Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science

Electronic Notes in Theoretical Computer Science 298 (2013) 325–348

www.elsevier.com/locate/entcs

# Automata-Theoretic Semantics of Idealized Algol with Passive Expressions

Uday S. Reddy[1]

*School of Computer Science*
*University of Birmingham*
*Birmingham, U.K*

**Abstract**

Passive expressions in Algol-like languages represent computations that read the state but do not modify it. The need for such read-only computations arises in programming logics as well as in concurrent programming. It is also a central facet in Reynolds's Syntactic Control of Interference. Despite its importance and essentially basic character, capturing the notion of passivity in semantic models has proved to be difficult. In this paper, we provide a new model of passive expressions using an automata-theoretic framework recently proposed by the author. The central idea is that the store of a program is viewed as an abstract form of an automaton, with a representation of its states as well as state transitions. The framework allows us to combine the strengths of conventional state-based models and the more recent event-based models to synthesize new "automata-based" models. Once this basic framework is set up, relational parametricity does the job of identifying passive computations.

*Keywords:* Idealized Algol, Relational parametricity, Functor categories, Reflexive graphs, Algebraic automata theory.

## 1 Introduction

We expect that denotational semantic models of programming languages provide a rigorous *conceptual* foundation for reasoning about programs. In devising such models, one is faced with the challenge of how best to capture the intuitions the programmers possess in understanding computations and incorporate them in a rigorous theoretical framework.

The traditional models for imperative programming languages, dating back to those of Scott and Strachey, are *state-based*. These models envisage that programs operate on a store which goes through states. Commands are interpreted as *functions* from states to states, factoring out all the internal state manipulation details carried out by them. Thus, these models may be regarded as being *extensional*

[1] Email: u.s.reddy@bham.ac.uk


http://dx.doi.org/10.1016/j.entcs.2013.09.020

in their treatment of the store. Examples of such models include the original models due to Scott and Strachey [42], the functor category models initiated by Reynolds [30,38,45] and their refinements using relational parametricity [26,27].

In more recent developments, an alternative *event-based* approach for modeling computations has come to the fore. These models eschew any notion of store or state. They view commands as *processes* that interact with the individual storage variables via interaction events. The process-based view of commands exposes all their internal state manipulation details and makes the models *intensional*. On balance, however, the data abstraction and information hiding aspects of storage variables are captured more directly in these models. They are also able to model the intensional aspects of the computations such as the idea of "irreversible state change," leading to strong full abstraction results. Examples of such event-based models include the process calculus models due to Milner and Hoare [14,21], Brookes's trace models [7], the author's object-based models [20,25,33,34] and the games models [1].

The difference between extensional and intensional models becomes manifest in reasoning about program equivalences such as:

$$\mathbf{gv}(x) \implies (x := x + 1;\ x := x + 1) \equiv (x := x + 2) \tag{1}$$

where $\mathbf{gv}(x)$ represents the condition that $x$ is a "good variable" obtained by variable allocation. Extensional models satisfy such equivalences because they capture the net effect of commands on the state, whereas intensional models do not. [2] However, the treatment of data abstraction (local variables) and irreversibility of state change is problematic in extensional models.

In an effort to combine the advantages of state-based and event-based models, we recently initiated a new approach using an automata-theoretic view of the store [35,36]. The store is viewed as an automaton with an explicit representation of the states as well as the state transitions. The use of states allows an extensional treatment of commands and the use of state transitions captures some aspects of the modelling available in event-based models. We showed that several program equivalences of third-order types that could not be validated in the pure state-based models are valid in this setting.

In this paper, we take a further step in the development of the automata-theoretic model by modelling *passive expressions*, as per Reynolds's original Idealized Algol [38]. Passive expressions read the storage variables to compute values, but they do not alter the store. Typical programming languages allow side-effects in expressions for practical reasons, leaving it to the programmer to use them judiciously. [3] However, passive expressions form an integral part of program reasoning. For instance, in Hoare Logic, expressions can be embedded in logical assertions,

[2] One might find it surprising that the intensional models, e.g., games models, fail to be "extensional" despite being fully abstract. The explanation is that full abstraction only guarantees the satisfaction of *unconditional* equivalences which seem inadequate to capture the extensionality of state-manipulation. The equivalence (1) is conditional.

[3] The evaluation order of expressions is often left unspecified or under-specified, so that an uncontrolled use of expression side effects is not a practical proposition in any case.

where any side effects can lead to an entirely incoherent formalism. In concurrent programming, passive expressions form an important tool for sharing resources across processes. Various program reasoning systems, ownership type systems etc. incorporate explicit annotation for "read-only" or "immutable" variables, which depend on notions of passive usage [18,22]. In particular, the use of "fractional permissions" is an advanced mechanism to capture the passive use of storage, currently an active area of research [5,6,37].

Modeling passivity in extensional models is a significant challenge because passivity appears to be an *intensional phenomenon*: what a computation does internally in order to produce its results. If we think of modelling expressions as extensional functions of type $\mathit{State} \to \mathit{Value}$, we have no handle on what such a computation might do. It might internally calculate a new state (which means a *state change* in computational terms), and do further computations within the new state to deliver the result. The new state is eventually discarded, and the expression would have had a "temporary side effect." This kind of a phenomenon can be captured syntactically by a "snap back" combinator of the form:

$$\textbf{do } C \textbf{ result } E$$

which means "execute the command $C$ and return the value of expression $E$, discarding the effects of $C$." The presence of such a snap back combinator in the semantic models breaks intuitive program equivalences. For instance, consider the equivalence: [4]

$$\begin{aligned} &\textbf{if } (\text{deref } x = 0) \textbf{ then } f(\text{deref } x) \textbf{ else } 2 \;\equiv \\ &\qquad\qquad \textbf{if } (\text{deref } x = 0) \textbf{ then } f(0) \textbf{ else } 2 \end{aligned} \tag{2}$$

where $f$ is a function procedure taking an expression argument. Since $f$ is called only in the case where $x$ is 0, giving it 0 as the argument instead of (deref $x$) should give equivalent results. However, in a semantic model that contains the snap back operator, there are functions $f$ that break this reasoning, for example:

$$f \;=\; \lambda e.\, \textbf{do } x := x + 1 \textbf{ result } e$$

With this function $f$, the LHS of (2) evaluates to 1 whereas the RHS evaluates to 0. Virtually all extensional models in the literature, with the exception of the Tennent's model [45], have such snap back combinators.

We get around the difficulty by viewing the store as an automaton, which has an explicit representation of its states $\mathcal{Q}_X$ as well as its allowed state transformations $\mathcal{T}_X$. The expression type may then be thought of as a type constructor parameterized by both the components of the automaton:

$$\textsc{Exp}(\mathcal{Q}_X, \mathcal{T}_X) = [\mathcal{Q}_X \to \mathit{Value}]$$

[4] Imperative programming languages usually involve an implicit coercion that allows a storage variable to be treated as an expression that reads its contents. We represent this coercion as "deref" for clarity of exposition. Recall also that Idealized Algol is a call-by-name typed lambda calculus. So, the argument is passed by name in $f(\text{deref } x)$.

All computations are expected to be *parametric* [10,27,32,40], i.e., they are interpreted by parametrically polymorphic families of the form:

$$\forall \mathcal{Q}_X, \mathcal{T}_X.\ F(\mathcal{Q}_X, \mathcal{T}_X) \to \textsc{Exp}(\mathcal{Q}_X, \mathcal{T}_X)$$

where $F(\mathcal{Q}_X, \mathcal{T}_X)$ represents the semantic type of the free identifiers. Since the result type $\textsc{Exp}(\mathcal{Q}_X, \mathcal{T}_X)$ is independent of the $\mathcal{T}_X$ components, parametricity says that the family should behave the "same way," no matter what type $\mathcal{T}_X$ is employed (subject to some constraints). In particular, it should produce the same results if $\mathcal{T}_X$ is replaced by a trivial collection of state transformations, such as the one with just the identity transformation and its possibly diverging approximations. It then follows that the expression computation cannot cause any state changes, not even temporary ones. Thus passivity is captured in an intuitively satisfactory form.

The definition of this model builds on two technical innovations from our past work (joint with B. P. Dunphy). The first is the categorical axiomatization of relational parametricity presented in [10]. Since the overall structure is that of a category-theoretic possible world model, as pioneered by Reynolds [38], a categorical treatment of parametricity is needed to build the model we seek. O'Hearn and Tennent [27] initiated the building-in of relational parametricity into categories. However, their model does not have the requisite axioms, and snap back operators are present in their model. Our axiomatization is based on the notion of *fibrations*, well-studied in category theory [13,16], using which strong representation results were obtained in [10]. Its employment here gives further evidence of its power. The second innovation is the automata-theoretic modeling of the store presented in [35,36]. In retrospect, this view of the store was already implicit in Reynolds's first functor category model [38]. However, the automata-theoretic intuitions behind his model were not recognized and subsequently ignored in all further work on functor category models. Our model seems to have been the first work that builds on Reynolds's ideas. In the present work, we generalize the automata-theoretic model in a significant way, which parallels Tennent's generalization of the Oles model [45], in order to capture the seemingly intensional phenomenon of passivity. In doing such a generalization, it is easy to go too far to the other way, i.e., to make the model so intensional that the equivalence (1) fails. Tennent's model, in fact, breaks this equivalence. (Contrary to expectation, the equivalence cannot be derived in Specification Logic.) We aim to achieve a delicate balance of intensional effects and extensionality in the present paper.

*Results*

The main contribution of this paper is to provide a denotational model of Idealized Algol that satisfactorily models passivity while being extensional. In particular, this means that passive expressions do not have side effects, not even temporary ones. In the main body of the paper, we do this for a language without divergence, but treat it in such a way that it generalizes to divergence. The issues of divergence are then briefly mentioned in Sec. 5. The treatment without divergence is also novel in

that it is the first model of passivity that is able to deal with a language without divergence. All the previous models [1,34,45] depend on the presence of divergence for modeling passivity. However, intuitively, passivity is independent of the issues of divergence. Our treatment is able to decouple the two issues.

We can explain the contribution in terms of the accuracy gained at first-order types [25,34]. In the absence of divergence:

- Morphisms of type $\mathbf{com} \to \mathbf{com}$ should be isomorphic to natural numbers. They are all expressible by closed terms of the form $\lambda c.\, c^n$ where $c^n$ means an $n$-fold sequential composition $c; \ldots ; c$. The model of [36] has this property.
- Morphisms of type $\mathbf{com} \to \mathbf{exp}[\delta]$ should be constant functions. They are expressible by closed terms of the form $\lambda c.\, E$ for closed expression terms $E$. The present model has this property.

## 2 Semantic Framework

The semantic framework used in this paper is that of a category-theoretic possible worlds model, as advocated by Reynolds [38]. That means that the *types* of the programming language are interpreted as *type constructors* parameterized by "store shapes" (formally *functors*). For example, $\textsc{Exp}(X)$ represents the collection of expression meanings appropriate for stores of shape $X$, $\textsc{Com}(X)$ represents the collection of command meanings appropriate for stores of shape $X$ *etc.* The store shapes must form a category where morphisms $f : X \to Y$ represent ways in which a store $Y$ may be regarded as a "future world" of $X$ (typically by allocating additional storage locations). It might in fact be helpful to think of such a morphism as a "function" going in the reverse direction, $f^\sharp : Y \to X$, capturing a way of "extracting" an $X$-typed store from a $Y$-typed one. The type functors, naturally, must map such morphisms to functions. For example, $\textsc{Exp}(f\colon X \to Y)$ denotes a function that allows us to convert an expression on $X$-typed stores to one on $Y$-typed stores, which is possible because $X$-typed stores can be extracted from $Y$-typed stores.

In addition to morphisms, we consider abstract *logical relations* between stores, used for formulating the uniformity conditions of relational parametricity. For every pair of stores $X$ and $X'$, we have a notion of logical relations $R : X \leftrightarrow X'$ and a notion of morphisms preserving such relations, which is written diagrammatically as a "square":

$$\begin{array}{ccc} X & \xrightarrow{\;f\;} & Y \\ R\,\big\updownarrow & & \big\updownarrow\,S \\ X' & \xrightarrow{\;f'\;} & Y' \end{array} \tag{3}$$

and textually as $f \left[R \to S\right] f'$. (The textual notation depends on the fact that all the structures we consider in this paper are *relational*, i.e., given $f$, $f'$, $R$ and $S$, there is at most one square of the above shape. Therefore $R \to S$ may be regarded as a normal set-theoretic relation between hom-sets $X \to Y$ and $X' \to Y'$.) The type

functors also map such logical relations between stores to relations between values and "squares" of the form (3) to relation-preservation squares between functions, e.g.,

$$\begin{array}{ccc} \textsc{Exp}(X) & \xrightarrow{\textsc{Exp}(f)} & \textsc{Exp}(Y) \\ \textsc{Exp}(R)\uparrow\downarrow & & \uparrow\downarrow\textsc{Exp}(S) \\ \textsc{Exp}(X') & \xrightarrow{\textsc{Exp}(f')} & \textsc{Exp}(Y') \end{array}$$

Formally, the four components: store shapes, morphisms between store shapes, logical relations between store shapes and squares between them, form a *reflexive graph* of categories. Further, they satisfy additional axioms laid out in [10] to form a *parametricity graph*. Formal definitions describing the structure may be found in the Appendix.

In addition to the reflexive graph of store shapes, which will be described in the remainder of this section, we also make use of the reflexive graph **Set**, whose objects and morphisms are sets and functions, "logical relations" are set-theoretic relations $R \subseteq A \times A'$ and "squares" $f \left[R \to S\right] f'$ represent relation-preservation facts $\forall a, a'.\, a \left[R\right] a' \implies f(a) \left[S\right] f'(a')$. The reflexive graph **Set** also satisfies the additional requirements of parametricity graphs.

*Reader monoids*

We choose to model stores as an abstract form of automata similar to those studied in algebraic automata theory [11,15]. Each such automaton has: [5]

- a set of states $\mathcal{Q}_X$,
- a monoid of allowed state transformations $\mathcal{T}_X \subseteq [\mathcal{Q}_X \to \mathcal{Q}_X]$ (containing the identity transformation, written as $1_X$, and closed under sequential composition $a \cdot b$), and
- an operation $\text{read}_X : (\mathcal{Q}_X \to \mathcal{T}_X) \to \mathcal{T}_X$ defined by $\text{read}_X\ p = \lambda x.\, p\, x\, x$.

A structure of this form $X = (\mathcal{Q}_X, \mathcal{T}_X, \text{read}_X)$ is called a *reader monoid*. It would also be appropriate to call it a *Reynolds monoid*. The $\text{read}_X$ operation was proposed by Reynolds [38], who called it "diagonalization." To see the motivation for it, consider interpreting a command of the form **if** $p$ **then** $c_1$ **else** $c_2$. The command reads the state to compute the expression $p$ and, depending on the result, executes either $c_1$ or $c_2$, which are both expected to denote allowed transformations. The if-then-else operator thus converts a state-dependent state transformation of type $(\mathcal{Q}_X \to \mathcal{T}_X)$ to a state transformation of type $\mathcal{T}_X$. It is definable using the $\text{read}_X$ combinator as $\text{cond}_X\, e\, a_1\, a_2 = \text{read}_X\ (\lambda x.\, \text{read}_X\, (es)\, a_1\, a_2)$. If a given automaton $(\mathcal{Q}_X, \mathcal{T}_X)$ does not have a $\text{read}_X$ operation, additional transformations can be added to $\mathcal{T}_X$ to obtain a reader monoid. We call it the "read-closure" of the original automaton.

[5] For reasons of exposition, we will ignore the issues of divergence in the main body of the paper. However, see Sec. 5 for the extensions needed for divergence.

As examples of reader monoids, consider a store $Z$ with

$$\mathcal{Q}_Z = Int \qquad \mathcal{T}_Z = \{\, a\colon Int \to Int \mid a(z) \geq z \,\}$$

This store contains a single integer variable and allows it to be increased during computations (but not decreased).

A "passive" store $W$ has some state set, but only the do-nothing transformation $\mathcal{T}_W = \{1_W\}$. For every store $X$, there is a corresponding *passive store of* $X$, denoted $X_0$, which has the same state set as that of $X$ and the trivial set of state transformations $\mathcal{T}_X = \{1_X\}$.

The automata used in [35,36,38], called Reynolds *transformation monoids*, have an additional element of structure:

- a monoid action of type $\alpha_X : \mathcal{T}_X \to (\mathcal{Q}_X \to \mathcal{Q}_X)$ which represents a way of "running" a transformation on the states.

Here, we drop this operation, obtaining generality in the structures as well as the corresponding morphisms and logical relations. The justification for the generalization is that states in imperative programs are "abstract," available for inspection only by other commands but not by external interfaces. By requiring that logical relations only preserve the read operation, and not the monoid action, we obtain more relations, which gives a stronger parametricity criterion. Recall the intuitive argument given in the Introduction, where we replace a state transformation component of $\mathcal{T}_X$ by a trivial one. Note that the new transformation will have *different* on the state from the one we replace. So, this generalization is crucial for modelling passivity.

A *logical relation* of reader monoids $R : X \leftrightarrow X'$ is a pair $(R_q, R_t)$ where

- $R_q : \mathcal{Q}_X \leftrightarrow \mathcal{Q}_{X'}$ is a normal relation of sets, and
- $R_t : \mathcal{T}_X \leftrightarrow \mathcal{T}_{X'}$ is a monoid relation (compatible with identity transformation and composition),

such that $\mathrm{read}_X \left[(R_q \to R_t) \to R_t\right] \mathrm{read}_{X'}$. The identity logical relation of a reader monoid $X$ is $I_X = (\Delta_{\mathcal{Q}_X}, \Delta_{\mathcal{T}_X})$ consisting of the diagonal relations on both the state sets and the transformations.

A *morphism* of reader monoids $f : X \to Y$, representing a way of expanding a "current world" $X$ to a "future world" $Y$, is a pair $(f_q, f_t)$: where

- $f_q : \mathcal{Q}_Y \to \mathcal{Q}_X$ is a *surjective* function (note the reversal of direction), and
- $f_t : \mathcal{T}_X \to \mathcal{T}_Y$ is an *injective* monoid morphism,
- satisfying $f_t(\mathrm{read}_X(p)) = \mathrm{read}_Y(f_t \circ p \circ f_q)$.

The condition on read can also be written using relational notation as $\mathrm{read}_X \left[\langle f_q \to f_t\rangle \to \langle f_t\rangle\right] \mathrm{read}_Y$, where the notation $\langle -\rangle$ denotes the graph of a function. The bidirectional information flow of $f_q$ and $f_t$, similar to that in intensional models [1,34], may be understood by thinking of the morphism $f$ as *extracting* $X$-typed stores from $Y$-typed stores in the *reverse* direction. To do such extraction, it should be possible to interpret all $Y$-typed states as $X$-typed states, which is

done by the function $f_q$. The transformations of the stores, on the other hand, are invoked by the computational environment in which the store is embedded. If the environment requests a transformation $a$ on the $X$-typed store, the extraction must simulate it as a transformation $f_t(a)$ on the $Y$-typed store.

A *square* of reader monoids is defined as a pair of relation-preservation squares (for sets and monoids, respectively):

$$\begin{array}{ccc} X & \xrightarrow{f} & Y \\ \big\updownarrow R & & S \big\updownarrow \\ X' & \xrightarrow{f'} & Y' \end{array} \iff \begin{array}{ccc} \mathcal{Q}_Y & \xrightarrow{f_q} & \mathcal{Q}_X \\ \big\updownarrow S_q & & R_q \big\updownarrow \\ \mathcal{Q}_{Y'} & \xrightarrow{f'_q} & \mathcal{Q}_{X'} \end{array} \wedge \begin{array}{ccc} \mathcal{T}_X & \xrightarrow{f_t} & \mathcal{T}_Y \\ \big\updownarrow R_t & & S_t \big\updownarrow \\ \mathcal{T}_{X'} & \xrightarrow{f'_t} & \mathcal{T}_{Y'} \end{array}$$

Note that the squares on the right (in **Set** and **Mon**) have their standard meaning:

$$\begin{array}{l} \forall y \in \mathcal{Q}_Y,\, y' \in \mathcal{Q}_{Y'}.\; y \left[S_q\right] y' \implies f_q(y) \left[R_q\right] f'_q(y') \\ \forall a \in \mathcal{T}_Y,\, a' \in \mathcal{T}_{Y'}.\; a \left[R_t\right] a' \implies f_t(a) \left[S_t\right] f'_t(a') \end{array}$$

This data constitutes a *reflexive graph category* **RM** of Reynolds monoids.

*Parametricity graphs*

The so-called "parametricity graphs" are reflexive graphs of categories satisfying certain axioms, proposed in [10] for modelling relational parametricity. A parametricity graph is a reflexive graph that is:

- *relational*, i.e., there is at most one square of a given shape,
- *fibred* with chosen cleavage, and
- satisfies the *identity condition*, i.e., whenever $f \left[I_A \to I_B\right] g$, we have $f = g$.

The "relational" condition essentially simplifies the theory. The "identity condition" gives semantics to the identity logical relations. The "fibred" condition is a categorical treatment of inverse images of relations. (See Appendix for full definitions.) Given $f$, $f'$ and $S$ as in the square on the left below, there must be a pre-image $\langle f, f' \rangle^* S$ that can fill the dotted line in a universal way:

$$\begin{array}{ccc} A & \xrightarrow{f} & B \\ \langle f, f' \rangle^* S \,\vdots & & \big\updownarrow S \\ A' & \xrightarrow{f'} & B' \end{array} \qquad \begin{array}{ccc} A & \xrightarrow{f} & B \\ \big\updownarrow R & & \vdots\, \langle f, f' \rangle_! R \\ A' & \xrightarrow{f'} & B' \end{array}$$

Squares of this form are called *cartesian squares*. The dual form of squares, *cocartesian squares*, give "direct images" $\langle f, f' \rangle_! R$. The reflexive graph **Set** has both pre-images and direct images, given by:

$$\begin{array}{l} \langle f, f' \rangle^* S = \{\, (x, x') \mid f(x) \left[S\right] f'(x') \,\} \\ \langle f, f' \rangle_! R = \{\, (f(x), f'(x')) \mid x \left[R\right] x' \,\} \end{array}$$

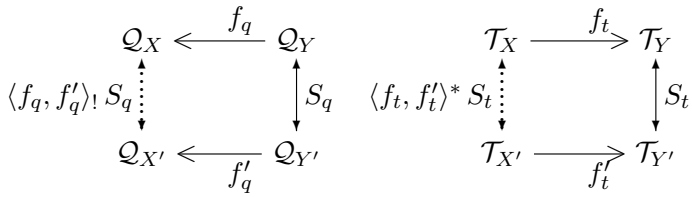
| | |
|---|---|
| $\textsc{Com}(X) = \mathcal{T}_X$ | $\textsc{Com}(R) = R_t$ |
| $\textsc{Exp}_\delta(X) = [\mathcal{Q}_X \to [\![\delta]\!]]$ | $\textsc{Exp}_\delta(R) = [R_q \to \Delta_{[\![\delta]\!]}]$ |
| $\textsc{Var}_\delta(X) = \textsc{Exp}_\delta(X) \times [[\![\delta]\!] \to \textsc{Com}(X)]$ | $\textsc{Var}_\delta(R) = \textsc{Exp}_\delta(R) \times [\Delta_{[\![\delta]\!]} \to \textsc{Com}(R)]$ |
| $(F \times G)(X) = F(X) \times G(X)$ | $(F \times G)(R) = F(R) \times G(R)$ |
| $(F \Rightarrow G)(X) = \forall_{h:Z \leftarrow X}\, [F(Z) \to G(Z)]$ | $(F \Rightarrow G)(R) = \forall_{S \leftarrow R}\, [F(S) \to G(S)]$ |

Fig. 1. Interpretation of Idealized Algol types

The reflexive graph **RM** is a also parametricity graph, in contrast to the one used by O'Hearn and Tennent [27] . It satisfies the identity condition because it is obtained by putting together **Set** and **Mon**, both of which satisfy the identity condition. It is fibred with chosen cleavage:

$$\langle f, f' \rangle^*\, S = (\langle f_q, f'_q \rangle_!\, S_q,\ \langle f_t, f'_t \rangle^*\, S_t)$$

This implies in particular that there is a "subsumption map" that maps each morphism $f : X \to Y$ to a logical relation $\langle f \rangle : X \leftrightarrow Y$, given by $\langle f \rangle = (\langle f_q \rangle^\smile, \langle f_t \rangle)$, such that commutative squares of morphism are sent to relation-preservation square [10]. Diagrammatically:




In addition to these facts, we note that the vertex category of **RM** satisfies the *right Ore condition* [17], allowing us to treat it as an atomic Grothendieck site.

**RM** is not cofibred in general, but it does have some useful co-cartesian squares which will be put to use in the next section.

*Type functors*

To interpret the types of Idealized Algol we use functors of appropriate kind from **RM** to **Set**, as shown in Fig. 1. This formalizes the intuition mentioned in Introduction that types are interpreted as "type constructors" parameterized by the store automaton.

A *reflexive graph-functor* (RG-functor) $F : \mathbf{G} \to \mathbf{H}$ between reflexive graphs maps all four components of the reflexive graph (objects, morphisms, logical relations and squares) preserving their structure. A *PG-functor* is a reflexive graph-functor that also preserves the *cartesian squares* and, in particular, the chosen cleavage:

$$F(\langle f, f' \rangle^* S) = \langle Ff,\ Ff' \rangle^* (FS)$$

We also insist that the functors used for interpreting Idealized Algol preserve all the *co-cartesian squares* that exist in **RM**. The category of PG-functors **RM** $\to$ **Set** of this kind is denoted $\mathcal{P}(\mathbf{RM})$, for "presheaves over **RM**."

The morphisms in $\mathcal{P}(\mathbf{RM})$ are transformations that preserve all morphisms (naturality) as well as all relations (parametricity). However, under the conditions of parametricity graphs, parametricity implies naturality [10,32]. So, we simply call them *parametric transformations*.

Next, we restrict to (atomic) *sheaves* over **RM** [17,19]. Intuitively, the idea of sheaves is that the functor actions $Ff$ on morphisms $f : X \to Y$ do not lose any information. Given an element $Ff(a) \in FY$, we can recover $a \in FX$ from it. The definition is as follows:

- Given a morphism $f : X \to Y$, a pair of morphisms $(g_1, g_2) : (Y, Y) \to (Z, Z)$ such that $f; g_1 = f; g_2$ is called a "match point" for $f$.
- An element $b \in FY$ is called a *matching element* for $f$ if, for all match points $(g_1, g_2)$, we have $Fg_1(b) = Fg_2(b)$.
- A presheaf $F$ in $\mathcal{P}(\mathbf{RM})$ satisfies the *sheaf axiom* for $f : X \to Y$ if, for all matching elements $b \in FY$, there is a unique $a \in FX$ such that $Ff(a) = b$.
- A presheaf $F$ is a *sheaf* if it satisfies the sheaf axiom for all morphisms.

It is easy to see that every image $Ff(a) \in FY$ is a matching element. The sheaf axiom says that these are the only matching elements. This being the standard definition of sheaves, there is a more elementary characterization:

**Lemma 2.1** *[17, 2.1.11(h)] A presheaf in $F$ in $\mathcal{P}(\mathcal{C})$ is an atomic sheaf iff, for every $f : X \to Y$ in $\mathcal{C}$,*

- *$Ff$ is an injective function.*
- *The image of $Ff$ is precisely the set of matching elements for $f$.*

The full subcategory of sheaves in $\mathcal{P}(\mathbf{RM})$ is denoted $\mathcal{S}(\mathbf{RM})$. The move from presheaves to sheaves is necessitated by the construction of exponentials for fibred reflexive-graphs. Their use in the semantics of state was prioneered by O'Hearn and Stark [23,43]. They also underlie framework of nominal sets [31] and, possibly, Separation Logic.

**Theorem 2.2** *If* **C** *is a parametricity graph satisfying the right Ore condition, the category $\mathcal{S}(\mathbf{C})$ of atomic sheaves over* **C** *preserving co-cartesian squares is cartesian closed.*

Products are given pointwise: $(F \times G)(X) = F(X) \times G(X)$ and $(F \times G)(R) = F(R) \times G(R)$. Exponents are given as in presheaf categories: $(F \Rightarrow G)(X) = \forall_{h:Z \leftarrow X}[F(Z) \to G(Z)]$, where $\forall$ denotes the "parametric limit" (in **Set**) indexed by morphisms $h$ originating from $X$ [10]. Explicitly, the parametric limit consists of families of the form

$$\langle t_h \in [F(Z) \to G(Z)] \rangle_{h:X \to Z}$$

that are *parametric* in the sense that

$$h \left[I_X \to S\right] h' \Rightarrow t_h \left[F(S) \to G(S)\right] t_{h'}$$

Since $F$ and $G$ are PG-functors, such families are automatically natural [10]. The relation $(F \to G)(R) = \forall_{S \leftarrow R}[F(S) \to G(S)]$ relates two families $\langle t_h \rangle_{h:X \to Z}$ and $\langle t'_{h'} \rangle_{h':X' \to Z'}$ iff, for all logical relations $S : Z \leftrightarrow Z'$ and all $h, h'$ of appropriate types:

$$h \left[R \to S\right] h' \Longrightarrow t_h \left[F(S) \to G(S)\right] t'_{h'}$$

# 3 Modeling Passivity

Intuitively, a computation is passive if it *reads* the state but carries out no state changes. Since our stores $X = (\mathcal{Q}_X, \mathcal{T}_X)$ have a state set component and a state transformation component, this means that passive computations should only depend on the $\mathcal{Q}_X$ components and be independent of the $\mathcal{T}_X$ components.

We use relational parametricity to formalize these concepts. Call a logical relation $R : X \leftrightarrow X'$ a *transformer relation* if its state set component is the diagonal relation: $R_q = \Delta_{\mathcal{Q}_X}$. There are no constraints on the transformation component of the logical relation (except those imposed by reader monoids).

**Definition 3.1** *Given a PG-functor $F$ in $\mathcal{S}(\mathbf{RM})$ and a store $X$, a value $d \in FX$ is said to be* passive *if, for all transformer relations $R : X \leftrightarrow X$, $d$ is related to itself by $FR$, i.e., $d \left[FR\right] d$.*

This accords with our intuition. Since transformer relations keep the state set components of worlds fixed but allow the transformation components to vary, if a value is related to itself under all such variations, it must be independent of the transformation components. It is easy to see that all values $e \in \textsc{Exp}(X)$ are passive, as one would expect. On the other hand, in $\textsc{Com}(X)$, a value $a$ is passive if and only if $a \left[R_t\right] a$ for all transformer relations $R$. This is only possible if $a = 1_X$, the do-nothing state transformation. (When we consider divergence, the passive command values include all approximations of $1_X$.)

A PG-functor itself may be regarded as a passive functor if all its values are passive (for all stores $X$). We require this uniformly for all stores $X$.

**Definition 3.2** *A PG-functor $F$ is said to be* passive *if, for all transformer relations $R : X \leftrightarrow X$, $FR = \Delta_{FX}$.*

Note that Exp is a passive functor, and Com is not. However, Com has a passive subfunctor, denoted $\wp$Com, which includes $1_X$ at every store shape $X$. We examine how to characterize the passive subfunctors.

*Passivity monomorphism*

Recall that, for every store $X$, there is a corresponding passive store $X_0$, which has the same state set as $X$ but has only trivial state transformations $\mathcal{T}_{X_0} = \{1_X\}$.

Since $X_0$ allows no state changes, we expect that all values $d \in FX_0$ are passive (for all PG-functors $F$).

There is a passivity monomorphism $p_X : X_0 \rightarrowtail X$ given by the identity on state sets and the injection $T_{X_0} \hookrightarrow \mathcal{T}_X$ for state transformations. By Lemma 2.1, a PG-functor in $\mathcal{S}(\mathbf{RM})$ sends it to an injection $Fp_X$, making $FX_0$ a subobject of $FX$. Under the assumption that $F$ preserves co-cartesian squares in addition to cartesian squares, we can show that all passive values of $FX$ are contained within the image of $FX_0$ under $Fp_X$.

**Lemma 3.3** *If $F$ is a PG-functor in $\mathcal{S}(\mathbf{RM})$ that preserves co-cartesian squares, then a value $d \in FX$ is passive if and only if there exists $d_0 \in FX_0$ such that $Fp_X(d_0) = d$.*

The "only if" direction is based on the fact that every transformer relation $R$ has the square shown on the left below:

$$\begin{array}{ccc} X_0 & \xrightarrow{p_X} & X \\ {\scriptstyle I_{X_0}}\uparrow\downarrow & & \uparrow\downarrow{\scriptstyle R} \\ X_0 & \xrightarrow{p_X} & X \end{array} \qquad \begin{array}{ccc} FX_0 & \xrightarrow{Fp_X} & FX \\ {\scriptstyle I_{FX_0}}\uparrow\downarrow & & \uparrow\downarrow{\scriptstyle FR} \\ FX_0 & \xrightarrow{Fp_X} & FX \end{array}$$

As the PG-functor $F$ maps it to the square on the right, all the values in the image under $Fp_X : FX_0 \to FX$ are related to themselves by $FR$. Hence all such values are passive. For the "if" direction, we use the co-cartesian square shown on the left below:

$$\begin{array}{ccc} X_0 & \xrightarrow{p_X} & X \\ {\scriptstyle I_{X_0}}\uparrow\downarrow & & \uparrow\vdots{\scriptstyle \varrho_X} \\ X_0 & \xrightarrow{p_X} & X \end{array} \qquad \begin{array}{ccc} FX_0 & \xrightarrow{Fp_X} & FX \\ {\scriptstyle I_{FX_0}}\uparrow\downarrow & & \uparrow\vdots{\scriptstyle F\varrho_X} \\ FX_0 & \xrightarrow{Fp_X} & FX \end{array}$$

where $\varrho_X : X \leftrightarrow X$ is given by $(\varrho_X)_q = \Delta_{\mathcal{Q}_X}$ and $(\varrho_X)_t = \{(1_X, 1_X)\}$. Since $F$ preserves co-cartesian squares, this implies that all passive values of $FX$ are contained within the image of $Fp_X$.

*Passivity retraction relationss*

In the category of worlds used by Tennent [24,45], the passivity monomorphisms have retractions $r_X : X \to X_0$ such that $p_X; r_X = \mathrm{id}_X$, making the reverse composite $\varpi_X = r_X; p_X$ an idempotent. O'Hearn et al. defined passive values of a functor $F$ as those satisfying $F\varpi_X(d) = d$.

In contrast, our category of worlds **RM** does not have such retractions because their state transformation components $\tau_{r_X}$ would need to send all transformations in $\mathcal{T}_X$ to $1_X \in \mathcal{T}_{X_0}$ and, so, fail to be injective. Nevertheless, we can simulate the effect of the retractions via logical relations. The *passivity retraction relation* $\xi_X : X \leftrightarrow X$ is given by $(\xi_X)_q = \Delta_{\mathcal{Q}_X}$ and $(\xi_X)_t = \{\, (a, 1_X) \mid a \in \mathcal{T}_X \,\}$. This relation satisfies an important property:

**Lemma 3.4** *For all Algol type functors $F$, the relation $F\xi_X : FX \leftrightarrow FX$ has, as its domain, the entire set $FX$, and, as its range, the passive subset of $FX$.*

*Passive subfunctors*

If $F$ is a PG-functor in $\mathcal{S}(\mathbf{RM})$, there is a passive PG-functor $\wp F$ in $\mathcal{S}(\mathbf{RM})$ defined by

$$\begin{aligned}(\wp F)X &= \text{the range of } Fp_X\\ (\wp F)f &= \text{the restriction of } Ff \text{ to } (\wp F)X\end{aligned}$$

This definition is based on the following property.

**Lemma 3.5** *If $F$ is a PG-functor in $\mathcal{S}(\mathbf{RM})$ and $f : X \to Y$ a morphism in $\mathbf{RM}$ then $Ff : FX \to FY$ sends passive values in $FX$ to passive values in $FY$.*

Using Lemma 3.4, we can show the following result, establishing that passive functors form what might be called a "sub-reflective" subcategory.

**Theorem 3.6** *If $F$ and $P$ are Algol functors in $\mathcal{S}(\mathbf{RM})$ where $P$ is passive, there is a natural injection from parametric transformations $F \to P$ to parametric transformations $\wp F \to P$.*

$$Par(F,\, P) \rightarrowtail Par(\wp F,\, P)$$

**Proof.** If $t : F \to P$ is a parametric transformation, the corresponding $t_0 : \wp F \to P$ has components $(t_0)_X$ that are just the restriction of components $t_X$ to passive values. We show that $t_0$ uniquely determines $t$. Since $t$ preserves all logical relations, in particular the transformer relation $\xi_X : X \leftrightarrow X$, we have a relation-preservation square (in **Set**):

$$\begin{array}{ccc} FX & \xrightarrow{\;t_X\;} & PX \\ {\scriptstyle F\xi_X}\updownarrow & & \updownarrow{\scriptstyle P\xi_X} \\ FX & \xrightarrow{\;t_X\;} & PX \end{array}$$

Since $\xi_X$ is a transformer relation, $P\xi_X = \Delta_{PX}$. So, the above square means:

$$\forall d, d_0 \in FX.\, d \left[F\xi_X\right] d_0 \implies t_X(d) = t_X(d_0)$$

Since the range of $F\xi_X$ consists of only passive values (by Lemma 3.4), this means that $t_X$ is uniquely determined by its action on passive values. □

Using this result, we can give a semantic interpretation to the "Passification" or "Co-promotion" rule as used in the "SCI Revisited" and "ILC Revisited" type systems [24,44]:

$$\frac{\Pi \;\mid\; i : \theta, \Gamma \vdash M : \phi}{\Pi, i : \theta \;\mid\; \Gamma \vdash M : \phi} \quad \text{Passification}$$

Here, the free identifiers to the left of "|" are in the "passive zone" and those to the right are in the "active zone." The type rule says that a free identifier can be moved from the active zone to the passive zone, when used in a term $M$ of a passive type. This is precisely the effect of the natural injection $\mathrm{Par}(F, P) \rightarrowtail \mathrm{Par}(\wp F, P)$. A

rule such as this would be needed to accommodate the "block expression" construct proposed by Tennent [46].

**Theorem 3.7** *The passive subfunctor operator $\wp$ is in turn a functor $\wp : \mathcal{S}(\mathbf{RM}) \to \mathcal{S}(\mathbf{RM})$. It enjoys the following isomorphisms and embedding:*

$$\begin{array}{rcll} \wp P & \cong & P & \text{for passive functors } P \\ \wp\textsc{Com} & \cong & \mathbf{1} & \\ \wp(F \times G) & \cong & \wp F \times \wp G & \\ F \Rightarrow P & \rightarrowtail & \wp F \Rightarrow P & \text{for passive functors } P \end{array}$$

**Proof.** If $t : F \to G$ is a parametric transformation, $\wp t : \wp F \to \wp G$ is just the restriction $t_0$ of $t$ that acts on passive values. The first isomorphism is, in fact, an equality $\wp P = P$, and follows from the fact that the passive subset of $PX$ is the entire $PX$. For $\textsc{Com}$, $1_X$ is the only passive value in $\textsc{Com}(X)$. So, $\wp\textsc{Com}(X)$ is a singleton. For $F \times G$, note that $(F \times G)p_X = Fp_X \times Gp_X : FX_0 \times GX_0 \to FX \times GX$. So, $(d, e)$ is in the range of $(F \times G)p_X$ iff $d$ is in the range of $Fp_X$ and $e$ is in the range of $Gp_X$. The last embedding follows from the definition $(F \Rightarrow P)(X) = \forall_{h:Z \leftarrow X} [FZ \to PZ]$, since $[FZ \to PZ]$ embeds into $[(\wp F)Z \to PZ]$. □

# 4 Applications

In this section, we examine the consequences of the theory developed in the previous sections.

*Interpretation of Idealized Algol*

Idealized Algol [38] is a simply typed lambda calculus (with call-by-name parameter passing) with basic types that support imperative computations.

The interpretation of types $\mathbf{exp}[\delta]$, $\mathbf{com}$, $\mathbf{var}[\delta]$, $\theta_1 \times \theta_2$ and $\theta_1 \to \theta_2$ is as PG-functors in $\mathcal{S}(\mathbf{RM})$, shown in Figure 1. For readability, we have used notation such as $\textsc{Exp}_\delta$ for $[\![\mathbf{exp}[\delta]]\!]$ etc.

The interpretation of a term $M$ with typing:

$$x_1 : \theta_1, \ldots, x_n : \theta_n \vdash M : \theta$$

is a parametric transformation of type

$$[\![M]\!] : (\textstyle\prod_{x_i} [\![\theta_i]\!]) \to [\![\theta]\!]$$

This means that, for each store shape $X$, $[\![M]\!]_X$ is a function of type $(\prod_{x_i} [\![\theta_i]\!](X)) \to [\![\theta]\!](X)$ such that all relations are preserved, i.e., for any relation $R : X \leftrightarrow X'$, we have $[\![M]\!]_X \left[ (\prod_{x_i} [\![\theta_i]\!](R)) \to [\![\theta]\!](R) \right] [\![M]\!]_{X'}$. To the extent that Idealized Algol is

| | |
|---|---|
| $\mathrm{equal} : \textsc{Exp}_\delta \times \textsc{Exp}_\delta \to \textsc{Exp}_{\mathbf{bool}}$ | $\mathrm{equal}_X(e_1, e_2) = \lambda s.\, e_1(s) = e_2(s)$ |
| $\mathrm{cond}^E : \textsc{Exp}_{\mathbf{bool}} \times \textsc{Exp}_\delta \times \textsc{Exp}_\delta \to \textsc{Exp}_\delta$ | $\mathrm{cond}^E_X(e, e_1, e_2) = \lambda s.\, e(s) \to e_1(s);\ e_2(s)$ |
| $\mathrm{skip} : \mathbf{1} \to \textsc{Com}$ | $\mathrm{skip}_X(*) = 1_X$ |
| $\mathrm{seq} : \textsc{Com} \times \textsc{Com} \to \textsc{Com}$ | $\mathrm{seq}_X(a, b) = a \cdot b$ |
| $\mathrm{cond}^C : \textsc{Exp}_{\mathbf{bool}} \times \textsc{Com} \times \textsc{Com} \to \textsc{Com}$ | $\mathrm{cond}^C_X(e, a, b) = \mathrm{read}_X\ \lambda s.\, e(s) \to a;\ b$ |
| $\mathrm{for} : \textsc{Exp}_{\mathbf{int}} \times \textsc{Com} \to \textsc{Com}$ | $\mathrm{for}_X(e, a) = \mathrm{read}_X\ \lambda s.\, a^{e(s)}$ |
| $\mathrm{deref} : \textsc{Var}_\delta \to \textsc{Exp}_\delta$ | $\mathrm{deref}_X(d, a) = d$ |
| $\mathrm{assign} : \textsc{Var}_\delta \times \textsc{Exp}_\delta \to \textsc{Com}$ | $\mathrm{assign}_X((d, a), e) = \mathrm{read}_X\ \lambda s.\, a(e(s))$ |
| $\mathrm{newvar} : (\textsc{Var}_\delta \Rightarrow \textsc{Com}) \to \textsc{Com}$ | $\mathrm{newvar}_X(p) = (\lambda s.\, (s, init_\delta)) \cdot p[\iota_1](\mathrm{mkvar}\uparrow^{X \star V}_V) \cdot (\lambda (s, n).\, s)$ where $V = (\llbracket\delta\rrbracket, T(\llbracket\delta\rrbracket))$ $\quad \mathrm{mkvar} = (\lambda n.\, n,\ \lambda k.\, \lambda n.\, k)$ |

Fig. 2. Primitive operators of Idealized Algol

a simply typed lambda calculus, this interpretation is standard [10,27].

$$\begin{aligned}
&\llbracket x \rrbracket_X(u) = u(x) \\
&\llbracket \lambda x\colon \theta.\, M \rrbracket_X(u) = \Lambda h\colon Z \leftarrow X.\, \lambda d\colon \llbracket\theta\rrbracket(Z).\, \llbracket M \rrbracket_Z(u\uparrow^Z_X[x \mapsto d]) \\
&\llbracket MN \rrbracket_X(u) = \llbracket M \rrbracket_X(u)[\mathrm{id}_X\colon X \to X](\llbracket N \rrbracket_X(u))
\end{aligned}$$

The parameter $u$ may be thought of as an "environment" that provides values for the free identifiers, specifically in the given world $X$. The meaning of a lambda abstraction of type $\theta \to \theta'$ is in $(\llbracket\theta\rrbracket \Rightarrow \llbracket\theta'\rrbracket)(X)$, which consists of families of the form $\langle t_h \rangle_{h:Z\leftarrow X}$. Here, we are using notation "$\Lambda h : Z \leftarrow X$" borrowed from the polymorphic lambda calculus to express the parameterization by $h$. Note that the body of the abstraction $\lambda x\colon \theta.\, M$ is interpreted in the future world $Z$ and the environment $u$ is "upgraded" to this world. We use the mnemonic short-hand notation $a\uparrow^Z_X$ for the value $\llbracket\theta\rrbracket(f)(a)$ when $f : X \to Z$ is the morphism available in the context and $\theta$ is the type of $a$. Parametricity in $Z$ is crucial for capturing the fact that $\llbracket M \rrbracket_Z$ does not directly access any information of the future world. In the interpretation of function application terms, we are again using the polymorphic lambda calculus notation to pass in the $h$ argument, which is $\mathrm{id}_X : X \to X$.

The imperative operations can be defined as a set of primitive constants, a sample of which are shown in Fig. 2. Their interpretations should be mostly self-explanatory. We are using the notation $p \to v_1; v_2$ to denote conditional expressions in semantic meta-language. Note that Reynolds's read operation is used in interpreting conditional commands as well as assignment, both of which use the current state information to construct a state transformation. Variable are represented as pairs of operations: an expression-typed operation that dereferences the variable and an "acceptor" that, given a value, stores it in the variable. The "newvar" primitive allocates a new variable in the context of a store $X$. It defines a new piece of store $V$ with the state set $\llbracket\delta\rrbracket$ and all state-transformations on it, denoted $T(\llbracket\delta\rrbracket)$. The "mkvar" construction provides the dereference-acceptor pair on this store. To add the store $V$ to the existing store $X$, we use a tensor product on stores denoted

$\star$ [36]. The store $X \star Y$ is defined as the reader monoid:

$$\begin{aligned} \mathcal{Q}_{X\star Y} &= \mathcal{Q}_X \times \mathcal{Q}_Y \\ \mathcal{T}_{X\star Y} &= \text{read-closure of } \{\, a \times b \mid a \in \mathcal{T}_X,\, b \in \mathcal{T}_Y \,\} \end{aligned}$$

This store has evident injections $\iota_1 : X \to X \star Y$ and $\iota_2 : Y \to X \star Y$.

*Examples*

In the first place, let us note that the snap back combinator (**do** $C$ **result** $E$) is ruled out. To interpret it we would need a parametric transformation of the form:

$$\begin{aligned} &\mathrm{do} : \textsc{Com} \times \textsc{Exp} \to \textsc{Exp} \\ &\mathrm{do}_X(a, e) = \lambda s.\, e(a(s)) \end{aligned}$$

We can see that it is not parametric. For example, the preservation of the relation $\xi_X : X \leftrightarrow X$ requires

$$\begin{array}{ccc} \textsc{Com}(X) \times \textsc{Exp}(X) & \xrightarrow{\mathrm{do}_X} & \textsc{Exp}(X) \\ \Big\uparrow \textsc{Com}(\xi_X) \Big\downarrow \quad \Big\uparrow \textsc{Exp}(\xi_X) \Big\downarrow & & \Big\uparrow \textsc{Exp}(\xi_X) \Big\downarrow \\ \textsc{Com}(X) \times \textsc{Exp}(X) & \xrightarrow{\mathrm{do}_X} & \textsc{Exp}(X) \end{array}$$

which says $e(a(s)) = e(1_X(s))$ for all $a \in \textsc{Com}(X)$, $e \in \textsc{Exp}(X)$ and states $s \in \mathcal{Q}_X$. (Note that $a \left[\textsc{Com}(\xi_X)\right] 1_X$ and $e \left[\textsc{Exp}(\xi_X)\right] e$.) Since $1_X(s) = s$, we are requiring $e(a(s)) = e(s)$. The condition would be violated, for example, if $X$ has at least two states, say $\{0, 1\}$, and $a$ causes a state change, perhaps by sending 0 to 1, and $e$ returns the integer in the current state.

Consider the equivalence stated in the Introduction:

**if** (deref $x = 0$) **then** $f$(deref $x$) **else** 2 $\equiv$ **if** (deref $x = 0$) **then** $f(0)$ **else** 2

This requires that, for all worlds $X$, values $(e, a) \in \textsc{Var}(X)$ and $f \in (\textsc{Exp} \Rightarrow \textsc{Exp})(X)$:

$$\big(\lambda s.\, (e\ s) = 0 \longrightarrow f[\mathrm{id}_X]\ e\ s;\ 2\big) \ = \ \big(\lambda s.\, (e\ s) = 0 \longrightarrow f[\mathrm{id}_X]\ \overline{0}\ s;\ 2\big)$$

Consider a relation given by

$$R_q = \{\, (s, s) \mid e\ s = 0 \,\} \qquad R_t = \{(1_X,\, 1_X)\}$$

Since $e \left[\textsc{Exp}(R)\right] \overline{0}$, we must have, for all states $s$ such that $e\ s = 0$,

$$f[\mathrm{id}_X]\ e\ s \left[\Delta_{Int}\right] f[\mathrm{id}_X]\ \overline{0}\ s$$

Noting that $\Delta_{Int}$ is nothing but the equality relation, we have a proof of the equivalence.

A more interesting variant of the above equivalence is:

$$\begin{aligned}&\textbf{if } (\text{deref } x = \text{deref } y) \textbf{ then } f(x) \textbf{ else } 2 \;\equiv\\ &\qquad\qquad \textbf{if } (\text{deref } x = \text{deref } y) \textbf{ then } f(y) \textbf{ else } 2\end{aligned}$$

where $f : \textbf{var} \to \textbf{exp}$. The difference from the previous example is that we are passing the function procedure $f$ the entire variable ($x$ or $y$) rather than just an expression dereferencing it. So, one might wonder if there is a possibility of $f$ changing the given variable. We argue abstractly, using the results of Theorem 3.7.

$$\begin{aligned}\textsc{Var} \Rightarrow \textsc{Exp} &\rightarrowtail \wp\textsc{Var} \Rightarrow \textsc{Exp}\\ &= \wp(\textsc{Exp} \times (\mathit{Int} \to \textsc{Com})) \Rightarrow \textsc{Exp}\\ &\cong \wp\textsc{Exp} \times \wp(\mathit{Int} \to \textsc{Com})) \Rightarrow \textsc{Exp}\\ &\cong \textsc{Exp} \times (\mathit{Int} \to \wp\textsc{Com}) \Rightarrow \textsc{Exp}\\ &\cong \textsc{Exp} \times (\mathit{Int} \to \mathbf{1}) \Rightarrow \textsc{Exp}\\ &\cong \textsc{Exp} \times \mathbf{1} \Rightarrow \textsc{Exp}\\ &\cong \textsc{Exp} \Rightarrow \textsc{Exp}\end{aligned}$$

For the fourth step, regard $\mathit{Int} \to \textsc{Com}$ as a product $\prod_{i \in \mathit{Int}} \textsc{Com}$ and use an infinitary version of the product isomorphism. The calculation shows that a function procedure that receives a variable argument only has the ability to use its deref operation.

## 5 Handling divergence

For modeling divergence, we use a strict function model similar to that described in [26, Sec. 6]. Define a parametricity graph of Reynolds monoids with divergence, denoted $\mathbf{RM}_\perp$, where

- "state sets" $\mathcal{Q}_X$ are flat cpo's, and
- transformations are complete ordered submonoids of the strict function space $[\mathcal{Q}_X \multimap \mathcal{Q}_X]$, equipped with a $\text{read}_X$ operation.

The functions involved in morphisms are required to be strict and continuous, and the relations are required to be complete (pointed and directed-complete). In effect, this is the construction of $\mathbf{RM}$ duplicated internal to $\mathbf{CPO}_\perp$, the parametric graph of pointed cpo's, strict continuous functions and complete relations, a structure studied in [10, Ch. 7] The semantic category is that of functors $(\mathbf{RM}_\perp)^{\text{op}} \to \mathbf{DCPO}$ that factor through $\mathbf{CPO}_\perp$. It is a result of Oles [29,25] that such a category is cartesian closed.

The passive store $X_0$ of a store $X$ has the same cpo of states as $X$ but, as transformations, all approximations of the do-nothing transformation:

$$\mathcal{T}_{X_0} = \{\, a \mid a \sqsubseteq 1_X \,\}$$

(The intermediate approximations are included by read-closure.) The passivity monomorphism $p_X : X_0 \rightarrowtail X$ involves the obvious injection of the complete ordered

monoid $\mathcal{T}_{X_0} \hookrightarrow \mathcal{T}_X$.

The passivity retraction relation $\xi_X : X \leftrightarrow X$ mentioned in Sec. 3 can be adapted to deal with divergence as follows:

$$(\xi_X)_q = \Delta_{\mathcal{Q}_X}$$
$$(\xi_X)_t = \{\, (a, a') \mid a \sqcap 1_X \sqsupseteq a' \,\}$$

It may be verified that $(\xi_X)_t$ is a monoid relation and $\xi_X$ itself is a reader monoid relation. Lemma 3.4 continues to hold for this relation $\xi_X$ and we can duplicate Theorem 3.6 as follows:

**Theorem 5.1** *If $F$ and $P$ are Algol functors in $\mathcal{S}(\mathbf{RM}_\perp)$ where $P$ is passive, every parametric transformation $t : F \to P$ is uniquely determined by its restriction $t_0 : \wp F \to P$, giving a natural injection:*

$$Par(F,\, P) \rightarrowtail Par(\wp F,\, P)$$

The proof is the same as that of Theorem 3.6. This means that computations of type $F \to P$ are uniquely determined by their restrictions to $\wp F \to P$. Hence, they cannot have side effects, not even temporary ones.

## 6 Related work

The model of Specification Logic, due to Tennent [45], was the first one to model passivity. The passivity aspects were further studied in [12,24]. Tennent's model does not employ relational parametricity, relying on morphisms instead of relations to capture the uniformity conditions. Passivity and other intensional properties are modelled through a form of "what if" modeling. Morphisms in the category of stores include, not only those needed for interpreting the programming language, but additional ones that are used in the logical analysis (including the retractions of passivity monomorphisms). One decides whether a computation is passive by asking the question what would happen to it under a morphism that prohibits all state changes. If it remains the same, then it is regarded as passive; otherwise not. While intuitively appealing, this model has the unfortunate effect of becoming *intensional* (despite working in an extensional framework). Two program terms are equivalent only if they behave identically under all possible state change constraints. For example, the equivalence (1) is not valid in the model, for the reason that the left hand side of the equivalence would be undefined if state changes were constrained to those that preserve the even-ness of $x$, whereas the right hand side would continue to be defined.

The possibility of commands becoming less defined when we move to a future world is prohibited in our model because the transformation components $f_t$ of morphisms are required to be *injective*. This has consequences for the programming language. For instance, if we were to add a "block expression" construct that precludes non-local state changes [46], Tennent's model would allow the execution of the block expression to diverge (or give an error) if it attempts non-local state changes.

In contrast, our model allows block expressions to be constructed only from commands that can be guaranteed not to cause non-local changes, for instance via a type system like that of "SCI Revisited" or "ILC Revisited" [24,44].

O'Hearn and Reynolds [26] provide an account of irreversible state change for the command type and active expressions via a syntactic translation to the polymorphic linear lambda calculus. While the explanation of state change via a linearly used state object is intuitively appealing, it does not have an interpretation for passive expressions. O'Hearn and Reynolds do not provide any treatment of passive expressions in their paper, and it is generally believed that it is not possible to do so in a purely linear setting. See, for example, Wadler [48] where an extension of linear lambda calculus is proposed for modeling "read-only" uses. At a semantic level, O'Hearn and Reynolds use strict functions on pointed cpo's to model state change, as previously recommended by the present author. This modeling eliminates snap back effects at the command type in the presence of divergence. It is adopted here in the same manner. However, our modeling of irreversible state change works even in the absence of divergence and, so, linearity and strictness are not central to it.

As remarked in Introduction, event-based models are able to model passivity with relative ease. However, all such models are intensional and do not satisfy extensional equivalences like (1). The "Passivity and Independence" model of the author [33] was historically the first one where the reflective subcategory structure of passive types was discovered. These ideas were later incorporated in the coherent space model [34] and the games model [1]. These models represent passivity by "fiat." Out of all possible events, certain event are designated as "passive," and the reflective subcategory structure is imposed via an axiom. In other words, these models state what is passive (rather correctly, it turns out), but do not explain what it *means* for a computation to be passive. The criticism that such a treatment lacks *explanatory force*, offered to the author by P. W. O'Hearn, P. Panangaden and others, formed the main driver for further investigation, culminating in the present results.

The Yoneda embedding of the coherent space model in a functor category shown in [25] bears a close intuitive resemblance to the present model. In that work, "object spaces," a form of comonoids of coherent spaces, were used for modelling stores. This was the first instance of sophisticated mathematical objects being used to model stores and provided inspiration for other models such as the one proposed here. Beyond this, it is hard to draw any firm conclusions about a correspondence between the two because the model of [25] is event-based and stateless, whereas representing states is an important objective of the present model.

In recent work, Ahmed, Dreyer and colleagues [2,9] have applied the ideas of possible worlds (similar to functor categories) and automata-theoretic reasoning in the setting of operational reasoning. While the ideas seem intuitively similar, it is difficult to make a formal comparison at the present stage because the starting points of denotational and operational approaches are quite different. Some remarks regarding the comparison may be found in [36]. It is also worth remarking that these researchers have not yet tackled the issues of passivity in their approaches.

In another line of work, Benton et al. [4] have proposed a semantic characterization of effect systems in a global store model using relation-preservation properties. They were led to analyze "observable read-only effects" (i.e., observable passivity) as well as its dual "observable write-only effects," and their characterization turns out to be quite similar to ours, *viz.*, passive computations are those that preserve the identity relations on states. These ideas have been extended to dynamic allocation of stores using Kripke logical relations (similar to our functor category models) in subsequent work [3,47]. The key difference between their work and ours is that they model effect systems, which may be thought of as Curry-style properties of computations, whereas we model type systems in the Church-style using semantic structures. The delicate balance of intensional and extensional effects does not seem to arise in this line of work.

# 7 Conclusion

We have defined a conceptually-based semantic model for imperative programs that captures the notion of "passivity". This is done using a recently developed automata-theoretic denotational framework, where stores are modelled as an abstract form of automata, with explicit representation of states as well as state transitions. Relational parametricity of the type and term interpretations then ensures that the properties of passive expressions are respected.

This approach contrasts with the intensional models such as the event-based and games models [1,34] where passivity is modelled by "fiat," by designating certain events or moves as passive ones. While such models have strong definability and full abstraction properties, they however lack an explanation of what it means for a computation to be passive. In our extensional framework, on the other hand, a computation is passive if it is *independent* of the state transformations that might be possible in the store. We believe this gives a clear answer to the semantic question of what passivity means.

One might wonder if the model presented here is fully abstract. We have not investigated the question in detail and it will perhaps involve considerable work to settle the question because functor categories are quite extensive and not enough is not known about what is definable in them. However, we are able to calculate explicit representation results for simple first order types such as $\textsc{Com} \Rightarrow \textsc{Com}$ and $\textsc{Com} \Rightarrow \textsc{Exp}$, which are accurate. We leave a full exploration of the full abstraction question to future work.

Other questions that this work might enable is a semantic understanding of the various notions of passivity present in specification and verification frameworks, e.g., program specification systems [18], ownership type systems [22] and fractional permission-based methods [6,37]. Secondly, the successful modeling of passivity takes us one step closer to modeling program logics such as Syntactic Control of Interference [24], Specification Logic [39,45] and Separation Logic [37,41]. We envisage that the model presented here will be helpful to streamline the semantic treatment of such programming logics.

## Acknowledgements

We are grateful to anonymous referees whose comments led to various improvements in the paper. Discussions with Claudio Hermida, Neil Ghani and Patricia Johann are warmly acknowledged.

## References

[1] S. Abramsky and G. McCusker. Full abstraction for Idealized Algol with passive expressions. *Theoretical Comput. Sci.*, 227(1-2):3–42, 1999.

[2] A. Ahmed, D. Dreyer, and A. Rossberg. State-dependent representation independence. In *Thirty Sixth Ann. ACM Symp. on Princ. of Program. Lang.* ACM, 2009.

[3] N. Benton, A. Kennedy, L. Beringer, and M. Hofmann. Relational semantics for effect-based program transformations with dynamic allocation. In *9th ACM SIGPLAN Intl. Conf. on Princ. and Practice of Declarative Program.*, pages 87–96. ACM, 2007.

[4] N. Benton, A. Kennedy, M. Hofmann, and L. Beringer. Reading, writing and relations. In N. Kobayashi, editor, *APLAS '06*, pages 114–130. Springer, 2006.

[5] R. Bornat, C. Calcagno, P.W. O'Hearn, and M. Parkinson. Permission accounting in Separation Logic. In *ACM Symp. on Princ. of Program. Lang.*, pages 59–70. ACM Press, 2005.

[6] J. Boyland. Checking interference with fractional permissions. In R. Cousot, editor, *Static Analysis: 10th Intern. Symp.*, volume 2694 of *LNCS*, pages 55–72. Springer, 2003.

[7] S. D. Brookes. A semantics for Concurrent Separation Logic. *Theoretical Comput. Sci.*, 375(1-3):227–270, Apr 2007.

[8] S. D. Brookes, M. Main, A. Melton, and M. Mislove, editors. *Math. Found. of Programm. Semantics: Eleventh Ann. Conference*, volume 1 of *Elect. Notes in Theor. Comput. Sci.* Elsevier, 1995.

[9] D. Dreyer, G. Neis, and L. Birkedal. The impact of higher-order state and control effects on local relational reasoning. In *ICFP*, 2010.

[10] B. P. Dunphy and U. S. Reddy. Parametric limits. In *Proc. 19th Ann. IEEE Symp. on Logic in Comp. Sci.*, pages 242–253. IEEE, July 2004.

[11] S. Eilenberg. *Automata, Languages, and Machines.* Academic Press, 1974. (Volumes A and B).

[12] P. J. Freyd, P. W. O'Hearn, A. J. Power, M. Takeyama, and R. D. Tennent. Bireflectivity. In Brookes et al. [8], pages 199–213.

[13] C. Hermida. Fibrations, logical predicates and indeterminantes. Ph.D. thesis and Technical Report ECS-LFCS-93-277, University of Edinburgh, 1993.

[14] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall International, London, 1985.

[15] W. M. L. Holcombe. *Algebraic Automata Theory.* Cambridge Studies in Advanced Mathematics. Cambridge Univ. Press, Cambridge, 1982.

[16] B. Jacobs. *Categorical Logic and Type Theory*, volume 141 of *Studies in Logic and the Foundations of Mathematics.* Elsevier, 1999.

[17] P. T. Johnstone. *Sketches of an Elephant: A Topos Theory Compendium (two volumes).* Clarendon Press, 2002.

[18] K. R. M. Leino. Dafny: An automatic program verifier for functional correctness. In *LPAR-16*, volume 6355 of *LNCS*, pages 348–370. Springer-Verlag, 2010.

[19] S. Mac Lane and I. Moerdijk. *Sheaves in Geometry and Logic: A First Introduction to Topos Theory.* Springer-Verlag, 1992.

[20] G. McCusker. A fully abstract relational model of syntactic control of interference. In *Computer Science Logic (CSL) 2002*, volume 2471 of *LNCS*, pages 247–261. Springer-Verlag, 2002.

[21] R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[22] J. Noble, J. Vitek, and J. Potter. Flexible alias protection. In E. Jul, editor, *ECOOP'98 - Object-oriented Programming*, volume 1445 of *LNCS*, pages 158–185. Springer-Verlag, 1988.

[23] P. W. O'Hearn. A model for syntactic control of interference. *Math. Struct. Comput. Sci.*, 3:435–465, 1993.

[24] P. W. O'Hearn, A. J. Power, M. Takeyama, and R. D. Tennent. Syntactic control of interference revisited. In Brookes et al. [8], pages 447–486. (Reprinted as Chapter 18 of [28]).

[25] P. W. O'Hearn and U. S. Reddy. Objects, interference and the Yoneda embedding. *Theoretical Computer Science*, 228(1):211–252, 1999.

[26] P. W. O'Hearn and J. C. Reynolds. From Algol to polymorphic linear lambda-calculus. *J. ACM*, 47(1):167–223, Jan 2000.

[27] P. W. O'Hearn and R. D. Tennent. Parametricity and local variables. *J. ACM*, 42(3):658–709, 1995. (Reprinted as Chapter 16 of [28]).

[28] P. W. O'Hearn and R. D. Tennent. *Algol-like Languages (Two volumes)*. Birkhäuser, Boston, 1997.

[29] F. J. Oles. *A Category-Theoretic Approach to the Semantics of Programming Languages*. PhD thesis, Syracuse University, 1982.

[30] F. J. Oles. Functor categories and store shapes. In *Algol-like Languages* [28], chapter 11, pages 3–12.

[31] A. M. Pitts. *Nominal Sets: Names and Symmetry in Computer Science*. Cambridge Univ. Press, 2013.

[32] G. Plotkin and M. Abadi. A logic for parametric polymorphism. In *Typed Lambda Calculi and Applications - TLCA '93*, LNCS, pages 361–375. Springer-Verlag, 1993.

[33] U. S. Reddy. Passivity and independence. In *Proc. Ninth Ann. IEEE Symp. on Logic in Comp. Sci.*, pages 342–352. IEEE, July 1994.

[34] U. S. Reddy. Global state considered unnecessary: An introduction to object-based semantics. *J. Lisp and Symbolic Computation*, 9:7–76, 1996. (Reprinted as Chapter 19 of [28]).

[35] U. S. Reddy and B. P. Dunphy. An automata-theoretic model of objects. In E. Zucca, editor, *2011 Intl. Workshop on Foundations of Object-Oriented Languages*, pages 1–15. electronic proceedings at http://www.disi.unige.it/person/ZuccaE/FOOL2011/, 2011.

[36] U. S. Reddy and B. P. Dunphy. An automata-theoretic model of Idealized Algol. In *Automata, Languages and Programming (ICALP 2012)*, volume 7392 of *LNCS*, pages 337–350. Springer-Verlag, 2012.

[37] U. S. Reddy and J. C. Reynolds. Syntactic control of interference for Separation Logic. In *Thirty Ninth Ann. ACM Symp. on Princ. of Program. Lang.*, pages 323–336. ACM, 2012. (ACM SIGPLAN Notices, 47:1:323-336).

[38] J. C. Reynolds. The essence of Algol. In J. W. de Bakker and J. C. van Vliet, editors, *Algorithmic Languages*, pages 345–372. North-Holland, 1981. (Reprinted as Chapter 3 of [28]).

[39] J. C. Reynolds. Idealized Algol and its specification logic. In D. Neel, editor, *Tools and Notions for Program Construction*, pages 121–161. Cambridge Univ. Press, 1982. (Reprinted as Chapter 6 of [28]).

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

[41] J.C. Reynolds. Separation Logic: A logic for shared mutable data structures. In *LICS*, pages 55–74, 2002.

[42] D. S. Scott and C. Strachey. Towards a mathematical semantics for computer languages. In J. Fox, editor, *Proc. of Symp. on Computers and Automata*, pages 19–46. Polytech Institute of Brooklyn Press, 1971. (original Tech. Report Oxford PRG-6.).

[43] Ian Stark. Categorical models for local names. *Lisp and Symbolic Computation*, 9(1):77–107, February 1996.

[44] V. Swarup, U. S. Reddy, and E. Ireland. Assignments for applicative languages. In *Algol-like Languages* [28], chapter 9, pages 235–272.

[45] R. D. Tennent. Semantical analysis of specification logic. *Inf. Comput.*, 85(2):135–162, 1990. (Reprinted as Chapter 13 of [28]).

[46] R. D. Tennent. Denotational semantics. In S. Abramsky, D. M. Gabbay, and T. S. E Maibaum, editors, *Handbook of Logic in Computer Science*, volume 3, pages 169–322. Oxford University Press, 1994.

[47] J. Thamsborg and L. Birkedal. A Kripke logical relation for effect-based program transformations. *SIGPLAN Not.*, 46(9):445–456, September 2011.

[48] P. Wadler. Linear types can change the world! In M. Broy and C. B. Jones, editors, *Program. Concepts and Methods*. North-Holland, Amsterdam, 1990. (Proc. IFIP TC 2 Working Conf., Sea of Galilee, Israel).

# Appendix

*Definitions*

In this section, we give a brief overview of the framework of reflexive graphs [27, Sec. 7] and parametricity graphs [10].

Formally, we are considering reflexive graph objects in **CAT**, the category of all (small) categories.

Unpacking the definition, we note that a reflexive graph $\mathbf{G}$ consists of two categories $\mathbf{G}_v$ and $\mathbf{G}_e$ (the "vertex" category and the "edge" category, respectively), and three functors between them $\partial_9, \partial_1 : \mathbf{G}_e \to \mathbf{G}_v$ and $I : \mathbf{G}_v \to \mathbf{G}_e$ such that $\partial_i \circ I = \mathrm{Id}_{\mathbf{G}_v}$. The functors $\partial_0$ and $\partial_1$ pick out the "source" and the "target" for the edges and their morphisms, whereas $I$ assigns to each vertex $X$ an "identity" edge $I_X$. The notation $R : X \leftrightarrow X'$ is used to denote the situation that $\partial_0(R) = X$ and $\partial_1(R) = X'$. The definition also generalize to edges of arbitrary arity in place of binary edges.

Reflexive graphs represent a special case of indexed categories. Hence, they form a 2-category with 1-cells being called "RG-functors" and 2-cells being called "parametric natural transformations".

Intuitively, this data means that we use two-dimensional categorical structures, where morphisms occupy one dimension and edges (modelling "relations") between categorical objects occupy the second dimension, as in the diagram below:

$$\begin{array}{ccc} X & \xrightarrow{f} & Y \\ R \uparrow\downarrow & & \uparrow\downarrow S \\ X' & \xrightarrow{f'} & Y' \end{array}$$

A diagram of this form, called a *square*, is the shape of a morphism in $\mathbf{G}_e$ (of type $R \to S$ with its "source" and "target" being $f$ and $f'$). It represents the property that the morphisms $f$ and $f'$ map $R$-related arguments to $S$-related results. The textual notation for the property is $f \left[R \to S\right] f'$.

A reflexive graph is called *relational* if there is at most one edge morphism of any given shape. In that case, the hom-set $\mathbf{G}_e[R, S]$ is a set-theoretic relation between $\mathbf{G}_v[X, Y]$ and $\mathbf{G}_v[X', Y']$.

The reflexive graphs we work with are called *parametricity graphs* [10]. They incorporate additional axioms to capture the idea that relations in the vertical dimension indeed behave like "relations" in the intuitive sense. A parametricity graph is a reflexive graph that (i) is relational (ii) satisfies the *identity condition*: $f \left[I_X \to I_Y\right] f' \Longrightarrow f = f'$ and (iii) has a cloven fibration $\langle \partial_0, \partial_1 \rangle : \mathbf{G}_e \to \mathbf{G}_v \times \mathbf{G}_v$. The last of these conditions, which is an established part of category theory [16], means the following. The right square $f \left[R \to S\right] f'$ in the diagram below is called a *cartesian square* if every square of the form of the outer square uniquely factors

through it:

$$\begin{array}{ccccc} X & \xrightarrow{g} & A & \xrightarrow{f} & B \\ {\scriptstyle P}\uparrow\downarrow & & {\scriptstyle R}\vdots & & \uparrow\downarrow{\scriptstyle S} \\ X & \xrightarrow{g'} & A' & \xrightarrow{f'} & B' \end{array}$$

The reflexive graph is *fibred* if, for all $f$, $f'$ and $S$ of matching types, there is an edge $R$ that fills the dotted arrow making it a cartesian square. The edge $R$ is unique up to isomorphism. A particular choice of such edges $\langle f, f' \rangle^* S = R$ is called a *cleavage* and the fibration is said to be *cloven*. Parametricity graphs are given with a chosen cleavage (even though in most of our examples, the cleavage is unique).

A parametricity graph-functor (PG-functor) is an RG-functor that preserves the chosen cleavage. A 2-cell between such functors (a parametric natural transformation) only needs to satisfy the parametricity condition; naturality follows from parametricity [10]. This is because parametricity graphs have a *subsumption map* $\langle - \rangle$ that sends morphisms $g : X \to X'$ to edges $\langle g \rangle : X \leftrightarrow X'$ with the property that a square of shape on the left below exists iff the square of morphisms on the right commutes:

$$\begin{array}{ccc} X & \xrightarrow{f} & Y \\ {\scriptstyle \langle g \rangle}\uparrow\downarrow & & \uparrow\downarrow{\scriptstyle \langle h \rangle} \\ X' & \xrightarrow{f'} & Y' \end{array} \quad \Longleftrightarrow \quad \begin{array}{ccc} X & \xrightarrow{f} & Y \\ {\scriptstyle g}\downarrow & & \downarrow{\scriptstyle h} \\ X' & \xrightarrow{f'} & Y' \end{array}$$

The subsumption map is given by $\langle g \rangle = \langle g, \mathrm{id}_{X'} \rangle^* I_{X'}$.

Dually, *co-cartesian squares* are of the form of the left inner square in the diagram:

$$\begin{array}{ccccc} A & \xrightarrow{f} & B & \xrightarrow{g} & X \\ {\scriptstyle R}\uparrow\downarrow & & {\scriptstyle S}\vdots & & \uparrow\downarrow{\scriptstyle T} \\ A & \xrightarrow{f'} & B' & \xrightarrow{g'} & X' \end{array}$$

so that all outer squares factor through them. An RG-functor is *cofibred* if it maps all co-cartesian squares that exist in its source graph to co-cartesian squares in the target graph. We make use of PG-functors that are cofibred. However, we do not require that the source graph itself should be cofibred, i.e., not all $R$, $f$ and $f'$ are required to have corresponding $S$ relations.