CATEGORICAL LOGIC OF CONCURRENCY AND INTERACTION. I: SYNCHRONOUS PROCESSES

DUŠKO PAVLOVIĆ
Department of Computing, Imperial College
180 Queen's Gate, London SW7 2BZ, United Kingdom
E-mail: D.Pavlovic@doc.ic.ac.uk

ABSTRACT

This is a report on a mathematician's effort to understand some concurrency theory. The starting point is a logical interpretation of Nielsen and Winskel's [30] account of the basic models of concurrency. Upon the obtained logical structures, we build a calculus of relations which yields, when cut down by bisimulations, Abramsky's interaction category of synchronous processes [2]. It seems that all interaction categories arise in this way. The obtained presentation uncovers some of their logical contents and perhaps sheds some more light on the original idea of processes as relations extended in time.

The sequel of this paper will address the issues of asynchrony, preemption, noninterleaving and linear logic in the same setting.

1 Introduction

Concurrency in computation is modelled in many different ways. Several attempts at unification have been made. Most recently, Abramsky [1, 2] has proposed the paradigm of *relations extended in time* as a foundation for theory of processes. His *interaction categories* are meant to capture this paradigm formally. Their structure is not fully axiomatized, but rather thoroughly motivated on examples.

In this paper, we make the way from some standard models of concurrency to the primal example of an interaction category — the category of synchronous processes SProc [2, sec. 2]. We begin, in section 2, by surveying some basic notions of categorical logic, particularly regular fibrations [27] and the induced

calculi of relations — which will turn out to be the source of much of the structure of process calculi. In section 3, the simplest models of processes, automata and synchronisation strees, are shown to form regular fibrations, with rich intrinsic logic. The structures spelled out in [30] (albeit in a different form) are analysed logically. Processes thus appear as predicates over the corresponding alphabets. In subsection 3.3, we refine them to predicates over safety specifications: logic of trees is relativised over logic of specifications. A non-logical sequent calculus, capturing logic of trees, is described in section 4. Concurrency theorists will recognize it as the "pre-equational" part of SCCS, extended with two apparently new operations, which correspond to the implication and the universal quantification of trees. While the logical contents of the basic process operations appear rather clearly, the computational significance of the implication remains to be explored.

Finally, the bicategory of relations, induced by the regular logic of trees as predicates over specifications, is aligned with Abramsky's interaction category of synchronous processes SProc in section 5. Even formally, it appears as an embodyment of Abramsky's paradigm of *processes as relations in time*. But SProc is not just a formal calculus of relations, though: it is a proper *quotient* of the bicategory of relations induced by regular logic of trees. The most interesting part is explaining this quotient. It is induced by the equational part of SCCS, which is also incorporated in the picture in section 5.

As a representative of a process, an automaton or a tree may be redundant: its geometry is in principle not completely determined by its computational behaviour; the other way around, some of its properties are computationally irrelevant. Therefore, processes are actually defined as classes of computationally equivalent, *bisimilar* representatives, rather than individual trees or automata. The equations of process calculi identify such classes, and determine the notion of of bisimilarity. It seems that the kernel of *process calculus is obtained by cutting down the logic of trees modulo bisimilarity.* The bisimilarity thus drives concurrency off the road of universal logical operations, and into an area unexplored in categorical logic.

In order to present processes modulo bisimilarity, we do not consider the formal bisimilarity quotient of the regular fibration of synchronisation trees, but rather use the concrete representation developed in [28]. Its main feature is that, on the chosen representatives, the graph morphisms capture all the computationally sound simulations — namely those that preserve bisimilarity. In [28, sec. 2.3], we called such simulations *sober*, and argued that they are *the* natural choice of computationally relevant morphisms.

The obtained category of processes uncovers a remarkable phenomenon:

while logical operations do preserve bisimilarity of representatives, they do not preserve bisimilarity of the sober simulations between those representatives. The upshot is that *the process operations cannot be extended to functors*, acting not only on processes but also on morphisms between them! Strong bisimilarity is a congruence with respect to the object part of the logical operations on trees (which is why they can induce the process operations on the bisimilarity classes in the first place); but it is *not* a congruence with respect to the arrow part (at least not on the bisimilarity preserving simulations, for which I see no alternative).

A straightforward solution of the problem, however, has already been built in into SProc: collapse the simulations $P \to Q$ to the *similarity* preorder $P \prec Q$, recording only whether Q can simulate P or not, and disregarding the distinctions between particular simulations. Indeed, SProc is not just quotient of a bicategory of relations, but also its preorder collapse. However, may a need for a more precise account of the dynamics of simulations ever arise, a different solution will have to be sought: a refined notion of bisimilarity, which will be a congruence not only with respect to processes, but also with respect to morphisms between them.

2 Elements of Categorical Logic

In their book of 1958, Curry and Feys remarked about a "striking analogy between the theory of functionality and the theory of implication" [9, section E]. Much later, this analogy had been developed into the paradigm of *propositions-as-types*, permeating logical frameworks for computation. The λ-calculus provides means for encoding constructive proofs as abstract functions.

Categorically, that "striking analogy" is explained by an adjunction: both the function-space constructor $(-)^A$ and the implication $\alpha \Rightarrow (-)$ arise as right adjoints — namely, of the functors $A \times (-)$ and $\alpha \wedge (-)$ respectively, which, on their own account, arise as analogous right adjoints as well.

$$\frac{C \xrightarrow{a} A \qquad C \xrightarrow{b} B}{C \xrightarrow{\langle a,b \rangle} A \times B} \qquad \frac{\gamma \vdash \alpha \qquad \gamma \vdash \beta}{\gamma \vdash \alpha \wedge \beta}$$

$$\frac{A \times B \xrightarrow{f} C}{B \xrightarrow{f'} C^A} \qquad \frac{\alpha \wedge \beta \vdash \gamma}{\beta \vdash \alpha \Rightarrow \gamma}$$

Based on this observation, made in the early sixties, Lawvere had defined *cartesian closed categories*, the variable-free alternative to typed λ-calculus. Pursuing

the idea of *logical-operations-as-adjunctions*, he went on to introduce *hyperdoctrines* [17][1], the structure capturing higher order predicate logic.

2.1 Hyperdoctrines

A hyperdoctrine $\wp$ is, basically, a contravariant functor from a "category of sets" $\mathcal{S}$ to the category CAT of categories. The category $\wp A$ is meant to be a "category of predicates" over the "set" $A \in \mathcal{S}$. The arrows of $\wp A$ are "proofs", while the arrows of $\mathcal{S}$ are "functions". The cartesian closed structures of $\wp A$ and of $\mathcal{S}$ respectively represent the basic *propositional* and the *set-theoretical* operations. All of them can be determined entirely in terms of adjunctions.

The structure of a functor $\wp : \mathcal{S}^{op} \to \mathsf{CAT}$ itself will correspond to the main operations of *predicate logic*. Its arrow part yields the substitution mechanism: given a "function" $f : A \to B$, the functor $\wp f : \wp B \to \wp A$ maps a "predicate" $Q(y) \in \wp(B)$ to $Q(f(x)) \in \wp A$. Following a simple but fundamental observation of Lawvere's[17], the quantifiers are implemented as the adjoints of such substitution functors

$$\exists f \dashv \wp f \dashv \forall f : \wp(A) \to \wp(B).$$

If $\wp f(Q)$ represents $Q(f(x))$, then $\exists f(P)$ should be read as $\exists x.f(x){\equiv}y \wedge P(x)$, and $\forall f(P)$ as $\forall x.f(x){\equiv}y \to P(x)$. With this interpretation, it turns out that the idea of quantifiers as adjoints to the substitution is already present in their definitions in sequent calculus, which can be summarized in the following invertible rules.

$$\frac{P(x) \vdash Q(f(x))}{\exists x.f(x){\equiv}y \wedge P(x) \vdash Q(y)}\,(\exists) \qquad \frac{Q(f(x)) \vdash P(x)}{Q(y) \vdash \forall x.f(x){\equiv}y \to P(x)}\,(\forall)$$

A hyperdoctrine is thus a functor $\wp : \mathcal{S}^{op} \to \mathsf{CCC}_{\exists\forall}$, where $\mathcal{S}$ is a cartesian closed category, and $\mathsf{CCC}_{\exists\forall}$ is the category of cartesian closed categories, where the structure preserving functors having both adjoints play the role of morphisms. That is the basis for abstract development of higher-order predicate logic. There is just one subtle point, which will be illustrated by example (iii) below.

Examples. (i) The simplest example of a hyperdoctrine is the ordinary power-set functor. The base category $\mathcal{S}$ is thus Set; to each set A the functor $\wp$ assigns the Boolean algebra $\wp A$ of its subsets. The functors $\wp f : \wp B \to \wp A$ map

[1] The term *doctrine* was previously used to denote a monad or an equational theory on CAT.

subsets of B to their inverse images along $f : A \to B$, while $\exists f : \wp A \to \wp B$ maps subsets to their direct images. In hyperdoctrines and related structures, we generally speak of the *inverse image functors* $\wp f$ and of the *direct image functors* $\exists f$ and $\forall f$ — on the *left* and on the *right* respectively. The inverse image functors are usually written $f^{\#} : \wp B \to \wp A$; a standard alternative for $\exists f$ is $f_!$, while $\forall f$ is often written $f_{\#}$. In calculations, this alternative notation seems more convenient.

(ii) Another simple example is the hyperdoctrine of *languages*. A language on a set Σ, seen as an alphabet, is simply set of finite sequences — *words* in Σ. Languages on Σ thus form Boolean algebra $\mathcal{K}(\Sigma) = \wp(\Sigma^*)$, where Σ^* is the free monoid generated by Σ. A function $f : \Sigma \to \Gamma$ induces a monoid morphism $f^* : \Sigma^* \to \Gamma^*$, which then induces the inverse and the direct images $\exists f \dashv f^{\#} \dashv \forall f : \mathcal{K}(\Sigma) \to \mathcal{K}(\Gamma)$.

In fact, we shall only consider the prefix closed languages here. A language S is *prefix closed* if for all $s, t \in \Sigma^*$, $st \in S$ implies $s \in S$. Since this implies that the empty word is contained everywhere, one usually considers only the *nonempty* prefix closed languages: the empty one leads to exceptions all the time. On the other hand, the nonempty prefix closed languages can be thought of as *specifications*. Of course, the complement of a specification is not a specification, and the sublattices $\mathcal{L}(\Sigma) \subseteq \mathcal{K}(\Sigma)$ of specifications are not Boolean. But they are complete Heyting algebras, thus cartesian closed. We have a hyperdoctrine $\mathcal{L} : \mathsf{Set}^{op} \to \mathsf{CCC}_{\exists\forall}$ again.

(iii) Finally, let $\mathcal{S}$ be the category Pos of posets and let the functor $\nabla : \mathsf{Pos}^{op} \to \mathsf{CCC}_{\exists\forall}$ take a poset A to the induced complete Heyting algebra of lower sets. The inverse and the direct images are obtained by taking the suitable lower closures. All the structure needed for predicate logic is available. Yet there is a sense in which this logic will be unsound.

Given a square

$$\begin{array}{ccc} A & \xrightarrow{f} & B \\ {\scriptstyle g}\downarrow & & \downarrow{\scriptstyle s} \\ C & \xrightarrow[t]{} & D \end{array} \qquad (1)$$

in $\mathcal{S}$ is a pullback, it seems reasonable to require that the equation $s(y) \equiv t(z)$ implies $\exists x. f(x) \equiv y \wedge g(x) \equiv z$. It follows that for every predicate $Q(y)$ over B, the formula $\exists y. s(y) \equiv t(z) \wedge Q(y)$ should imply $\exists x. g(x) \equiv z \wedge Q(f(x))$.

Well, for the hyperdoctrine ∇, it does not! Take, for instance, the arrows s and t on (1) to be inclusions of $\mathbf{1} = \{*\}$ in $\mathbf{2} = \{0 \leq 1\}$, with $s(*) = 1$ and $t(*) = 0$. So we have $B = C = \mathbf{1}$, $D = \mathbf{2}$ and the pullback of s and t is $A = \mathbf{0}$.

Now if $Q \in \nabla(\mathbf{1})$ is $\mathbf{1}$, then $(\exists y.s(y) \equiv t(z) \wedge Q(y)) = t^{\#}(s_!(Q))$ is $\mathbf{1}$ again, while $(\exists x.g(x) \equiv z \wedge Q(f(x))) = g_!\left(f^{\#}Q\right)$ is empty. So the former does not imply the latter.

To preclude pathologies of this kind, a hyperdoctrine must satisfy stability conditions below.

Definition. (Lawvere [18]) *A functor* $\wp : \mathcal{S}^{op} \to \mathsf{CCC}_{\exists\forall}$ *is a hyperdoctrine if the canonical arrows in* $\wp C$

$$g_!\left(f^{\#}Q\right) \;\to\; t^{\#}\left(s_!Q\right) \text{ and} \tag{2}$$

$$g_!\left(P \wedge g^{\#}R\right) \;\to\; g_!(P) \wedge R \tag{3}$$

derived from the adjunctions, are isomorphisms, for all $P \in \wp A$, $Q \in \wp B$ *and* $R \in \wp C$, *for any pullback* (1), *and any arrow* $g : A \to C$ *in* $\mathcal{S}$. *The former requirement is known as the* Beck-Chevalley *condition, the latter as the* Frobenius *condition.*

As explained in [25], without these conditions, the interpretation of variables goes wrong: different variables interfere in substitution and quantification. We shall come back to this point later.

2.2 Indexed and Fibred Categories

For the present paper the full structure of hyperdoctrine will not be necessary. It will be mostly concerned with *regular logic*, which is limited to conjunction $\wedge$ and the existential quantification $\exists$ over sorts that can be combined using the finite products $\times$. Of course, a basic substitution mechanism must be provided as well, as reindexing of "predicates" along "functions" between "sets". For certain purposes, it will be more convenient to present this mechanism in a *fibred* rather than *indexed* form. This distinction is the theme of the present subsection. Regular logic will be explained in subsection 2.4.

In the world of sets, every indexed family $\langle A_i \rangle_{i \in I}$ can be presented as a function $a : A \to I$, with $A = \sum A_i$. The components A_i can be recovered from this function as the *fibres* $a^{-1}(i)$. Hence the equivalence of categories $\mathsf{Set}^I \simeq \mathsf{Set}/I$.

When I is a topological space, the continuous I-indexed families can be presented as *sheaves*. Again, every I-sheaf induces a morphism — a continuous function to I. This time, however, not every such function comes from a sheaf: only the local homeomorphisms over I correspond to some sheaves. Hence the embedding $\mathsf{Sh}(I) \hookrightarrow \mathsf{Sp}/I$ of sheaves into spaces over I. But this passage is

less trivial than the previous one, from Set^I to Set/I. In fact, there is a sense in which the total view of a sheaf, as a space in Sp/I, can offer essentially more insight than its indexed form in $\mathsf{Sh}(I)$. For instance, when I is a circle, the sheaf corresponding to the projection of a cylinder can be distinguished from the sheaf corresponding to the projection of a Möbius band only by the global view. Similar, perhaps more convincing examples can be made in higher dimensions (e.g., using torus and Klein's bottle, or various combinations). The same phenomena occur on the level of categories.

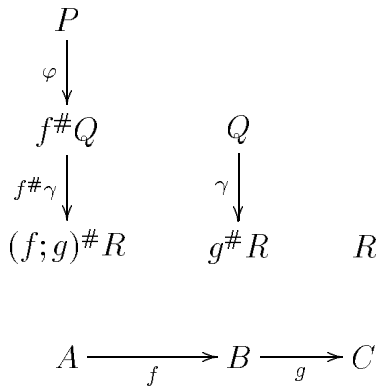
An $\mathcal{S}$-indexed category is in principle something like a functor $\mathcal{C} : \mathcal{S}^{op} \to \mathsf{CAT}$. The idea is still that $\mathcal{S}$ is a "category of sets". Each category $\mathcal{C}(A)$ is now thought of as the category $\mathbb{C}^A$ of A-indexed families from some $\mathbb{C}$. Ordinary category theory can be viewed as the theory of Set-indexed categories: each category $\mathbb{C}$ can be presented as a functor $\mathbb{C}^{\cdot} : \mathsf{Set}^{op} \to \mathsf{CAT}$, with $\mathbb{C}^{\cdot}(A) = \mathbb{C}^A$ and $\mathbb{C}^{\cdot}(f) = f^{\#} : \mathbb{C}^B \to \mathbb{C}^A$ reindexing each $B \xrightarrow{Q} \mathbb{C}$ as $A \xrightarrow{f} B \xrightarrow{Q} \mathbb{C}$. In a sense, categories are tacitly given in this form, together with all its powers and reindexing. Basic categorical concepts need not be much modified to be expressed in this setteing. For instance, $\mathbb{C}$ will have all the set-indexed (co)products if and only if $\mathbb{C}^{\cdot}$ has the right (resp. left) direct images. Note that an indexed category in this form always satisfies the Beck-Chevalley condition.

Passing from Set to an abstract base $\mathcal{S}$, one lifts the ordinary category theory in abstract, $\mathcal{S}$-indexed theory: for instance, the (co)products are *defined* as the right (resp. left) direct images, satisfying the Beck-Chevalley condition. Other concepts are dealt with in a similar fashion.

The Grothendieck construction. The transition from indexed categories $\mathsf{CAT}^{\mathcal{S}^{op}}$ to a fibred presentation in $\mathsf{CAT}/\mathcal{S}$ is actually simpler than the passage from sheaves $\mathsf{Sh}(I) \subseteq \mathsf{Set}^{\mathcal{O}(I)^{op}}$ to local homeomorphisms from Sp/I, because it does not involve reconstruction of points. Given a functor $\mathcal{C} : \mathcal{S}^{op} \to \mathsf{CAT}$, we form (abusing notation) a category $\mathcal{C}$ by glueing all $\mathcal{C}(A)$ together. The fibration $\mathsf{C} : \mathcal{C} \to \mathcal{S}$ then projects each $\mathcal{C}(A)$ to A.

More precisely, the objects of $\mathcal{C}$ are pairs $\langle A, P\rangle$, where $P \in \mathcal{C}(A)$. A $\mathcal{C}$-arrow from $\langle A, P\rangle$ to $\langle B, Q\rangle$ is a pair $\langle f, \varphi\rangle$, with $f : A \to B$ in $\mathcal{S}$, and $\varphi : P \to f^{\#}Q$ in $\mathsf{C}(A)$. The composite of $\langle f, \varphi\rangle : \langle A, P\rangle \to \langle B, Q\rangle$ and $\langle g, \gamma\rangle : \langle B, Q\rangle \to \langle C, R\rangle$

is the pair $\big\langle (f;g), (\varphi; f^{\#}\gamma) \big\rangle$.



(4)

Obviously, the functor C should project each pair to the first component. This functor is the *fibration* induced by $\mathcal{C} : \mathcal{S}^{op} \to \mathsf{CAT}$. The category $\mathcal{C}$ is *fibred* by C over $\mathcal{S}$.

Examples. **(i)** For every category $\mathbb{C}$, the indexed category $\mathbb{C}^{\cdot} : \mathsf{Set}^{op} \to \mathsf{CAT}$ can also be viewed as the fibred category $\mathbb{C}^{\cdot}$. Its object are all the set-indexed families of objects from $\mathbb{C}$. A morphism from $A \xrightarrow{P} \mathbb{C}$ to $B \xrightarrow{Q} \mathbb{C}$ is a pair $\langle f, \varphi \rangle$, where $f : A \to B$ is a function while $\varphi : P \to (f;Q)$ is a natural transformation — in fact, just a family $\langle \varphi_x : P(x) \to Q(f(x)) \rangle_{x \in A}$.

(ii) The powerset hyperdoctrine $\wp : \mathsf{Set}^{op} \to \mathsf{CAT}$ induces the fibred *category of predicates* $\wp$. The objects of $\wp$ are pairs of sets $\langle A \supseteq P \rangle$. An arrow from $\langle A \supseteq P \rangle$ to $\langle B \supseteq Q \rangle$ is a function $f : A \to B$ which maps P to Q. Formally, this means $f_! P \subseteq Q$, or equivalently $P \subseteq f^{\#}Q$. This latter inclusion is φ in the formal morphism $\langle f, \varphi \rangle$ as above. But if there is such a companion for f, it is unique, so we need not mention it explicitly.

(iii) The fibred category of specifications $\mathcal{L}$ is obtained by glueing together all the lattices $\mathcal{L}(A)$. The objects of $\mathcal{L}$ are thus pairs $A = \langle \Sigma_A, S_A \rangle$, where S_A is a prefix closed language in alphabet Σ_A. A morphism $f : A \to B$ in $\mathcal{S}$ is a function $f : \Sigma_A \to \Sigma_B$ which maps every word $s \in S_A$ to a word $f^*(s) \in S_B$. (Of course, the monoid morphism $f^* : \Sigma_A^* \to \Sigma_B^*$ takes each string $a_1 a_2 \cdots a_n$ to $f(a_1) f(a_2) \cdots f(a_n)$.)

What is a fibration? How do we recover the indexed category $\mathcal{C} : \mathcal{S}^{op} \to \mathsf{CAT}$ from the derived fibration $\mathsf{C} : \mathcal{C} \to \mathcal{S}$? Clearly, the categories $\mathcal{C}(A)$ are isomorphic with the fibres $\mathcal{C}_A$ of C. By definition, a *fibre* $\mathcal{C}_A$ is the subcategory of $\mathcal{C}$ over A and its identity. We say that an object or arrow $X \in \mathcal{C}$ is *over* $A \in \mathcal{S}$ if $\mathsf{C}(X) = A$.

To recover the arrow part of the functor $\mathcal{C}$, observe that the inverse image

of $Q \in \mathcal{C}(B)$ along $f : A \to B$ appears in the domain of the arrow $\vartheta^f_Q = \langle f, \mathrm{id}\rangle : \langle A, f^\# Q\rangle \to \langle B, Q\rangle$ in $\mathcal{C}$. Moreover, every arrow over f to Q factorizes through such ϑ^f_Q through a unique *vertical* arrow, i.e. from the fibre.

$$\begin{array}{ccc} \langle A, P\rangle & & \\ {\scriptstyle \langle \mathrm{id},\varphi\rangle}\downarrow & \searrow^{\langle f,\varphi\rangle} & \\ \langle A, f^\# Q\rangle & \xrightarrow[\langle f,\mathrm{id}\rangle]{} & \langle B, Q\rangle \end{array} \qquad (5)$$

$$A \xrightarrow[f]{} B$$

In other words, ϑ^f_Q is couniversal among the $\mathcal{C}$-arrows over f to Q. The functor $f^\# : \mathcal{C}(B) \to \mathcal{C}(A)$ can be recovered from this couniversal property: each object $Q \in \mathcal{C}(B)$ whould go to the domain of ϑ^f_Q; and the couniversality of this arrow determines the arrow part of $f^\#$.

These couniversal arrows are called *cartesian*. They characterize fibrations, i.e. the functors obtained by the Grothendieck construction, or equivalent to such functor in a strong sense. A functor will be a fibration if and only if the vertical and the cartesian arrows induced by it form a factorisation system.

Definition. *Consider a functor* $\mathsf{E} : \mathcal{E} \to \mathcal{S}$. *An* $\mathcal{E}$*-arrow* $\vartheta : Q' \to Q$ *is* cartesian *with respect to* E *if for every* $\varphi : P \to Q$, *with* $\mathsf{E}(\varphi) = \mathsf{E}(\vartheta)$, *there is a unique vertical arrow* $\varphi' : P \to Q'$ *such that* $\varphi = (\varphi'; \vartheta)$.

We say that E has cartesian liftings *in if for every arrow* $f : A \to B$ *in* $\mathcal{S}$ *and every* $Q \in \mathcal{E}_B$ *there is a cartesian arrow* ϑ *over* f, *with the codomain* Q. *Such liftings will be generically denoted* $\vartheta^f_Q : f^\#(Q) \to Q$.

$\mathsf{E} : \mathcal{E} \to \mathcal{S}$ *is a* fibration *if it has cartesian liftings and they are closed under the composition.*

A morphism of fibrations $\mathsf{F} : \mathsf{E} \to \mathsf{E}'$ *is a functor* $\mathsf{F} : \mathcal{E} \to \mathcal{E}'$ *such that* $\mathsf{E} = \mathsf{F}\mathsf{E}'$, *preserving the cartesian arrows.*

As explained above, the reconstruction of the the *total* fibred category from an indexed category may yield insights, but also technical advantages. Having all "predicates" together in one category, rather than stratified over the corresponding "sets", may be indispensable in some logical constructions [27].

Another important point is that the crucial part of the *structure* of indexed categories — the substitution functors $f^\# : \mathcal{C}(B) \to \mathcal{C}(A)$ — get encoded in fibrations as a couniversal *property*. Transformations of this kind distinguish the categorical approach from the algebraic one. On the way back, the original

structure is recovered up to isomorphism. Indeed, an arrow $f : A \to B$ can have many isomorphic cartesian liftings at each Q over B, determining many isomorphic inverse images $f^{\#}Q$. In order to extract an indexed category from an abstractly given fibration, one must make choices among isomorphic representants of each $f^{\#}Q$. It may not be possible to make these choices in such a way as to ensure strict equality between the functors $(f;g)^{\#}$ and $\left(g^{\#};f^{\#}\right)$. However, *any* choice will yield them canonically isomorphic. In principle, fibrations over $\mathcal{S}$ thus do not induce functors, but *pseudo*functors $\mathcal{S}^{op} \to \mathsf{CAT}$, which preserve composition only up to a family $c^{fg} : (g^{\#};f^{\#}) \to (f;g)^{\#}$ of coherent natural isomorphisms [13]. Pseudofunctors where c^{fg} are identities are just functors.

So perhaps the reader will now be willing to accept the inconvenience of allowing pseudofunctors, with all their coherences, among indexed categories too. Such indexed categories come about naturally. For instance, it should be possible to develop $\mathcal{S}$ itself into an $\mathcal{S}$-indexed category: it is the "category of sets" there. As a Set-indexed category, Set is, of course, the functor $\mathsf{Set}^{\cdot} : \mathsf{Set}^{op} \to \mathsf{CAT}$, with $\mathsf{Set}^{\cdot}A = \mathsf{Set}^{A}$. However, if $\mathcal{S}$ is not Set, the category $\mathcal{S}^{A}$ may not be defined for abstract $A \in \mathcal{S}$. A solution of this problem is suggested by the equivalence $\mathsf{Set}^{A} \simeq \mathsf{Set}/A$. The slice $\mathcal{S}/A$ is defined for any $\mathcal{S}$ and all $A \in \mathcal{S}$. So we can put $\mathcal{S}^{\cdot}(A) = \mathcal{S}/A$ as the object part of $\mathcal{S}^{\cdot} : \mathcal{S}^{op} \to \mathsf{CAT}$. Looking at the example of sets again, we see that the reindexing $\mathcal{S}^{\cdot}(f) = f^{\#} : \mathcal{S}/B \to \mathcal{S}/A$ now becomes a choice of pullbacks along $f : A \to B$. But it will very seldom be possible to make this choice globally, in such a way that the pullback functors $(f;g)^{\#}$ and $(g^{\#};f^{\#})$ come out strictly equal. Most of the time, $\mathcal{S}^{\cdot} : \mathcal{S}^{op} \to \mathsf{CAT}$ will be incurably a pseudofunctor — with a choice of pullbacks and canonical isomorphisms between them.

On the other hand, the corresponding fibration is simply the codomain functor $\mathsf{Cod} : \mathcal{S}/\mathcal{S} \to \mathcal{S}$, where $\mathcal{S}/\mathcal{S}$ is the category of arrows from $\mathcal{S}$, with the commutative squares as morphisms. This is the *base fibration* over $\mathcal{S}$. Its cartesian arrows are the cartesian (i.e. pullback) squares in $\mathcal{S}$. (Hence the terminology.)

Using the ambiguity of notation, we shall freely switch between fibred and indexed categories.[2]

[2]For more on this theory, the reader may wish to consult the sources [14, 13, 23]. No suitable textbook exposition exists — *perhaps for a good reason.* Conceptually, the story of fibred and indexed categories contains very little beyond the familiar notions from ordinary categories. Technically, some of these notions do tend to become more demanding in an abstract setting, and need to be carefully worked out; but books which do such things for you are usually not much fun.

The "propositional" and the "set theoretical" operations are described in a fibred category in the same manner as in an indexed category — the latter in the base $\mathcal{S}$, the former in the fibres, and preserved by the inverse images. However, the notions related to the "quantification" appear in a different light.

Definition. *The functor* $\mathsf{E} : \mathcal{E} \to \mathcal{S}$ *is an* opfibration *if* $\mathsf{E}^{op} : \mathcal{E}^{op} \to \mathcal{S}^{op}$ *is a fibration. The cartesian arrows with respect to* E^{op} *are called* opcartesian *with respect to* E. *The opcartesian lifting of* $f : A \to B$ *at* P *over* A *is written* $\sigma_P^f : P \to f_!P$.

A functor is a bifibration *if it is both a fibration and an opfibration. A morphism of bifibrations must preserve both cartesian and opcartesian arrows.*

While fibrations over $\mathcal{S}$ induce contravariant functors $\mathcal{S}^{op} \to \mathsf{CAT}$, opfibrations induce covariant pseudofunctors $\mathcal{S} \to \mathsf{CAT}$. If E is a bifibration, the inverse image functors derived from E^{op} are left adjoint to the inverse image functors derived from E: the universal property of σ_P^f and the couniversal property of ϑ_Q^f (5) ensure this. Therefore, bifibrations induce pseudofunctors $\mathcal{S}^{op} \to \mathsf{CAT}_\exists$. The morphisms of $\mathsf{CAT}_\exists$ are the functors with a left adjoint.

If we start from a bifibration $\mathsf{E} : \mathcal{E} \to \mathcal{S}$ such that the fibrewise dual fibration $\mathsf{E}^o : \mathcal{E}^o \to \mathcal{S}$ is a bifibration too, we get a pseudofunctor from $\mathcal{S}^{op}$ to $\mathsf{CAT}_{\exists\forall}$, where morphisms have both adjoints. The category $\mathcal{E}^o$ is constructed from $\mathcal{E}$ by formally inverting the vertical arrows, and *only* them. The objects of $\mathcal{E}^o$ are thus the same as in $\mathcal{E}$; the arrows from P to Q are the isomorphism classes of spans in the form $P \overset{\psi}{\leftarrow} f^\# Q \overset{\vartheta}{\to} Q$, where ψ is vertical and ϑ cartesian. (Note that the arrows of $\mathcal{E}$ itself can be presented as the isomorphism classes of decompositions $P \overset{\varphi}{\to} f^\# Q \overset{\vartheta}{\to} Q$.)

A *hyperfibration*, corresponding to a hyperdoctrine, is thus a fibrewise cartesian closed fibration $\mathsf{E} : \mathcal{E} \to \mathcal{S}$, such that both E and E^o are bifibrations. Of course, the Beck-Chevalley and the Frobenius conditions must be satisfied. There are various ways to express the Beck-Chevalley condition for a bifibration, and even for a mere fibration [25]. For instance, the pseudofunctor induced by a bifibration will satisfy the Beck-Chevalley condition if and only if the opcartesian arrows are stable under the pullbacks along the cartesian arrows [27, prop. A.8(a)]. The induced pseudofunctor will satisfy the Frobenius condition if and only if the opcartesian arrows are stable under the pullbacks along the vertical projections [27, prop. A.8(b)]. For this reason, we sometimes refer to the Beck-Chevalley and the Frobenius conditions as the *stability* conditions.

Definition. *A functor* $\mathsf{E} : \mathcal{E} \to \mathcal{S}$ *is a* regular fibration *if*

- $\mathcal{S}$ *has the finite products,*
- $\mathcal{E}$ *has the finite fibrewise products,*
- E *is a bifibration, satisfying the Beck-Chevalley and the Frobenius conditions. (In other words,* $\mathcal{E}$ *has the small fibrational coproducts and they distribute over the finite products.)*

Morphisms of regular fibrations are bifibration morphisms preserving the finite fibrewise products.

This structure supports *regular logic*, the $\{\exists, \wedge\}$-fragment of the many-sorted first order logic with equality. Indeed, the structure of a regular fibration can be described in logical notation, with variables and the equality predicate. Let us explain this informally, but in some detail.

Firstly, recall that a "function" $f : A \to B$ can be "substituted" in $R(x, y) \in \mathcal{E}(A \times B)$ using the inverse image:

$$R\Big(x, f(x')\Big) = (\mathrm{id} \times f)^{\#}(R).$$

Multiple variables are manipulated using projections and diagonals. If the variable x needs to be repeated in two positions, it is substituted along the diagonal $d : A \to A \times A$.

$$R\Big(x, f(x)\Big) = d^{\#}\Big((\mathrm{id} \times f)^{\#}(R)\Big) = \langle \mathrm{id}, f \rangle^{\#}(R).$$

On the other hand, a dummy variable z of type $C \in \mathcal{S}$ can be substituted along the projection $p : A \times B \times C \to A \times B$:

$$R(x, y, \not{z}) = p^{\#}(R).$$

The conjunction corresponds to the fibrewise products and $R(x, y) \wedge S(y, z)$ actually denotes the product $R(x, y, \not{z}) \wedge S(\not{x}, y, z)$ in $\mathcal{E}(A \times B \times C)$.

The direct images along the projections correspond to the quantification:

$$\exists y.R(x, y) \;\; = \;\; p_!(R).$$

However, this makes sense only when the Beck-Chevalley and the Frobenius conditions are satisfied. Namely, the conveniton of writing the quantifiers on one side of the formula, while the substitution occurs on the other, is based on

the assumption that these two operations commute: the formula $\exists y.R(x,y,\not\equiv)$ does not tell whether the quantification, or the substitution of the dummy had occurred first. As explained in [25], the Beck-Chevalley condition ensures that the order of these operations indeed does not matter. The Frobenius condition, on the other hand, makes isomorphic the predicates $P(x) \wedge \exists y.R(x,y)$ and $\exists y.P(x) \wedge R(x,y)$. In a sense, these two conditions take care that different variables do not interfere.

While the direct images along the projections $p : A \times B \to A$ provide for the quantifiers, the direct images along the diagonals $d : B \to B \times B$ yield the equality predicate [18]

$$(y' \overset{B}{\equiv} y'') \quad = \quad d_!(\top_B).$$

Using the stability conditions, we get things like

$$\begin{array}{rcccc} \left(f(x) \overset{B}{\equiv} y\right) & = & (f \times B)^{\#}(\overset{B}{\equiv}) & \cong & \langle id_A, f\rangle_!(\top_A) \\ \left(\exists x.f(x) \overset{B}{\equiv} y\right) & \cong & p_!\left(\langle id_A, f\rangle_!(\top_A)\right) & \cong & f_!(\top_A) \\ \left(\exists x.f(x) \overset{B}{\equiv} y \wedge P(x)\right) & \cong & \ldots & \cong & f_!(P) \end{array}$$

Putting d for f in the second formula yields $(y' \equiv y'') \cong (\exists y.d(y) \equiv \langle y', y''\rangle)$, which perhaps sheds some light on the idea behind the above definition of the equality. Substituting in the last formula the identity for f shows that this equality is indeed sound for the substitution, since $P(x) \cong (\exists x'.x \equiv x' \wedge P(x'))$. More importantly, that last formula allows us to express all direct images in a logical form, using quantifiers and equations. But to interpret these quantifiers and equations, we only need direct images along projections and diagonals. So we can derive all direct images from this special class.

Proposition. *Consider a fibration $\mathsf{E} : \mathcal{E} \to \mathcal{S}$, and suppose that there are finite products in $\mathcal{S}$ and the finite fibrewise products in $\mathcal{E}$. Then E is a regular fibration as soon as it has stable opcartesian liftings of the projections and of the diagonals.*

2.5 Regular Logic and Calculus of Relations

Every regular fibration $\mathsf{E} : \mathcal{E} \to \mathcal{S}$ induces a *bicategory of relations* $\mathcal{R} = \mathsf{Rel}(\mathsf{E})$. The objects (0-cells) of $\mathcal{R}$ are the same as in $\mathcal{S}$. The hom-categories are

$$\mathcal{R}(A,B) \quad = \quad \mathcal{E}_{A \times B}. \tag{6}$$

The 1-cells — the *relations* — are thus the binary predicates $P(x,y)$ over $A \times B$, while the 2-cells are the proofs $\psi : P \to \tilde{P}$. As a 1-cell, a relation $P(x,y) \in \mathcal{R}(A,B)$ will be written $P : A \nrightarrow B$. The composite with $Q : B \nrightarrow C$ will be

$$(P\,;Q)(x,z) \;=\; \exists y.P(x,y) \wedge Q(y,z). \tag{7}$$

In standard notation, this would be

$$(P\,;Q) \;=\; r_! \left(p^{\#} P \wedge q^{\#} Q \right), \tag{8}$$

with projections $p : A \times B \times C \to A \times B$, $q : A \times B \times C \to B \times C$ and $r : A \times B \times C \to A \times C$. This construct is clearly functorial. Its arrow part thus tells how the 2-cells should be composed. The stability conditions ensure the associativity up to a canonical isomorphism, and also that the equality predicates

$$(\overset{A}{\equiv}) \;=\; d_!(\top_A) \tag{9}$$

can play the role of identity 1-cells.

Examples. Applied to the powerset hyperfibration (examples 2.1(i) and 2.2(ii)), the construction described above yields the ordinary bicategory of sets and relations. More generally, the calculus of relations in a category $\mathcal{C}$ arises from the regular fibration $\mathsf{Cod} : \mathsf{Mon}/\mathcal{C} \to \mathcal{C}$, where $\mathsf{Mon}/\mathcal{C}$ is the subcategory of the arrow category of $\mathcal{C}$, spanned by the monics. This fibration is regular if and only if the category $\mathcal{C}$ regular. Otherwise the relations in $\mathcal{C}$ do not form a bicategory.

Furthermore, every stable factorisation through a given family $\mathcal{M}$ of abstract "monics" in $\mathcal{C}$ induces a regular fibration $\mathsf{Cod} : \mathcal{M}/\mathcal{C} \to \mathcal{C}$ — and hence an abstract bicategory of relations. This was studied in [26]. More general regular fibrations, induced among others, by sites and triposes over $\mathcal{C}$, were studied in [27].

Structure. The structural correspondence of regular fibrations and the induced bicategories is rather subtle. In their book [11], Freyd and Scedrov have thoroughly analyzed it for the interval between regular categories and toposes. E.g., the former correspond to *unitary tabular allegories* [*op.cit.*, sec. 2.154]. Carboni and Walters' *cartesian bicategories* [7] accomodate similar analyses, even more general.

In the bicategory $\mathcal{R} = \mathsf{Rel}(\mathsf{E} : \mathcal{E} \to \mathcal{S})$, the product $\times$ from $\mathcal{S}$ induces a tensor product $\otimes$. The diagonals and the projections from $\mathcal{S}$ induce diagonals and projections in $\mathcal{R}$, but these do not form natural families. Indeed, the diagonals and projections from Set are not natural with respect to all relations in the ordinary (bi)category of relations, but just with respect to the total and

single-valued ones. In general, are *lax* natural, but this is not enough for a universal property. Indeed the coproducts from of sets become biproducts in the bicategory of relations — both ordinary and abstract.

The structure of cartesian bicategories seems to be based on the idea that the tensor $\otimes$ in $\mathcal{R}$ can be recognized as the rudiment of the cartesian structure of $\mathcal{S}$. The distinctive property of this $\otimes$ as a "relational view of the cartesian product" is the fact that each object carries a canonical structure of a $\otimes$-comonoid — with the diagonal as the comultiplication, and the terminal arrow as the augmentation. Hence, the structure of a cartesian bicategory thus consists of a tensor $\otimes$ and prescribes a canonical $\otimes$-comonoid structure on each object. These data are pinned down to diagonals and terminals from some old cartesian category by requiring that they are lax natural, and self-adjoint — which means that they are total and single-valued as relations[3], i.e. *maps*. Of course, the structure must be "disciplined" by natural but tedious coherence conditions, which Carboni and Walters have avoided restricting attention to the posetal case.

The bicategories of relations, induced by regular fibrations, are always cartesian — although seldom posetal. In fact, they are rather special cartesian bicategories: their logical origin is captured by the *discreteness* condition [7, def. 2.1], echoing the stability of logical operations (i.e., the Beck-Chevalley and the Frobenius conditions). At any rate, the described construction yields a functor[4] Rel from regular fibrations to cartesian bicategories.

Adjunction. In a suitable sense, the Rel-construction has a right adjoint. In principle, it echoes the reconstruction of the powerset hyperfibration from the ordinary bicategory of relations. In technical details, though, this general reconstruction is considerably more complicated, and reaches far beyond the scope and the needs of the present paper. We shall just outline the main steps, which glorify[5] the idea that *the passage from predicate logic to calculus of relations is just a change of presentation, preserving the contents and the expressive power.*

Given a cartesian bicategory $\mathcal{R}$, the associated regular fibration $\mathsf{E} : \mathcal{E} \to \mathcal{S}$ will be derived as follows.

The base category $\mathcal{S}$ will consist of the cocommutative $\otimes$-comonoids in $\mathcal{R}$, discrete in the mentioned Carboni-Walters sense [*op. cit.*], and such that the comonoid structure 1-cells are maps. It can be proved that the morphisms

[3]The unit of a self-adjoint relation says that it is total; the counit — that it is single-valued [27, sec. 4].

[4]some people would call it *bicategory homomorphism*, but I am a member of the club (founded probably by Kelly) which tries to use "functors", "adjunctions" etc. in all dimensions

[5]i.e., "lift into categories" (This is almost a technical term, first used Lawvere.)

between such comonoids are maps as well. The right adjoint of a comonoid morphism $P : A \mapsto B$ will be the relation $P^{\#}$, defined as the composite

$$\begin{array}{ccccc} B & \cong & B \otimes I & \xrightarrow{B \otimes h} & B \otimes A \otimes A \\ {\scriptstyle P^{\#}}\downarrow & & & & \downarrow{\scriptstyle B \otimes P \otimes A} \\ A & \cong & I \otimes A & \xleftarrow[e \otimes A]{} & B \otimes B \otimes A \end{array} \tag{10}$$

where the arrow $h : I \mapsto A \mapsto A \otimes A$ is obtained from the comonoid structure on A, while $e : B \otimes B \mapsto B \mapsto I$ is derived from the adjoints of the comonoid structure on B.

Since the 2-cells between maps like this are always iso, [26, sec. 3], the isomorphism classes of these comonoid morphisms will form an ordinary category $\mathcal{S}$, with trivial 2-cells. It is not hard to see [10] that the tensor $\otimes$ becomes the cartesian product in it.

The fibre $\mathcal{E}_A$ — the predicates over $A \in \mathcal{S}$ — should be the relations on A, which are captured in the hom-category $\mathcal{R}(I, A)$. Of course, I is the unit of $\otimes$. The direct and the inverse images along a map $P : A \mapsto B$ in $\mathcal{S}$ are now given by postcomposing with P, or its right adjoint $P^{\#}$ respectively.

$$\begin{array}{ccc} \mathcal{E}_B & = & \mathcal{R}(I, B) \\ {\scriptstyle P_!}\uparrow \dashv \downarrow{\scriptstyle P^{\#}} & & {\scriptstyle (-;P)}\uparrow \dashv \downarrow{\scriptstyle (-;P^{\#})} \\ \mathcal{E}_A & = & \mathcal{R}(I, A) \end{array} \tag{11}$$

The Frobenius condition for the resulting bifibration is just the the *modularity law*, derivable in a bicategory of relations — essentially using the discreteness condition [7, thm. 2.4]. On the other hand, to get an idea why the Beck-Chevalley condition is satisfied, notice that $(s_!; t^{\#}) \cong (f^{\#}; g_!)$, interpreted logically as in 2.1(iii), comes close to saying that the square formed by the maps f, g, s and t is a pullback (cf. [25, 28, (Appendix B)]).

Finally, the fibrewise product $R \wedge_A \check{R} \in \mathcal{E}_A$, for $R, \check{R} \in \mathcal{R}(I, A)$ will be defined using the diagonals and their adjoints:

$$\begin{array}{ccc} I & \xrightarrow{\cong} & I \otimes I \\ {\scriptstyle R \wedge \check{R}}\downarrow & & \downarrow{\scriptstyle R \otimes \check{R}} \\ A & \xleftarrow[d']{} & A \otimes A \end{array} \tag{12}$$

Some typical examples of regular fibrations can be found in [27]. They can be thought of as a common generalisation of sites and triposes. Here, we shall first study two concrete regular fibrations, closely related to the examples from [30]: automata and synchronisation trees, fibred over the alphabets. Synchronisation trees actually form a hyperfibration: they have the implication and the universal quantifiers. In the end of the section, we refine them into a hyperfibration over specifications.

For simplicity, it can be assumed that the base category $\mathcal{S}$ is Set, although any pretopos with natural numbers will do as well.

3.1 Automata, Transition Systems

Nondeterministic automata can be viewed as directed graphs with labelled edges. The vertices are the states, the edges — the transitions. One vertex is distinguished as the initial state. Starting from it, the automaton reads some input — a symbol from a given alphabet — and tries to make an accordingly labelled transition. This is where nondeterminism comes in: there may be any number of such transitions from a state; or none, in which case the automaton is *deadlocked*. If it is not, it chooses a suitable transition, and then reads the next symbol from the input and looks for a suitable transition again. It runs like this until it exhausts the input, or deadlocks. In the former case, we say that it has *accepted* the word. A *run* of an automaton is thus a directed path of transitions. The *trace* of a run is the word accepted in it, i.e. its sequence of labels. The *accepted language* of an automaton — its trace — is, by definition, the set of traces of all runs through it.

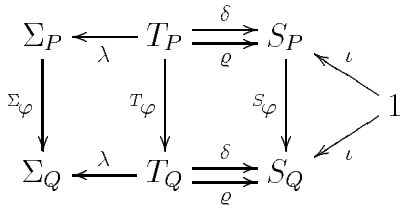
The structure of automata sometimes includes a distinguished set of final states as well. A run, as described above, is then required to terminate in one of final states. We shall assume that all states can be final. This means that the automaton can halt at any moment, so that the accepted language has to be prefix closed.

Formally, an automaton P will be a diagram

$$\Sigma_P \xleftarrow[\lambda]{} T_P \underset{\varrho}{\overset{\delta}{\rightrightarrows}} S_P \xleftarrow[\iota]{} 1 \tag{13}$$

in $\mathcal{S}$. The operations λ, δ and ϱ assign to each transition respectively a label, a source and a target; the constant ι is the initial state. A morphism $\varphi : P \to Q$

of automata is a natural transformation between the diagrams as (13).



(14)

So φ is in fact a triple of functions $\langle {}^{\Sigma}\varphi, {}^{T}\varphi, {}^{S}\varphi\rangle$. The latter two form a graph morphism, preserving the initial state and the labelling — in the sense that it takes an a-labelled transition to a ${}^{\Sigma}\varphi(a)$-labelled one. With morphisms like this, the automata form the category $\mathcal{A}$. It is fibred by the functor $\Sigma_{\mathcal{A}} : \mathcal{A} \to \mathcal{S}$, which projects each automaton to the corresponding alphabet. A morphism φ is cartesian if and only if the left-hand square in (14) is a pullback, while ${}^{S}\varphi$ is an isomorphism. A morphism φ is opcartesian if and only if ${}^{T}\varphi$ and ${}^{S}\varphi$ are both isomorphisms. Clearly, the opcartesian liftings always exist; and the cartesian liftins only require that $\mathcal{S}$ has pullbacks. The inverse image $f^{\#}Q$ along $f : A \to B$ will thus be constructed by putting on the set of the states S_Q exactly $f^{-1}(b)$ copies of each b-labelled transition. The direct image $f_!P$ is obtained by simply relabeling the underlying graph of P: just put $f(a)$ in place of a. The Beck-Chevalley and the Frobenius conditions are immediate.

The product $Q \wedge \widetilde{Q}$ within the fibre over $B \in \mathcal{S}$ is obtained by taking the product $S_Q \times S_{\widetilde{Q}}$ as $S_{Q\wedge\widetilde{Q}}$; and the set of the pairs of equally labelled transitions as $T_{Q\wedge\widetilde{Q}}$. This latter set can be obtained as a pullback of $\lambda : T_Q \to B$ and $\widetilde{\lambda} : T_{\widetilde{Q}} \to B$.

The functor $\Sigma_{\mathcal{A}} : \mathcal{A} \to \mathcal{S}$ is thus a regular fibration. Hence the predicate logic of automata. For future reference, let us mention that the coproducts in $\mathcal{S}$ yield the fibrewise coproducts in $\mathcal{A}$, which are calculated in a straightforward way.

An automaton is *deterministic* if each label determines exactly one transition from every state. This means that $\langle \delta, \lambda\rangle : T \longrightarrow S \times \Sigma$ must be an isomorphism. A *transition system*, on the other hand, is an automaton where there is at most one transition with a given label between any two given states. In other words, $\langle \delta, \lambda, \varrho\rangle : T \longrightarrow S \times \Sigma \times S$ must be monic.

We shall not go into deterministic automata here, and just remark that each automaton induces a transition system: the image of $\langle \delta, \lambda, \varrho\rangle$ determines the new set of transitions. In other words, we are identifying all the transitions with the same sources, same labels and same targets. Clearly, this operation changes nothing in the computational behaviour of the automaton. It determines a left

adjoint functor to the inclusion $\mathcal{A}^t \hookrightarrow \mathcal{A}$ of transition systems into automata. Both the inclusion and its reflection preserve the inverse images: if the codomain of a cartesian arrow is a transition system, the domain must be too. The transition systems thus form a fibred reflective subcategory of the category of automata. The fibrewise products are inherited. The direct images are induced by the reflection. So we have a regular fibration $\mathcal{A}^t \to \mathcal{S}$ again.

There is also a regular fibration $\mathcal{A}^r \to \mathcal{S}$, spanned by *reachable* automata, where each state can be reached from the initial state. The inclusion $\mathcal{A}^r \hookrightarrow \mathcal{A}$ has a right adjoint, i.e. reachable automata span a coreflective subcategory of automata. The unreachable states, just like the parallel transitions, are computationally irrelevant, and all three fibred categories, $\mathcal{A}$, $\mathcal{A}^t$ and $\mathcal{A}^r$ are conceptually equivalent, as far as the computation is concerned; but calculating in $\mathcal{A}$ is somewhat simpler. However, the fibred subcategory $\mathcal{T}$ of synchronisation trees, coreflective in all three of them, offers more significant advantages.

3.2 Trees

Let $\mathbf{N}$ be the poset of natural numbers

$$1 \longrightarrow 2 \longrightarrow 3 \longrightarrow \cdots$$

A tree can be understood as a presheaf over $\mathbf{N}$, i.e. a diagram P

$$P_1 \xleftarrow{p_1} P_2 \xleftarrow{p_2} P_3 \xleftarrow{p_3} \cdots$$

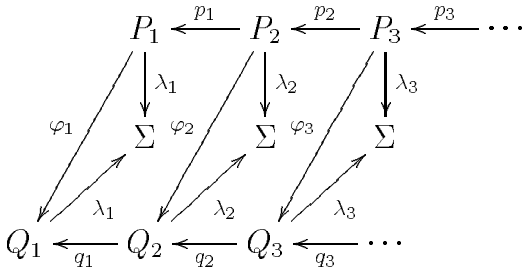
in $\mathcal{S}$. So trees form a presheaf topos $\mathcal{T}_1 = \mathcal{S}^{\mathbf{N}^{op}}$. The elements of each P_i are the vertices at the i-th level of P; the function p_{i-1} projects them on their predecessors. There is an edge from x to y if $x = p_{i-1}(y)$ for some i. The root is left implicit.

A *synchronisation tree* P can be presented as a diagram

$$\begin{array}{ccccccc} P_1 & \xleftarrow{p_1} & P_2 & \xleftarrow{p_2} & P_3 & \xleftarrow{p_3} & \cdots \\ {\scriptstyle\lambda_1}\downarrow & & {\scriptstyle\lambda_2}\downarrow & & {\scriptstyle\lambda_3}\downarrow & & \\ \Sigma & & \Sigma & & \Sigma & & \end{array} \tag{15}$$

in $\mathcal{S}$. It is thus simply a tree with all vertices, except the root, labelled from $\Sigma \in \mathcal{S}$. But this can be better understood as edge-labelling — since each vertex, except the root, receives exactly one edge.

Synchronisation trees are clearly presheaves again. They form a topos $\mathcal{T}$. A morphism $\varphi : P \to Q$ is a natural family, preserving the labelling and the predecessors.



(16)

If each synchronisation tree P is projected to its alphabet Σ, we get the fibration $\Sigma_{\mathcal{T}} : \mathcal{T} \to \mathcal{S}$. The topos $\mathcal{T}_1$ of unlabelled (or trivially labelled) trees appears as its fibre over 1. In fact, the other fibres are toposes too: the category $\mathcal{T}_\Sigma$ of Σ-labelled trees is isomorphic with the slice category $\mathcal{T}_1/G\Sigma$, where $G : \mathcal{S} \to \mathcal{T}_1$ is the right adjoint to the forgetful functor $\mathcal{T}_1 \to \mathcal{S}$, which maps each tree to the set of its vertices. The tree $G\Sigma$ is thus cofree over the set Σ. It is fairly easy to describe: starting from the root, add at each vertex exactly Σ edges. At the i-th level, there will be $(G\Sigma)_i = \Sigma^i$ vertices. The function $p_{i-1} : \Sigma^i \to \Sigma^{i-1}$ projects away the i-th component.

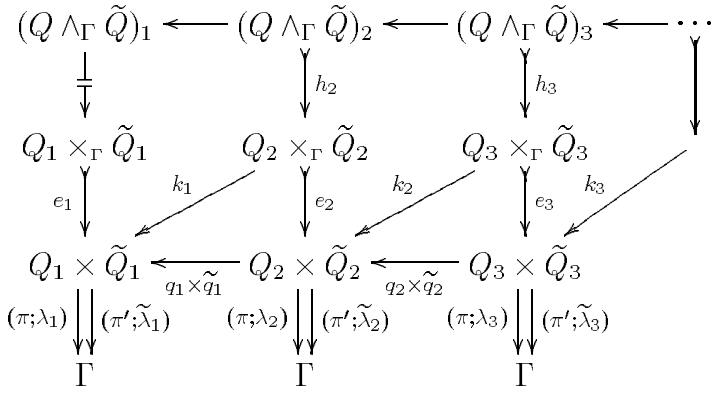
Standard topos theory now shows that $\Sigma_{\mathcal{T}} : \mathcal{T} \to \mathcal{S}$ is a hyperfibration. Synchronisation trees thus support full higher order predicate logic. We shall calculate some relevant operations, and try to grasp their meaning.

The left direct image of an Σ-tree P along $f : \Sigma \to \Gamma$ is again just its relabelling. If P is presented as on (15), we relabel it by postcomposing all λ_i with f. The inverse image of a Γ-tree Q along f is also based on a similar idea as for automata: a b-labelled edge in Q should be replaced by $f^{-1}(b)$ edges coming out of the same vertex. But these edges now introduce new vertices in the tree. The subtree of Q coming out of that b-labelled edge we started with should be copied above each of these new vertices, before we go on with replacing the other b-edges up the tree.

Similarly for the product $Q \wedge \widetilde{Q}$. Its vertices should, of course, be pairs of vertices from Q and $\widetilde{Q}$ labelled by the same letter. However, their predecessor should also be such a pair, and so on. So we come to the requirement that the paired vertices from Q and $\widetilde{Q}$ should have the same trace. The trace of a vertex is the sequence of labels obtained by climbing up the tree from the root to that vertex.

Let us formalize this construction, in order to see that indeed yields the product. The sets $(Q \wedge \widetilde{Q})_i$ of pairs of vertices with the same trace of length i

are obtained by recurrent pullbacks.

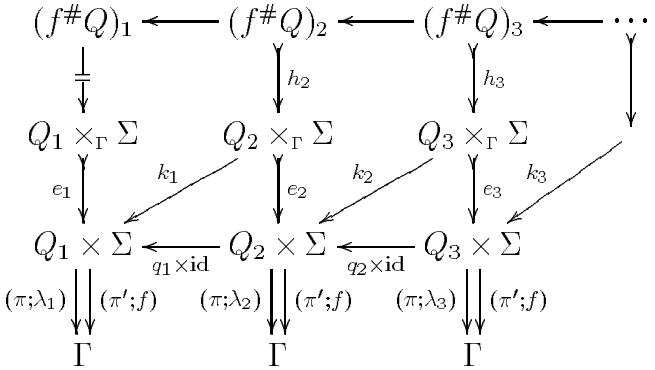


(17)

First calculate all the equalizers e_i (which are actually the pullbacks of $\lambda_i : Q_i \to \Gamma$ and $\lambda_1 : Q_i' \to \Gamma$). Hence $k_i = (e_{i+1}; (q_i \times q_i'))$. Starting from $h_1 = \mathrm{id}$, pull back each $(h_i; e_i)$ along k_i to get h_{i+1}.

One can directly check that this construction yields the product in $\mathcal{T}_\Gamma$; or observe that we have calculated a pullback over $G\Gamma$ in $\mathcal{T}_1$, which is the product in $\mathcal{T}_1/G\Gamma$.

The inverse image $f^\# Q$, described informally further above, are formalised along the same lines. The construction again follows a tower of pullbacks



(18)

which corresponds to pulling back along $Gf : G\Sigma \to G\Gamma$ in $\mathcal{T}_1$. This is the functor $f^\# : \mathcal{T}_1/G\Gamma \to \mathcal{T}_1/G\Sigma$.

Besides regular logic of $\wedge$ and $\exists$, the fibration $\Sigma_{\mathcal{T}} : \mathcal{T} \to \mathcal{S}$ also supports $\vee$, $\Rightarrow$ and $\forall$. The disjunction is represented by the fibrewise coproducts of trees, which are calculated in a completely straightforward way; the implication $Q \Rightarrow \tilde{Q}$ is the exponent in the topos $\mathcal{T}_1/G\Gamma = \mathcal{T}_\Gamma$; and the universal quantification $\forall f(Q) = f_\# Q$ is the dependent product along $Gf : G\Sigma \to G\Gamma$ in the topos $\mathcal{T}_1$. We spell out the latter two in some detail, as they seem not to have been noticed in a computational setting.

The descriptions are simplified by the fact that the topos $\mathcal{T}_\Gamma$ is generated by the tree Γ^*. The words $s \in \Gamma^*$ are represented as *branches*, i.e. the trees with at most one vertex at each level. The fact that they generate means that each Γ-tree is a colimit of Γ-branches[6]. We actually need a weaker fact, namely that each Γ-tree is a colimit of its one-word truncations:

$$Q \;=\; \varinjlim_{s\in\Gamma^*}(Q \wedge s) \tag{19}$$

Seen as the full subcategory of $\mathcal{T}_\Gamma$, the set Γ^* is a meet semilattice, with $s \leq s'$ means that s is a prefix of s'. The product $Q \wedge s$, on the other hand, is the subtree of Q consisting of those vertices which leave s or one of its prefixes as the trace[7]. This follows from the above description of the products. More precisely, for $s \in \Gamma^i$ and $j \leq i$, the set $(Q \wedge s)_j$ will consist of those vertices from Q_j which trace $s \restriction_j \in \Gamma^j$, the j-prefix of s. For $j > i$, $(Q \wedge s)_j$ will be empty. So $Q \wedge s$ is indeed the truncation of Q by s.

Using (19) we derive a useful representation of the hom-sets of $\mathcal{T}_\Gamma$

$$\begin{aligned}
\mathcal{T}_\Gamma(Q, \widetilde{Q}) &\overset{i}{\cong} \mathcal{T}_\Gamma\left(\varinjlim Q \wedge s, \widetilde{Q}\right) \\
&\overset{ii}{\cong} \varprojlim \mathcal{T}_\Gamma\left(Q \wedge s, \widetilde{Q}\right) \\
&\overset{iii}{\cong} \varprojlim \mathcal{T}_\Gamma(Q \wedge s, \widetilde{Q} \wedge s) \\
&\overset{iv}{\cong} \varprojlim_{s\in\Gamma^*} \mathcal{T}_1(Q \wedge s, \widetilde{Q} \wedge s).
\end{aligned} \tag{20}$$

Step i is just (19); step ii follows from the definition of a colimit; step iii is a consequence of the fact that every branch s is a subobject of the terminal object; finally, step iv is based on the fact that the elements of each $(Q \wedge s)_j$ are all labelled by the j-th letter of s — so that the label preservation is automatic. In other words, the $\mathcal{T}_\Gamma$-morphisms $Q \wedge s \longrightarrow \widetilde{Q} \wedge s$ can be viewed as ordinary tree morphisms.

In terms of representation (20), one directly proves that the following formulas define the exponents in $\mathcal{T}_\Gamma$ and the right direct image functor $f_\# : \mathcal{T}_\Sigma \to \mathcal{T}_\Gamma$ respectively:

$$(Q \Rightarrow_\Gamma \widetilde{Q})_i \;=\; \sum_{s\in\Gamma^i} \mathcal{T}_1(Q \wedge s, \widetilde{Q} \wedge s) \tag{21}$$

[6]This is meant internally in the topos $\mathcal{S}$. Otherwise, the statement is generally not true, since the terminal object 1 ("at most one vertex") does not generate.

[7]Recall that the trace of a vertex is the sequence of the labels of all its predecessors, plus its own.

$$(f_{\#}P)_i \;=\; \sum_{s\in\Gamma^i} \mathcal{T}_\Sigma(f^{\#}s, P) \qquad (22)$$

The elements of sets (21–22) are thus in the form $\langle s, \varphi\rangle$, where s is a word of the length i, and φ is a suitable morphism — a natural transformation where only the first i components are nontrivial. In both cases, the operation λ_i should assign to each $\langle s, \varphi\rangle$ the last element of s as the label. The predecessor operation $q_{i-1} : (Q \Rightarrow \widetilde{Q})_i \longrightarrow (Q \Rightarrow \widetilde{Q})_{i-1}$ should truncate $\langle s, \varphi\rangle$ to $\langle s \restriction_{i-1}, \varphi \restriction_{i-1}\rangle$. The same holds for $f_{\#}P$. Of course, $s \restriction_{i-1} \in \Gamma^{i-1}$ is the string s without the last element, while $\varphi \restriction_{i-1}$ is the natural transformation φ without the last component.

To better understand (21), let us describe a vertex $\langle s, \varphi\rangle \in (Q \Rightarrow \widetilde{Q})_i$ "in words". Suppose that s is the sequence $b_1 b_2 \ldots b_i$. First extract from Q_1 all the b_1-labelled vertices; among their successors in Q_2, extract only those that are labelled by b_2; from Q_3 take only the further successors that are labelled by b_3...So you have $Q \wedge s$. Do the same with $\widetilde{Q}$. Now φ is just a tree morphism between the obtained trees. It begins as an ordinary function at the level 1, assigning to each b_1-labelled vertex x_1 a b_1-labelled vertex $\varphi_1(x_1)$. Further assign to each b_2-labelled successor x_2 of x_1 a b_2-labelled successor $\varphi_2(x_2)$ of $\varphi_1(x_1)$. And so on.

Finally, to understand (22), first look at the tree $f^{\#}s$. It is a subobject in the terminal object $G\Sigma$ of $\mathcal{T}_\Sigma$. If b_j is the j-th element of s and if we write Σ_j for $f^{-1}(b_j)$, then

$$(f^{\#}s)_n \;=\; \prod_{j=1}^{n} \Sigma_j.$$

From each vertex of $f^{\#}s$ at the level n, there are exactly Σ_{n+1} edges coming out.

Now we describe $f_{\#}P$. At the level 1, a b_1-labelled vertex will be a choice of one a-labelled element of P_1 for each $a \in \Sigma_1$. To get a b_2-labelled element of $(f_{\#}P)_2$, among the successors of each previously chosen vertex from P_1 we must further select from P_2 an a-labelled vertex for each $a \in \Sigma_2$. The elements of $(f_{\#}P)_i$ will thus be the various copies of the trees $f^{\#}s$ in P, for various $s \in \Gamma^i$.

3.3 Relativising over Specifications

Mathematically, processes are usually presented either as languages, or as synchronisation trees, or as automata (transition systems). As explained in [30],

these three categories are related by adjunctions.

$$\mathcal{L} \underset{J}{\overset{S_{\mathcal{T}}}{\underset{\longrightarrow}{\longleftarrow}}} \mathcal{T} \underset{\longrightarrow}{\overset{\widehat{(-)}}{\longleftarrow}} \mathcal{A} \tag{23}$$

We comment on them briefly, since the presentations here are different. To present a tree P as an automaton P, augment it first with the root $p_0 : P_1 \to P_0 = 1$. The vertices of P are now in $S_P = \sum_{i=0}^{\infty} P_i$. Together, the predecessor operations p_i induce the obvious endomorphism $p : S_P \to S_P$. The edges of the graph P run from $p(y)$ to y; so the set of the transitions is $T_P = \{\langle x, y\rangle \in S^2 | x = p(y)\}$.

Hence the full and faithful functor $\mathcal{T} \to \mathcal{A}$. It is (bi)fibred, since it preserves the inverse (and the left direct) images. It does not preserve the products, so it cannot have a left adjoint. The right adjoint $\widehat{(-)} : \mathcal{A} \to \mathcal{T}$ is obtained by *unfolding* each automaton as the tree of its runs, i.e. the directed paths of starting from ι. They become the vertices of the associated tree.

To describe the unfolding of an automaton P formally, first consider the functor $\widetilde{P} : \mathcal{S}/S_P \longrightarrow \mathcal{S}/S_P$, which maps

$$\left(X \xrightarrow{\jmath} S_P\right) \quad \longmapsto \quad \left(\{\langle x, f\rangle \in X \times T_P | \jmath(x) = \delta(f)\} \xrightarrow{\pi} T \xrightarrow{\varrho} S\right).$$

What does $\widetilde{P}$ actually do? The domain of the arrow $\widetilde{P}(\iota)$, for instance, consists of all the edges coming out of the initial state ι. $\widetilde{P}(\iota)$ itself projects each of them onto its target. The edges coming out of these target states will be added if we further apply $\widetilde{P}$ to $\widetilde{P}(\iota)$. The domain of $\widetilde{P}^2(\iota)$ consists of the two-step runs of P, i.e. of the pairs of "composable" transitions starting from ι. In general, the set $\mathsf{Dom}\left(\widetilde{P}^i(\iota)\right)$ will consist of the runs of length i. The vertices of the unfolding of P at the level i should thus be

$$\widehat{P}_i \quad = \quad \mathsf{Dom}\left(\widetilde{P}^i(\iota)\right). \tag{24}$$

These sets form a synchronisation tree, since each $\mathsf{Dom}\left(\widetilde{P}(\jmath)\right)$ comes with the obvious projection to $\mathsf{Dom}(\jmath)$ and with the labelling inherited from T_P.

The trace functor $S_{\mathcal{T}} : \mathcal{T} \to \mathcal{L}$ assigns to each tree the language it accepts. The right adjoint $J : \mathcal{L} \to \mathcal{T}$ of $S_{\mathcal{T}}$, on the other hand, displays a specification $A = \langle \Sigma, S\rangle$ as a tree JA, with

$$(JA)_n \quad = \quad \prod_{j=1}^{n} S_j, \tag{25}$$

where $S_j \subseteq \Sigma^j$ is the set of all safe words of length j, contained S. Seen as a vertex, each word is labelled by its last symbol. Clearly, J is full and faithfull.

Viewed as a process, the language of a tree or an automaton is, in a sense, its "extensional collapse": it only records the performance, and neglects the behaviour. On the other hand, a *safety specification*, as a performance requirement, can also be represented as a prefix-closed language. A process is then said to satisfy such a specification if the accepted language is contained in it.

In a sense, the hyperfibration $\Sigma_{\mathcal{L}} : \mathcal{L} \to \mathcal{S}$ displays the logic of specifications as predicates over alphabets. The hyperfibration $\Sigma_{\mathcal{T}} : \mathcal{T} \to \mathcal{S}$ of synchronisation trees refers to the logic of processes. The fibred adjunction between $\mathcal{T}$ and $\mathcal{L}$, displayed on (23), allows us to refine this logic by *relativizing* it over specifications. This is achieved by a method sometimes applied in model theory: first, the universe is enlarged by new sets, definable in a given theory (in our case, by specifications); the logical operations are then relativized with respect to the added sets; finally, a new theory, using the relativised operations, is built over the refined universe.

Formalised categorically, the described procedure will expand the fibration of trees $\mathcal{T} \to \mathcal{S}$ into (at least) a regular *fibrewise* fibration over the fibration of specifications $\mathcal{L} \to \mathcal{S}$. In other words, we shall build a regular fibration in the category of (regular) fibrations over $\mathcal{S}$, rather than in the ordinary category of (regular) categories. Fortunately, (regular, resp. hyper)fibrations in the category of (regular, hyper)fibrations are just those morphisms the fibrewise components of which happen to be (regular, hyper)fibrations in the ordinary sense [24, prop. II.3.84(ii) and 85(ii)]. Because of this, expanding $\mathcal{T} \to \mathcal{S}$ into a fibrewise regular fibration over $\mathcal{L} \to \mathcal{S}$ turns out to be fairly simple.

First of all, the functor $S_{\mathcal{T}} : \mathcal{T} \to \mathcal{L}$ happens to be a fibration itself. The inverse image of a tree Q along an arrow $f : A \to S_{\mathcal{I}}Q$ in $\mathcal{L}$ is obtained by taking a pullback

$$\begin{array}{ccc} f^{\#}Q & \xrightarrow{\vartheta} & Q \\ \downarrow & & \downarrow \eta \\ JA & \xrightarrow[Jf]{} & JS_{\mathcal{I}}Q \end{array} \qquad (26)$$

in $\mathcal{T}$. The only problem is that $S_{\mathcal{T}}$ does not have the direct images. As they will be needed, we construct the free regular fibration generated by $S_{\mathcal{T}}$.

In principle, the free regular fibration induced by a fibration $\mathsf{E} : \mathcal{E} \to \mathcal{B}$ will be obtained by the comma construction $\mathsf{E}/\mathcal{B}$, provided that $\mathcal{B}$ has finite limits and $\mathcal{E}$ finite fibrewise products [6, 27, (3.4,7)]. But we must lift this construction from ordinary fibrations, in the category of categories, to fibrewise fibrations,

in the category of fibrations over $\mathcal{S}$. Therefore we take the *fibrewise* comma construction, instead of the ordinary one. Given some morphisms $\mathsf{F}, \mathsf{G} : \mathcal{E} \to \mathcal{E}'$ of fibrations, the objects of the fibrewise comma category F/G are the triples $\langle A, \mathsf{F}A \xrightarrow{f} \mathsf{G}B, B\rangle$, where f is a vertical arrow.

The free fibrewise regular fibration generated by $S_{\mathcal{T}} : \mathcal{T} \to \mathcal{L}$ will thus be the functor $S_{\mathcal{P}} : \mathcal{P} \to \mathcal{L}$, where the fibred category $\mathcal{P} = S_{\mathcal{T}}/\mathcal{L}$ is obtained by the fibrewise comma construction. The adjunction $S_{\mathcal{T}} \dashv J$ yields an alternative presentation, since $S_{\mathcal{T}}/\mathcal{L}$ is isomorphic with $\mathcal{T}/J$. The objects of $\mathcal{P}$ can thus be viewed as triples $U = \langle \Sigma, P, S\rangle$, where the synchronisation tree P satisfies the specification $S \subseteq \Sigma^*$ — i.e. $S_{\mathcal{T}}P \subseteq S$, or equivalently $P \to JS$[8]. This last arrow is unique because JS is the subobject of the terminal object $J\Sigma^*$ in $\mathcal{T}_\Sigma$.

The point is now that the functors $\Sigma_{\mathcal{P}} : \mathcal{P} \to \mathcal{S}$ and $S_{\mathcal{P}} : \mathcal{P} \to \mathcal{L}$, projecting a process U respectively to the alphabet Σ and to the specification $\langle \Sigma, S\rangle$ — are both hyperfibrations. Moreover, as a morphism in the category of hyperfibrations over $\mathcal{S}$, the latter is a fibrewise hyperfibration. We describe the structure, but leave all checking to the reader.

Let $f : \Sigma \to \Gamma$ be a function in $\mathcal{S}$, and $U = \langle \Sigma, P, S\rangle$ and $V = \langle \Gamma, Q, T\rangle$ predicates in $\mathcal{P}$. The inverse and the direct images with respect to $\Sigma_{\mathcal{P}} : \mathcal{P} \to \mathcal{S}$ are

$$f^{\#}V = \langle \Sigma, f^{\#}Q, f^{\#}T\rangle \tag{27}$$

$$f_{\Box}U = \langle \Gamma, f_{\Box}P, f_{\Box}S\rangle, \tag{28}$$

where $\Box \in \{!, \#\}$. The structure appearing on the right hand side comes from the hyperfibrations $\Sigma_{\mathcal{T}} : \mathcal{T} \to \mathcal{S}$ (18,22) and $\Sigma_{\mathcal{L}} : \mathcal{L} \to \mathcal{S}$ (easy). Ditto in

$$V \Box_\Gamma \tilde{V} = \langle \Gamma, Q \Box_\Sigma \tilde{Q}, T \Box_\Sigma \tilde{T}\rangle, \tag{29}$$

where $\Box \in \{\wedge, \Rightarrow\}$.

Now we want to calculate the inverse and the direct images with respect to $S_{\mathcal{P}} : \mathcal{P} \to \mathcal{L}$. Recall that a morphism $f : A \to B$ of specifications $A = \langle \Sigma, S\rangle$ and $B = \langle \Gamma, T\rangle$ is a function $f : \Sigma_A \to \Sigma_B$ such that for every safe string $s \in S \subseteq \Sigma^*$ holds $f^*(s) \in T \subseteq \Gamma^*$. The existential quantifiers

$$f_!U = \langle \Gamma, f_!P, T\rangle. \tag{30}$$

which are strict and canonical, are obtained directly from the definition of $\mathcal{P}$ as the comma category $S_{\mathcal{T}}/\mathcal{L}$. The tree $f_!P$ on the right hand side of (30) is the

[8]More precisely, one should write $\langle \Sigma, S\rangle$, and not just S. But the alphabet Σ is fixed here and it can be safely omitted.

direct image with respect to $\Sigma_{\mathcal{T}} : \mathcal{T} \to \mathcal{S}$, i.e. the relabelling of P along f. Its traces are surely contained in T, because the traces of P are in S and $s \in S$ implies $f^*(s) \in T$.

However, the inverse images and the right direct images (universal quantifiers) with respect to $\Sigma_{\mathcal{T}} : \mathcal{T} \to \mathcal{S}$ do not in general the satisfaction of specifications. Their results will have to be restricted to the safe strings, in order to yield the corresponding structures for $S_{\mathcal{P}} : \mathcal{P} \to \mathcal{L}$. This restriction can be obtained using the fibrewise products $\wedge$, described on (17):

$$f^{\#}V = \langle \Sigma, f^{\#}Q \wedge_{\Sigma} JA, S \rangle \tag{31}$$

$$f_{\#}U = \langle \Gamma, f_{\#}P \wedge_{\Gamma} JB, T \rangle. \tag{32}$$

The rest of the structure on the right hand side is from (18) and (22). Note that $f^{\#}Q \wedge_{\Sigma} JA$ is a subtree of $f^{\#}Q$, since JA is a subobject of the terminal object $J\langle \Sigma, \Sigma^* \rangle$ in $\mathcal{T}_{\Sigma}$. For the same reason holds $f_{\#}P \wedge_{\Gamma} JB \subseteq f_{\#}P$.

For propositional operations, on one hand we have that the product $Q \wedge_{\Gamma} \tilde{Q}$ in $\mathcal{T}_{\Gamma}$ has an arrow to JB as soon as either Q or $\tilde{Q}$ has it. Therefore, it must satisfy any specification that either of them satisfies. The fibrewise product over a specification B is thus

$$V \wedge_B \tilde{V} = \langle \Gamma, Q \wedge_{\Gamma} \tilde{Q}, T \rangle. \tag{33}$$

On the other hand, the exponent $Q \Rightarrow_{\Sigma} \tilde{Q}$ (21) may not satisfy the same specification as Q and $\tilde{Q}$, and must be restricted, in the same way as above:

$$V \Rightarrow_B V' = \left\langle \Gamma, (Q \Rightarrow_{\Gamma} \tilde{Q}) \wedge_{\Gamma} JB, T \right\rangle. \tag{34}$$

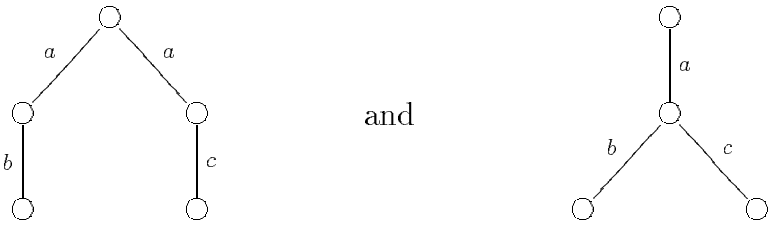
Alternatively, the restriction in formulas (32) and (34) could be enforced by presenting them respectively in form (22) and (21), but with restricted indexing: the sums should not be taken over $s \in \Gamma^i$ but over $s \in T_i$, where $T_i \subseteq \Gamma^i$ is the set of the safe strings from T of the length i.

4 Logic in Process Calculus

In this section, we present the described logical operations on trees as operations on indexed sets — but *"extended in time"*. In this form, their correspondence with the SCCS-operations becomes apparent.

Traditionally, automata as processes were studied from the point of view of the accepted languages. The finite ones were completely reduced to the calculus of regular expressions [15, ch. 2–3]. The computational behaviour and nondeterminism were thus left out of the picture. Indeed, for every automaton there is a deterministic one (such that $\langle \delta, \lambda \rangle : T \to S \times \Sigma$ is isomorphism) accepting the same language.

The issue of the computational behaviour was first addressed in Milner's Calculus of Communicating Systems (CCS) [19]. The idea is that, when we observe an automaton running, we not only get to know which words it accepts and which lead to a deadlock, but also wittness the deadlocks that may occur, due to nondeterminism, in reading otherwise acceptable words. For instance, Milner pointed out, the processes



accept the same language $\{a, ab, ac\}$, yet they behave in *observably* different ways. While the one on the right hand side always accepts an element of this language, the other one deadlocks half of the times: whenever it chooses to go left while reading ac, and when it chooses do go right while reading ab. If nondeterminism is to be taken into account, the above two processes should not be identified. As they depict, respectively, the regular expressions $ab + ac$ and $a(b+c)$, Milner concluded that the distributive law should be dropped from the algebra of processes.

Dropping the distributivity brings us in the realm of trees. Finite Σ-labelled trees can be viewed as the elements of the free algebra with no generators, with a single constant $\emptyset$, with Σ unary operations, and one commutative, associative, binary operation $+$, for which $\emptyset$ is the unit. No distributivity. The constant $\emptyset$ represents the trivial tree, just one vertex. Each operation $a \in \Sigma$ maps a tree P to the tree aP, obtained from P by adding a new root, and connecting it to the old one by an a-labelled edge. This is called *prefixing*. It *delays* the proces P until the action a is executed. The tree $a\emptyset$, with a single edge, is usually abbreviated to a.

The sum $P + P'$ corresponds to the tree obtained by identifying the roots of P and P'. Computationally, $P + P'$ represents the nondeterministic choice

between the processes P and P'. Categorically, this is the coproduct in the category $\mathcal{T}_\Sigma$. Logically, it is the disjunction $P \vee P'$ of the predicates P and P' over Σ. Finally, if P and P' are interpreted as $\in$-trees of some sets, this operation should correspond to the union $P \cup P'$ — which is just what $P + P'$ used to be in the calculus of regular expressions.

Generated by delays and the nondeterministic choices, trees are the "moral minimum" of process calculi.

4.2 Trees as Sets in Time

In an $\in$-tree, each vertex represents a set. The elements of a set are represented as its successors: an edge $P \to Q$ means $P \ni Q$. Further successors capture the elements of Q and so on. A constructible set [12], unfolded as a tree, displays its history: how it came about in the cumulative hierarchy. *The tree depicting a set is its extent in time.*

This brings us close to Abramsky's idea of processes as relations extended in time [2, sec. 2.1]. It is couched in Aczel's view of processes arising similarly to constructible sets — in a coinductive iteration of the power-set operator [4].

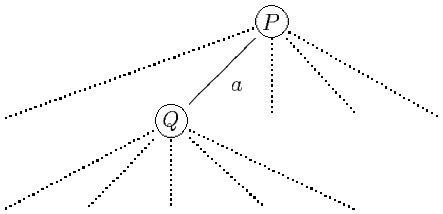
Of course, the poetry of trees in time can also be expressed categorically. If the chain $\mathbf{N}$ — the least fixed point of the delay functor — is taken as a picture of time (endless, discrete, irreversible...), then trees, presented as in subsection 3.2, are just sets extended in time. The tree morphisms appear as functions extended in time. If trees P and Q represent sets, an ordinary function $P \to Q$ just assigns to each element of P an element of Q. These elements are the vertices from P_1 and Q_1 — so we have a function $\varphi_1 : P_1 \to Q_1$. A function extended in time does not stop here: having assigned x to $\varphi_1(x)$, it goes on and assigns to each element of x an element of $\varphi_1(x)$, etc. It is a *hereditary* function. Indeed, every $x \in P_1$ represents a set as well, namely $p_1^{-1}(x) \subseteq P_2$; while $\varphi_1(x) = y \in Q_1$ represents $q_1^{-1}(y) \subseteq Q_2$. That function from x to $\varphi_1(x)$ is thus displayed in our presentation of trees as the restriction of $\varphi_2 : P_2 \to Q_2$ to $p_1^{-1}(x)$...

The exponents of trees, depicted in (21), just collect these functions in time. The product $P \wedge Q$ of trees similarly extends the product of sets through time, in the sense that it does not collect just the pairs of elements $\langle x, y \rangle$, but their actual *(hereditary) products*:

$$P \wedge Q \quad = \quad \{x \wedge y | x \in P \text{ and } y \in Q\} \tag{35}$$

This formula is a very succint form of (17). Describing all logical operations on trees according to this set-theoretical idea, we get a system similar to SCCS.

A sequent in the form $P \overset{a}{\ni} Q$ means that the tree aQ is contained in P.



A tree P, as on (15), becomes a set of sequents $p_i(x) \overset{\lambda(x)}{\ni} x$, for $x \in P_{i+1}, i \in \mathbf{N}$; and $x \in P_1$ become $P \overset{\lambda(x)}{\ni} x$.

Let us write $P = \{{}_a x, {}_a x', {}_b y, {}_c z \ldots\}$ to abbreviate a set of sequents

$$P \overset{a}{\ni} x, P \overset{a}{\ni} x', P \overset{b}{\ni} y, P \overset{c}{\ni} z \ldots$$

Writing, suggestively $\vee$ instead of $+$, we can now present the operations from subsection 4.1 in the form

$$aP = \{{}_a P\} \qquad \text{and} \tag{36}$$

$$P \vee Q = \{{}_a x | P \overset{a}{\ni} x \text{ or } Q \overset{a}{\ni} x\} \tag{37}$$

An alternative presentation is provided in the sequent calculus below, which also captures the operations from subsection 3.2.

Notation. While $P \overset{a}{=} \{x, x' \ldots\}$ denotes the set of all a-labelled elements of P, an I-tuple of a-labelled edges, possibly with repetitions, is $\left(P_i \overset{a}{\ni} P'_i\right)_{i \in I}$.

$$\frac{}{\{{}_a P\} \overset{a}{\ni} P}\,(\cdot)$$

$$\frac{P_j \overset{a}{\ni} P'}{\bigvee_{i \in I} P_i \overset{a}{\ni} P'}\,(\bigvee_I^j) \qquad\qquad \frac{\left(P_i \overset{a}{\ni} P'_i\right)_{i \in I}}{\bigwedge_{i \in I} P_i \overset{a}{\ni} \bigwedge_{i \in I} P'_i}\,(\bigwedge_I)$$

$$\frac{Q \overset{a}{=} \{Q_i\}_{i \in I} \qquad \tilde{Q} \overset{a}{=} \{\tilde{Q}_j\}_{j \in J} \qquad g : I \to J}{\left(Q \Rightarrow \tilde{Q}\right) \overset{a}{\ni} \bigwedge_{i \in I} \left(Q_i \Rightarrow \tilde{Q}_{g(i)}\right)}\,(\Rightarrow)$$

$$\frac{P \overset{a}{\ni} P'}{f_! P \overset{f(a)}{\ni} f_! P'}\,(\exists) \qquad\qquad \frac{Q \overset{f(a)}{\ni} Q'}{f^{\#} Q \overset{a}{\ni} f^{\#} Q'}\,(\#)$$

$$\frac{\left(P \overset{a}{\ni} P_a\right)_{a \in f^{-1}(b)}}{f_{\#} P \overset{b}{\ni} \bigwedge_{a \in f^{-1}(b)} f_{\#} P_a}\,(\forall)$$

Remarks. Note that these sequents do not represent logical entailment, but edges of trees. Therefore, the logical meaning of the displayed operations cannot be deciphered simply by comparing the corresponding rules with familiar Gentzen-style sequent calculi. There does not seem to be a way of avoiding the logical analysis of the category of trees, like in subsection 3.2.

On the other hand, with the exception of $(\Rightarrow)$ and $(\forall)$, the rules are familiar from Milner's Calculus of Communicating Systems [20]. $(\bigvee)$ is the nondeterministic sum (Σ), $(\exists)$ is the relabelling, while the restriction appears as the special case of $(\#)$. Finally, $(\bigwedge)$ corresponds to the *synchronous* product [*op.cit.*, sec. 9.3] — so that we are modelling a version of the *synchronous* calculus SCCS. The derived calculus of relations will thus yield the category of synchronous processes.

Of the two new rules, $(\forall)$ is readily understood by comparison with $(\exists)$. Given a process P satisfying a specification A, and a morphism $f : A \to B$, rule $(\exists)$ says that a b-action in $f_! P$ chooses nondeterministically for *some* $a \in f^{-1}(b)$ a process $f_! P'$, such that $P \overset{a}{\ni} P'$. On the other hand, rule $(\forall)$ says that a b-action in $f_{\#} P$ invokes for *every* $a \in f^{-1}(b)$ a process $f_{\#} P_a$, $P \overset{a}{\ni} P_a$, and executes all of them in parallel. In a sense, $f_{\#} P$ is a *parallel* relabelling, since it pursues all actions relabelled to b at the same time, whereas the ordinary relabelling $f_! P$ is *nondeterministic*, since it pursues only one of them, arbitrarily chosen.

Rule $(\Rightarrow)$ seems a bit more difficult to explain computationally. A computation s in $Q \Rightarrow \tilde{Q}$ corresponds to a choice, for each way s that can be traced in Q, of a way to simulate it in $\tilde{Q}$.

5 SProc

Let us put the threads together: apply the construction described in 2.5 to hyperfibration from 3.3, and use the calculus from 4.3 to interpret the result.

The obtained bicategory of relations

$$\mathcal{R} \;=\; \mathsf{Rel}\,(S_{\mathcal{P}} : \mathcal{P} \to \mathcal{L})\,,$$

will turn out to be the closest "logical relative" of the category of synchronous processes SProc.

5.1 Synchronous Processes as Relations

First of all, the objects of the two categories clearly coincide: are just specifications. So we must compare the hom-categories

$$\mathcal{R}(A,B) \;=\; \mathcal{P}_{A\times B} \tag{38}$$

$$\mathsf{SProc}(A,B) \;=\; \{\langle A,P,B\rangle | P \models A \otimes B\} \tag{39}$$

the former based on (6), the latter summarising [2, 2.2]. But an object $P \in \mathcal{P}_{A\times B}$ is a tree P satisfying the specification $A\times B$, i.e. such that $S_T P \subseteq A\times B$ — which is exactly what Abramsky writes $P \models A\otimes B$, since the tensor product $\otimes$ in SProc is just the cartesian product $\times$ of $\mathcal{L}$.

Furthermore, the formulas for the identities and the composition in SProc exactly correspond to the identity relations and the standard relational composition of $\mathcal{R}$, only presented in the CCS-style. The identity relation $(\overset{A}{\equiv})$, defined in (9), can for $\mathcal{R}$ be derived

$$\dfrac{\dfrac{}{\top_A \overset{a}{\ni} \top_{A/a}}\,(\cdot)}{d_!(\top_A) \overset{\langle a,a\rangle}{\ni} d_!(\top_{A/a})}\,(\exists)$$

The terminal object $\top_A$ in $\mathcal{P}_A$ is the tree[9] JA, described in (25). The specification A/a has the same alphabet as A, but its safe strings are $\{s \in \Sigma_A^* | as \in S_A\}$. The first step in the above derivation is thus just another form of (25). Substituting its conclusion in (9), we get

$$(\overset{A}{\equiv}) \;=\; \bigvee_{a\in\Sigma_A} \left\{ a\left(\overset{A/a}{\equiv}\right)\right\}$$

which is clearly the original definition of the identity in SProc, just expressed in terms of (36–37).

[9] Strictly speaking, it is the triple $\langle \Sigma_A, JA, S_A\rangle$; but we abuse notation, and denote it all by JA.

As for the relational composition, expanding formula (8) in a CCS-style derivation yields

$$\dfrac{\dfrac{P \overset{\langle a,b\rangle}{\ni} P'}{p^{\#}P \overset{\langle a,b,c\rangle}{\ni} p^{\#}P'}\,(\#) \qquad \dfrac{Q \overset{\langle b,c\rangle}{\ni} Q'}{q^{\#}Q \overset{\langle a,b,c\rangle}{\ni} q^{\#}Q'}\,(\#)}{\dfrac{p^{\#}P \wedge q^{\#}Q \overset{\langle a,b,c\rangle}{\ni} p^{\#}P' \wedge q^{\#}Q'}{r_!\left(p^{\#}P \wedge q^{\#}Q\right) \overset{\langle a,c\rangle}{\ni} r_!\left(p^{\#}P' \wedge q^{\#}Q'\right)}\,(\exists)}\,(\wedge_2)$$

Substituting (8), this derivation can be summarized as the composite rule

$$\dfrac{P \overset{\langle a,b\rangle}{\ni} P' \qquad Q \overset{\langle b,c\rangle}{\ni} Q'}{(P\,;Q) \overset{\langle a,c\rangle}{\ni} (P'\,;Q')} \tag{40}$$

— which is just the original formula for composition in SProc.

Remark. Cockett and Spooner [8] have proposed a presentation of SProc as a formal quotient of the bicategory of spans in $\mathcal{L}$. Of course, this bicategory can be viewed as the bicategory of relations, induced by the basic regular fibration $\mathsf{Cod} : \mathcal{L}/\mathcal{L} \longrightarrow \mathcal{L}$. The derivation of SProc can be explained in terms of a cartesian functor $\mathcal{L}/\mathcal{L} \longrightarrow \mathcal{P}$, which maps each specification morphism $f : A \to B$ to the direct image $f_!\top A$. This functor is tacitly used in the explanation how the SProc-morphisms can be obtained from spans in $\mathcal{L}$. It is surjective on objects and faithful, but apparently not much more. Nevertheless, the surjectiveness suffices for reconstructing all relations of $\mathcal{R}$, and hence morphisms of SProc.

5.2 The difference

However, in spite of the clear structural and conceptual parallelism, SProc does not actually coincide with the bicategory of relations $\mathcal{R}$: it has fewer morphisms. Namely, the SProc-morphisms must be the canonical representatives of processes [2, 2.2.], and they do not exhaust all the labelled trees. This turns out to be a fairly essential point, with considerable conceptual weight, and significant structural repercussions. Understanding the effect of this restriction seems to be the most difficult part of the purported analysis, since it distorts the described logical operations. However, similar transformations under a change of context are a frequent phenomenon, rather than a pathology: cf. the way in which the standard tensor product in a category of relations is obtained from what used to be the cartesian product in the corresponding category of sets. The tensor

product of vector spaces boils down to the cartesian product of the chosen bases in a similar fashion.

Irredundant trees. As mentioned in the introduction, if trees (or automata) as processes are observed only *via* computations, some of them turn out to be indistinguishable, i.e. represent the same process, in spite of their geometric differences. Therefore, processes should not be presented as individual trees (or automata), but as classes of computationally equivalent, *bisimilar* representatives. But rather than working with such bisimilarity classes, one usually chooses a canonical representative for each of them, *irredundant* in the sense that it only carries computationally relevant data. Irredundant representatives of processes can be obtained as elements of Aczel's terminal coalgebra [4, ch. 8] or as its small subcoalgebras [28]. The morphisms of SProc are defined using the former class, the irredundant trees. An intrinsic characterisation of such trees can be worked out, with a bit of effort, along the lines of the last section of [5]. If it is well-founded, an irredundant tree can be recognized by the fact that it has a trivial group of automorphisms. This means that, viewed as a $\ni$-tree, it satisfies the law

$$\{x, x, y \ldots\} \quad = \quad \{x, y \ldots\} \tag{41}$$

— i.e. represents a set. Indeed, well-founded irredundant trees were considered as a model for set theory long time ago [29, 4]. Irredundant trees in general can be obtained as colimits of towers of well-founded ones. In other words, *a tree is irredundant if and only if each of its well-founded subtrees can be embedded in a well-founded tree with a trivial automorphism group.*

From a different direction, irredundant trees can be obtained by unfolding irredundant automata [28]. Either way, the same (up to isomorphism) category of processes $\widetilde{\mathcal{P}} \hookrightarrow \mathcal{P}$ is obtained. Its objects are the canonical representatives of processes. The graph morphisms between them, preserving the labels and the initial states, capture all the *sober*, i.e. bisimilarity preserving simulations.

Restricting operations. It is easy to see that the class $\widetilde{\mathcal{P}}$ is not closed under any of the described operations on $\mathcal{P}$, except prefixing. However, every tree represents some process, and has an irredundant collapse. In other words, there is a mapping $|\mathcal{P}| \longrightarrow |\widetilde{\mathcal{P}}|$, left inverse to the object part of the inclusion $\widetilde{\mathcal{P}} \hookrightarrow \mathcal{P}$. Since the morphisms of SProc are objects of $\widetilde{\mathcal{P}}$, while the morphisms of $\mathcal{R}$ are objects of $\mathcal{P}$, the morphism part of the structure of $\mathcal{R}$ is transferred on SProc using this inclusion and its retraction. In this way, the categorical structure of SProc is completely induced from $\mathcal{R}$. This holds not only for the identities and the composition, but also for the tensor, biproducts and the cofree comonoid construction. All this is inherited from the relational calculus.

However, at the level of 2-cells, the parallelism does not pertain. The reason is that the irredundant collapse construction $|\mathcal{P}| \longrightarrow |\widetilde{\mathcal{P}}|$ cannot be extended into a functor on all tree morphisms. It can be defined on the sober ones, and we get a left adjoint of the inclusion $\widetilde{\mathcal{P}} \hookrightarrow \mathcal{P}_{\mathrm{sober}}$. However, none of the relevant operations on $\mathcal{P}$, taken as functors, preserve the sobriety of morphisms — none of them restricts to $\mathcal{P}_{\mathrm{sober}}$. The arrow part of the logical structure of $\mathcal{P}$ does not restrict to $\mathcal{P}_{\mathrm{sober}}$, and cannot be transferred to $\widetilde{\mathcal{P}}$. In a rather nonaccidental way, SProc does *not* inherit the 2-dimensional structure of $\mathcal{R}$, which is induced by this arrow part.

5.3 Dynamics is preorder

According to its original definition, SProc does not distinguish particular simulations between processes, but is enriched only by the simulation *preorder* $P \prec Q$, which records whether there exists some simulation $P \to P'$ between processes $P, P' : A \nrightarrow B$. In other words, SProc should not be derived from $\mathcal{R}$, but from its preorder collapse $\mathcal{R}_{\prec}$. Indeed, there is a functor

$$\widetilde{(-)} \;:\; \mathcal{R}_{\prec} \longrightarrow \mathsf{SProc}, \tag{42}$$

which leaves the objects unchanged, and maps each tree $P : A \nrightarrow B$ to its irredundant collapse $\widetilde{P} : A \nrightarrow B$. Since $P \prec P'$ implies $\widetilde{P} \prec \widetilde{P'}$, this is an enriched functor. Since $\mathsf{SProc}(A, B)$ appears as a reflective subpreorder of $\mathcal{R}_{\prec}(A, B)$, it forms a soft equivalence with the inclusion $\mathsf{SProc} \hookrightarrow \mathcal{R}_{\prec}$.

But note that $\mathcal{R}_{\prec}$ is actually the bicategory of relations corresponding to the regular fibred preorder $\mathcal{P}_{\prec} \longrightarrow \mathcal{L}$, obtained by making the functor $\mathcal{P} \longrightarrow \mathcal{L}$ faithful, which collapses its fibres to preorders. Finally, SProc itself *can be obtained as the calculus of relations induced by* $\widetilde{\mathcal{P}}_{\prec} \longrightarrow \mathcal{L}$, a regular preorder too!

One could thus assert that SProc, after all, does realize the slogan of processes-as-relations in a strictly formal sense. The signalled disharmony between $\mathcal{R}$ and SProc simply disappears upon the passage from simulations to the similarity relation. — So *categories* of processes should be abandoned, always reduced to *preorders*? On one hand, the dynamic laws [20, 3.2] of CCS, which make the nondeterministic sum $P \vee Q$ into the join of processes, do enforce this. The resulting notion of bisimilarity creates the described gap between $\mathcal{R}$ and SProc. On the other hand, we have seen that the basic process operations stem from logical operations, universal up to isomorphism, and not just modulo similarity both ways. The similarity preorder, imposed by the dynamic laws, is too loose to capture their contents. A more precise setting, in which the nature of process

operations would be reflected on process morphisms, seems to require a refined notion of bisimilarity.

Acknowledgements

Thanks to Samson Abramsky for his patience and interest; and to Lindsay Errington and Greg Meredith for answering some of my naive questions.

References

[1] S. Abramsky, Interaction Categories and Communicating Sequential Processes, *in:* A. W. Roscoe (ed.), *A Classical Mind: Essays in Honour of C.A.R. Hoare*, (Prentice Hall 1994)

[2] S. Abramsky et al., Interaction categories and the foundations of typed concurrent programming, *in:* Proceedings of the Marktoberdorf Summer School 1994 (Springer 1995)

[3] S. Abramsky, S. Gay and R. Nagarajan, Constructing and verifying typed processes, abstract of a talk given at the CONFER Workshop (Edinburgh 1993)

[4] P. Aczel, *Non-Well-Founded Sets*, Lecture Notes 14 (CSLI 1988)

[5] M. Barr, Terminal coalgebras in well-founded set theory, *Theoret. Comput. Sci.* 114(1993) 299–315

[6] J. Bénabou, Fibrations petites et locallement petites, *C. R. Acad. Sci. Paris Sér. A, Math.* 281(1975) 897–900

[7] A. Carboni and R.F.C. Walters, Cartesian bicategories I, *J. Pure Appl. Algebra* 49(1987) 11-32

[8] J.R.B. Cockett and D. Spooner, SProc categorically, *in:* Proceedings of CONCUR 94, Lecture Notes in Computer Science (Springer 1994)

[9] H.B. Curry and R. Feys (with W. Craig), *Combinatory Logic I*, Stud. Log. Found. Math. (North-Holland, 1958)

[10] T. Fox, Coalgebras and cartesian categories, *Com. in Algebra* 4(1976) 665–667

[11] P.J. Freyd and A. Scedrov, *Categories, Allegories*, Mathematical Library 39 (North-Holland 1990)

[12] K. Gödel, *The Consistency of the Continuum Hypothesis*, Ann. Math. Studies 3 (Princeton Univ. Press 1940)

[13] J.W. Gray, Fibred and cofibred categories, *in:* S. Eilenberg (ed.), *Proceedings of the Conference on Categorical Algebra, La Yolla 1965*, (Springer, 1966) 21–84

[14] A. Grothendieck, Catégories fibrées et descente, Exposé VI, *Revêtements Etales et Groupe Fondamental (SGA1)*, Lecture Notes in Mathematics 224 (Springer, 1971) 145–194

[15] J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages and Computation* (Addison-Wesley 1979)

[16] A. Joyal, M. Nielsen and G. Winskel, Bisimulation and open maps, *Proceedings of the Eight Symposium on Logic in Computer Science* (IEEE 1993) 418–427

[17] F.W. Lawvere, Adjointness in foundations, *Dialectica* 23(1969), 281–296

[18] F.W. Lawvere, Equality in hyperdoctrines and comprehension schema as an adjoint functor, *Proc. Sympos. Pure Math* XVII(1970), 1–14

[19] R. Milner, *A Calculus of Communicating Systems*, Lecture Notes in Computer Science 92 (Springer 1980)

[20] R. Milner, *Communication and Concurrency*, Internat. Ser. in Comp. Sci. (Prentice Hall, 1989)

[21] P. Panangaden et al., A Categorical view of concurrent constraint programming, *in:* J.W. de Bakkee et al. (eds.), *Semantics: Foundations and Applications*, Lecture Notes in Computer Science 666 (Springer 1993) 457–476

[22] D. Park, *Concurrency and Automata on Infinite Sequences*, Lecture Notes in Computer Science 104 (Springer 1980)

[23] R. Paré and D. Schumacher, Abstract families and the adjoint functor theorems, in: *Indexed categories and their applications*, Lecture Notes in Mathematics661 (Springer 1978) 1–125

[24] D. Pavlović, *Predicates and Fibrations*, thesis (Rijksuniversiteit Utrecht 1990)

[25] D. Pavlović, Categorical interpolation: descent and the Beck-Chevalley condition without direct images, *in:* A. Carboni et al. (eds.), *Category Theory*, Lecture Notes in Mathematics 1488 (Springer, 1991) 306–326

[26] D. Pavlović, Maps I: relative to a factorisation system, *Report 93-06 from McGill University* (March 1993), 35 pp.; to appear in *J. Pure Appl. Algebra*

[27] D. Pavlović, Maps II: Chasing diagrams in categorical proof theory, *Report 93-08 from McGill University* (May 1993; revised), 46 pp.; *submitted*

[28] D. Pavlović, Convenient categories of processes and simulations I: modulo strong bisimilarity, *in:* D. Pitt et al. (eds.), *Category Theory in Computer Science*, Lecture Notes in Computer Science (Springer 1995)

[29] D. Scott, A different kind of model for set theory, *unpublished lecture* given at the Congress of Logic, Methodology and Philosophy of Science (Stanford 1960)

[30] G. Winskel and M. Nielsen, Models for concurrency, em in: S. Abramsky et al. (eds.), *Handbook of Logic in Computer Science*, vol. IV (Clarendon Press) *to appear*