

Presheaves as Configured Specifications Submitted to the FACS Journal (changed “module” to “instance” on 25.10.99)

Steven Vickers¹ and Gillian Hill²

¹Department of Computing, Imperial College, London SW7 2BZ, UK;

²Department of Computer Science, City University, Northampton Square, London,
EC1V 0HB and Department of Computing, Imperial College, London SW7 2BZ, UK.

Keywords: category; presheaf; colimit; configuration; specification; diagram; instance

Abstract. The paper addresses a notion of configuring systems, constructing them from specified component parts with stipulated sharing. This notion is independent of any underlying specification language and has been abstractly identified with the taking of colimits in category theory. In particular, Oriat has shown how to use a category of diagrams $\text{Diag}(\mathbf{C})$ to express colimits of primitive specifications from a category \mathbf{C} . Mathematically it is known that these can equally well be expressed by presheaves over \mathbf{C} and the present paper applies this idea to configuration. We develop an algebraic account of presheaves, and show that Oriat’s category $\text{Diag}(\mathbf{C})$ is equivalent to the category of finitely presented presheaves over \mathbf{C} . The presheaf structure is interpreted in terms of instances of specifications and instance reduction and a configuration language is outlined that expresses precisely the finite presentations of presheaves but also describes configuration in a natural way.

1. Introduction

The language of category theory is now well-established in computer science, and it has long been recognized that constructing complex specifications out of simpler ones can often be understood categorically in terms of colimits.

Correspondence and offprint requests to: Steven Vickers (sjv@doc.ic.ac.uk), Gillian Hill (gah@doc.ic.ac.uk), Department of Computing, Imperial College, London SW7 2BZ, UK.

A categorical semantics was given by Burstall and Goguen [BG77, BG79] for Clear: a language for writing structured specifications. A Clear text denotes a many-sorted theory; the structuring of theories is expressed by colimits in order to take account of shared sub-theories. This has continued in work in the algebraic approach to specification [EM90, EFHLP87] and also in proof-theoretic approaches [MFS90, SJ95].

Oriat [Ori96] has modelled the composition of specifications by working within an equiv-category of diagrams over an equiv-category of base specifications. An “equiv-category” is a category with a (suitably well-behaved) equivalence relation on each hom-set, the point being that in constructing a finite cocompletion (which is what is needed) the diagrams are the right objects but the diagram morphisms are too concrete — the equivalence relation must be factored out.

There is an alternative construction of cocompletions using presheaves instead of diagrams, which is described by, for instance, MacLane and Moerdijk [MM92]. If \mathbf{C} is a small category, then the Yoneda embedding \mathbf{y} is a full and faithful functor from \mathbf{C} to the cocomplete category $\mathbf{Set}^{\mathbf{C}^{\text{op}}}$ of presheaves. Every presheaf P can be expressed as a colimit of representable presheaves (i.e. those of the form $\mathbf{y}(X)$) as follows. Associated with P is its *category of elements* $\int_{\mathbf{C}} P$: its objects are pairs (x, X) where X is an object of \mathbf{C} and $x \in P(X)$, while a morphism from (x, X) to (y, Y) is a morphism $u : X \rightarrow Y$ in \mathbf{C} such that $x = P(u)(y)$. The obvious functor from $\int_{\mathbf{C}} P$ to \mathbf{C} gives an $\int_{\mathbf{C}} P$ -shaped diagram in \mathbf{C} , and composing with \mathbf{y} gives an $\int_{\mathbf{C}} P$ -shaped diagram in $\mathbf{Set}^{\mathbf{C}^{\text{op}}}$ of which P is a colimit. Every functor from \mathbf{C} to a cocomplete category extends in an essentially unique way to a cocontinuous functor from $\int_{\mathbf{C}} P$. We shall be interested in colimits of finite diagrams, and this corresponds to a restriction to presheaves that are *finitely presented* in a standard algebraic sense.

We shall show that, mathematically, the account using presheaves is equivalent to — though simpler than — Oriat’s account using diagrams: finite diagrams correspond to finite presentations of presheaves, Oriat’s morphisms using ‘zigzags’ correspond to definitions of presheaf homomorphisms, and her equivalence between morphisms corresponds to equality of the homomorphisms. However, our investigation shows that the presheaves are more than just a sophisticated mathematical trick. Using generators and relations algebraically to present a presheaf corresponds in a direct conceptual way to specifying components and the sharing of subcomponents in a composite system.

In [Hil95, Hil97] an attempt was made to define a “configuration language”, in which composite “configured” specifications could be described as colimits of specifications already defined, in a categorical way that was independent of the underlying logic of the specifications. The present paper develops those attempts and outlines a concise configuration language with a syntax based directly on the algebraic presentation of presheaves.

At the same time those papers also used the term “module” to mean “uniquely-named instance of a specification”: the relationship between a specification and a module here was the same as that between an object class (in the object-oriented approach) and an instance or object of it. The “unique naming” of a module was its physical identity, analogous to the memory address of an object in the execution of an object-oriented program. One should bear in mind that this conflicts with some other uses of the word “module”. For instance, module as “composite structure wrapped up as a single unit” (such as a program module) is more like

the way we use “configured specification”. For this reason we now use the term “instance” rather than “module”. We shall show how the (categorical) objects and morphisms of a category $\int_{\mathbf{C}} P$ of elements can be understood as instances and “instance reductions” (i.e. extracting subcomponents), and thereby give a direct physical reading to the configuration language.

The paper is arranged as follows. In Section 2 we present the algebraic theory of presheaves and demonstrate that finite diagrams are finite presentations of presheaves. We then show, in Section 3 how presheaf theory leads both to a new physical interpretation of instances as instances of specifications and to the notion of instance reduction. The specifications that express the configuration of systems are themselves viewed as presheaves in Section 4, where we outline the new configuration language. Conclusions are drawn in Section 5.

2. Presheaves as Many-Sorted Algebras

Throughout this section we shall fix a small category \mathbf{C} and consider presheaves over it. As is well known, they are the algebras for a many-sorted algebraic theory, and we shall examine some consequences of this from universal algebra.

2.1. Presentations of Presheaves

We define a presheaf P on \mathbf{C} as a functor from \mathbf{C}^{op} to \mathbf{Set} . The category $\mathbf{Set}^{\mathbf{C}^{\text{op}}}$ is the category of all presheaves on \mathbf{C} , the morphisms being natural transformations.

The basis of our account is to treat presheaves as many-sorted algebras, the sorts being the objects of \mathbf{C} . The carrier of the presheaf P for the sort X is $P(X)$. For each morphism $u : X \rightarrow Y$ in \mathbf{C} there is a unary operator from sort Y to sort X . The corresponding operation in P is $P(u)$, and we shall write uy for $P(u)(y)$ when $y \in P(Y)$. The laws that correspond to the contravariant functoriality of P are,

$$\begin{aligned} \text{id}_X x &= x \\ (u ; v)x &= u(vx) \end{aligned}$$

We use general algebraic definitions to define the notion of homomorphism between presheaves. Let P and Q be presheaves on \mathbf{C} . Then a homomorphism $\alpha : P \rightarrow Q$ is a family of functions α_X from $P(X)$ to $Q(X)$ (indexed by the objects X of \mathbf{C}) that preserves the operations. That is, for each $u : X \rightarrow Y$ in \mathbf{C} we have $\forall y \in P(Y) . \alpha_X(uy) = u(\alpha_Y(y))$. This second condition is simply the naturality square for the family (α_X) at u , so it follows that presheaf homomorphisms are precisely natural transformations.

The idea of a presentation by generators and relations makes sense for any algebraic theory, see [Coh91], and we shall investigate it in the case of presheaves. For presheaves as many-sorted algebras the generators and relations must be sorted. That is, the set G of generators must be equipped with a function $D : G \rightarrow \text{ob } \mathbf{C}$ that assigns a sort to each generator in G . Also each relation $e_1 = e_2$ in R must relate two expressions of the same sort. Here, e_1 and e_2 denote elements of sort X of the free presheaf generated by the set G . It will turn out that the only expressions of sort X are got by taking some generator g of sort Y and

applying an operation $u : X \rightarrow Y$ so every relation is of the form $u_1 g_1 = u_2 g_2$, where each g_i is of sort Y_i and $u_i : X \rightarrow Y_i$.

Notation 2.1.1 We write $\text{PreSh}\langle G \rangle$ for the presheaf freely generated by G and $\text{PreSh}\langle G \mid R \rangle$ for the presheaf presented by generators G and relations R .

Let us recall the universal property of $\text{PreSh}\langle G \mid R \rangle$. A *model* of (G, R) in a presheaf P is an interpretation of each $g \in G$ as an element of $P(D(g))$ such that each relation $r \in R$ holds in P . The presheaf $\text{PreSh}\langle G \mid R \rangle$, itself, has a canonical, *generic* model of (G, R) , so any homomorphism α from $\text{PreSh}\langle G \mid R \rangle$ to P yields a model of (G, R) in P , by applying α to the generic model. The universal property of $\text{PreSh}\langle G \mid R \rangle$ is that this gives a bijection between, on the one hand, homomorphisms from $\text{PreSh}\langle G \mid R \rangle$ to P , and, on the other, models of (G, R) in P . Universal algebra assures us that a $\text{PreSh}\langle G \mid R \rangle$ satisfying this exists, and is unique up to isomorphism.

Example 2.1.2 Suppose \mathbf{C} is the category with two objects, X and Y , and one morphism $u : X \rightarrow Y$ (and two identity morphisms, of course). A presheaf P over \mathbf{C} is a pair of sets $P(X)$ and $P(Y)$ equipped with a function, the u operation from $P(Y)$ to $P(X)$. Suppose we present a presheaf Q by generators g_1 and g_2 (both of sort Y) subject to $u g_1 = u g_2 : Q = \text{PreSh}\langle g_1, g_2 : Y \mid u g_1 = u g_2 \rangle$. Then $Q(Y) = \{g_1, g_2\}$, and $Q(X)$ has a single element to which u maps both g_1 and g_2 . A model of the presentation in a presheaf P is a pair of elements $g'_1, g'_2 \in P(Y)$ for which $u g'_1 = u g'_2$, and it is easy to check that these correspond exactly to homomorphisms from Q to P .

It is important to understand the universal property of $\text{PreSh}\langle G \mid R \rangle$, since later it will form the basis of our account of homomorphisms between finitely presented presheaves.

Proposition 2.1.3 Let X be an object of \mathbf{C} . Then the *representable presheaf* $\mathbf{y}(X)$, defined by $\mathbf{y}(X)(Y) = \mathbf{C}(Y, X)$, is freely generated by $\text{id}_X \in \mathbf{y}(X)(X)$: in other words $\text{PreSh}\langle g : X \rangle \cong \mathbf{y}(X)$ with g corresponding to id_X .

Proof. This is precisely the Yoneda lemma. Let P be a presheaf and $\text{id}_X \mapsto g'$ a function from $\{\text{id}_X\} \rightarrow P(X)$, where $g' \in P(X)$. By Yoneda, there is a unique natural transformation (presheaf homomorphism) α from $\mathbf{y}(X)$ to P for which $\alpha_X(\text{id}_X) = g'$. \square

Proposition 2.1.4 Let G be a set, sorted by a function $D : G \rightarrow \text{ob } \mathbf{C}$, and let P_G be the free presheaf $\text{PreSh}\langle G \rangle$. Then $P_G(X) \cong \coprod_{g \in G} \mathbf{C}(X, D(g))$.

Proof. By Yoneda, P_G is the coproduct of the representables corresponding to elements of G , and it is easy to check that the coproduct of presheaves is just sortwise disjoint union. \square

Let us now investigate the effect of adding relations $e_1 = e_2$, where the expressions e_1 and e_2 can be taken to be elements of $\text{PreSh}\langle G \rangle$. From Proposition 2.1.4 we see that these relations can be written in the form $u g = u' g'$ where g, g' are in G , $u : X \rightarrow D(g)$ and $u' : X \rightarrow D(g')$ are morphisms in \mathbf{C} with common source X .

That is the general form of presentation, but we shall show how to reduce it to a diagrammatic form that presents the same presheaf (up to isomorphism) but can be expressed in the form of a diagram. The idea is that for each relation $u g = u' g'$, we introduce a new generator h with relations in a special form:

$h = ug$ and $h = u'g'$. Since the new generators are explicitly defined in terms of the old ones, they don't make any difference to the presheaf presented.

Suppose we have a presentation, with G the set of generators and R the set of relations. We can construct a new presentation (G', R') as follows. G' is the disjoint union of G and R . The sorting function $D' : G' \rightarrow \text{ob } \mathbf{C}$ agrees with D on G , and on R the sort of the relation $ug = u'g'$ is the common source X of u and u' . The set R' has two relations r_l and r_r for each relation $r \in R$. If r is $ug = u'g'$, then r_l is $\text{id}_X r = ug$, r_r is $\text{id}_X r = u'g'$.

Lemma 2.1.5 $\text{PreSh}\langle G \mid R \rangle$ is isomorphic to $\text{PreSh}\langle G' \mid R' \rangle$.

Proof. Let us write P and P' for $\text{PreSh}\langle G \mid R \rangle$ and $\text{PreSh}\langle G' \mid R' \rangle$. Define $\alpha : P \rightarrow P'$ by $\alpha(g) = g$ in G' . For a relation $ug = u'g'$ in R , we have

$$\begin{aligned} u\alpha(g) &= ug = \text{id}_X r \text{ (by } r_l) \\ &= u'g' \text{ (by } r_r) \\ &= u'\alpha(g') \end{aligned}$$

Hence α respects the relations R and defines a homomorphism. Now we define $\beta : P' \rightarrow P$ by $\beta(g) = g$, $\beta(r) = ug$ where r is $ug = u'g'$. For r_l we have,

$$\text{id}_X \beta(r) = \text{id}_X (ug) = (\text{id}_X ; u)g = ug = u\beta(g).$$

For r_r we have,

$$\text{id}_X \beta(r) = ug = u'g' = u'\beta(g').$$

Hence β defines a homomorphism.

We have $\beta(\alpha(g)) = \beta(g) = g$, so $\beta \circ \alpha = \text{id}_P$. Also $\alpha(\beta(g)) = \alpha(g) = g$ and $\alpha(\beta(r)) = \alpha(ug) = u\alpha(g) = ug = \text{id}_X r = r$, so $\alpha \circ \beta = \text{id}_{P'}$. \square

2.2. Presentations as Diagrams

In the light of Lemma 2.1.5, we can always manipulate a presentation into an equivalent one (equivalent in the sense of presenting an isomorphic presheaf) in which all relations are of the form $g = ug'$ where $u : D(g) \rightarrow D(g')$. But the data of such a presentation are just those of a diagram $D : \Gamma \rightarrow \mathbf{C}$. G is the node set of the shape graph, D interprets nodes as objects of \mathbf{C} , the edge set is R , and the formal equation $e : g = ug'$ gives e a source and target (g and g') and an interpretation $D(e) = u$ as a morphism of \mathbf{C} . (So if we start with D given, then the relations are $g = D(e)g'$ for each edge $e : g \rightarrow g'$.) Hence presentations of presheaves by generators and relations in this specialized form are simply diagrams in \mathbf{C} . Finite presentations are equivalent to finite diagrams. We shall say that such a system of generators and relations is in *diagrammatic* form. For specification purposes it turns out to be useful to have the flexibility of the general form of relation. However, to make precise the link with Oriat's work we need to devote some consideration to the diagrammatic form.

We define diagrams as graph morphisms from a shape graph [Mac71, BW90] rather than, as is often done, functors from a shape category [Ori96]. Although we could extend a graph morphism to a functor (by replacing the shape graph by its path category) there is no gain — in fact the link with presheaf theory becomes more tenuous. This will become clearer when we define morphisms between diagrams.

Proposition 2.2.1 Let $D : \Gamma \rightarrow \mathbf{C}$ be a diagram giving generators and rela-

tions G and R as above. Then $\text{PreSh}\langle G \mid R \rangle$ is a colimit of the corresponding diagram of representable presheaves.

Proof. We show that $\text{PreSh}\langle G \mid R \rangle$ is a colimit of $D ; \mathbf{y}$. A cocone $j : D ; \mathbf{y} \rightarrow Q$ comprises, first, for each node g in G a homomorphism $j(g) : \mathbf{y}(D(g)) \rightarrow Q$. That is, (by Yoneda) an element $x_g = j(g)(\text{id}_{D(g)})$ in $Q(D(g))$. Next, the commutativity part of being a cocone requires that for each edge $e : g \rightarrow g'$ in R we have $j(g) = \mathbf{y}(D(e)) ; j(g')$. But for this equation to hold, it suffices — again by Yoneda — that the two sides should agree on $\text{id}_{D(g)}$. That is,

$$x_g = j(g)(\text{id}_{D(g)}) = j(g')(\mathbf{y}(D(e))(\text{id}_{D(g)})) = j(g')(D(e)) = D(e)x_{g'}$$

Hence describing a cocone from $D ; \mathbf{y}$ is equivalent to describing a model of (G, R) , and it follows by comparing universal properties that $\text{PreSh}\langle G \mid R \rangle$ is a colimit of $D ; \mathbf{y}$. \square

Proposition 2.2.2 Let G and R be generators and relations in diagrammatic form, and let us write P_G, P for $\text{PreSh}\langle G \rangle$ and $\text{PreSh}\langle G \mid R \rangle$. For each X in $\text{ob } \mathbf{C}$, define a relation \sim on $P(X)$ by $ug \sim vh$ iff there is some w such that $v = u ; w$ and $g = wh$ is a relation in R . Let \equiv be the equivalence relation generated by \sim . Then \equiv is a congruence, and P_G / \equiv is isomorphic to P .

Proof. If $ug \sim vh$ then clearly $tug \sim tvh$ for all t , from which it follows that \equiv is a congruence and P_G / \equiv is a presheaf. If $g = wh$ is a relation in R then $g \sim wh$, so the congruence classes of the generators give a model for (G, R) in P_G / \equiv and hence a homomorphism β from P to P_G / \equiv . Now let α be the obvious homomorphism from P_G to P , then $\alpha(g) = g$. If $ug \sim vh$, with w as in the definition, then $\alpha(ug) = u\alpha(g) = ug = uwh = vh = v\alpha(h) = \alpha(vh)$, and it follows that α factors via α' from P_G / \equiv to P . The homomorphisms α' and β are mutually inverse, as is readily checked by their actions on the generators. \square

2.3. Diagram Morphisms and Zigzags

There is an obvious notion of morphism between diagrams that corresponds to natural transformations between functors: if $D : \Gamma \rightarrow \mathbf{C}$ and $D' : \Gamma' \rightarrow \mathbf{C}$ are the diagrams, then it comprises a graph morphism $T : \Gamma \rightarrow \Gamma'$ with morphisms $\alpha_g : D(g) \rightarrow D'(T(g))$, for each $g \in G$, such that for each edge $e : g \rightarrow g'$ in R , $\alpha_g ; D'(T(e)) = D(e) ; \alpha_{g'}$. Unfortunately, as Oriat shows [Ori96], it is inadequate for constructing a diagram category that can serve as the finite cocompletion of \mathbf{C} . She corrects this with two mechanisms. First she generalizes the diagram morphisms using “zigzags”. This gains enough morphisms, but they are too concrete. The second mechanism is to define an equivalence relation “equiv” on them. (Actually equiv is a rather special case of 2-category structure.) The following definitions are based on Oriat’s in [Ori96].

Definition 2.3.1 (A Zigzag) Let Γ be a graph. A *zigzag* Z of Γ is a tuple (k, Z_N, Z_E) where,

- $k \in \mathbb{N}$ is the *length* of Z
- Z_N is a sequence $\langle g_0, \dots, g_k \rangle$ of length $(k + 1)$ of nodes of Γ
- Z_E is a sequence $\langle \epsilon_0, \dots, \epsilon_{k-1} \rangle$ of length k such that for $0 \leq i \leq k - 1$, either

ϵ_i is an edge e_i of Γ with $e_i : g_i \rightarrow g_{i+1}$ or ϵ_i is a ‘reversed edge’ \bar{e}_i with $e_i : g_{i+1} \rightarrow g_i$

That is, each edge e_i has an arbitrary orientation as in the zigzag,

$$g_0 \xrightarrow{e_0} g_1 \xleftarrow{e_1} g_2 \xleftarrow{e_2} g_3 \cdots g_{k-1} \xrightarrow{e_{k-1}} g_k$$

that is denoted by $Z : g_0 \rightsquigarrow g_k$.

For each node g of Γ there is an empty zigzag $0_g : g \rightsquigarrow g$. We denote the graph with the same node set G as Γ and with edges that are zigzags of Γ by $Zigzag(\Gamma)$.

$Zigzag(\Gamma)$ is actually a category — zigzags can be composed. However, it has an additional structure in that zigzags can be *reversed*. Abstractly, we can see this as an *involution* on the category.

Definition 2.3.2 (An Involution) An *involution* on a category \mathbf{C} is a functor $\tau : \mathbf{C} \rightarrow \mathbf{C}^{\text{op}}$ that is the identity on objects, and for which $\tau ; \tau^{\text{op}} = \text{Id}_{\mathbf{C}}$. (\mathbf{C}, τ) is an *involution category*. An *involution functor* is a functor that preserves involution. We shall not prove this here, but it is easy to show that $Zigzag(\Gamma)$ is the free involution category over the graph Γ .

Definition 2.3.3 (A Generalized Graph Morphism) A *generalized graph morphism* T from graph Γ to Γ' , denoted by $T : \Gamma \rightsquigarrow \Gamma'$ is a graph morphism from Γ to $Zigzag(\Gamma')$.

By the universal property mentioned above, we see that generalized graph morphisms from Γ to Γ' are equivalent to involution functors from $Zigzag(\Gamma)$ to $Zigzag(\Gamma')$, and it follows that they can be composed. In fact, the category of graphs and generalized graph morphisms is just the Kleisli category for the $Zigzag$ monad.

Before defining morphisms between diagrams we need a further notion.

Definition 2.3.4 (Connecting Morphisms by a Zigzag in a Diagram)

Let Γ be a graph. Let $D : \Gamma \rightarrow \mathbf{C}$ be a diagram in \mathbf{C} and g, g' be nodes of Γ . Let $u, u' \in \text{mor } \mathbf{C}$ where $u : X \rightarrow D(g)$ and $u' : X \rightarrow D(g')$ and let $Z : g = g_0 \rightsquigarrow g' = g_k$ be a zigzag in Γ , through nodes g_i as in Definition 2.3.1. Then a *connection* from u to u' by Z is a sequence of morphisms $c_i : X \rightarrow D(g_i)$, $(0 \leq i \leq k)$ such that $u = c_0, u' = c_k$ and for $0 \leq i \leq k-1$,

$$\begin{aligned} c_i ; D(e_i) &= c_{i+1} \text{ if } \epsilon_i = e_i : g_i \rightarrow g_{i+1} \\ c_{i+1} ; D(e_i) &= c_i \text{ if } \epsilon_i = \bar{e}_i, e_i : g_{i+1} \rightarrow g_i \end{aligned}$$

When such a connection exists we say that u is *connected* to u' by Z and write $u \sim_D u'[Z]$. The morphisms c_i can be viewed as a cone from $X \in \text{ob } \mathbf{C}$ to the diagram corresponding to the zigzag Z .

Lemma 2.3.5 Let $D : \Gamma \rightarrow \mathbf{C}$ be a diagram with node and edge sets G and R . Let $P = \text{PreSh}(G \mid R)$. Then $ug = u'g'$ in P iff u is connected to u' by a zigzag from g to g' .

Proof. Let P_G be the free presheaf $\text{PreSh}(G)$. By Proposition 2.2.2, $ug = u'g'$ in P iff $ug \equiv u'g'$ in P_G , where \equiv is the equivalence closure of \sim . In other words, there are elements $c_i h_i$ of P_G , where $0 \leq i \leq k$, with $u = c_0, g = h_0, u' = c_k, g' = h_k$, and for every i either $c_i h_i \sim c_{i+1} h_{i+1}$ or $c_{i+1} h_{i+1} \sim c_i h_i$. In the former case we have $e_i : h_i \rightarrow h_{i+1}$ with $c_i ; D(e_i) = c_{i+1}$. The latter case is similar but the

other way round. This sequence of edges gives us a zigzag from g to g' , and the morphisms c_i give the connection from u to u' along it. \square

Morphisms between diagrams are now defined. We make only minor modifications to the definition by Oriat [Ori96] which is based on a path category.

Definition 2.3.6 (A Diagram Morphism) Let $D : \Gamma \rightarrow \mathbf{C}$ and $D' : \Gamma' \rightarrow \mathbf{C}$ be diagrams over \mathbf{C} . Then a *morphism*, F , from D to D' is a pair (T, σ) where $T : \Gamma \rightsquigarrow \Gamma'$ is a generalised graph morphism and σ assigns to each node g in Γ a morphism $\sigma_g : D(g) \rightarrow D'(T(g))$ of \mathbf{C} , in such a way that for every edge $e : g \rightarrow h$ in Γ , the morphisms σ_g and $D(e)$; σ_h are connected by the zigzag $T(e)$ in the graph Γ' .

Oriat shows that given \mathbf{C} , the diagrams and diagram morphisms form a category. We denote this category by $\text{Diag}(\mathbf{C})$. She also defines an equivalence between diagram morphisms, and the following definition is a modification of hers.

Definition 2.3.7 (Equivalence \approx) Let $D : \Gamma \rightarrow \mathbf{C}$ and $D' : \Gamma' \rightarrow \mathbf{C}$ be diagrams over \mathbf{C} , and let $F, F' : D \rightarrow D'$ be diagram morphisms defined by the pairs (T, σ) and (T', σ') respectively. Then $F \approx F'$ if for all nodes g of Γ , σ_g and σ'_g are connected in the diagram D' by a zigzag.

Oriat shows that \approx is a congruence relation in the category $\text{Diag}(\mathbf{C})$, and it makes $\text{Diag}(\mathbf{C})$ into what she calls an *equiv-category*.

Theorem 2.3.8

Let $\text{Diag}(\mathbf{C})$ be the equiv-category of diagrams and let $\text{PreSh}_{\text{fp}}(\mathbf{C})$ be the full subcategory of $\mathbf{Set}^{\mathbf{C}^{\text{op}}}$ whose objects are the finitely presented presheaves. Then *colim* gives a functor from $\text{Diag}(\mathbf{C})$ to $\text{PreSh}_{\text{fp}}(\mathbf{C})$ that is essentially surjective, full and ‘equiv-faithful’ — by ‘equiv-faithful’ we mean that for any morphisms $F, F' : D \rightarrow D'$ in $\text{Diag}(\mathbf{C})$ then $\text{colim} F = \text{colim} F'$ in $\text{PreSh}_{\text{fp}}(\mathbf{C})$ iff $F \approx F'$.

Proof. On objects the functor *colim* takes the colimit of the corresponding diagram of representable presheaves. By Proposition 2.2.1 this is just $\text{PreSh}(G \mid R)$, and by Lemma 2.1.5 any finitely presented presheaf is isomorphic to one of this form, so that proves that *colim* is essentially surjective.

As for morphisms, let $F = (T, \sigma)$ be a diagram morphism from D to D' (we shall use the usual notation Γ, G, R, P , etc. as in Lemma 2.3.5). For each node g of Γ we can define $\alpha(g) = \sigma_g T(g) \in P'$. For each edge $e : g \rightarrow g'$ of Γ we have σ_g connected to $D(e)$; $\sigma_{g'}$ by $T(e)$, and it follows from Lemma 2.3.5 that $\sigma_g T(g) = D(e) \sigma_{g'} T(g')$ in P' , in other words that e is respected by α , so $\alpha = \text{colim} F$ is a homomorphism from P to P' .

If F is the identity diagram morphism $(\text{id}_{\Gamma}, (g \mapsto \text{id}_{D(g)}))$ then $\alpha(g) = g$ and so α is the identity homomorphism on P . Now suppose F is general and we also have $F' : D' \rightarrow D''$ with composite $F'' = F ; F'$. We have $T''(g) = T'(T(g))$ and $\sigma''_g = \sigma_g ; \sigma'_{T(g)}$, so $\alpha''(g) = \sigma_g \sigma'_{T(g)} T'(T(g)) = \sigma_g \alpha'(T(g)) = \alpha'(\sigma_g T(g)) = \alpha'(\alpha(g))$. This proves the functoriality of *colim*.

The argument can be reversed to show fullness. Suppose we have a homomorphism α from P to P' . By the universal property of P , α corresponds to a model of (G, R) in P' , so for each g in G we have a corresponding $\alpha(g)$ in P' , which must be of the form (not necessarily unique) vh , where h is in G' and $v : D(g) \rightarrow D'(h)$ is in $\text{mor } \mathbf{C}$. We thus have, for each g in G , some $T(g) = h$

in G' and some $\sigma_g = v : D(g) \rightarrow D'(T(g)) \in \text{mor } \mathbf{C}$. We also have, for each relation e in R , that $\sigma_g T(g) = D(e) \sigma_{g'} T(g')$ in P' , so by Lemma 2.3.5 we have a zigzag $T(e)$, say, from $T(g)$ to $T(g')$ in Γ' , and a connection by it from σ_g to $D(e) ; \sigma_{g'}$. Then T is a generalized graph morphism from Γ to Γ' , and (T, σ) is a diagram morphism for which $\text{colim}(T, \sigma) = \alpha$.

To prove that colim is equiv-faithful we have to show that $\text{colim} F$ and $\text{colim} F'$ are equal as morphisms in $\text{PreSh}_{\text{fp}}(\mathbf{C})$ iff $F \approx F'$ in $\text{Diag}(\mathbf{C})$. Let $\text{colim} F$ be α and $\text{colim} F'$ be α' such that $\alpha, \alpha' : P \rightarrow P'$. Equality between α and α' means that they agree on the mapping of the generators in P . In other words, for each $g \in G$,

$$\begin{aligned} \alpha &: g \mapsto \sigma_g T(g) \\ \alpha' &: g \mapsto \sigma'_g T'(g) \end{aligned}$$

and $\sigma_g T(g) = \sigma'_g T'(g)$ in P' . By Lemma 2.3.5, this holds if and only if σ_g and σ'_g are connected by a zigzag in the diagram D' , and this is the definition of $F \approx F'$.

We have therefore proved that colim is equiv-faithful. \square

3. Instances

The use of colimits to build complex specifications out of simpler ones dates back to the work of Burstall and Goguen [BG77] and Oriat [Ori96] has developed the use of diagrams to express these colimits. Our results of Section 2 show that these can equivalently be interpreted in terms of presheaves, but one might very reasonably question what has been achieved by this. Our goal in this section is to show that the presheaf formulation fits naturally with specificational ideas of configuration by components with sharing.

3.1. Specifications and their Models

Let us first review the way that theories and theory morphisms are used as specifications. A logical theory can be taken as specifying a class of structures, namely its models: the theory specifies that for a model you want carriers with a particular set of functions and predicates, satisfying a set of axioms. An interpretation $i : T_1 \rightarrow T_2$ between theories (using any one of a number of more or less liberal definitions) shows how the ingredients of T_1 can be interpreted in T_2 ; but it also shows how from any model of T_2 can be extracted a model of T_1 . This is known as model *reduction*. As an example from algebra, the theory of monoids can be interpreted in the theory of rings in two different ways, one as the additive structure and one as the multiplicative. Correspondingly, any ring can be considered as (‘reduced to’) a monoid in two different ways, using its additive or multiplicative structure.

Such models are formal set-theoretic constructions. In specification we are interested in specifying systems in the real world, but nevertheless informally we can still imagine such systems as being specified by theories, asserting our interest in systems with certain collections (as informal carriers of the sorts), transformations (for functions) and properties (for predicates). We shall use the term *instance* for such an informal, real-world model.

3.2. Instance Reduction

We shall leave aside here any deeper discussion of the nature of instances of first-order theories, though it has been argued elsewhere [Vic95] that not all of classical logic — or, indeed, intuitionistic logic — has sufficient physical content and the first-order theories considered should be *geometric* theories. Let us instead just take an informal postulate that we work over a category \mathbf{C} , and that each object of \mathbf{C} has an associated notion of instance. We think of each object of \mathbf{C} as a specification and of each instance as associated with its specification.

We shall need two additional primitive informal notions. The first is that of *equality* between instances. By this we mean physical identity, so that two peas in a pod (or oven-control instances in a box) may be totally indistinguishable but are not considered equal. The second is that of *instance reduction*, analogous to model reduction, by which instances of one class can be extracted from instances of another.

In working over a category \mathbf{C} , we take an additional informal postulate that each morphism of \mathbf{C} corresponds contravariantly to a notion of instance reduction. If $f : X \rightarrow Y$ in \mathbf{C} , and I is an instance for Y , then we shall write fI for the corresponding reduct for X .

Example 3.2.1 Consider a system of lifts in an office building. We suppose *Lift* is a class of instances — the instances are the actual lifts. Each lift has (amongst other things) a series of call buttons, so there could be a class *CButton* whose instances are series of call buttons. There is instance reduction from *Lift* instances to *CButton* instances: given any lift, we can point out its call buttons and so display a *CButton* instance.

In terms of first-order theories we might well have theories for *Lift* and *CButton* with an interpretation from *CButton* to *Lift* for which the reduction is as described. However, for our present purposes there is no need to do this. We shall simply assume that we have a category \mathbf{C} of primitive specifications and that it includes objects *Lift* and *CButton* with a morphism $i : \mathbf{CButton} \rightarrow \mathbf{Lift}$. If L is a *Lift* instance, then iL will denote its call buttons.

Now suppose we want two lifts, working side by side. Naturally they should share their call buttons, so we can specify this *configuration* by saying we want:

- an instance L_1 for *Lift*
- an instance L_2 for *Lift*
- such that $iL_1 = iL_2$

But this is formally equivalent to presenting a presheaf over \mathbf{C} ! The generators are L_1 and L_2 , and the one relation is $iL_1 = iL_2$. Example 2.1.2 shows its concrete structure with two elements in the carrier for *Lift* and one in that for *CButton*. We can express it in the diagrammatic form as:

- an instance L_1 for *Lift*
- an instance L_2 for *Lift*
- an instance CB for *CButton*
- such that $CB = iL_1$
- and $CB = iL_2$

Lemma 2.1.5 shows that this presents an isomorphic presheaf.

We thus see that the notation of presheaf presentations is just what we need

for describing systems as configured from a number of components (instances, corresponding to generators) with some sharing (corresponding to relations).

A point worth noting is that although we can specify equality between instances, we can’t specify *inequality*. Merely specifying two instances (such as lifts L_1 and L_2) does not in itself force them to be different. Technically this is simply an inherent limitation of what can be expressed using the colimits. With generators and relations (G, R) , distinct generators can always get mapped to the same element in a model. Specificationally, perhaps the way to view this is by plain economics. If one lift is sufficient, why should we use two? Maybe there are good reasons — carrying capacity perhaps — why one is not enough, but these could be made explicit and not left to the configuration calculus.

3.3. Instances for Configured Specifications

Provided we know what the instances are for each object of the category \mathbf{C} , we can use this knowledge to say that an instance for a configured specification (presheaf presentation) comprises a selection of instances as specified that satisfy the equations given (for instance, two *Lift* instances sharing a *CButton* instance). To put this on a slightly more formal footing, imagine that we do have a set-theoretic description of all the instances for each object of \mathbf{C} . This amounts to a presheaf over \mathbf{C} , *Stock*, say, describing all available instances. We choose a presheaf because we need to know the instance reductions. If another presheaf P is presented by generators and relations, then an instance for the configured specification P is simply a presheaf homomorphism from P to *Stock*. Moreover, if we have another presheaf P' , and a homomorphism $f : P \rightarrow P'$, then precomposition with f gives a uniformly defined instance reduction from instances for P' to instances for P . The actual value of *Stock* here was irrelevant; the general argument shows if we thought \mathbf{C} originally related informally to some idea of physically existing instances, then from Theorem 2.3.8 we can reasonably suppose that $\text{PreSh}_{\text{fp}}(\mathbf{C})$ does too. This can be repeated to give general specifications constructed by repeated acts of configuration.

4. A Configuration Language

A system configuration is a description of how a system is built from its component parts and we shall describe this using a language that reflects the primitive notions of “instance of a specification”, instance reduction, and equality of instances. A configuration in the language asserts the requirement of certain components with certain equalities and sharing amongst them: in brief, it describes how instances are combined with sharing.

At the same time, such configurations are formally equivalent to presentations of presheaves by generators and relations, and this enables us to view them (using the results of Section 2) as colimits of specifications. Consequently, the diagrams of Oriat come to be seen rather concretely as configurations. As we shall see, this is even more striking with the diagram morphisms, which are seen as describing how to implement one kind of configuration as part of another.

From a straightforward reading of Section 2 we might consider that we start off with a category \mathbf{C} of “specifications” and construct a new category $\text{Config}(\mathbf{C})$ of “configurations”, constructed out of instances for specifications in

\mathbf{C} — that is, in fact, how we shall develop this section. However, the arguments of this section suggest that configurations themselves can be regarded as “configured” specifications. Certainly we can explain what their instances are and what the reduction along diagram morphisms is. We therefore take the view that configuration “configures new specifications out of old ones” and can be iterated. An iterated construction is an explicitly hierarchical, unflattened specification, and our semantic aim is to express, independently of the logic, this hierarchical structure that does so much to aid comprehension of the human reader by grouping the information into “mind-sized bites” (to use Seymour Papert’s phrase in [PAP80]). The categorical formalization of this has already been suggested in [Hil95, Hil97] and much further work is required to elaborate it rigorously in the presheaf setting. For the moment, note that the syntax of the language is not intended to distinguish between primitive specifications and configurations (nor between the corresponding morphisms).

The language treats three concepts, namely the categorical objects, morphisms and equality between morphisms, and these are described in the following three subsections. We shall not give a detailed syntax but outline it by simple examples.

4.1. Configured Specifications

The categorical objects are the configured specifications. These correspond to presheaf presentations or diagrams.

Example 4.1.1 Consider Example 3.2.1. The configuration described there would appear in our language (using the diagrammatic form of presentation) as:

```
spec Diag.Twolifts is
  instances
    CB : CButton ;
    L1 : Lift ;
    L2 : Lift ;
  equations
    e1: CB = i L1 ;
    e2: CB = i L2
endspec
```

or, equivalently and more briefly (in the general form of presentation) as:

```
spec Twolifts is
  instances
    L1 : Lift ;
    L2 : Lift ;
  equations
    e1: i L1 = i L2
endspec
```

If the specifications of the component parts and morphisms between the specifications are objects and morphisms of some category \mathbf{C} (this is non-trivial if we allow iterated configuration but we defer the question to further study) then a configuration describes generators and relations for a presheaf. The generators are the instances, the sorting function is implicit in their declarations, and their

relations are as described. Note that we have labelled the relations. We shall see the labels in use when we look at configuration morphisms.

4.2. Configuration Morphisms

We now turn to morphisms between configured specifications. Formally these correspond to presheaf homomorphisms or diagram homomorphisms. For specifications they correspond to implementations of one configuration in another (and in fact we shall call them implementations). Equivalently, but in the reverse direction, they also correspond to descriptions of instance reductions.

Recall the discussion of Section 2. If $P = \text{PreSh}\langle G \mid R \rangle$ and $P' = \text{PreSh}\langle G' \mid R' \rangle$, then a homomorphism from P to P' is defined by two things. First, we define an assignment $M \mapsto fM'$ for each $M \in G$. (We have changed the notation to match our view of generators as instances — $M' \in G'$ and $f \in \text{mor } \mathbf{C}$). Second, for each relation $M = hN$ in R , we must show that the equation $fM' = hgN'$ (where $N \mapsto gN'$ is the assignment for N) holds in P' . This equation will not in general be itself in R' , but must be deduced from those that are using a chain of equations linking fM' to hgN' . Each link can be:

1. $uK' = uvL'$ for some relation $K' = vL'$ in R' (such a link corresponds to \sim in Proposition 2.2.2), or
2. the same but with the equation the other way round, that is, $uvL' = uK'$ (corresponding to \sim^{op}), or
3. $uK' = vK'$ where the equality of morphisms $u = v$ is already known (corresponding to nothing in Proposition 2.2.2, since it is already equality in the free presheaf $\text{PreSh}(G')$).

If the presentations are treated as configured specifications, S and S' , then a homomorphism is to be thought of as an *implementation* of S by S' — that is to say, any instance for S' also implements S , because it contains within itself an instance for S . To describe this implementation we must say how the individual ingredients of S are interpreted in S' (by the assignments $M \mapsto fM'$) and discharge a proof obligation that the equations in S are respected by these assignments. Our syntax for configuration morphisms therefore includes both assignments and proof, as in the following example:

Example 4.2.1 (from Example 4.1.1) We define a morphism from the configuration *Twolifts* to the configuration, in diagrammatic form, *Diag_Twolifts*. The definition gives the assignment of the instance components of *Twolifts* to the instance components of *Diag_Twolifts*, as well as checking that the equation in *Twolifts* is respected by these assignments.

implementation h : *Twolifts* \rightarrow *Diag_Twolifts*

$L_1 \mapsto L_1$;

$L_2 \mapsto L_2$;

To check *e1* of *Twolifts*:

$i L_1 \mapsto i L_1$

= CB by *e1* backwards of *Diag_Twolifts*

= $i L_2$ by *e2* forwards of *Diag_Twolifts*

$\leftarrow i L_2$

endimp

The proof that the equation $i L_1 = i L_2$ in *Twolifts* is respected by the assignment of instances to *Diag_Twolifts*, is by a deduction that uses the two equations given in *Diag_Twolifts*: $e1$ is used in a backwards direction; $e2$ is used in a forwards direction. The symbol \mapsto denotes the assignment from *Twolifts* to *Diag_Twolifts* of the instance on the left hand side of $e1$ of *Twolifts*. The deduction is then made from the chain of equations in *Diag_Twolifts*. Finally the symbol \leftarrow denotes the assignment from $i L_2$ in *Twolifts* to $i L_2$ in *Diag_Twolifts*.

Our configuration morphisms are strikingly different from the diagram morphisms of Oriat [Ori96], even though they are mathematically equivalent, for the presheaf perspective has made us redistribute the structure. Recall that a diagram morphism (T, σ) comprised a generalized graph morphism T (involving zigzags) and a correspondingly generalized natural transformation. When we compare this with the presheaf homomorphism structured as (assignments, proof), we find that the structurings go in different directions. The assignments comprise the node part of T together with the components σ_g of σ , while the proof comprises the edge part of T (with zigzags corresponding to chains of equations) and the generalized naturality conditions of σ .

At the object level, the switch to presheaf presentations was relatively shallow mathematically, but for the morphisms it has appreciably clarified the way we handle them.

4.3. Equivalence between Configuration Morphisms

We know from presheaf theory that two of Oriat's diagram morphisms may correspond to the same presheaf homomorphism: Theorem 2.3.8 gives in detail the correspondence between equality of presheaf homomorphisms and equivalence between diagram morphisms.

In this section we define a notion of equivalence between configuration morphisms in the equiv-category of configurations, $\text{Config}(\mathbf{C})$. The point of defining $\text{Config}(\mathbf{C})$ as an equiv-category is to distinguish between:

- syntactically equal configurations, with the same names for instances and morphisms, although the order of declarations may differ, and
- semantically equal configurations, which can be proved to have the same meaning.

Definition 4.3.1 (Equivalence)

Let $S, S' \in \text{ob } \text{Config}(\mathbf{C})$, and $f, f' \in \text{mor } \text{Config}(\mathbf{C})$ such that $f, f' : S \rightarrow S'$. Then $f \approx f'$ if for all instances M declared in S , $f : M \mapsto g M'_1$ and $f' : M \mapsto g' M'_2$ and we can prove, using the relations in S' , that $g M'_1 = g' M'_2$.

The notion of proof in this definition corresponds to the connection by a zigzag for the diagram morphisms in Definition 2.3.7.

Example 4.3.2 (from Example 4.2.1) The following morphisms k and l are equivalent:

implementation $k: \text{Diag_Twolifts} \rightarrow \text{Twolifts}$

$L_1 \mapsto L_1$;
 $L_2 \mapsto L_2$;
 $CB \mapsto i L_1$;

```

To check e1 of Diag_Twolifts:
CB  $\mapsto i L_1$ 
   $\leftarrow i L_1$  ;
To check e2 of Diag_Twolifts:
CB  $\mapsto i L_1$ 
  =  $i L_2$  by e1 forwards of Twolifts
   $\leftarrow i L_2$ 
endimp

```

implementation l : $\text{Diag_Twolifts} \rightarrow \text{Twolifts}$

```

 $L_1 \mapsto L_1$  ;
 $L_2 \mapsto L_2$  ;
CB  $\mapsto i L_2$  ;
To check e1 of Diag_Twolifts:
CB  $\mapsto i L_2$ 
  =  $i L_1$  by e1 backwards of Twolifts
   $\leftarrow i L_1$  ;
To check e2 of Diag_Twolifts:
CB  $\mapsto i L_2$ 
   $\leftarrow i L_2$ 
endimp

```

The morphism k assigns $\text{CB} \mapsto i L_1$, whereas l assigns $\text{CB} \mapsto i L_2$. But in *Twolifts* the equation $i L_1 = i L_2$ gives the equivalence between k and l .

The reader might like to prove as an exercise that $h ; k = \text{Id}_{\text{Twolifts}}$ and $k ; h \approx \text{Id}_{\text{Diag_Twolifts}}$. This shows that the configurations *Diag_Twolifts* and *Twolifts* are equivalent (presenting isomorphic presheaves).

In future work we shall define $\text{Config}(\mathbf{C})$ as a hierarchical structure, with unflattened configurations as objects. This will involve the definition of configuration morphisms that cross the levels of structuring, both upwards and downwards.

5. Conclusions

We have started with Oriat’s general work on using colimits, but replaced her diagrams with presheaf presentations. We have shown these to be mathematically equivalent, but the presheaves have a number of advantages.

- The presheaves allow the extra flexibility and convenience of using the “non-diagrammatic” presentations, which are usually simpler than their diagrammatic equivalents.
- The presheaf homomorphisms, though equivalent to diagram morphisms, gain some simplicity by rearranging the data that defines them.
- Presheaf presentations correspond naturally to the idea of specifying configurations as made of components with sharing of subcomponents.

We have outlined a configuration language based on presheaf presentations, but further work is needed to extend it to allow nested configurations and also to give it a formal syntactic definition.

Acknowledgments

Paul Taylor's macros were used for the diagram.

References

- [BG77] Burstall, R. M. and Goguen, J. A.: Putting Theories Together to Make Specifications. *Proceedings of the 5th. International Joint Conference on Artificial Intelligence*, Cambridge, Mass., pp. 1045–1058, 1977.
- [BG79] Burstall, R. M. and Goguen, J. A.: The Semantics of Clear, A Specification Language. *Abstract Software Specifications, LNCS 86*. Springer-Verlag, 1979.
- [BW90] Barr, M. and Wells, C.: *Category Theory for Computing Science*. Prentice Hall International, 1990.
- [Coh91] Cohn, P. M.: *Algebra*. Chichester: Wiley, second edition, volume 2, 1991.
- [EFHLP87] Ehrig, H. and Fey, W. and Hansen, H. and Lowe, M. and Papis-Prsicce, F.: Algebraic Theory of Modular Specification Development. Technical University, Berlin, 1987.
- [EM90] Ehrig, H. and Mahr, B.: *Fundamentals of Algebraic Specification 2: Module Specifications and Constraints*. Springer-Verlag, 1990.
- [Hil95] Hill, G.: Constructing Specifications and Modules in a KZ-Doctrine, *Theory and Formal Methods '94, Proceedings of the Second Imperial College, Department of Computing, Workshop on Theory and Formal Methods*, September, Hankin, C. L., Mackie, I. and Nagarajan, R., editors, Imperial College Press, distributed by World Scientific Publishing Co. ISBN 1-86094-003-X, pp 219–236, 1995.
- [Hil97] Hill, G.: A Categorical Workspace for System Configuration, *Advances in Theory and Formal Methods of Computing, Proceedings of the Third Imperial College Workshop, Christchurch, Oxford, 1–3 April 1996*, Edalat A., Jourdan, S. and McCusker, G., editors, Imperial College Press, distributed by World Scientific Publishing Co., 1997.
- [Mac71] MacLane, S.: *Categories for the Working Mathematician*, Springer-Verlag, 1971.
- [MFS90] Maibaum, T. and Fiadeiro, J. and Sadler, M.: Stepwise Program Development in II Institutions, Imperial College, November, 1990.
- [MM92] MacLane, S. and Moerdijk, I.: *Sheaves in Geometry and Logic: A First Introduction to Topos Theory*, Springer-Verlag, 1992.
- [Ori96] Oriat, C.: *Modular Specifications: Constructions with Finite Colimits, Diagrams, Isomorphisms*, LSR-IMAG, 681 rue de la Passerelle, 38402 Saint Martin d'Herès Cedex, RR 964-I-LSR 3, 1996.
- [PAP80] Papert S.: *Mind-Storms*, The Harvester Press, 1980.
- [SJ95] Srinivas, Y. V. and Jellig, R.: Formal Support for Composing Software. *Proceedings of the Conference on the Mathematics of Program Construction, Kloster Irsee, Germany*, Kestrel Institute Technical Report KES.U.94.5, July, 1995.
- [Vic95] Vickers, S. J.: Geometric Logic as a Specification Language, *Theory and Formal Methods '94, Proceedings of the Second Imperial College, Department of Computing, Workshop on Theory and Formal Methods*, September, Hankin, C. L., Mackie, I. and Nagarajan, R., editors, Imperial College Press, distributed by World Scientific Publishing Co. ISBN 1-86094-003-X, 1995.