Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 264 (2010) 63–81
www.elsevier.com/locate/entcs

Families of Symmetries as Efficient Models of Resource Binding

Vincenzo Ciancia[1]

Institute for Logic, Language and Computation - Amsterdam (NL)

Alexander Kurz[2]

University of Leicester (UK)

Ugo Montanari[3]

Università di Pisa (IT)

Abstract

Calculi that feature resource-allocating constructs (e.g. the pi-calculus or the fusion calculus) require special kinds of models. The best-known ones are presheaves and nominal sets. But named sets have the advantage of being finite in a wide range of cases where the other two are infinite. The three models are equivalent. Finiteness of named sets is strictly related to the notion of finite support in nominal sets and the corresponding presheaves. We show that named sets are generalisd by the categorical model of families, that is, free coproduct completions, indexed by symmetries, and explain how *locality of interfaces* gives good computational properties to families. We generalise previous equivalence results by introducing a notion of minimal support in presheaf categories indexed over small categories of monos. Functors and categories of coalgebras may be defined over *families*. We show that the final coalgebra has the greatest possible symmetry up-to bisimilarity, which can be computed by iteration along the terminal sequence, thanks to finiteness of the representation.

Keywords: Presheaves, Families, Named Sets, History-dependent Automata, Coalgebras, Symmetry Reduction, Partition Refinement

1 Introduction

Full abstraction and nominal calculi. One of the greatest concerns in programming language semantics is to find *fully abstract* models, where all the semantically

[1] Research supported by the Comunidad de Madrid program *PROMESAS* (S-0505/TIC/0407), and by the VICI grant 639.073.501 of the Netherlands Organization for Scientific Research (NWO)

[2] Research partially supported by EPSRC EP/G041296/1

[3] Research partially supported by the EU FP6-IST IP 16004 project *SENSORIA*

doi:10.1016/j.entcs.2010.07.014

equivalent programs are identified. A difficult question is how to do this for the so-called *interactive systems*, where the focus is not the final result of the computation, but rather on the interactions with the environment along the possibly non-terminating *behaviour* of a system. For languages such as the CCS [35] or the π-calculus [36], the operational semantics is expressed in terms of *labelled transition systems* (LTS), and the fully abstract model is the quotient of all the possbile systems with respect to *bisimilarity*.

Calculi with resource allocation mechanisms (the so called *nominal calculi*) typically have a notion of bisimulation that does not coincide with the standard one over LTS. Thus, standard definitions and algorithms can not be reused. This is solved by resorting to *presheaf categories*, that is, categories of functors from a small category $\mathtt{C}$ to $\mathtt{Set}$ (see [23,10,9,24,34,33], and the foundational work by Moggi [37]), or to *nominal sets* [25] as done in [38]. Presheaves handle names, and in general resources, as having a *global* meaning across all possible processes. Thus, each freshly generated name must be different from all the previous ones, giving rise to infinite states in the presence of loops. Therefore, the operational semantics of a calculus typically has infinite states even for very simple processes, making it difficult to *compute* the abstract semantics, or to *implement* finite state methods, such as minimisation, equivalence checking or model checking.

Named sets. In the parallel research line of *named sets* [40,41], these difficulties were overcome using *local* names; in this case, establishing a binding between names of elements is necessary whenever two elements are related. This machinery allows one to reuse previously generated names that have been discarded. In [41], many formalisms (e.g. Petri nets and process calculi) have been mapped into named sets in a fully abstract way. The most important finding here is that modelling the *symmetry group* of each agent is necessary to have a unique abstract model of the π-calculus, leading to [20,43,21], where a coalgebraic minimsation (partition refinement) algorithm for the π-calculus has been implemented, based on *history-dependent automata*, that is, coalgebras in the category of *named sets*. The importance of modelling symmetries is recognised both in the theory of programming language semantics [45] and in practical applications such as model checking [18]. Due to well known results of group theory (in particular Lagrange's theorem, see e.g. [17], §3.3), finite groups have an efficient representation in terms of generators, which is logarithmic with respect to the size of the group. Moreover, many operations on groups can be computed on the compressed representation [32].

The categorical equivalence between nominal sets, named sets and the *pullback-preserving* full subcategory [4] of $\mathtt{Set}^{\mathtt{I}}$, called the *Schanuel topos*, has been established in [27,22]. In [12,13], a number of *ad-hoc* constructions on named sets used for the π-calculus are turned into categorical notions such as products, coproducts, the power set and name abstraction, thus allowing one to reuse the same machinery to represent the semantics of other calculi with names.

Our contribution. An advantage of presheaf categories is the flexibility that can be obtained by varying the index category $\mathtt{C}$, giving rise more complex struc-

[4] Here $\mathtt{I}$ is the category of finite subsets of the natural numbers and injections between them.

tures than *pure names* (see e.g. [28], or [3]). This flexibility is lost when using named sets, since the index category is fixed to be $\mathtt{I}$. First, in §2 we introduce *families* as concrete representation of free coproduct completions. Our contribution starts in §3 observing that named sets with symmetries are generalised by the categorical model of *families* over a category of groups of automorphisms and related morphisms, that we call $\mathtt{Sym}(\mathtt{C})$. This model is equivalent in the categorical sense to a full subcategory of $\mathtt{Set}^{\mathtt{C}}$, namely coproducts of *symmetrised representables*, that is, representables quotiented by composition with groups of automorphisms. Presheaves are represented by families as sets of *elements* that have an attached symmetry on their available *local interfaces*.

In a sense, this already generalises the equivalence results of [27,22]. However, the exact characterisation of which presheaves are (isomorphic to) coproducts of symmetrised representables is a difficult problem. Perhaps the most important topic in [25] is the notion of *finite support*, which generalises the notion of *free variables* in terms. The support is in turn the key ingredient to define named sets and the categorical equivalence between the two. In §4 we introduce a general notion of support in presheaf categories. Exploiting this definition, we show that the equivalence result of [27,22] can be extended to presheaves indexed by small categories, respecting three conditions: the index category has wide pullbacks, and the presheaves preserve them; the index category is made up of monos; all the arrows of the index category from an object to itself are isomorphisms. A non-trivial example respecting these conditions is the category $\mathtt{E}$ of finite equivalence relations and injective maps between their underlying sets, used in [3,4] to represent explicit fusions of names in process calculi.

Presheaves and families have a very different nature. We refer to this as *locality of interfaces*. In §5 we give a mathematical explanation of this property, which is reflected in the product construction. The product is just computed *point-wise* in presheaves, while it involves a mapping of the local interfaces of each involved element into a greater one, in the case of families. This corresponds to two radically different, though equivalent, views on how systems with interfaces may be related: either assuming a *naming authority* giving a global meaning to each available resource, or relying on locally scoped *links* that connect the different systems.

In §6, we show how to compute the *behavioural symmetry* of an element of a coalgebra, that is, the greatest group of isomorphisms that leave an element bisimilar to itself. We remark that §5 and §6 do not depend on the conditions of §4, but rather they are in the general framework of §3.

Related work. To the best of our knowledge, the study of families for an efficient representation of the semantics of programming languages, and the interpretation of their properties as a theory of *locality of interfaces*, are new and have never been investigated before. Coproducts of symmetrised representables are also interesting as a generalisation of the *analytic functors* of Joyal [30]. This is shown by Adámek and Velebil [2] for the case of *locally presentable* index categories. That research line is different in scope and aim from this work: there, a characterisation of the morphisms between analytic functors (the *regular* natural transformations of

[30]) would be desirable, but it is still an open problem. Instead, in §4 we develop an equivalence of categories, characterising *all* the natural trasformations of the subcategory by the means of morphisms of families. Moreover, the conditions of [2] to characterise coproducts of symmetrised representables and ours do not imply each other, and there are examples of categories, relevant for our purposes, that only fall under our conditions (see §4).

2 Background

Here we introduce the basic notions related to the *family* construction $\mathtt{Fam}(\mathtt{C})$, which is a representation of the *free coproduct completion* of $\mathtt{C}$.

Remark 2.1 *(notational conventions).* For $\mathtt{C}$ a category, we denote with $|\mathtt{C}|$ its objects, with $\mathtt{C}(n,m)$ the set of arrows from n to m. We extend some categorical notations to *sets* of arrows. Let $F \subseteq \mathtt{C}(n,m)$ be a set; we define $dom(F) = n$ and $cod(F) = m$. When F and G are two such sets, with $dom(F) = cod(G)$, $f : cod(G) \to m'$, and $g : m'' \to dom(F)$, we define $f \circ G = \{f \circ g \mid g \in G\}$, $F \circ g = \{f \circ g \mid f \in F\}$, and $F \circ G = \{f \circ g \mid f \in F, g \in G\}$. As a notation for the elements of the coproduct $\coprod_{x \in S} P_x$ in $\mathtt{Set}$, we use the set of pairs $\{\langle x, p\rangle \mid x \in S, p \in P_x\}$. The copairing of a tuple of arrows $f_{i \in I}$ is denoted with $\coprod_{i \in I} f_i$. We often omit the parenthesis in function and functor application, e.g. we write $\mathtt{F}fx$ to denote the action of the functor $\mathtt{F} : \mathtt{C} \to \mathtt{Set}$ on the arrow f, applied to the element x. With *pullbacks* we actually refer to *wide*, but small, pullbacks, that is, limits of small diagrams made up of an arbitrary number of arrows into the same object.

A direct description of the free coproduct completion of a category $\mathtt{C}$ is obtained by the *family* construction, defined as follows.

Definition 2.2 Given a small category $\mathtt{C}$, *objects* of the category $\mathtt{Fam}(\mathtt{C})$ are *families* of objects of $\mathtt{C}$, that is, coproducts $\coprod_{i \in I}\{n_i\}$ of singletons in $\mathtt{Set}$, where I is a set, and, for each $i \in I$, $n_i \in |\mathtt{C}|$. An *arrow* from $\coprod_{i \in I}\{n_i\}$ to $\coprod_{j \in J}\{m_j\}$ is a tuple $\langle f, \coprod_{i \in I}\{\mathcal{H}_i^f\}\rangle$, where $f : I \to J$ and, for each $i \in I$, $\mathcal{H}_i^f : n_i \to m_{f(i)}$.

A family is a set I, where each $i \in I$ has an associated $\mathtt{C}$-object n_i. The set I may represent, for example, the set of states of a system. The object n_i represents the *interface* of the state i. For example, n_i can be a set of names, a network topology, or any other possible feature associated to the states of a process calculus. Each arrow is a function f between two sets I and J, and for each $i \in I$ there is a map $\mathcal{H}_i^f$ from the interface of i to that of $f(i)$. This reflects the idea that interfaces are *local* to each element, therefore to properly define a function between such elements, one also has to specify how the interfaces of destination and source elements are related. When we use families to represent presheaves these maps go in the other direction, that is, from the destination to the source. Looking at the above definition, this does not make a big difference, as one can just consider the category $\mathtt{Fam}(\mathtt{C}^{op})$ to get these "backwards" arrows, as we shall do in the following. A real-world example of local interfaces which can help the intuition is the injective relabelling of memory

locations that may happen after an invocation of the garbage collector in a garbage-collected language. System states in this case have an associated memory layout (its "interface" in our terminology), that may change at each step of the execution. The relabelling is the "backward" arrow that we mention, mapping the memory layout of the destination into that of the source, thus tracking the history of variables and their memory locations along the computation. The coproducts in $\mathtt{Fam}(\mathtt{C})$ are *freely* generated, and described as follows.

Definition 2.3 The *coproduct* in $\mathtt{Fam}(\mathtt{C})$ of two objects $\coprod_{i\in I}\{n_i\}$ and $\coprod_{j\in J}\{m_j\}$ is defined as $\coprod_{k\in I+J}\{o_k\}$, where $o_k = n_i$ if $k = \langle I, i\rangle$, and $o_k = m_j$ if $k = \langle J, j\rangle$.

3 Families of symmetries

In this section we introduce a condition on presheaves in $\mathtt{Set}^{\mathtt{C}}$, namely being *coproducts of* symmetrised *representables*. The terminology is borrowed from [2]. In the rest of the paper we will discuss the good computational properties of such a representation, and introduce a representability criterion for presheaves over index categories of monos.

3.1 The category $\mathtt{Sym}(\mathtt{C})$

First, given a small category $\mathtt{C}$, we define a category of groups of automorphisms, and morphisms between them, that we call $\mathtt{Sym}(\mathtt{C})$.

Definition 3.1 We define the (small) category $\mathtt{Sym}(\mathtt{C})$ of *symmetries over* $\mathtt{C}$:

$$|\mathtt{Sym}(\mathtt{C})| = \coprod_{n\in|C|}\{\Phi \subseteq \mathtt{C}(n,n) \mid \Phi \text{ is a group w.r.t. composition}\}$$

$$\mathtt{Sym}(\mathtt{C})(\Phi_1,\Phi_2) = \{h\circ\Phi_1 \mid h \in \mathtt{C}(dom(\Phi_1), dom(\Phi_2)) \wedge \Phi_2 \circ h \subseteq h \circ \Phi_1\}$$

The identity of each object is $id_\Phi = id_{dom(\Phi)}\circ\Phi = \Phi$; the composition of $f_1 = h_1\circ\Phi_1$ and $f_2 = h_2 \circ \Phi_2$ is defined as $f_2 \circ f_1 = h_2 \circ h_1 \circ \Phi_1$.

An object of $\mathtt{Sym}(\mathtt{C})$ is just denoted by the group Φ, omitting the index n of the coproduct that is recovered as $dom(\Phi)$, the common domain of all the automorphisms in Φ. Arrows of the category are sets of arrows from $\mathtt{C}$, obtained by composition of a group of isomorphisms with a single arrow. Notice that the composition symbol on the left hand side of the last equation is the composition in $\mathtt{Sym}(\mathtt{C})$ which is being defined, while the composition on the right is composition of sets of arrows, as from Remark 2.1. However the following lemma ensures that the two possible interpretations coincide. This is a consequence of the condition $\Phi_2 \circ h \subseteq h \circ \Phi_1$.

Lemma 3.2 *Consider two* $\mathtt{Sym}(\mathtt{C})$ *arrows* $h_2\circ\Phi_2 : \Phi_2 \to \Phi_3$ *and* $h_1\circ\Phi_1 : \Phi_1 \to \Phi_2$. *It holds that* $(h_2 \circ h_1)\circ\Phi_1 = \{h_2 \circ \varphi_2 \circ h_1 \circ \varphi_1 \mid \varphi_2 \in \Phi_2 \wedge \varphi_1 \in \Phi_1\}$.

Finally we note that $\mathtt{C}$ has a full embedding into $\mathtt{Sym}(\mathtt{C})$.

Definition 3.3 The embedding $\mathtt{J} : \mathtt{C} \to \mathtt{Sym}(\mathtt{C})$ is defined on objects as $\mathtt{J}(n) = \{id_n\}$ and on arrows as $\mathtt{J}(f) = \{f\}$.

3.2 Coproducts of symmetrised representables as families

Throughout the paper, we let $\mathtt{C}$ denote a small category. We recall that the (covariant) hom functor $\mathtt{C}(n, -) : \mathtt{C} \to \mathtt{Set}$, for n an object of $\mathtt{C}$, acts on each object m as $\mathtt{C}(n, m)$, and on each arrow $f : m_1 \to m_2$ as $\mathtt{C}(n, f)(g : n \to m_1) = f \circ g : n \to m_2$. A *representable* presheaf in $\mathtt{Set}^{\mathtt{C}}$ is a functor which is isomorphic to $\mathtt{C}(n, -)$, for n an object of $\mathtt{C}$.

Definition 3.4 Let Φ be an object of $\mathtt{Sym}(\mathtt{C})$ with domain n. We call a *symmetrised representable* $\mathtt{C}(n, -)_{/\Phi}$ a representable quotiented by the indexed relation $g_1 \equiv_m g_2 \iff \exists \rho \in \Phi. g_1 = g_2 \circ \rho$, for $g_1, g_2 : n \to m$.

The equivalence classes of such a quotient at each index m are conveniently described as the composition of each possible arrow with Φ, that is $(\mathtt{C}(n, -)_{/\Phi})m = \{h \circ \Phi \mid h : n \to m\}$. Hereafter we assume that symmetrised representables are in this form. Notice that any $f \circ \Phi$ is an arrow of $\mathtt{Sym}(\mathtt{C})$, which gives rise to the representation we propose. For convenience we also state what is the action of symmetrised representables on arrows of $\mathtt{C}$, namely $(\mathtt{C}(n, -)_{/\Phi})f(h \circ \Phi) = f \circ h \circ \Phi$.

Among the presheaves in $\mathtt{Set}^{\mathtt{C}}$, some of them are isomorphic to a *coproduct* of symmetrised representables, giving rise to a full subcategory of $\mathtt{Set}^{\mathtt{C}}$. This subcategory is equivalent to $\mathtt{Fam}(\mathtt{Sym}(\mathtt{C})^{op})$. In the rest of the paper we shall advocate that a representation using families is appealing for computer science applications. First of all, even though the proof of equivalence is easily understood, we make it precise by the means of the following well-known proposition (see [8], Lemma 42), also used in [42], to prove the equivalence between named sets and the Schanuel topos.

Proposition 3.5 *Let $\mathtt{D}'$ be a locally small category having small coproducts, and $\mathtt{D}$ a small category. A functor $\mathtt{F} : \mathtt{D} \to \mathtt{D}'$ can be extended to an equivalence from $\mathtt{Fam}(\mathtt{D})$ to $\mathtt{D}'$ if it satisfies the following conditions: $\mathtt{F}$ is an embedding (it is injective on objects and morphisms); objects in the image of $\mathtt{F}$ are indecomposable (for each n in $|\mathtt{D}|$, the hom functor $\mathtt{D}'(\mathtt{F}n, -)$ preserves coproducts); every object of $\mathtt{D}'$ is a coproduct of objects in the image of $\mathtt{F}$.*

Here we instantiate the theorem with $\mathtt{D} = \mathtt{Sym}(\mathtt{C})$ and $\mathtt{D}'$ the subcategory of coproducts of symmetrised representables in $\mathtt{Set}^{\mathtt{C}}$. First, recall that if $\mathtt{C}$ is small, the functor category $\mathtt{Set}^{\mathtt{C}}$ is locally small and has coproducts (defined pointwise), hence Prop. 3.5 is applicable. We now exhibit a functor $\mathtt{F} : \mathtt{Sym}(\mathtt{C})^{op} \to \mathtt{Set}^{\mathtt{C}}$.

Definition 3.6 The functor $\mathtt{F}$ acts on objects as $\mathtt{F}\Phi = \mathtt{C}(dom(\Phi), -)_{/\Phi}$. $\mathtt{F}$ acts on each arrow $h \circ \Phi_1 : \Phi_2 \to \Phi_1$ of $\mathtt{Sym}(\mathtt{C})^{op}$ returning a natural transformation, defined at each index n as $(\mathtt{F}(h \circ \Phi_1))_n(h' \circ \Phi_2) = h' \circ h \circ \Phi_1$.

Next, we show that $\mathtt{F}$ respects the first and second conditions of Prop. 3.5. The third condition is satisfied by construction, when restricting the codomain of $\mathtt{F}$ to symmetrised representables.

Proposition 3.7 $\mathtt{F}$ *is a functor, and in particular an* embedding, *i.e. injective on objects and morphisms. For each object* $\Phi : \mathtt{Sym}(\mathtt{C})$, $\mathtt{F}\Phi$ *is indecomposable, that is, the homset functor* $\mathtt{Set}^{\mathtt{C}}(\mathtt{F}\Phi, -)$ *preserves coproducts.*

As $\mathtt{Set}^{\mathtt{C}}$ has coproducts, $\mathtt{F}$ extends to a functor from $\mathtt{Fam}(\mathtt{Sym}(\mathtt{C})^{op})$ to $\mathtt{Set}^{\mathtt{C}}$.

Definition 3.8 The functor $\mathtt{Presh} : \mathtt{Fam}(\mathtt{Sym}(\mathtt{C})^{op}) \to \mathtt{Set}^{\mathtt{C}}$ maps an object $\coprod_{i\in I}\{\Phi_i\}$ into $\coprod_{i\in I}\mathtt{F}\Phi_i$ and an arrow $\langle f, \coprod_{i\in I}\{\mathcal{H}_i^f\}\rangle : \coprod_{i\in I}\{\Phi_i\} \to \coprod_{j\in J}\{\Phi'_j\}$ into the natural transformation $\coprod_{i\in I}(\iota_{f(i)} \circ \mathtt{F}\mathcal{H}_i^f)$, where $\iota_{f(i)}$ denotes the $f(i)^{th}$ injection of the coproduct $\coprod_{j\in J}\mathtt{F}\Phi'_j$.

By definition, each presheaf in the image of $\mathtt{Presh}$ is a coproduct of symmetrised representables. The functor is full and faitful, and becomes one direction of a categorical equivalence when its codomain is restricted to its image.

The other direction is given by the functor $\mathtt{K}$ mapping coproducts of symmetrised representables into $\mathtt{Fam}(\mathtt{Sym}(\mathtt{C})^{op})$. The action on objects is rather trivial. Given $P = \coprod_{i\in I}\mathtt{C}(dom(\Phi_i), -)_{/\Phi_i}$, we have $\mathtt{K}P = \coprod_{i\in I}\{\Phi_i\}$. The action on arrows is more interesting: let $Q = \coprod_{j\in J}\mathtt{C}(dom(\Phi_j), -)_{/\Phi_j}$, and $g : P \to Q$ be a natural transformation. We define the morphism between families $\mathtt{K}(g : P \to P') = \langle f, \coprod_{i\in I}\{\mathcal{H}_i^f\}\rangle$. For each $i \in I$, let $g_n(\langle i, id_{dom(\Phi_i)} \circ \Phi_i\rangle) = \langle j, h' \circ \Phi_j\rangle$. Then we let $f(i) = j$ and $\mathcal{H}_i^f = h' \circ \Phi_j$. The function f is well defined by indecomposability of objects in the image of $\mathtt{F}$ (Prop. 3.7), in turn coming from naturality of g.

The action of $\mathtt{K}$ on arrows may be roughly explained by the idea of *local interfaces* in families. This is better understood after having introduced the notion of *orbit* and *representative*, which is done in §4.

4 Pullback-preservation, monos and minimal support

In this section we illustrate a characterisation of the coproducts of symmetrised representables in categories indexed by monos, as functors that preserve all pullbacks. We consider the *finite support* condition in the work by Gabbay and Pitts on nominal syntax [26]: each system has a unique minimal "interface". Preservation of pullbacks means preservation of "intersection of interfaces" in a very general sense, and makes it possible to recover a notion of *support* of an element $x \in Pn$ of a presheaf P over an arbitrary category $\mathtt{C}$ as the *minimal* index n' where an element $x' \in Pn'$ exists, such that $Pfx' = x$ for some arrow f.

The results presented here are similar in spirit to the representation of *analytic functors* as species given by Joyal [30], and therefore to [2], where conditions similar to ours are sketched to identify the coproducts of symmetrised representables. We emphasize that the latter research line aims to characterise and extend Joyal's analytic functors and *regular* natural transformations (the latter is still an open problem), whereas we are interested in *all* natural transformations between two coproducts of symmetrised representables. For this reason, we are able to provide an equivalence of categories. Moreover, the index category in [2] should be locally presentable (or at least should have an initial object, see §3 therein), thus ruling

out discrete categories and coproducts of categories (hence our results and [2] are logically independent).

The connection between representability of presheaves as families and pullback preservation has been studied in various works. A well known one is [7]. There, the connection between existence of connected limits, wide pullback preservation and familial representability is explained. But there the index category of the familial representation is still the same index category $\mathtt{C}$, of the presheaf category, and not a category of symmetries over it. Indeed the latter provides one a bit more structure, which we then use for the symmetry reduction procedure of §6.

The idea of representing pullback-preserving presheaves by families of symmetries comes from Staton [42], where it appears as a proof technique to show that named sets and the Schanuel topos are equivalent. The technical results that we present in this section are a direct generalisation of that work, even though the purposes are different, since we aim to explain the computational properties of the families model, which is done in the rest of the paper.

A *wide pullback* is the limit of a cocone of arbitrary cardinality (whereas an ordinary pullback is the limit of a cocone of just two arrows). Notice that in the special case of the Schanuel topos of [42], these diagrams are necessarily finite, and thus wide pullbacks are determined by the binary ones. From now on, we let $\mathtt{Set}^{\mathtt{C}}_{\Diamond}$ denote the wide-pullback-preserving full subcategory of $\mathtt{Set}^{\mathtt{C}}$. Our theory can be instantiated under the following conditions.

Criterion 4.1 *We assume that all the arrows of* $\mathtt{C}$ *are monic,* $\mathtt{C}$ *has (small, wide) pullbacks, and for every object* n *of* $\mathtt{C}$*, each* $f \in \mathtt{C}(n,n)$ *is an isomorphism.*

Notice that we do *not* require strong properties on $\mathtt{C}$ e.g. completeness or cocompleteness. Some examples may clarify the applicability of the characterisation.

Discrete categories: the one-object and one-arrow category $\mathtt{1}$ can be used as an index, resulting in a degenerate instantiation of the framework that actuall just contains sets and functions. This is correct, as $\mathtt{Set}^{\mathtt{1}}$ is $\mathtt{Set}$. More generally, *discrete* categories can be used, in this case the representation that we will define is just the set of elements of each presheaf, that is, pairs $\langle n, x\rangle$ where n is the index where x lives. This is a very natural representation of *multi-sorted* sets. These two examples show that the definition works also in these degenerate cases, giving the expected representation.

Coproducts of categories The coproducts of two non-empty categories certainly does not have an initial object and it is not complete. However, from the programming language semantics perspective, these index categories can be used represent calculi that feature several distinct kinds of agents, each one having a different notion of associated interface.

Finite sets and injections: in this case, the obtained equivalence is that between the Schanuel topos and named sets of [22,27]. The associated categories have been used in a wide range of applications as we already emphasized. The correspondence between families and named sets is made clear by the categorical definitions

given in [44,13]; the category $\mathtt{Symset}$ defined therein is $\mathtt{Sym}(\mathtt{I})$.

Finite graphs and injections: this category can be used to model calculi whose network structure is made explicit in the semantics (as opposed to the π-calculus, where the network structure is left implicit in the knowledge of channels by agents) and whose semantics is closed with respect to adding links to the network. The *network coordination policies* calculus (NCP) [11], has been developed by the first author et al. in the context of formal methods for service-oriented computing. In the calculus, states are pairs consisting of the network topology, represented as a graph, and a policy, which is a program. Entire fresh sub-topologies can be dynamically allocated along the transitions of the operational semantics. Even though category theory is not used in that work, it seems clear that the semantics can be represented using the standard presheaf approach, with finite graphs and injections as the index category. In NCP, bisimulation is used for the definition of conformance of the specification and the implementation, thus the implementation of an efficient bisimulation checker (taking into account the dynamic allocation capabilities of the framework) is of high relevance. Therefore, the calculus will be an appealing case study for the symmetry reduction algorithm that we sketch in this work.

Fusions: Fusions may be described by an indexing category $\mathtt{E}$ of equivalence relations with monic arrows [3]. This category has pullbacks, falls into the conditions of our framework, and it has a rich structure of objects that is used for fusions (see also [28,34]).

4.1 The symmetric decomposition of a presheaf

We now show that under Crit. 4.1, functors in $\mathtt{Set}^{\mathtt{C}}_{\Diamond}$ are isomorphic to coproducts of symmetrised representables, that is objects in the image of the functor $\mathtt{Presh}$. Therefore the full category $\mathtt{Set}^{\mathtt{C}}_{\Diamond}$ coincides exactly with the subcategory of coproducts of symmetrised representables.

We pursue our goal emplying Prop. 3.5 again. $\mathtt{F}$ being an embedding, and indecomposability of objects in its image are not affected by the additional hypothesis. However, we must prove that each presheaf in the image of $\mathtt{F}$ is pullback-preserving.

Theorem 4.2 *For each* Φ*, assuming Crit. 4.1,* $\mathtt{F}\Phi$ *preserves wide pullbacks.*

The rest of the section is devoted to prove the last required condition of Prop. 3.5, that is, each pullback-preserving presheaf is a coproduct of symmetrised representables. We recall the notion of *element* of a presheaf. Hereafter, we let $\mathtt{G}$ denote an arbitrary functor in $\mathtt{Set}^{\mathtt{C}}_{\Diamond}$.

Definition 4.3 The set of *elements* of $\mathtt{G}$ is defined as $El(\mathtt{G}) = \coprod_{n \in |\mathtt{C}|} \mathtt{G}n$.

For readability, but without loss of generality, in the following we assume that all the $\mathtt{G}n$ are disjoint, so that we are able to denote with just x the element $\langle n, x\rangle \in El(\mathtt{G})$. When necessary, we denote the stage n of x as $st(x)$.

Roughly, we aim to represent presheaves by quotienting all the elements that

are "reachable" from some common element by the action of arrows. To make this formal, we introduce the notion of *orbit*.

Definition 4.4 Given $x \in El(\mathtt{G})$, its *orbit* $\mathcal{O}_x$ is the set of elements $y \in El(\mathtt{G})$ such that there exist a span $st(x) \xleftarrow{f_x} s \xrightarrow{f_y} st(y)$ and an element $z \in \mathtt{G}s$, with $\mathtt{G}f_x z = x$ and $\mathtt{G}f_y z = y$.

In other words, an orbit is a connected component in the *category of elements*. In the following, for $x \in El(\mathtt{G})$, we let D^x be the diagram in $\mathtt{C}$ consisting of the morphisms $\{d : n \to st(x) \mid \exists y \in \mathtt{G}(n).\mathtt{G}dy = x\}$, for n ranging over $|\mathtt{C}|$. Notice that, for each d, y is uniquely determined: $\mathtt{G}d$ is injective because $\mathtt{G}$ is pullback-preserving, hence mono-preserving.

The following lemma forms the grounds of our representation. It is perhaps the most important property of orbits, due to pullback preservation of $\mathtt{Set}^{\mathtt{C}}_{\Diamond}$.

Lemma 4.5 *Let x and y belong to the same orbit. Let n be the pullback object of D^x and m be the pullback object of D^y. There exists an isomorphism between n amd m making n a pullback of D^y.*

We now define the *support* of an element x, which is, roughly speaking, the smallest index where an element having the same properties of x can be found.

Definition 4.6 Let $x^{\mathcal{O}}$ denote a choice of an element in $\mathcal{O}_x$. We define the *support* of x, denoted with $\mathcal{S}_x$, as the pullback object of $D^{(x^{\mathcal{O}})}$, and the *normalising arrow* $\mathcal{N}_x : \mathcal{S}_x \to st(x)$ as the diagonal of the pullback diagram of D^x, where we choose $\mathcal{S}_x$ as the pullback object by Lemma 4.5.

With *diagonal* here we mean the composition of any arrow in D^x with the corresponding arrow making the pullback commute.

We are going to see that an object of $\mathtt{Set}^{\mathtt{C}}_{\Diamond}$ is determined (up-to isomorphism) just by a set of representatives $\widehat{x}$ of elements, called *proper elements*, and by the set of isomorphisms over the stage of each $\widehat{x}$ whose action leaves $\widehat{x}$ unchanged. Preservation of pullbacks plays a fundamental role here, allowing us to prove the following lemma and to define the representative of an element.

Lemma 4.7 *There exists a unique element $\widehat{x} \in \mathtt{G}\mathcal{S}_x$ such that $\mathtt{G}\mathcal{N}_x\widehat{x} = x$.*

Definition 4.8 Let $x \in El(\mathtt{G})$. We denote with $\widehat{x}$ the *representative* of x, that is, the element of $\mathtt{G}\mathcal{S}_x$ such that $\mathtt{G}\mathcal{N}_x(\widehat{x}) = x$. The set of *proper elements* of $\mathtt{G}$ is defined as $Pel(\mathtt{G}) = \{\widehat{x} \mid x \in El(\mathtt{G})\}$.

In this construction, $\mathcal{N}_x$ plays the role of a canonical arrow whose action recovers x from its representative $\widehat{x}$. The *symmetry* associates to each proper element an object of $\mathtt{Sym}(\mathtt{C})$.

Definition 4.9 The *symmetry* of $\widehat{x} \in Pel(\mathtt{G})$ is the group of isomorphisms $\mathcal{G}_{\widehat{x}} = \{\rho : \mathcal{S}_x \to \mathcal{S}_x \mid \mathtt{G}\rho\widehat{x} = \widehat{x}\}$.

Now we can define a functor from $\mathtt{Set}^{\mathtt{C}}_{\Diamond}$ to $\mathtt{Fam}(\mathtt{Sym}(\mathtt{C})^{op})$ which, together with the functor $\mathtt{Presh}$ of Def. 3.8, completes the categorical equivalence.

Definition 4.10 The *symmetric decomposition* $\mathtt{SymDec} : \mathtt{Set}^{\mathtt{C}}_{\diamondsuit} \to \mathtt{Fam}(\mathtt{Sym}(\mathtt{C})^{op})$ is defined on each presheaf $\mathtt{G}$ and natural transformation $f : \mathtt{G}_1 \to \mathtt{G}_2$ as

$$\mathtt{SymDec}(\mathtt{G}) = \coprod_{\widehat{x} \in Pel(\mathtt{G})} \{\mathcal{G}_{\widehat{x}}\} \qquad \mathtt{SymDec}(f) = \langle \lambda \widehat{x}. \widehat{f_{\mathcal{S}_x}(\widehat{x})}, \coprod_{\widehat{x} \in Pel(\mathtt{G}_1)} \{\mathcal{N}_{f(\widehat{x})} \circ \mathcal{G}_{\widehat{f_{\mathcal{S}_x}(\widehat{x})}}\} \rangle$$

The action of the functor on objects just records the proper elements of $\mathtt{G}$, and their symmetry. The action on arrows is an arrow of $\mathtt{Fam}(\mathtt{Sym}(\mathtt{C})^{op})$, thus a function between the two index sets, and a family of arrows in $\mathtt{Sym}(\mathtt{C})^{op}$. The former returns, for each representative $\widehat{x}$, the representative of $f_{\mathcal{S}_x}(\widehat{x})$. The mappings associated to the arrow are the normalising arrows of every obtained element, composed with the corresponding symmetry. Using it, one can reconstruct $f_{\mathcal{S}_x}(\widehat{x})$ from its representative. A bit more intuition may be obtained by considering the support and symmetry of an element as a *local* interface of that element. The arrow $\mathcal{N}_{f(\widehat{x})} \circ \mathcal{G}_{\widehat{f_{\mathcal{S}_x}(\widehat{x})}}$ embeds the interface of $\widehat{f_{\mathcal{S}_x}(\widehat{x})}$ into the interface of $f_{\mathcal{S}_x}(\widehat{x})$, which is the same of $\widehat{x}$ because f is defined pointwise. The normalising arrow is the so-called *history of names along morphisms* [5] used in the literature on named functions, and in coalgebras it plays a similar role to the injective relabelling of memory locations done by garbage collectors in the implementation of programming languages.

Lemma 4.11 *We have* $\widehat{\mathtt{G}h\widehat{x}} = \widehat{x}$, *and* $\mathcal{N}_{\mathtt{G}h\widehat{x}} \in h \circ \mathcal{G}_{\widehat{x}}$.

Theorem 4.12 *Every presheaf* $\mathtt{G}$ *in* $\mathtt{Set}^{\mathtt{C}}_{\diamondsuit}$ *is isomorphic to* $\mathtt{Presh}(\mathtt{SymDec}(\mathtt{G}))$, *therefore* $\mathtt{Set}^{\mathtt{C}}_{\diamondsuit}$ *is equivalent to* $\mathtt{Fam}(\mathtt{Sym}(\mathtt{C})^{op})$.

Remark 4.13 A great advantage of the proposed representation of presheaves using families is to reduce the size (the number of elements) of the represented presheaf, even getting a finite set out of an infinite one, while preserving the categorical properties. For example, the "inclusion" presheaf $Gn = n, Gf = f$ in $\mathtt{Set}^{\mathtt{I}}$, that is, the object of names in $\mathtt{Set}^{\mathtt{I}}$, is represented by a family having a single element [6] in $\mathtt{Fam}(\mathtt{Sym}(\mathtt{I})^{op})$, namely $\coprod_{i \in 1}\{id_1\}$. The intuitive meaning of this assertion is that each natural number is not distinguishable from any other, and has a single "name" (and trivial symmetry) as its interface. This "finitistic" representation is the main reason why named sets and history-dependent automata have been considered appealing for the static analysis of nominal calculi (model checking [29], and bisimulation checking [21]).

5 Locality of interfaces: the product construction

In [44], one of the authors extended the equivalence of [27,22] to the categories of coalgebras of equivalent endofunctors, in order to give a categorical characterisation of the various constructions that had been used in the past for named sets (including minimisation of the π -calculus). Here we generalise the results on the product of named sets presented therein.

[5] In our case, we should call it the history of *interfaces* along morphisms.

[6] G is different from the final object $\coprod_{i \in 1}\{id_0\}$, having a single element with trivial interface

Multi-(co)products are a specialisation of the notion of multi-(co)limit, studied in detail by Diers [16]. It is well known (see e.g. [14], remark 5) that $\mathtt{Fam}(\mathtt{C})$ has products whenever $\mathtt{C}$ has multi-products, and dually, $\mathtt{Fam}(\mathtt{C}^{op})$ has products if $\mathtt{C}$ has multi-coproducts. Here we provide a concrete characterization of the functor, that emphasizes the difference between *global* and *local* interfaces. The results presented here do not rely on arrows of $\mathtt{C}$ being mono.

Definition 5.1 Given a diagram D consisting of a tuple of objects $\langle n_1, \ldots, n_k \rangle$, the *multi-coproduct* of D is a set $mcp(D)$ of cocones over D such that for all cocones $L' = \langle f_1 : n_1 \to m', \ldots, f_k : n_k \to m' \rangle$ over D there exists a unique cocone $L = \langle \iota_1 : n_1 \to m, \ldots, \iota_k : n_k \to m \rangle \in mcp(D)$, and a unique arrow $u_{L'} : m \to m'$ making the diagram $L \cup L' \cup u_{L'}$ commute. The unique cocone L will be denoted, with a bit of overloading, with $mcp(L')$.

In words, the multi-coproduct of two objects P and Q is a set of *canonical* cospans between them, in the sense that they are quotiented by isomorphisms of cospans, and they are minimal.

We note that $\mathtt{Sym}(\mathtt{C})$ has multi-coproducts.

Theorem 5.2 *If* $\mathtt{C}$ *has wide pullbacks, then* $\mathtt{Sym}(\mathtt{C})$ *has multi-coproducts.*

In the following definitions, we assume that $\mathtt{C}$ has multi-coproducts, that $P = \coprod_{i \in I}\{n_i\}$, $Q = \coprod_{i \in J}\{m_j\}$, $R = \coprod_{k \in K}\{o_k\}$ are three arbitrary objects of $\mathtt{Fam}(\mathtt{C}^{op})$, and we denote with S the set $\{\langle i, j, \langle \iota_1, \iota_2 \rangle \rangle \mid i \in I \wedge j \in J \wedge \langle \iota_1, \iota_2 \rangle \in mcp(\langle n_i, m_j \rangle)\}$.

Definition 5.3 The *product* of P and Q in $\mathtt{Fam}(\mathtt{C}^{op})$ is defined as the object $P \times Q = \coprod_{\langle i,j,\langle \iota_1,\iota_2 \rangle \rangle \in S}\{cod(\iota_1)\}$.

Elements of the product $P \times Q$ are triples, formed by an element of P, an element of Q, and a (canonical) cospan relating their symmetry.

Definition 5.4 Let π'_1 and π'_2 denote the first two projections of the ternary product S. The *projections* $\pi_1 : P \times Q \to P$ and $\pi_2 : P \times Q \to Q$ are defined as $\pi_1 = \langle \pi'_1, \coprod_{\langle i,j,\langle \iota_1,\iota_2 \rangle \rangle \in S}\{\iota_1\} \rangle$, $\pi_2 = \langle \pi'_2, \coprod_{\langle i,j,\langle \iota_1,\iota_2 \rangle \rangle \in S}\{\iota_2\} \rangle$.

Definition 5.5 The *pairing* of $\langle f, \coprod_{k \in K}\{\mathcal{H}^f_k\} \rangle : R \to P$ and $\langle g, \coprod_{k \in K}\{\mathcal{H}^g_k\} \rangle : R \to Q$ is the arrow $\langle h, \coprod_{k \in K}\{\mathcal{H}^h_k\} \rangle$, where $h(k) = \langle f(k), g(k), mcp(\langle \mathcal{H}^f_k, \mathcal{H}^g_k \rangle) \rangle$, and $\mathcal{H}^h_k = u_{\langle \mathcal{H}^f_k, \mathcal{H}^g_k \rangle}$.

Theorem 5.6 *The product, projections and pairing given above identify up to isomorphism the binary product in* $\mathtt{Fam}(\mathtt{C}^{op})$.

In the above definition, $mcp(\langle \mathcal{H}^f_k, \mathcal{H}^g_k \rangle)$ and $u_{\langle \mathcal{H}^f_k, \mathcal{H}^g_k \rangle}$ come from Def. 5.1. We keep on with the intuition that the index category $\mathtt{C}$ in $\mathtt{Set}^{\mathtt{C}}$ should be perceived as a set of possible *types*, or *interfaces* of elements of the presheaf. In this light, the definition of the product above gives a notion of *locality* of interfaces in families, as opposed to a notion of *global* interfaces in presheaf categories.

In $\mathtt{Set}^{\mathtt{C}}$ the product is defined pointwise, and two elements may be related by just pairing them if they are in an appropriate (common) context. That is, any two interfaces have a natural choice of an embedding into a common, greater interface, thus their relative meaning is established once and for all. In the case of names (that is, where the index category is $\mathtt{I}$), this is the vision adopted by the π-calculus, where the names of all the non-restricted channels of an agent have a global, unique meaning across all participating parallel components of a system, as if there was a *naming authority* assigning a meaning to any name.

In $\mathtt{Fam}(\mathtt{C}^{op})$, whenever we put two elements in a relation, we have to explicitly establish a link between their interfaces by exhibiting them as subobjects of a common object, acting as the interface of the obtained tuple. In the case of names, this corresponds to having to "pull wires" among all parallel components of a system to make explicit how they can interact. This may be the most natural choice whenever one wants to model systems that do not have a naming authority, such as *peer-to-peer* systems.

As an example, bisimilarity in $\mathtt{Fam}(\mathtt{C}^{op})$ is made up of triples, because it is a subobject of the product: in order to compare two systems, we need to establish a correspondence between their local interfaces.

6 Symmetry reduction by final semantics

The presheaf approach to operational semantics roughly consists in defining a presheaf P of *terms*, that is, the initial algebra of some endofunctor over a presheaf category, and a coalgebra from P to $\mathtt{T}P$ for some endofunctor $\mathtt{T}$, providing the semantics of the calculus. The unique morphism into the final coalgebra of $\mathtt{T}$ then gives the coinductive definition of the abstract semantics. Here we link the symmetry of elements in $\mathtt{Fam}(\mathtt{Sym}(\mathtt{C})^{op})$ with *behavioural equivalence*, defined as the pullback object of a coalgebra morphism. We note that coalgebraic bisimilarity and behavioural equivalence coincide if the behavioural functor $\mathtt{T}$ preserves weak pullbacks (see [31] or [1] for details). Given a coalgebra in $\mathtt{Fam}(\mathtt{Sym}(\mathtt{C})^{op})$, and an element i, having symmetry Φ with $dom(\Phi) = n$, we explain how computing the image of i along the unique morphism into the final coalgebra corresponds to identify the subobject of n that is *active* in the semantics of i, and the greatest possible symmetry over this object that preserves behavioural equivalence.

The interest of this result is in providing a clean framework (namely, the equivalence between presheaves and families) for symmetry reduction of the semantics of programming languages. Symmetry reduction is an actively researched topic in computer science that consists in finding compressed representations of systems that have a symmetry (see [15] and subsequent works, or the more recent [19]). This is typically done exploiting equations on the syntax of calculi, or by adding symmetry information "by hand" to models. Our approach is very different: it allows one to *compute* the behavioural symmetry, that is, the best symmetry up-to bisimulation. This is certainly wanted in all the cases where bisimulation is the equivalence relation of choice (e.g. static analysis in service oriented computing and model checking

of Hennessy-Milner-like logics). Model checking can be performed efficiently in the presence of symmetry [18].

6.1 Symmetry reduction

Remark 6.1 Equivalences extend to categories of coalgebras of suitable "equivalent" endofunctors. In particular, each endofunctor $\mathtt{T}'$ over the full subcategory of coproducts of symmetrised representables in $\mathtt{Set}^{\mathtt{C}}$ that has a final coalgebra has an equivalent endofunctor over $\mathtt{Fam}(\mathtt{Sym}(\mathtt{C})^{op})$ admitting a final coalgebra, obtained (up to isomorphism) as $\mathtt{T} = \mathtt{SymDec} \circ \mathtt{T}' \circ \mathtt{Presh}$.

We assume in the following such a pair of equivalent endofunctors $\mathtt{T}'$ and $\mathtt{T}$. Even if for the scope of this work the given definition of $\mathtt{T}$ is sufficent, it may be necessary to have a *compositional* definition of $\mathtt{T}$ so that the elements of $\mathtt{T}(P)$ are derived from those of P. In the case of the product, for example, the definition of §5 is isomorphic to the one that we just mentioned, but not the same. This topic has been studied in detail in [44].

We now observe that each natural transformation between coproducts of symmetrised representables induces a symmetry on elements of its source, explicitly represented in the corresponding arrow of $\mathtt{Fam}(\mathtt{Sym}(\mathtt{C})^{op})$. Consider a presheaf $\mathtt{G} = \coprod_{i\in I} \mathtt{F}\Phi_i$, a natural transformation $f : \mathtt{G} \to \mathtt{G}'$, and the corresponding arrow $\langle g, \coprod_{i\in I}\{\mathcal{H}_i^g\}\rangle : \coprod_{i\in I}\{\Phi_i\} \to \coprod_{j\in J}\{\Phi'_j\}$.

Definition 6.2 Let R_n^f denote the relation coming from the kernel pair of the component f_n of f at n. Let $x \in \mathtt{G}n$. We call the set $\mathcal{G}_x^h = \{\rho : n \to n \mid \mathtt{G}\rho x R_n^f x\}$ the symmetry on x induced by f.

Proposition 6.3 *For each $i \in I$, $n \in |\mathtt{C}|$, $h \circ \Phi_i \in \mathtt{F}\Phi_i n$, and $\rho : n \to n$, we have $(\mathtt{F}\Phi_i\rho(h \circ \Phi_i))R_n^f(h \circ \Phi_i)$ if and only if $\rho \circ h \circ \mathcal{H}_i^g = h \circ \mathcal{H}_i^g$.*

Observe that $\rho \circ h \circ \mathcal{H}_i^g = h \circ \mathcal{H}_i^g$ implies that, for each h' in $h \circ \mathcal{H}_i^g$, there is an isomorphism $\rho' \in \Phi'_{g(i)}$ such that $\rho \circ h' = h' \circ \rho'$, that is, the symmetry induced by f is reflected in $\Phi'_{g(i)}$.

It is now obvious to observe that the symmetry induced by coalgebra morphisms respects bisimulation. When f is the unique morphism into the final coalgebra, the induced symmetry is the greatest possible such subset. We call it the *behavioural symmetry*. In this case, the arrows in $h \circ \mathcal{H}_i^g$ identify a subobject of n that intuitively is the *active* "sub-interface" of an element, i.e. operations that do not touch it may not affect the semantics. To make this more precise, observe that, for each $h' \in h \circ \mathcal{H}_i^g$, we either have $\rho \circ h' \neq h'$ or $\rho \circ h' = h'$. The first case is the one where the symmetry $\Phi'_{g(i)}$ actually plays a role. In the second case, as all the arrows in $h \circ \mathcal{H}_i^g$ are obtained by composition of h' with an arrow in $\Phi'_{g(i)}$, composition with ρ leaves *all* of them unchanged. Then ρ is acting in some sense *outside* of the subobject identified by $h \circ \mathcal{H}_i^g$. For example, when the index category is $\mathtt{I}$, the image of h is the set of *active names* of a system, that is, names that are observable in the final semantics.

6.2 Partition refinement as a generic symmetry reduction algorithm

Here and in the next section we explain how to compute bisimilarity on a subset of the terms of a calculus, if certain finiteness conditions hold.

Consider a calculus equipped with a semantics in $\mathtt{Set}^{\mathtt{C}}$, $s : P \to \mathtt{T}'P$ for P representing the syntax. As we know (see Rem. 6.1), if P is a coproduct of symmetrised representables, there is a corresponding coalgebra $t : P' \to \mathtt{T}P'$ in $\mathtt{Fam}(\mathtt{Sym}(\mathtt{C})^{op})$ of a suitable endofunctor $\mathtt{T}$ corresponding to $\mathtt{T}'$.

The partition refinement in $\mathtt{Fam}(\mathtt{Sym}(\mathtt{C})^{op})$ can be computed on an object $\coprod_{q\in Q}\{\mathcal{G}_q\}$ (intended to be a subobject of P' above) as follows. First, we give an abstract description of the general algorithm, then we explain in detail the single steps and discuss some finiteness conditions to compute them in $\mathtt{Fam}(\mathtt{Sym}(\mathtt{C})^{op})$.

Definition 6.4 Coalgebraic partition refinement in $\mathtt{Fam}(\mathtt{Sym}(\mathtt{C})^{op})$ is an iterative algorithm using three variables, f, h and z, denoting arrows in $\mathtt{Fam}(\mathtt{Sym}(\mathtt{C})^{op})$.

Initialization: Let $f = t$, let $h : \coprod_{q\in Q}\{\mathcal{G}_q\} \to 1$ be the unique morphism into the final object of $\mathtt{Fam}(\mathtt{Sym}(\mathtt{C})^{op})$, and z the unique morphism from $\mathtt{T}1$ to 1.

Iteration step(f, h, z): If z restricted to $Im(\mathtt{T}h \circ f)$ is an isomorphism in $\mathtt{Fam}(\mathtt{Sym}(\mathtt{C})^{op})$ then return $\mathtt{T}h \circ f$. Otherwise let $f' = \mathtt{T}f \circ f$, $h' = \mathtt{T}h$, $z' = \mathtt{T}z$, and compute **Iteration step(f', z', w')**.

Correctness of the algorithm is well known by the theory of coalgebras (see e.g. [46]). An intuition can be given as follows. At the n^{th} iteration of the algorithm, the kernel of $\mathtt{T}h \circ f : \coprod_{q\in Q}\{\mathcal{G}_q\} \to \mathtt{T}^n 1$ is a partition of Q, which quotients elements that have the same observations in n steps. At each step, this partition is refined, that is, possibly split, according to the observations made in the n^{th} iteration of the system. When z is an isomorphism, a fixed point is reached, and it is guaranteed that in all successives steps, the partition will remain unchanged. Therefore, the elements of Q that are equalised by $\mathtt{T}h \circ f$ are bisimilar. The isomorphism z is a subobject of the final coalgebra that represents the behaviour of the elements of Q.

Convergence of the algorithm is equivalent to deciding the semantics of a program, therefore it can not be guaranteed *a priori* for all calculi. For Turing-equivalent languages, the algorithm converges on an undecidable subset of all the possible programs. In labelled transition systems, one gets convergence if the set of states reachable from a given set of initial states is *finite*. When using coalgebras over presheaves, even trivial programs have infinite states, but finiteness of the elements of the corresponding family is enough to guarantee convergence. This leads to a more refined notion of finiteness for presheaves.

Static constraints may be used (e.g. the *finite-control* π-calculus agents of [21]) to identify a subset of the convergent instantiations of the algorithm.

The pairs of bisimilar systems in Q are described by the kernel pair of the final value of the arrow $\mathtt{T}h \circ f$, and the behavioural symmetry of each element $q \in Q$ is reflected in the symmetry of its image along the same arrow. When $\mathtt{C}$ is the free category over one object and $\mathtt{T} = \mathcal{P}_{fin}(L \times -)$, then $\mathtt{Fam}(\mathtt{Sym}(\mathtt{C})^{op})$ is $\mathtt{Set}$, L is a set of labels, and the algorithm is the classical partition refinement for labelled

transition systems. When $\mathtt{C}$ is $\mathtt{I}$, there is a suitable endofunctor [13] such that the algorithm above is the partition refinement procedure for the π-calculus of [39,21].

Computing the semantics

Two basic assumptions are needed. First, objects and arrows of $\mathtt{C}$ should be "finite", in the sense that they can be represented as data structures. Then, f should be computable in each step of the algorithm. Without these assumptions, the algorithm can not be implemented. Indeed, the cases studied in the literature on presheaves for process calculi fall under these hypotheses.

To be able to compute partition refinement, we first need to describe the final object in $\mathtt{Fam}(\mathtt{Sym}(\mathtt{C}^{op}))$. In a similar fashion to Thm. 5.6, the final object in $\mathtt{Fam}(\mathtt{C})$ is a family of *multi-initial* objects, that is, a set MI of $\mathtt{C}$-objects such that for each object c of $\mathtt{C}$ there is a unique element $i \in MI$ and a unique arrow $u : i \to c$. Similarly to Thm. 5.2, it is possible to show that if $\mathtt{C}$ has pullbacks, then $\mathtt{Sym}(\mathtt{C})^{op}$ has a set of multi-initial objects.

Proposition 6.5 *Given a set MI of multi-initial objects in $\mathtt{Sym}(\mathtt{C})$, the object $P = \coprod_{\Phi \in MI}\{\Phi\}$ is a final object in $\mathtt{Fam}(\mathtt{Sym}(\mathtt{C})^{op})$. The unique arrow from $\coprod_{j\in J}\{\Phi_j\}$ to P is $\langle \lambda j.i_{\Phi_j}, \coprod_{\Phi\in\Phi_j}\{u_{\Phi_j}\}\rangle$, where i_{Φ_j} and u_{Φ_j} denote respectively the unique element of MI and the unique arrow corresponding to Φ_j in MI.*

It holds that if a category has an initial object i, then the singleton $\{i\}$ is a family of multi-initial objects. Getting back to partition refinement, to compute h, z and f one needs that Q is finite and that from each object of q the corresponding element of the final object is computable.

One also needs that the image of f is finite on all the elements of Q, in order to be able to enumerate the elements on which z has to be an isomorphism. This requirement is certainly satisfied if $\mathtt{T}$ sends finite families into finite families. This happens in many interesting cases, including polynomial functors, name allocation, and certain non finite subfunctors of the power set. Remarkably, in [44] such a "finitistic" representation is given for the *early semantics* of the π-calculus, which is defined as an infinitary transition system, due to the input transitions.

Under the above restrictions, one has to check if $z = \langle f_z, \coprod_{i\in Im(\mathtt{T}h\circ f)}\{\mathcal{H}_i^{f_z}\}\rangle$ is an isomorphism. The criterion in $\mathtt{Fam}(\mathtt{Sym}(\mathtt{C})^{op})$ is that f_z is an isomorphism in $\mathtt{Set}$ and each $\mathcal{H}_i^{f_z}$ is an isomorphism in $\mathtt{Sym}(\mathtt{C})$. To check the latter, it is necessary to determine the symmetry of elements of $\mathtt{T}^n 1$ for each n. Having an effective procedure to compute this symmetry depends on the chosen functor. In [44] it is shown how to do this for polynomials, name abstraction and subfunctors of the power set. We conjecture that these results generalise to other categories of finite structures.

6.3 Garbage collection

We consider the representation using families appealing because it may allow one to implement iteration along the terminal sequence, starting from a coalgebra defining the operational semantics, in the presence of fresh resouce allocation. We emphasize

that fresh resources are perhaps the most important reason to employ presheaves for the semantics of programming languages.

In presheaf models, whenever behavioural functors that may *allocate* new resources, such as the functor δ for name abstraction of [24], are used to build coalgebras, the operational semantics obtained by rules typically becomes infinite even in very simple cases. Again, this comes from the fact that interfaces have a *global* meaning in presheaves, whereas in the family representation the symmetry of each element is *local*. This is reflected in the definition of arrows: in presheaves, one does not need to provide information on how the interface of the destination is mapped in the interface of the source, while this is exactly the role of the family of arrows in $\mathtt{Sym}(\mathtt{C})$ (one for each element) that are the second component of an arrow of $\mathtt{Fam}(\mathtt{Sym}(\mathtt{C})^{op})$. Thus, elements that have the same behaviour up-to an operation on their interface are not identified using presheaves. This is particularly problematic for recursive processes that allocate some resources while discarding older ones, keeping a finite quantity of resources allocated in each state (as explained in [12]). Using families, on the other hand, all these equivalent elements are identified. It is the purpose of the family of maps associated to each arrow of the category to identify a "sub-interface" of each source state, which is preserved in the destination state, thus discarding unused resources.

7 Concluding remarks

We have introduced a framework to represent the semantics of programming languages that deal with *resources* or *interfaces* attached to system states: coalgebras over *presheaf* categories obeying to certain constraints, that give rise to a "finitistic" representation using *families*. This representation removes the redundant information coming from the notion of interfaces being *global* rather than *local*.

First of all, a complete example of application should be developed. The field of presheaf semantics for process calculi is still a relatively new research field, and there is not so much literature on calculi different from the π-calculus. However, by providing a representation theory, we prepare the grounds on which to build up new applications. An interesting case study is [4], since the presheaf category employed there respects the conditions of §4.

Applications are of great interest in the area of *service-oriented computing*, where resource allocation in the presence of *network topologies* [11], or constraints [6] is an active field of research, and finite representations are of vital importance for the implementation of analysis algorithms. An efficient implementation of the generic symmetry reduction algorithm that we have presented should be studied. For that, one may take advantage of algorithms on permutation groups exploiting the generators [32]. Finally, similar consideration apply to model checking. The study of a Stone-type duality for coalgebras over families in a similar fashion to [5], and a corresponding model checking algorithm exploiting the cases where the representation is finite, are one of our most important long-term goals.

It is expected that the categorical equivalence that we presented, combining the

ease of specifying the semantics using presheaves with the implementative advantages of named sets, will enable the development of a general framework to specify (using presheaves) and analyse (using families) the semantics of calculi that have richer interfaces than pure names, thus advancing the research line of presheaves, named sets and history dependent automata.

References

[1] J. Adamek. Introduction to coalgebra. *Theory and Applications of Categories*, 14(8):157–199, 2005.

[2] J. Adamek and J. Velebil. Analytic functors and weak pullbacks. *Theory and Applications of Categories*, 21(11):191–209, 2008.

[3] F. Bonchi, M. Buscemi, V. Ciancia, and F. Gadducci. A Category of Explicit Fusions. *LNCS - Festschrift for Ugo Montanari*, 5065, 2008.

[4] F. Bonchi, M. Buscemi, V. Ciancia, and F. Gadducci. A presheaf environment for the calculus of explicit fusions. *Submitted*, 2009.

[5] M. M. Bonsangue and A. Kurz. Pi-calculus in logical form. In *LICS*, pages 303–312. IEEE Computer Society, 2007.

[6] M. G. Buscemi and U. Montanari. Cc-pi: A constraint-based language for specifying service level agreements. In R. De Nicola, editor, *ESOP*, volume 4421 of *LNCS*, pages 18–32. Springer, 2007.

[7] A. Carboni and P. Johnstone. Connected limits, familial representability and the artin glueing. *Mathematical Structures in Computer Science*, 5, 1995.

[8] A. Carboni and E. Vitale. Regular and exact completions. *Journal of Pure and Applied Algebra*, 125(1-3):79 – 116, 1998.

[9] G. L. Cattani and P. Sewell. Models for name-passing processes: Interleaving and causal. In *LICS*, pages 322–332, 2000.

[10] G. L. Cattani, I. Stark, and G. Winskel. Presheaf models for the π-calculus. In *Category Theory and Computer Science*, pages 106–126, 1997.

[11] V. Ciancia, G. L. Ferrari, R. Guanciale, and D. Strollo. Event based choreography. *Science of Computer Programming*, To appear.

[12] V. Ciancia and U. Montanari. A name abstraction functor for named sets. *Electr. Notes Theor. Comput. Sci.*, 203(5):49–70, 2008.

[13] V. Ciancia and U. Montanari. Symmetries, local names and dynamic (de)-allocation of names. *Information and Computation*, 2009. To appear.

[14] C. Cirstea. Semantic constructions for the specification of objects. *Theor. Comput. Sci.*, 260(1-2):3–25, 2001.

[15] E. M. Clarke, E. A. Emerson, S. Jha, and A. P. Sistla. Symmetry reductions in model checking. In *Computer Aided Verification, 10th International Conference*, volume 1427 of *LNCS*, pages 147–158, 1998.

[16] Y. Diers. Familles universelles de morphismes. *Ann. Soc. Sci. Bruxelles*, 93:175–195, 1979.

[17] J. D. Dixon and B. Mortimer. *Permutation Groups*, volume Permutation Groups of *Graduate Texts in Mathematics*. Springer, 2006.

[18] E. A. Emerson and A. P. Sistla. Symmetry and model checking. *Formal Methods in System Design*, 9(1/2):105–131, 1996.

[19] E. A. Emerson and T. Wahl. Dynamic symmetry reduction. In N. Halbwachs and L. D. Zuck, editors, *TACAS 2005*, volume 3440 of *Lecture Notes in Computer Science*, pages 382–396. Springer, 2005.

[20] G. L. Ferrari, U. Montanari, and M. Pistore. Minimizing transition systems for name passing calculi: A co-algebraic formulation. In *FoSSaCS*, pages 129–158, London, UK, 2002. Springer-Verlag.

[21] G. L. Ferrari, U. Montanari, and E. Tuosto. Coalgebraic minimization of hd-automata for the pi-calculus using polymorphic types. *Theor. Comput. Sci.*, 331(2-3):325–365, 2005.

[22] M. Fiore and S. Staton. Comparing operational models of name-passing process calculi. *Inf. Comput.*, 204(4):524–560, 2006.

[23] M. P. Fiore, E. Moggi, and D. Sangiorgi. A fully-abstract model for the pi-calculus (extended abstract). In *LICS*, pages 43–54, 1996.

[24] M. P. Fiore and D. Turi. Semantics of name and value passing. In *LICS*, pages 93–104, 2001.

[25] M. Gabbay and A. Pitts. A new approach to abstract syntax involving binders. In *LICS*, pages 214–224, 1999.

[26] M. Gabbay and A. M. Pitts. A new approach to abstract syntax with variable binding. *Formal Asp. Comput.*, 13(3-5):341–363, 2002.

[27] F. Gadducci, M. Miculan, and U. Montanari. About permutation algebras, (pre)sheaves and named sets. *Higher-Order and Symbolic Computation*, 19(2-3):283–304, 2006.

[28] N. Ghani, K. Yemane, and B. Victor. Relationally staged computations in calculi of mobile processes. *Electr. Notes Theor. Comput. Sci.*, 106:105–120, 2004.

[29] S. Gnesi and G. Ristori. A model checking algorithm for π-calculus agents. In *Proc. Second International Conference on Temporal Logic (ICTL '97)*. Kluwer Academic Publishers, 1997.

[30] A. Joyal. Foncteurs analytiques et espces de structures. In *Combinatoire Ènumérative*, volume 1234 of *Springer Lecture Notes in Mathematics*. Springer Verlag, 1985.

[31] A. Kurz. *Logics for Coalgebras and Applications for Computer Science*. PhD thesis, Ludwig-Maximilians-Universitat Munchen, 2000.

[32] E. M. Luks. Permutation Groups and Polynomial Time Computation. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 11:139–175, 1993.

[33] M. Miculan. A categorical model of the fusion calculus. *Electr. Notes Theor. Comput. Sci.*, 218:275–293, 2008.

[34] M. Miculan and K. Yemane. A unifying model of variables and names. In V. Sassone, editor, *FoSSaCS*, volume 3441 of *Lecture Notes in Computer Science*, pages 170–186. Springer, 2005.

[35] R. Milner. A calculus of communicating systems. *Lecture Notes in Computer Science*, 92, 1980.

[36] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, part i. *Information and Computation*, 100(1):1–40, 1992.

[37] E. Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, 1991.

[38] U. Montanari and M. Pistore. pi-calculus, structured coalgebras, and minimal hd-automata. In *MFCS*, volume 1893 of *LNCS*, pages 569–578, 2000.

[39] U. Montanari and M. Pistore. Structured coalgebras and minimal hd-automata for the pi-calculus. *Theoretical Computer Science*, 340:539–576, 2005.

[40] U. Montanari, M. Pistore, and D. Yankelevich. Efficient minimization up to location equivalence. In *ESOP*, pages 265–279, 1996.

[41] M. Pistore. *History Dependent Automata*. PhD thesis, Università di Pisa, Dipartimento di Informatica, 1999. available at University of Pisa as PhD. Thesis TD-5/99.

[42] S. Staton. Name-passing process calculi: operational models and structural operational semantics. Technical Report UCAM-CL-TR-688, University of Cambridge, Computer Laboratory, 2007.

[43] E. Tuosto. *Non-Functional Aspects of Wide Area Network Programming*. PhD thesis, Dipartimento di Informatica, Università di Pisa, May 2003. TD-8/03.

[44] Vincenzo Ciancia. *Accessible Functors and Final Coalgebras for Named Sets*. PhD thesis, University of Pisa, 2008.

[45] G. Winskel. Symmetry and concurrency. In *CALCO*, pages 40–64, 2007.

[46] J. Worrell. Terminal sequences for accessible endofunctors. *Electr. Notes Theor. Comput. Sci.*, 19, 1999.