Parametricity and Local Variables

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

R. D. Tennent†
Department of Computing and Information Science
Queen's University
Kingston, Ontario, Canada K7L 3N6
`rdt@qucis.queensu.ca`

Abstract

We propose that the phenomenon of local state may be understood in terms of Strachey's concept of parametric (i.e., uniform) polymorphism. The intuitive basis for our proposal is the following analogy: a non-local procedure is independent of locally-declared variables in the same way that a parametrically polymorphic function is independent of types to which it is instantiated.

A connection between parametricity and representational abstraction was first suggested by J. C. Reynolds. Reynolds used logical relations to formalize this connection in languages with type variables and user-defined types. We use relational parametricity to construct a model for an Algol-like language in which interactions between local and non-local entities satisfy certain relational criteria. Reasoning about local variables essentially involves proving properties of polymorphic functions. The new model supports straightforward validations of all the test equivalences that have been proposed in the literature for local-variable semantics, and encompasses standard methods of reasoning about data representations. It is not known whether our techniques yield fully abstract semantics. A model based on partial equivalence relations on the natural numbers is also briefly examined.

Categories and Subject Descriptors: D.3.1 [**Programming Languages**]: Formal Definitions and Theory - *semantics*; F.3.2 [**Logics and Meanings of Programs**]: Semantics of Programming Languages - *denotational semantics*.

General Terms: Languages, Theory.

Additional Key Words and Phrases: parametric polymorphism, logical relations, Algol-like languages, local state.

---

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

†This author was supported by an operating grant from the Natural Sciences and Engineering Research Council of Canada and a research fellowship from the Science and Engineering Research Council of Great Britain.

Contents

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 (Naur et al., 1963). 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 (Meyer and Sieber, 1988; O'Hearn and Tennent, 1992). 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 (Meyer and Sieber, 1988):

```
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 $P(\mathit{skip})$, where *skip* does nothing, for *every* possible meaning of P. But this equivalence fails in all previous denotational models of local variables!

The reader's reaction to this example might be that it is contrived, and that it 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" syntax, and one could write something like

```
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, 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 (Reynolds, 1978) to make the same point.) See (Reynolds, 1981b; Tennent, 1991) for discussion of Algol-like languages, and (Reynolds, 1975; Cook, 1991) 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 (Meyer and Sieber, 1988; O'Hearn and Tennent, 1993b) support validation of *invariance* principles often used for reasoning about data representations, as in (Hoare, 1972). 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
```

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
  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; see, for example, (Mitchell, 1986). 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 (Hoare, 1972)(Reynolds, 1981a, 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!

This discussion of data abstraction motivates our link with the concept of *parametricity*, introduced by Strachey (1967) in the following remarks:

> There seem to be two main classes [of polymorphism], which can be called *ad hoc* polymorphism and parametric polymorphism.
>
> In *ad hoc* polymorphism there is no single systematic way of determining the type of the result from the type of the arguments. There may be several rules of limited extent which reduce the number of cases, but these are themselves *ad hoc* both in scope and content. All the ordinary arithmetic operators and functions come into this category. It seems, moreover, that the automatic insertion of transfer functions by the compiling system is limited to this class.
>
> Parametric polymorphism is more regular and may be illustrated by an example. Suppose f is a function whose argument is of type α and whose result is of type β (so that the type of f might be written $\alpha \Rightarrow \beta$), and that L is a list whose elements are all of type α (so that the type of L is $\alpha\,\mathbf{list}$). We can imagine a function, say *Map*, which applies f in turn to each member of L and makes a list of the results. Thus $Map[f, L]$ will produce a $\beta\,\mathbf{list}$. We would like *Map* to work on all types of list provided f was a suitable function, so that *Map* would have to be polymorphic. However its polymorphism is of a particularly simple parametric type which could be written
>
> $$(\alpha \Rightarrow \beta, \alpha\,\mathbf{list}) \Rightarrow \beta\,\mathbf{list}$$
>
> where α and β stand for any types.

Although a complete understanding of the ramifications of this notion of parametricity is not yet available (cf., (Freyd et al., 1992b; Plotkin and Abadi, 1993)), Reynolds (1974; 1983) has emphasized the close relationship with representational abstraction. The idea is that a parametric polymorphic function must work in a way that is independent of the types to which it is instantiated. For instance, (in the absence of recursion) the only parametric elements of type $\forall\alpha.\,\alpha \rightarrow \alpha \rightarrow \alpha$ are the two functions with two arguments that return either the first argument or the second argument, respectively. On the other hand, a function that would return its first argument when instantiated to a function on integers, and its second otherwise, is *not* parametric because it works differently at different types. Intuitively, a parametric function cannot make use of knowledge about the types to which it is instantiated, which is to say that type variables are treated "abstractly."

We propose that the *independence* of non-local entities and local variables is in essence similar to the sense in which a parametric function is *independent* of the specific types to which it is instantiated. Stated in terms of abstraction, the principle that a non-local procedure cannot access a local variable (except through using arguments that access the variable) is analogous to the principle that the representation of an abstract type cannot be directly accessed by programs that use it (except through the provided operations of the type). We will define a semantics for an Algol-like language in which non-local procedures are modeled as parametric functions that can be instantiated with pieces of local state. The independence of the procedure itself from a local variable will then be explained in terms of the independence of a polymorphic function from type arguments, which here play the role of local state.

The approach to representational abstraction that we will follow is based on the work of (Reynolds, 1983), where the technique of "logical" relations (Plotkin, 1980; Mitchell, 1990) was used to give a rigorous formulation of abstraction that is appropriate for functional languages with higher-order and polymorphic procedures and programmer-defined types. We can illustrate 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 sets, regarded as alternative "representations" of the type variable;
- $[\![\theta]\!]W_0$ is the set 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 set 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 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 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 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 $([\![P]\!]W_0, [\![P]\!]W_1)$ is a *relation-preserving* pair of functions; i.e., for all $u_0 \in [\![\pi]\!]W_0$ and $u_1 \in [\![\pi]\!]W_1$,

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

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 (Strachey, 1967) and (Reynolds, 1983), 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.

The connection between logical relations and polymorphic functions emphasized by Reynolds is that if the above relation-preservation property is to hold in a polymorphic language, then values of $\forall$-types *must* be constrained so as to satisfy similar relation-preservation conditions. In our model for local variables, function types will themselves have a polymorphic flavour, and will be constrained by such a parametricity condition.

Relational parametricity is commonly thought to prescribe *necessary* properties that parametric functions must satisfy. What is less clear is whether, particularly in the binary-relation form, it is *sufficient* to characterize the intuitive concept. Another appealing approach to parametricity, possessing a fairly coherent conceptual basis, uses partial equivalence relations (PERs); e.g., (Longo and Moggi, 1991). In the PER approach, polymorphic types are interpreted as infinitary intersections, so that a (realizer for a) polymorphic function is an untyped meaning that is type-correct for all instantiations of a type variable. This captures, to a certain (not completely understood) degree, the intuition that a polymorphic function is given by a uniform algorithm. On the other hand, the relational approach captures, to a certain (not completely understood) degree, intuitions about representation independence.

The larger part of our effort in this paper will be directed toward examining the relational approach of Reynolds. The semantic model we define will represent quite directly the informal reasoning about local variables and data abstraction alluded to above. However, we will also briefly outline how PERs can be used to treat variable declarations. A comparison of the two models will be given in Section 11.

Our method of incorporating parametricity builds on the functor-category approach to local variables pioneered by Reynolds (1981b) (Reynolds, 1981b) and Oles (1982, 1985). In the remainder of this Introduction, we will briefly review the basic elements of this approach, and indicate how relational parametricity will enter the picture. (The expository article (O'Hearn and Tennent, 1992) and textbook (Tennent, 1991) contain introductions to this approach.)

The key insight of the Reynolds-Oles work is that, in a language with local-variable declarations, the concept of state is not constant—represented by a single set of states—but rather varies as storage variables are allocated and de-allocated. That is, there are *different* possible sets of states depending on the "shape" of the run-time stack; i.e., the number and type of variables that have been allocated.

To account for this, the semantics is *parametrized* by abstract "store shapes," effectively building the variance in the concept of state into the semantics in a way that logically precedes any assignment of meanings to phrases. In general, the meaning of a type is not a single domain, but a whole family of domains. For example, the type of commands is often interpreted as $S \rightarrow S_\bot$, where S is a set of states. But in a language with variable declarations S itself varies, and so there is a domain $X \rightarrow X_\bot$ for *each* possible set X of states. In particular, a local-variable declaration changes the set of states from X to $X \times Y$, where Y is the set of values the new variable may hold; the Y-valued component of each element of $X \times Y$ represents the new variable. Similarly, if the the domain of command meanings prior to a variable declaration is $X \longrightarrow X_{\perp}$, then, after declaration, it becomes $X \times Y \longrightarrow (X \times Y)_{\perp}$.

The semantic set-up can be elegantly described using basic concepts of category theory. The variance in the concept of state is modeled using a category of "possible worlds." Each possible world determines the set of storage states needed to represent the values of currently available variables, and a morphism of worlds "expands" the current state by allocating space for additional storage variables (the Y-valued component above). This variance in the concept of state induces a similar variance into types, which is represented by interpreting types 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 the meaning of any phrase P is portrayed by the following commutative diagram:

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

where $f\colon W_0 \longrightarrow W_1$ is a morphism of possible worlds, and $\llbracket \theta \rrbracket$ and $\llbracket \pi \rrbracket$ are type and environment functors.

Notice that, in many respects, this is similar to the relational-parametricity picture discussed earlier. Parametrization by possible worlds is roughly analogous to abstraction on a type variable. In fact, if we think of the possible worlds as certain kinds of types, then $\llbracket P \rrbracket$ is a family of functions indexed by these types and so is, in a certain sense, polymorphic. It is therefore certainly conceivable to require a family of this form to satisfy a parametricity constraint.

This analogy between possible worlds and type variables suggests how relational parametricity can be incorporated. We consider binary relations between worlds, regarded as relating different "representations" of the store shape, and the semantics of types is then arranged so that each such relation induces a relation between the meanings of a type at different store shapes. The meanings of terms are then families of maps satisfying a relational-parametricity constraint.

The naturality requirements of Reynolds and Oles will not be abandoned. However, to make the presentation more accessible, we will begin with a "category-free" description of our model. The naturality conditions are implicit in this presentation, but will later be shown to be implied by relational parametricity.

This category-free description has the advantage of being quite simple, and it also puts the role of parametricity clearly on display. But a consideration of relevant category-theoretic issues is crucial for a deeper understanding of the model. The category-free presentation appears very *ad hoc* in some respects; a fully satisfactory justification for some of the definitions will come from categorical considerations. Further, while we will show that in certain circumstances naturality is implied by relational parametricity, it must be emphasized that, in general, these are *different* kinds of uniformity, with neither being stronger than the other. It will be seen, in fact, that the connection between these two concepts is somewhat delicate. (In Section 9, we show an example where "parametricity implies naturality" is not stable under

Currying isomorphisms; this, for us, came as a surprise.)

To study this combination of relational parametricity and naturality, we will define a suitable cartesian closed category 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 will be taken up in Sections 7–9. The earlier parts of the paper are devoted to the category-free presentation of the model.

2 Types

2.1 Syntax

Our language is an Algol-like language in the sense of (Reynolds, 1981b). The language does not include jumps or subtypes, but it raises the key issues related to variable allocation. The types are as follows:

$$\delta ::= \mathbf{int} \mid \mathbf{bool} \cdots \qquad \text{(data types)}$$

$$\beta ::= \mathbf{comm} \mid \mathbf{exp}[\delta] \mid \mathbf{var}[\delta] \qquad \text{(primitive phrase types)}$$

$$\theta ::= \beta \mid \vec{\theta} \longrightarrow \beta \qquad \text{(phrase types)}$$

Data types are the types of values that can be stored, while phrase types consist of meanings that can be denoted by identifiers and, more generally, program phrases, but that cannot be stored. This distinction allows variable declarations to obey a stack discipline.

comm is the type of commands. Executing a command causes a state change, but does not return a value. $\mathbf{var}[\delta]$ is the type of storage variables that accept δ-typed values. $\mathbf{exp}[\delta]$ is the type of expressions that return values of type δ. Expressions are "read only," in that they are state dependent but do not cause side effects. So all state changes are concentrated in the type **comm**. In particular, procedures use by-name parameters, and can only change the state indirectly, by being used within a phrase of type **comm**.

In procedure types $\vec{\theta}$ is a non-empty sequence of phrase types. This "uncurried" formulation of the syntax of procedure types is not essential, but is most amenable to a category-free description of the model.

2.2 Semantics

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 we will work exclusively with *binary* relations, our definitions (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 $$\begin{tikzcd} W_0 \arrow[d, leftrightarrow, "R"'] \\ W_1 \end{tikzcd}$$ to mean that R is a binary relation with domains W_0 and W_1, and $w_0[R]w_1$ to mean $\langle w_0, w_1\rangle \in \operatorname{graph} R$.

If W is any set,

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

We use $W \to X$ and $W \times X$ for the function space and product of sets. 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 $\langle w_0, x_0\rangle[R \times S]\langle w_1, x_1\rangle \iff w_0[R]w_1$ and $x_0[S]x_1$;
- $R \to S\colon W_0 \to X_0 \longleftrightarrow W_1 \to X_1$ is defined by $f_0[R \to S]f_1 \iff$ for all $w_0 \in W_0$, $w_1 \in W_1$, if $w_0[R]w_1$ then $f_0(w_0)[S]f_1(w_1)$. We often use the diagrammatic notation

  $$\begin{tikzcd} W_0 \arrow[r, "f_0"] \arrow[d, leftrightarrow, "R"'] & W_1 \arrow[d, leftrightarrow, "S"] \\ X_0 \arrow[r, "f_1"'] & X_1 \end{tikzcd}$$

  to indicate that $f_0[R \to S]f_1$. Notice that this notation makes the domains of the relations, and the domains and codomains of the functions, evident.

The collection Σ of "store shapes" is a set of sets that includes desired data types, such as $2 = \{\mathit{true}, \mathit{false}\}$ and $Z = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$, and all finite (set) products of these. We won't be more specific about Σ, except to emphasize that it must be a small collection. For each data type δ, we assume a set $[\![\delta]\!]$ in Σ, with $[\![\mathbf{int}]\!] = Z$ and $[\![\mathbf{bool}]\!] = 2$.

Following (Reynolds, 1983), we define a "twin" semantics of phrase types, where each θ determines two functions

$$[\![\theta]\!] : \Sigma \longrightarrow \mathbf{Sets}$$

$$[\![\theta]\!] : \mathrm{rel}(\Sigma) \longrightarrow \mathrm{rel}(\mathbf{Sets}).$$

Here, **Sets** stands for the class of sets, rel(**Sets**) for the class of binary relations between sets, and rel(Σ) for the set of binary relations between store shapes. The relational component of the semantics will be used to enforce parametricity constraints.

The interpretation of the command type is as follows:

- for every store shape W in Σ,

  $$[\![\mathbf{comm}]\!]W \;=\; W \to W \text{ ; and}$$

- for every $R\colon W_0 \leftrightarrow W_1$ in rel(Σ),

  $$[\![\mathbf{comm}]\!]R \;=\; R \to R.$$

For expressions:

- for every store shape W in Σ,

  $$[\![\mathbf{exp}[\delta]]\!]W \;=\; W \to [\![\delta]\!] \text{ ; and}$$

- for every relation $R\colon W_0 \leftrightarrow W_1$ in rel(Σ),

  $$[\![\mathbf{exp}[\delta]]\!]R \;=\; R \to \Delta_{[\![\delta]\!]}.$$

For variables:

- for every store shape W in Σ,

$$[\![\mathbf{var}[\delta]]\!]W \;=\; ([\![\delta]\!] \longrightarrow [\![\mathbf{comm}]\!]W) \times [\![\mathbf{exp}[\delta]]\!]W \quad ;\ \text{and}$$

- for every relation $R\colon W_0 \leftrightarrow W_1$ in $\mathrm{rel}(\Sigma)$,

$$[\![\mathbf{var}[\delta]]\!]R \;=\; (\Delta_{[\![\delta]\!]} \longrightarrow [\![\mathbf{comm}]\!]R) \times [\![\mathbf{exp}[\delta]]\!]R\ .$$

The two components of a variable allow for, respectively, updating and accessing its contents. This "object-oriented" approach to variables is from (Reynolds, 1981b).

For vectors $\vec{\theta} = \theta_1, ..., \theta_n$:

- for every store shape W in Σ, $[\![\vec{\theta}]\!]W \;=\; [\![\theta_1]\!]W \times \cdots \times [\![\theta_n]\!]W$; and
- for every $R\colon W_0 \leftrightarrow W_1$ in $\mathrm{rel}(\Sigma)$, $[\![\vec{\theta}]\!]R \;=\; [\![\theta_1]\!]R \times \cdots \times [\![\theta_n]\!]R$

Officially, meanings for types $[\![\theta]\!]$ and vectors $[\![\vec{\theta}]\!]$ are defined by a simultaneous induction.

For procedure types one might expect to use a pointwise definition, where $[\![\vec{\theta} \longrightarrow \beta]\!]W = [\![\vec{\theta}]\!]W \longrightarrow [\![\beta]\!]W$, and similarly for the relation part. However, a pointwise definition is *not* appropriate in the present context. The reason is that we think of the sets in Σ as "store shapes," which can grow between the point of definition of a procedure and the point of call. For example, if the store shape is W when a procedure identifier P is bound, and P is called after an integer variable is declared, then the shape of the stack for the call will be $W \times Z$, not W.

$$\textit{binding of } P \cdots \;\; \mathbf{begin\ integer}\, x;\ \cdots\ P(\cdots x \cdots)\ \cdots\ \mathbf{end}$$

Thus, a procedure meaning at store shape W must be applicable at an expanded shape $W \times X$, where X corresponds to additional variables that have been allocated. This is accounted for in (Reynolds, 1981b; Oles, 1982) by defining a procedure meaning to be a *family* of functions, indexed by extra components X representing pieces of local state that can be added to the stack. We will follow the same route here, except that these families of functions will be subject to parametricity conditions.

A procedure type $\vec{\theta} \longrightarrow \beta$ is interpreted as follows.

- For every store shape W in Σ,

  $$[\![\vec{\theta} \rightarrow \beta]\!]W \;=\; \forall X.\, [\![\vec{\theta}]\!](W \times X) \longrightarrow [\![\beta]\!](W \times X)\ ;$$

  that is, $p \in [\![\vec{\theta} \longrightarrow \beta]\!]W$ is a *family* of functions

  $$p[-]\colon [\![\vec{\theta}]\!](W \times -) \longrightarrow [\![\beta]\!](W \times -)$$

  indexed by store shapes X, satisfying the following parametricity constraint: for all relations $R\colon X_0 \leftrightarrow X_1$ between store shapes,

$$\begin{tikzcd}
[\![\vec{\theta}]\!](W \times X_0) \arrow[r, "{p[X_0]}"] \arrow[d, leftrightarrow, "{[\![\vec{\theta}]\!](\Delta_W \times R)}"'] & [\![\beta]\!](W \times X_0) \arrow[d, leftrightarrow, "{[\![\beta]\!](\Delta_W \times R)}"] \\
[\![\vec{\theta}]\!](W \times X_1) \arrow[r, "{p[X_1]}"'] & [\![\beta]\!](W \times X_1)
\end{tikzcd}$$

Function $p[X]$ models the behaviour of the procedure instantiated at the "expanded" store shape $W \times X$.

- For every relation $R\colon X_0 \leftrightarrow X_1$ in $\mathrm{rel}(\Sigma)$, $p[\llbracket\vec{\theta} \rightarrow \beta\rrbracket R]\,q$ iff, for all relations $S\colon Y_0 \leftrightarrow Y_1$ in $\mathrm{rel}(\Sigma)$,

$$\begin{tikzcd}
\llbracket\vec{\theta}\rrbracket(X_0 \times Y_0) \arrow[r, "p_0[Y_0]"] \arrow[d, leftrightarrow, "\llbracket\vec{\theta}\rrbracket(R \times S)"'] & \llbracket\beta\rrbracket(X_0 \times Y_0) \arrow[d, leftrightarrow, "\llbracket\beta\rrbracket(R \times S)"] \\
\llbracket\vec{\theta}\rrbracket(X_1 \times Y_1) \arrow[r, "p_1[Y_1]"'] & \llbracket\beta\rrbracket(X_1 \times Y_1)
\end{tikzcd}$$

Notice how the relational and domain-theoretic semantics become intertwined at this point. This is motivated by the use of a relational condition to constrain values of $\forall$ types in (Reynolds, 1983). The identity relation Δ_W plays the same role as the identity relations there. (Of course, the foundational difficulties described in (Reynolds, 1983; Reynolds and Plotkin, 1993) do not arise here, because the source collection Σ, over which indexing is done, is small.)

2.3 Recursion

The presentation thus far is for a recursion-free dialect of Algol. Recursion can be dealt with by using domains in place of sets, as follows. (We still use sets, or discretely-ordered predomains, for the store shapes.)

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 $\perp$.

If D and E are directed-complete partially-ordered sets then a relation $R\colon D \leftrightarrow E$ is

- *complete*, if its graph is a directed-complete subset of the pointwise-ordered product of the domains of the relation; and
- *pointed*, if D and E are pointed and R relates their least elements.

The semantics can then be defined by mapping store shapes to domains, and relations on store shapes to pointed complete relations on domains. For the command type:

- for every store shape W in Σ,

$$\llbracket\mathbf{comm}\rrbracket W \;=\; W \rightarrow W_\perp \text{ ; and}$$

- for every $R\colon W_0 \leftrightarrow W_1$ in $\mathrm{rel}(\Sigma)$,

$$\llbracket\mathbf{comm}\rrbracket R \;=\; R \rightarrow R_\perp .$$

Here, the $\longrightarrow$ acts on (pre)domains as the continuous-function space constructor, and on complete relations by producing the evident complete relation on the function spaces. The definitions of the other base types can be modified in a similar fashion, and procedure types are exactly as before, but with the $\longrightarrow$ in the definition understood as constructing the continuous-function space, and the families $p[-]$ ordered component-wise.

The restriction to complete relations is standard. It is needed for the fixed-point operator to satisfy the appropriate parametricity constraints, and also for domain-theoretic structure to be respected when using parametricity to constrain procedure types. As the consideration of recursion would add little to our discussion of locality, we will for simplicity concentrate on the set-theoretic semantics in the remainder of the paper.

3 Properties of Types

3.1 Basic Properties

We now turn to some basic properties satisfied by this semantics. These are all essentially consequences of the polymorphic view of phrase types sketched in the previous section.

First, as in (Reynolds, 1983), each $\llbracket\theta\rrbracket$ preserves identity relations.

Lemma 1 (Identity Extension)

For each phrase type θ and store shape W, $\llbracket\theta\rrbracket\Delta_W = \Delta_{\llbracket\theta\rrbracket W}$.

Proof: By induction on types. For base types this is immediate.

We will consider the function type in some detail to indicate how the proofs go. For $\vec{\theta} \longrightarrow \beta$, if $p \in \llbracket\vec{\theta} \longrightarrow \beta\rrbracket W$ then, by definition, $p[\llbracket\vec{\theta} \longrightarrow \beta\rrbracket\Delta_W]p$ iff, for all $R\colon X_0 \leftrightarrow X_1$,

$$\begin{tikzcd}
\llbracket\vec{\theta}\rrbracket(W \times X_0) \arrow[r, "{p[X_0]}"] \arrow[d, leftrightarrow, "\llbracket\vec{\theta}\rrbracket(\Delta_W \times R)"'] & \llbracket\beta\rrbracket(W \times X_0) \arrow[d, leftrightarrow, "\llbracket\beta\rrbracket(\Delta_W \times R)"] \\
\llbracket\vec{\theta}\rrbracket(W \times X_1) \arrow[r, "{p[X_1]}"'] & \llbracket\beta\rrbracket(W \times X_1)
\end{tikzcd}$$

As this is none other than the parametricity constraint on procedure meanings, we may conclude that $\llbracket\vec{\theta} \longrightarrow \beta\rrbracket\Delta_W$ contains the diagonal.

Conversely, if $p[\llbracket\vec{\theta} \longrightarrow \beta\rrbracket\Delta_W]q$ then, for $R\colon X_0 \leftrightarrow X_1$,

$$\begin{tikzcd}
\llbracket\vec{\theta}\rrbracket(W \times X_0) \arrow[r, "{p[X_0]}"] \arrow[d, leftrightarrow, "\llbracket\vec{\theta}\rrbracket(\Delta_W \times R)"'] & \llbracket\beta\rrbracket(W \times X_0) \arrow[d, leftrightarrow, "\llbracket\beta\rrbracket(\Delta_W \times R)"] \\
\llbracket\vec{\theta}\rrbracket(W \times X_1) \arrow[r, "{q[X_1]}"'] & \llbracket\beta\rrbracket(W \times X_1)
\end{tikzcd}$$

In particular, taking R as a diagonal Δ_X and applying the induction hypothesis (both for $\vec{\theta}$ and for β) gives that $p[X]a = q[X]a$ for all X and $a \in \llbracket\vec{\theta}\rrbracket X$, and so $p = q$. (We are using the fact that the identity property can be seen to hold for $\llbracket\vec{\theta}\rrbracket$ whenever it holds for each element of the vector.) ■

A further related property, emphasized in (Freyd et al., 1992b), is that each $\llbracket\theta\rrbracket$ is functorial on isomorphisms. We say that a relation in $\mathrm{rel}(\Sigma)$ (respectively, $\mathrm{rel}(\mathbf{Sets})$) is an isomorphism iff it is the graph of a bijection. (In a domain-theoretic model, we would consider continuous isomorphisms, i.e. continuous, order-reflecting bijections.)

It will be well, for future reference, to have an explicit description of functional isomorphisms induced by bijections between store shapes (even though these isomorphisms could alternatively be read off from the semantics of types, using a relational isomorphism). If $f\colon W \longrightarrow X$ is a bijection between store shapes then the isomorphism $f_\theta\colon \llbracket\theta\rrbracket W \longrightarrow \llbracket\theta\rrbracket X$ is defined as follows.

$$\begin{array}{lcl}
f_{\mathbf{comm}} & = & f^{-1} \longrightarrow f \\
f_{\mathbf{exp}[\delta]} & = & f^{-1} \longrightarrow \mathrm{id}_{\llbracket\delta\rrbracket} \\
f_{\mathbf{var}[\delta]} & = & (\mathrm{id}_{\llbracket\delta\rrbracket} \longrightarrow f_{\mathbf{comm}}) \times f_{\mathbf{exp}[\delta]} \\
f_{\vec{\theta}\to\beta} & = & \lambda p\, \Lambda Y.\, (f^{-1} \times Y)_{\vec{\theta}}\ ;\ p[Y]\ ;\ (f \times Y)_{\beta}
\end{array}$$

Here we are using the action of exponentiation $\longrightarrow$ and product $\times$ (in the category of sets) on morphisms, and id is an identity. In the last equation, the right-hand side denotes the function that takes $p \in \llbracket\vec{\theta} \longrightarrow \beta\rrbracket W$ and a store shape Y to the bottom of the following diagram

$$\begin{tikzcd}
\llbracket\vec{\theta}\rrbracket(X \times Y) \arrow[r, "pY"] & \llbracket\beta\rrbracket(X \times Y) \arrow[d, "(f \times Y)_{\beta}"] \\
\llbracket\vec{\theta}\rrbracket(X \times Y) \arrow[u, "(f^{-1} \times Y)_{\vec{\theta}}"] \arrow[r] & \llbracket\beta\rrbracket(X \times Y)
\end{tikzcd}$$

where $f_{\vec{\theta}}$ for vectors is defined in the obvious component-wise way.

Lemma 2 (Isomorphism Functoriality)

Each $\llbracket\theta\rrbracket$ is functorial on isomorphisms. That is, for all isomorphisms $R\colon W \leftrightarrow X$ in $\mathrm{rel}(\Sigma)$,

1. *if R is an isomorphism then so is the induced relation $\llbracket\theta\rrbracket R$, and*

2. *if $R\colon X \leftrightarrow Y$ and $S\colon Y \leftrightarrow W$ are isomorphisms, then $\llbracket\theta\rrbracket R\,;\,\llbracket\theta\rrbracket S = \llbracket\theta\rrbracket(R\,;\,S)$ where semicolon is relational composition.*

(Preservation of identities is the identity extension lemma.) Furthermore, if $f\colon W \longrightarrow X$ is a bijection between store shapes and $R_f\colon W \leftrightarrow X$ is the relation with the same graph as f then the relation $\llbracket\theta\rrbracket R_f$ and the function f_θ have the same graph.

Proof: First, showing that f_θ is iso follows by a straightforward argument, where the function type case is much as in the proof of the identity extension lemma. Second, that $\llbracket\theta\rrbracket R_f$ and the function f_θ have the same graph can be shown by induction on types, where the function-type case follows immediately from the induction hypothesis. It is then not difficult to show that $(\cdot)_\theta$ preserves identities and composites, when applied to bijections. ■

Of course, relational composition is not preserved for *all* relations.

In the following, much use will be made of the canonical unity and associativity isomorphisms between store shapes. (Here, $\mathbf{1}$ is a singleton store shape.)

$$\mathit{unl}\colon W \times \mathbf{1} \longrightarrow W \qquad \mathit{unr}\colon W \longrightarrow W \times \mathbf{1}$$

$$
\begin{aligned}
\mathit{assl}\colon W \times (X \times Y) &\rightarrow (W \times X) \times Y \\
\mathit{assr}\colon (W \times X) \times Y &\rightarrow W \times (X \times Y).
\end{aligned}
$$

These isomorphisms satisfy a special parametricity property.

Lemma 3 (Canonical-Isomorphism Parametricity)

If $R_i\colon X_i \leftrightarrow Y_i$ *are relations between store shapes, for* $i = 1, 2, 3$, *then, for all types* θ,

$$
\mathit{unl}\Big[[\![\theta]\!](R_1 \times \Delta_{\mathbf{1}}) \rightarrow [\![\theta]\!]R_1\Big]\,\mathit{unl} \qquad \mathit{unr}\Big[[\![\theta]\!]R_1 \rightarrow [\![\theta]\!](R_1 \times \Delta_{\mathbf{1}})\Big]\,\mathit{unr}
$$

$$
\mathit{assl}\Big[[\![\theta]\!](R_1 \times (R_2 \times R_3)) \rightarrow [\![\theta]\!]((R_1 \times R_2) \times R_3)\Big]\,\mathit{assl}
$$

$$
\mathit{assr}\Big[[\![\theta]\!]((R_1 \times R_2) \times R_3) \rightarrow [\![\theta]\!](R_1 \times (R_2 \times R_3))\Big]\,\mathit{assr}
$$

Proof: A routine induction on θ. ■

Notice that the notation for these canonical isomorphisms does not make the domains and codomains explicit. Perhaps we could write, e.g., $\mathit{assl}_{X_1 X_2 X_3\,\theta}$; however, no ambiguity will arise as the relevant information will always be clear from context.

3.2 Expansions

There is further structure in the semantics that derives from the conception of elements of Σ as representing "shapes" of the run-time stack. Specifically, the expansion of store shapes caused by variable declarations is accompanied by mappings that convert semantic entities at a shape W to any expanded shape $W \times X$.

If W and X are store shapes, for each type θ we define a function

$$
\mathit{expand}_{\theta}(W, X)\colon [\![\theta]\!]W \longrightarrow [\![\theta]\!](W \times X)
$$

This goes by induction on types.

$$
\begin{aligned}
\mathit{expand}_{\mathbf{comm}}(W, X)\, c\, \langle w, x\rangle &= \langle c\, w\ , x\rangle \\[1ex]
\mathit{expand}_{\mathbf{exp}[\delta]}(W, X)\, e\, \langle w, x\rangle &= e\, w \\[1ex]
\mathit{expand}_{\mathbf{var}[\delta]}(W, X) &= (\mathrm{id}_{[\![\delta]\!]} \rightarrow \mathit{expand}_{\mathbf{comm}}(W, X)) \times (\mathit{expand}_{\mathbf{exp}[\delta]}(W, X)) \\[4ex]
\mathit{expand}_{\vec{\theta} \longrightarrow \beta}(W, X)\, p\, Y &= \mathit{assr}_{\vec{\theta}}\ ;\ p[X \times Y]\ ;\ \mathit{assl}_{\beta}
\end{aligned}
$$

This treatment of *expand* maps would surely benefit from a dose of category theory. For now we will push on and complete the concrete description of the model, leaving the tidying up of categorical matters to Sections 7–9.

There is a special uniformity property that the expansion functions satisfy. It states that expansions preserve relations on non-local states, and also produce meanings at expanded shapes that satisfy all relations on the local part of a store shape.

Lemma 4 (Expansion Parametricity)

If $R\colon W_0 \leftrightarrow W_1$ *and* $S\colon X_0 \leftrightarrow X_1$, *then*

$$\begin{tikzcd}
[\![\theta]\!]W_0 \arrow[r, "expand_\theta(W_0, X_0)"] \arrow[d, leftrightarrow, "[\![\theta]\!]R"'] & [\![\theta]\!](W_0 \times X_0) \arrow[d, leftrightarrow, "[\![\theta]\!](R \times S)"] \\
[\![\theta]\!]W_1 \arrow[r, "expand_\theta(W_1, X_1)"'] & [\![\theta]\!](W_1 \times X_1)
\end{tikzcd}$$

Proof: By induction on θ. Base types are immediate. We will indicate the proof for the function type.

Suppose $p_0[[\![\vec{\theta} \to \beta]\!]R]p_1$. For any $S\colon Y_0 \leftrightarrow Y_1$, the definition of $[\![\vec{\theta} \to \beta]\!]R$ implies

$$p_0[X_0 \times Y_0]\Big[[\![\vec{\theta}]\!](R \times (S \times Q)) \to [\![\beta]\!](R \times (S \times Q))\Big]p_1[X_1 \times Y_1] \;.$$

By the Canonical-Isomorphism Parametricity Lemma we get

$$(assr\, ; p_0[X_0 \times Y_0]\, ; assl)\Big[[\![\vec{\theta}]\!]((R \times S) \times Q) \to [\![\beta]\!]((R \times S) \times Q)\Big](assr\, ; p_1[X_1 \times Y_1]\, ; assl)$$

and, by the definition of *expand*, this is just what we wanted to show. ∎

4 Valuations

Whereas the category-free semantics of types is quite simple, the semantic equations for terms will turn out to be comparatively complex. This is a presentation trade-off: the valuations in the categorical semantics given later are much simpler, but require a more sophisticated interpretation of types.

A type assignment π is a finite function from (an unspecified set of) identifiers to phrase types. Some typing rules are in Table 1. The rules are in a natural deduction format. The rules for abstraction and application are for the uncurried syntax of types. The pairing construct uses the "object-oriented" approach to variables. We write write $\pi \vdash M : \theta$ to indicate that $M : \theta$ is derivable from (undischarged) assumptions π.

The example blocks in the Introduction can easily be desugared into this language. A block **begin** $\delta\, x\, ;\, C$ **end** is rendered as $\mathbf{new}_\delta(\lambda x\colon \mathbf{var}[\delta].\, C)$. We will arrange matters so **new** always assigns an initial value to the variable created. Of course, we could alternatively let the programmer supply this value, in which case the type of $\mathbf{new}_\delta$ would be

$$(\mathbf{exp}[\delta], (\mathbf{var}[\delta] \to \mathbf{comm})) \to \mathbf{comm}\;.$$

If π is a type assignment then the π-compatible environments, and relations between them, are as follows.

- for each store shape W, $[\![\pi]\!]W = \prod_{x \in \mathrm{dom}(\pi)} [\![\pi(x)]\!]W$; and
- for each relation $R\colon W_0 \leftrightarrow W_1$ between store shapes,

$$u_0\,[[\![\pi]\!]R]\,u_1 \iff \forall x \in \mathrm{dom}(\pi).\, u_0(x)\,[[\![\pi(x)]\!]R]\,u_1(x).$$

$$\frac{\begin{matrix}[x:\theta]\\ \vdots \\ M : \vec{\theta} \rightarrow \beta\end{matrix}}{\lambda x\colon\theta.\, M : \theta, \vec{\theta} \rightarrow \beta} \qquad \mathbf{skip} : \mathbf{comm} \qquad \frac{M : \theta, \vec{\theta} \rightarrow \beta \quad N : \theta}{(M\ N) : \vec{\theta} \rightarrow \beta}$$

$$\frac{V : \mathbf{var}[\delta]}{\mathbf{deref}\, V : \mathbf{exp}[\delta]} \qquad \frac{V : \mathbf{var}[\delta] \quad E : \mathbf{exp}[\delta]}{V := E : \mathbf{comm}}$$

$$\frac{A : \mathbf{exp}[\delta] \rightarrow \mathbf{comm} \quad E : \mathbf{exp}[\delta]}{\langle A, E\rangle : \mathbf{var}[\delta]} \qquad \frac{B : \mathbf{exp}[\mathbf{bool}] \quad M : \theta \quad N : \theta}{\mathbf{if}\, B\, \mathbf{then}\, M\, \mathbf{else}\, N : \theta}$$

$$\frac{C_1 : \mathbf{comm} \quad C_2 : \mathbf{comm}}{C_1 \,;\, C_2 : \mathbf{comm}} \qquad \frac{C : \mathbf{comm} \quad E : \mathbf{exp}[\delta]}{\mathbf{do}_\delta\, C\, \mathbf{result}\, E : \mathbf{exp}[\delta]}$$

$$\mathbf{new}_\delta \ : (\mathbf{var}[\delta] \rightarrow \mathbf{comm}) \rightarrow \mathbf{comm}$$

Table 1: Typing Rules

The meaning function associated with a judgement $\pi \vdash P : \vec{\theta} \rightarrow \beta$ will be a family of functions $[\![P]\!]_{\pi(\vec{\theta} \rightarrow \beta)}W\colon [\![\pi]\!]W \rightarrow [\![\vec{\theta}]\!]W \rightarrow [\![\beta]\!]W$ indexed by store shapes W. In the case of base types β, we will simply omit the $\vec{\theta}$ argument and have $[\![P]\!]_{\pi\beta}W\colon [\![\pi]\!]W \rightarrow [\![\beta]\!]W$. (The functionality of these valuations derives from categorical considerations on the model.)

We begin with identifiers. If $\pi \vdash x : \beta$ then the valuation is, as usual: $[\![x]\!]Wu = u(x)$. In the case of function types $\pi \vdash x : \vec{\theta} \rightarrow \beta$, given $\vec{d} \in [\![\vec{\theta}]\!]W$ we must produce $[\![x]\!]Wu\,\vec{d} \in [\![\beta]\!]W$. We can apply the meaning of x at the store shape $\mathbf{1}$ to obtain a function

$$u(x)[\mathbf{1}]\colon [\![\theta]\!](W \times \mathbf{1}) \rightarrow [\![\beta]\!](W \times \mathbf{1}),$$

and then we can apply unity isomorphisms to get a function $[\![\theta]\!](W) \rightarrow [\![\beta]\!](W)$. So we define $[\![x]\!]Wu = \mathit{unl}\,;\, u(x)[\mathbf{1}]\,;\, \mathit{unr}$.

Readers familiar with functor categories will notice that this valuation for identifiers is similar to what one obtains by uncurrying a projection $A \times (B \Rightarrow C) \longrightarrow (B \Rightarrow C)$, where $B \Rightarrow C$ is the functor exponent. In general, all of the valuations in the category-free semantics are obtained by uncurrying maps in the more standard category-theoretic presentation.

Most of the valuations for the language are in Table 2. In each equation u is an environment in $[\![\pi]\!]W$ for the appropriate π and a store shape W and $\vec{d} \in [\![\vec{\theta}]\!]$ is an appropriately typed vector of arguments. It is understood that this vector is omitted when the term in question is of base type. In the equation for **if**, t is either **comm** or of the form $\mathbf{exp}[\delta]$. The conditional extends to other types in the usual inductive fashion. The rules for abstraction and application are for the uncurried form of types.

$$
\begin{array}{lcl}
\llbracket \lambda x{:}\,\theta.\, M \rrbracket\, W\, u\, d, \vec{d} & = & \llbracket M \rrbracket W\, (u \mid x \mapsto d)\, \vec{d} \\[1ex]
\llbracket \mathbf{deref}\, V \rrbracket W\, u & = & \mathrm{snd}(\llbracket V \rrbracket W\, u) \\[1ex]
\llbracket V := E \rrbracket W\, u\, w & = & (\mathrm{fst}(\llbracket V \rrbracket W\, u)\ (\llbracket E \rrbracket W\, u\, w))\ w \\[1ex]
\llbracket \mathbf{skip} \rrbracket W\, u\, s & = & s \\[1ex]
\llbracket \langle A, E \rangle \rrbracket W\, u\, s & = & \langle (\lambda v\, \lambda w.\, \llbracket A \rrbracket W u (\lambda s.\, v) w\,,\ \llbracket E \rrbracket W u \rangle \\[1ex]
\llbracket C_1; C_2 \rrbracket W\, u\, s & = & \llbracket C_2 \rrbracket W\, u\, (\llbracket C_1 \rrbracket W\, u\, s) \\[1ex]
\llbracket \mathbf{do}_\delta\, C\, \mathbf{result}\, E \rrbracket W\, u\, s & = & \llbracket E \rrbracket W\, u\, (\llbracket C \rrbracket W\, u\, s) \\[1ex]
\llbracket \mathbf{if}_t\, B\, \mathbf{then}\, M\, \mathbf{else}\, N \rrbracket W\, u\, s & = & \begin{cases} \llbracket M \rrbracket W\, u\, s, & \text{if } \llbracket B \rrbracket W\, u\, s \,=\, \text{true} \\ \llbracket N \rrbracket W\, u\, s, & \text{if } \llbracket B \rrbracket W\, u\, s \,=\, \text{false} \end{cases}
\end{array}
$$

Table 2: Valuations

We will not give denotations, or syntax rules, for ordinary arithmetic and logical operations. These can be defined by lifting a function $f{:}\,\llbracket \delta_1 \rrbracket \times \cdots \times \llbracket \delta_n \rrbracket \longrightarrow \llbracket \delta \rrbracket$ to an interpretation for a combinator of type $\mathbf{exp}[\delta_1] \times \cdots \times \mathbf{exp}[\delta_n] \longrightarrow \mathbf{exp}[\delta]$ in the evident fashion.

The block expression **do** $\cdots$ **result** $\cdots$ warrants some explanation; **do** C **result** E returns the value of expression E in the state that results after executing C. For example, in

$$x := 2;\ y := (\mathbf{do}\, x := 1\, \mathbf{result}\ \mathbf{deref}\ x)$$

the final value of y is 1 whereas the final value of x is 2. Reynolds calls this "snapback semantics," because the state change caused by $x := 1$ is temporary: the state snaps back to its initial value on termination of the expression evaluation. There is a problem with snapback semantics: it violates what is often called the "single-threaded" nature of state in imperative languages (Schmidt, 1985). Intuitively, if a state change occurs, the old state is no longer available, so there is no way to backtrack to an earlier state. We will discuss this issue further in the Conclusion.

We now turn our attention to the key cases of **new** and application.

For store shape W, $p \in \llbracket \mathbf{var}[\delta] \longrightarrow \mathbf{comm} \rrbracket W$ and state $w \in W$,

$$\llbracket \mathbf{new}_\delta \rrbracket W\, u\, p\, w \;=\; \mathrm{fst}(p[\llbracket \delta \rrbracket]\ \langle a, e \rangle\ \langle w, \bar{\delta} \rangle)$$

where $\bar{\delta} \in \llbracket \delta \rrbracket$ is a standard initial value of new variables of type δ, and $\langle a, e \rangle \in \llbracket \mathbf{var} \rrbracket (W \times \llbracket \delta \rrbracket)$ is the new variable, defined as follows: $e \langle w, x \rangle = x$ and $a(y) \langle w, x \rangle \;=\; \langle w, y \rangle$. The "acceptor" a overwrites the $\llbracket \delta \rrbracket$-valued component of the state. The intuition behind this definition is that procedure p is executed in an expanded store shape, where the additional $[\![\delta]\!]$-valued component holds the value of the new variable. The argument $\langle a, e\rangle$ provides the capability for updating and accessing this variable. The final value of the variable is discarded using the projection fst. This is as in (Reynolds, 1981b; Oles, 1982).

The semantics of **new** is where the parametricity constraints in the model come into play. Because of the definition of procedure types, a call to p at an expanded store shape $W \times [\![\delta]\!]$ is required to satisfy uniformity conditions induced by relations involving $[\![\delta]\!]$. In the next section we will consider a number of examples showing these parametricity conditions at work.

Next, we consider application. Suppose that we are given $\pi \vdash M : \theta, \vec{\theta} \longrightarrow \beta$ and $\pi \vdash N : \theta$. If $\theta = \beta'$ is a primitive type then the semantics is simple, obtained by prepending the meaning of N onto a suitable vector.

$$[\![M\,N]\!]W\,u\,\vec{d} \;=\; [\![M]\!]W\,u\,([\![N]\!]Wu, \vec{d}\,)$$

It is clear that when $\vec{\theta}$ is empty this is the obvious application.

The case when θ is not a primitive type is more complex. If $\theta = \vec{\theta'} \longrightarrow \beta'$ then we need to prepend an element of $[\![\vec{\theta'} \longrightarrow \beta']\!]W$ onto a vector. Recall that a meaning of this type is a family of functions indexed by store shapes: we need to obtain such a family from the meaning of N.

For a fixed environment $u \in [\![\pi]\!]W$, define g as follows; for all X,

$$g[X] \;=\; [\![N]\!](W \times X)(\mathit{expand}_{\pi}(W, X)\,u)$$

where expansion maps are extended to type assignments pointwise:

$$\mathit{expand}_{\pi}(W, X)(u)(x \in \mathrm{dom}\pi) \;=\; \mathit{expand}_{\pi(x)} u(x).$$

Notice that $g[X]\colon [\![\vec{\theta'}]\!](W \times X) \longrightarrow [\![\beta']\!](W \times X)$, so g is certainly of the right form to be in $[\![\vec{\theta'} \longrightarrow \beta']\!]W$. It will be shown to satisfy the necessary parametricity constraints in the course of proving the Abstraction Theorem below. The semantics of application is

$$[\![M\,N]\!]W\,u\,\vec{d} \;=\; [\![M]\!]W\,u\,g, \vec{d}$$

The reader familiar with semantics in functor categories will notice that expansions come into this uncurried style of presentation in the case of application, whereas they appear when treating λ-abstraction when the semantics is presented in a more conventional curried form.

Theorem 5 (Abstraction)

Suppose $\pi \vdash P : \vec{\theta} \longrightarrow \beta$ and $R\colon W_0 \leftrightarrow W_1$ is a relation between store shapes; if $u_0\,[[\![\pi]\!]R]\,u_1$ and $\vec{d}_0\,[[\![\vec{\theta}]\!]R]\,\vec{d}_1$ then

$$[\![P]\!]W_0\,u_0\,\vec{d}_0 \left[[\![\beta]\!]R\right] [\![P]\!]W_1\,u_1\,\vec{d}_1$$

(This statement applies to terms of primitive type by omitting various vectors.)

Proof: The Abstraction Theorem and the well-definedness of $[\![P]\!]$ are proven simultaneously by structural induction on P. Well-definedness is immediate in all cases except application (which is the only case where the simultaneity is used in a non-trivial way).

For the well-definedness of application, suppose $\pi \vdash M : \theta, \vec{\theta} \longrightarrow \beta$ and $\pi \vdash N : \theta$. If $\theta = \beta'$ is a primitive type then the result is immediate, so suppose $\theta = \vec{\theta'} \longrightarrow \beta'$. Well-definedness will be assured if we can show that the family of functions

$$g[-] \colon [\![\vec{\theta'}]\!](W \times -) \longrightarrow [\![\beta']\!](W \times -)$$

satisfies the parametricity condition for $[\![\vec{\theta'} \longrightarrow \beta']\!]W$. For $S \colon X_0 \leftrightarrow X_1$ and $u \in [\![\pi]\!]W$ define $u_i = \mathit{expand}_\pi(W, X_i)\, u$. By the Identity Extension Lemma and the Expansion Parametricity Lemma, $u_0\,[[\![\pi]\!](\Delta_W \times S)]\,u_1$. (The evident version of the expansion lemma for type assignments is a corollary of the one for types.) By the Abstraction Theorem for N (induction hypothesis),

$$[\![N]\!](W \times X_0)\, u_0 \Big[[\![\vec{\theta'}]\!](\Delta_W \times S) \longrightarrow [\![\beta']\!](\Delta_W \times S)\Big] [\![N]\!](W \times X_1)\, u_1$$

and so

$$g[X_0]\Big[[\![\vec{\theta'}]\!](\Delta_W \times S) \longrightarrow [\![\beta']\!](\Delta_W \times S)\Big] g[X_1]\ .$$

This shows that $g \in [\![\vec{\theta'} \longrightarrow \beta']\!]W$, as desired.

For the Abstraction Theorem, we will consider application and **new**; all other cases are routine.

For application we have $\pi \vdash M : \theta, \vec{\theta} \longrightarrow \beta$ and $\pi \vdash N : \theta$. Suppose $u_0\,[[\![\pi]\!]R]\,u_1$ and $\vec{d_0}\,[[\![\vec{\theta}]\!]R]\,\vec{d_1}$. If $\theta = \beta'$ is primitive then the Abstraction Theorem for N (induction hypothesis) guarantees that $[\![N]\!]W_0\, u_0 [[\![\beta']\!]R]\, [\![N]\!]W_1\, u_1$ and then the Abstraction Theorem for M implies that

$$[\![M]\!]W_0\, u_0\, ([\![N]\!]W_0 u_0), \vec{d_0} \Big[[\![\beta]\!]R\Big] [\![M]\!]W_1\, u_1\, ([\![N]\!]W_1 u_1), \vec{d_1}.$$

In the case that $\theta = \vec{\theta'} \longrightarrow \beta'$ we reason in the same manner, but use $g_0[[\![\vec{\theta'} \longrightarrow \beta']\!]R]\,g_1$, where g_i is the meaning determined by the environment u_i, as in the definition of application. This last property follows from the Abstraction Theorem for N, with a proof similar to the well-definedness of g above using R in place of Δ_W.

For **new**, suppose $p_0\,[[\![\mathbf{var}[\delta] \longrightarrow \mathbf{comm}]\!]R]\,p_1$ and $w_0[R]w_1$. We must show that

$$\mathrm{fst}(p_0\,[[\![\delta]\!]]\, \langle a_0, e_0\rangle\, \langle w_0, \bar{\delta}\rangle) \Big[R\Big] \mathrm{fst}(p_1\,[[\![\delta]\!]]\, \langle a_1, e_1\rangle\, \langle w_1, \bar{\delta}\rangle)\ .$$

The key property is

$$\langle a_0, e_0\rangle \Big[[\![\mathbf{var}[\delta]]\!](R \times \Delta_{[\![\delta]\!]})\Big] \langle a_1, e_1\rangle$$

for the new variables $\langle a_0, e_0\rangle \in [\![\mathbf{var}[\delta]]\!](W_0 \times [\![\delta]\!])$ and $\langle a_1, e_1\rangle \in [\![\mathbf{var}[\delta]]\!](W_1 \times [\![\delta]\!])$. This is straightforward to verify. The assumption that p_0 and p_1 are related then implies

$$p_0\,[[\![\delta]\!]]\, \langle a_0, e_0\rangle\, \langle w_0, v\rangle \Big[R \times \Delta_{[\![\delta]\!]}\Big] p_1\,[[\![\delta]\!]]\, \langle a_1, e_1\rangle\, \langle w_1, v\rangle$$

for any $v \in [\![\delta]\!]$, and this ensures that the first components of the $p_i\,[[\![\delta]\!]]\, \langle a_i, e_i\rangle\, \langle w_i, v\rangle$ are R-related. ■

5 Examples of Reasoning

In each of the examples that follow, an unknown non-local procedure is passed a limited capability for accessing a local variable, in much the same way that an abstract type gives to its "clients" a limited capability for accessing its representation. The reasoning method employed involves choosing a relation that is satisfied by different arguments to the procedure, and then applying the parametricity property to infer a relational property that pairs of procedure calls must satisfy.

For the sake of readability, we continue to use sugared notation for code in the examples. The desugarings into the language of the previous sections should be clear.

We begin by describing a class of relations that can be used in several examples. Suppose W is any store shape 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

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

Consider any $c \in [\![\mathbf{comm}]\!](W \times Z)$ 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, if $p \in [\![\mathbf{comm} \to \mathbf{comm}]\!]W$, parametricity implies that

$$(*) \qquad p^*(\mathit{skip})\Big[[\![\mathbf{comm}]\!]R_E\Big]p[Z](c)$$

where $p^* = \mathit{unl}\,;p[\mathbf{1}]\,;\mathit{unr} : [\![\mathbf{comm}]\!]W \to [\![\mathbf{comm}]\!]W$. Hence, $p^*(\mathit{skip})$ is the semantics of an isolated procedure call $P(\mathbf{skip})$. 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 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 $\mathbf{skip}$; more precisely, if we define $\mathit{inc} \in [\![\mathbf{comm}]\!](W \times Z)$ by

$$\mathit{inc}\langle w, z\rangle \ = \ \langle w, z+1\rangle,$$

then

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

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

$$p^*(\mathit{skip})w\Big[(R_Z)_\perp\Big]p[Z](\mathit{inc})\langle w, z\rangle\ .$$

This means that the first component of $p[Z](\mathit{inc})\langle w, z\rangle$ is equal to $p^*(\mathit{skip})w$. Clearly, then, the semantics of variable declarations ensures the first equivalence considered in the Introduction:

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

It is important here that $w = w'$ when $w[R_E]\langle w', z\rangle$: the parametricity property always acts as the identity relation on non-local variables. This is where the identity extension lemma and the use of identities in the parametricity constraint on procedure types come into play.

We would like to emphasize that the reasoning method in this example is simply an instance of reasoning about polymorphic functions using Reynolds parametricity ((Reynolds, 1983); see also (Wadler, 1989) for numerous examples of this form of reasoning). The equivalence reduces to the following property

$$\mathrm{fst}(p[\mathbf{1}](\lambda x.\, x)) \quad = \quad \mathrm{fst}(p[Z](\lambda y.\, \langle \mathrm{fst}\, y, (\mathrm{snd}\, y) + 1\rangle))$$

for a polymorphic function $p{:}\,\forall\gamma.\, (\alpha \times \gamma \to \alpha \times \gamma) \to (\alpha \times \gamma \to \alpha \times \gamma)$. This is what we mean when we say that reasoning about local variables often amounts to proving properties of polymorphic functions. Of course, it *is* fairly significant that the polymorphism that we are concerned with is predicative in nature; but the point remains that the reasoning method we employ is *exactly* as in (Reynolds, 1983; Wadler, 1989). These methods will be seen below to lead to remarkably straightforward validations of previously troublesome equivalences.

Before continuing, it is worth pausing to explain why typical counterexamples to this equivalence, which exist in previous models, are not present here. Let $W = \{\mathit{true}, \mathit{false}\}$. One counterexample is essentially a family of functions

$$p[X]{:}\, [\![\mathbf{comm}]\!](W \times X) \longrightarrow [\![\mathbf{comm}]\!](W \times X)$$

such that

$$p[X](c)\langle b, x\rangle \;=\; \begin{cases} \langle \neg b, x\rangle, & \text{if } c\langle b, x\rangle \neq \langle b, x\rangle \\ \langle b, x\rangle, & \text{otherwise} \end{cases}$$

Such a p would break the equivalence, because the left-hand block would negate the state (which consists of a single boolean), whereas $P(\mathbf{skip})$ would leave the state unchanged. However, this p fails to satisfy the parametricity condition, for though $\mathit{skip}\,[[\![\mathbf{comm}]\!]R_Z]\,\mathit{inc}$ and $w[R]\langle w, z\rangle$, it is not the case that

$$p^*(\mathit{skip})(b)\left[R_Z\right]p[Z](\mathit{inc})\langle b, z\rangle\ ,$$

as $p^*(\mathit{skip})b$ is b, while $p[Z](\mathit{inc})\langle w, z\rangle$ is $\langle \neg b, x\rangle$. The equality test on states is the culprit in the definition of p: any two states $\langle w, z\rangle$ and $\langle w, z'\rangle$ are "indistinguishable" from the point of view of the second domain of the relation R_Z, so branching on the equality test violates parametricity.

Our second example demonstrates that the invariant-preserving properties of the models described in (Meyer and Sieber, 1988; O'Hearn and Tennent, 1993b) are encompassed by parametricity. If $Z^{\oplus}$ is the set of *nonnegative* integers, we again get $\mathit{skip}\,[[\![\mathbf{comm}]\!]R_{Z^{\oplus}}]\,\mathit{inc}$. The property $(*)$ now ensures that z is non-negative when $p[Z](\mathit{inc})\langle w, 0\rangle = \langle w', z\rangle$. This can be used to verify that the value of local variable z is still nonnegative on termination of the procedure call in

```
begin
  integer z;
  z := 0;
  P(z := z + 1);
  ...
end
```

Our last example using relations of the form R_E is

```
begin
  integer z;
  z := 0;            ≡       P(0)
  P(z)
end
```

where P: **exp**[**int**] → **comm**; we have left the de-referencing coercion (**deref**) from **var**[**int**] to **exp**[**int**] implicit 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 $R_{\{0\}}$. 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 $W \times Z \to Z$. These denotations are then related by $[\![\mathbf{exp}]\!]R_{\{0\}}$; i.e.,

$$\begin{tikzcd}
W \arrow[r, "0"] \arrow[d, leftrightarrow, "R_{\{0\}}"'] & Z \arrow[d, leftrightarrow, "\Delta_Z"] \\
W \times Z \arrow[r, "z"'] & Z
\end{tikzcd}$$

because if argument states are related, the Z-valued component is always 0. We can then use the parametricity of P, as in the other examples, to conclude that (the denotations of) the calls $P(0)$ and $P(z)$ are $[\![\mathbf{comm}]\!]R_{\{0\}}$-related, and the equivalence follows from the valuation for variable declarations.

Next we consider a relation that does not fit into the R_E pattern: the relation $\Delta_W \times R$, where $R\colon Z \leftrightarrow Z$ is defined by

$$z_0[R]z_1 \iff -z_1 = z_0 \geq 0 \ .$$

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 of the procedures *inc* and *val* are directly related by $\Delta_W \times R$, and we can use the parametricity property of procedures to conclude that the calls to non-local procedure P are related. This implies the desired equivalence because the semantics of **new** disposes of the Z-valued component of $W \times Z$ on termination, and we are left with Δ_W-related results.

We should mention that this last equivalence is in fact valid in the models of (Oles, 1982; Tennent, 1990). These models can typically handle representation independence when the different representations being considered are isomorphic. Our final example shows how non-isomorphic representations can be dealt with.

The example involves a simple abstract "switch." A switch will have two associated operations.

flick: turns the switch on; and

on: a predicate that tests whether the switch has been flicked on.

The switch is initially off, but remains on after it has been flicked for the first time.

One representation of the switch will be the evident one using a boolean variable. In the other, 0 will correspond to the switch being off, and the on position will be represented by any positive integer. These representations are given in the following two blocks, where P is of type $(\mathbf{comm}, \mathbf{exp}[\mathbf{bool}]) \rightarrow \mathbf{comm}$.

```
begin                                        begin
  boolean z;                                   integer z;
  procedure flick; z := true;                  procedure flick; z := z + 1;
  boolean procedure on; on := z;      ≡        boolean procedure on; on := z ≥ 1;
  z := false;                                  z := 0;
  P(flick, on)                                 P(flick, on)
end                                          end
```

A typical counterexample, which exists (in one form or another) in the models of (Oles, 1982; Meyer and Sieber, 1988; O'Hearn and Tennent, 1992), is p such that

$$p[X]\,\langle c, e\rangle\,\langle b, x\rangle \;=\; \begin{cases} \langle \neg b, z\rangle, & \text{if } c(c\langle b, z\rangle) \;=\; c\langle b, z\rangle \\ \langle b, z\rangle, & \text{otherwise} \end{cases}$$

The equality test on states is once again the culprit.

This equivalence can be validated in our semantics using a relation of the form $\Delta_W \times R$, where $R\colon [\![\mathbf{bool}]\!] \leftrightarrow [\![\mathbf{int}]\!]$ is the least relation such that

$$\mathit{false}[R]0 \;\wedge\; (n \geq 1 \;\Rightarrow\; \mathit{true}[R]n)\ .$$

6 Algebraic Aspects of First-Order Types

A standard test for the parametricity of models of polymorphism involves connections between free algebras and the denotations of certain lower-order polymorphic types (Reynolds, 1983). For example, in a model that is "sufficiently parametric," the type $\forall\gamma.\,(\gamma \rightarrow \gamma) \rightarrow (\gamma \rightarrow \gamma)$ of Church numerals will (in the absence of recursion) in fact denote a natural numbers object, and the type $\forall\gamma.\,(\alpha \times \gamma \rightarrow \gamma) \times \gamma \rightarrow \gamma$ will be isomorphic to the type $\mathrm{list}[\alpha]$ of finite lists over α. These representations supply a very clear picture of low-order polymorphic types, and are an indication of the constraining effect of the parametricity conditions under consideration. Our purpose in this section is to describe how our parametric semantics yields similar representations of first-order Algol types.

To begin, we consider $[\![\mathbf{comm} \rightarrow \mathbf{comm}]\!]\mathbf{1}$. We can use an argument of Plotkin (1980) to characterize precisely the elements in this set. If $p \in [\![\mathbf{comm} \rightarrow \mathbf{comm}]\!]\mathbf{1}$ then there is a number n such that $p[N]\,(\mathrm{id}_{\{*\}} \times \mathit{succ})\,\langle *, 0\rangle \;=\; \langle *, n\rangle$, where N is the set of natural numbers and succ is the successor function. Then for any X, $c\colon X \rightarrow X$ and $x \in X$, we can set up a relation $R\colon N \leftrightarrow X$ where $0[R]x$ and $m[R]x' \;\Rightarrow\; m+1[R]c(x')$. The functions succ and c are then related by $R \rightarrow R$ and we can use parametricity to conclude that $p[N]\,(\mathrm{id}_{\{*\}} \times \mathit{succ})\,\langle *, 0\rangle$ and $p[X]\,(\mathrm{id}_{\{*\}} \times c)\,\langle *, x\rangle$ are R-related and, in particular, the latter is $\langle *, c^n(x)\rangle$, where $c^0 = \mathbf{skip}$ and $c^{n+1} = c\,;c^n$. Thus, p is the n-th Church numeral.

In an Algol-like language, the n-th Church numeral is defined by $\lambda c\colon \mathbf{comm}.\, c^n$. From this we can immediately see two interesting facts. First, every element of $[\![\mathbf{comm} \rightarrow \mathbf{comm}]\!]\mathbf{1}$ is definable by a closed term. Second, up to semantic equivalence, the local-variable declarator

new does not figure into closed terms of this type at all, for any closed term of this type will be equivalent to one that doesn't use **new**. One has to go up to closed terms of second-order type, or to open terms of first-order, for **new** to make a difference.

What we have done here is to follow the analogy between type variables and store shapes. $[\![\mathbf{comm} \rightarrow \mathbf{comm}]\!]\mathbf{1}$ corresponds to $\forall\gamma.\, (\mathbf{1}\times\gamma \rightarrow \mathbf{1}\times\gamma) \rightarrow (\mathbf{1}\times\gamma \rightarrow \mathbf{1}\times\gamma)$, and, as $\mathbf{1}\times\gamma \cong \gamma$, this should in turn be the Church numerals. The reader familiar with (Reynolds, 1983) will then be able to see how similar representations can be obtained for other first-order Algol types. We collect a few examples into the following proposition.

Proposition 6 (Reynolds)

We have the following isomorphisms, where α is a store shape.

$$\begin{array}{rcl} [\![\mathbf{exp}[\delta] \rightarrow \mathbf{exp}[\delta']]\!]\alpha & \cong & (\alpha \rightarrow [\![\delta]\!]) \rightarrow (\alpha \rightarrow [\![\delta']\!]) \\ [\![\mathbf{exp}[\delta] \rightarrow \mathbf{comm}]\!]\alpha & \cong & (\alpha \rightarrow [\![\delta]\!]) \rightarrow (\alpha \rightarrow \alpha) \\ [\![\mathbf{comm} \rightarrow \mathbf{exp}[\delta]]\!]\alpha & \cong & (\alpha \times \mathrm{list}[\alpha] \rightarrow \alpha) \rightarrow (\alpha \rightarrow [\![\delta]\!]) \\ [\![\mathbf{comm} \rightarrow \mathbf{comm}]\!]\alpha & \cong & (\alpha \times \mathrm{list}[\alpha] \rightarrow \alpha) \rightarrow (\alpha \rightarrow \alpha \times \mathrm{list}[\alpha]) \end{array}$$

Proof: These isomorphisms are based on observations in (Reynolds, 1983). We will outline the proof of the last isomorphism to indicate that these arguments do go through for our semantics of Algol types.

Any $c\colon \alpha \times Z \rightarrow \alpha \times Z$, for some store shape Z, can be decomposed into two functions $c_1\colon \alpha \times Z \rightarrow \alpha$ and $c_2\colon \alpha \times Z \rightarrow Z$. For a fixed initial state $\langle s, z\rangle \in \alpha \times Z$, let $R\colon Z \leftrightarrow \mathrm{list}[\alpha]$ be the smallest relation such that

$$z[R]\epsilon \qquad z'[R]\ell \Rightarrow c_2\langle a, z'\rangle[R]\mathrm{cons}\langle a, \ell\rangle$$

where ϵ is the empty list. One can then define a suitable c_1^* such that

$$c_1[\Delta_\alpha \times R \rightarrow \Delta_\alpha]c_1^*$$

and we have

$$\{c_1, c_2\}\,[[\![\mathbf{comm}]\!]\Delta_\alpha \times R]\,\{c_1^*, \mathrm{cons}\}$$

where here we are using $\{f, g\} : A \rightarrow B \times C$ to denote the tupling function obtained from $f : A \rightarrow B$ and $g : A \rightarrow C$. Then if $p \in [\![\mathbf{comm} \rightarrow \mathbf{comm}]\!]\alpha$ we get that

$$p[Z]\, c\,\langle s, z\rangle \Big[\Delta_\alpha \times R\Big] p\,[\mathrm{list}[\alpha]]\,\{c_1^*, \mathrm{cons}\}\,\langle s, \epsilon\rangle \ ,$$

and so p is completely determined by the action of the function $p[\mathrm{list}[\alpha]]$. Furthermore, the arguments to this function can be taken to be of the form $\{f, \mathrm{cons}\}$ and $\langle s, \epsilon\rangle$ so, as cons and ϵ are fixed, this is determined by a function of type $(\alpha \times \mathrm{list}[\alpha] \rightarrow \alpha) \rightarrow (\alpha \rightarrow \alpha \times \mathit{list}[\alpha])$. Conversely, it is easy, using this R, to see how any function of this type determines an element of $[\![\mathbf{comm} \rightarrow \mathbf{comm}]\!]\alpha$. ■

Notice that function types with $\mathbf{exp}[\delta]$ in a contravariant position are represented in a pointwise fashion. A meaning at such a type can be applied at a "later stage," after local variables have been added to the stack, but such a function is completely determined by its behaviour at the "present stage." The reason is that expressions may read from, but not write to, local variables. If we pass an argument $e \in [\![\mathbf{exp}[\delta]]\!](\alpha \times \gamma)$ and evaluate the resulting function call in state $\langle s, n\rangle$, then parametricity can be used to show that this is equivalent to passing the evident corresponding expression $e' \in [\![\mathbf{exp}[\delta]]\!]\alpha \times \{n\}$. The pointwise exponentiation arises because $\alpha \times \{n\} \cong \alpha$. (This principle was at work in the example from the previous section involving $P(0)$ and $P(z)$). Of course, not all elements of these types will be definable; for example, definability of all elements of **exp**[**int**] $\rightarrow$ **exp**[**int**] is not possible for computability reasons.

For the types with **comm** in a contravariant position, changes to a local variable by a command argument are mirrored by cons: a list of α's records non-local states when a command argument is executed. The representation of **comm** $\rightarrow$ **exp**$[\delta]$ illustrates the non-single-threaded nature of the semantics. In a semantics that captured single-threading properly we expect that the occurrences of list$[\alpha]$ would disappear, because single-threading should mandate that commands cannot be executed within expressions.

These representations are limited to first-order types: we do not know of characterizations of level-two types such as, for example, (**comm** $\rightarrow$ **comm**) $\rightarrow$ **comm**. A similar phenomenon occurs in models of polymorphism: much is known about level-two polymorphic types, but considerably less for level three. (Here our understanding breaks down at level two because these types correspond semantically to level-three functional types.)

The situation in the presence of recursion is more complex due to lifting, and we do not have a clear general picture, given by a clean scheme like the one in (Reynolds, 1983), of the denotations of all first-order types in the presence of recursion. Characterizations of certain specific types have been obtained, however; we illustrate with $[\![\mathbf{comm} \rightarrow \mathbf{comm}]\!]\mathbf{1}$.

Let *Vnat* be the vertical natural numbers, i.e., the natural numbers with the usual "less than" order, and with an extra top element ∞. Vnat^{op} is the vertical naturals with the ordering reversed. Then

$$[\![\mathbf{comm} \rightarrow \mathbf{comm}]\!]\mathbf{1} \quad \cong \quad N_\bot \otimes \mathit{Vnat}^{op}$$

where $\otimes$ is the smash product.

An outline of the proof of this isomorphism is as follows. Using the isomorphism $\mathbf{1} \times \alpha \cong \alpha$, a meaning in $[\![\mathbf{comm} \rightarrow \mathbf{comm}]\!]\mathbf{1}$ will be a family of continuous functions

$$p[\alpha] \colon (\alpha \rightarrow \alpha_\bot) \rightarrow (\alpha \rightarrow \alpha_\bot)$$

satisfying the parametricity condition. If $p[N]\,\mathit{succ}\,0 \;=\; \bot$ then p corresponds to $\langle \bot, \infty\rangle$. If $p[N]\,\mathit{succ}\,0 \;=\; n$ then there will be a smallest m such that $p[N]\,(\mathit{succ}[n+m])\,0 \;=\; n$, where $(\mathit{succ}[k])\,a \;=\; a+1$ if $a \;<\; k$ and $\bot$ otherwise. In this case p corresponds to $\langle n, m\rangle$. The desired isomorphism can then be shown using parametricity with an argument similar to the one used by Plotkin for the Church numerals: we define an appropriate relation that relates an argument in $\alpha \rightarrow \alpha_\bot$ to a $\mathit{succ}[k]$.

As before, every element of this domain can be defined by a closed term, with the appropriate boolean tests and a term **diverge**: **comm** that denotes the constantly-$\bot$ function. Specifically, $\langle \bot, \infty\rangle$ is defined by **diverge**, and $\langle n, m\rangle$ is defined by

$$\lambda c \colon \mathbf{comm}.\, \mathbf{if}\, (\mathbf{do}\; c^{n+m}\; \mathbf{result}\; 1) = 1\; \mathbf{then}\; c^n\; \mathbf{else\, skip}$$

The test $(\mathbf{do}\, c^{n+m}\, \mathbf{result}\ 1) = 1$ will converge, and return true, iff c^{n+m} converges. The **skip** branch never gets executed.

It is now possible to appreciate the role of $Vnat^{op}$. It concerns "lookahead," in the sense that we look to see if $n+m$ executions of c will converge and, if so, we execute c n times. This illustrates how a semantics that properly captures single threading could perhaps lead to simpler representations. For example, the closed terms of type $\mathbf{comm} \rightarrow \mathbf{comm}$ definable *without* $\mathbf{do} - \mathbf{result} -$ are, semantically, in correspondence with $N_\perp$, which is considerably simpler than $N_\perp \otimes Vnat^{op}$.

7 Relations and Reflexive Graphs

The category-free presentation, though quite elementary, is also rather *ad hoc* in some respects. In the next few sections we will endeavour to place the model into a categorical context, providing some justification for the definitions.

A first attempt would be to say that the model lives in a category of "relators" (Mitchell and Scedrov, 1993; Ma and Reynolds, 1992; Abramsky and Jensen, 1991). The objects map store shapes and relations between them to sets and relations between them in a way that preserves identity relations, and the morphisms are families of functions, indexed by store shapes, satisfying a parametricity constraint. While it is true that each type in our model determines a relator, the relator viewpoint is not quite satisfactory. The appropriate notion of exponentiation for relators is pointwise: $(A \rightarrow B)(X) = A(X) \rightarrow B(X)$ for X a store shape or relation. A better categorical explanation of the model would connect our interpretation of procedure types with exponentiation, and our interpretation is not pointwise.

This is the point at which we must bring out the functor-category structure, which shows up in the category-free presentation in the use of expansion functions. It will be seen that each type determines a functor from the category of store shapes (from (Oles, 1982)) to the category of sets. The interpretation of procedure types then has some of the flavour of a functor-category exponential, but with additional parametricity constraints. A suitable category will be obtained by considering *both* the relator and functor aspects of types, along with naturality and parametricity conditions on morphisms.

The reader might have noticed that naturality properties were never used in proving any of our results in previous sections, or in reasoning about example equivalences. The place where naturality does come in is in trying to prove the validity of the laws of the typed λ-calculus. It would have to be accounted for if we were to validate these laws directly in the category-free semantics; in the categorical semantics it will be crucial to get a Cartesian closed category.

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 of a *reflexive graph*. 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, familiar to category theorists, where *categories* of vertices and edges are allowed (Barr and Wells, 1990; Lawvere, 1989; Johnstone, 1989); 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 (Ma and Reynolds, 1992; Mitchell and Scedrov, 1993).

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 (Mac Lane, 1971), 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 $R\colon W_0 \leftrightarrow W_1$ and co-domain $S\colon X_0 \leftrightarrow X_1$ is a quadruple (R, f_0, f_1, S) such that $f_0[R \rightarrow S]f_1$. 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 $R\colon W_0 \leftrightarrow W_1$ 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 $\mathcal{S}_{\delta_i}\colon \mathcal{S}_e \longrightarrow \mathcal{S}_v$ for $i = 0, 1$ are defined by $\mathcal{S}_{\delta_i}(R\colon W_0 \leftrightarrow W_1) = W_i$ 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 $\mathcal{S}_I\colon \mathcal{S}_v \longrightarrow \mathcal{S}_e$ is such that $\mathcal{S}_I(W) = \Delta_W$ and $\mathcal{S}_I(f\colon W \longrightarrow X) =$

$$\begin{tikzcd} W \arrow[r, "f"] \arrow[d, leftrightarrow, "\Delta_W"'] & X \arrow[d, leftrightarrow, "\Delta_X"] \\ W \arrow[r, "f"'] & X \end{tikzcd}$$

Category $\mathcal{S}_e$ is the category of relations over sets presented in (Ma and Reynolds, 1992). 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 (Ma and Reynolds, 1992) 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. Composites and identities are evident.
- The functors $\mathcal{D}_{\delta_i}\colon \mathcal{D}_e \longrightarrow \mathcal{D}_v$ for $i = 0, 1$ and $\mathcal{D}_I\colon \mathcal{D}_v \longrightarrow \mathcal{D}_e$ 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 (Oles, 1982) as its vertex category $\mathcal{W}_v$. The category Σ is as follows.

- The objects are (certain) sets, including desired data types, such as $\{\mathit{true}, \mathit{false}\}$ and $\{-2, -1, 0, 1, 2, \ldots\}$, 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 $x \in X$,
  $$\rho\langle\phi(x), x\rangle \;=\; x \;;$$
  - for all $x \in X$ and $w \in W$,
  $$\phi(\rho\langle w, x\rangle) \;=\; w \;;$$

  - for all $x \in X$ and $w, w' \in W$,

$$\rho\langle w, \rho\langle w', x\rangle\rangle \;=\; \rho\langle w, x\rangle \ .$$

For example, there is an "expansion" morphism $(\phi, \rho){:}\, W \longrightarrow X$ such that $X = W \times Y$ for some data type Y with $\phi\langle w, y\rangle = w$ and $\rho\langle \overline{w}, \langle w, y\rangle\rangle = \langle \overline{w}, y\rangle$; 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 $(\phi, \rho){:}\, W \longrightarrow X$ induces a set isomorphism $X \cong W \times Y$ for some non-empty set Y; that is, up to isomorphism, *every* morphism is an expansion.

- The composite of morphisms $(\phi, \rho){:}\, W \longrightarrow X$ and $(\phi', \rho'){:}\, X \longrightarrow Y$ is $(\phi'', \rho''){:}\, W \longrightarrow Y$ such that $\phi'' = \phi' \mathbin{;} \phi$ and $\rho''\langle w, y\rangle = \rho'\langle \rho\langle w, \phi'(y)\rangle, y\rangle$.
- 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 $\mathbf{\Sigma}$ can be defined as follows.

- The objects are relations $R{:}\, W \leftrightarrow X$, where W and X are $\mathbf{\Sigma}$-objects.
- A morphism with domain $R{:}\, W_0 \leftrightarrow W_1$ and co-domain $S{:}\, X_0 \leftrightarrow X_1$ is a quadruple $(R, (\phi_0, \rho_0), (\phi_1, \rho_1), S)$ such that $\phi_0[S \longrightarrow R]\phi_1$ and $\rho_0[R \times S \longrightarrow S]\rho_1$. 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 $\mathbf{\Sigma}$.

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" $\mathbf{\Sigma}$-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 $\mathcal{W}_e$ iff 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). Notice that $\mathcal{W}_e$ is not a category of relations over $\mathbf{\Sigma}$ in the sense of (Ma and Reynolds, 1992); in fact, $\mathbf{\Sigma}$ does not even have a terminal object.

8 Parametric Functors and Natural Transformations

Next we describe a category of "parametric" functors and natural transformations. The description will be highly tailored to the specific definitions of $\mathcal{W}$ and $\mathcal{S}$, but at the end of the section we sketch a more general setting for the definitions.

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

- a mapping F_0 from $\mathcal{W}_v$-objects to $\mathcal{S}_v$-objects;
- a mapping F_1 from $\mathcal{W}_v$-morphisms to $\mathcal{S}_v$-morphisms; and
- a mapping F_2 from $\mathcal{W}_e$-objects to $\mathcal{S}_e$-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 $\mathcal{S}_v$;
- $F_1(\mathrm{id}_w) = \mathrm{id}_{F_0(w)}$ 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 $\mathcal{S}_e$;
- $F_2(\Delta_w) = \Delta_{F_0(w)}$, 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 $\mathcal{S}_e$.

The first three conditions say that F_0 and F_1 constitute a conventional functor from $\mathcal{W}_v$ to $\mathcal{S}_v$; the next two conditions say that F_0 and F_2 constitute a "relator" (Mitchell and Scedrov, 1993; Abramsky and Jensen, 1991); and the last condition is a parametricity constraint. This last condition is closely related to the Expansion Parametricity Lemma and is crucial for function types to behave properly, e.g. for currying to satisfy relevant parametricity conditions. 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 $\mathcal{S}$, η is a *parametric natural transformation from F to G* if it maps $\mathcal{W}_v$-objects to $\mathcal{S}_v$-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 $\mathcal{S}_e$.

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 (like "vertical" composition of natural transformations). The category having all parametric functors from

$\mathcal{W}$ to $\mathcal{S}$ as objects and all parametric natural transformations of these as morphisms will be denoted $\mathcal{S}^{\mathcal{W}}$.

Theorem 7

$\mathcal{S}^{\mathcal{W}}$ 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. A terminal object $\mathbf{1}$ has $\mathbf{1}(X) = \{*\}$, $\mathbf{1}(\phi, \rho) = \mathrm{id}_{\{*\}}$ and $\mathbf{1}(R) = \Delta_{\{*\}}$

For exponentiation, we first define the analogue of "representable" functors (Mac Lane, 1971; Lambek and Scott, 1986). If X and Y are store shapes then $h^X(Y) = \mathit{Hom}_{\mathcal{W}_v}(X, Y)$, and for f and g maps in Σ, $h^f(g) = \mathit{Hom}_{\mathcal{W}_v}(f, g)$, so that $h^f g\,(h) \;=\; f; h; g$. If $R: X_0 \leftrightarrow X_1$ and $S: Y_0 \leftrightarrow Y_1$, then $h^R S: h^{X_0}Y_0 \longleftrightarrow h^{X_1}Y_1$ is such that

$$g[h^R S]f \qquad \text{iff} \qquad \begin{tikzcd} X_0 \arrow[r, "g"] \arrow[d, leftrightarrow, "R"'] & Y_0 \arrow[d, leftrightarrow, "S"] \\ X_1 \arrow[r, "f"'] & Y_1 \end{tikzcd} \text{ in } \mathcal{W}_e.$$

We write $h^X(-)$ for the parametric functor that sends Y to $h^X Y$, f to $h^{\mathrm{id}_X} f$ and R to $h^{\Delta_X}(R)$.

Exponentiation is then defined on store shapes as follows:

$$G^F\, X \;=\; \mathit{Hom}_{\mathcal{S}^{\mathcal{W}}}(h^X \times F, G)\ ;$$

on morphisms,

$$(G^F\, f\, p)\,[Z]\langle g, a\rangle \;=\; p\,[Z]\,\langle f\,; g, a\rangle\ ;$$

and on relations, $p[G^F R]q$ iff

$$\forall S: W_0 \leftrightarrow W_1.\, p[W_0]\Big[h^R S \times G(S) \rightarrow F(S)\Big]q[W_1]$$

It is not difficult to show that G^F satisfies the functor and relator requirements, and the condition that a parametric functor send related $\mathcal{W}_v$-morphisms to related $\mathcal{S}_v$-morphisms.

The application and currying maps are exactly as in presheaf categories. The application map $\mathit{app}: F \times G^F \mathrel{\dot{\rightarrow}} G$ is

$$\mathit{app}\,[W]\,\langle a, p\rangle \;=\; p[W]\langle \mathrm{id}_W, a\rangle.$$

Naturality follows as usual. To see that it is parametric, assume $p_0[(G^F)R]p_1$ and $a_0[F(R)]a_1$. As

$$\begin{tikzcd} W_0 \arrow[r, "\mathrm{id}_{W_0}"] \arrow[d, leftrightarrow, "R"'] & W_0 \arrow[d, leftrightarrow, "R"] \\ W_1 \arrow[r, "\mathrm{id}_{W_1}"'] & W_1 \end{tikzcd}$$

, we have that $\langle \mathrm{id}_{W_0}, a_0\rangle[h^R R \times F(R)]\langle \mathrm{id}_{W_1}, a_1\rangle$, and the definition of $G^F(R)$ implies $(p_0[W_0]\langle \mathrm{id}, a_0\rangle)\,[G(R)]\,(p_1[W_1]\langle \mathrm{id}, a_1\rangle)$. The Currying map

$$\mathit{curry}: \mathit{Hom}(F \times G, H) \longrightarrow \mathit{Hom}(F, H^G)$$

is

$$\mathit{curry}\, m\, W\, a\, \langle f, b\rangle \;=\; m\, X\, \langle F(f)a, b\rangle$$

where $f{:}\, W \longrightarrow X$ in $\mathcal{W}_v$. The naturality of $\mathit{curry}\, m$ is shown as usual, and parametricity is proved using the condition that F send related $\mathcal{W}_v$-morphisms to related $\mathcal{S}_v$-morphisms. That *curry* and *app* have the required properties of exponentiation is straightforward; this is where the naturality requirements are crucial. ■

We now show how to interpret types as parametric functors from $\mathcal{W}$ to $\mathcal{S}$. We use the angled brackets $\langle\!\langle\cdot\rangle\!\rangle$ to distinguish the parametric-functor semantics from $[\![\cdot]\!]$.

For expressions:

- for every $\mathcal{W}_v$-object W,
$$\langle\!\langle \mathbf{exp}[\delta]\rangle\!\rangle W \;=\; W \longrightarrow [\![\delta]\!]\ ,$$
- for every $\mathcal{W}_v$-morphism $(\phi, \rho){:}\, W \longrightarrow X$ and $e \in \langle\!\langle \mathbf{exp}[\delta]\rangle\!\rangle W$,
$$\langle\!\langle \mathbf{exp}[\delta]\rangle\!\rangle(\phi, \rho)\ e \;=\; \phi\, ;\, e\ ,$$
  and
- for every $R{:}\, W_0 \leftrightarrow W_1$,
$$\langle\!\langle \mathbf{exp}[\delta]\rangle\!\rangle R \;=\; R \longrightarrow \Delta_{[\![\delta]\!]}\ .$$

For commands:

- for every $\mathcal{W}_v$-object W,
$$\langle\!\langle \mathbf{comm}\rangle\!\rangle W \;=\; W \longrightarrow W\ ;$$
- for every $\mathcal{W}_v$-morphism $(\phi, \rho){:}\, W \longrightarrow X$, $x \in X$, and $c \in \langle\!\langle \mathbf{comm}\rangle\!\rangle W$,
$$\langle\!\langle \mathbf{comm}\rangle\!\rangle(\phi, \rho)\ c\ x = \;\rho\langle c(\phi(x)), x\rangle,$$
  and
- for every $R{:}\, W_0 \leftrightarrow W_1$,
$$\langle\!\langle \mathbf{comm}\rangle\!\rangle R \;=\; R \longrightarrow R\ .$$

For the morphism part what we do is execute c on the small part of the stack, i.e. $c(\phi(x))$, and then use ρ to replace the small part of x with the resulting final state.

The parametricity conditions on these functors 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

$$\langle\!\langle \mathbf{exp}[\delta]\rangle\!\rangle \;=\; S \longrightarrow \Box[\![\delta]\!]$$

$$\langle\!\langle \mathbf{comm}\rangle\!\rangle \;=\; S \longrightarrow S$$

using a *parametric* version of "contra-exponentiation" (O'Hearn and Tennent, 1992), where $\Box D$ is the constant functor whose object, morphism, and relation parts always yield D, id_D, and Δ_D, respectively. This is an indication of the effectiveness of the parametricity constraints.

For storage variables:

$$\begin{aligned}
\langle\!\langle \mathbf{var}[\delta] \rangle\!\rangle X &= (\llbracket \delta \rrbracket \to \langle\!\langle \mathbf{comm} \rangle\!\rangle X) \times \langle\!\langle \mathbf{exp} \rangle\!\rangle X \\
\langle\!\langle \mathbf{var}[\delta] \rangle\!\rangle (\phi, \rho) &= (\mathrm{id}_{\llbracket \delta \rrbracket} \to \langle\!\langle \mathbf{comm} \rangle\!\rangle (\phi, \rho)) \times \langle\!\langle \mathbf{exp} \rangle\!\rangle (\delta, \rho) \\
\langle\!\langle \mathbf{var}[\delta] \rangle\!\rangle R &= (\Delta_{\llbracket \delta \rrbracket} \to \langle\!\langle \mathbf{comm} \rangle\!\rangle R) \times \langle\!\langle \mathbf{exp} \rangle\!\rangle R \ .
\end{aligned}$$

For procedures we use exponentiation in $\mathcal{S}^{\mathcal{W}}$:

$$\langle\!\langle \vec{\theta} \to \beta \rangle\!\rangle \;=\; \langle\!\langle \beta \rangle\!\rangle^{\langle\!\langle \vec{\theta} \rangle\!\rangle} \ .$$

Here, $\langle\!\langle \vec{\theta} \rangle\!\rangle$ is the product of $\langle\!\langle \theta_i \rangle\!\rangle$ for the components of the vector. Of course, as $\mathcal{S}^{\mathcal{W}}$ is a ccc we could ignore vectors and interpret procedure types in a curried syntax: $\langle\!\langle \theta \to \theta' \rangle\!\rangle = \langle\!\langle \theta' \rangle\!\rangle^{\langle\!\langle \theta \rangle\!\rangle}$.

The interpretations of terms can be given exactly as in (Oles, 1982). We have already seen the application and currying maps in the proof of Proposition 7, and these are exactly as in functor categories. We will define

$$\mathit{new}\colon \langle\!\langle \mathbf{var} \to \mathbf{comm} \rangle\!\rangle \stackrel{.}{\to} \langle\!\langle \mathbf{comm} \rangle\!\rangle$$

to indicate how variable declarations are treated. For $\mathcal{W}_v$-object W, $p \in \langle\!\langle \mathbf{var} \to \mathbf{comm} \rangle\!\rangle W$ and $w \in W$,

$$\mathit{new}_\delta\; W\; p\; w \;=\; \mathrm{fst}\,(p[W \times \llbracket \delta \rrbracket]\,\langle f, \langle a, e\rangle\rangle \langle w, \bar{\delta}\rangle)$$

where $f\colon W \to W \times \langle\!\langle \delta \rangle\!\rangle$ is an "expansion" morphism in $\mathcal{W}_v$, $\bar{\delta} \in \langle\!\langle \delta \rangle\!\rangle$ is the standard initial value of new variables, and $\langle a, e\rangle \in \langle\!\langle \mathbf{var} \rangle\!\rangle (W \times Z)$ is the new variable, defined as follows: $a(z')\langle w, z\rangle = \langle w, z'\rangle$ and $e\langle w, z\rangle = z$.

We conclude this section by sketching a more general context for these definitions; it can be skipped without loss of continuity. A morphism $M\colon \mathcal{G} \to \mathcal{H}$ 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 component-wise.

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* (Kelly and Street, 1974). (More precisely, the category $\mathcal{S}^{\mathcal{W}}$ is equivalent to the category having natural transformations between the graphs $\mathcal{W}$ and $\mathcal{S}$ (viewed as functors) as objects and modifications as morphisms.) This gives some assurance of the appropriateness of the various conditions in the definition $\mathcal{S}^{\mathcal{W}}$, which uses simplifications that depend on specific structure in $\mathcal{W}$ and $\mathcal{S}$.

Another perspective on our model can be given in terms of *internal categories*. As is well known, reflexive graphs in **CAT** can be equivalently viewed as internal categories in a category of (large enough) reflexive graphs. Parametric functors then correspond to internal functors between internal categories, and parametric natural transformations to internal natural transformations. We gave the "reflexive graphs in **CAT**" presentation here because we felt that it might be (slightly) more accessible.

However, the internal category viewpoint perhaps shows more directly the connection to (Reynolds, 1981b; Oles, 1982): our semantics could be considered as essentially that of (Oles, 1982), but re-cast in a context where terms like "functor" must be understood as pertaining to categories that live in another category. This is the reason why the definitions of currying, application, **new**, etc., for (the categorical presentation of) our model are just like those given by Oles. Of course, the interest in our model derives more from the semantics of types than of valuations. The uniformity conditions arising from relational parametricity give us stronger reasoning principles than in a standard functor-category framework.

9 When Parametricity Implies Naturality

We now undertake to explain the connection between the category-free and parametric-functor presentations of our semantics, and also to uncover why an "uncurried" treatment of types is used in the category-free version.

First, we need a result from (Oles, 1982) about morphisms in the category $\mathcal{W}_v$ of store shapes.

Lemma 8 (Expansion Factorization (Oles))

Every $\mathcal{W}_v$-morphism $W \longrightarrow X$ can be factored into an expansion followed by an isomorphism:

$$W \xrightarrow{e} W \times Y \xrightarrow{i} X.$$

Recall that the Isomorphism Functoriality Lemma played an important role in the category-free semantics. A condition analogous to it was not needed in the definition of reflexive graph because of the following result which, it should be noted, applies to *any* parametric functor (and not just definable ones).

Lemma 9 (Isomorphism Correspondence)

For every parametric functor A, if $(\phi, \rho): W \longrightarrow X$ is an isomorphism then the function $A(\rho, \phi)$ and relation $A(R)$ have the same graph, where $R: W \leftrightarrow X$ is the relation having the same graph as ϕ^{-1}.

Proof: Let $f = (\phi, \rho)$. From the definition of related $\mathcal{W}_v$ morphisms we have

$$\begin{tikzcd} W \arrow[r, "\mathrm{id}_W"] \arrow[d, leftrightarrow, "\Delta_W"'] & W \arrow[d, leftrightarrow, "R"] \\ W \arrow[r, "f"'] & X \end{tikzcd} \quad \text{and} \quad \begin{tikzcd} W \arrow[r, "\mathrm{id}_W"] \arrow[d, leftrightarrow, "R"'] & W \arrow[d, leftrightarrow, "\Delta_W"] \\ X \arrow[r, "f^{-1}"'] & W \end{tikzcd}.$$

As A is a parametric functor, we obtain

$$\begin{tikzcd}
A(W) \arrow[r, "A(\mathrm{id}_W)"] & A(W) \\
A(W) \arrow[u, leftrightarrow, "A(\Delta_W)"] \arrow[r, "A(f)"'] & A(X) \arrow[u, leftrightarrow, "A(R)"']
\end{tikzcd}$$

and

$$\begin{tikzcd}
A(W) \arrow[r, "A(\mathrm{id}_W)"] & A(W) \\
A(X) \arrow[u, leftrightarrow, "A(R)"] \arrow[r, "A(f^{-1})"'] & A(W) \arrow[u, leftrightarrow, "A(\Delta_W)"']
\end{tikzcd}.$$

If $a \in A(W)$ then the left-hand diagram implies that $a[A(R)]A(f)a$, using the fact that A preserves diagonal relations and identity morphisms. Conversely, if $a[A(R)]b$ then the right-hand diagram implies that $A(f)a = b$, and the graphs of $A(f)$ and $A(R)$ are therefore equal. ■

We are now in a position to give (sufficient) conditions under which the naturality requirements are superfluous.

Theorem 10 (Naturality)

Suppose $A\colon \mathcal{W} \to \mathcal{S}$ *is a parametric functor and* $p[-]\colon A(-) \to \langle\!\langle\beta\rangle\!\rangle(-)$ *is a family of functions satisfying the following parametricity condition: for all* $R\colon X_0 \leftrightarrow X_1$,

$$\begin{tikzcd}
A(X_0) \arrow[r, "p{[X_0]}"] & \langle\!\langle\beta\rangle\!\rangle(X_0) \\
A(X_1) \arrow[u, leftrightarrow, "A(R)"] \arrow[r, "p{[X_1]}"'] & \langle\!\langle\beta\rangle\!\rangle(X_1) \arrow[u, leftrightarrow, "\langle\!\langle\beta\rangle\!\rangle(R)"']
\end{tikzcd}.$$

Then p *is automatically natural: for all* $g\colon X \to Y$ *in* $\mathcal{W}_v$,

$$\begin{tikzcd}
A(X) \arrow[r, "p{[X]}"] \arrow[d, "A(g)"'] & \langle\!\langle\beta\rangle\!\rangle(X) \arrow[d, "\langle\!\langle\beta\rangle\!\rangle(g)"] \\
A(Y) \arrow[r, "p{[Y]}"'] & \langle\!\langle\beta\rangle\!\rangle(Y)
\end{tikzcd}.$$

Proof: Consider any g. By the Expansion Factorization Lemma it can be factored into a composite $e\,;i$, where $e\colon X \to X \times W$ is an expansion and $i\colon X \times W \to Y$ is an isomorphism. The result will follow if we can show commutativity of

$$\begin{tikzcd}
A(X) \arrow[r, "p{[X]}"] \arrow[d, "A(e)"'] & \langle\!\langle\beta\rangle\!\rangle(X) \arrow[d, "\langle\!\langle\beta\rangle\!\rangle(e)"] \\
A(X \times W) \arrow[r, "p{[X \times W]}"] \arrow[d, "A(i)"'] & \langle\!\langle\beta\rangle\!\rangle(X \times W) \arrow[d, "\langle\!\langle\beta\rangle\!\rangle(i)"] \\
A(Y) \arrow[r, "p{[Y]}"] & \langle\!\langle\beta\rangle\!\rangle(Y)
\end{tikzcd}$$

The commutativity of the bottom part follows immediately from the Isomorphism Correspondence Lemma and the parametricity property for p. We will show the commutativity of the top part for $\beta = \mathbf{comm}$; the other base types are treated similarly.

Consider any $w \in W$. Define $R_w: X \leftrightarrow X \times W$ by $x[R_w](x', w')$ iff $x = x'$ and $w = w'$. Clearly we have

$$\begin{tikzcd}
X \arrow[r, "\mathrm{id}_X"] \arrow[d, leftrightarrow, "\Delta_X"'] & X \arrow[d, leftrightarrow, "R"] \\
X \arrow[r, "e"'] & X \times W
\end{tikzcd}$$

. Thus, as A is a parametric functor, for any $a \in A(X)$ we have that $a[A(R_w)]A(e)(a)$, and so, using the parametricity of p, we get

$$p[X]\, a\, x\ [R_w]\ p[X \times W](A(e)a)\, \langle x, w\rangle$$

for any $x \in X$. From the definition of R_w, and of $\langle\!\langle \mathbf{comm} \rangle\!\rangle$ on morphisms, this implies that

$$\langle\!\langle \mathbf{comm} \rangle\!\rangle(e)\, (p[X]\, a)\, \langle x, w\rangle \quad = \quad p[X \times W](A(e)a)\, \langle x, w\rangle.$$

As this argument works for any $w \in W$, the commutativity of the top part of the diagram follows. ■

Note that the naturality constraints in $\langle\!\langle \vec{\theta} \longrightarrow \beta \rangle\!\rangle X$ are also superfluous by this result, taking $A = h^X \times \langle\!\langle \vec{\theta} \rangle\!\rangle$.

We are finally in a position to see where the category-free semantics of types given earlier comes from. First, in a type $\vec{\theta} \longrightarrow \beta$ we can do away with all naturality constraints, as these are implied by parametricity. Second, by the Expansion Factorization Lemma any $\mathcal{W}_v$-morphism factors into a "true expansion" followed by an isomorphism. Further, by the Isomorphism Correspondence Lemma the action of a procedure meaning on the isomorphism part of such a factor is completely determined by the action of parametric functors on relations. Thus, when defining a procedure meaning p at store shape W we do not need to consider *all* $\mathcal{W}_v$-morphisms out of W, but only the "true expansions" of the form $W \longrightarrow W \times X$. (An analogous property for certain functor categories has been observed by I. Stark.)

Theorem 11 (Representation)

Suppose $A: \mathcal{W} \longrightarrow \mathcal{S}$ is a parametric functor. Then $(\langle\!\langle \beta \rangle\!\rangle^A)W$ is isomorphic to the collection of those families

$$p[\text{-}]: A(W \times \text{-}) \longrightarrow \langle\!\langle \beta \rangle\!\rangle(W \times \text{-})$$

satisfying the following parametricity condition: for all $R: X_0 \leftrightarrow X_1$,

$$\begin{tikzcd}
A(W \times X_0) \arrow[r, "{p[X_0]}"] \arrow[d, leftrightarrow, "A(\Delta_W \times R)"'] & \langle\!\langle \beta \rangle\!\rangle(W \times X_0) \arrow[d, leftrightarrow, "\langle\!\langle \beta \rangle\!\rangle(\Delta_W \times R)"] \\
A(W \times X_1) \arrow[r, "{p[X_1]}"'] & \langle\!\langle \beta \rangle\!\rangle(W \times X_1)
\end{tikzcd}$$

Proof: Let D denote the collection of p's satisfying parametricity. We will set up an isomorphism $f: \langle\!\langle \beta \rangle\!\rangle^A W \longrightarrow D$ with inverse g. First we have $f\, m\, [X] = m[W \times X]\langle e, \cdot\rangle$, where $e: W \longrightarrow W \times X$ is the expansion. Conversely, if we have a map $(\phi, \rho): W \longrightarrow Z$ then this factors into an expansion followed by an isomorphism $W \xrightarrow{e} W \times Y \xrightarrow{i} Z$. Then we set $g\, p\, [Z]\, \langle(\phi, \rho), a\rangle = \langle\!\langle \beta \rangle\!\rangle(i)\, (p[Y](A(i^{-1})a))$, where i^{-1} is the inverse of the iso i.

(In this definition of g, the factors e and i are not uniquely determined; however, it is easy to show, using parametricity on isomorphisms, that $\langle\!\langle \beta \rangle\!\rangle(i)\, (p[Y](A(i^{-1})a))$ is uniquely determined for any factorizations. In any case, Oles has shown how a canonical choice of factors is possible.)

That $f\,m$ satisfies parametricity is immediate from the parametricity of m, using the fact that
$$\begin{tikzcd} W \arrow[r, "e"] \arrow[d, leftrightarrow, "\Delta_W"'] & W \times X \arrow[d, leftrightarrow, "\Delta_W \times R"] \\ W \arrow[r, "e"'] & W \times X \end{tikzcd}$$
, for any R, and for e the expansion. That $g\,p$ satisfies the parametricity condition for $\langle\!\langle\beta\rangle\!\rangle^A W$ follows from the parametricity condition on p, together with the Isomorphism Correspondence Lemma and the Isomorphism Factorization Lemma. Naturality is then a result of the Naturality Theorem. Thus we see that f and g are well-defined. We can show that they are inverse as follows.

$$\begin{array}{rcl} g\,(f\,m)\,[Z]\,\langle(\phi,\rho),a\rangle & = & \langle\!\langle\beta\rangle\!\rangle\, i\,(f\,m\,[Y]\,(A(i^{-1})a)) \\ & = & \langle\!\langle\beta\rangle\!\rangle\, i\,(m\,[W\times Y]\,\langle e,(A(i^{-1})a)\rangle) \\ & = & m\,[Z]\,\langle(e\,;i),a\rangle \\ & = & m\,[Z]\,\langle(\phi,\rho),a\rangle \end{array}$$

where the second-last step uses naturality of m and the fact that i and i^{-1} are inverse isos. Conversely, the definitions of f and g give us

$$\begin{array}{rcl} f\,(g\,p)\,[X]\,a & = & g\,p\,[W\times X]\,\langle(e\colon W\longrightarrow W\times X),a\rangle \\ & = & \langle\!\langle\beta\rangle\!\rangle\, i\,(p[X]a) \end{array}$$

and in the factorization of e (in the last step) we can take i as the identity (because $e\,;\mathrm{id}=e$), so $f\,(g\,p)[X]a = p[X]a$. ■

Thus we see that the calculation of (the object part of) function types in the category-free semantics is isomorphic to what is obtained from exponentiation in the parametric-functor semantics. It is also not difficult to see that the relation parts of the two semantics are isomorphic, and that the *expand* maps correspond to the morphism parts of parametric functors. Furthermore, the semantics of λ-abstraction and application that were given are precisely those obtained (after suitable uncurrying) from the Cartesian closed structure of $\mathcal{S}^{\mathcal{W}}$. The details of these aspects of the correspondence should be abundantly clear to a reader who has followed so far, and are sufficiently routine to warrant omission.

There is one final matter that we must clear up. We have thus far steadfastly adhered to an "uncurried" presentation of the semantics of types, whereas in the ccc $\mathcal{S}^{\mathcal{W}}$ this is of course not necessary. The uncurried presentation is needed, however, for the category-free semantics to work properly. The reason is that parametricity does not imply naturality in general, but only for parametric functors of a specific form.

It will be simpler if we discuss this relationship between parametricity and naturality first in the context of the category-free semantics, and then translate to the categorical one. Consider the type **comm** → **comm**, and the family of elements $m[-] \in [\![\mathbf{comm} \rightarrow \mathbf{comm}]\!][-]$ defined by

$$m[X][Y]\,c\,\langle x,y\rangle \;=\; \langle x,y'\rangle,\;\text{ where } c\langle x,y\rangle = \langle x',y'\rangle.$$

This family of elements is "parametric" in the following sense: for all relations $R\colon X \leftrightarrow X'$ between store shapes, $m[X]\,[[\![\mathbf{comm} \rightarrow \mathbf{comm}]\!]R]\,m[X']$. (Following the analogy with polymorphism, m is essentially an element of $\forall\alpha\forall\gamma.\,(\alpha\times\gamma \rightarrow \alpha\times\gamma) \rightarrow (\alpha\times\gamma \rightarrow \alpha\times\gamma))$. For

m to be natural with respect to expansions we would need that

$$\begin{array}{ll} \text{if} & m[X \times Y][Z]c\langle\langle x, y\rangle, z\rangle = \langle\langle x1, y1\rangle, z1\rangle \\ \text{and} & m[X][Y \times Z]c^*\langle x, \langle y, z\rangle\rangle = \langle x2, \langle y2, z2\rangle\rangle \\ \text{then} & \quad x1 = x2,\ y1 = y2,\ \text{ and }\ z1 = z2 \end{array}$$

where c^* is obtained from c by the evident associativity isomorphism. From the definition of m, if $c\langle\langle x, y\rangle, z\rangle = \langle\langle x, y'\rangle, z\rangle$, so $c^*\langle x, \langle y, z\rangle\rangle = \langle x, \langle y', z\rangle$, we get that

$$m[X \times Y][Z]c\langle\langle x, y\rangle, z\rangle = \langle\langle x, y\rangle, z\rangle$$

while

$$m[X][Y \times Z]c^*\langle x, \langle y, z\rangle\rangle = \langle x, \langle y', z\rangle\rangle.$$

The naturality property fails because y and y' need not be equal, as c can certainly change this component.

Expressing this argument more categorically, we can define a family of functions

$$m[-]\colon \mathbf{1}(-) \rightarrow \langle\!\langle \mathbf{comm} \rightarrow \mathbf{comm} \rangle\!\rangle(-)$$

that satisfies parametricity, but not naturality. The definition is

$$m[X](*)[Z]\langle(\phi, \rho), c\rangle\, s \ = \ \rho\langle\phi(s), c(s)\rangle.$$

This clearly satisfies parametricity, but the naturality diagram

$$\begin{tikzcd} \{*\} \arrow[r, "{m[X]}"] \arrow[d, "\mathrm{id}"'] & \langle\!\langle \mathbf{comm} \rightarrow \mathbf{comm} \rangle\!\rangle X \arrow[d, "\langle\!\langle \mathbf{comm} \rightarrow \mathbf{comm} \rangle\!\rangle e"] \\ \{*\} \arrow[r, "{m[X \times Y]}"'] & \langle\!\langle \mathbf{comm} \rightarrow \mathbf{comm} \rangle\!\rangle (X \times Y) \end{tikzcd}$$

fails, for e an expansion, using essentially the same counterexample as above. That is, for

$$e'\colon X \times Y \rightarrow (X \times Y) \times Z$$

the state $m[X](*)[(X \times Y) \times Z]\langle(e\,;e'), c\rangle\,\langle\langle x, y\rangle, z\rangle$ will not necessarily be equal to the state $m[X \times Y](*)[(X \times Y) \times Z]\langle e', c\rangle\,\langle\langle x, y\rangle, z\rangle$.

From this we see a curious property. While parametricity implies naturality for all families of maps in the correct position to qualify as a transformation from $\langle\!\langle \mathbf{comm} \rangle\!\rangle$ to $\langle\!\langle \mathbf{comm} \rangle\!\rangle$, the analogous property does not hold for maps from $\mathbf{1}$ to $\langle\!\langle \mathbf{comm} \rightarrow \mathbf{comm} \rangle\!\rangle$. Thus, we see the reason for the uncurried presentation of types that we gave in the category-free semantics: *the property that relational parametricity implies naturality is not stable under currying and uncurrying isomorphisms.*

At this point it is worth mentioning that these observations are not at all at odds with the result of (Plotkin and Abadi, 1993) that relational parametricity implies (di)naturality. This result applies under assumptions that are not met here. (For instance, in (Plotkin and Abadi, 1993) the source and target categories are the same, while here the source $\mathcal{W}_v$ is different from the target $\mathcal{S}_v$.)

10 The PER Model

In presenting a model based on partial equivalence relations we are taking the opposite tack to the one taken with the relational model. We begin with a presentation based on functors and natural transformations, and work our way back towards a functor-free description.

Once the decision has been made to re-cast the ideas of (Reynolds, 1981b; Oles, 1982) in a realizability setting, the definition of the model falls out almost immediately. We work with a category of "realizable" functors PER^{Σ}, where PER is the usual category of partial equivalence relations and Σ is a suitable version of Oles's category of store shapes. As most of the definitions are essentially as in (Oles, 1982), we will move fairly quickly over the material in this section. The point of the development is to show how this simple re-casting of the Reynolds–Oles ideas gives us good uniformity conditions for reasoning about local variables.

We will be working with categories equipped with a notion of realizability. These structures can be viewed elegantly as internal categories in the effective topos, or in the category of ω-sets (see (Hyland, 1988; Hyland et al., 1989; Longo and Moggi, 1991)). To simplify the presentation we will keep internal-category aspects of the model in the background (though this viewpoint certainly guides the definitions).

We use $m \cdot n$ to denote Kleene application on ω, the natural numbers (i.e., the application of the m'th partial recursive function to n). $\langle -,- \rangle$ is a recursive bijection from $\omega \times \omega$ to ω, and fst and snd are numbers such that $\mathit{fst} \cdot \langle m, n \rangle = m$ and $\mathit{snd} \cdot \langle m, n \rangle = n$. We let pid denote a code for the identity function on ω, and pcomp a realizer for composition in diagrammatic order, so $\mathit{pcomp} \cdot m \cdot n \cdot a = n \cdot (m \cdot a)$. (We adopt the convention that Kleene application associates to the left.)

A per A is a partial equivalence relation (transitive, symmetric) on the natural numbers. The equivalence class of n is $[n]_A = \{m \mid n[A]m\}$. The set of equivalence classes is $Q(A) = \{[p]_A \mid p[A]p\}$. The domain of A is $\mathit{dom}(A) = \{n \mid n[A]n\}$.

A morphism $f \colon A \longrightarrow B$ of pers is a function from $Q(A)$ to $Q(B)$ such that

$$\exists n.\, \forall p.\, p[A]p \text{ implies } f([p]_A) = [n \cdot p]_B$$

(This assumes that $n \cdot p$ is defined.) We say that n realizes f (notation: $n \models f$), and often write $|n| \colon A \longrightarrow B$ to indicate a map that n realizes. Composition is just composition of functions. This defines the category PER.

Ob(PER) and Mor(PER) are the sets of objects and morphisms of PER. There is no notion of realizability for objects of PER, or rather this notion is trivial:

$$\forall A \in \mathrm{Ob}(\mathrm{PER})\, \forall n.\, n \models A.$$

PER is cartesian closed. A terminal object $\mathbf{1}$ is the per that relates all natural numbers, so it has one equivalence class. If A and B are pers, then the pers $A \times B$ and $A \Rightarrow B$ are defined by

$$\langle a, b \rangle\, [A \times B]\, \langle a', b' \rangle \ \text{ iff } \ a[A]a' \wedge b[B]b',$$

$$m\, [A \Rightarrow B]\, n \ \text{ iff } \ \forall a, a'.\, a[A]a' \text{ implies } (m \cdot a)\, [B]\, (n \cdot a')$$

Again we will ignore recursion in this semantics. It could be incorporated using one of the PER categories that possess domain-theoretic structure (Amadio, 1989; Freyd et al., 1990; Phoa, 1990).

10.1 Store Shapes

Oles's construction of the category of store shapes can be carried out starting from any category $\mathbf{C}$ with finite products, by expressing the equational constraints on morphisms as commutative diagrams. The resulting category $\Sigma(\mathbf{C})$ is as follows. (The proof that this is indeed a category follows routinely as in (Oles, 1982).)

Objects. The objects are those of $\mathbf{C}$.

Morphisms. A $\Sigma(\mathbf{C})$-morphism from W to X is a pair of maps $\phi : X \longrightarrow W$ and $\rho : W \times X \longrightarrow X$ in $\mathbf{C}$ such that the following three diagrams commute:

$$\begin{tikzcd}
X \times X \arrow[r, "\phi \times \mathrm{id}"] & W \times X \arrow[d, "\rho"] \\
X \arrow[u, "\mathrm{diag}"] \arrow[r, "\mathrm{id}"'] & X
\end{tikzcd}
\qquad
\begin{tikzcd}
X \arrow[r, "\phi"] & W \\
W \times X \arrow[u, "\rho"] \arrow[ur, "\mathrm{fst}"'] &
\end{tikzcd}$$

$$\begin{tikzcd}
W \times (W \times X) \arrow[r, "\mathrm{fst} \times (\mathrm{snd}\,;\mathrm{snd})"] \arrow[d, "\mathrm{id} \times \rho"'] & W \times X \arrow[d, "\rho"] \\
W \times X \arrow[r, "\rho"'] & X
\end{tikzcd}$$

Identities. The identity on X is $(\mathrm{id}_X, \mathrm{fst})$, where fst is the first projection.

Composition. If $\phi, \rho : X \longrightarrow Y$ and $\phi', \rho' : Y \longrightarrow Z$, their composite is ϕ'', ρ'' where $\phi'' = \phi' \,;\, \phi$ and ρ'' is $\langle ((\mathrm{id} \times \phi') \,;\, \rho), \mathrm{snd} \rangle \,;\, \rho'$. (Here $\langle \cdot, \cdot \rangle$ is the pairing associated with the Cartesian structure in $\mathbf{C}$, not the recursive pairing bijection on ω.)

For example, $\Sigma(\mathbf{C})$ is Oles's category of store shapes for a suitable small cartesian subcategory $\mathbf{C}$ of the category of sets. More interestingly (as pointed out by A. Pitts), the category $\mathcal{W}_e$ of relations between store shapes from Section 7 is also a category of the form $\Sigma(\mathbf{C})$, for $\mathbf{C}$ a suitable (small) subcategory of the category $\mathcal{S}_e$ of binary relations and relation-preserving pairs of functions. This is further justification for the definition of related $\mathcal{W}_v$-morphisms.

We are going to work with $\Sigma(\mathrm{PER})$ as our category of store shapes; in this section, we simply call this Σ. As with PER, there will be no realizability relation for objects. For morphisms, if $(\phi, \rho) : W \longrightarrow X$ in Σ then $\langle m, n \rangle \models (\phi, \rho)$ iff $m \models \phi : X \longrightarrow W$ and $n \models \rho : W \times X \longrightarrow W$ as PER maps. Note that here $\langle m, n \rangle$ is not a pair, but a number produced by the pairing bijection. We again use the notation $|m| : X \longrightarrow Y$ for a morphism in Σ realized by m. (The ambiguity in the notations $\models$ and $| \cdot |$, which are used both for PER and Σ, is always resolved by the context.)

The expansion maps $X \longrightarrow X \times V$ are realized by