Comodels and Effects in Mathematical Operational Semantics

Faris Abou-Saleh[1] and Dirk Pattinson[2]

[1] Department of Computing, Imperial College London
[2] Research School of Computer Science, Australian National University

Abstract. In the mid-nineties, Turi and Plotkin gave an elegant categorical treatment of denotational and operational semantics for process algebra-like languages, proving compositionality and adequacy by defining operational semantics as a distributive law of syntax over behaviour. However, its applications to stateful or effectful languages, incorporating (co)models of a countable Lawvere theory, have been elusive so far. We make some progress towards a coalgebraic treatment of such languages, proposing a congruence format related to the evaluation-in-context paradigm. We formalise the denotational semantics in suitable Kleisli categories, and prove adequacy and compositionality of the semantic theory under this congruence format.

1 Introduction

Operational models of programming languages and process algebras are often described by a transition system, with transitions given by elementary, atomic evolution steps. For stateful languages, this involves an explicit notion of state, such as the values of program variables at each step of the execution. More abstractly, this state is described by a comodel [19]; computational effects characterise the dependency on state, and other phenomena, in terms of computational branches [5]. We may then understand the denotation of a program as an accumulation of state transformations mapping initial to final states, or an effect-tree describing every possible branch of the computation.

This gives us a powerful tool for reasoning about programs: two programs can be substituted for one another as long as they have the same behaviour, i.e. represent the same mapping from initial to final states, or effect-tree. This reasoning is aided by two key properties: the denotational semantics should be *adequate*, i.e. behaviourally equivalent programs should receive the same denotation, and *compositional*, i.e. the denotation of a program can be expressed in terms of the denotations of its components. These properties must often be proved on a case-by-case basis for different languages. To simplify this task, one often shows these properties are satisfied if the languages are given by operational rules in a particular *congruence format*.

Turi and Plotkin applied this approach in an abstract categorical setting [24], and obtained an elegant proof of adequacy and compositionality for a variety of process algebras. They represented program syntax as an initial algebra, and the semantic domain of denotations as a final coalgebra. Behavioural equivalence was given by coalgebraic bisimilarity, and the congruence format was expressed by a *distributive law* of syntax

F. Pfenning (Ed.): FOSSACS 2013, LNCS 7794, pp. 129–144, 2013.

over behaviour. In concrete instances, this leads to generalisations of the well-known GSOS rule format for a large class of process algebras [9].

However, so far the applications have centered around process algebra; applications of the theory to effectful [15] and comodel-based [14] languages has only been hinted at in the literature. The main stumbling block in applying the theory to these languages is that the final coalgebra is a very fine-grained semantic domain, recording the entire sequence of comodel manipulations or effects, rather than their accumulation.

To solve this problem, we may break the symmetry of the original approach, with syntax as an initial-algebra as before, but expressing behaviour, and the semantic domain of the final coalgebra, in a Kleisli category for a suitable monad. This approach gives a more appropriate characterisation of program behaviour, accumulating state manipulations and/or effects. However, it requires a new treatment of syntactic rule formats, and requires a different approach to proving adequacy and compositionality.

In our previous paper [1], we outlined how the existence of a semantic domain in the Kleisli category requires an enrichment with respect to the category $\mathsf{Cpo}_!$ of ω-complete partial orders with strict, continuous maps. We gave a method for extending operational specifications with effects, and by restricting to a rule format related to evaluation-in-context, we sketched a proof of adequacy and compositionality in terms of syntactic effect trees. However, without a categorical proof, we could not account for effects with equations, or complete the analysis for languages with comodels.

In this paper, we formalise and extend the analysis of our previous paper to incorporate both effects and comodels. We begin by defining transition systems to describe operational models incorporating comodels, effects, or both. We propose congruence formats related to the concept of evaluation-in-context [7], and after characterising behavioural and denotational equivalence in Kleisli categories, we prove adequacy and compositionality of the resulting denotational semantics.

Related Work. Mathematical operational semantics is described in [22,24] and applied to process algebras in [23]. Effects and monads in the semantics of languages are introduced by Moggi in [12] and subsequently by Plotkin, Power et al in [15,16]. Comodels for global state are discussed in [14,19]. Operational rules and adequacy proofs for a purely-effectful, functional language are given in [15]; the conclusion contains the words "one would wish to reconcile this work with the co-algebraic treatment of operational semantics in [33]".

2 Syntax and Behaviour for Stateful and Effectful Languages

This section introduces three kinds of transition systems to represent effectful and/or stateful programming languages, before a formal coalgebraic treatment. We employ two languages as running examples, called While and NDWhile.

Definition 1. (ND)While Syntax. *We define syntax for a language* While *as follows, in three sorts – numeric and boolean expressions, and commands.*

$$N ::= x \mid n \mid N + N \mid +_n(N) \qquad E ::= b \mid N{=}{=}N \mid \ {=}{=}_n(N) \mid \neg E \mid E \wedge E$$
$$P ::= x{:=}N \mid \mathtt{skip} \mid P\,;P \mid \mathtt{while}\,(E)\,\mathtt{do}\,\{P\} \mid \mathtt{if}\,(E)\,\mathtt{then}\,\{P\}\,\mathtt{else}\,\{P\}$$

Here, x is a global variable drawn from locations L, n is a numeral in $\mathbb{N}$, and b is a boolean in $\mathbb{B} = \{true, false\}$. The auxiliary operators $+_n(\cdot)$ and $==_n(\cdot)$ can be read as "add n to $\cdot$" and "n equals $\cdot$". The language NDWhile *adds binary* $\texttt{choose}(\cdot,\cdot)$ *operators at each type, representing a non-deterministic choice of either expression.*

Although this syntax is multi-sorted, for theoretical simplicity we work in a single-sorted setting ($N = E = P$); badly-typed terms will produce an error return value.

2.1 Transition Systems for Stateful and Effectful Languages

We introduce three kinds of transition system to represent operational models with effects and/or persistent state, to be given by a comodel.

The first kind of transition system consists of pairs $\langle p, s\rangle$ of a program p and a state s (drawn from a collection S). This is typified by While, where the states $S = \mathbb{N}^L$ are assignments to global variables $x \in L$ ('locations') of natural numbers $\mathbb{N}$. Program execution is represented by changes in the program and state $\langle p, s\rangle \to \langle p', s'\rangle$, and it may eventually return a final state s' and a value v drawn from some collection V: $\langle p, s\rangle \to \langle \underline{v}, s'\rangle$. Typical transitions are $\langle x, s\rangle \to \langle \underline{s(x)}, s\rangle$ – looking up and returning the value of a variable x in the store – and variable updates, $\langle x = 5, s\rangle \to \langle \underline{*}, s[x \mapsto 5]\rangle$. (We write $*$ for the 'void' return value.) In general, a notion of state S may be derived canonically from a comodel (Definition 2); with this in mind, we will refer to such a stateful transition system as a *comodel-based transition system*, or CTS.

The second kind of operational model, an *effectful transition system* or ETS, records the paths a program execution may take, in terms of syntactic effects or their semantic equivalence classes (see [7]). For instance, given a global variable x, the first step of evaluating an expression like $3 + x$ depends on the value of x. If x is 0, the first step will be $3 + 0$; and so on for other values of x. We may record this information syntactically by a 'read' effect, as follows: $\mathbf{rd}_x(3{+}0, 3{+}1, \ldots)$. Similarly, the first step of evaluating $x = 1; x = 2$ involves setting x to 1, leaving us to evaluate $x = 2$. We record the request to update x by a 'write' effect: $\mathbf{wr}_{x,1}(x = 2)$. A further evaluation step involves another update, giving the result $\mathbf{wr}_{x,1}(\mathbf{wr}_{x,2}(\underline{*}))$ (where $*$ is again the 'void' return value). Here is an example execution combining both effects:

$$x{=}y \to \mathbf{rd}_y(x{=}0, x{=}1, x{=}2, \ldots) \to \mathbf{rd}_y(\mathbf{wr}_{x,0}(\underline{*}), \mathbf{wr}_{x,1}(\underline{*}), \mathbf{wr}_{x,2}(\underline{*}), \ldots).$$

Thus, instead of tracking the state s as in a CTS, one could evaluate While expressions in terms of 'read' and 'write' effects. One must also have a means of identifying syntax trees which we would not want to distinguish semantically, such as $\mathbf{wr}_{x,1}(\mathbf{wr}_{x,2}(\underline{*}))$ and $\mathbf{wr}_{x,2}(\underline{*})$. This amounts to an equational theory for the effects [16].

Another example is given by non-determinism. Given a non-deterministic 'zero or one' function $\mathsf{zo} := \texttt{choose}(0, 1)$, evaluating $\mathsf{zo} + 5$ gives either $0 + 5$ or $1 + 5$, and we represent this situation using a binary effect operator: $\mathbf{or}(0 + 5, 1 + 5)$. Another step produces the final result: $\mathbf{or}(\underline{5}, \underline{6})$. Evaluating $\mathsf{zo} + \mathsf{zo}$ gives multiple nested $\mathbf{or}$'s.

Again, one does not want to distinguish some effect-trees; such equivalences can be enforced by three equations: idempotence $\mathbf{or}(x, x) = x$, symmetry $\mathbf{or}(x, y) = \mathbf{or}(y, x)$, and associativity $\mathbf{or}(x, \mathbf{or}(y, z)) = \mathbf{or}(\mathbf{or}(x, y), z)$.

Note that some execution paths may terminate while others do not: for instance, a 'maybe stop' program could behave as follows: $\mathsf{ms} \to \mathbf{or}(\underline{*}, \mathsf{ms})$. Thus, we express the general form of ETS transitions with notation $p \to \delta((b_i)_{i\in I})$ where δ is an effect-syntax term with I-indexed arguments b_i which may be either terminal values $\underline{v}_i$ or programs p'_i. For convenience, we sometimes use vector notation: $p \to \delta(\tilde{b})$. We will call such a transition system a *syntactic* ETS if we see effects purely as syntax, ignoring the semantic equations on the effects; if we instead quotient the syntactic effect-trees δ by these equations, obtaining transitions involving *equivalence classes* of effect-trees, we call the resulting transition system a *semantic* ETS. For instance, in a syntactic ETS we would distinguish transitions $p \to p'$ and $p \to \mathbf{or}(p', p')$, but not a semantic ETS.

The final kind of transition system combines state S with effects, as needed for languages like NDWhile. Here, keeping track of global variables again require a store; but non-deterministic execution means we must track multiple possible stores and program states, as illustrated by this example execution (again using the 'zero or one' function):

$$\langle x{=}\mathsf{zo}, s\rangle \to \mathbf{or}(\langle x{=}0, s\rangle, \langle x{=}1, s\rangle) \to \mathbf{or}(\langle \underline{*}, s[x \mapsto 0]\rangle, \langle \underline{*}, s[x \mapsto 1]\rangle)$$

Generally, such transitions are of form $\langle p, s\rangle \to \delta(\langle b_i, s_i\rangle_{i\in I})$, where each b_i is either a terminal value or a program term (like an ETS). We call such a transition system a *syntactic comodel and effect-based transition system (*CETS*)*, and again, if we choose to identify semantically equivalent effect-trees, we call the result a *semantic* CETS.

2.2 Transition Systems, Categorically

Now we give an overview of the categorical structure we will use to build a semantic theory for the transition systems described above. The main three constructions are (1) program syntax; (2) effect syntax, or semantic equivalence classes as given by a Lawvere theory; and (3) a semantic domain for programs. Syntax can be constructed by initial algebras for suitable polynomial functors. Effect structure will be described by models of a countable Lawvere theory (Definition 2); we may build effect-trees, as given by *free* models of the theory, if the category is locally countably presentable (l.c.p.) [17]. Moreover, to manipulate effect-trees (Definition 7), we need monadic strength which we will obtain using ($\otimes$-)monoidal-closure of the category, and assume $\otimes$ distributes over coproducts $+$: i.e. that $[\mathsf{inl} \otimes \mathsf{id}, \mathsf{inr} \otimes \mathsf{id}]$ is a natural isomorphism $A{\otimes}C{+}B{\otimes}C \to (A{+}B){\otimes}C$, with an inverse we call $\mathsf{dist}_{A,B,C}$. Finally, the least-fixpoint construction of a semantic domain requires the category to be order-enriched [8], with left and right-strict composition; we assume $\mathsf{Cpo}_!$-enrichment (see below), so that denotations may be canonically assigned to programs by a final coalgebra morphism in a Kleisli category (Proposition 4).

The structure we need is exemplified by the category $\mathsf{Cpo}_!$ of ω-complete pointed partial-orders, with strict ω-continuous maps. It is l.c.p. as it is essentially algebraic (see [3] p.163). Its closed monoidal structure $\otimes$ is the smash product $A \otimes B$, with strict function space $A \to_\perp B$ as exponential B^A; the cartesian product $A \times B$ is pointwise ordered, and coproducts are coalesced sums $A + B$. Lastly, it is enriched over itself, with strict composition. Most of the other requirements are met because we are assured the existence of *initial algebras and final coalgebras* for locally continuous functors

F.[1] These include polynomial functors incorporating constants, $+$ and $\times$, and what we call '$\otimes$-polynomial' functors, where $\otimes$ replaces $\times$.

Syntax. We may represent syntax constructors for a programming language in terms of a $\otimes$-polynomial functor Σ. The functor mapping $X \mapsto X \otimes \cdots \otimes X$ constructs a collection of n-tuples over X; coproducts of such functors combine these collections. For convenience, given a set S, we write $S \cdot A$ for the S-fold coproduct of A.

Example 1. To represent the syntax of the first line of Definition 1, we would take $\Sigma X = L_\perp + \mathbb{N}_\perp + X \otimes X + \mathbb{N} \cdot X + \mathbb{B}_\perp + X \otimes X$. Its elements comprise: constants x and n drawn from the flat cpos $L_\perp$ and $\mathbb{N}_\perp$; pairs (x_1, x_2) in $X \otimes X$ representing $x_1 + x_2$; elements (n, x_1) of $\mathbb{N} \cdot X$ representing $+_n(x_1)$; and so on.

For a syntax functor Σ, we write TX for the *free Σ-algebra over X* (equivalently, the initial $(X + \Sigma)$-algebra), with structure $\psi_X : \Sigma TX \to TX$. Assuming the category $\mathcal{C}$ is suitably concrete, we may consider the 'elements' of TX as individual syntax terms, as we did in Section 2.1. Above, the use of a $\otimes$-polynomial functor Σ is motivated by the fact that it constructs *finite* syntax terms in $\mathsf{Cpo}_!$; ordinary polynomial functors generate countably-deep syntax.

The closed program terms of the language are given by $T0$ where 0 is the initial object. We recall that the *free Σ-algebra functor* T is in fact a monad [22].

Effects and Comodels. An equational theory of effects may be encapsulated by a countable Lawvere theory $\mathcal{L}$ [17], in which the objects n represent n-tuples and n-ary effects become arrows $e : n \to 1$; composing and tupling these gives arrows $n \to m$.

Definition 2. *Let $\aleph_1$ be a skeleton of the category of countable sets. (See [5] Definition 1). A* (countable) Lawvere theory *is a category $\mathcal{L}$ with countable products and a countable strictly product-preserving, identity-on-objects functor $F : \aleph_1^{op} \to \mathcal{L}$. If $\mathcal{C}$ has countable products, the* category of models $\mathsf{Mod}(L, C)$ *of $\mathcal{L}$ in $\mathcal{C}$ has as objects all countable product-preserving functors $L \to C$, and as arrows all natural transformations between them. The category of* comodels *has as objects the countable* coproduct-*preserving functors $L^{op} \to C$, with arrows given by natural transformations.*

A model of a theory is a functor $G : \mathcal{L} \to \mathcal{C}$ with carrier $G1$; an n-ary effect e induces a corresponding function $(G1)^n \cong Gn \overset{G(e)}{\to} G1$, where the isomorphism is by product-preservation. (Following [5], our 'models' are up to such isomorphisms.) Similarly, comodels $C : \mathcal{L}^{op} \to \mathcal{C}$ have carrier $C1$, but the effect e now corresponds to a comodel-transition function $C1 \overset{C(e^{op})}{\to} Cn \cong n \cdot (C1)$ which, given a state in $C1$, 'chooses' a branch $\{1, \ldots, n\}$ of the effect, and returns a new state.

Example 2. The standard notion of state for While programs, $\mathbb{N}^L$, is the carrier $C1$ of a comodel (in fact, the final comodel) for global store; see [19] for details.

Given a set $\mathbb{E}$ of 'effects', arrows $e : n \to 1$, one may define a corresponding polynomial syntax functor Δ. Then the countably-deep syntactic effect-trees over X – which

[1] This is because it is *algebraically ω-compact*, being cocomplete (by local presentability) and $\mathsf{Cpo}_!$-enriched [2].

we have notated $\delta((x_i)_{i\in I})$ or $\delta(\tilde{x})$ – are generated by the *free-Δ-algebra monad*, which we will call T_{e}.

However, one often wishes to impose equations on this effect syntax $T_{\mathrm{e}}X$, obtaining equivalence classes. This amounts to seeking the free model of the Lawvere theory over X, which we denote $N_{\mathrm{e}}X$. It is given by UFX, where F is left adjoint to the forgetful functor $U : \mathsf{Mod}(L, \mathcal{C}) \to \mathcal{C}$. The left adjoint exists as $\mathcal{C}$ is l.c.p. [17].

By giving $N_{\mathrm{e}}X$ a natural Δ-algebra structure (via F), we obtain a Δ-algebra morphism $\mathsf{quot}_X : T_{\mathrm{e}}X \to N_{\mathrm{e}}X$ which performs this quotienting. We may prove:

Proposition 1. *The maps* $\mathsf{quot}_X : T_{\mathrm{e}}X \to N_{\mathrm{e}}X$ *define a monad morphism.*

To ensure existence of our semantic domain for programs (Proposition 4), we must ensure the monads T_{e}, N_{e} are $\mathsf{Cpo}_!$-monads. This rules out nullary effects $e : 0 \to 1$ like exceptions, and indirectly enforces effect equations $e(\bot, \ldots, \bot) = \bot$. We may prove directly that T_{e} is a $\mathsf{Cpo}_!$-monad, as effect-syntax Δ now cannot have constants $(-) + A$. To show N_{e} is a $\mathsf{Cpo}_!$-monad, one may consider the enriched [5] or discrete [6] $\mathsf{Cpo}_!$-Lawvere theories freely generated by $\mathcal{L}$, and use the results in [6] as follows. By assumption, $\mathcal{C}$ is l.c.p., so by Theorems 14 and 15 of [6], for either freely generated theory the forgetful $\mathsf{Cpo}_!$-functor $\mathsf{Mod}(L', \mathcal{C}) \to \mathcal{C}$ has a left adjoint which induces a $\mathsf{Cpo}_!$-monad N'_{e} whose underlying, ordinary monad coincides with N_{e}.

$\mathsf{Cpo}_!$-enrichment also equips a monad M with a *monadic strength* with respect to the monoidal structure $\otimes$ – a natural transformation $\mathsf{st}_{X,Y} : X \otimes MY \to M(X \otimes Y)$ satisfying certain coherence conditions (see [12] Definition 3.2 and Remark 3.3).

Transition Systems. The transition systems of Section 2.1 are equivalent to *coalgebras* for suitable endofunctors. First, we define a primitive transition functor, $BX = V + X$, describing atomic transition steps $x \to x'$ or termination $x \to \underline{v}$.

Given a comodel state-space $C1$ and values V, we may consider a comodel-based transition system (CTS) as a function $(P \otimes C1) \to ((V + P) \otimes C1)$. By $\otimes$-closedness, this is equivalent to a function $P \to (BP \otimes C1)^{C1}$, where $BP = V + P$. By defining a monad $N_{\mathrm{c}}X = (X \otimes C1)^{C1}$ (essentially the side-effect monad), the CTS becomes a function $P \to N_{\mathrm{c}}BP$, i.e. an $N_{\mathrm{c}}B$-coalgebra.

Using B, given a set of effects E from a Lawvere theory $\mathcal{L}$, we may also express a syntactic effect-based transition system (ETS) as a $T_{\mathrm{e}}B$-coalgebra, and a semantic ETS as an $N_{\mathrm{e}}B$-coalgebra. We may quotient a syntactic ETS into a semantic ETS by post-composing with the monad morphism quot.

Finally, we may represent a syntactic or semantic CETS by an arrow $(P \otimes C1) \to M((V + P) \otimes C1)$ where $M = T_{\mathrm{e}}$ or N_{e} respectively. One may consider a CETS as combining effects from two Lawvere theories via their tensor $\mathcal{L}_1 \otimes \mathcal{L}_2$ [6], where the effects E come from $\mathcal{L}_1$, and the comodel $C1$ is for $\mathcal{L}_2$.Either way, defining monads $T_{\mathrm{ce}}X := (T_{\mathrm{e}}(X \otimes C1))^{C1}$ and $N_{\mathrm{ce}}X := (N_{\mathrm{e}}(X \otimes C1))^{C1}$, we may express a CETS as a $T_{\mathrm{ce}}B$ or $N_{\mathrm{ce}}B$-coalgebra.

3 Three Evaluation-in-Context Rule Formats

Having described operational models as coalgebraic transition systems, we give concrete presentations of rule formats for specifying these operational models, which will

give rise to compositional and adequate semantics.[2] The formats are based on the Evaluation-In-Context paradigm for sequential languages (c.f. [7]).

Here are some of the (standard) operational rules for While, considered as a CTS.

$$\frac{\langle p,s\rangle \to \langle p',s'\rangle}{\langle p\,;q,s\rangle \to \langle p'\,;q,s'\rangle} \qquad \frac{\langle p,s\rangle \to \langle \underline{*},s'\rangle}{\langle p\,;q,s\rangle \to \langle q,s'\rangle} \qquad \frac{\langle u,s\rangle \to \langle u',s'\rangle}{\langle x{=}u,s\rangle \to \langle x{=}u',s'\rangle} \qquad \frac{\langle u,s\rangle \to \langle \underline{n},s'\rangle}{\langle x{=}u,s\rangle \to \langle \underline{*},s'[x\mapsto n]\rangle}$$

$$\overline{\langle \texttt{while}\,(e)\,\texttt{do}\,\{p\},s\rangle \to \langle \texttt{if}\,(e)\,\texttt{then}\,\{p;\texttt{while}\,(e)\,\texttt{do}\,\{p\}\}\,\texttt{else}\,\{\texttt{skip}\},s\rangle}$$

These rules divide the syntax constructors into what we call *context* and *redex* terms. Examples of the former are addition operators $+, +_n$, `if` statements, sequential composition ; and assignments $x{=}u$ (see below). To evaluate them, we must evaluate a distinguished argument; when it terminates with some value, we may have to evaluate another term, or produce another terminal value.

By contrast, a redex term evaluates independently of how its arguments behave. This includes: elementary terms $n \in \mathbb{N}, b \in \mathbb{B}$, `skip` which terminate as $\underline{n}$, $\underline{b}$, and $\underline{*}$; variable lookups $x \in L$, returning the value $\underline{s(x)}$ of the store at x; and `while` statements. This generalises to our first rule-format for specifying operational models as CTS's.

Definition 3. *Given a countable set of syntax variables P, the first kind of* Evaluation-In-Context specification (EIC1) *for values V and comodel C consists of the following, where for each rule below we assume $\tilde{x} = (x_i)_{i\in I}$ and $\tilde{y}_\bullet = (y_j)_{j\in J_\bullet}$ are such that $\{x_i : i \in I\} \subseteq P$ are pairwise distinct and disjoint from $\{x, x'\} \subseteq P$; and $t_\bullet(\tilde{y}_\bullet)$ stands for a terminal value $\underline{v}$ or a syntax term, where $\{y_j : j \in J_\bullet\} \subseteq \{x_i : i \in I\}$.*
– For every context-term constructor σ, the left-hand rule (CTXL) *below, and an instance of the right-hand rule* (CTXR) *for every $v \in V$ and comodel state $c \in C$1, with corresponding terminal value or term $t_{v,c}(\tilde{y}_{v,c})$ and new comodel state $c'_{v,c}$;*

$$\frac{\langle x,s\rangle \to \langle x',s'\rangle}{\langle \sigma(x,\tilde{x}),s\rangle \to \langle \sigma(x',\tilde{x}),s'\rangle}\ (\mathsf{CTXL}) \qquad \frac{\langle x,s\rangle \to \langle \underline{v},c\rangle}{\langle \sigma(x,\tilde{x}),s\rangle \to \langle t_{v,c}(\tilde{y}_{v,c}),c'_{v,c}\rangle}\ (\mathsf{CTXR})$$

– For redex constructors ρ, a rule $\overline{\langle \rho(\tilde{x}),c\rangle \to \langle t_c(\tilde{y}_c),c'_c\rangle}$ (REDX) *for each comodel state c, with terminal value or term $t_c(\tilde{y}_c)$ and new comodel state c'_c.*

Example 3. Consider the four operational rules for While given earlier (with syntax variables $P = \{p, q, u, e\}$). The first rule for sequential composition $\sigma(x, \tilde{x}) = p\,;q$ is merely (CTXL) and the other is an instance of (CTXR) where $v = *$ and $t_{v,c}(\tilde{x}_{v,c}) = q$. The first rule for variable update $x{=}u$ is again (CTXL), and the second is (CTXR) where v is n, $t_{v,c}(x_{j\in J})$ is $\underline{*}$ and $c' = c[x \mapsto n]$ for every $n \in \mathbb{N}$.

This approach also permits us to specify languages combining effects and comodels. Below are the rules for branching `choose` and assignments $x{=}u$ in NDWhile, where

[2] For simplicity, order-theoretic details are omitted; the important point is that divergence $p \to \bot$ is not treated as an ordinary terminal value, but rather propagated in the natural manner.

$\delta(\ldots)$ stands for an arbitrary **or**-tree of pairs $\langle b_k, c_k\rangle$ in which b_k is either a terminal value $\underline{v}$ or a program state u', and c_k a comodel-state; the general format follows.

$$\frac{}{\langle \mathtt{choose}(x,y),s\rangle \to \mathbf{or}(\langle x,s\rangle,\langle y,s\rangle)} \qquad \frac{\langle u,s\rangle \to \delta(\langle b_k,c_k\rangle_{k\in K})}{\langle x{=}u,s\rangle \to \delta\left(\left\{\begin{array}{ll}\langle x{=}u', c_k\rangle & \text{if } b_k{=}u'\\ \langle \underline{*}, c_k[x\mapsto n]\rangle & \text{if } b_k{=}\underline{n}\end{array}\right\}_{k\in K}\right)}$$

Definition 4. *Analogously to Definition 3, an* Evaluation-In-Context 2 (EIC2) Specification *for values V, comodel C, and a collection $\mathbb{E}$ of syntactic effects, consists of:*

– For redex constructors ρ, rules $\overline{\langle \rho(\tilde{x}), c\rangle \to \epsilon_c(\langle t_l(\tilde{y}_l), c'_l\rangle_{l\in L_c})}$ (REDX) for all comodel states c, with syntactic effect-trees ϵ_c whose L_c-indexed leaves $\langle t_l(\tilde{y}_l), c'_l\rangle$ are pairs of a terminal value or term, and a new comodel state.
– For every context-term constructor σ, a rule (CTXB) *for every effect-tree δ with leaves $\{\langle b_k, c_k\rangle : k \in K\}$ given by pairs $\langle b_k, c_k\rangle$ of either a syntax variable x_k or a terminal value $\underline{v}_k$, and a comodel-state c_k. We assume these x_k are all distinct, and do not include x or x_i for $i \in I$. Each rule is given by a $V \otimes C1$-indexed collection of effect-trees $\epsilon_{v,c}$ with $L_{v,c}$-indexed leaves $\langle t_l(\tilde{y}_l), c'_l\rangle$ as above.*

$$\frac{\langle x,s\rangle \to \delta(\langle b_k,c_k\rangle_{k\in K})}{\langle \sigma(x,\tilde{x}),s\rangle \to \delta\left(\left\{\begin{array}{ll}\langle \sigma(x_k,\tilde{x}), c_k\rangle & \text{if } b_k = x_k\\ \epsilon_{v,c_k}(\langle t_l(\tilde{y}_l), c'_l\rangle_{l\in L_{v,c_k}}) & \text{if } b_k = \underline{v}\end{array}\right\}_{k\in K}\right)} \text{(CTXB)}$$

Finally, by removing all mention of comodels from the above format, we gain a rule format for specifying ETS's. We could specify an ETS for While with rules such as the following; the general rule format is given below.

$$\frac{}{x \to \mathbf{rd}_x(\underline{0},\underline{1},\underline{2},\ldots)} \qquad \frac{u \to \delta((b_k)_{k\in K})}{x{=}u \to \delta\left(\left\{\begin{array}{ll}x{=}u' & \text{if } b_k = u'\\ \mathbf{wr}_{x,n}(\underline{*}) & \text{if } b_k = \underline{n}\end{array}\right\}_{k\in K}\right)}$$

Definition 5. Evaluation-In-Context 3 (EIC3) *is analogous to EIC2, but with rules*

$$\frac{}{\rho(\tilde{x}) \to \epsilon((t_l(\tilde{y}_l))_{l\in L})} \qquad \frac{x \to \delta(b_{k\in K})}{\sigma(x,\tilde{x}) \to \delta\left(\left\{\begin{array}{ll}\sigma(x_k,\tilde{x}) & \text{if } b_k = x_k\\ \epsilon_v((t_l(\tilde{y}_l))_{l\in L_v}) & \text{if } b_k = \underline{v}\end{array}\right\}_{k\in K}\right)}$$

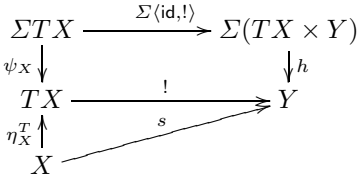
These three kinds of EIC rule format allows us to specify operational models as CTS's or syntactic (C)ETS's. As formalised below, structural recursion then defines transition behaviour for program terms TX over syntax variables X, once we have specified transition behaviour for the variables X – the 'base cases' of the recursion.

3.1 From EIC Specifications to Operational Models

We now formalise the specifications of the previous section as natural transformations, and show how they induce coalgebraic operational models by structural recursion.

There are various ways of expressing operational specifications as distributive laws of syntax over behaviour [11]. For our purposes, the 'abstract GSOS' specifications of [24] suffice: natural transformations $\epsilon : \Sigma(Id \times B) \Rightarrow BT$, where Σ is the program syntax functor, T the free Σ-algebra monad, and B a coalgebraic behaviour functor. These specifications induce operational models $T0 \to BT0$ by structural recursion:

Proposition 2. *[24] Given an arrow $h : \Sigma(TX \times Y) \to Y$ and an arrow $s : X \to Y$, there is a unique arrow $! : TX \to Y$ making the below diagram commute.*



Given an operational specification $\epsilon : \Sigma(Id \times B) \Rightarrow BT$ and a transition structure $\gamma : X \to BX$, we can derive a transition structure $T^\epsilon(\gamma) : TX \to BTX$ for terms TX over generators X as follows. Defining $\epsilon' := B\mu^T \circ \epsilon_T : \Sigma(T \times BT) \Rightarrow BT$ [24], we apply the result with $Y = BTX$, $s = B\eta^T \circ \gamma : X \to BTX$, and $h = \epsilon'_X$. The resulting map $! : TX \to BTX$ is the required transition structure on terms TX. In particular, taking $X = 0$ and the unique arrow $\gamma : 0 \to B0$, we obtain an operational model for closed program terms, an arrow $T0 \to BT0$ which we call $T^\epsilon(0)$.

The operational models considered in this paper are MB-coalgebras for various monads M, where $BX = V + X$. Replacing B above with MB, operational specifications ϵ of type $\Sigma(X \times MBX) \to MBTX$ induce operational models of form $T^\epsilon(0) : T0 \to MBT0$.

It remains to encode EIC specifications as natural transformations ϵ. Example 9 will demonstrate that non-EIC specifications ϵ may result in semantics which are not adequate or compositional; however, EIC-specified languages will be shown to have these properties. First, we express redex term constructors by a syntax functor R, and context terms by $X \otimes HX$, with active argument X and context HX.

Definition 6. *In a symmetric $\otimes$-monoidal category $\mathcal{C}$ with coproducts, an endofunctor Σ is said to be* Redex-Context (R-C) *if $\Sigma X = RX + X \otimes HX$ for some functors R, H.*

Formalising the data of EIC specifications, we represent redex-terms $\rho(\tilde{x})$ over X by RX, context terms $\sigma(x, \tilde{x})$ by $X \otimes HX$, and arbitrary terms by TX. A 'terminal value or term $t(\tilde{x})$' is a basic transition over terms in BTX. Syntactic effect-trees $\delta(\tilde{x})$ over X are given by the free Δ-algebra monad: $T_{\mathrm{e}}X$. For EIC 1.0 (i.e. CTS), the rules (REDX) combine into a natural transformation $\alpha_X : (RX \otimes C1) \to (BTX \otimes C1)$, indicating, for each redex RX and initial comodel-state $C1$, the transition behaviour BTX and new comodel-state $C1$. Similarly, (CTXR) gives a natural transformation $\beta_X : (V \otimes HX \otimes C1) \to (BTX \otimes C1)$.

Example 4. Consider the fragment of While given by variable lookups l in L (so $RX = L_\perp$) and updates $l = x$ (context terms $l = (-)$ with context-data HX given by a location, so that $HX = L_\perp$). These commands are specified by $\alpha_X : (l, s) \mapsto (\mathsf{inl}(\underline{s(l)}), s)$

and $\beta_X : (n, l, s) \mapsto (\mathsf{inl}(\underline{*}), s[l \mapsto n])$ where we underline the 'terminal values', left components of the coproduct $BTX = V + TX$.

For EIC 2.0 (i.e. CETS), the rules (REDX) give a natural transformation $\alpha_X : (RX \otimes C1) \to T_{\mathrm{e}}(BTX \otimes C1)$ – where the codomain describes syntactic effect-trees of transition behaviours. Similarly, (CTXR) gives $\beta_X : (V \otimes HX \otimes C1) \to T_{\mathrm{e}}(BTX \otimes C1)$.

Example 5. Variable lookups and updates in NDWhile are specified using the unit $\eta^{T_{\mathrm{e}}}$ of the effect-syntax monad T_{e} (i.e. 'no non-determinism'): we define $\alpha_X : (l, s) \mapsto \eta^{T_{\mathrm{e}}}(s(l), s)$ and $\beta_X : (n, l, s) \mapsto \eta^{T_{\mathrm{e}}}(\underline{*}, s[l \mapsto n])$. The choice operator $\mathtt{choose}(x, y)$ ($RX = X \otimes X$) is given by $\alpha_X : ((x, y), s) \mapsto \mathbf{or}((\mathsf{inr}(x), s), (\mathsf{inr}(y), s))$ where the inr are 'new program terms', right-components of the coproduct $BTX = V + TX$.

Finally, for EIC 3.0, the rules (REDX) give a natural transformation $\alpha_X : RX \to T_{\mathrm{e}}BTX$ and (CTXR) gives $\beta_X : (V \otimes HX) \to T_{\mathrm{e}}BTX$.

Example 6. Considering While as a syntactic ETS, variable lookups would be specified by $\alpha_X : l \mapsto \mathbf{rd}_l(\mathsf{inl}(\underline{0}), \mathsf{inl}(\underline{1}), \ldots)$ and updates by $\beta_X : (n, l) \mapsto \mathbf{wr}_{l,n}(\mathsf{inl}(\underline{*}))$.

We may now use the $\otimes$-monoidal closed structure to show that in all cases (REDX) corresponds to natural transformations $r_X : RX \to MBTX$ where $M = N_{\mathrm{c}}, T_{\mathrm{e}}$ and T_{ce} respectively. Similarly, (CTXR) corresponds to $e_X : X \otimes HX \to MBTX$. This data induces an operational specification $\epsilon_X : \Sigma(X \times MBX) \to MBTX$ as follows:

Definition 7. *Let Σ be an R-C syntax functor, M a $\otimes$-strong monad with costrength* cost*, and B a behaviour functor $BX = V + X$. For given natural transformations $r_X : RX \to MBTX$ and $e_X : V \otimes HX \to MBTX$, the corresponding* abstract EIC specification *$\epsilon_X : \Sigma(TX \times MBTX) \to MBTX$ is given by*

$$\epsilon_X : R(TX \times MBTX) + (H \otimes Id)(TX \times MBTX) \xrightarrow{[\mathsf{aosr}_X, \mathsf{aosc}_X]} MBTX$$

where aosr *and* aosc *are defined below (we have abbreviated* $\mathsf{cost}_{BTX,HTX}$*).*

$$\begin{aligned}
\mathsf{aosr}_X : \quad & R(TX \times MBTX) \xrightarrow{R\pi_1} RTX \xrightarrow{r_{TX}} MBTTX \xrightarrow{MB\mu_X} MBTX \\
\mathsf{aosc}_X : \quad & (Id \times H)(TX \times MBTX) \xrightarrow{\pi_2 \otimes H\pi_1} MBTX \otimes HTX \\
& \xrightarrow{\mathsf{cost}} M(BTX \otimes HTX) \xrightarrow{M\mathsf{dwc}_X} M^2BTX \xrightarrow{\mu_{BTX}} MBTX
\end{aligned}$$

Here, dwc *('deal with contexts') is defined as follows, with sub-cases handled by* $\mathsf{dwc}^{(\mathsf{v})}$ *('values') and* $\mathsf{dwc}^{(\mathsf{b})}$ *('non-terminal behaviour'). We abbreviate* $\mathsf{dist}_{V,TX,HTX}$*. Recall that $\psi_X : \Sigma TX \to TX$ is the Σ-algebra structure of TX, the free Σ-algebra over X.*

$$\begin{aligned}
\mathsf{dwc}_X &: (V+TX)\otimes HTX \xrightarrow{\mathsf{dist}} V\otimes HTX + TX\otimes HTX \xrightarrow{[\mathsf{dwc}^{(\mathsf{v})}{}_X, \mathsf{dwc}^{(\mathsf{b})}{}_X]} MBTX \\
\mathsf{dwc}^{(\mathsf{v})}{}_X &: V \otimes HTX \xrightarrow{e_X} MBTTX \xrightarrow{MB\mu_X} MBTX \\
\mathsf{dwc}^{(\mathsf{b})}{}_X &: TX \otimes HTX \xrightarrow{\mathsf{inr}} \Sigma TX \xrightarrow{\psi_X} TX \xrightarrow{\mathsf{inr}} BTX \xrightarrow{\eta_{BTX}} MBTX
\end{aligned}$$

The rule (REDX) is described by aosr. For context terms $TX \otimes HTX$, given the behaviour $MBTX$ of the active term TX (isolated by the first line of aosc), the costrength attaches the context HTX to each computation branch. The map dwc decides what to do for each computation branch; if a branch terminates, it is handled by $\mathsf{dwc}^{(\mathsf{v})}$ which corresponds to (CTXR); otherwise, it is handled by $\mathsf{dwc}^{(\mathsf{b})}$, corresponding to (CTXL).

4 Behavioural Equivalence in a Kleisli Category

Following Turi and Plotkin's method, we would take the semantic domain to be the final MB-coalgebra, where $BX = V + X$ and M is the monad for the transition systems under consideration; however, this distinguishes comodel-manipulations and effects at every execution step. For instance, the (CTS or ETS) While programs $x{=}0\,;x{=}1$ and $x{=}2\,;x{=}1$ would be considered to have different behaviour.

A natural solution to this problem is to move to a Kleisli category for the monad M, and construe transition systems as $\overline{B}$-coalgebras, where $\overline{B}$ is a lifting of B [1]. In fact, MB-coalgebras $X \to MBX$ coincide with $\overline{B}$-coalgebras $X \to \overline{B}X$; but $\overline{B}$-coalgebra morphisms, and the final $\overline{B}$-coalgebra, which we write $\langle \overline{D}, \overline{s} : \overline{D} \to \overline{BD}\rangle$, are generally different. For any ($\overline{B}$ or) MB-coalgebra $\gamma : X \to MBX$, there is a unique $\overline{B}$-coalgebra morphism $\beta : X \to \overline{D}$, of underlying type $X \to M\overline{D}$.

Thus, if a lifting $\overline{B}$ and the final $\overline{B}$-coalgebra $\overline{D}$ exist, we obtain a map into $M\overline{D}$, which we will see gives a more appropriate characterisation of program behaviour. The first point is easily addressed: liftings $\overline{B}$ of B are in 1-1 correspondence with distributive laws $\lambda : BM \Rightarrow MB$ (see e.g. [20,4]).

Remark 1. For the functor $BX = V+X$ and any monad M, the natural transformation $\lambda_X := [\eta^M \circ \mathsf{inl}, M\mathsf{inr}] : V+MX \to M(V+X)$ is a distributive law of B over M.

As for existence of the final $\overline{B}$-coalgebra, we draw on the following result (quoted and proved in [4] as Proposition 3.9):

Proposition 3. *Let $\mathcal{D}$ be a $\mathsf{Cpo}_!$-enriched category with left-strict composition, and G a locally-continuous endofunctor on $\mathcal{D}$. An initial G-algebra $\theta : G\overline{D} \cong \overline{D}$, if it exists, yields a final G-coalgebra $\theta^{-1} : \overline{D} \cong G\overline{D}$. Given a G-coalgebra $\gamma : X \to GX$, the corresponding unique G-coalgebra morphism $\beta : X \to \overline{D}$ is the least fixpoint of the operator $\Phi : (X \xrightarrow{f} \overline{D}) \mapsto (X \xrightarrow{\gamma} GX \xrightarrow{Gf} G\overline{D} \xrightarrow{\theta} \overline{D})$ – i.e. β is the join of the approximants $\beta^{(n)} := \Phi^n(\bot_{X,\overline{D}})$ for $n < \omega$.*

We apply this result to the Kleisli category $\mathcal{D} = \mathsf{Kl}(M)$ and $G = \overline{B}$, a lifting of B; this will give the required final $\overline{B}$-coalgebra $\overline{D}$. The $\mathsf{Cpo}_!$-enrichedness of $\mathsf{Kl}(M)$ follows from that of the underlying category $\mathcal{C}$; if B is locally continuous, it is easy to show $\overline{B}$ must also be. If B has an an initial algebra $\alpha : B\overline{D} \to \overline{D}$, we may show $\overline{D}$ is also an initial algebra for $\overline{B}$, with structure $\eta^M \circ \alpha$ ([4] Proposition 3.2). Finally, if M is a $\mathsf{Cpo}_!$-monad, one may show $M\bot_{A,B} = \bot_{MA,MB}$; if composition in $\mathcal{C}$ is both left and right-strict, this implies left-strictness of Kleisli-composition. In summary:

Proposition 4. *Let $\mathcal{C}$ be a $\mathsf{Cpo}_!$-enriched category with strict composition, on which M is a $\mathsf{Cpo}_!$-enriched monad, and B a locally continuous functor with a lifting $\overline{B}$ to $\mathsf{Kl}(M)$. If B has an initial algebra $\alpha : B\overline{D} \cong \overline{D}$, the final $\overline{B}$-coalgebra is given by $\eta^M \circ \alpha^{-1} : \overline{D} \to \overline{BD}$.*

Example 7. We illustrate how the fixpoint construction of Proposition 4 assigns denotations to While programs. Taking $BX = V+X$, the initial B-algebra $\overline{D}$ has carrier $\mathbb{N}\cdot V$, whose elements $(n, \underline{v})$ characterise individual computation branches by their length n and return-value v. Its algebra-structure $\alpha : B\overline{D} \to \overline{D}$ is defined by $\alpha(\mathsf{inl}(\underline{v})) = (1, \underline{v})$

and $\alpha(\mathsf{inr}(n, \underline{v})) = (n+1, \underline{v})$. The functor B has a lifting $\overline{B}$ given by the distributive law $\lambda : BM \Rightarrow MB$ of Remark 1.

If we work in $\mathsf{Cpo}_!$ and take M to be a $\mathsf{Cpo}_!$-monad, then Proposition 4 applies. Thus, $\overline{D}$ is also the carrier of the final $\overline{B}$-coalgebra. For any operational model $T^\epsilon(0) : T0 \to MBT0$ (construed as a $\overline{B}$-coalgebra), there will be a $\overline{B}$-coalgebra morphism β from $T0$ into the semantic domain $\overline{D}$, of underlying type $T0 \to M\overline{D}$. It is given by the join of the approximants $\beta^{(n)}$ which may be defined as follows: $\beta^{(0)} = \bot_{T0, M\overline{D}} : T0 \to M\overline{D}$ (i.e. the bottom arrow $T0 \to \overline{D}$ in $\mathsf{Kl}(M)$); and then

$$\beta^{(n+1)} : T0 \xrightarrow{T^\epsilon(0)} MBT0 \xrightarrow{MB\beta^{(n)}} MBM\overline{D} \xrightarrow{M\lambda} M^2B\overline{D} \xrightarrow{\mu} MB\overline{D} \xrightarrow{M\alpha} M\overline{D}$$

We now instantiate these results for While as a CTS, taking $M = N_\mathrm{c}$ with the comodel of Example 2. This gives a semantic domain of $M\overline{D} = ((\mathbb{N} \cdot V) \otimes C1)^{C1}$: each program p is assigned a function which, given an initial comodel state s, tells us about the execution of the program p with initial comodel-state s: a pair (n, v) of the number of steps-to-termination n and the return-value v, as well as the final comodel state s'. (Non-terminating pairs (p, s) receive the value $\bot$.)

Suppose we have obtained an (un-curried) operational model $T^\epsilon(0) : T0 \to (BT0 \otimes C1)^{C1}$ for While as a CTS, as in Section 3.1 (see Example 4). We illustrate the action of the approximants $\beta^{(n)}$ on the programs $p_n := (x = n)$ and $q := (x = 5; x = 8)$. (We abbreviate the series of maps $(\overline{B}\beta^{(0)})^\dagger = \mu^{N_\mathrm{c}} \circ N_\mathrm{c}\lambda \circ N_\mathrm{c}B\beta^{(0)}$, which have no effect on the terminated value $\mathsf{inl}(\underline{*})$.)

$$\begin{aligned}\beta^{(1)} : x = n \xrightarrow{T^\epsilon(0)} \lambda s.(\mathsf{inl}(\underline{*}), s[x \mapsto n]) &\xrightarrow{(\overline{B}\beta^{(0)})^\dagger} \lambda s.(\mathsf{inl}(\underline{*}), s[x \mapsto n]) \\ &\xrightarrow{N_\mathrm{c}\alpha} \lambda s.((1, \underline{*}), s[x \mapsto n])\end{aligned}$$

This yields the desired denotation of the assignment $x = n$. Note that higher approximants $\beta^{(n)}$ will assign the same value. We may now show that $\beta^{(2)}$ assigns the desired denotation to the program $x = 5; x = 8$:

$$\begin{aligned}\beta^{(2)} : (x = 5; x = 8) &\xrightarrow{T^\epsilon(0)} \lambda s.(\mathsf{inr}(x = 8), s[x \mapsto 5]) \\ &\xrightarrow{N_\mathrm{c}B\beta^{(1)}} \lambda s.(\mathsf{inr}(\lambda s'.((1, \underline{*}), s'[x \mapsto 8])), s[x \mapsto 5]) \\ &\xrightarrow{N_\mathrm{c}\lambda} \lambda s.((\lambda s'.(\mathsf{inr}(1, \underline{*}), s'[x \mapsto 8])), s[x \mapsto 5]) \\ &\xrightarrow{\mu^{N_\mathrm{c}}} \lambda s.(\mathsf{inr}(1, \underline{*}), s[x \mapsto 8]) \xrightarrow{N_\mathrm{c}\alpha} \lambda s.((2, \underline{*}), s[x \mapsto 8])\end{aligned}$$

This illustrates that in a CTS, the map β identifies two programs p, q if and only if: for every initial comodel-state s, $\langle p, s\rangle$ and $\langle q, s\rangle$ both: (a) terminate with the same final comodel-state s' and terminal value $\underline{v}$ in the same number of steps n; or (b) do not terminate. One may check that in a syntactic ETS, the map β identifies two programs p, q if their executions produce the same effect-tree $\delta((n_i, \underline{v}_i)_{i \in I})$ of terminal values $\underline{v}_i$ paired with the number of steps n_i they took to appear; and the situation for CETS's combines features of both CTS's and ETS's. On this basis, we may take the final coalgebra maps β as a characterisation of behavioural equivalence. However, the EIC specifications induce *syntactic* ETS's and CETS's; the corresponding maps $\beta : T0 \to T_\mathrm{e}\overline{D}$, $\beta : T0 \to T_{\mathrm{ce}}\overline{D}$ distinguish semantically-equivalent effect-trees. An appropriate behavioural equivalence must quotient by the equations on the effects:

Definition 8. *[Kleisli Behavioural Equivalence] Under the assumptions of Proposition 3, two states p, q of a* CTS *satisfy $p \cong_{\mathrm{c}} q$ if they are identified by the final $\overline{B}$-coalgebra morphism β into the final $\overline{B}$-coalgebra $\overline{D}$ in $\mathsf{Kl}(N_{\mathrm{c}})$. For the appropriate final coalgebra maps β in $\mathsf{Kl}(T_{\mathrm{e}})$ or $\mathsf{Kl}(T_{\mathrm{ce}})$, states p, q of a* syntactic ETS *satisfy $p \cong^{\mathrm{N}}_{\mathrm{e}} q$, if they are identified by $\mathsf{quot}_{\overline{D}} \circ \beta : T0 \to N_{\mathrm{e}}\overline{D}$; and states p, q of a* syntactic CETS *satisfy $p \cong^{\mathrm{N}}_{\mathrm{ce}} q$ if they are identified by $(\mathsf{quot}_{\overline{D} \otimes C1})^{C1} \circ \beta : T0 \to N_{\mathrm{ce}}\overline{D}$.*

Example 8. Suppose we construe While as a CTS, and NDWhile as a CETS. Define

$$p_1 := (x{=}0\,;x{=}1), \quad p_2 := (x{=}1\,;x{=}1), \quad p_3 := (x{=}1), \quad \text{and} \quad p_4 := \mathtt{choose}(p_3, p_3).$$

For While as a CTS, we have $\beta(p_1) = \beta(p_2) = \lambda c.((2, \underline{*}), c[x \mapsto 1])$ and $\beta(p_3) = \lambda c.((1, \underline{*}), c[x \mapsto 1])$; hence $p_1 \cong_{\mathrm{c}} p_2 \not\cong_{\mathrm{c}} p_3$. As programs of NDWhile, we will have $p_2 \not\cong^{\mathrm{T}}_{\mathrm{ce}} p_4$, due to the appearance of a syntactic **or**-effect when we evaluate p_4; however, the semantic equation $\mathbf{or}(x, x) = x$ will imply that $p_2 \cong^{\mathrm{N}}_{\mathrm{ce}} p_4$.

Example 9. Relaxing the EIC rule formats may give non-compositional syntax constructors. Considering While as a CTS, the 'one-step timeout' $p \rhd q$ executes the first step of p, and continues with q; the interleaver $p \mid q$ alternates steps of p and q.

$$\frac{\langle p, s\rangle \to \langle p', s'\rangle}{\langle p \rhd q, s\rangle \to \langle q, s'\rangle} \qquad \frac{\langle p, s\rangle \to \langle \underline{v}, s'\rangle}{\langle p \rhd q, s\rangle \to \langle q, s'\rangle} \qquad \frac{\langle p, s\rangle \to \langle p', s'\rangle}{\langle p \mid q, s\rangle \to \langle q \mid p', s'\rangle} \qquad \frac{\langle p, s\rangle \to \langle \underline{v}, s'\rangle}{\langle p \mid q, s\rangle \to \langle q, s'\rangle}$$

Letting $p_1 := (x{=}0\,;x{=}2)$, $p_2 := (x{=}1\,;x{=}2)$, and $p_3 := (y{=}x)$, we have $p_1 \cong_{\mathrm{c}} p_2$, but $(p_1 \rhd p_3) \not\cong_{\mathrm{c}} (p_2 \rhd p_3)$ and $(p_1 \mid p_3) \not\cong_{\mathrm{c}} (p_2 \mid p_3)$. Syntax constructors sensitive to individual execution steps can take 'behaviourally equivalent' arguments with different results, breaking compositionality. This cannot occur in EIC specifications.

5 Compositionality and Adequacy

We have defined operational equivalence in terms of mappings $T0 \to M\overline{D}$ into semantic domains $M\overline{D}$ for various monads M. We now define a corresponding denotational model, and prove adequacy and compositionality of the resulting denotational semantics. We do this first for CTS's, and then for syntactic (C)ETS's.

In order to treat the semantic domain $M\overline{D}$ as a denotational model, we must define interpretations $[\![\sigma]\!]$ of syntax constructors σ on denotations. First, we assign transition behaviour to denotations, by giving the semantic domain a natural MB-coalgebra structure: $\eta^M \circ M\alpha^{-1} : M\overline{D} \to MB\overline{D} \to MBM\overline{D}$.

We now take denotations d_i as base-cases for structural recursion (Proposition 2), yielding an MB- (i.e. $\overline{B}$-)coalgebra transition structure for *program terms over denotations* $TM\overline{D}$, like $\sigma((d_i)_{i\in I})$. As a result, we obtain a $\overline{B}$-coalgebra morphism ζ, of underlying type $TM\overline{D} \to M\overline{D}$, characterising the behaviour of such terms by mapping back into the semantic domain $M\overline{D}$ (by finality). This gives a Σ-algebra structure χ to the semantic domain $M\overline{D}$, interpreting the syntax constructors $[\![\sigma]\!]$ on denotations, via the following composition (where ψ_X is the free Σ-algebra structure of TX): $\Sigma M\overline{D} \xrightarrow{\Sigma\eta^T} \Sigma TM\overline{D} \xrightarrow{\psi_{M\overline{D}}} TM\overline{D} \xrightarrow{\zeta} M\overline{D}$.

By repeated application of the functions $[\![\sigma]\!]$ (with base-cases given by nullary symbols), one inductively constructs denotations $[\![t]\!]$ of arbitrary terms t. Formally, this assignment $[\![-]\!] : T0 \to M\overline{D}$ is given by the initial Σ-algebra morphism from $T0$ into $M\overline{D}$, equipped with the above structure χ. This gives a suitable denotational semantics for CTS's, where we take $M = N_{\mathrm{e}}$. That it is a Σ-algebra morphism implies *compositionality* of the denotational semantics; i.e. that the denotation $[\![\sigma((t_i)_{i\in I})]\!]$ of a term $\sigma((t_i)_{i\in I})$ can be constructed from the denotations $[\![t_i]\!]$ of its parts: $[\![\sigma((t_i)_{i\in I})]\!] = [\![\sigma]\!]([\![t_i]\!]_{i\in I})$. This makes it a congruence with respect to the syntax constructors of the language: $[\![s_i]\!] = [\![t_i]\!]$ for all i implies $[\![\sigma((s_i)_{i\in I})]\!] = [\![\sigma((t_i)_{i\in I})]\!]$.

However, it remains to show *adequacy*, i.e. that denotational equivalence implies operational equivalence for CTS's. A convenient method is to show that the denotational map $[\![-]\!]$ coincides with the map β characterising behavioural equivalence for CTS's (Definition 8), by showing that the latter is also a Σ-algebra morphism; by initiality, there can be only one such map. This is achieved through the following theorem.

Theorem 1. *Given an abstract EIC specification ϵ (Definition 7) inducing an operational model $T^{\epsilon}(0) : T0 \to MBT0$, the underlying arrow $\beta : T0 \to M\overline{D}$ of the $\overline{B}$-coalgebra morphism into the final $\overline{B}$-coalgebra $\overline{D}$ is a Σ-algebra morphism.*

The broad strategy is to factor the 'coarse-grained' denotational map β through its 'fine-grained' analogue, the final MB-coalgebra D. The proof involves manipulating colimit diagrams and limit-colimit coincidences in the Kleisli category, and a detailed inspection of the mechanics of the EIC specifications.

Now we consider denotational semantics for ETS's (CETS's are exactly analogous). Recall that we characterised behavioural equivalence not by the semantic domain $T_{\mathrm{e}}\overline{D}$, consisting of syntactic effect-trees, but by the equivalence classes $N_{\mathrm{e}}\overline{D}$ generated by applying the quotienting map $\mathsf{quot}_{\overline{D}} : T_{\mathrm{e}}\overline{D} \to N_{\mathrm{e}}\overline{D}$. To treat the domain $N_{\mathrm{e}}\overline{D}$ as a denotational model, we must give it a Σ-algebra structure.

We may follow an analogous method to the one outlined above, by moving from syntactic ETS's ($T_{\mathrm{e}}B$-coalgebras) to semantic ETS's ($N_{\mathrm{e}}B$-coalgebras) and considering rule-formats and structural recursion directly in terms of equivalence-classes of effect-trees. This amounts to seeking an abstract EIC specification (Definition 7) in terms of the monad N_{e} rather than T_{e}. Note that a syntactic EIC specification in terms of T_{e} – given by natural transformations $r : RX \Rightarrow T_{\mathrm{e}}BT$ and $e : V \otimes HX \Rightarrow T_{\mathrm{e}}BT$ – translates into a specification in terms of N_{e}, by postcomposing with $\mathsf{quot}_{BT} : T_{\mathrm{e}}BT \Rightarrow N_{\mathrm{e}}BT$. Formally, structural recursion induces an operational model $T0 \to NBT0$ directly in terms of equivalence-classes. The semantic analysis of Section 4 may also be transplanted from $\mathsf{Kl}(T_{\mathrm{e}})$ to $\mathsf{Kl}(N_{\mathrm{e}})$: again, there is a lifting $\overline{B}^{N_{\mathrm{e}}}$ of B given by Remark 1, and Proposition 4 guarantees the existence of a final $\overline{B}^{N_{\mathrm{e}}}$ coalgebra in $\mathsf{Kl}(N_{\mathrm{e}})$, whose carrier is the initial B-algebra $\overline{D}$. There is a unique $\overline{B}^{N_{\mathrm{e}}}$-coalgebra morphism $\beta^{N_{\mathrm{e}}}$ from the operational model into $\overline{D}$, now of underlying type $T0 \to N_{\mathrm{e}}\overline{D}$.

As above, structural recursion over the semantic domain $M\overline{D}$, where M is now N_{e}, induces a Σ-algebra structure $\chi^{N_{\mathrm{e}}}$ on $N_{\mathrm{e}}\overline{D}$; initiality gives a unique Σ-algebra morphism $[\![-]\!] : T0 \to N_{\mathrm{e}}\overline{D}$, i.e. a denotational semantics which is automatically *compositional*. Theorem 1 implies that $\beta^{N_{\mathrm{e}}}$ is a Σ-algebra morphism, and so $\beta^{N_{\mathrm{e}}} = [\![-]\!]$. To prove *adequacy* of this denotational semantics for ETS's, we may show that the

denotational map $[\![-]\!]$ coincides with the map $\mathsf{quot}_{\overline{D}} \circ \beta : T0 \to N_{\mathrm{e}}\overline{D}$ characterising behavioural equivalence. The following result will imply that $\mathsf{quot}_{\overline{D}} \circ \beta = \beta^{N_{\mathrm{e}}}$, and hence $\mathsf{quot}_{\overline{D}} \circ \beta = [\![-]\!]$ as required, by taking the monad morphism $m = \mathsf{quot}$.

Proposition 5. *Let M, N be strong monads on $\mathcal{C}$, $m : M \Rightarrow N$ a strong monad morphism, and B the endofunctor $BX = V + X$ with liftings $\overline{B}^M$ and $\overline{B}^N$ to $\mathsf{Kl}(M)$ and $\mathsf{Kl}(N)$, with final coalgebras of carrier $\overline{D}$ and underlying structure $\eta^M \circ \alpha^{-1}$ and $\eta^N \circ \alpha^{-1}$ for some arrow $\alpha^{-1} : \overline{D} \to B\overline{D}$. Given an abstract EIC specification ϵ in terms of monad M, and its translation via m into a specification in terms of monad N, the corresponding final coalgebra maps β^M, β^N from the induced operational models $T0 \to MBT0, T0 \to NBT0$ into $M\overline{D}$ and $N\overline{D}$ satisfy $\beta^N = m \circ \beta^M$, provided the distributive laws λ^M, λ^N lifting B satisfy this equation (†): $\lambda^N_X \circ Bm_X = m_{BX} \circ \lambda^M_X$.*

Note that if M and N are $\mathsf{Cpo}_!$-monads, then the monad morphism m is strong if it is a $\mathsf{Cpo}_!$-natural transformation (see Remark 1.4 of [10]). As the functor $\mathsf{Cpo}_!(I, -) : \mathsf{Cpo}_! \to \mathsf{Set}$ is faithful, $\mathsf{Cpo}_!$-naturality is essentially equivalent to ordinary naturality ([8] Section 1.3); thus the monad morphism quot may be assumed to be strong. In addition, the condition (†) is easily verified for the liftings $\overline{B}$ we used in Remark 1.

One may give compositional and adequate denotational semantics to CETS's by exactly the same method, where $M = T_{\mathrm{ce}}, N = N_{\mathrm{ce}}$, and the monad morphism is given by $(\mathsf{quot}_{\overline{D}\otimes C1})^{C1} : T_{\mathrm{ce}} \Rightarrow N_{\mathrm{ce}}$. This gives us the main result of our paper:

Corollary 1. *For a language induced by an EIC specification – a (syntactic)* CTS, ETS *or* CETS *transition system – the above assignments of denotations $[\![-]\!]$ to programs are adequate and compositional with respect to the behavioural equivalences $\cong_{\mathrm{c}}, \cong^{\mathrm{N}}_{\mathrm{e}}, \cong^{\mathrm{N}}_{\mathrm{ce}}$. That is, two programs have the same denotation iff they are operationally equivalent; and the assignment of denotations is a congruence.*

6 Conclusion

In this paper, we have given operational and denotational semantics, and syntactic rule formats, for a class of sequential imperative languages with notions of state and/or effects. We have given proofs that under these rule formats, the induced semantics are adequate and compositional. We anticipate applications with various combinations of user input/output, probabilistic non-determinism, and perhaps local state [18,21].

However, there are some limitations on the effect-theories permitted, due to the $\mathsf{Cpo}_!$-enrichment ensuring existence of a final Kleisli-coalgebra. Exceptions are inexpressible in $\mathsf{Cpo}_!$, and the semantics of user I/O is unsatisfactory: divergent programs are identified even if their I/O behaviour is clearly different. In addition, the coalgebraic semantics may still be too fine-grained, in that it records the number of execution steps of computations. One way around these limitations might be to move towards a *weakly-final* semantics in non-strict Cpo, again taking the semantic domain to be $M\overline{D}$ where $\overline{D}$ is the initial B-algebra, $M\overline{D}$, but where the semantic map is now a least fixpoint.

For commutative effect monads M, another way of extending Turi and Plotkin's framework may be to lift both syntax and behaviour functors into the Kleisli category $\mathsf{Kl}(M)$ and use it as the base category (rather than splitting the approach as we have done). An application might be trace semantics for CCS-like languages, as in [13].

References

1. Abou-Saleh, F., Pattinson, D.: Towards effects in mathematical operational semantics. Electr. Notes Theor. Comput. Sci. 276, 81–104 (2011)
2. Adamek, J.: Recursive data types in algebraically w-complete categories. Information and Computation 118(2), 181–190 (1995)
3. Adámek, J., Rosický, J.: Locally Presentable and Accessible Categories. Cambridge University Press (1994)
4. Hasuo, I., Jacobs, B., Sokolova, A.: Generic trace semantics via coinduction. Logical Methods in Computer Science 3(4) (2007)
5. Hyland, M., Plotkin, G., Power, J.: Combining effects: sum and tensor. Theor. Comput. Sci. 357, 70–99 (2006)
6. Hyland, M., Power, J.: Discrete lawvere theories and computational effects. Theor. Comput. Sci. 366(1-2), 144–162 (2006)
7. Johann, P., Simpson, A., Voigtländer, J.: A generic operational metatheory for algebraic effects. In: Proc. LICS 2010, pp. 209–218. IEEE Computer Society (2010)
8. Kelly, G.M.: Basic concepts of enriched category theory. Reprints in Theory and Applications of Categories (10), 1–136 (2005)
9. Klin, B.: Bialgebraic methods in structural operational semantics. Electron. Notes Theor. Comput. Sci. 175(1), 33–43 (2007)
10. Kock, A.: Strong functors and monoidal monads. Archiv der Mathematik 23 (1972)
11. Lenisa, M., Power, J., Watanabe, H.: Category theory for operational semantics. Theor. Comput. Sci. 327(1-2), 135–154 (2004)
12. Moggi, E.: Notions of computation and monads. Inf. Comput. 93(1), 55–92 (1991)
13. Monteiro, L.: A Coalgebraic Characterization of Behaviours in the Linear Time – Branching Time Spectrum. In: Corradini, A., Montanari, U. (eds.) WADT 2008. LNCS, vol. 5486, pp. 251–265. Springer, Heidelberg (2009)
14. Plotkin, G., Power, J.: Tensors of comodels and models for operational semantics. Electron. Notes Theor. Comput. Sci. 218, 295–311 (2008)
15. Plotkin, G.D., Power, J.: Adequacy for Algebraic Effects. In: Honsell, F., Miculan, M. (eds.) FOSSACS 2001. LNCS, vol. 2030, pp. 1–24. Springer, Heidelberg (2001)
16. Plotkin, G., Power, J.: Notions of Computation Determine Monads. In: Nielsen, M., Engberg, U. (eds.) FOSSACS 2002. LNCS, vol. 2303, pp. 342–356. Springer, Heidelberg (2002)
17. Power, J.: Countable lawvere theories and computational effects. Electr. Notes Theor. Comput. Sci. 161, 59–71 (2006)
18. Power, J.: Semantics for local computational effects. Electr. Notes Theor. Comput. Sci. 158, 355–371 (2006)
19. Power, J., Shkaravska, O.: From comodels to coalgebras: State and arrays. Electron. Notes Theor. Comput. Sci. 106 (2004)
20. Power, J., Turi, D.: A coalgebraic foundation for linear time semantics. In: Category Theory and Computer Science. Elsevier (1999)
21. Staton, S.: Completeness for Algebraic Theories of Local State. In: Ong, L. (ed.) FOSSACS 2010. LNCS, vol. 6014, pp. 48–63. Springer, Heidelberg (2010)
22. Turi, D.: Functorial Operational Semantics and its Denotational Dual. PhD thesis, Free University, Amsterdam (June 1996)
23. Turi, D.: Categorical modelling of structural operational rules: Case studies. In: Category Theory and Computer Science, pp. 127–146 (1997)
24. Turi, D., Plotkin, G.D.: Towards a mathematical operational semantics. In: LICS, pp. 280–291 (1997)