Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 303 (2014) 149–180

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Logical Relations and Parametricity - A Reynolds Programme for Category Theory and Programming Languages

Claudio Hermida[1] Uday S. Reddy[2]

University of Birmingham

Edmund P. Robinson[3]

Queen Mary, University of London

Dedicated to the memory of John C. Reynolds, 1935-2013

Abstract

In his seminal paper on "Types, Abstraction and Parametric Polymorphism," John Reynolds called for homomorphisms to be generalized from functions to relations. He reasoned that such a generalization would allow type-based "abstraction" (representation independence, information hiding, naturality or parametricity) to be captured in a mathematical theory, while accounting for higher-order types. However, after 30 years of research, we do not yet know fully how to do such a generalization. In this article, we explain the problems in doing so, summarize the work carried out so far, and call for a renewed attempt at addressing the problem.

Keywords: Universal algebra, Category Theory, Homomorphisms, Logical Relations, Natural Transformations, Parametric polymorphism, Relational Parametricity, Data abstraction, Information hiding, Definability, Reflexive Graphs, Fibrations, Relation lifting.

1 Introduction

Starting with the pioneering work of Emmy Noether, Emil Artin and van der Waerden in the 1930's [51], homomorphisms have firmly established themselves as the foundation for modern algebra. In due course, they led to the formulation of category theory [10], whose central concept is that of "natural transformation," embodying the principle of uniformity with respect to homomorphisms.

[1] Email: claudio.hermida@gmail.com

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

[3] Email: e.p.robinson@qmul.ac.uk

http://dx.doi.org/10.1016/j.entcs.2014.02.008

1571-0661/© 2014 Elsevier B.V. All rights reserved.

John Reynolds's pioneering work on logical relations and relational parametricity [42] (with support from the earlier insights of Christopher Strachey, Dana Scott and Gordon Plotkin) casts doubt on this central place accorded to homomorphisms, and raises new questions. By a "Reynolds programme for category theory and programming languages," we mean a programme to answer these questions and to carry out a thorough investigation of the relative place of homomorphisms and logical relations in the broader mathematical thought.

Let us first note the striking similarities in the pre-theoretic intuitions expressed for natural transformations and parametric polymorphism:

> This exhibition of the isomorphism $L \cong T(T(L))$ is "natural" in that it is given *simultaneously* for *all* finite-dimensional vector spaces L.
>
> — Eilenberg and Mac Lane [10, p. 232]

> Commutativity conditions like (49) ... play an important role in describing why a morphism ω_A, defined for all modules A, is "natural," that is, independent of artificial choices.
>
> — Mac Lane and Birkhoff [26, p. 192]

> In *ad hoc* polymorphism there is no single systematic way of determining the type of the result from the type of the arguments.... Parametric polymorphism is more regular and may be illustrated by an example. ... We would like *Map* to work on all types of list, ... so that *Map* would have to be polymorphic.
>
> — Christopher Strachey [49, Sec. 3.6.4]

> Intuitively, a parametric polymorphic function is one that behaves the same way for all types, while an *ad hoc* polymorphic function may have unrelated meanings for different types.
>
> — John Reynolds [42, Sec. 7]

Interpreted in a suitable way, the pre-theoretic intuitions expressed by the pioneers of category theory and those of programming language theory match up essentially word for word: "*artificial choices*" corresponds to "ad hoc *polymorphism*;" "*given simultaneously*" corresponds "*work on all types*" and "*behave the same way*;" and "*natural*" corresponds to "*parametric polymorphism*." So, one might expect to have a single mathematical theory that captures the intuitions expressed in both the contexts. Unfortunately, there is no such a theory, yet. The stumbling block is the choice of the "raw material" used for defining uniformity. In category theory, it is homomorphisms. In programming language theory, it is logical relations.

In the programming language context, one expects to use a type expression such as $[X \to X]$ (or X^X) just like any other. However, such a type expression is not a functor. Given a morphism $f : X \to Y$, there is no canonical way to extend it to a morphism of type $[X \to X] \to [Y \to Y]$. Famously, the "$\to$" constructor is *contravariant* in its first argument and *covariant* in the second. If the same type variable occurs in both the positions, the resulting type expression fails to be either covariant or contravariant. The problem was very likely known to Eilenberg and Mac Lane, who carry around both covariant and contravariant type variables separately in their treatment [10]. Later, *dinatural transformations* (short for "diagonally natural" transformations) were advanced [25, Sec. IX.4] to address the situation of the same type variable occurring in both positions. However, dinatural transformations do not compose. So, they do not provide a general solution. (See Scott [47] for a review of known results.)

Logical relations arose early in the study of typed lambda calculus (or intuition-

istic proof theory). See Statman [48] and the references cited there. The special case of logical partial equivalence relations appeared even earlier in the work of Turing and Gandy in characterizing "virtual types." The essential purpose of logical relations in this context is to characterize properties satisfied by lambda terms, or functions computed by lambda terms, so as to be consistent with the implicit operations of the typed lambda calculus. When sufficiently abstracted, the essence of the logical relations proof method can be squeezed out as the idea that the "$\rightarrow$" type constructor is a *relator*. Given relations $R \subseteq X \times X'$ and $S \subseteq Y \times Y'$, there is a corresponding lifting $[R \rightarrow S] \subseteq [X \rightarrow Y] \times [X' \rightarrow Y']$ for function types that is consistent with operations of the typed lambda calculus, *viz.*,

$$[R \rightarrow S] = \{\, (f, f') \mid \forall x, x'.\, (x, x') \in R \implies (f(x),\, f'(x')) \in S \,\}$$

Note that this form of a definition works for relations of any arity, not only binary relations. In his seminal paper on lambda-definability [38], Plotkin demonstrates that all functions definable in typed lambda calculus are invariant under all such "logical" relations, and, moreover, functions invariant under a Kripke variant of logical relations are precisely the lambda-definable ones.

A second, independent discovery of logical relations occurred in automata theory [14,15]. We envision automata as having internal sets of states that are hidden from the environment, while the observable behaviour is stated in terms of the inputs and outputs of the automata. It was noticed in this context that homomorphisms between automata were inadequate to capture the equivalence of behaviour. A form of structure-preserving relations was necessary. Such relations were called "*weak homomorphisms*," "*covering relations*" [9] or "*simulation relations*" [27]. In due course, they developed into a mathematical theory of "information hiding" or "data abstraction" in computer science, which is the underpinning conceptual framework for the practice of object-oriented programming.

Reynolds integrated these two strands of thought and formulated a general principle of relational parametricity that is applicable to a wide range of contexts for capturing the notion of "information hiding" or "abstraction." Unfortunately, we believe that the magnitude of this achievement has not been sufficiently recognized. As Reynolds attempted to give a set-theoretic semantics for impredicative polymorphic lambda calculus using parametricity, which was found to be impossible upon subsequent investigation [43], the popular image that has developed in subsequent years has tied parametricity to polymorphic lambda calculus. The fact that parametricity has broad applications throughout mathematics has gone unnoticed. In fact, parametricity counters and challenges the foundations of 20th century mathematics for its reliance on homomorphisms. (See, for example, Freyd [12].) This challenge has not yet been answered.

In this article, we attempt to re-explain relational parametricity from the first principles, giving an indication of its broad applicability to mathematical concepts. We make no mention of any "calculus" and pay no attention to impredicative type systems. While these applications are interesting and important, they are not the main point of parametricity in our view. Rather, parametricity re-invokes the same

intuitions that led to the notion of natural transformations and the definition of categories in 1940's and provides a different, novel answer. How to incorporate this answer into the prevailing categorical foundations of mathematics is a question that interests us. Through this exposition, we hope to spur further interest in this question to carry forward Reynolds's legacy.

Our presentation is targeted at readership from both Mathematics and Computer Science. The aim is to explain the issues involved in addressing Reynolds's challenge, but we do not attempt to survey the entire literature on the subject. Other expository treatments of relational parametricity, by O'Hearn [11] and Scott [47] may be perused for fuller surveys of the literature as well as alternative view points.

2 Logical Relations

The way out of this impasse is to generalize homomorphisms from functions to relations. — *Reynolds [42, Sec. 1]*

A "logical relation" for a particular form of a mathematical structure is a structure-preserving relation just as a "homomorphism" is a structure-preserving function. Such relations are also often called "compatible relations" in algebraic literature because structure-preservation for relations is thought of as "compatibility" with structure. In this article, we will treat "structure preservation" and "compatibility with structure" interchangeably.

Example 2.1 A *group* is an algebraic structure involving a set along with a binary associative "multiplication" operation "$\cdot$", a unit 1 for multiplication and a multiplicative inverse x^{-1} for each element x. We write the underlying set of a group A as $|A|$. The three operations of groups then have the types $\cdot : |A| \times |A| \to |A|$, $1 : |A|$ and $(\,)^{-1} : |A| \to |A|$.

A *(binary) logical relation of groups* $R : A \leftrightarrow A'$ between groups[4] A and A' is a binary relation $R \subseteq |A| \times |A'|$ such that:

$$\begin{array}{l} x\ [R]\ x' \ \wedge\ y\ [R]\ y' \implies xy\ [R]\ x'y' \\ 1_A\ [R]\ 1_{A'} \\ x\ [R]\ x' \implies x^{-1}\ [R]\ (x')^{-1} \end{array} \tag{1}$$

Using the relation operators that are introduced later in this section, these formulas amount to saying the operations of the two groups A and A' are related as follows: $\cdot\ [R \times R \to R]\ \cdot$, $1_A\ [R]\ 1_{A'}$ and $(\,)^{-1}\ [R \to R]\ (\,)^{-1}$. A logical relation

With this definition, easy calculations give the following examples:

- The equality relation $I_A \subseteq |A| \times |A|$ is a logical relation (the "identity" logical relation $I_A : A \leftrightarrow A$).

[4] Even though we focus on binary relations for the ease of exposition, all the concepts of logical relations discussed in this paper, except for that of ordered structures, generalize to relations of arbitrary arity. Note that $A \leftrightarrow A'$ and $A' \leftrightarrow A$ are different types of logical relations. There is no implicit symmetry assumed.

- An equivalence relation $\sim$ on $|A|$ is a logical relation of groups if and only if it is a congruence relation of groups.
- If $h : |A| \to |B|$ is a function between the underlying sets of groups A and B, then the graph of h, denoted $\langle h \rangle \subseteq |A| \times |B|$, is a logical relation if and only if h is a group homomorphism.
- $R \subseteq |A| \times |B|$ is a logical relation if and only if it forms a subgroup of the product group $A \times B$.

A standard early result in group theory is that a function between groups $h : A \to B$ is a group homomorphism if and only if it is a monoid homomorphism (preservation of inverses comes for free). This is not the case with logical relations.

This example generalizes to Universal Algebra.

Example 2.2 In classical Universal Algebra, algebras are defined with respect to a *signature*, which consists of a set of names, each equipped with an *arity*, n, representing operations. An *algebra* A is a carrier set $|A|$ equipped with interpretations of the operations. If ω is a member of the signature with arity n, then the interpretation of ω is a function $f_\omega : |A|^n \to |A|$. A relation $R \subseteq |A| \times |A|$ is *compatible with operation* ω if and only if $\forall 1 \leq i \leq n.\ a_i[R]a'_i \Rightarrow f_\omega(a_1, \ldots a_n)[R]f_\omega(a'_1, \ldots a'_n)$. The notion of logical relation is a generalization of this concept for relations between possibly different algebras. If A and A' are algebras for Ω then a *logical relation* $R : A \leftrightarrow A'$ is a relation $R \subseteq |A| \times |A'|$ that is "compatible" with all the operations ω in the sense that $\forall 1 \leq i \leq n.\ a_i[R]a'_i \Rightarrow f_\omega(a_1, \ldots a_n)[R]f'_\omega(a'_1, \ldots a'_n)$, where f_ω and $f"_\omega$ are the interpretations of ω in the algebras A and A'. The properties listed of logical relations for groups (identity, congruence, homomorphism and characterization as sub-algebra of the direct product) extend to this setting.

Example 2.3 In that part of Universal Algebra dealing with the lattice of algebraic theories, a key tool is the Galois connection between sets of operations and sets of relations compatible with them, originally developed independently by Geiger [13] and by Bodnarchuk et al. [4] and subsequently refined and extended by many authors (e.g. Pöschel [40]), in which the closed sets of operations are precisely the clones. The core notion of compatibility coincides with ours. A relation $R \subseteq A \times A$ is compatible with a set of operations (e.g. a clone) if and only if it is compatible with each operation in the set. The standard universal algebraic theory differs from that presented here in considering families of relations in order to establish the Galois connection.

In contrast to the theories above, we are motivated by examples of structures arising as parts of computer programs, as well as in conventional mathematics. In this context operations can take values of arbitrary types in the language as parameters and produce values of arbitrary types as results.

Let now a *signature* be defined in terms of:

- a finite set of "sorts" $\vec{a} = a_1, \ldots, a_n$ (replacing the single carrier A), and
- a suite of operation symbols $\Omega = \{\omega_k : F_k(a_1, \ldots, a_n)\}_{k \in I}$,

where each F_k is a type expression built from (i) the sorts $a_1, \ldots, a_n$, (ii) "known types" such as boolean or integer, and (iii) type operators such as $\times$, $+$ and $\to$. Given a signature $\langle \vec{a}, \Omega \rangle$, an *algebra* $\mathcal{A}$ for the signature consists of

- an assignment $A_{a_1}, \ldots, A_{a_n}$ of sets for each sort, and
- an assignment A_{ω_k} of elements, for each $\omega_k \in \Omega$, in the corresponding sets $F_k(A_{a_1}, \ldots, A_{a_n})$.

This kind of algebra can be seen in e.g. [46], except that we allow known types to appear directly in expressions, rather than regarding them as special sorts. We also assume that we have function types, and therefore can equate an operation with the corresponding value of function type. This allows us to deal only with constants (many of which represent functions) and not operations.

From the type-theoretic point of view, the "sorts" are type variables and "known types" are constant types. The entire signature $\langle \vec{a}, \Omega \rangle$ is then of a higher-order "sum type"

$$\Sigma_{a_1,\ldots,a_n} \left(\Pi_{k \in I} \, F_k(a_1, \ldots, a_n) \right) \tag{2}$$

where each of the type variables $a_1, \ldots, a_n$ ranges over all sets. For example, the type of groups is $\Sigma_a \, [a \times a \to a] \times a \times [a \to a]$. In this type-theoretic point of view, an algebra is just an element of the type (2).

Given two algebras $\mathcal{A}$ and $\mathcal{A}'$ for a signature $\langle \vec{a}, \Omega \rangle$, a *logical relation* $R : \mathcal{A} \leftrightarrow \mathcal{A}'$ is a family of binary relations $R_{a_1} \subseteq A_{a_1} \times A'_{a_1}$, $\ldots$, $R_{a_n} \subseteq A_{a_n} \times A'_{a_n}$ such that, for each operation symbol $\omega_k : F_k(a_1, \ldots, a_n)$ in Ω, its interpretations A_{ω_k} and A'_{ω_k} are related by $F_k(R_1, \ldots, R_n)$. This definition is fashioned after Mitchell's treatment of logical relations for applicative structures [28,29], but specialized to the "set-theoretic type frame."

Basic Type operators

To complete the picture, we must give interpretations of type operators F not only as operations on sets, but also as operations on binary relations between sets. We refer to the latter as the "relation action" of F. For each n-ary type operator F:

- $F(A_1, \ldots, A_n)$ must be a set (for given sets $A_1, \ldots, A_n$), and
- if $R_1 \subseteq A_1 \times A'_1$, $\ldots$, $R_n \subseteq A_n \times A'_n$ are binary relations between sets, then $F(R_1, \ldots, R_n) \subseteq F(A_1, \ldots, A_n) \times F(A'_1, \ldots, A'_n)$ must be a binary relation.

This interpretation is subject to the "identity extension" property:

$$F(I_{A_1}, \ldots, I_{A_n}) = I_{F(A_1,\ldots,A_n)} \tag{3}$$

where $I_X \subseteq X \times X$ is the identity relation for each set X. This is the part of the theory of functors that deals with identities. By omitting requirements involving composition, Reynolds was able to admit more type operators than categorical functors can accommodate, chief among them the function type constructor $\to$. We will use the framework of reflexive graphs in Secs. 3 and 6 to formalize these intuitions at the level of categories.

We begin with product, sum, function space, powerset and predicate types. In the following, type-forming operations are applied to sequences $A, B, C, \ldots$ and relations are between A and A', B and B' ($R \subseteq A \times A'$, $S \subseteq B \times B'$, etc).

(i) Product: as usual, the cartesian product of two sets A and B is the set of ordered pairs. The relation $R \times S \subseteq (A \times B) \times (A' \times B')$ is defined by: $(a,b)[R \times S](a',b') \iff a[R]a' \wedge b[S]b'$.

(ii) Sum: we define the sum of two sets as the disjoint union of copies: $A + B = \{0\} \times A + \{1\} \times B$. The relation $R + S \subseteq (A + B) \times (A' + B')$ is defined by: $(i,x)[R + S](j,y) \iff (i = j = 0 \wedge x[R]y) \vee (i = j = 1 \wedge x[S]y)$.

(iii) Function space: Let $[A \to B]$ be the set of (total) functions $f : A \to B$. We define the relation $[R \to S] \subseteq (A \to B) \times (A' \to B')$ by $f[R \to S]f' \iff \forall a, a'.\, a[R]a' \Rightarrow f(a)[S]f'(a')$.

(iv) Powerset: if $\mathcal{P}A$ is the set of all subsets of A, then we define the extension of the powerset operator to relations by $u[\mathcal{P}R]u' \iff (\forall a \in u.\, \exists a' \in u'.\, a[R]a') \wedge (\forall a' \in u'.\, \exists a \in u.\, a[R]a')$. This definition corresponds to the Egli-Milner ordering for powerdomains [39].

(v) Predicates: The collection of predicates over a set A, which we denote by $\widehat{\mathcal{P}}A$, also has the set of all subsets of A as carrier. However, the relation action is defined by $u[\widehat{\mathcal{P}}A]u' \iff (\forall a, a'.\, a[R]a' \Rightarrow (a \in u \iff a' \in u'))$. Note that the relation action of $\widehat{\mathcal{P}}A$ corresponds to that of $[A \to \mathbf{2}]$.

Lemma 2.4 *The above definitions satisfy the identity extension property.*

(i) $I_A \times I_B = I_{A \times B}$

(ii) $I_A + I_B = I_{A+B}$

(iii) $[I_A \to I_B] = I_{[A \to B]}$

(iv) $\mathcal{P}I_A = I_{\mathcal{P}A}$

(v) $\widehat{\mathcal{P}}I_A = I_{\widehat{\mathcal{P}}A}$

The left to right inclusion parts of these equations, e.g., $[I_A \to I_B] \subseteq I_{[A \to B]}$, amount to extensionality of higher type values. The right to left inclusions, e.g., $I_{[A \to B]} \subseteq [I_A \to I_B]$, say that the relation actions are consistent with the observable information of higher type values.

Nondeterministic functions. Operations in both mathematics and computer science are often partial (not defined for all inputs) and sometimes nondeterministic (different executions may produce different results). We therefore want to handle the type of "nondeterministic functions". We write $x \xrightarrow{f} y$ to mean that y is a possible result of applying the nondeterministic function f to input x, and the type of nondeterministic functions as $A \rightsquigarrow B$. It is convenient to reduce $[A \rightsquigarrow B]$ to $[A \to \mathcal{P}B]$. The derived relation $[R \rightsquigarrow S]$ is then $[R \to \mathcal{P}S]$. Expressed directly in

terms of nondeterministic functions, it says:

$$f\ \left[R \rightsquigarrow S\right]\ f' \iff \forall x, x'.\ x\ \left[R\right]\ x' \implies \begin{pmatrix} (\forall y.\ x \xrightarrow{f} y \implies \exists y'.\ x' \xrightarrow{f'} y' \wedge y\ \left[S\right]\ y') \wedge \\ (\forall y'.\ x' \xrightarrow{f'} y' \implies \exists y.\ x \xrightarrow{f} y \wedge y\ \left[S\right]\ y') \end{pmatrix} \tag{4}$$

Coinductive logical relations defined using this notion of $[R \rightsquigarrow S]$ often go by the name of *bisimulation* relations. (See Sangiorgi [45] for a historical account.)

Note that this definition is different from the one we would obtain by interpreting $A \rightsquigarrow B$ as $\mathcal{P}(\mathcal{A} \times \mathcal{B})$.

Partial function space. For sets A and B, their partial function space $[A \rightharpoonup B]$ consists of partial functions from A to B. It is convenient to treat partial functions $A \rightharpoonup B$ as total functions $A \to \mathcal{P}^1 B$, where $\mathcal{P}^1 B$ is the restriction of $\mathcal{P}B$ to subsets of cardinality at most 1. Whenever $f : A \rightharpoonup B$ is undefined for an element $x \in A$, its representative $f' : A \to \mathcal{P}^1 B$ sends x to $\emptyset$. Thus we identify $[A \rightharpoonup B] = [A \to \mathcal{P}^1 B]$. Given relations $R \subseteq A \times A'$ and $S \subseteq B \times B'$, the relation $[R \rightharpoonup S]$ is simply $[R \to \mathcal{P}^1 S]$. Spelling out the detail, this means that

$$f\ [R \rightharpoonup S]\ f' \iff \forall x, x'.\ x\ [R]\ x' \implies \begin{pmatrix} f(x) = \emptyset \wedge f'(x') = \emptyset \vee \\ f(x) \neq \emptyset \wedge f'(x') \neq \emptyset \wedge f(x)\ [S]\ f'(x') \end{pmatrix} \tag{5}$$

Example 2.5 A *field* is a set F equipped with a commutative group structure $(+, 0, -)$ and a partial commutative group structure $(\cdot, 1, (\ )^{-1})$ such that $\cdot$ distributes over $+$. The multiplicative group structure is "partial" in that the inverse is defined for only non-zero elements: $x^{-1} \neq \emptyset \iff x \neq 0$. A *logical relation* of fields $R : F \leftrightarrow F'$ thus requires the two inverse operations to be related by the relation $R \rightharpoonup R$, *i.e.*,

$$\forall x, x'.\ x\ \left[R\right]\ x' \implies (x = 0 \wedge x' = 0) \vee (x \neq 0 \wedge x' \neq 0 \wedge x^{-1}\ \left[R\right]\ (x')^{-1})$$

Thus a logical relation of fields can relate 0 to *only* 0. Since 0 is the unit of the additive group structure, this has the consequence that a logical relation of fields is always a *partial bijection*. A homomorphism of fields, regarded as a total and single-valued logical relation, is therefore necessarily *injective*. This is a well-known fact in field theory, but logical relations provide an abstract reason for why it is so.

It is also worth noting that Reynolds's leading example of type abstraction in [42] involves Bessel's and Decartes's representations for the field of complex numbers. The logical relation involved there is indeed a partial bijection.

Further examples

Algebraic structures such as monoids, semigroups, rings, semirings, etc. can be treated in the same way as Examples 2.1 and 2.5. They are one-sorted structures

involving no "known types." Next we look at *actions*, which bring out connections with modules and vector spaces on the one hand, and algebraic automata theory on the other. Automata theory happens to be one of the first areas in Computer Science where logical relations were discovered.

A *monoid* is an algebraic structure M involving a set along with an associative binary operation "$\cdot$" and its unit 1. A logical relation of monoids $R : M \leftrightarrow M'$ is a relation between the underlying sets such that the two multiplication operations are related by $R \times R \to R$ and the two units by R.

An *action* of monoid M on a set X, also called a *module* for M, is a monoid homomorphism $\alpha : M \to [X \to X]$, where $[X \to X]$ is the collection of endomorphisms on X viewed as a monoid under composition. It is conventional to write $\alpha(m)(x)$ as $m \cdot x$, treating it as a form of "scalar multiplication" of type $M \times X \to X$. We use the notation ${}_M X$ to talk about the module as a structure. A *logical relation* of M-actions $R : {}_M X \leftrightarrow {}_M X'$ is a relation $R \subseteq X \times X'$ compatible with scalar multiplication:

$$x \ [R] \ x' \implies m \cdot x \ [R] \ m \cdot x' \tag{6}$$

If $R = \langle h \rangle$ is the graph of a function then h is a *homomorphism* of M-actions. Note that, with reference to our universal algebraic description, an M-action is a structure involving a "known type" M and a single sort a standing for the underlying set X, *i.e.*, has the type $\Sigma_a \, M \to [a \to a]$. This leads to the requirement that the action maps must be related by $I_M \to [R \to R]$ as indicated in (6). Actions of rings or semirings (modules) and those of fields (vector spaces) can be treated in a similar way.

3 Reflexive graph framework for logical relations

Even before we look at parametricity, it is worth taking an abstract view of the structure underlying logical relations. We need a treatment similar to the definition of categories, retaining identities but dropping composition. The resulting structure is called a reflexive graph [32,35,44], improving on the previous work on scones [31].

Definition 3.1 A *reflexive graph* $\mathcal{G}$ involves a collection $\mathcal{G}_v$ of "vertices" and a collection $\mathcal{G}_e$ of "edges" or "abstract relations," along with functions $\partial_0, \partial_1 : \mathcal{G}_e \to \mathcal{G}_v$ and $I : \mathcal{G}_v \to \mathcal{G}_e$ that satisfy $\partial_0(I(A)) = \partial_1(I(A)) = A$. The functions ∂_0 and ∂_1 pick out the "source" and "target" of abstract relations and I gives a distinguished "identity" relation for each vertex. We write $R : A \leftrightarrow A'$ whenever A and A' are the source and target of an abstract relation R, and abbreviate $I(A)$ to I_A.

A *morphism* of reflexive graphs $F : \mathcal{G} \to \mathcal{H}$, which we call a *relator*, is a pair of functions $F = (F_v, F_e)$ mapping the vertices and edges respectively, preserving all the structure on the nose: $\partial_i(F_e(R)) = F_v(\partial_i(R))$ and $F_e(I_A) = I_{F_v(A)}$. We normally drop subscripts of F_v and F_e using the context to disambiguate which is meant.

Fact 3.2 *Reflexive graphs and relators form a category* **RG**. *It has pointwise prod-*

ucts, i.e., $(\mathcal{G} \times \mathcal{H})_v = \mathcal{G}_v \times \mathcal{H}_v$ *and* $(\mathcal{G} \times \mathcal{H})_e = \mathcal{G}_e \times \mathcal{H}_e$.

We borrow the term *relator* from [1,31]. Note that the property $F(I_A) = I_{F(A)}$ of relators is precisely the identity extension property mentioned in (3). It is a fundamental axiom underlying Reynolds's theory of logical relations and parametricity.

The prototypical example of reflexive graphs is **set**, which has all sets as its vertices and binary relations $R \subseteq A \times A'$ as edges. The identity edge $I_A : A \leftrightarrow A$ is the equality relation $=_A$. All the "type operators" mentioned in Sec. 2 are relators:

$$\mathcal{P}, \mathcal{P}^1, \widehat{\mathcal{P}} : \mathbf{set} \to \mathbf{set}$$
$$\times, +, \to, \rightharpoonup, \rightsquigarrow : \mathbf{set} \times \mathbf{set} \to \mathbf{set}$$

That means that they have an action on sets, such as $\mathcal{P}A$ or $[A \to B]$, as well as an action on relations, such as $\mathcal{P}R$ and $[R \to S]$. The type expressions $F_k(\vec{a})$ in the definition of algebraic structures are built from such relators as well as the *constant relators* for "known types" $Const_K : \mathbf{set}^n \to \mathbf{set}$ given by $Const_K(\vec{A}) = K$ and $Const_K(\vec{R}) = I_K$.

Algebras for signatures $\langle \vec{a}, \Omega \rangle$, along with their logical relations, give rise to reflexive graphs in turn, denoted $\mathbf{alg}(\vec{a}, \Omega)$.

More generally, for any relator $F : \mathcal{G} \to \mathbf{set}$, an *F-algebra* is a pair $A = \langle A, f \rangle$ of a vertex A of $\mathcal{G}$ and an element $f \in F(A)$. A logical relation of such algebras $R : A \leftrightarrow A'$ is an abstract relation R in $\mathcal{G}$ such that $f\ [F(R)]\ f'$ in **set**. This gives a reflexive graph $\mathbf{alg}(F)$. Note that $\mathbf{alg}(\vec{a}, \Omega)$ is a special case of this where we use $\mathcal{G} = \mathbf{set}^n$ (with $\vec{a}$ ranging over its vertices) and Ω is treated as a relator $\mathbf{set}^n \to \mathbf{set}$.

Reflexive graphs from categories

The example of **set** can be generalized. For any category $\mathcal{C}$ with finite products, define $\mathbf{rel}(\mathcal{C})$ as the reflexive graph whose vertices are just the objects of $\mathcal{C}$ and edges $R : A \leftrightarrow A'$ are subobjects $R \rightarrowtail A \times A'$.[5] The diagonal morphisms $\delta_A : A \rightarrowtail A \times A$ serve as the identity edges. Note that the reflexive graph **set** is nothing but $\mathbf{rel}(\mathbf{Set})$. An algebraic example is $\mathbf{ab} = \mathbf{rel}(\mathbf{Ab})$, the reflexive graph of abelian groups. An edge $R \subseteq A \times A'$ is a subgroup of the categorical product (direct sum) $A \times A'$, the same as the "logical relations of groups" mentioned in Example 2.1. Examples of relators on **ab** include $\otimes, \oplus, \to : \mathbf{ab} \times \mathbf{ab} \to \mathbf{ab}$.

Similarly $\mathbf{poset} = \mathbf{rel}(\mathbf{Poset})$, $\mathbf{dcpo} = \mathbf{rel}(\mathbf{DCPO})$ and $\mathbf{cpo}_\perp = \mathbf{rel}(\mathbf{CPO}_\perp)$ give examples from programming language semantics. In **dcpo**, the relation edges $R \subseteq A \times A'$ are directed-complete subsets of $A \times A'$ which are also called "directed-complete relations." In $\mathbf{cpo}_\perp$ the relation edges are pointed, directed-complete relations, dubbed "*complete relations*" by Reynolds. See Pitts [35] for a treatment of these examples.

If $\mathcal{C}$ is a concrete category with a generating object c_0 and $\mathcal{C}$ has finite products, then evidently we can form algebras for signatures $\langle \vec{a}, \Omega \rangle$ *internal* to $\mathcal{C}$. The sorts are interpreted as objects of $\mathcal{C}$ and operations $\omega : F(\vec{a})$ as "points" $A_\omega : c_o \to F(\vec{A})$.

[5] If a category does not have products, one can use jointly monic spans of $\mathcal{C}$ for the same purpose.

Using relations from $\mathbf{rel}(\mathcal{C})$, we can define logical relations for such algebras. Thus, we obtain a reflexive graph $\mathbf{alg}_{\mathcal{C}}(\vec{a}, \Omega)$ of algebras internal to $\mathcal{C}$. Examples of such algebras abound. For example, monoids and monoid actions internal to **Ab** are well-known as rings and modules. Monoids and monoid actions internal to **Poset** are called "pomonoids" (short for partially ordered monoids) and "M-posets."

Reflexive graphs of the form $\mathbf{rel}(\mathcal{C})$ and $\mathbf{alg}_{\mathcal{C}}(\vec{a}, \Omega)$ have additional categorical structure which we discuss in Sec. 6.

Ordered structures

When we deal with ordered structures, we have the option of using the partial order of the structure as the "identity edge" as noted by Reynolds [42, Sec. 5-6]. For example, the reflexive graph $\mathbf{poset}_{\sqsupseteq}$ is similar to **poset** except that the identity edge $I_A : A \leftrightarrow A$ is the partial order $\sqsupseteq_A$. We still have relators such as $\times$ and $\to$ (product and monotone function space) on $\mathbf{poset}_{\sqsupseteq}$ because they preserve the new "identity edges": $\sqsupseteq_A \times \sqsupseteq_B = \sqsupseteq_{A\times B}$ and $[\sqsupseteq_A \to \sqsupseteq_B] = \sqsupseteq_{[A\to B]}$. We say that a relation R in $\mathbf{poset}_{\sqsupseteq}$ represents the "graph" of a monotone function $f : A \to B$ if $x \ [R] \ y \iff f(x) \sqsupseteq_B y$. We write this relation as $\langle f \rangle_{\sqsupseteq}$ rather than $\langle f \rangle$. For algebras in $\mathbf{poset}_{\sqsupseteq}$, homomorphisms derived from logical relations will be "lax homomorphisms." Dually, those in $\mathbf{poset}_{\sqsubseteq}$ will be "oplax homomorphisms."

Reynolds's concept of the "identity relations" is thus an abstract concept.

4 Abstract types and Information hiding

One of the first instances of logical relations being formulated in computer science was in automata theory [15]. (See also [9,14].) They were called "weak homomorphisms" or "covering relations" in this context, and directly inspired the use of logical relations in programming theory (Milner's simulation relations [27] and Hoare's representation functions [20]), leading to the notion of "abstract types" extensively developed in succeeding years. (See, for example [5,46] for book length treatments.) Reynolds's own insights came partly from these developments [41, Chapter 5], allowing him to link logical relations with "abstraction." We also refer to this idea of "abstraction" as "information hiding," using a term initiated by Parnas [34] and used widely in software development, which captures the more general phenomenon at play.

A *monoid semiautomaton*, also called a *deterministic labelled transition system*, is the action of a monoid on a set in terms of partial functions, $\alpha : M \to [X \rightharpoonup X]$. Note that $[X \rightharpoonup X]$, the collection of partial functions from X to itself, forms a monoid under partial function composition and α is a monoid homomorphism. As usual, we use the "scalar multiplication" notation $m \cdot x$ for $\alpha(m)(x)$. The set X is thought of as the set of *states* for the automaton and the monoid M represents the collection of *operations* or "*inputs*." (An ordinary semiautomaton is obtained by specializing M to the free monoid Σ^* generated by an alphabet of symbols Σ.)

A *monoid automaton* is a semiautomaton equipped with a distinguished element called the *start state*, giving a structure $\langle X, \alpha, x_0 \rangle$. As usual, a logical

relation of automata $R : \langle X, \alpha, x_0\rangle \leftrightarrow \langle X', \alpha', x_0'\rangle$ is a relation $R \subseteq X \times X'$ such that $\alpha \left[I_M \to [R \rightharpoonup R]\right] \alpha'$ and $x_0 \left[R\right] x_0'$. A homomorphism of automata $h : \langle X, \alpha, x_0\rangle \to \langle Y, \beta, y_o\rangle$ is a function $X \to Y$ such that $h(m \cdot x) \simeq m \cdot h(x)$ and $h(x_0) = y_0$. (Here, "$\simeq$" denotes the so-called "Kleene equality:" either both the sides are undefined or both are defined and equal.) Note that the graph of a homomorphism $\langle h\rangle$ is a logical relation.

The consideration of automata and semiautomata brought the phenomenon of "information hiding" to the realm of algebra, perhaps for the first time. The intuition is that an automaton is a black box with the states forming the internal implementation, hidden to the outside. The externally observable behaviour is whether the automaton converges for particular inputs in M. For an automaton $\mathcal{M} = \langle X, \alpha, x_0\rangle$, we can define its *external behaviour* as

$$\mathcal{B}(\mathcal{M}) = \{\, m \in M \mid m \cdot x_0 \neq \emptyset \,\}$$

Two automata $\mathcal{M} = \langle X, \alpha, x_0\rangle$ and $\mathcal{M}' = \langle X', \alpha, x_0'\rangle$ are said to be *behaviorally equivalent* if $\mathcal{B}(\mathcal{M}) = \mathcal{B}(\mathcal{M}')$. It is possible to show that logical relations represent a complete reasoning principle for behavioral equivalence:

Fact 4.1 *Two automata $\mathcal{M}$ and $\mathcal{M}'$ are behaviorally equivalent if and only if there exists a logical relation between $\mathcal{M}$ and $\mathcal{M}'$.*

(While the existence of isomorphisms or homomorphisms is sufficient to ensure behavioral equivalence, neither of them gives a complete reasoning principle for it.)

To see this fact, let R^∞ be the greatest relation $R \subseteq X \times X'$ such that:

$$x \left[R\right] x' \implies \forall m \in M.\, m \cdot x \left[\mathcal{P}^1 R\right] m \cdot x'$$

(The existence of R^∞ can be inferred using the Tarski's fixed point theorem.) If $(x, x') \notin R^\infty$, then there must be some $m \in M$ such that $m \cdot x = \emptyset$ and $m \cdot x' \neq \emptyset$ or *vice versa*, *i.e.*, x and x' are not behaviorally equivalent states in their respective machines. So, if $\mathcal{M}$ and $\mathcal{M}'$ are behaviorally equivalent then $(x_0, x_0') \in R^\infty$, and R^∞ is the required logical relation.

Similar situation persists with other kinds of abstract machines. A *Mealy machine* of type $M \Rightarrow O$, where M and O are "known" monoids, is a set X equipped with a monoid homomorphism $\alpha : M \to [X \to X \times O]$. The *behaviour* of a Mealy machine with an initial state $\mathcal{M} = (X, \alpha, x_0)$ is defined as the input-output mapping $\mathcal{B}(\mathcal{M}) = \{\, (m, o) \mid \exists x.\, m \cdot x_0 = (x, o) \,\}$. Once again, homomorphisms represent an incomplete reasoning principle for behavioral equivalence of Mealy machines whereas logical relations represent a complete reasoning principle.

These observations suggest that isomorphisms, homomorphisms and logical relations make up a *spectrum* of correspondences between mathematical structures, with homomorphisms being a "halfway house." When we deal with information hiding, we face the *symmetric* concept of behavioral equivalence, which is unlikely to be characterized by the *asymmetric* concept of homomorphism. Isomorphisms seem appropriate when there is no information hiding involved and logical relations seem appropriate when there is information hiding involved.

We can capture information hiding type-theoretically using *existential types* proposed by Mitchell and Plotkin [30]. A signature $\langle \vec{a}, \Omega \rangle$, where all the sorts represent "hidden" types, can be expressed as the higher-order type:

$$\exists_{a_1,\ldots,a_n} (\Pi_{k \in I} F_k(a_1, \ldots, a_n)) \tag{7}$$

The existential operator "$\exists$," replacing the sum "Σ" in (2), captures the idea of information hiding. Thus M-automata have the type $\exists_a ([M \to [a \rightharpoonup a]] \times a)$ and Mealy machines have the type $\exists_a ([M \to [a \to a \times O]] \times a)$.

We call the elements of the type (7) *abstract algebras*. An abstract algebra is an *equivalence class* of algebras for $\langle \vec{a}, \Omega \rangle$ under "behavioral equivalence." Supported by the evidence from automata theory, we take behavioral equivalence to be the equivalence relation generated by the existence of logical relations. More precisely, we say that

(i) two algebras are *similar*, $\mathcal{A} \sim \mathcal{A}'$, iff there exists a logical relation $R : \mathcal{A} \leftrightarrow \mathcal{A}'$, and

(ii) two algebras are *behaviorally equivalent*, $\mathcal{A} \approx \mathcal{A}'$, iff there is sequence of algebras $\mathcal{A} = \mathcal{A}_0 \sim \mathcal{A}_1 \sim \cdots \sim \mathcal{A}_n = \mathcal{A}'$, where the successive algebras are similar with a logical relation between them.

Note that we do *not* require the similarity relation to be transitive, *i.e.*, no requirement for logical relations to compose. Indeed, whenever function types or other mixed variant type operators are involved, logical relations do not compose. We do not see this as a loss. (There have been proposals for composable forms of logical relations [37] but they are not uniformly defined.)

The notion of *abstract types* in programming languages and specification languages is the same as that of abstract algebras above. As a simple example, consider an abstract type **intset** for finite sets of integers, equipped with the operations $e : \mathbf{intset}$, $i : \mathbf{int} \times \mathbf{intset} \to \mathbf{intset}$, and $m : \mathbf{int} \times \mathbf{intset} \to \mathbf{bool}$ (for the empty set, the insertion of an element into a set, and the membership test in a set). Two simple ways to implement the abstract type are in terms of *(unordered) lists* (with possible duplicate copies) and *ordered lists*. (Many other sophisticated implementations such as binary search trees and hash tables etc. can be found in text books on data structures.) The two implementations would be behaviorally equivalent if there is a logical relation between them. The natural candidate for the logical relation is:

$$L \left[R\right] L' \iff |L| = |L'| \wedge \mathrm{ordered}(L')$$

where $|L|$ denotes the set of elements of list L. To show that R is a logical relation, one must show that the implementations of the operations are related: $e \left[R\right] e'$, $i \left[I_{\mathbf{int}} \times R \to R\right] i'$ and $m \left[I_{\mathbf{int}} \times R \to I_{\mathbf{bool}}\right] m'$. A fine method for proving the correctness of a data structure implementation is to prove that it is behaviorally equivalent to a naive implementation. Examples of such proofs may be found in Reynolds's *Craft of Programming* [41, Chapter 5] and de Roever and Englehardt[5].

5 Relational Parametricity

Parametric transformation is the concept parallel to natural transformation that works with logical relations instead of morphisms. Just as natural transformations are "maps of functors," parametric transformations are "maps of relators."

To keep this discussion concrete, we restrict attention to reflexive graphs of the form $\mathbf{rel}(\mathcal{C})$ for categories $\mathcal{C}$, with $\mathbf{set} = \mathbf{rel}(\mathbf{Set})$ being the prototypical example. We will refer to the category $\mathcal{C}$ as the "underlying category" of the reflexive graph. A more satisfactory axiomatization of the structure is given in Sec. 6.

Given reflexive graphs $\mathcal{G}$ and $\mathcal{H}$ and two relators $F, G : \mathcal{G} \to \mathcal{H}$, a parametric transformation $\eta : F \stackrel{\circ}{\to} G$ is a family of maps $\eta_A : F(A) \to G(A)$, indexed by vertices A of $\mathcal{G}$, such that, for all edges $R : A \leftrightarrow A'$ in $\mathcal{G}$, we have

$$\eta_A \left[F(R) \to G(R)\right] \eta_{A'} \tag{8}$$

Intuitively, η is a "parametrically polymorphic function" that preserves all logical relations between the vertices of $\mathcal{G}$.

Example 5.1 If the multiplication operation of a group is commutative, it is called an *abelian group*. There is a canonical abelianization of a group G, whose construction illustrates relators and parametric transformations in algebraic settings.

A *commutator* in a group G is a product of the form $aba^{-1}b^{-1}$, denoted by the short hand notation $[a, b]$. The *commutator subgroup* $C(G)$ of G is the collection of products of the form $[a_1, b_1] \cdots [a_n, b_n]$, for $n \geq 0$. Whenever $R : G \leftrightarrow G'$ is a logical relation of groups, there is a corresponding logical relation $C(R) : C(G) \leftrightarrow C(G')$ given by

$$C(R) = \{\, (\textstyle\prod_{i=1}^{n} [a_i, b_i],\ \prod_{i=1}^{n} [a'_i, b'_i]) \mid \forall i = 1, n.\ a_i \left[R\right] a'_i \wedge b_i \left[R\right] b'_i \,\}$$

Thus, we have a relator $C : \mathbf{grp} \to \mathbf{grp}$. (Intuitively, a commutator $[a, b]$ represents the equivalence relation $ab \approx ba$ and the group $C(G)$ represents the congruence relation generated by all such equivalences.)

The commutator subgroup $C(G)$ is closed under all the automorphisms of G (which are nothing but one-to-one logical relations) and, so, forms a normal subgroup of G. The quotient group $A(G) = G/C(G)$ is called the "abelianization" of G. Its elements are "cosets" $(x) = \{\, cx \mid c \in C(G) \,\}$ and multiplication is defined by $(x)(y) = (xy)$. $A(G)$ is an abelian group. We can make the A operation on groups into a relator $\mathbf{grp} \to \mathbf{grp}$ by defining $A(R) : A(G) \leftrightarrow A(G')$ as the relation that relates (x) and (x') whenever $x \left[R\right] x'$.

The projection maps $\nu_G : x \mapsto (x)$ that send elements $x \in G$ to cosets form a parametric transformation $\nu : \mathrm{Id} \stackrel{\circ}{\to} A : \mathbf{grp} \to \mathbf{grp}$. Whenever $x \left[R\right] x'$, we have $\nu_G(x) \left[A(R)\right] \nu_{G'}(x')$ directly from the definition of $A(R)$.

Note that the family of projection maps, ν, is a prototypical example of a natural transformation [25]. The example illustrates that the arguments we make for the naturality of such families easily generalize to parametricity. However, parametricity is a *more general* concept since it works with the larger class of *relators*,

which are not required to preserve composition. Hence parametricity applies when naturality fails to apply. The following examples illustrate this fact.

Example 5.2 Consider the *composition* of functions as a family of maps:

$$\circ_{ABC} : [B \to C] \times [A \to B] \to [A \to C]$$

where the type expressions on both the sides of $\to$ are treated as relators $\mathbf{set}^3 \to \mathbf{set}$ in the three type variables A, B, C. This is a parametric transformation (but the version defined with maps is not natural in B). Given relations $R : A \leftrightarrow A'$, $S : B \leftrightarrow B'$ and $T : C \leftrightarrow C'$, we have:

$$g \left[S \to T\right] g' \wedge f \left[R \to S\right] f' \implies g \circ f \left[R \to T\right] g' \circ f'$$

The "*evaluation*" map $ev_{AB} : [A \to B] \times A \to B$ given by $ev(f, x) = f(x)$ is similarly parametric in both A and B (as opposed to just B).

The reader will be able to construct similar examples for the internal homs in other closed categories.

Example 5.3 The family of *iteration* maps:

$$\tau_X : \mathbb{N} \to [[X \to X] \to [X \to X]]$$

given by $\tau_X(n)(f) = f^n$ is parametric. We have, for all relations $R : X \leftrightarrow X'$,

$$\tau_X \left[I_{\mathbb{N}} \to [[R \to R] \to [R \to R]]\right] \tau_{X'}$$

In fact, since the source type $\mathbb{N}$ is independent of X, we can regard τ as a *function* of type:

$$\tau : \mathbb{N} \to \forall_X [[X \to X] \to [X \to X]]$$

for a suitable internalization $\forall_X F(X) \to G(X)$ of parametric transformations. Once we do this, we notice that τ is in fact an *isomorphism*, i.e., *every* parametric transformation $\phi_X : [X \to X] \to [X \to X]$ is of the form $\lambda f.\, f^n$, the n'th *Church numeral* (Consider parametricity with respect to the relation $R : \mathbb{N} \leftrightarrow X$ given by $n \left[R\right] y \iff f^n(x) = y$.)

This result applies to total functions, but not partial or nondeterministic functions. For example, the type $\forall_X [[X \rightharpoonup X] \to [X \rightharpoonup X]]$ operating on partial functions has the following parametric "snap back" operation:

$$\lambda f.\, \lambda x.\, \begin{cases} \emptyset, & \text{if } f(x) = \emptyset \\ x, & \text{otherwise} \end{cases}$$

which evaluates f at x but discards the result, returning the original input x. O'Hearn and Tennent [32, Sec. 6] show that this polymorphic type is isomorphic $\mathbb{N}_\perp \times Vnat^{\mathrm{op}}$.

Arguments of this kind are familiar from the use of the Yoneda lemma for Hom-sets. Relational parametricity allows us to internalize them and reason about *internal homs* (function spaces) in the same way. The generalization to relations seems essential for dealing with internal homs. For example, note that, even though the relation $R : \mathbb{N} \leftrightarrow X$ in the above example is the graph of a function, the relation $R \to R$ is not the graph of a function.

To make the discussion concrete, we give a definition of $\forall_X F(X)$ in algebraic settings, parallel to that of $\exists_X F(X)$ in Sec. 2. In doing this, we run into a cardinality issue because the type variable X ranges over large collections such as "all sets," "all groups" etc. An element of $\forall_X F(X)$ is a family $\varphi = \{\varphi_X\}_X$ indexed by all types of this form and, so, is too large to be a set. Our preferred solution to the problem is to use Grothendieck universes, but we do not wish to belabor this point. We use the notation $\varphi \in \Pi_X F(X)$ to denote that φ is such a family.

If $F : \mathcal{G} \to \mathbf{set}$ is a relator then $\forall_X F(X)$ is the collection of families $\phi \in \Pi_{X \in \mathcal{G}_v} F(X)$ subject to the *parametricity requirement*:

$$\forall R : X \leftrightarrow X'.\ \varphi_X \ \big[F(R)\big]\ \varphi_{X'} \tag{9}$$

In general, we might have type expressions that have other type variables. So, more generally, if $F : \mathcal{G} \times \mathcal{H} \to \mathbf{set}$ is a relator then $\forall_X F(X, Y)$ is a relator $\mathcal{H} \to \mathbf{set}$ that sends vertices $Y \in \mathcal{H}_v$ to sets $\forall_X F(X, Y) \subseteq \Pi_X F(X, Y)$ and edges $S : Y \leftrightarrow Y'$ to relations $\forall_X F(X, S) : \forall_X F(X, Y) \leftrightarrow \forall_X F(X, Y')$. These are determined by the formulas:

$$\begin{aligned} \varphi \in \forall_X F(X, Y) &\iff \forall R : X \leftrightarrow X'.\ \varphi_X \ \big[F(R, I_Y)\big]\ \varphi_{X'} \\ \varphi \ \big[\forall_X F(X, S)\big]\ \varphi' &\iff \forall R : X \leftrightarrow X'.\ \varphi_X \ \big[F(R, S)\big]\ \varphi_{X'} \end{aligned}$$

The generalization of these concepts to reflexive graphs other than **set** appears in Sec. 6.

Local information hiding

The semantic intuition underlying parametric polymorphism is information hiding, the same phenomenon we have seen with existential types (Sec. 4), but working in a dual fashion. Whereas $\exists_X F(X)$ represents the construction of a "black box" that hides the information about the type X to the "outside," the type $\forall_X F(X)$ represents a construction that is generic in X and hence the information about X is hidden from it. We might say that the information hiding involved in $\exists_X$ is global whereas that involved in $\forall_X$ is local.

Whereas the phenomenon of global information hiding seems rare in mathematics, that of local information hiding is quite common. All the examples mentioned by Eilenberg and Mac Lane [10] for "natural" transformations exhibit this phenomenon. This also appears to be the sense in which the term "logical" was used by Plotkin [38] in naming "logical relations." Since the term-forming operations of the typed lambda calculus are "logical," they are independent of the specific

type information about the ground types. This led to the thesis that the lambda-definable elements should be invariant under all relations that carry out value substitutions of ground types ("permutations" in a general sense). Plotkin's results imply that selective information hiding is also captured by logical relations. An element lambda-definable from a set of elements Σ, preserves all relations R that are preserved by the elements of Σ. We see this notion at play, for instance, in Church numerals of type $\forall_X\,[X \to X] \to [X \to X]$. Given an unknown type X and values $f : X \to X$ and $z : X$, the definable elements of X are exactly those that preserve all relations preserved by f and z. Thus, parametricity gives us an extensional characterization of the intensional aspect of lambda-definability.

Parametric behaviour

The information hiding aspects of automata in Sec. 4 were captured by specifying a particular form of *external behavior* for an automaton. We can state the behavior as a polymorphic function of type

$$\mathcal{B} : \forall_X\,([M \to [X \rightharpoonup X]] \times X) \to (M \to \mathbf{2})$$

and note that it is parametric in the state set X. Thus the "global" information hiding for the state sets of automata is reducible to the "local" information hiding of its designated behavior function.

Generalizing this, we can postulate the following equivalence of types:

$$\exists_X\,F(X) \;\cong\; \forall_Y\,(\forall_X\,F(X) \to Y) \to Y \tag{10}$$

The type $\forall_X\,F(X) \to Y$ on the right hand side is the type of a possible behaviour function, which should be *parametric* in X. The type variable Y stands for the type of the observable behaviour, *e.g.*, $M \to \mathbf{2}$ in the case of automata. The universal quantifier $\forall_Y$ represents the idea that $\exists_X\,F(X)$ hides its representation type X from all possible observable behaviour functions.

It is possible to prove the equivalence (10) as a theorem from our definitions of $\exists_X$ and $\forall_X$ quantifiers. However, Reynolds [42] turned the equivalence (10) around and used it as the *definition* of $\exists_X$ in polymorphic lambda calculus. His calculus only has the $\forall_X$ quantifier built-in and $\exists_X$ is a derived notion.

In the literature on logical relations [28,29,48], it is common to find a "fundamental theorem" or "basic lemma" to the effect that all terms of a syntactic calculus preserve the defined logical relations. Reynolds's insight means that this result can be obtained in a syntax-independent way, by showing that all the primitives and combinators involved in the calculus are parametric. The "fundamental theorem" for logical relations is equivalent to the parametricity of the calculus. It would be worthwhile reexamining the literature on logical relations from this perspective.

6 Reflexive graph categories

To formalize the notion of parametric transformations and their internalization represented by the $\forall_X$ quantifier, we need an explicit notion of morphisms, *i.e.*, a suitable structure of categories.

Since reflexive graphs of Sec. 3 capture logical relations satisfactorily, an appropriate structure would be to consider categories internal to **RG**. Such a category would have have a reflexive graph of "objects" and a reflexive graph of "morphisms," We encourage the reader familiar with internal categories (see, for example, [25, Sec. XII.1]) to follow through this construction.

However, in one of those beautiful symmetries of nature, categories internal to **RG** turn out to be the same as reflexive graphs internal to **Cat**. Following O'Hearn and Tennent [32], we follow the latter approach.

Definition 6.1 A *reflexive graph category* (or a *reflexive graph of categories*) $\mathcal{G}$ involves a category $\mathcal{G}_v$ of "vertices" and a category $\mathcal{G}_e$ of "edges" along with functors $\partial_0, \partial_1 : \mathcal{G}_e \to \mathcal{G}_v$ and $I : \mathcal{G}_v \to \mathcal{G}_e$ that satisfy $\partial_0(I(A)) = \partial_1(I(A)) = A$.

A *reflexive graph-functor* (or *relational functor*) $F : \mathcal{G} \to \mathcal{H}$ is a pair of functors $F = (F_v : \mathcal{G}_v \to \mathcal{H}_v,\ F_e : \mathcal{G}_e \to \mathcal{H}_e)$ acting on the vertex category and edge category respectively, preserving all the structure on the nose: $\partial_i(F_e(E)) = F_v(\partial_i(E))$ and $F_e(I_A) = I_{F_v(A)}$. We normally drop subscripts of F_v and F_e using the context to disambiguate which is meant.

This is a straightforward generalization of Definition 3.1. However, it is important to note that there are four kinds of entities involved in a reflexive graph category: the objects and morphisms of $\mathcal{G}_v$ (which we continue to call "objects" and "morphisms") and the objects and morphisms of $\mathcal{G}_e$ (which we call "edges" and "edge morphisms"). If $R : A \leftrightarrow A'$ and $S : B \leftrightarrow B'$ are edges with an edge morphism $\phi : R \to S$, then the functors ∂_0 and ∂_1 pick out the morphisms $f : A \to B$ and $f' : A \to B'$ that ϕ "spans," as in the diagram below

$$\begin{array}{ccc} A & \xrightarrow{f} & B \\ {\scriptstyle R}\uparrow\downarrow & \phi & \uparrow\downarrow{\scriptstyle S} \\ A' & \xrightarrow[f']{} & B' \end{array} \tag{11}$$

The identity functor I gives an identity edge I_A for each object A as well as an identity edge morphism $I_f : I_A \to I_B$ for each morphism $f : A \to B$. Thus, in a reflexive graph category, we accommodate homomorphisms as well as logical relations side by side with an "equal status." The reader familiar with double categories [8,23] is invited to view reflexive graph categories as a weakening of the structure of double categories, omitting the notion of composition for vertical morphisms. In analogy with double categories, we also refer to edge morphisms such as ϕ as "squares."

Reflexive graph categories can be viewed as adding a category structure to reflexive graphs of Sec. 3. The *underlying reflexive graph* of a reflexive graph category

$\mathcal{G}$, obtained by omitting all the morphisms of $\mathcal{G}$, is denoted $|\mathcal{G}|$. We also feel free to treat $|\mathcal{G}|$ as a *discrete* reflexive graph category by implicitly adding identity arrows for all vertices A and edges R. A functor $F : |\mathcal{G}| \to \mathcal{H}$ from a discrete category is referred to as a "nonvariant" relational functor. It is essentially a relator because it ignores all the morphisms of $\mathcal{G}$. If $F : |\mathcal{G}| \to \mathcal{H}$ and $F' : |\mathcal{H}| \to \mathcal{K}$ are nonvariant relational functors, then there is a composite $F'F : |\mathcal{G}| \to \mathcal{K}$, and there is an "identity" nonvariant functor $\mathrm{Id}_{\mathcal{G}} : |\mathcal{G}| \to \mathcal{G}$. Hence, we can use the normal categorical notation for nonvariant functors. If $F : \mathcal{G} \to \mathcal{H}$ is a relational functor, we use the notation $|F| : |\mathcal{G}| \to \mathcal{H}$ to refer to its underlying nonvariant functor.

Reflexive graphs of the form $\mathbf{rel}(\mathcal{C})$ give rise to reflexive graph *categories* $\mathbf{Rel}(\mathcal{C})$. The underlying category $\mathcal{C}$ of the reflexive graph serves as the vertex category of $\mathbf{Rel}(\mathcal{C})$. Edges $R : A \leftrightarrow A'$ are subobjects $R \rightarrowtail A \times A'$ and edge morphisms $\phi : R \to S$ that span $f : A \to B$ and $f' : A' \to B'$ are the unique factors induced by $f \times f' : A \times A' \to B \times B'$.

$$\begin{array}{ccc} R & \overset{\phi}{\dashrightarrow} & S \\ \downarrow & & \downarrow \\ A \times A' & \xrightarrow{f \times f'} & B \times B' \end{array} \tag{12}$$

In this situation, we also say that ϕ is *above* the pair of morphisms (f, f') or that ϕ is a *lifting* of the pair (f, f') to $\mathcal{G}_e$. The identity edge $I_A : A \leftrightarrow A$ is the diagonal morphism $\delta_A : A \rightarrowtail A \times A$ and the identity edge morphism $I_f : I_A \to I_B$ above (f, f) is nothing but f.

The reader is invited to contemplate the correspondence between the diagrams (11) and (12), the first representing an abstract higher-dimensional view of relation-preservation and the second representing a normal categorical view.

The definition of $\mathbf{Rel}(\mathcal{C})$ allows us to view *every* category with finite products automatically as a reflexive graph category. From this point on, we change our "call signs," using names such as **Set**, **Grp**, **Mod** etc. to refer to the *reflexive graph categories obtained in this way.*

The reflexive graph category **Set**, for example, has functions $f : A \to B$ as its morphisms, binary relations $R \subseteq A \times A'$ as its edges, and relation preservation facts $f \left[R \to S\right] f'$ as its edge morphisms. Examples of relational functors include:

$$\begin{array}{ll} \mathcal{P}, \mathcal{P}^1 : \mathbf{Set} \to \mathbf{Set} & \times, + : \mathbf{Set} \times \mathbf{Set} \to \mathbf{Set} \\ \widehat{\mathcal{P}} : \mathbf{Set}^{\mathrm{op}} \to \mathbf{Set} & \to, \rightharpoonup, \rightsquigarrow : \mathbf{Set}^{\mathrm{op}} \times \mathbf{Set} \to \mathbf{Set} \end{array}$$

Note that, in any reflexive graph category of the form $\mathbf{Rel}(\mathcal{C})$, there is at most one edge morphism of type $R \to S$ for any given R and S. A reflexive graph category that has this property is said to be *relational.* Such a category has a hom-functor $\mathrm{Hom} : \mathcal{G}^{\mathrm{op}} \times \mathcal{G} \to \mathbf{Set}$, whose vertex part sends pairs of objects (A, B) to the set of morphisms $\mathrm{Hom}(A, B)$ and edge part sends pairs of edges (R, S) to relations $\mathrm{Hom}(R, S) \subseteq \mathrm{Hom}(A, B) \times \mathrm{Hom}(A', B')$.

Definition 6.2 Given two relational functors $F, G : \mathcal{G} \to \mathcal{H}$, a *parametric natural transformation* $\eta : F \dot{\to} G$ is a pair of natural transformations $\eta = (\eta_v : F_v \to G_v, \eta_e : F_e \to G_e)$ preserving the reflexive graph structure: $\partial_i \eta_e = \eta_v \partial_i$ and $I\eta_v = \eta_e I$. (As usual, we omit the subscripts of η_v and η_e when they can be discerned from the context.) If $\mathcal{G}$ is a discrete reflexive graph category, we simply call η a *parametric transformation* and write $\eta : F \overset{\circ}{\to} G$.

The vertex part η_v is an ordinary natural transformation. The edge part η_e is a family $\{\eta_R : F(R) \to G(R)\}_R$ indexed by edges $R : A \leftrightarrow A'$ in $\mathcal{G}$. The components η_R are edge morphisms in $\mathcal{H}$ that span morphisms $\eta_A : F(A) \to G(A)$ and $\eta_{A'} : F(A') \to G(A')$. Diagrammatically, we can picture this as follows:

$$\begin{array}{ccc} A & & F(A) \xrightarrow{\eta_A} G(A) \\ \uparrow\downarrow R & & F(R)\uparrow\downarrow \quad \eta_R \quad \uparrow\downarrow G(R) \\ A' & & F(A') \xrightarrow[\eta_{A'}]{} G(A') \end{array}$$

Remembering that, in a relational reflexive graph category such as $\mathbf{Rel}(\mathcal{C})$, there is at most one edge morphism η_R spanning η_A and $\eta_{A'}$ this condition says the same thing as formula (8), *viz.* parametricity involves the preservation of all logical relations between vertices.

However, note that Definition 6.2 imposes parametricity and naturality as two separate conditions of uniformity. This is redundant for categories like $\mathbf{Rel}(\mathcal{C})$. Since their logical relations subsume the morphisms via the graph construction $\langle - \rangle$, the preservation of logical relations automatically implies the preservation of morphisms as well. Secondly, the use of naturality limits us to functors, which run into variance problems with higher-order types. To get around the second problem, we use the notion of *nonvariant functors* described earlier in this section. A parametric natural transformation $\eta : F \dot{\to} G$ between nonvariant functors as per Definition 6.2 is simply referred to as a *parametric transformation* because its naturality is trivial.

Fact 6.3 *Reflexive graph categories, relational functors and parametric natural transformations form a 2-category* **RGCat**. *This 2-category is cartesian closed with products given pointwise and exponentials that have relational functors as objects.*

The products are given pointwise: $(\mathcal{G} \times \mathcal{H})_v = \mathcal{G}_V \times \mathcal{H}_v$ and $(\mathcal{G} \times \mathcal{H})_e = \mathcal{G}_e \times \mathcal{H}_e$. This is similar to reflexive graphs in Sec. 3, the difference being that we are now dealing with categories rather than sets.

For exponentials, the category $(\mathcal{H}^{\mathcal{G}})_v$ has relational functors $\mathcal{G} \to \mathcal{H}$ as objects and parametric natural transformations $\eta : F \dot{\to} G$ as morphisms. The edge category $(\mathcal{H}^{\mathcal{G}})_e$ has objects representing "higher-order relations" between relational functors. If F and F' are relational functors, an edge $\rho : F \leftrightarrow F'$ is a family $\{\rho_R\}_{R \in \mathcal{G}_e}$ of edges $\rho_R : F(A) \leftrightarrow F'(A')$ indexed by edges $R : A \leftrightarrow A'$ of $\mathcal{G}$. An edge morphism $\phi : \rho \to \sigma$ in $(\mathcal{H}^{\mathcal{G}})_e$ spanning $\eta : F \to G$ and $\eta' : F' \to G'$ is a family $\{\phi_R\}_R$ of edge morphisms $\phi_R : \rho_R \to \sigma_R$ in $\mathcal{H}$. Diagrammatically, an edge

morphism in $\mathcal{H}^{\mathcal{G}}$ shown on the left is a family of edge morphisms in $\mathcal{H}$ shown on the right:

$$\begin{array}{ccc} F & \xrightarrow{\eta} & G \\ {\scriptstyle\rho}\downarrow & \phi & \downarrow{\scriptstyle\sigma} \\ F' & \xrightarrow[\eta']{} & G' \end{array} \qquad \begin{array}{ccc} F(A) & \xrightarrow{\eta_A} & G(A) \\ {\scriptstyle\rho_R}\downarrow & \phi_R & \downarrow{\scriptstyle\sigma_R} \\ F'(A') & \xrightarrow[\eta'_{A'}]{} & G'(A') \end{array}$$

The identity edge $I_F : F \leftrightarrow F$ for a relational functor F is the family $\{FR\}_R$ and the identity edge morphism $I_\eta : I_F \to I_G$ for a parametric natural transformation $\eta : F \dot{\to} G$ is the family $\{\eta_R\}_R$. (In other words, the identity edge functor simply selects the edge parts: $I_F = F_e$ and $I_\eta = \eta_e$.)

Since **RGCat** is a 2-category, we have the notions of adjunctions, monads and comonads available for them. So, we can define the usual adjunctions

$$\begin{aligned} \Delta &\dashv \times && : \mathcal{G} \times \mathcal{G} \to \mathcal{G} \\ + &\dashv \Delta && : \mathcal{G} \to \mathcal{G} \times \mathcal{G} \\ (- \times A) &\dashv [A \to -] && : \mathcal{G} \to \mathcal{G} \end{aligned}$$

When we calculate these for particular reflexive graph categories such as **Set**, we obtain exactly the relation actions described in Sec. 2. A more detailed analysis of such adjunctions in the context of fibrations is given in Sec. 7.

Fact 6.4 *There is a "diagonal" relational functor $\Delta : \mathcal{H} \to \mathcal{H}^{\mathcal{G}}$ that sends the vertices of $\mathcal{H}$ to constant relational functors $\mathcal{G} \to \mathcal{H}$ and edges of $\mathcal{H}$ to constant higher-order relations between constant functors.*

For a vertex K of $\mathcal{H}$, the constant relational functor $\Delta_v(K) : \mathcal{G} \to \mathcal{H}$ sends every vertex of $\mathcal{G}$ to K, every morphism to id_K, every edge to the identity edge I_K and every edge morphism to I_{id_K}. For any edge $S : K \leftrightarrow K'$ in $\mathcal{H}$, $\Delta_e(S)$ in $(\mathcal{H}^{\mathcal{G}})_e$ is the constant family $\{S\}_{R \in \mathcal{G}_e}$ that maps every edge of $\mathcal{G}$ to S.

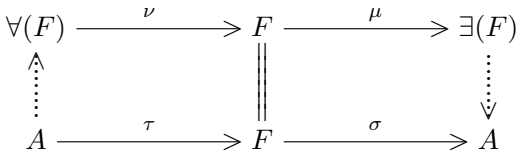
Finally, we define the type quantifiers $\exists$ and $\forall$ as the left and right adjoints to the diagonal functor:

$$\exists \dashv \Delta \dashv \forall \quad : \quad \mathcal{H}^{\mathcal{G}} \to \mathcal{H}$$

To explicate the detail, given a relational functor $F : \mathcal{G} \to \mathcal{H}$, the vertices $\forall(F)$ and $\exists(F)$, which are also denoted $\forall_X F(X)$ and $\exists_X F(X)$, are characterized as follows:

- $\forall_X F(X)$ comes equipped with a parametric natural transformation $\nu : \forall_X F(X) \dot{\to} F$ that is universal, i.e., any other parametric natural transformation $A \dot{\to} F$ for a vertex A of $\mathcal{H}$ uniquely factors through ν. We call this the *parametric limit* of F.
- $\exists_X F(X)$ comes equipped with a parametric natural transformation $\mu : F \dot{\to} \exists_X F(X)$ that is universal, i.e., any other parametric natural transformation $F \dot{\to} A$ for a vertex A of $\mathcal{H}$ uniquely factors through μ. We call this the *parametric colimit* of F.

Diagrammatically:



Note that these are the expected generalizations of the categorical concepts of "limit" and "colimit" to reflexive graph categories. When we calculate the adjoints for the case $\mathcal{H} = \mathbf{Set}$, we obtain the definitions given in Sec. 2 and Sec. 5.

It is also possible to define *parametric ends* and *coends* as the internalizations of parametric dinatural transformations in a similar way.

6.1 Subsumption

The framework of reflexive graph categories allows us to interpret type expressions in signatures for algebras as well as type expressions in polymorphic programming languages as *nonvariant relational functors* of the form $|\mathcal{G}| \to \mathcal{H}$. However, type expressions in which a type variable X occurs only positively, *e.g.*, $K \to X$ or $(X \to K) \to K$, are expected to be "functorial." They have an action on morphisms of $\mathcal{G}$. This is facilitated by the fact that the usual examples of reflexive graph categories such as **Set** or **Grp** have graph relations $\langle g \rangle$ for every morphism g, which play the same role as the underlying morphisms. To avoid overuse of the term "graph," we call $\langle g \rangle$ the "subsumption" relation of the morphism g. In this section, we give an axiomatic treatment of this property.

Every category $\mathcal{C}$ can be treated as a reflexive graph category $\mathbf{Arr}(\mathcal{C})$ whose vertex category is $\mathcal{C}$ and the edge category is the category of arrows of $\mathcal{C}$. In other words, an edge between A and A' is an arrow $g : A \to A'$ and an edge morphism $\phi : g \to h$ spanning $f : A \to B$ and $f' : A' \to B'$ is just a commuting square $h \circ f = f' \circ g$. This is a relational reflexive graph category. A reflexive graph-functor between such "arrow categories" is just an ordinary functor, and a parametric natural transformation between such functors is just a natural transformation.

Definition 6.5 A reflexive graph category $\mathcal{G}$ is said to be *subsumptive* if there is a (chosen) full and faithful functor, the "graph" functor, $\langle - \rangle : \mathbf{Arr}(\mathcal{G}_v) \to \mathcal{G}$ that is identity on the vertex category. A relational functor $F : \mathcal{G} \to H$ between such categories said to be *subsumptive* if it commutes with the graph functor: $F\langle g \rangle = \langle Fg \rangle$ for all morphisms g of $\mathcal{G}_v$.

The definition means that for every arrow $g : A \to A'$ of $\mathcal{G}_v$, there is an edge $\langle g \rangle : A \leftrightarrow A'$ such that $\langle \mathrm{id}_A \rangle = I_A$. Moreover, the full faithfulness requirement means that an edge morphism $\phi : \langle g \rangle \to \langle h \rangle$ between graph relations is the image

of a commutative square:

$$\begin{array}{ccc} A & \xrightarrow{f} & B \\ {\scriptstyle g}\downarrow & & \downarrow{\scriptstyle h} \\ A' & \xrightarrow{f'} & B' \end{array} \quad \Longleftrightarrow \quad \begin{array}{ccc} A & \xrightarrow{f} & B \\ {\scriptstyle \langle g\rangle}\uparrow & & \uparrow{\scriptstyle \langle h\rangle} \\ A' & \xrightarrow{f'} & B' \end{array} \tag{13}$$

The reflexive graph category $\mathcal{G}$ is relational at least between graph relations. So, relational squares on the right of the implication are unique when they exist. Further, since $I_A = \langle \mathrm{id}_A \rangle$, we obtain the *identity condition* [24]:

$$(f, f') : I_A \to I_B \iff f = f' \tag{14}$$

Reflexive graph categories of the form $\mathbf{Rel}(\mathcal{C})$ give examples of subsumptive reflexive graph categories with the choice of $\langle g\rangle : A \leftrightarrow A'$ as the monic $\langle \mathrm{id}_A, g\rangle : A \rightarrowtail A \times A'$.

Fact 6.6 *If $F, G : \mathcal{G} \to \mathcal{H}$ are subsumptive relational functors then parametric transformations of type $|F| \stackrel{\circ}{\to} |G|$ are bijective with parametric natural transformations of type $F \dot{\to} G$.*

In other words, *parametricity subsumes naturality.*

For dealing with contravariant functors, we need another notion. If $\mathcal{G}$ is a reflexive graph category, then we write $\mathcal{G}^{\mathrm{co}}$ for the reflexive graph category that has the same vertex and edge categories as $\mathcal{G}$ but the source and target functors ∂_0 and ∂_1 are exchanged. So, an edge $R : A \leftrightarrow B$ in $\mathcal{G}$ becomes an edge $B \leftrightarrow A$ in $\mathcal{G}^{\mathrm{co}}$. The reflexive graph category $\mathcal{G}^{\mathrm{op}}$, on the other hand, has reversed morphisms $f^\sharp : B \to A$ for every morphism $f : A \to B$ of $\mathcal{G}$ and reversed squares $\phi^\sharp : S \to R$ for every square $\phi : R \to S$ of $\mathcal{G}$.

Definition 6.7 A reflexive graph category $\mathcal{G}$ is said to have *converses* if there is an isomorphism $(\,)^\smile : \mathcal{G} \cong \mathcal{G}^{\mathrm{co}}$ that is identity on the vertex category. A relational functor $F : \mathcal{G} \to \mathcal{H}$ *preserves converses* if $F(R^\smile) = F(R)^\smile$.

Note that $\mathbf{Rel}(\mathcal{C})$ has converses. The converse of $m : R \rightarrowtail A \times B$ is $\mathrm{exch}_{AB} \circ m : R \rightarrowtail B \times A$. Every relational functor on $\mathbf{Rel}(\mathcal{C})$ preserves converses because it preserves isomorphisms. On the other hand, $\mathbf{Arr}(\mathcal{C})$ does not have converses.

Fact 6.8 *If a reflexive graph category $\mathcal{G}$ is subsumptive and has converses then $\mathcal{G}^{\mathrm{op}}$ is subsumptive, with the graph functor $\langle - \rangle : \mathbf{Arr}(\mathcal{G}_v^{\mathrm{op}}) \to \mathcal{G}^{\mathrm{op}}$ that sends edges $g^\sharp$ to $\langle g\rangle^\smile$ and commuting squares $(f_2^\sharp, f_1^\sharp) : h^\sharp \to g^\sharp$ to squares $(f_2^\sharp, f_1^\sharp) : \langle h\rangle^\smile \to \langle g\rangle^\smile$.*

Now, we can deal with contravariant functors in the same way as covariant functors above. If F and G are subsumptive relational functors of type $\mathcal{G}^{\mathrm{op}} \to \mathcal{H}$ then parametric transformations of type $|F| \stackrel{\circ}{\to} |G|$ are one-to-one with parametric natural transformations of type $F \dot{\to} G$. More interestingly, mixed variant functors and dinatural transformations can also be treated in the same way.

Fact 6.9 *If $F, G : \mathcal{G}^{\mathrm{op}} \times \mathcal{G} \to \mathcal{H}$ are subsumptive relational functors, where $\mathcal{G}$ and $\mathcal{H}$ are subsumptive reflexive graph categories and $\mathcal{G}$ has converses, then parametric*

dinatural transformations of type $F \stackrel{\cdot\cdot}{\to} G$ *are bijective with parametric transformations of type* $|F|\Delta \stackrel{\circ}{\to} |G|\Delta : |\mathcal{G}| \to \mathcal{H}$.

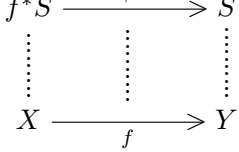
7 Fibrational framework for logical relations

Reflexive graph categories of the previous sections allow us to overlay a category of edges on top of a category of "types" for modelling the "properties" or "relations" over the types. The uniformity properties of parametrically polymorphic functions are characterized in terms of *invariance* under all properties and relations. Such properties and relations have a "logical" character, which can be understood using the framework of fibrations. We use fibrations as a framework of categorical logic, representing an alternative view of Lawvere's hyperdoctrines, as it has a good fit with logical relations and parametricity. See [21,50] for background on fibrations.

A *fibration* involves two categories $\mathcal{E}$ and $\mathcal{B}$ and a functor $\partial : \mathcal{E} \to \mathcal{B}$, subject to an axiom called "cartesian lifting" stated below. We also say that $\mathcal{E}$ is *fibred* over $\mathcal{B}$. The category $\mathcal{B}$, called the *base category*, models types. The category $\mathcal{E}$, called the *total category*, models the "edges," *i.e.*, abstract properties or relations over types. The functor ∂ designates the underlying types of the properties. If $\partial(R) = X$, we say that R is "above" X, and understand that R is a property or a relation of X-typed values. If $\phi : R \to S$ is a morphism such that $\partial(\phi) = f : X \to Y$, we say that ϕ is "above" f, and understand that ϕ represents an abstract proof witnessing the fact that f maps R-satisfying values of X to S-satisfying values of Y.

For any binary reflexive graph category $\mathcal{G}$, the functor $\partial = \langle \partial_0, \partial_1 \rangle : \mathcal{G}_e \to \mathcal{G}_v \times \mathcal{G}_v$ is a functor of this form. The notation $R : A \leftrightarrow A'$ of the previous sections implies that R is "above" the pair of vertices (A, A'). If ∂ satisfies the cartesian lifting property, we say that $\mathcal{G}$ is a *fibred reflexive graph category*.

The *cartesian lifting* property for ∂ is that, for any morphism $f : X \to Y$ in $\mathcal{B}$ and any edge S above Y, there is a canonical edge f^*S above X along with an edge morphism $\phi : f^*S \to S$ above f which is universal, *i.e.*, any other edge morphism $\psi : P \to S$ above a morphism of the form $g; f : X \to Y$ uniquely factors through ϕ. We can picture this in the following view:



where the dotted lines represent the idea of being "above" an object or arrow. The edge f^*S is called the *reindexing* of S along f and the edge morphism ϕ is called the *cartesian lifting* of f at S.

Example 7.1 Reflexive graph categories of the form $\mathbf{Rel}(\mathcal{C})$ are fibred provided the category $\mathcal{C}$ has pullbacks. Given a pair of arrows $(f, f') : (A, A') \to (B, B')$ and an edge $S \rightarrowtail B \times B'$ above (B, B'), the reindexing $(f, f')^*S$ is the pullback of $f \times f' : A \times A' \to B \times B'$ and $S \rightarrowtail B \times B'$. For example, in the reflexive graph

category **Set**, the reindexing is just the preimage:

$$(f, f')^* S = \{\, (x, x') \mid (f(x), f'(x')) \in S \,\}$$

In the Plotkin-Abadi logic for parametricity [36], the reindexed relation $(f, f')^* S$ can be expressed as a "definable relation:" $(x\colon A, x'\colon A')\, f(x)\, [S]\, f'(x')$.

The inverse image of an object X under ∂ is a subcategory of $\mathcal{E}$ called the *fibre* above X and denoted $\mathcal{E}_X$. Its objects are the edges above X and arrows are the edge morphisms above id_X. Reindexing along a morphism $f : X \to Y$ sends edges S in $\mathcal{E}_Y$ to edges $f^* S$ in $\mathcal{E}_X$ and, thus, can be regarded as a functor $f^* : \mathcal{E}_Y \to \mathcal{E}_X$ on the fibres.

The fibre $\mathcal{E}_X$ represents a logic for the properties of X-typed values. Its products $R \mathbin{\widehat{\times}} S$ represent conjunctions $R \wedge S$, the exponentials $R \mathbin{\widehat{\to}} S$ represent implications $R \Rightarrow S$ etc. The universal and existential quantifiers are obtained as the adjoints to the reindexing functors. If $\pi_{ZX} : Z \times X \to Z$ is the projection of the first component then, the reindexing functor $(\pi_{ZX})^* : \mathcal{E}_Z \to \mathcal{E}_{Z \times X}$ is thought of as the "weakening" operation which regards a property over Z as a property over $Z \times X$ by ignoring the X component. The existential quantifier $\coprod_{\pi_{ZX}}$ and the universal quantifier $\prod_{\pi_{ZX}}$ are the left and right adjoints of this operator:

$$\coprod_{\pi_{ZX}} \;\dashv\; (\pi_{ZX})^* \;\dashv\; \prod_{\pi_{ZX}} \;:\; \mathcal{E}_{Z \times X} \to \mathcal{E}_Z$$

7.1 Relation lifting

The formulas given in Sec. 2 and 5 for logical relations corresponding to various type constructors such as product and function space can be generalized to arbitrary fibrations as follows.

Given fibrations $\partial : \mathcal{E} \to \mathcal{B}$ and $\partial' : \mathcal{E}' \to \mathcal{B}'$, we talk of "lifting" a functor $F : \mathcal{B} \to \mathcal{B}'$ on the base categories to a functor $\widetilde{F} : \mathcal{E} \to \mathcal{E}'$ on the total categories. To say that $\widetilde{F}$ is a *lifting* of F is to say that the pair $(\widetilde{F}, F)$ preserves the fibration functor: $\partial' \circ \widetilde{F} = F \circ \partial$. Note that such a lifting need not be unique, and there is no requirement for it to preserve reindexing.

If $\eta, \epsilon : F \dashv G : \mathcal{B}' \to \mathcal{B}$ is an adjunction, then a corresponding adjunction $\tilde{\eta}, \tilde{\epsilon} : \widetilde{F} \dashv \widetilde{G} : \mathcal{E}' \to \mathcal{E}$ is called a *lifting* of the base adjunction if the functors $\widetilde{F}$ and $\widetilde{G}$ are liftings of the corresponding base functors F and G and the unit/counit $\tilde{\eta}$ and $\tilde{\epsilon}$ are above η and ϵ. A theory for lifting adjunctions in this way was developed in [16,17]. We illustrate it with examples for products and exponentials in the fibration $\partial : \mathbf{Rel} \to \mathbf{Set}$ of unary relations (predicates) over the category of sets.

For lifting products, we start with the adjunction $\Delta \dashv \times : \mathbf{Set}^2 \to \mathbf{Set}$ and proceed as indicated in Fig. 1. The lifting $\widetilde{\Delta} : \mathbf{Rel} \to \mathbf{Rel}^2$ is the evident functor $R \mapsto (R, R)$, $\phi \mapsto (\phi, \phi)$. It is split into two parts $\widehat{\Delta}$ and $\overline{\Delta}$ with the motivation of using the "change of base" lifting represented in the right square [17, Lemma 4.1].

- The category $\mathbf{Rel}_\Delta$ represents the pullback of ∂^2 and Δ in **Cat**. Its objects are pairs of relations (R, S) above the same set A.

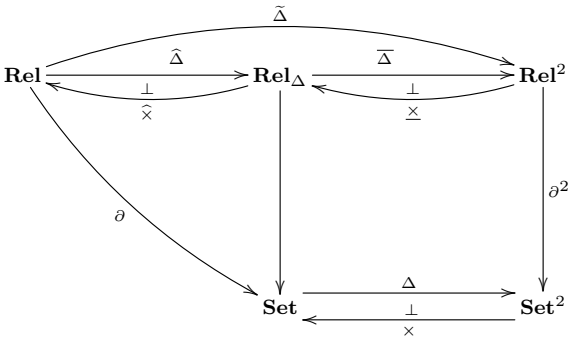


Fig. 1. Lifting of the product adjunction

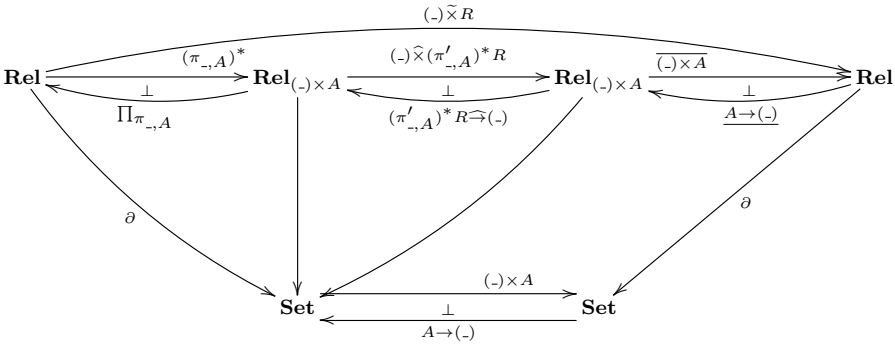


Fig. 2. Lifting of the exponential adjunction

- The functor $\widehat{\Delta} : \mathbf{Rel} \to \mathbf{Rel}_{\Delta}$ is the fibre-wise diagonal functor sending R above A to (R, R) above A. Its right adjoint is the fibre-wise product functor $\widehat{\times}$, representing the conjunction or intersection of relations.
- The functor $\overline{\Delta} : \mathbf{Rel}_{\Delta} \to \mathbf{Rel}^2$ sends (R, S) above A to (R, S) above (A, A). Its right adjoint $\underline{\times}$ is obtained by reindexing along the counit of the base adjunction, *viz.*, (π, π'). So, $R \underline{\times} S = ((\pi_{AB})^* R,\ (\pi'_{AB})^* S)$.
- The right adjoint to $\widetilde{\Delta}$ is then the composite of the two right adjoints:

$$R \mathbin{\widetilde{\times}} S \ = \ (\pi_{AB})^* R \mathbin{\widehat{\times}} (\pi'_{AB})^* S$$

This gives the formula used in Sec. 2, *viz.*, $(R \mathbin{\widetilde{\times}} S)(a, b) \Leftrightarrow R(a) \wedge S(b)$.

Lifting the exponential adjunction from **Set** to **Rel** is a little more involved as shown in Fig. 2. The middle and right adjunctions in the total category are similar to those for the product adjunction. The left adjunction amounts to requiring the fibration to have right adjoints to reindexing along projections, which is nothing

but universal quantification. On the whole, this gives the formula:

$$R \mathrel{\widetilde{\to}} S \;=\; \prod_{\pi_{A\to B,A}} ((\pi'_{A\to B,A})^* R \mathrel{\widehat{\to}} (ev_{A,B})^* S)$$

Expressed in terms of elements, it means $(R \mathrel{\widetilde{\to}} S)(f) \Leftrightarrow \forall a\colon A.\, R(a) \Rightarrow S(f(a))$.

7.2 Fibred reflexive graph categories

As noted above, the fibration concept can be combined with that of reflexive graph categories (which we abbreviate to "RG-categories") by asking for the edges to be fibred over tuples of vertices. More precisely, a fibred RG-category satisfies the condition that the functor $\partial = \langle \partial_0, \partial_1 \rangle : \mathcal{G}_e \to \mathcal{G}_v \times \mathcal{G}_v$ is a fibration. Explored in [6,7], this structure allows us to import the "logical character" of fibrations to the setting of RG-categories.

The reindexing operation of ∂ implies that there is a "pre-image" edge $(f, f')^* S$ as shown in the square on the right:

$$\begin{array}{ccc} (f,f')^*S & \xrightarrow{\ \phi\ } & S \\ \vdots & & \vdots \\ (A,A') & \xrightarrow{\ (f,f')\ } & (B,B') \end{array} \qquad \begin{array}{ccc} A & \xrightarrow{\ f\ } & B \\ {\scriptstyle (f,f')^*S}\uparrow\downarrow & \phi & \uparrow\downarrow{\scriptstyle S} \\ A' & \xrightarrow[\ f'\ ]{} & B' \end{array}$$

From reindexing, we immediately obtain a general formula for the *graph* of a morphism $\langle f \rangle = (f, \mathrm{id}_B)^* I_B$. In **Set**, this reduces to $\langle f \rangle = \{\, (x,y) \mid f(x) =_B y \,\}$. In $\mathbf{Poset}_{\sqsupseteq}$, it reduces to the relation $\{\, (x,y) \mid f(x) \sqsupseteq_B y \,\}$, which we denoted by $\langle f \rangle_{\sqsupseteq}$ in Sec. 3.

By requiring the *identity condition*, $f \left[I_A \to I_B \right] f' \iff f = f'$, we obtain subsumptive RG-categories mentioned in Sec. 6.1. By requiring an "ordered" identity condition for locally ordered categories, $f \left[I_A \to I_B \right] f' \iff f \sqsupseteq f'$, we obtain weaker structures that we might call lax-subsumptive RG-categories, which include $\mathbf{Poset}_{\sqsupseteq}$. Thus, fibred RG-categories give us a more general framework than simple subsumption.

The natural choice of 1-cells for fibred RG-categories would be that of *fibred RG-functors*, *i.e.*, RG-functors $\mathcal{G} \to \mathcal{H}$ that send cartesian morphisms in $\mathcal{G}$ to cartesian morphisms in $\mathcal{H}$. However, this choices runs into the problem that fibred RG-categories do not have duals. If $\mathcal{G}$ is a fibred RG-category, the RG-category $\mathcal{G}^{\mathrm{op}}$ obtained by reversing its arrows is not fibred. It is *cofibred*, with universal cocartesian morphisms $R \to (f, f')_* R$ above pairs of arrows (f, f'). Therefore, we cannot treat the function type constructor $\to : \mathcal{G}^{\mathrm{op}} \times \mathcal{G} \to \mathcal{G}$ as a fibred RG-functor. We would need *bifibred* RG-categories, *i.e.*, those that are both fibred and cofibred, to accommodate the function type constructor.

8 Structural Induction and coinduction

Programming languages allow recursively defined types, which may be viewed as freely generated algebras. For example, the type of lists over A defined by

$$List\, A \;\cong\; 1 + A \times List\, A$$

represents a freely generated algebra with a single sort X and an operation $in : (1 + A \times X) \to X$. The source of the operation is given by an endofunctor $TX = 1+A\times X$ and the single operation of the algebra $in : TX \to X$ is called its "structure map." The freely generated algebra, which is the intent of the recursive definition, is the initial object in the category of such algebras. The initial algebra facilitates inductive definition of functions as well as inductive proofs of properties. Relational parametricity plays a key role in interpreting such principles of induction.

Suppose $T : \mathcal{G} \to \mathcal{G}$ is a relational functor. An *algebra* for T is a pair $\langle X,\, \alpha\rangle$ of a "carrier" object X of $\mathcal{G}$ and a "structure map" $\alpha : TX \to X$. A *logical relation* of T-algebras is a pair $\langle R, \phi\rangle : \langle X, \alpha\rangle \leftrightarrow \langle X', \alpha'\rangle$ of an edge $R : X \leftrightarrow X'$ and an edge morphism $\phi : TR \to R$ above (α, α'). The identity logical relation is $I_{\langle X,\alpha\rangle} = \langle I_X, I_\alpha\rangle$. A *morphism* of T-algebras $f : \langle X, \alpha\rangle \to \langle Y, \beta\rangle$ is a morphism $f : X \to Y$ in $\mathcal{G}$ such that $\beta \circ Tf = f \circ \alpha$. With a suitable notion of edge morphisms, we obtain a reflexive graph category $\mathbf{Alg}(T)$ of T-algebras.

The initial object in $\mathbf{Alg}(T)$ is called the *initial algebra* of T, denoted $\mu(T)$ or $\mu_X T(X)$. Lambek made the important observation that the structure map of the initial algebra $\alpha : T(\mu(T)) \to \mu(T)$ is an isomorphism. From a computer science point of view, we may also think of $\mu(T)$ as the "least fixed point" of T that is inductively generated.

The type of lists over A is thus the initial algebra $\mu_X\,(1 + A \times X)$ of the endofunctor $TX = 1 + A \times X$. Note that the functor has a corresponding relation action $T_eR = I_1 + I_A \times R$. To give an inductive definition of a function on lists, *e.g.*, to determine the length of a list $length : List\, A \to \mathbb{Z}$, it is adequate to give T-algebra structure on $\mathbb{Z}$. The structure map in this case is $\beta : T\mathbb{Z} \to \mathbb{Z}$ given by $\beta = [\lambda z.\, 0,\ \lambda(a, x).\, 1 + x]$. Since the initial algebra $\mu(T)$ has a unique morphism to any other algebra $\langle Y, \beta\rangle$ in $\mathbf{Alg}(T)$, we obtain a map from lists to integers. We denote the unique morphism in question by $fold_{\langle Y,\beta\rangle} : \mu(T) \to \langle Y, \beta\rangle$.

To prove a property P of such an inductively defined function for "all lists," *e.g.*, to prove that $length(l)$ is non-negative, the structural induction principle says that it is enough to prove it for the empty list and for a nonempty list (a, l') assuming it to be true for l'. The proof amounts to showing that the structure map of $\langle Y,\, \beta : TY \to Y\rangle$ preserves the property P in the sense that, whenever a value $z : TY$ satisfies T_eP, $\beta(z) : Y$ satisfies P. In other words, there is an edge morphism $\psi : T_eP \to P$ above $\beta : TY \to Y$, which is to say we have a T_e-algebra $\langle P, \psi\rangle$ above $\langle Y, \beta\rangle$ in the reflexive graph category $\mathbf{Alg}(T)$.

Structural induction allows us to conclude from this that $fold_{\langle Y,\beta\rangle} : \mu(T) \to \langle Y, \beta\rangle$ has an edge morphism above it $I_{\mu(T)} \to \langle P, \psi\rangle$ in $\mathbf{Alg}(T)$. Postulate this as a unique edge morphism $fold_{\langle P,\psi\rangle} : I_{\mu(T)} \to \langle P, \psi\rangle$. In other words, we expect

that $I_{\mu(T)}$ is the initial algebra $\mu(T_e)$ of the T functor at the level of edges. Recalling that $I_T = T_e$ in the functor category $\mathcal{G}^{\mathcal{G}}$ (Fact 6.3), our expectation amounts to $I_{\mu(T)} \cong \mu(I_T)$, which is nothing but the *identity extension* property for the μ operator.

The idea that structural induction is nothing but the identity extension property of the μ operator was proposed by Hermida and Jacobs [19] in the setting of fibrations.

Similarly, a coalgebra for a functor $T : \mathcal{G} \to \mathcal{G}$ is a pair $\langle X, \alpha : X \to TX \rangle$. There is a reflexive graph category $\mathbf{CoAlg}(T)$ of such coalgebras. The final object in the category is denoted $\nu(T)$ or $\nu_X T(X)$. Properties of final coalgebras can be proved using a coinduction principle, whose substance is again an identity extension property: $I_{\nu(T)} \cong \nu(I_T)$.

In fact, both initial algebras and final coalgebras can be expressed in terms of parametric limits and colimits using the formulas proposed by Plotkin and Abadi [36]:

$$\mu_X\, T(X) = \forall_X\, (T(X) \to X) \to X$$
$$\nu_X\, T(X) = \exists_X\, (X \to T(X)) \times X$$

Here $T(X)$ is a type expression that is functorial. The equations hold in many parametric models of the (impredicative) polymorphic lambda calculus. Birkedal and Mogelberg [3] give a categorical axiomatization of such models. What happens beyond these models is not entirely clear. See Dunphy [6] for some results in this direction. Characterizing the classes of models where these equations hold and where they fail would form important steps in advancing our understanding of parametricity.

9 Further work

Reynolds's exhortation to generalize homomorphisms from functions to relations comes to us as a "bolt from the blue." Its implications will no doubt be far-reaching. In this article, we have attempted to give some idea of how Reynolds's ideas might apply to mathematical considerations, for universal algebra and category theory. This work is far from complete. We outline some possible directions for future investigation.

Formulating a suitable categorical structure for representing logical relations and relational parametricity tops our list of priorities. In our treatment, we presented two possible approaches: *reflexive graph categories*, which model the action of type constructors on abstract relations along with the identity extension postulate, and *fibrations*, which explain in a syntax-independent manner, the origin of logical relations formulae needed to obtain the relation actions. It appears to us that the eventual theory of parametricity needs to integrate the two approaches. This might involve understanding and isolating the stumbling blocks for the use of fibred functors. As well as the contravariance needed for the function type constructor, see Pitts [35, Sec. 4] for some of the issues in preserving cartesian liftings.

The category theorist would no doubt wonder about the notion of composition

for logical relations. The problem is that, in the first place, the function type constructor does not preserve the composition of relations, and, secondly, composition brings back the variance issues of functors, which logical relations are meant to avoid. Note that homomorphisms pre- and post-compose with logical relations, although the resulting bimodule structures remains to be explored. It would also be interesting to consider composite logical relations of higher arities [22]. On the other hand, the structure of relations under composition leads to the consideration of bicategories of relations, which might also be appropriate in some contexts. The categorical analysis of relational modalities carried out in [18] considers this direction. The lifting of endofunctors in this context, with applications to coalgebraic bisimulation in the spirit of Sec. 2 is addressed in [2] and the references therein.

Cross connections with other areas that employ relational correspondences need to be made. We have in mind, for instance, the work in universal algebra dealing with Galois connections between theories and relations, as well as the growing body of work in computer science on coalgebraic bisimulations and modalities.

A logic for parametricity has been proposed in [36], where all the previously known consequences of Reynolds's identity extension postulate have been formally derived. The soundness of such a system is established in [3] using fibrational models. Dunphy [6] proposes a logic called System P for reasoning about fibred reflexive graph categories appropriate for polymorphic lambda calculus. Such logics can be useful for abstracting from some of the sophisticated categorical machinery involved in the models.

References

[1] Abramsky, S. A. and T. P. Jensen, *A relational approach to strictness analysis for higher-order polymorphic functions*, in: *Eighteenth Ann. ACM Symp. on Princ. of Program. Lang.*, ACM, 1991, pp. 49–54.

[2] Bílková, M., A. Kurz, D. Petrisan and J. Velebil, *Relation lifting, with an application to the many-valued cover modality*, Logical Methods in Comp. Sci. **9** (2013), pp. 1–48.

[3] Birkedal, L. and R. E. Møgelberg, *Categorical models of Abadi-Plotkin's logic for parametricity*, Math. Struct. Comput. Sci. **15** (2005), pp. 709–772.

[4] Bodnarchuk, V. G., L. A. Kaluzhnin, N. N. Kotov and B. A. Romov, *Galois theory for Post algebras, I*, Cybernetics **5** (1969), pp. 243–252.

[5] de Roever, W.-P. and K. Engelhardt, "Data Refinement: Model-Oriented Proof Methods and their Comparison," Cambridge Univ. Press, 1998.

[6] Dunphy, B. P., "Parametricity as a Notion of Uniformity in Reflexive Graphs," Ph.D. thesis, University of Illinois, Dep. of Mathematics (2002), available electronically from http://www.cs.bham.ac.uk/~udr.

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

[8] Ehresmann, C., *Catégories structurées*, Ann. Sci. École Norm. Sup. **80** (1963), pp. 349–425.

[9] Eilenberg, S., "Automata, Languages, and Machines; Vol. B," Academic Press, 1976.

[10] Eilenberg, S. and S. Mac Lane, *General theory of natural equivalences*, Trans. Amer. Math. Society **58** (1945), pp. 231–294.

[11] Fiore, M. P., A. Jung, E. Moggi, P. W. O'Hearn, J. Riecke, G. Rosolini and I. Stark, *Domains and denotational semantics: History, accomplishments and open problems*, Bulletin of the European Assoc. for Theoretical Computer Science **59** (1996), pp. 227–256.

[12] Freyd, P., *Core algebra revisited*, Theoretical Comput. Sci. **375** (2007), pp. 193–200.

[13] Geiger, D., *Closed systems of functions and predicates*, Pacific J. Math. **27** (1968), pp. 95–100.

[14] Ginzburg, A., "Algebraic Theory of Automata," Academic Press, New York, 1968.

[15] Ginzburg, A. and M. Yoeli, *Products of automata and the problem of covering*, Trans. Amer. Math. Soc **116** (1965), pp. 253–266.

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

[17] Hermida, C., *Some properties of **fib** as a fibred 2-category*, J. Pure and Applied Algebra **134** (1999), pp. 83–109.

[18] Hermida, C., *A categorical outlook on relational modalities and simulations*, Inf. Comput. **209** (2011), pp. 1505–1517.

[19] Hermida, C. and B. Jacobs, *Structural induction and coinduction in a fibrational setting*, Inf. Comput. **145** (1998), pp. 107–152.

[20] Hoare, C. A. R., *Proof of correctness of data representations*, Acta Informatica **1** (1972), pp. 271–281.

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

[22] Jung, A. and J. Tiuryn, *A new characterization of lambda definability*, 1993 pp. 245–257.

[23] Kelly, G. M. and R. Street, *Review of the elements of 2-categories*, in: G. M. Kelly, editor, *Proc. Sydney Category Seminar*, Lect. Notes Math. **420**, Springer-Verlag, 1974 pp. 75–103.

[24] Kinoshita, Y., P. W. O'Hearn, A. J. Power, M. Takeyama and R. D. Tennent, *An axiomatic approach to binary logical relations with applications to data refinement*, in: M. Abadi and T. Ito, editors, *Theoret. Aspects of Comp. Softw.*, LNCS **1281**, Springer-Verlag, 1997 pp. 191–212.

[25] Mac Lane, S., "Categories for the Working Mathematician," Springer-Verlag, 1991, second edition.

[26] Mac Lane, S. and G. Birkhoff, "Algebra," Chelsea, New York, 1993, third edition.

[27] Milner, R., *An algebraic definition of simulation between programs*, in: *Proc. Second Intern. Joint Conf. on Artificial Intelligence*, The British Computer Society, London, 1971, pp. 481–489.

[28] Mitchell, J. C., *Type systems for programming languages*, in: J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B*, North-Holland, Amsterdam, 1990 pp. 365–458.

[29] Mitchell, J. C., "Foundations of Programming Languages," MIT Press, 1997.

[30] Mitchell, J. C. and G. D. Plotkin, *Abstract types have existential types*, ACM Trans. Program. Lang. Syst. **10** (1988), pp. 470–502.

[31] Mitchell, J. C. and A. Scedrov, *Notes on sconing and relators*, in: *Computer Science Logic '92, Selected Papers*, LNCS **702**, Springer-Verlag, 1993 pp. 352–378.

[32] O'Hearn, P. W. and R. D. Tennent, *Parametricity and local variables*, J. ACM **42** (1995), pp. 658–709, (Reprinted as Chapter 16 of [33]).

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

[34] Parnas, D. L., *Information distribution aspects of design methodology*, in: *IFIP Congress 71*, North-Holland, 1971, pp. 339–344.

[35] Pitts, A. M., *Relational properties of domains*, Inf. Comput. **15** (1996), p. 66.

[36] Plotkin, G. and M. Abadi, *A logic for parametric polymorphism*, 1993, pp. 361–375.

[37] Plotkin, G., J. Power, D. Sannella and R. Tennent, *Lax logical relations*, in: *Intern. Colloq. Aut., Lang. and Program.*, Springer-Verlag, 2000 pp. 85–102.

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

[39] Plotkin, G. D., *Domains (Pisa notes)*, Electronic manuscript, University of Edinburgh (1983), available from http://homepages.inf.ed.ac.uk/gdp/publications/.

[40] Pöschel, R., *Galois connections for operations and relations*, in: K. Denecke, M. Erne and S. L. Wismath, editors, *Galois Connections and Applications*, Kluwer, 2004 pp. 231–258.

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

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

[43] Reynolds, J. C., *Polymorphism is not set-theoretic*, in: G. Kahn, D. B. MacQueen and G. Plotkin, editors, *Semantics of Data Types*, LNCS **173**, Springer-Verlag, 1984 pp. 145–156.

[44] Robinson, E. and G. Rosolini, *Reflexive graphs and parametric polymorphism*, in: *Proc. Ninth Ann. IEEE Symp. on Logic in Comp. Sci.*, IEEE, 1994, pp. 364–371.

[45] Sangiorgi, D., *On the origins of bisimulation and coinduction*, ACM Trans. Program. Lang. Syst. **31** (2009), p. 15.

[46] Sannella, D. and A. Tarlecki, "Foundations of Algebraic Specification and Formal Software Development," Springer-Verlag, 2012.

[47] Scott, P. J., *Some aspects of categories in computer science*, in: M. Hazewinkel, editor, *Handbook of Algebra, Vol. 2*, Elsevier, 2000 pp. 3–77.

[48] Statman, R., *Logical relations and the typed lambda calculus*, Inf. Control **65** (1985), pp. 85–97.

[49] Strachey, C., *Fundamental concepts in programming languages*, J. Higher-order Symbolic Comput. **13** (2000), pp. 11–49, (original lecture notes, Copenhagen, 1967).

[50] Streicher, T., *Fibred categories à la Bénabou* (1999), lecture Notes, available electronically from http://www.mathematik.tu-darmstadt.de/~streicher/.

[51] van der Waerden, B. L., "Modern Algebra," Unger, New York, 1949, second edition, (Translated from German by Fred Blum, original version 1930-31).