] \arrow[d, leftrightarrow, "R"'] & X_0 \arrow[d, leftrightarrow, "S"] \\ W_1 \arrow[r, "f_1"'] & X_1 \end{tikzcd}\right) = f_i \ .$$

- Functor $\mathcal{S}_I: \mathcal{S}_v \to \mathcal{S}_e$ is such that $\mathcal{S}_I(W) = \Delta_W$ and $\mathcal{S}_I(f: W \to X) = \begin{tikzcd} W \arrow[r, "f"] \arrow[d, leftrightarrow, "\Delta_W"'] & X \arrow[d, leftrightarrow, "\Delta_X"] \\ W \arrow[r, "f"'] & X \end{tikzcd}$.

Category $\mathcal{S}_e$ is, of course, the usual category of relations over sets, as in [MR92]. Furthermore, the $\mathcal{S}_{\delta_i}$ are similar to the forgetful functor U used there in a categorical treatment of the (first-order) "abstraction theorem," and $\mathcal{S}_I$ is similar to the functor J used there in a categorical treatment of the "identity extension lemma." Hence, some of the key entities introduced in [MR92] are incorporated in the reflexive graph $\mathcal{S}$.

As our second example, we define a reflexive graph $\mathcal{D}$ (domains) as follows.

- $\mathcal{D}_v$ is the category of directed-complete partially-ordered sets and continuous functions.
- $\mathcal{D}_e$ has complete binary relations as objects, and relation-preserving pairs of continuous functions as morphisms (where a binary relation on posets is *complete* if its graph is a directed-complete subset of the pointwise-ordered product of the domains of the relation.) Composites and identities are evident.
- The functors $\mathcal{D}_{\delta_i}: \mathcal{D}_e \to \mathcal{D}_v$ for $i = 0, 1$ and $\mathcal{D}_I: \mathcal{D}_v \to \mathcal{D}_e$ are defined exactly as for $\mathcal{S}$.

Finally, we define a reflexive graph $\mathcal{W}$ (worlds) having the small category Σ of "state shapes" described in [Ole82] as its vertex category $\mathcal{W}_v$. The category Σ is as follows.

- The objects are (certain) sets, including desired data types, such as $\{true, false\}$ 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 $x \in X$,
  $$\rho\big(\phi(x), x\big) \;=\; x \ ;$$
  - for all $x \in X$ and $w \in W$,
  $$\phi\big(\rho(w, x)\big) \;=\; w \ ;$$
  - for all $x \in X$ and $w, w' \in W$,
  $$\rho\big(w, \rho(w', x)\big) \;=\; \rho(w, x) \ .$$

  For example, there is an "expansion" morphism $(\phi, \rho): W \to X$ such that $X = W \times Z$ for some data type Z with $\phi\langle w, z\rangle = w$ and $\rho(\overline{w}, \langle w, z\rangle) = \langle \overline{w}, z\rangle$; 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 large stack. In fact, Oles shows that any $(\phi, \rho): W \to X$ 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 morphisms $(\phi, \rho): W \to X$ and $(\phi', \rho'): X \to Y$ is $(\phi'', \rho''): W \to Y$ such that $\phi'' = \phi' \,;\, \phi$ and $\rho''(w, y) = \rho'\big(\rho(w, \phi'(y)), y\big)$.

- The identity morphism on W is (ϕ, ρ) such that $\phi(w) = w$ and $\rho(w, w') = w$.

A category $\mathcal{W}_e$ of relations over Σ can be defined as follows.

- The objects are relations $R\colon W \leftrightarrow X$, where W and X are Σ-objects.
- A morphism with domain $R\colon W_0 \leftrightarrow W_1$ and co-domain $S\colon X_0 \leftrightarrow X_1$ is a quadruple $\big(R, (\phi_0, \rho_0), (\phi_1, \rho_1), S\big)$ such that $\phi_0[S \to R]\phi_1$ and $\rho_0[R \times S \to S]\rho_1$. Again, we use diagrams of the form
$$\begin{tikzcd} W_0 \arrow[r, "{(\phi_0,\rho_0)}"] \arrow[d, leftrightarrow, "R"'] & X_0 \arrow[d, leftrightarrow, "S"] \\ W_1 \arrow[r, "{(\phi_1,\rho_1)}"'] & X_1 \end{tikzcd}$$
to depict morphisms in $\mathcal{W}_e$.
- Composition and identities are defined straightforwardly in terms of those in Σ.

We can now complete the definition of $\mathcal{W}$ by using diagonal relations for the identities, and defining the domain functors in the evident fashion.

The definition of "related" Σ-morphisms above is particularly noteworthy:
$$\begin{tikzcd} W_0 \arrow[r, "{(\phi_0,\rho_0)}"] \arrow[d, leftrightarrow, "R"'] & X_0 \arrow[d, leftrightarrow, "S"] \\ W_1 \arrow[r, "{(\phi_1,\rho_1)}"'] & X_1 \end{tikzcd}$$
is a morphism in $\mathcal{W}_e$ if and only if both
$$\begin{tikzcd} X_0 \arrow[r, "\phi_0"] \arrow[d, leftrightarrow, "S"'] & W_0 \arrow[d, leftrightarrow, "R"] \\ X_1 \arrow[r, "\phi_1"'] & W_1 \end{tikzcd}$$
and
$$\begin{tikzcd} W_0 \times X_0 \arrow[r, "\rho_0"] \arrow[d, leftrightarrow, "R \times S"'] & X_0 \arrow[d, leftrightarrow, "S"] \\ W_1 \times X_1 \arrow[r, "\rho_1"'] & X_1 \end{tikzcd}$$
are morphisms in $\mathcal{S}_e$. This definition ensures that appropriate relations will be preserved by variable de-allocation (using the "projections" ϕ_i) and by state changes in larger worlds induced by changes at smaller ones (using the "replacements" ρ_i). This illustrates why the flexibility provided by reflexive graphs is needed: $\mathcal{W}_e$ is not a category of relations over Σ in the sense of [MR92]; in fact, Σ does not even have a terminal object. Another example that takes advantage of the flexibility will be discussed in Section 6, where we consider a location-oriented model.

3 Parametric Functors and Natural Transformations

In this section we describe a category of "parametric" functors and natural transformations. This will be similar to the functor category $\mathcal{W}_v \Rightarrow \mathcal{D}_v$ used to model local variables in [Ole82], but with the relational structure in the reflexive graphs $\mathcal{W}$ and $\mathcal{D}$ also taken into account. The description will be highly tailored to the specific definitions of $\mathcal{W}$ and $\mathcal{D}$, but at the end of the section we sketch a more general setting for the definitions.

A *parametric functor from $\mathcal{W}$ to $\mathcal{D}$* consists of

- a mapping F_0 from $\mathcal{W}_v$-objects to $\mathcal{D}_v$-objects;
- a mapping F_1 from $\mathcal{W}_v$-morphisms to $\mathcal{D}_v$-morphisms; and
- a mapping F_2 from $\mathcal{W}_e$-objects to $\mathcal{D}_e$-objects

such that

- if $f\colon w \to x$ in $\mathcal{W}_v$ then $F_1(f)\colon F_0(w) \to F_0(x)$ in $\mathcal{D}_v$;
- $F_1(\mathrm{id}_w) = \mathrm{id}_{F_0(w)}$ for every $\mathcal{W}_v$-object w;
- $F_1(f \,;\, g) = F_1(f) \,;\, F_1(g)$ for all composable $\mathcal{W}_v$-morphisms f and g;
- if $R\colon w \leftrightarrow x$ in $\mathcal{W}_e$ then $F_2(R)\colon F_0(w) \leftrightarrow F_0(x)$ in $\mathcal{D}_e$;
- $F_2(\Delta_w) = \Delta_{F_0(w)}$, for every $\mathcal{W}_v$-object w; and
- if
$$\begin{tikzcd} w_0 \arrow[r, "f_0"] \arrow[d, leftrightarrow, "R"'] & x_0 \arrow[d, leftrightarrow, "S"] \\ w_1 \arrow[r, "f_1"'] & x_1 \end{tikzcd}$$
in $\mathcal{W}_e$, then
$$\begin{tikzcd} F_0(w_0) \arrow[r, "F_1(f_0)"] \arrow[d, leftrightarrow, "F_2(R)"'] & F_0(x_0) \arrow[d, leftrightarrow, "F_2(S)"] \\ F_0(w_1) \arrow[r, "F_1(f_1)"'] & F_0(x_1) \end{tikzcd}$$
in $\mathcal{D}_e$.

The first three conditions say that F_0 and F_1 constitute a conventional functor from $\mathcal{W}_v$ to $\mathcal{D}_v$; the next two conditions say that F_0 and F_2 constitute a "relator" [MS91, AJ91]; and the last condition is a parametricity constraint. We will adopt the usual notational abuse of using a single symbol such as F to denote all three mappings.

If F and G are parametric functors from $\mathcal{W}$ to $\mathcal{D}$, η is a *parametric natural transformation from F to G* if it maps $\mathcal{W}_v$-objects to $\mathcal{D}_v$-morphisms such that

- for every $\mathcal{W}_v$-object w, $\eta(w)\colon F(w) \to G(w)$;
- for every $\mathcal{W}_v$-morphism $f\colon w \to x$,
$$\begin{tikzcd} F(w) \arrow[r, "\eta(w)"] \arrow[d, "F(f)"'] & G(w) \arrow[d, "G(f)"] \\ F(x) \arrow[r, "\eta(x)"'] & G(x) \end{tikzcd}$$
commutes; and

- for every $R\colon w_0 \leftrightarrow w_1$ in $\mathcal{W}_e$,

$$\begin{tikzcd} F(w_0) \arrow[r, "\eta(w_0)"] \arrow[d, leftrightarrow, "F(R)"'] & G(w_0) \arrow[d, leftrightarrow, "G(R)"] \\ F(w_1) \arrow[r, "\eta(w_1)"'] & G(w_1) \end{tikzcd}$$

  in $\mathcal{D}_e$.

The first two conditions say that η is a conventional natural transformation from F to G, and the last condition is a parametricity constraint.

Parametric natural transformations compose in the obvious point-wise way. The category having all parametric functors from $\mathcal{W}$ to $\mathcal{D}$ as objects and all parametric natural transformations of these as morphisms will be denoted $\mathcal{W} \Rightarrow \mathcal{D}$.

Proposition 1 *$\mathcal{W} \Rightarrow \mathcal{D}$ is cartesian closed.*

Proof. 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 (parametric) projections.

For exponentiation, $(F \to G)(w)$ is the pointwise-ordered domain of families

$$p \in \prod_{f:w \to x} F(x) \to G(x)$$

of continuous functions indexed by $\mathcal{W}_v$-morphisms out of w, such that

- for all $f\colon w \to x$ and $g\colon x \to y$ in $\mathcal{W}_v$,

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

  commutes, and

- if $\begin{tikzcd} w \arrow[r, "f_0"] \arrow[d, leftrightarrow, "\Delta_w"'] & x_0 \arrow[d, leftrightarrow, "S"] \\ w \arrow[r, "f_1"'] & x_1 \end{tikzcd}$ in $\mathcal{W}_e$ then

$$\begin{tikzcd} F(x_0) \arrow[r, "p(f_0)"] \arrow[d, leftrightarrow, "F(S)"'] & G(x_0) \arrow[d, leftrightarrow, "G(S)"] \\ F(x_1) \arrow[r, "p(f_1)"'] & G(x_1) \end{tikzcd}$$

  in $\mathcal{D}_e$.

The first condition is the usual naturality condition for exponentials in pre-sheaf categories [Gol79, Ten91], and the second condition is a parametricity constraint. The morphism part of $F \to G$ is defined by

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

and the relation part is as follows:

$$p_0\big[(F \to G)(R\colon w_0 \leftrightarrow w_1)\big]\,p_1$$

if and only if, for all $f_0\colon w_0 \to x_0$, $f_1\colon w_1 \to x_1$, and $S\colon x_0 \leftrightarrow x_1$,

if $\begin{tikzcd} w_0 \arrow[r, "f_0"] \arrow[d, leftrightarrow, "R"'] & x_0 \arrow[d, leftrightarrow, "S"] \\ w_1 \arrow[r, "f_1"'] & x_1 \end{tikzcd}$ then $\begin{tikzcd} F(x_0) \arrow[r, "p_0(f_0)"] \arrow[d, leftrightarrow, "F(S)"'] & G(x_0) \arrow[d, leftrightarrow, "G(S)"] \\ F(x_1) \arrow[r, "p_1(f_1)"'] & G(x_1) \end{tikzcd}$. Application and currying are defined as in conventional pre-sheaf categories; the necessary parametricity properties are easily proved. ☐

It is noteworthy that a parametricity 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 parametricity condition. This is strongly reminiscent of the use of a parametricity condition to constrain values of $\forall$ types in [Rey83]. In fact, the identity relation Δ_w 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 $\mathcal{W}_v$, over which indexing is done, is small.)

The use of squares of the form $\begin{tikzcd} w \arrow[r, "f_0"] \arrow[d, leftrightarrow, "\Delta_w"'] & x_0 \arrow[d, leftrightarrow, "S"] \\ w \arrow[r, "f_1"'] & x_1 \end{tikzcd}$ to constrain the relations S that can be used when applying the parametricity property is very important in practice. For example, when f_0 and f_1 are both the expansion from set of states W to $W \times Z$, this condition requires S 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 relations S, the requirement $\begin{tikzcd} F(x_0) \arrow[r, "p(f_0)"] \arrow[d, leftrightarrow, "F(S)"'] & G(x_0) \arrow[d, leftrightarrow, "G(S)"] \\ F(x_1) \arrow[r, "p(f_1)"'] & G(x_1) \end{tikzcd}$ 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.

In the remainder of this section, we briefly consider a categorical context for the preceding definitions; it can be skipped without loss of continuity. A morphism

$M: \mathcal{G} \to \mathcal{H}$ of reflexive graphs is a pair of functors M_e and M_v that map edges to edges and vertices to vertices in a way that preserves domains and identities; i.e.,

$$\begin{tikzcd}
\mathcal{G}_e \arrow[r, "M_e"] \arrow[d, "\mathcal{G}_{\delta_i}"'] & \mathcal{H}_e \arrow[d, "\mathcal{H}_{\delta_i}"] \\
\mathcal{G}_v \arrow[r, "M_v"] \arrow[d, "\mathcal{G}_I"'] & \mathcal{H}_v \arrow[d, "\mathcal{H}_I"] \\
\mathcal{G}_e \arrow[r, "M_e"] & \mathcal{H}_e
\end{tikzcd}$$

commutes for $i = 0, 1$. Composition of graph morphisms is defined componentwise. This defines the category of reflexive graphs.

Notice that a morphism of reflexive graphs is nothing other than a natural transformation between graphs viewed as functors. Furthermore, what we called a "parametric" natural transformation above is an instance of the concept of *modification* [KS74]. This gives some assurance of the appropriateness of the various conditions in the definition $\mathcal{W} \Rightarrow \mathcal{D}$, which uses simplifications that depend on specific structure in $\mathcal{W}$ and $\mathcal{D}$.

The category of reflexive graphs is cartesian closed; in fact this is true of the functor category $\mathbf{X} \Rightarrow \mathbf{CAT}$ for any small category $\mathbf{X}$. Products are definable pointwise, and the exponentiation $\mathcal{H}^{\mathcal{G}}$ of graphs can then be defined using standard methods for treating exponentiation in functor categories [Nel81]. The connection between this exponentiation and our category of "parametric functors" can be summed up as follows.

Proposition 2 *$\mathcal{W} \Rightarrow \mathcal{D}$ is isomorphic to the vertex category $(\mathcal{D}^{\mathcal{W}})_v$.*

Proof. This essentially requires verifying the correspondence just mentioned between parametric functors and natural transformations, and between parametric natural transformations and modifications. □

We say that a reflexive graph $\mathcal{G}$ is *small* if the vertex and edge categories $\mathcal{G}_v$ and $\mathcal{G}_e$ are small, and that $\mathcal{G}$ is *cartesian closed* if $\mathcal{G}_v$ and $\mathcal{G}_e$ are cartesian closed *and* the functors $\mathcal{G}_I$ and $\mathcal{G}_{\delta_i}$ between them preserve (on the nose) their specified cartesian closed structure. It is easy to show that $\mathcal{D}$ is a cartesian closed reflexive graph. This would continue to hold if we were to redefine the identity relation on every object D to be the *full* relation (i.e., the graph of $\mathcal{D}_I(D)$ is $D \times D$) or to be the partial order $\sqsubseteq_D$ on D. We are using diagonal relations as "identities" because this leads to better parametricity properties. On the other hand, if we were to use the *empty* relation as the identity, we would still have a reflexive graph, but not one that is cartesian closed. $\mathcal{W}$ is small but, of course, is not cartesian closed.

Finally, Proposition 1 is actually an instance of the following result, which is a reflexive-graph analogue of the standard fact that the category of functors from any small category into the category of sets is cartesian closed.

Proposition 3 *For any small reflexive graph $\mathcal{G}$, the exponential graph $\mathcal{D}^{\mathcal{G}}$ is cartesian closed.*

We omit the proof from this preliminary report.

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 as parametric functors from $\mathcal{W}$ to $\mathcal{D}$, and phrases as parametric natural transformations of such functors.

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

- for every $\mathcal{W}_v$-object W,
$$[\![\mathbf{exp}]\!]W \;=\; W \to Z_\perp \;,$$
where Z is the set of integers;
- for every $\mathcal{W}_v$-morphism $(\phi, \rho): W \to X$ and $e \in [\![\mathbf{exp}]\!]W$,
$$[\![\mathbf{exp}]\!](\phi, \rho)\; e \;=\; \phi\, ; e \;,$$
and
- for every $R: W_0 \leftrightarrow W_1$,
$$[\![\mathbf{exp}]\!]R \;=\; R \to (\Delta_Z)_\perp \;.$$

For commands:

- for every $\mathcal{W}_v$-object W,
$$[\![\mathbf{comm}]\!]W \;=\; W \to W_\perp \;;$$
- for every $\mathcal{W}_v$-morphism $(\phi, \rho): W \to X$, $x \in X$, and $c \in [\![\mathbf{comm}]\!]W$,
$$[\![\mathbf{comm}]\!](\phi, \rho)\; c\; x \;=\; \begin{cases} \rho\big(c(\phi(x)), x\big), & \text{if } c(\phi(x)) \neq \perp \\ \perp, & \text{if } c(\phi(x)) = \perp \end{cases}$$
and
- for every $R: W_0 \leftrightarrow W_1$,
$$[\![\mathbf{comm}]\!]R \;=\; R \to R_\perp \;.$$

The parametricity 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

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

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

using a *parametric* version of "contra-exponentiation" [OT92], where $\Box D$ is the constant functor whose object, morphism, and relation parts always yield D, id_D, and Δ_D, respectively. This is already an indication of the effectiveness of the parametricity constraints.

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

$$[\![\mathbf{var}]\!] \;=\; \left(\Box Z \rightarrow [\![\mathbf{comm}]\!]\right) \times [\![\mathbf{exp}]\!]$$

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, Ten91].

Finally, for procedures,

$$[\![\theta \rightarrow \theta']\!] \;=\; [\![\theta]\!] \rightarrow [\![\theta']\!] \ ,$$

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

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

$$assign\colon [\![\mathbf{var}]\!] \times [\![\mathbf{exp}]\!] \stackrel{.}{\rightarrow} [\![\mathbf{comm}]\!]$$

defined as follows:

$$\begin{aligned} &assign\, W\big((a,e),e'\big)(w) \\ &\quad = \begin{cases} a(\mathrm{id}_W)(z)(w), & \text{if } e'(w) = z \\ \perp, & \text{if } e'(w) = \perp \end{cases} \end{aligned}$$

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

We will define

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

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

$$new\ W\ p\ w \;=\; \begin{cases} w', & \text{if } p(f)(a,e)\langle w,\bar{z}\rangle = \langle w',z'\rangle \\ \perp, & \text{if } p(f)(a,e)\langle w,\bar{z}\rangle = \perp \end{cases}$$

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

To see the parametricity of new, consider any worlds W_0 and W_1, relation $R\colon W_0 \leftrightarrow W_1$, and $w_i \in W_i$ and $p_i \in [\![\mathbf{var} \rightarrow \mathbf{comm}]\!]W_i$ for $i = 0,1$ such that $w_0[R]w_1$ and $p_0\big[[\![\mathbf{var} \rightarrow \mathbf{comm}]\!]R\big]p_1$. We must show that $new\ W_0\ p_0\ w_0\big[R_\perp\big]new\ W_1\ p_1\ w_1$. It can be verified that

$$\begin{tikzcd} W_0 \arrow[r, "f_0"] \arrow[d, leftrightarrow, "R"'] & W_0 \times Z \arrow[d, leftrightarrow, "R \times \Delta_Z"] \\ W_1 \arrow[r, "f_1"'] & W_1 \times Z \end{tikzcd}$$

where $f_0\colon W_0 \rightarrow W_0 \times Z$ and $f_1\colon W_1 \rightarrow W_1 \times Z$ are expansions in $\mathcal{W}_v$; hence,

$$p_0(f_0)\big[[\![\mathbf{var}]\!](R \times \Delta_Z) \rightarrow [\![\mathbf{comm}]\!](R \times \Delta_Z)\big]p_1(f_1) \ .$$

Furthermore, it can be verified that

$$(a_0,e_0)\big[[\![\mathbf{var}]\!](R \times \Delta_Z)\big](a_1,e_1)$$

for the new variables $(a_0,e_0) \in [\![\mathbf{var}]\!](W_0 \times Z)$ and $(a_1,e_1) \in [\![\mathbf{var}]\!](W_1 \times Z)$, and so

$$p_0(f_0)(a_0,e_0)\langle w_0,\bar{z}\rangle\big[(R \times \Delta_Z)_\perp\big]p_1(f_1)(a_1,e_1)\langle w_1,\bar{z}\rangle$$

and this ensures that

$$new\ W_0\ p_0\ w_0\big[R_\perp\big]new\ W_1\ p_1\ w_1 \ .$$

5 Equivalences

It is clear that the model described in the preceding is *computationally adequate*, in the sense that denotational equality implies observational equivalence using a reasonable operational semantics; see [Len92]. We do not know if the converse is true (full abstraction), but 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

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

It can be verified that

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

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

Suppose $c \in [\![\mathbf{comm}]\!](W \times Z)$ is such that

$$skip\,[[\![\mathbf{comm}]\!]R_E]\,c,$$

where $skip \in [\![\mathbf{comm}]\!]W$ is defined by $skip(w) = w$. Then, for any $p \in [\![\mathbf{comm} \to \mathbf{comm}]\!]W$, parametricity implies that

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

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.,

$$w[R_Z]\langle w', z\rangle \iff w = w' \ .$$

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 $z := z + 1$ and **skip**; more precisely, if we define $inc \in [\![\mathbf{comm}]\!](W \times Z)$ by

$$inc\langle w, z\rangle \ = \ \langle w, z+1\rangle,$$

then

$$skip\,[[\![\mathbf{comm}]\!]R_Z]\,inc.$$

Then we can use the property $(*)$ to conclude

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

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

$$\begin{array}{l} \mathbf{begin} \\ \quad \mathbf{integer}\ z; \\ \quad P(z := z+1) \\ \mathbf{end} \end{array} \qquad \equiv \qquad P(\mathbf{skip})$$

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

$$\begin{array}{l} \mathbf{begin} \\ \quad \mathbf{integer}\ z; \\ \quad z := 0; \\ \quad P(z := z+1); \\ \quad \cdots \\ \mathbf{end} \end{array}$$

Our last example using relations of the form R_E is

$$\begin{array}{l} \mathbf{begin} \\ \quad \mathbf{integer}\ z; \\ \quad z := 0; \\ \quad P(z) \\ \mathbf{end} \end{array} \qquad \equiv \qquad P(0)$$

where $P\colon \mathbf{exp} \to \mathbf{comm}$; we have assumed a dereferencing 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 parametricity 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 $W \times Z \to Z$.)

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

$$\langle w_0, z_0\rangle[R]\langle w_1, z_1\rangle \iff w_0 = w_1 \text{ and } -z_1 = z_0 \geq 0$$

This can be used to validate the equivalence between blocks that use non-negative and non-positive implementations of a counter in exactly the manner discussed in the Introduction: the representations are directly related and then parametricity of procedures is used, as in the previous examples. The methods described in [Hoa72] and [Rey81a, Chapter 5] are special cases of this technique.

6 Locations

The traditional approach to storage variables in denotational semantics is to use "locations." In this section, we describe a reflexive graph $\mathcal{L}$ of worlds and "relations" suitable for a location-oriented semantics of variables, and then the corresponding interpretations of language primitives.

A suitable category $\mathcal{L}_v$ of worlds [Mog90, OT92, Pit92] 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 $\{0, 1, 2, \ldots, m-1\}$.

A suitable category $\mathcal{L}_e$ of "relations" over these worlds is much more complicated. This is perhaps not so surprising; many authors [Hoa75, Don77, Rey81b, Bro85] have argued that locations provide a very "low-level" semantics for variables. Technically, it would be possible to consider the relations 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 parametricity conditions would be rather weak for a language with states.

We need to allow for relations on *states*, as well as on *locations*, and these will be suitably constrained to ensure the parametricity of the relevant primitives.

The $\mathcal{L}_e$-objects will be pairs (R, R') such that, for some $\mathcal{L}_v$-objects n_0 and n_1,

- $R: L(n_0) \leftrightarrow L(n_1)$, where, for every i, $L(i) = \{0, 1, \ldots, i-1\}$ is the set of locations in world i;
- $R': S(n_0) \leftrightarrow S(n_1)$, where, for every i, $S(i) = L(i) \to Z$ is the set of storage states in world i;
- for all $i, i' \in L(n_0)$ and $j, j' \in L(n_1)$, if $i[R]j$ and $i'[R]j'$ then $i = i' \iff j = j'$;
- for all $w \in S(n_0)$, $x \in S(n_1)$, $i \in L(n_0)$, $j \in L(n_1)$ and $z \in Z$, if $w[R']x$ and $i[R]j$ then
$$(w \mid i \mapsto z)[R'](x \mid j \mapsto z),$$
where $(w \mid i \mapsto z)$ is the state like w except that it maps i to z; and
- for all $w \in S(n_0)$, $x \in S(n_1)$, $i \in L(n_0)$ and $j \in L(n_1)$, if $w[R']x$ and $i[R]j$ then $w(i) = x(j)$.

The third condition on (R, R') says that R must be single-valued and injective, i.e., given by a span $n_0 \leftarrow k \rightarrow n_1$ of morphisms between worlds. This is needed to ensure the parametricity of an operation for location-equality testing. The last two conditions ensure the parametricity of operations for updating and accessing of state components.

An $\mathcal{L}_e$-morphism from
$$\big(R: L(n_0) \leftrightarrow L(n_1), R': S(n_0) \leftrightarrow S(n_1)\big)$$
to
$$\big(T: L(m_0) \leftrightarrow L(m_1), T': S(m_0) \leftrightarrow S(m_1)\big)$$
is a quadruple $\big((R, R'), f_0, f_1, (T, T')\big)$ such that

- $f_0[R \to T]f_1$;
- for all $j_0 \in L(m_0)$ and $j_1 \in L(m_1)$, if $j_0[T]j_1$ then $j_0 \in f_0\big(L(n_0)\big) \iff j_1 \in f_1\big(L(n_1)\big)$;
- $\phi_{f_0}[T' \to R']\phi_{f_1}$; and
- $\rho_{f_0}[R' \times T' \to T']\rho_{f_1}$,

where functions
$$\begin{aligned}\phi_{f_0}&: S(m_0) \to S(n_0)\\ \rho_{f_0}&: S(n_0) \times S(m_0) \to S(m_0)\end{aligned}$$
are defined from f_0 by
$$\phi_{f_0}(x) = f_0 \,;\, x$$
and
$$\rho_{f_0}(w, x)(i) = \begin{cases} w(j), & \text{if } i = f_0(j) \\ x(i), & \text{otherwise,} \end{cases}$$
and functions
$$\begin{aligned}\phi_{f_1}&: S(m_1) \to S(n_1)\\ \rho_{f_1}&: S(n_1) \times S(m_1) \to S(m_1)\end{aligned}$$
are similarly defined from f_1.

The first, third, and fourth conditions say, respectively, that f_0 and f_1 map related locations to related locations, and that the induced "projection" and "replacement" maps are relation-preserving, as in $\mathcal{W}_e$. The second condition is perhaps less expected. It says that newly allocated locations, i.e., those not in the image of f_0 or f_1, cannot be related to old ones. If we were to omit this condition, the parametricity constraints would be *too* restrictive.

Finally, we define $\mathcal{L}_I$ by $\mathcal{L}_I(n) = (\Delta_{L(n)}, \Delta_{S(n)})$ and $\mathcal{L}_I(f: n \to m) = \big(\mathcal{L}_I(n), f, f, \mathcal{L}_I(m)\big)$, and $\mathcal{L}_{\delta_i}$ by $\mathcal{L}_{\delta_i}\big(R: L(n_0) \leftrightarrow L(n_1), R': S(n_0) \leftrightarrow S(n_1)\big) = n_i$ and $\mathcal{L}_{\delta_i}\big((R, R'), f_0, f_1, (T, T')\big) = f_i$.

We now interpret the basic types as parametric functors from $\mathcal{L}$ to $\mathcal{D}$ as follows.

$$\begin{aligned}
[\![\mathbf{var}]\!]n &= L(n)\\
[\![\mathbf{var}]\!]f &= f\\
[\![\mathbf{var}]\!](R, R') &= R\\
[\![\mathbf{exp}]\!]n &= S(n) \to Z_\perp\\
[\![\mathbf{exp}]\!]f\ e &= \phi_f \,;\, e\\
[\![\mathbf{exp}]\!](R, R') &= R' \to (\Delta_Z)_\perp\\
[\![\mathbf{comm}]\!]n &= S(n) \to \big(S(n)\big)_\perp
\end{aligned}$$
$$[\![\mathbf{comm}]\!]f\ c\ x = \begin{cases} \rho_f\big(c(\phi_f(x)), x\big), & \text{if } c\big(\phi_f(x)\big) \neq \perp \\ \perp, & \text{if } c\big(\phi_f(x)\big) = \perp \end{cases}$$
$$[\![\mathbf{comm}]\!](R, R') = R' \to (R')_\perp$$

where $L(n)$, $S(n)$, ϕ_f and ρ_f are defined as discussed previously.

We now define parametric natural transformations
$$\begin{aligned}
&\mathit{assign}: [\![\mathbf{var}]\!] \times [\![\mathbf{exp}]\!] \dot{\to} [\![\mathbf{comm}]\!]\\
&\mathit{deref}: [\![\mathbf{var}]\!] \dot{\to} [\![\mathbf{exp}]\!]\\
&\mathit{new}: [\![\mathbf{var} \to \mathbf{comm}]\!] \dot{\to} [\![\mathbf{comm}]\!]\\
&\mathit{eqloc}: [\![\mathbf{var}]\!] \times [\![\mathbf{var}]\!] \dot{\to} [\![\mathbf{exp}]\!]
\end{aligned}$$
as follows: for every natural number n,

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

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 $w \in S(n)$;

- $new(n)(p)(w)$
$$= \begin{cases} f\,; w', & \text{if } p(f)(n)(w \mid n \mapsto \bar{z}) = w' \\ \perp, & \text{if } p(f)(n)(w \mid n \mapsto \bar{z}) = \perp \end{cases}$$

where $p \in [\![\mathbf{var} \to \mathbf{comm}]\!]n$, $w \in S(n)$ 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 parametricity constraints, it is possible to validate equivalences like Example 6 in [MS88].

7 Discussion

The interpretations of Algol presented here are the best denotational models of local variables currently available. They support very straightforward 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 is not validated.

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.

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

- strengthening Proposition 3 (possibly using ideas from [FRR92]) to a result for reflexive graphs comparable to the following property of conventional functor categories [Nel81]: let $\mathbf{X}$ be any small category; if $\mathbf{Y}$ is *complete* and cartesian closed, so is $\mathbf{X} \Rightarrow \mathbf{Y}$;

- application to the semantic description of fully dynamic storage allocation and other dynamic-allocation regimes (cf. [Mog90, Pit92]); 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."

Acknowledgements

We are grateful to Andy Pitts, Edmund Robinson, John Power and Barry Jay for advice on categorical matters, and to Benli Pierce, Phil Wadler and Qingming Ma for comments. John Reynolds's macros were used for diagrams.

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.

[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, Berlin, 1991.

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

[FRR92] P. J. Freyd, E. P. Robinson, and G. Rosolini. Functorial parametricity. In [LICS92], pages 444–452.

[Gol79] R. Goldblatt. *Topoi, The Categorial Analysis of Logic.* North-Holland, Amsterdam, 1979.

[Gri78] D. Gries, editor. *Programming Methodology, A Collection of Articles by IFIP WG 2.3.* Springer-Verlag, New York, 1978.

[Hoa72] C. A. R. Hoare. Correctness of data representations. *Acta Informatica*, 1:271–281, 1972. Reprinted in [Gri78], pages 269-281.

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

[Joh89] P. T. Johnstone. Affine categories and naturally Mal'cev categories. *Journal of Pure and Applied Algebra*, 61:251–256, 1989.

[Kri65] S. A. Kripke. Semantical analysis of intuitionistic logic I. In J. N. Crossley and M. A. E. Dummett, editors, *Formal Systems and Recursive Functions*, pages 92–130. North-Holland, Amsterdam, 1965.

[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, 1972/1973*, volume 420 of *Lecture Notes in Mathematics*, pages 75–103. Springer-Verlag, Berlin, 1974.

[Law89] F. W. Lawvere. Qualitative distinctions between some toposes of generalized graphs. In J. W. Gray and A. Scedrov, editors, *Categories in Computer Science and Logic*, volume 92 of *Contemporary Mathematics*, pages 261–300. American Mathematical Society, 1989.

[Len92] A. F. Lent. The category of functors from state shapes to bottomless CPOs is adequate for block structure. Master's thesis, Massachusetts Institute of Technology, 1992.

[LICS92] *Proceedings, 7th Annual IEEE Symposium on Logic in Computer Science*, Santa Cruz, California, 1992. IEEE Computer Society Press, Los Alamitos, California.

[Mit86] J. C. Mitchell. Representation independence and data abstraction. In *Conf. Record 13th ACM Symp. on Principles of Programming Languages*, pages 263–276, St. Petersburg, Florida, 1986. ACM, New York.

[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 [LICS92], pages 186–197.

[NB+63] P. Naur, J. W. Backus, et al. Revised report on the algorithmic language ALGOL 60. *Comm. ACM*, 6(1):1–17, 1963.

[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.

[Ole85] F. J. Oles. Type algebras, functor categories and block structure. In M. Nivat and J. C. Reynolds, editors, *Algebraic Methods in Semantics*, pages 543–573. Cambridge University Press, Cambridge, England, 1985.

[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, England, 1992.

[Pie91] B. C. Pierce. *Basic Category Theory for Computer Scientists*. The MIT Press, Cambridge, Mass., 1991.

[Pit92] A. M. Pitts. A model of the ν-calculus. Unpublished notes, January 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, Rocquencourt, France, 1975. Reprinted in [Gri78], pages 309-317.

[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*.

[Str67] C. Strachey. *Fundamental Concepts in Programming Languages*. Unpublished lecture notes, International Summer School in Computer Programming, Copenhagen, August 1967.

[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.

[THM83] B. A. Trakhtenbrot, J. Y. Halpern, and A. R. Meyer. From denotational to operational and axiomatic semantics for ALGOL 60-like languages: an overview. In E. M. Clarke and D. Kozen, editors, *Logics of Programs 1983*, volume 164 of *Lecture Notes in Computer Science*, pages 474–500, Pittsburgh, PA, 1983. Springer-Verlag, Berlin, 1984.