Towards a Mathematical Operational Semantics

Daniele Turi* <dt@dcs.ed.ac.uk>

Gordon Plotkin† <gdp@dcs.ed.ac.uk>

Department of Computer Science
Laboratory for Foundations of Computer Science
University of Edinburgh, The King's Buildings
Edinburgh EH9 3JZ, Scotland

To appear in *Proc. LICS'97*.

Abstract

We present a categorical theory of 'well-behaved' operational semantics which aims at complementing the established theory of domains and denotational semantics to form a coherent whole. It is shown that, if the operational rules of a programming language can be modelled as a natural transformation of a suitable general form, depending on functorial notions of syntax and behaviour, then one gets both an operational model and a canonical, internally fully abstract denotational model for free; moreover, both models satisfy the operational rules. The theory is based on distributive laws and bialgebras; it specialises to the known classes of well-behaved rules for structural operational semantics, such as GSOS.

Introduction

Operational semantics, a fundamental tool in language design and verification, provides a formal description of the behaviour of programs. It is often defined in terms of atomic, elementary transitions, describing local behaviour. Mathematically, these transitions can be modelled as the elements of a relation, the intended operational model of the language. A convenient way of specifying such a transition relation is by induction on the structure of the programs, starting from suitable operational rules for the basic constructs of the language [21].

Traditionally, operational semantics is contrasted with the mathematical interpretation of programs called denotational semantics, where programs are mapped into a suitable semantic domain endowed with an operation for each construct of the language. Both operational and denotational semantics are necessary for a complete description of a programming language: the former for specifying the execution of the programs and the latter for reasoning about them in terms of abstract, mathematical entities. It is therefore fundamental that a denotational semantics be *adequate*, ie that it determines the operational behaviour of programs [24].

For languages without variable binding, but possibly multi-sorted, a denotational model can be seen as a Σ-algebra, where Σ is the signature of the language corresponding to the basic constructs. The programs themselves form the initial such Σ-algebra and the corresponding unique homomorphism from the programs to the denotational model is called *initial algebra semantics* [12].

The semantic domain, ie the carrier of the denotational model, can often be regarded as the final solution of a domain equation $X \cong B(X)$, for a suitable 'behaviour' functor B. In other words, the semantic domain is the final B-coalgebra. The transition relations may also be seen as B-coalgebras and, therefore, so can the intended operational model of a language. The corresponding unique coalgebra homomorphism, given by finality, from the intended operational model to the semantic domain is called *final coalgebra semantics* [2, 23]; under suitable assumptions on B, it is fully abstract with respect to behavioural equivalence. When initial algebra and final coalgebra semantics coincide,

*Research supported by EuroFOCS.
†Research supported by an EPSRC Senior Fellowship.

one has an adequate denotational semantics [23].

Adequacy proofs can be quite demanding, hence general criteria ensuring adequacy are of interest. For process algebras, as used for specifying nondeterministic and concurrent programs [17, 5], there exist syntactic restrictions on the format of the operational rules which ensure that bisimulation [17] is a congruence. Among the rules in these formats, GSOS rules [8] are the best known and (negative) tree rules [11] are the most general. In [22], the 'processes as terms' method, based on such a congruence result, is presented which allows for the systematic derivation of adequate denotational models from 'tyft rules' [13], a class of rules equivalent to tree rules.

We present here a categorical reformulation and generalisation of the above adequacy meta-results. First, we show that certain sets $\mathcal{R}$ of GSOS rules can be modelled as natural transformations $[\![\mathcal{R}]\!]$ depending on the functorial notions of signature Σ and behaviour B. Next, it is shown that the mapping $\mathcal{R} \mapsto [\![\mathcal{R}]\!]$ is an essentially 1-1 correspondence. The naturality of $[\![\mathcal{R}]\!]$ accounts for the syntactic restrictions on the occurrences of meta-variables in GSOS rules and provides a categorical explanation of their good behaviour.

The first advantage of the above approach is that the GSOS rules can be modelled not only in Set, but also in every category with enough structure such as the category of cpos and continuous functions used in denotational semantics. This is a step towards bridging the gap between operational and domain theory.

A second advantage is that the mathematical modelling of the rules is a useful semantic tool in the investigation of syntactic formats. For instance, in Set the 'dual' of the type of natural transformation corresponding to GSOS also corresponds to an interesting format, namely the safe tree rules: these form a natural subclass of (negative) tree rules which always possess a satisfying transition relation. Interestingly, the failure to fit the class of (simple negative) tree rules in the present approach brought to light a slight inaccuracy in the literature and, eventually, led to the discovery of the safe tree rules.

A third advantage is that by varying Σ and B a wide variety of notions of program constructs and behaviour can be accommodated. (See also [30].) Further, one can study abstract notions of operational rules ρ, such as 'abstract GSOS' and 'abstract tree rules', applicable to languages other than process algebras and whose properties can be studied in general.

In this theory we assume that Σ freely generates a monad T which is thought of as corresponding to the syntax of the language. The first result is that such abstract operational rules ρ induce an *operational monad* T_ρ lifting the monad T to the B-coalgebras, ie to the operational models, in the sense that its action on the carriers is the same as the monad T.

If ρ is of abstract tree rules form rather than abstract GSOS, then, by duality, one first coinductively derives a *denotational comonad* D_ρ. The assumption here is that the functor B cofreely generates a comonad D which should correspond to the *global* behaviours of the language. The comonad D_ρ is a lifting of this comonad D to the Σ-algebras, ie to the denotational models. However, one can still speak of the operational monad defined by some abstract tree rules because a general theorem shows that liftings of D to the Σ-algebras and liftings of T to the B-coalgebras are in 1-1 correspondence.

In fact, these liftings are also in 1-1 correspondence with the *distributive laws* λ of the monad T over the comonad D., which generalise both abstract GSOS and abstract tree rules. One is led now to consider the bialgebras of such distributive laws. When λ corresponds to some abstract operational rules ρ, the λ-bialgebras can be seen as combinations of operational and denotational models which satisfy the rules. Henceforth they are called *ρ-models*; they specialise to the GSOS models of [25] and to models of tree rules (with an appropriate definition).

The primary fact about ρ-models is that, from results in [15], it easily follows that the forgetful functors to each of the categories of denotational and operational models have adjoints. One adjunction implies that there exists an initial ρ-model – the intended operational model $T_\rho(0)$ for the initial algebra of programs. By the definition of morphism of ρ-models, this also implies that every ρ-model is *adequate* with respect to the intended operational model in the sense that the behaviour of the programs can be determined from any ρ-model up to a generalised, coalgebraic notion [4, 16] of bisimulation.

The other adjunction implies that there exists a final ρ-model – the canonical denotational model $D_\rho(1)$ over the final coalgebra of abstract, global behaviours. It is necessarily adequate; further, it is internally fully abstract with respect to coalgebraic bisimulation. The derivation of this final model specialises to the above mentioned processes-as-terms method.

The unique homomorphism from the initial to the final ρ-model is both the initial algebra and final coalgebra semantics for the abstract rules ρ. It is called here *universal semantics*; it is the most abstract compositional interpretation of programs preserving behavioural distinctions. Moreover, if the behaviour functor B satisfies a certain mild condition, every ρ-model has a greatest (generalised) bisimulation which, moreover, is a (generalised) congruence. This specialises to the fact that bisimulation is a congruence for GSOS and for tree rules.

The generalised, coalgebraic notion of bisimulation considered here is to be understood as the behavioural equivalence corresponding to the functor B under consideration. It might take forms quite different from ordinary (strong) bisimulation. For instance, for the behaviour functor in [14] it specialises to the much coarser (complete) trace equivalence. As a corollary, one has an abstract format of rules ensuring that trace equivalence is a congruence [30].

To some extent, one can also deal with weak bisimulation in this setting. As shown, eg in [13], weak bisimulation for a given set of rules can be reduced to strong bisimulation by adding three special rules for the τ-action. (See also [3].) These rules are in the tyft/tyxt format, but they can be compiled into safe tree rules, hence the present theory can be applied. This way of dealing with weak bisimulation is quite indirect, but that just reflects the absence of an established denotational model for it. A more direct treatment of weak bisimulation might arise following [9].

1 The Motivating Example: GSOS

Consider the language with signature Σ consisting of a constant symbol 'nil', a set of unary action prefixing operators indexed by a *finite* set A of actions ranged over by a, and a binary parallel composition operator '$\|$'. This signature freely generates, for every set X of variables x, the set TX of terms t given by the abstract grammar

$$t ::= x \mid \mathsf{nil} \mid a.t \mid t \parallel t$$

This set TX is the carrier of the *free* Σ*-algebra* over X, where, in general, a Σ*-algebra* is given by a (carrier) set Y and a function h mapping each operator σ of arity n in the signature to a function of type $Y^n \to Y$. More concisely, the function h can be written as

$$h : \coprod_{\sigma \in \Sigma} Y^{arity(\sigma)} \to Y \tag{1}$$

using the disjoint union functor '$\coprod$' (coproduct in Set) to glue the interpretation of the various operators together.

Next, let the operational rules $\mathcal{R}$ inductively defining the (labelled) transitions performable by the programs of the above language be

$$a.x \xrightarrow{a} x \qquad \frac{x \xrightarrow{a} x'}{x \parallel y \xrightarrow{a} x' \parallel y} \qquad \frac{y \xrightarrow{a} y'}{x \parallel y \xrightarrow{a} x \parallel y'}$$

For instance, the simple program $(a.\mathsf{nil}) \parallel (a.\mathsf{nil})$ can either perform the action a becoming $\mathsf{nil} \parallel (a'.\mathsf{nil})$ or perform a becoming $(a.\mathsf{nil}) \parallel \mathsf{nil}$. The (local) behaviour of this program can be modelled as the function from A to finite subsets of terms mapping a to the set $\{(a.\mathsf{nil}) \parallel \mathsf{nil}, (a.\mathsf{nil}) \parallel \mathsf{nil}\}$ and all other actions to the empty set. In general, the type B of the behaviour of the above language is

$$BX = (\mathcal{P}_{\mathrm{fi}} X)^A \tag{2}$$

the (covariant) functor mapping a set X to the set of functions from A to finite subsets of X.

Let x and y range over X, β range over $(\mathcal{P}_{\mathrm{fi}} X)^A$, and let us write $a \rightsquigarrow \{x_1, \ldots, x_n\}$ for the function from A to $\mathcal{P}_{\mathrm{fi}} X$ mapping a to $\{x_1, \ldots, x_n\}$ and all other elements of A to the empty set. Then, for each operator σ of the signature, the corresponding rules can be modelled as a function

$$[\![\sigma]\!] : (X \times (\mathcal{P}_{\mathrm{fi}} X)^A)^{arity(\sigma)} \to (\mathcal{P}_{\mathrm{fi}} TX)^A$$

as follows.

$$\begin{array}{lcl} [\![\mathsf{nil}]\!] & = & a \mapsto \emptyset \\ [\![a.\,]\!](x, \beta) & = & a \rightsquigarrow \{x\} \\ (x, \beta)[\![\ \|\ ]\!](y, \beta') & = & a \mapsto \begin{array}{c} \{x' \parallel y \mid x' \in \beta(a)\} \\ \cup \\ \{x \parallel y' \mid y' \in \beta'(a)\} \end{array} \end{array}$$

Using the universal property of coproducts, these functions can then be glued into a single function, say

$$[\![\mathcal{R}]\!]_X : 1 \amalg (\textstyle\coprod_A (X \times BX)) \amalg (X \times BX)^2 \to BTX$$

Note one has a function $[\![\mathcal{R}]\!]_X$ for each set X of variables. In fact, one should think of the variables in the rules as being 'meta-variables'. Most importantly, the above definition of $[\![\mathcal{R}]\!]_X$ is *natural* in X: for every renaming of the variables (possibly involving equating some of the variables), first renaming and then applying the rules is the same as first applying the rules and then renaming. As shown in §5 and §7 the naturality of $[\![\mathcal{R}]\!]$ explains the good behaviour of $\mathcal{R}$.

More generally, let A_i and B_i range over subsets of A and let $\mathcal{R}$ be a set of rules of the form

$$\frac{\{x_i \xrightarrow{a} y_{ij}^a\}_{1 \le j \le m_i^a}^{1 \le i \le n,\, a \in A_i} \qquad \{x_i \stackrel{b}{\not\longrightarrow}\}_{b \in B_i}^{1 \le i \le n}}{\sigma(x_1, \ldots, x_n) \xrightarrow{c} t} \tag{3}$$

which is *image finite* in the sense that there are finitely many rules for each operator σ in Σ and action c in A. For every set X, one can associate to $\mathcal{R}$ a function

$$[\![\mathcal{R}]\!]_X : \coprod_{\sigma \in \Sigma} (X \times (\mathcal{P}_{\mathrm{fi}} X)^A)^{arity(\sigma)} \to (\mathcal{P}_{\mathrm{fi}} TX)^A \tag{4}$$

as follows. For all t in TX, c in A, x_i in X, and β_i in $(\mathcal{P}_{\mathrm{fi}}X)^A$, put

$$t \in [\![\mathcal{R}]\!]_X(\sigma((x_1,\beta_1),\ldots,(x_n,\beta_n)))(c)$$

if and only if the following condition holds.

Condition 1.1 There exists a (possibly renamed) rule (3) in $\mathcal{R}$ such that $\{y^a_{i1},\ldots,y^a_{im^a_i}\}$ is a subset of $\beta_i(a)$, for a in A_i, and $\beta_i(b)$ is empty, for b in B_i. □

Note the function $[\![\mathcal{R}]\!]_X$ does not need to be natural in the set X.

Definition 1.1 (GSOS [8]) A *GSOS rule* is a rule of type (3) such that the x_i and y^a_{ij} are all distinct and, moreover, these are the only variables which can occur in the term t. □

Two sets of rules are called *equivalent* if they prove the same rules in the sense of [11, Def. 2.5].

Theorem 1.1 (GSOS is natural) There is a correspondence between natural transformations of type

$$\coprod_{\sigma\in\Sigma}(X\times(\mathcal{P}_{\mathrm{fi}}X)^A)^{arity(\sigma)} \to (\mathcal{P}_{\mathrm{fi}}TX)^A \tag{5}$$

and image finite sets of GSOS rules for a signature Σ (over a fixed denumerably infinite set of variables V). Moreover, this correspondence is 1-1 up to equivalence of sets of rules.

Proof. We just describe the correspondence. One direction is given by the above mapping $\mathcal{R} \mapsto [\![\mathcal{R}]\!]$. As for naturality, let us introduce some useful abbreviation first: for every function $f : X \to X'$, write f^* for the function $(\mathcal{P}_{\mathrm{fi}}f)^A$, Γ for the set $[\![\mathcal{R}]\!]_X(\sigma((x_1,\beta_1),\ldots,(x_n,\beta_n)))(c)$, and Γ' for the set $[\![\mathcal{R}]\!]_{X'}(\sigma((fx_1,f^*\beta_1),\ldots,(fx_n,f^*\beta_n)))(c)$. Then the claim is that

1. $\forall t\in\Gamma,\ \exists t'\in\Gamma',\ t'=(Tf)(t)$

and, conversely,

2. $\forall t'\in\Gamma',\ \exists t\in\Gamma,\ t'=(Tf)(t)$

Consider the first clause. If t is in Γ then Condition 1.1 holds. Clearly, $\beta_i(b)=\emptyset$ if and only if $(f^*\beta_i)(b)=\emptyset$, and $\{y^a_{i1},\ldots,y^a_{im^a_i}\}\subseteq\beta_i(a)$ implies $\{fy^a_{i1},\ldots,fy^a_{im^a_i}\}\subseteq(f^*\beta_i)(a)$, therefore $(Tf)(t)$ is in Γ', because the x_i and $y^a_{im^a_i}$ are the only variables occurring in the rule. For the second clause, one also uses the fact that the x_i and the $y^a_{im^a_i}$ in a GSOS rule are all distinct, hence the value of Γ' does not depend on any of the possible identifications made by the renaming function f.

In the converse direction, given a natural transformation ρ of type (5), one can define a set of rules as follows. Let V be $x_1,x_2,\ldots$ and let $x\xrightarrow{a}\{y_1,\ldots,y_k\}$ be an abbreviation for $x\xrightarrow{a}y_1,\ldots,x\xrightarrow{a}y_k$. Choose $\beta_i(a)=\{y^a_{ij}\mid j=1,\ldots,m^a_i\}$ so that the x_i and y^a_{ij} are all distinct. Then write a rule

$$\frac{\{x_i\xrightarrow{a}\beta_i(a)\}^{1\le i\le n}_{a\in A_i\subseteq A}\qquad\{x_i\stackrel{b}{\not\longrightarrow}\}^{1\le i\le n}_{b\in A\setminus A_i}}{\sigma(x_1,\ldots,x_n)\xrightarrow{c}t} \tag{6}$$

whenever $t\in\rho_V(\sigma((x_1,\beta_1),\ldots,(x_n,\beta_n)))(c)$ and $A_i=\{a\in A\mid\beta_i(a)\neq\emptyset\}$. Naturality ensures this is a GSOS rule. It can be further shown that naturality and the finiteness of A ensure that the resulting set of rules is equivalent to an image finite set. □

We do not understand this situation for infinite A, although the above definition of $[\![\mathcal{R}]\!]$ still works.

2 GSOS is Categorical

In this section, let $\mathcal{C}$ be a distributive category with infinite coproducts and a commutative free semi-lattice monad $\mathcal{P}_{\mathrm{f}}$. The claim is that GSOS rules can be modelled in every such category.

Note, first, that to every signature Σ one can associate an endofunctor on Set with the same name:

$$\Sigma X = \coprod_{\sigma\in\Sigma}X^{arity(\sigma)}$$

Clearly, this definition also makes sense in $\mathcal{C}$.

Next, rewrite $BX=(\mathcal{P}_{\mathrm{fi}}X)^A$ as $BX=(1+\mathcal{P}_{\mathrm{f}}X)^A$, which, again, makes sense in $\mathcal{C}$: the power Y^A is the product $\prod_A Y$, $\mathcal{P}_{\mathrm{f}}$ is the free semi-lattice monad which in Set is the relevant part of the endofunctor $\mathcal{P}_{\mathrm{fi}}$ obtained by removing the empty set, and '+' is just another notation for the binary coproduct.

It remains to generalise T. For this, let Σ be an arbitrary endofunctor on $\mathcal{C}$ and let Σ-Alg be the corresponding *category of Σ-algebras*: objects are pairs $\langle X,h\rangle$, where the 'carrier' X is an object and the 'structure' $h:\Sigma X\to X$ is a morphism of $\mathcal{C}$; the 'homomorphisms' $f:\langle X,h\rangle\to\langle X',h'\rangle$ are the morphisms $f:X\to X'$ between the carriers such that $f\circ h=h'\circ(\Sigma f)$. If the forgetful functor

$$U^\Sigma:\Sigma\text{-}\mathsf{Alg}\to\mathcal{C}\qquad\langle X,h\rangle\mapsto X$$

mapping Σ-algebras to their carriers, has a left adjoint F^Σ, then the corresponding monad

$$T=U^\Sigma F^\Sigma \tag{7}$$

is the *monad freely generated by* Σ.

For finitary endofunctors Σ as the one above, it suffices that $\mathcal{C}$ has ω-colimits for the adjunction $F^\Sigma \dashv U^\Sigma$ to hold and T to be defined. Thus one can take T to be the monad freely generated by the endofunctor $\Sigma X = \coprod_{\sigma \in \Sigma} X^{arity(\sigma)}$. In Set, its value TX at a set X is the set of terms corresponding to the operators σ of the signature and with variables x in X; the unit $\eta_X : X \to TX$ is the insertion-of-variables function which maps a variable x in X to the same variable but seen as a term; and the multiplication $\mu_X : T^2X \to TX$ is the operation which allows one to plug terms into contexts.

Theorem 2.1 For any image finite set $\mathcal{R}$ of GSOS rules, a natural transformation

$$[\![\mathcal{R}]\!] : \coprod_{\sigma \in \Sigma} (X \times (1 + \mathcal{P}_f X)^A)^{arity(\sigma)} \to (1 + \mathcal{P}_f TX)^A$$

can be defined in the internal language of distributive categories with infinite coproducts and a commutative free semi-lattice monad $\mathcal{P}_f$. In the case of Set it specialises to the transformation (4).

Proof. The transformation $[\![\mathcal{R}]\!]$ in (4) can be defined categorically using: projections and injections, pairing and copairing, and the associativity, symmetry, unit, and distributive laws for products and coproducts; the unique map to the final object 1; the unit and the free structure of the monad T; the join of free semi-lattices; the unit and the strength of the commutative monad $\mathcal{P}_f$. (The use of the strength depends on the assumption that, because $\mathcal{R}$ is image finite and A is finite, for each operator the set of rules is finite.) $\square$

The characterisation of GSOS given by the above theorem allows one, for instance, to realise GSOS rules in the category of cpos and continuous functions as used in domain theory, rather than in Set.

3 Abstract GSOS

In general, given a cartesian category $\mathcal{C}$ and arbitrary functorial notions of program constructs Σ and behaviour B on $\mathcal{C}$, with Σ freely generating the syntax T, one can define a corresponding *abstract* notion of *operational rules* as the natural transformations ρ of type

$$\Sigma(Id \times B) \Rightarrow BT \tag{8}$$

We shall need the following characterisation.

Proposition 3.1 There is a 1-1 correspondence between natural transformations of type $\Sigma(Id \times B) \Rightarrow BT$ and those of type $\Sigma(T \times BT) \Rightarrow BT$.

Proof. One direction of the correspondence is given by the mapping $\rho \mapsto \varrho = B\mu \circ \rho_T : \Sigma(T \times BT) \Rightarrow BT$, for $\rho : \Sigma(Id \times B) \Rightarrow BT$. In the converse direction, simply precompose $\varrho : \Sigma(T \times BT) \Rightarrow BT$ with $\Sigma(\eta \times B\eta)$. $\square$

Several examples illustrating the use of the generality of (8) are given in [30]. Here is a brief summary thereof. Firstly, the operational rules of deterministic programs with exceptions and side-effects can be modelled instantiating (8) with the behaviour endofunctor $BX = (S \cdot (1 + X))^S$, where $S \cdot Y$ is the copower $\coprod_S Y$.

As shown below, the behavioural equivalence corresponding to $BX = (\mathcal{P}_{fi} X)^A$ is bisimulation; a coarser equivalence, namely *trace equivalence*, can be obtained by considering the endofunctor $BX = 1 + A \cdot X$ on the category $SL(\mathcal{C})$ of semi-lattices in a category $\mathcal{C}$. (This is a simplified version of the behaviour in [14].) The program construct endofunctor to be considered then is $\Sigma^\otimes : SL(\mathcal{C}) \to SL(\mathcal{C})$, a monoidal generalisation of the endofunctor Σ on cartesian categories. For instance, for the language of §1, $\Sigma^\otimes X = 1 + \coprod_A X + (X \otimes X)$, where '$\otimes$' is the tensor product of semi-lattices.

Note that (8) can be instantiated to rules not only for single-sorted languages but also for multi-sorted ones; it suffices to work with signatures (and behaviours) over 'power categories'.

Finally, we briefly consider recursion. GSOS stands for 'SOS for non-deterministic programs with *guarded* recursion', because the full definition also allows for definitions of programs by guarded recursion. In [30], a functorial notion of guard is given which allows one to generalise the definitions by guarded recursion to abstract rules of type (8). Moreover, one can also treat *unguarded recursion* by realising the abstract rules, for instance, in the category of cpos and partial continuous functions and exploiting algebraic compactness. This involves precomposing the endofunctor Σ with the lifting endofunctor, so that one freely generates not only finite but also partial and infinite terms, the latter being used to unfold recursive definitions of programs.

4 Coalgebras

The *intended operational model* of a set of concrete GSOS rules is the least relation $R \subseteq T\emptyset \times A \times T\emptyset$ which satisfies the rules, where $x \xrightarrow{a} x'$ stands for $\langle x, a, x' \rangle \in R$. In general, a relation of type $X \times A \times X$ is called a *labelled transition system* [21] with set of states X and set of labels A.

For image finite sets of GSOS rules it suffices to consider *image finite* transition systems, where, for each state and each action, the image of the transition relation is a finite set. These are in 1-1 correspondence with functions $k : X \to (\mathcal{P}_{\mathrm{fi}} X)^A$ as follows.

$$x \xrightarrow{a} x' \iff x' \in k(x)(a) \tag{9}$$

If, as considered here, the set A is finite, then image finite transition systems cut down to *finitely branching* transition systems, where for each state, the set of outgoing transitions is finite.

A function $k : X \to (\mathcal{P}_{\mathrm{fi}} X)^A$ is a coalgebra of the endofunctor $BX = (\mathcal{P}_{\mathrm{fi}} X)^A$ on Set. Formally, given an endofunctor $B : \mathcal{C} \to \mathcal{C}$, a *$B$-coalgebra* is a pair $\langle X, k\rangle$, where the carrier X is an object and the structure $k : X \to BX$ is a morphism of $\mathcal{C}$. One often identifies a coalgebra $\langle X, k\rangle$ with its structure k.

The B-coalgebras form a category B-Coalg, with homomorphisms $f : \langle X, k\rangle \to \langle X', k'\rangle$ the morphisms

$$\begin{tikzcd} X \arrow[r, "f"] \arrow[d, "k"'] & X' \arrow[d, "k'"] \\ BX \arrow[r, "Bf"'] & BX' \end{tikzcd}$$

$f : X \to X'$ between the carriers such that $k' \circ f = (Bf) \circ k$. Note the forgetful functor

$$U_B : B\text{-}\mathsf{Coalg} \to \mathcal{C} \qquad \langle X, k\rangle \mapsto X$$

mapping coalgebras to their carriers.

For $BX = (\mathcal{P}_{\mathrm{fi}} X)^A$, the coalgebra homomorphisms are, up to the correspondence (9), the same as the P-*open morphisms* of [16], where P is a suitable category of finite sequences of actions. (Thus, for this choice of B, B-Coalg is a proper subcategory of the standard category of transition systems [31].) As a consequence, two transition systems are (strongly) *bisimilar* [17] if and only if there is a span of coalgebra homomorphisms between them. This leads to the following coalgebraic notion of bisimulation, a mild generalisation of the one in [4].

Definition 4.1 (Coalgebraic Bisimulation) A *B-bisimulation* between two coalgebras $\langle X_1, k_1\rangle$ and $\langle X_2, k_2\rangle$ of an endofunctor B is a triple $\langle X, f_1, f_2\rangle$ such that such that there exists a coalgebra structure $k : X \to BX$ making $\langle\langle X, k\rangle, f_1, f_2\rangle$ a span

$$\begin{tikzcd} & \langle X, k\rangle \arrow[dl, "f_1"'] \arrow[dr, "f_2"] & \\ \langle X_1, k_2\rangle & & \langle X_2, k_2\rangle \end{tikzcd}$$

of coalgebra homomorphisms f_1,f_2. □

One can form the *category of B-bisimulations* between two coalgebras $\langle X_1, k_1\rangle$ and $\langle X_2, k_2\rangle$ of an endofunctor B on a category $\mathcal{C}$: the morphisms $g : \langle X, f_1, f_2\rangle \to \langle X', f_1', f_2'\rangle$ are those $g : X \to X'$ in $\mathcal{C}$ (thus not necessarily coalgebra homomorphisms) such that $f_i = f_i' \circ g$, for $i = 1, 2$.

Let B be an endofunctor on a category $\mathcal{C}$ with kernel pairs and let the *internal equality* of a coalgebra $\langle X, k\rangle$ be the kernel pair (in the underlying category $\mathcal{C}$) of the identity on its carrier X. One can easily prove that:

Proposition 4.1 (Strong Extensionality) Internal equality is the final B-bisimulation of the final B-coalgebra. □

In general, final coalgebras need not exist, but if $\mathcal{C}$ has a final object 1, and the forgetful functor U_B has a right adjoint $G_B : \mathcal{C} \to B$-Coalg, then $G_B 1$ is the final B-coalgebra. For the endofunctor $BX = (\mathcal{P}_{\mathrm{fi}} X)^A$ on Set, such a right adjoint G_B exists [6]. It follows [29, §13] that the final coalgebra $G_B 1$ is the set of rooted, image finite trees, with branches labelled by $a \in A$, quotiented by (ordinary) bisimulation. This is the set of 'abstract global behaviours', ie the (abstract) non-deterministic processes.

Semantically, the above strong extensionality result specialises then to the fact that such a final coalgebra is *internally fully-abstract* [1] with respect to bisimulation, ie its largest bisimulation is the equality, hence bisimilar elements are indistinguishable.

5 Operational Monads

Definition 5.1 Let T and B be endofunctors on the same category $\mathcal{C}$. An endofunctor $\widetilde{T}$ on the category of B-coalgebras *lifts the endofunctor* T to the B-coalgebras if $U_B \widetilde{T} = T U_B$, ie the diagram

$$\begin{tikzcd} B\text{-}\mathsf{Coalg} \arrow[r, "\widetilde{T}"] \arrow[d, "U_B"'] & B\text{-}\mathsf{Coalg} \arrow[d, "U_B"] \\ \mathcal{C} \arrow[r, "T"'] & \mathcal{C} \end{tikzcd}$$

commutes. (Cf [15].)

When both T and $\widetilde{T}$ are monads, $\widetilde{T}$ *lifts the monad* T to the B-coalgebras if the forgetful functor $U_B : B$-$\mathsf{Coalg} \to \mathcal{C}$ (together with the identity natural transformation) is a monad morphism [27] from $\widetilde{T}$ to T. □

Remark 5.1 A monad $\widetilde{T}$ lifts a monad $T = \langle T, \eta, \mu\rangle$ to the B-coalgebras if and only if $U_B\widetilde{T} = TU_B$ and, for every B-coalgebra $k : X \to BX$, the diagram

$$\begin{tikzcd}
X \arrow[r, "\eta_X"] \arrow[d, "k"'] & TX \arrow[d, "\widetilde{T}(k)"'] & T^2X \arrow[l, "\mu_X"'] \arrow[d, "\widetilde{T}^2(k)"] \\
BX \arrow[r, "B\eta_X"'] & BTX & BT^2X \arrow[l, "B\mu_X"]
\end{tikzcd}$$

commutes. ◻

Consider now T to be the monad freely generated by an endofunctor Σ. The adjunction $F^\Sigma \dashv U^\Sigma$ gives a well-known structural recursion theorem which specialises to the ordinary recursion (or iteration) theorem for natural numbers, covering the simplest form of primitive recursive functions, but not others such addition, multiplication, exponentiation, etc, which need parameters and 'accumulators'. (By structural recursion we mean definition by structural induction.) Here we shall need the following 'folklore' structural recursion theorem [20] with accumulators, ie with terms as parameters of the recursive definition.

Theorem 5.1 (Structural Recursion) Let T be a monad freely generated by an endofunctor Σ on a cartesian category $\mathcal{C}$ and let $\psi_X : \Sigma TX \to TX$ be the structure of the free Σ-algebra over an object X of $\mathcal{C}$. For all morphisms $f : X \to Y$ and $h : \Sigma(TX \times Y) \to Y$ in $\mathcal{C}$ there exists a unique morphism $f^\sharp : TX \to Y$ in $\mathcal{C}$ such that

$$\begin{tikzcd}
X \arrow[r, "\eta_X"] \arrow[dr, "f"'] & TX \arrow[d, dashed, "f^\sharp"] & \Sigma TX \arrow[l, "\psi_X"'] \arrow[d, "\Sigma\langle id, f^\sharp\rangle"] \\
& Y & \Sigma(TX \times Y) \arrow[l, "h"]
\end{tikzcd}$$

commutes.

Proof. Turn h into the Σ-algebra structure $\langle \psi_X \circ \Sigma\pi_1, h\rangle : \Sigma(TX \times Y) \to TX \times Y$ over the product $TX \times Y$ and then apply the ordinary structural recursion theorem to it and $\langle \eta_X, f\rangle : X \to TX \times Y$. ◻

Recall Proposition 3.1. For every map

$$\varrho_X : \Sigma(TX \times BTX) \to BTX \tag{10}$$

and every coalgebra $k : X \to BX$, define the coalgebra $T_\varrho(k) : TX \to BTX$ to be the unique map

$$\begin{tikzcd}
X \arrow[r, "\eta_X"] \arrow[d, "k"'] & TX \arrow[dd, dashed, "T_\varrho(k)"] & \Sigma TX \arrow[l, "\psi_X"'] \arrow[dd, "\Sigma\langle id, T_\varrho(k)\rangle"] \\
BX \arrow[dr, "B\eta_X"'] & & \\
& BTX & \Sigma(TX \times BTX) \arrow[l, "\varrho_X"]
\end{tikzcd}$$

given by the above theorem.

Proposition 5.1 If the morphism ϱ_X is natural in X, then the above construction $k \mapsto T_\varrho(k)$ extends to a monad T_ϱ lifting T to the B-coalgebras.

Proof. First one needs to prove that, for every coalgebra homomorphism $f : \langle X, k\rangle \to \langle X', k'\rangle$, Tf is a coalgebra homomorphism, ie

$$T_\varrho(k') \circ Tf = BTf \circ T_\varrho(k)$$

so that one can define $T_\varrho f$ to be Tf. For this, simply note that both composites $T_{\mathcal{R}}(k') \circ Tf$ and $BTf \circ T_{\mathcal{R}}(k)$ fit as the unique morphism

$$\begin{tikzcd}
X \arrow[r, "\eta_X"] \arrow[d, "f"'] & TX \arrow[dddd, dashed, "!"] & \Sigma TX \arrow[l, "\psi_X"'] \arrow[d, "\Sigma\langle id, !\rangle"] \\
X' \arrow[d, "k'"'] & & \Sigma(TX \times BTX') \arrow[ddd, "\Sigma\langle Tf, id\rangle"] \\
BX' \arrow[ddr, "B\eta_{X'}"'] & & \\
& & \\
& BTX' & \Sigma(TX' \times BTX') \arrow[l, "\varrho_{X'}"]
\end{tikzcd}$$

given by Theorem 5.1, hence they must be equal. (The naturality of ϱ is essential here!)

Next, one has to verify that the endofunctor T_ϱ lifts the operations of the monad T. From Remark 5.1, it suffices to show that, for every coalgebra structure $k : X \to BX$, $T_\varrho(k) \circ \eta_X = B\eta_X \circ k$ and $T_\varrho(k) \circ \mu_X = B\mu_X \circ T_\varrho^2(k)$, ie the unit and the multiplication of T are coalgebra homomorphisms. For the unit, this is immediate by definition of the functor T_ϱ, while for the multiplication one also needs to use the naturality of ρ and the fact that μ is defined by (ordinary) structural recursion on the free algebra structure. ◻

Definition 5.2 The *operational monad* induced by some abstract operational rules $\rho : \Sigma(Id \times B) \Rightarrow BT$, is the monad T_ϱ corresponding to the composite natural transformation $\varrho = B\mu \circ \rho_T : \Sigma(T \times BT) \Rightarrow BT$. We write T_ρ for this monad. ◻

Let us try and understand the operational monad T_ρ when $\rho = [\![\mathcal{R}]\!]$, for $\mathcal{R}$ a set of concrete GSOS rules. Firstly, applying ρ to TX amounts to instantiating the meta-variables of the rules with the terms in TX. Formally, in this way the term t in a GSOS rule (3) might contain terms as variables: one needs to apply to it the multiplication of the term monad T in order to 'unbracket' it and obtain an elementary term. This is achieved here by composing ρ_{TX} with $B\mu_X$.

Next, recall the correspondence (9) between coalgebras $k : X \to BX = (\mathcal{P}_{\mathrm{fi}}X)^A$ and image finite transition systems. By regarding X as a set of constants rather than as a set of states, the correspondence (9) can also be seen as being between coalgebras and sets of *δ-rules* [8], ie axiom rules. Up to these two correspondences, one can then check that $k \mapsto T_\rho(k)$ is the usual construction of a transition system for a finite set of GSOS rules $\mathcal{R}$ and a possibly infinite (but image finite) set k of δ-rules. In particular, if X is the empty set, hence k is the trivial coalgebra $0 : \emptyset \to B\emptyset$ and $TX = T\emptyset$ is the set of closed terms, this construction gives the intended operational model for the rules.

These remarks hold for arbitrary rules of type (3) and, correspondingly, to possibly non-natural functions $[\![\mathcal{R}]\!]_X$. The naturality of GSOS ensures that T_ρ is an operational monad, which is essential for applying the theory in §7.

6 'Dualising' GSOS: Tree Rules

The duality between algebras and coalgebras can be exploited to find a format of rules 'dual' to abstract GSOS as follows.

Let Σ and B be two endofunctors on a cocartesian category $\mathcal{C}$ and let $D = \langle D, \varepsilon, \delta\rangle$ be the *cofree comonad generated* by B, that is, the forgetful functor U_B has a right adjoint $G_B : \mathcal{C} \to B\text{-}\mathsf{Coalg}$ and

$$D = U_B G_B \tag{11}$$

By the dual of Theorem 5.1 and Definition 5.1, every natural transformation

$$\varrho : \Sigma D \Rightarrow B(D + \Sigma D)$$

coinductively defines a lifting D_ρ of the comonad D to the Σ-algebras:

$$\begin{tikzcd} \Sigma\text{-}\mathsf{Alg} \arrow[r, "D_\varrho"] \arrow[d, "U^\Sigma"'] & \Sigma\text{-}\mathsf{Alg} \arrow[d, "U^\Sigma"] \\ \mathcal{C} \arrow[r, "D"'] & \mathcal{C} \end{tikzcd}$$

In particular, such a lifting can be obtained from natural transformations

$$\rho : \Sigma D \Rightarrow B(Id + \Sigma) \tag{12}$$

by dualising Proposition 3.1 and putting $\varrho = \rho_D \circ \Sigma\delta : \Sigma D \Rightarrow B(D + \Sigma D)$. This is the *denotational comonad* D_ρ coinduced by ρ.

Let Σ freely generate a monad T. In the next section, Theorem 7.1 shows that liftings of the comonad D to the Σ-algebras are in 1-1 correspondence with liftings of the monad T to the B-coalgebras. Therefore, if Σ corresponds to some program constructs and B to some behaviour, every natural transformation ρ as in (12) defines also an operational monad, say T_ρ (with a slight abuse of notation).

As mentioned in §4, for the endofunctor $BX = (\mathcal{P}_{\mathrm{fi}}X)^A$ on Set the adjunction $U_B \dashv G_B$ exists. The value of the corresponding cofree comonad $D = U_B G_B$ at a set X is the set of 'global behaviours with states x in X'. Formally, it is a quotient of the set of rooted, image finite trees, with branches labelled by $a \in A$, and nodes labelled by $x \in X$; the quotient is taken with respect to a form of bisimulation taking into account the name of the nodes [29, §13]. The counit $\varepsilon : D \Rightarrow Id$ is the operation which extracts the root from a tree and the comultiplication $\delta : D \Rightarrow D^2$ is the operation which replaces the name of every node in a tree by the subtree starting at that node.

Next, consider rules of type

$$\frac{\{z_i \xrightarrow{a_i} y_i\}_{i\in I} \qquad \{v_j \xrightarrow{b_j}\!\!\!\!\!\not\ \ \}_{j\in J}}{\sigma(x_1, \ldots, x_n) \xrightarrow{c} t} \tag{13}$$

where the x_k, y_i, z_i, and v_j are all variables, and I and J are countable, possibly infinite index sets. It is convenient to consider the *dependency graph* [13] of such a rule, namely the directed graph having the variables of the rule as nodes, $z_i \xrightarrow{a_i} y_i$, for i in I as 'positive' edges, and $v_j \xrightarrow{b_j}\!\!\!\!\!\not\ \ $ as 'negative', targetless edges. A rule of type (13) is *well-founded* if all backwards chains of edges in its dependency graph are finite [13].

Definition 6.1 (Tree rules [10, 11]) A (*simple negative*) *tree rule* is a well-founded rule of type (13) such that the x_k and the y_i are all distinct variables and are the only variables occurring in the rule (ie the z_i and v_j are all occurrences of the x_k and y_i).

A tree rule is *safe* if the term t either is a variable x or is of the form $\sigma'(x'_1, \ldots, x'_m)$ for some operator σ' of the signature and some (not necessarily distinct) variables $x'_1, \ldots, x'_m$. $\square$

Tree rules are more general than GSOS: they allow for 'lookahead', in that one can look not only at the local behaviour (a single transition $x \xrightarrow{a} y$) of the states like in GSOS, but also at the global one, as in $x \xrightarrow{a} y \xrightarrow{b} y'$. (See [13] for some examples.) The safety restriction does not affect the expressive power of the rules, provided one is allowed to add sufficiently many auxiliary operators to the signature.

A tree rule (13) has the property that its dependency graph is equal to the graph reachable from the nodes $x_1, \ldots, x_n$. Moreover, the subgraph reachable from a node x_k is a tree – the *dependency tree* with root x_k. Let us call a set of tree rules *allowed* if it is an image finite set (in the sense of §1) of tree rules whose dependency trees are image finite. Then, an allowed set $\mathcal{R}$ of tree rules defines, for every X, a function

$$[\![\mathcal{R}]\!]_X : \Sigma DX \to (\mathcal{P}_{fi}TX)^A \tag{14}$$

as follows.

For all t in TX, c in A, and d_k in DX, put

$$t \in [\![\mathcal{R}]\!]_X(\sigma(d_1, \ldots, d_n))(c)$$

if and only if there exists a (possibly renamed) rule (13) in $\mathcal{R}$ such that the root of d_k is x_k, for $1 \le k \le n$, and the dependency trees of the rule can be embedded in the d_k (where the convention is that a tree with a negative edge $v_j \overset{b_j}{\not\longrightarrow}$ can be embedded into d_k only if the variable in d_k corresponding to v_j does not have an outgoing edge labelled by b_j).

Theorem 6.1 (Tree rules are natural) Let D be the comonad cofreely generated by the endofunctor $BX = (\mathcal{P}_{fi}X)^A$ on Set.

For every allowed set $\mathcal{R}$ of tree rules the function $[\![\mathcal{R}]\!]_X$ in (14) is natural in X.

Proof. Similar to the proof of naturality in Theorem 1.1. Note the well-foundedness of tree rules is needed. For instance, the non-well-founded rule with premise $x \xrightarrow{a} x$ and conclusion $a.x \xrightarrow{a} \mathsf{nil}$ is not natural because: first applying $[\![a.]\!]$ to $(x \xrightarrow{a} y)$ and then renaming y as x yields $a \rightsquigarrow \{x\}$, while the same operations but in the reverse order yield $a \rightsquigarrow \{x, \mathsf{nil}\}$, which fact violates naturality. □

In particular, if the rules in $\mathcal{R}$ are safe, the natural transformation $[\![\mathcal{R}]\!]$ is of type

$$\Sigma D \Rightarrow (\mathcal{P}_{fi}(Id + \Sigma))^A$$

Therefore, for every allowed set $\mathcal{R}$ of safe tree rules there exists a transition system which satisfies the rules, namely $T_\rho(0)$, where $\rho = [\![\mathcal{R}]\!]$ and T_ρ is the corresponding operational monad. Contrarily to what is stated in [11], this fails for (simple negative) tree rules, as the failure to fit these latter rules in the present theory brought to light. In fact, the safe tree rules themselves have been suggested to us by Rob van Glabbeek as a natural subclass of (negative) tree rules possessing a satisfying transition system.

7 Combining Operational and Denotational Models

When T is the monad freely generated by an endofunctor Σ on a category $\mathcal{C}$, then one can easily see that the category Σ-Alg of algebras of the endofunctor Σ is isomorphic to the category T-Alg of algebras of the monad $T = \langle T, \eta, \mu \rangle$, with objects those morphisms $h : TX \to X$ in $\mathcal{C}$ such that $h \circ \eta_X = id$ and $h \circ Th = h \circ \mu_X$. Dually, the category B-Coalg of B-coalgebras is isomorphic to the category D-Coalg of coalgebras of the comonad D cofreely generated by B [29, §7]. The results in this section should be read up to these two isomorphisms of categories

$$\Sigma\text{-}\mathsf{Alg} \cong T\text{-}\mathsf{Alg} \qquad B\text{-}\mathsf{Coalg} \cong D\text{-}\mathsf{Coalg}$$

7.1 Distributive Laws

Given a monad $T = \langle T, \eta, \mu \rangle$ and a comonad $D = \langle D, \varepsilon, \delta \rangle$ on a category $\mathcal{C}$, a *distributive law* [7] of the monad T over the comonad D is a natural transformation $\lambda : TD \Rightarrow DT$ satisfying the laws

$$\lambda \circ \eta_D = D\eta \qquad \lambda \circ \mu_D = D\mu \circ \lambda_T \circ T\lambda$$

and their dual

$$T\varepsilon = \varepsilon_T \circ \lambda \qquad D\lambda \circ \lambda_D \circ T\delta = \delta_T \circ \lambda$$

The following theorem may well be folklore.

Theorem 7.1 For a monad T and a comonad D on the same category, the following notions are mutually equivalent.

- Distributive laws λ of T over D.
- Liftings $\widetilde{T}$ of T to the D-coalgebras.
- Liftings $\widetilde{D}$ of D to the T-algebras.

Proof. Given a distributive law λ, one can define the corresponding liftings as follows.

$$T_\lambda(k) = \lambda_X \circ Tk \qquad D_\lambda(h) = Dh \circ \lambda_X$$

Conversely, consider a lifting $\widetilde{T}$ of the comonad D to the T-algebras, hence $U_D\widetilde{T} = TU_D$. By Lemma 1 in [15], this determines a distributive law λ of the monad T over the *endofunctor* D as follows: first take the natural transformation $T\varepsilon : U_D\widetilde{T}G_D = TU_DG_D = TD \Rightarrow T$, then transpose it across the adjunction $U_D \dashv G_D$ obtaining $\overline{\lambda} : \widetilde{T}G_D \Rightarrow G_DT$, and finally define λ to be $U_D\overline{\lambda} : U_D\widetilde{T}G_D = TD \Rightarrow DT$. It is easy to prove that λ actually is a distributive law over the whole comonad D. Dually, given a lifting $\widetilde{D}$, take $\eta_D : D \Rightarrow DT = DU^TF^T = U^T\widetilde{D}F^T$, transpose it across $F^T \dashv U^T$ obtaining $\overline{\lambda} : F^TD \Rightarrow \widetilde{D}F^T$, and define λ to be $U^T\overline{\lambda} : U^TF^TD = TD \Rightarrow DT = U^T\widetilde{D}F^T$. The constructions are easily seen to be mutually inverse. $\Box$

When T is syntax and D is (global) behaviour, the type of the distributive law λ might thought of as 'the most general type of well-behaved rules'. Note one can also consider monads T corresponding to algebraic theories, with equations between the derived operators. (See [29, §10] for an elementary example.)

7.2 Bialgebras as Models

Given a distributive law $\lambda : TD \Rightarrow DT$, one can consider the *category* λ-Bialg *of* λ-*bialgebras*. Its objects are pairs $TX \xrightarrow{h} X \xrightarrow{k} DX$ of T-algebras and D-coalgebras with a common carrier X which satisfy the following 'pentagonal law':

$$k \circ h = Dh \circ \lambda_X \circ Tk$$

(Cf [28].) This law makes h a coalgebra homomorphism and k an algebra homomorphism. The morphisms $f : \langle X, h, k\rangle \to \langle X', h', k'\rangle$ of λ-Bialg are those morphisms $f : X \to X'$ between the carriers which are both T-algebra and D-coalgebra homomorphisms.

Remark 7.1 The λ-bialgebras are the same as the algebras of the monad T_λ of Theorem 7.1, and, dually, the same as the coalgebras of the comonad D_λ:

$$T_\lambda\text{-}\mathsf{Alg} \cong \lambda\text{-}\mathsf{Bialg} \cong D_\lambda\text{-}\mathsf{Coalg}$$

Remark 7.2 When λ is the distributive law induced by a finite set of concrete GSOS rules, the λ-bialgebras are the GSOS-models of [25]. $\Box$

Given a Σ-algebra $h : \Sigma X \to X$, let $h^* : TX \to X$ be its inductive extension to a T-algebra. When λ is induced by some abstract operational rules ρ, no matter whether of type (8) or the dual (12), λ-bialgebras are equivalent to pairs $\Sigma X \xrightarrow{h} X \xrightarrow{k} BX$ such that

$$k \circ h = B(h^*) \circ \rho_X \circ \Sigma\langle id, k\rangle \qquad (15)$$

The algebra structure can be thought of as a denotational model, the coalgebra structure as an operational model, and the pentagonal law (15) says that the combination of the two models satisfies the rules ρ. Henceforth such bialgebras are called ρ-*models*.

7.3 Adequacy Meta-Results

Consider the forgetful functor

$$U^\lambda : \lambda\text{-}\mathsf{Bialg} \to D\text{-}\mathsf{Coalg} \qquad \langle X, h, k\rangle \mapsto \langle X, k\rangle$$

which forgets the algebra structure of a λ-bialgebra.

Theorem 7.2 U^λ has a left adjoint, namely:

$$(X \xrightarrow{k} DX) \overset{F^\lambda}{\longmapsto} (T^2X \xrightarrow{\mu_X} TX \xrightarrow{T_\lambda(k)} DTX)$$

Proof. Dualise Theorem 4 of [15] and apply it to

$$\begin{array}{ccc} D\text{-}\mathsf{Coalg} & \xleftarrow{U^\lambda} & D_\lambda\text{-}\mathsf{Coalg} \\ \downarrow{\scriptstyle U_D} & & \downarrow{\scriptstyle U_{D_\lambda}} \\ \mathcal{C} & \xleftarrow[U^T]{} & T\text{-}\mathsf{Alg} \end{array}$$

where λ-$\mathsf{Bialg} \cong D_\lambda$-$\mathsf{Coalg}$ by Remark 7.1. $\Box$

Corollary 7.1 The category of λ-bialgebras has an initial object, namely $F^\lambda 0$, where 0 is the trivial initial D-coalgebra. $\Box$

In particular, there exists an initial ρ-model, which can be regarded as the intended operational model over the initial algebra of programs $T0$. This implies that every ρ-model M is *adequate* with respect to the intended operational model of ρ. Indeed, the unique ρ-model homomorphism to M given by initiality is a denotational interpretation which preserves the behavioural distinctions of the intended operational model. This makes M adequate.

Now, consider the 'dual' of U^λ, namely the functor

$$U_\lambda : \lambda\text{-}\mathsf{Bialg} \to T\text{-}\mathsf{Alg} \qquad \langle X, h, k\rangle \mapsto \langle X, h\rangle$$

which forgets the D-coalgebra structure of a λ-bialgebra. Correspondingly, the following is dual to Theorem 7.2.

Theorem 7.3 U_λ has a right adjoint, namely:

$$(TX \xrightarrow{h} X) \overset{G_\lambda}{\longmapsto} (TDX \xrightarrow{D_\lambda(h)} DX \xrightarrow{\delta_X} D^2X)$$

Corollary 7.2 The category of λ-bialgebras has a final object, namely $G_\lambda 1$, where 1 is the trivial final T-algebra. $\Box$

In particular, there exists a final ρ-model which is the *canonical* denotational model for ρ; it has the final D-coalgebra as carrier which, as mentioned in §4, is internally fully abstract with respect to B-bisimulation. The construction $1 \mapsto G_\varrho 1 = \langle D1, D_\varrho(1), \delta_1 \rangle$ generalises the 'processes as terms' construction of [22], which is a systematic method for deriving adequate denotational models from 'tyft rules' [13] (a class of rules equivalent to the tree rules without negative premises [10]). For more details, see [30].

Corollary 7.3 The unique (both by initiality and finality) homomorphism from the initial to the final ρ-model is both the initial algebra semantics and the final coalgebra semantics for ρ. □

The above, say, *universal semantics* for ρ is thus a compositional interpretation of the programs which preserves their behavioural distinctions. In Set, the latter means that two programs with the same universal semantics are B-bisimilar. One can easily see that, under the additional hypothesis that B preserves weak pullbacks, the converse also holds: two programs have the same universal semantics if and only if they are B-bisimilar. In other words:

Corollary 7.4 If B preserves weak pullbacks, the universal semantics associated to some abstract rules ρ is *fully abstract with respect to B-bisimulation.* □

Next, recall Definition 4.1: by replacing spans of coalgebra homomorphisms with spans of T-algebra homomorphisms, one has a corresponding notion of T-*congruence* which specialises to the ordinary notion of congruence. Similarly, by considering spans of λ-bialgebra homomorphisms one has a notion of, say, λ-*bicongruence* and a corresponding category. We can ask then whether there exists a final bicongruence for a λ-bialgebra. Now, if pullbacks of cospans of carriers of B-coalgebras are B-bisimulations, then, by the universal property of pullbacks, a final B-bisimulation between two coalgebras exists: it is the pullback of the respective unique coalgebra homomorphisms to the final coalgebra. This is a T-congruence as well, because the forgetful functor $U^T : T\text{-}\mathsf{Alg} \to \mathcal{C}$ creates limits. Therefore, by definition of final bialgebra:

Corollary 7.5 If B preserves weak pullbacks, then every λ-bialgebra has a final bicongruence. □

In particular, the behavioural endofunctor B in (2) preserves weak pullbacks, hence the above corollary specialises to the well-known fact that (strong) bisimulation is a congruence for GSOS and tree rules.

Future Work

The major challenge ahead is the operational semantics of the languages with variable binders, such as the π-calculus and the λ-calculus. (At the moment, by working in a suitable functor category, we are able to give a functorial description of syntax with variable binders, but it is not yet clear whether this fits our purposes.) We would also like to obtain adequacy results when working in categories of partial maps.

There is an obvious question about Moggi's computational monads [19] and our behaviour functors which remains to be investigated. In a different direction, we would like to understand the relationship between the transitional approach considered here and others, such as the the *reductional* one arising in the λ-calculus and term-rewriting in general.

Further developments of the present theory could lead to applications in modular compiler development technology. Perhaps there will be a useful theory of the combination of operational semantics of different languages (cf [18]). Again, perhaps one can relate the operational semantics of a language with that of its translation into another target language (cf [26]).

Acknowledgements. Thanks to Marcelo Fiore and Alex Simpson for discussions. Part of this study is based on the first author's thesis; he wishes to thank Jaco de Bakker and Bart Jacobs for their guidance.

References

[1] S. Abramsky. A domain equation for bisimulation. *Information and Computation*, 92:161–218, 1991.

[2] P. Aczel. *Non-well-founded sets.* Number 14 in Lecture Notes. CSLI, 1988.

[3] P. Aczel. Final universes of processes. In *Mathematical Foundations of Programming Semantics, Proc. 9th Int. Conf.*, volume 802 of *LNCS*, pages 1–28. Springer-Verlag, 1994.

[4] P. Aczel and N. Mendler. A final coalgebra theorem. In D.H. Pitt et al., editors, *Proc. category theory and computer science*, volume 389 of *LNCS*, pages 357–365. Springer-Verlag, 1989.

[5] J.C.M. Baeten and W.P. Weijland. *Process Algebra.* Cambridge University Press, 1990.

[6] M. Barr. Terminal coalgebras in well-founded set theory. *Theoretical Computer Science*, 144(2):299–315, 1993.

[7] J. Beck. Distributive laws. In B. Eckmann, editor, *Seminar on Triples and Categorical Homology Theory*, volume 80 of *Lecture Notes in Mathematics*, pages 119–140. Springer-Verlag, 1969.

[8] B. Bloom, S. Istrail, and A.R. Meyer. Bisimulation can't be traced. *Journal of the ACM*, 42(1):232–268, jan 1995. A preliminary report appeared in *Proc. 3rd LICS*, pages 229-239, 1988.

[9] J.R.B. Cockett and D.A. Spooner. Categories for synchrony and asynchrony. *Electronic Notes in Theoretical Computer Science*, 1, 1995.

[10] W. Fokkink. The tyft/tyxt format reduces to tree rules. In M. Hagiya and J.C. Mitchell, editors, *Proc. TACS94*, number 789 in LNCS, pages 440–453. Springer-Verlag, 1994.

[11] W. Fokkink and R. van Glabbeek. Ntyft/ntyxt rules reduce to ntree rules. *Information and Computation*, 126(1):1–10, 1996.

[12] J.A. Goguen, J.W. Thatcher, and E.G. Wagner. An initial algebra approach to the specification, correctness and implementation of abstract data types. In R.T. Yeh, editor, *Current Trends in Programming Methodology*, volume IV, pages 80–149. Prentice Hall, 1978.

[13] J.F. Groote and F. Vaandrager. Structured operational semantics and bisimulation as a congruence. *Information and Computation*, 100(2):202–260, 1992.

[14] M.C.B. Hennessy and G.D. Plotkin. Full abstraction for a simple parallel programming language. In J. Bečvář, editor, *Proc. 8th MFCS*, volume 74 of *LNCS*, pages 108–120. Springer-Verlag, 1979.

[15] P.T. Johnstone. Adjoint lifting theorems for categories of algebras. *Bull. London Math. Soc.*, 7:294–297, 1975.

[16] A. Joyal, M. Nielsen, and G. Winskel. Bisimulation and open maps. In *Proc. Eighth IEEE Symp. on Logic In Computer Science*, pages 418–427, 1993.

[17] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *LNCS*. Springer-Verlag, 1980.

[18] E. Moggi. A category-theoretic account of program modules. *Mathematical Structures in Computer Science*, 1, 1991.

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

[20] A. M. Pitts. Categorical logic. Technical Report 367, University of Cambridge Computer Laboratory, May 1995.

[21] G.D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Computer Science Department, Aarhus University, 1981.

[22] J. Rutten. Processes as terms: non-well-founded models for bisimulation. *Mathematical Structures in Computer Science*, 2:257–275, 1992.

[23] J. Rutten and D. Turi. Initial algebra and final coalgebra semantics for concurrency. In J. de Bakker et al., editors, *Proc. of the REX workshop A Decade of Concurrency – Reflections and Perspectives*, volume 803 of *LNCS*, pages 530–582. Springer-Verlag, 1994.

[24] D. Scott. Outline of a mathematical theory of computation. In *Proc. 4th Annual Princeton Conference on Inf. Sciences and Systems*, pages 169–176, 1970.

[25] A.K. Simpson. Compositionality via cut-elimination: Hennessy-Milner logic for an arbitrary GSOS. In *Proc. Tenth IEEE Symp. on Logic In Computer Science*, 1995.

[26] C. Stone and R. Harper. A type-theoretic account of Standard ML 1996. Technical Report CMU-CS-96-136, Computer Science Department, Carnegie-Mellon University, 1996.

[27] R. Street. The formal theory of monads. *Journal of Pure and Applied Algebra*, 2:149–168, 1972.

[28] M.E. Sweedler. *Hopf Algebras.* W.A. Benjamin Inc., New York, 1969.

[29] D. Turi. *Functorial Operational Semantics and its Denotational Dual.* PhD thesis, Free University, Amsterdam, June 1996. Accessible from `<http://www.dcs.ed.ac.uk/home/dt/>`.

[30] D. Turi. Categorical modelling of structural operational rules: case studies. Preprint, accessible from `<http://www.dcs.ed.ac.uk/home/dt/>`, March 1997.

[31] G. Winskel and M. Nielsen. Models for concurrency. In S. Abramsky et al., editors, *Handbook of logic in computer science*, volume 4. Clarendon Press, Oxford, 1995.