I

1 Initial Algebras, Induction and Program Syntax

The *syntax* of a programming language is usually defined by *induction* on some basic constructs $\sigma \in \Sigma$. Formally, Σ is a *signature* and the syntax is the initial Σ-algebra. Equivalently, the signature defines an endofunctor with action $X \mapsto \coprod_{\sigma \in \Sigma} X^{\mathrm{ar}(\sigma)}$, whose algebras are the same as the algebras of the signature. This leads to the standard categorical construction of initial Σ-algebras as suitable ω-colimits.

Consider, as an example, a simple imperative language whose constructs are some primitive *actions* $a \in \mathsf{Act}$, a *sequential composition* operator '$;$', and an 'inert' program nil. Correspondingly, the (single-typed) **signature** Σ of the above language is given by a set $\mathsf{Act} + 1$ of constants (ie operators of arity 0) and an operator of arity two.

The **programs** or **terms** t induced by the above signature Σ and some variables $x \in X$ are given by the grammar:

$$t ::= x \mid a \mid \mathsf{nil} \mid (t\ ;\ t)$$

Denote this set of programs by TX. In particular, for $X = 0$, ie the empty set, the set $T0$ gives the **closed terms** of the language:

$$t ::= a \mid \mathsf{nil} \mid (t\ ;\ t)$$

An alternative way of describing the set $T0$ of closed terms is as the carrier of the initial algebra of the signature Σ, that is, the initial object in the category of Σ-algebras, where Σ is the above signature. In general, given a signature Σ, the category of Σ-algebras has as objects pairs $\langle X, h\rangle$, where the **carrier** X is a set, and the **structure** h is a function interpreting each operator σ in the signature as a function $h(\sigma) : X^{\mathrm{arity}(\sigma)} \to X$. An arrow $f : \langle X, h\rangle \to \langle Y, k\rangle$ in this category is a function $f : X \to Y$ between the underlying sets such that, for every operator σ in the signature, the following diagram commutes.

$$\begin{tikzcd}
X^{\mathrm{ar}(\sigma)} \arrow[r, "f^{\mathrm{ar}(\sigma)}"] \arrow[d, "h\sigma"'] & Y^{\mathrm{ar}(\sigma)} \arrow[d, "k\sigma"] \\
X \arrow[r, "f"'] & Y
\end{tikzcd}$$

That is,

$$f(h\sigma)(x_1,\ldots,x_{\mathrm{ar}(\sigma)}) = (k\sigma)(fx_1,\ldots,fx_{\mathrm{ar}(\sigma)})$$

Notice that if the arity of an operator σ is zero, then $X^{\mathrm{ar}(\sigma)}$ is simply 1, the singleton set. The corresponding function $h\sigma : 1 \to X$ maps $*$, the unique element of 1, into an element of X. This gives the interpretation of a constant σ in the algebra.

For any signature Σ, the initial algebra always exists. It is the *term algebra* having as carrier the set $T0$ of closed terms over the signature and as algebra structure the evident one which maps, for every operator σ, a tuple $(t_1,\ldots,t_{\mathrm{ar}(\sigma)})$ of terms into the term $\sigma(t_1,\ldots,t_{\mathrm{ar}(\sigma)})$. Indeed, given any Σ-algebra $\langle X,h\rangle$, there is a unique arrow from the term algebra into $\langle X,h\rangle$, namely the function, say,

$$h^{\#} : T0 \to X$$

inductively defined as follows.

$$h^{\#}(\sigma(t_1,\ldots,t_{\mathrm{ar}(\sigma)})) = (h\sigma)(h^{\#}t_1,\ldots,h^{\#}t_{\mathrm{ar}(\sigma)})$$

Notice that the term algebra is initial also in the category of *partial* Σ-algebras, that is, algebras where the operators of the signature might be interpreted not only as total but also as partial functions.

A more compact way of describing the category of Σ-algebras is by taking the coproduct $\coprod_{\sigma\in\Sigma} X^{\mathrm{ar}(\sigma)}$, that is, the disjoint union of the domains of the operations. More formally, every signature Σ can be seen as a functor $\Sigma : \mathbf{Set} \to \mathbf{Set}$ (thus an **endofunctor** on **Set**) defined on objects as follows.

$$X \longmapsto \coprod_{\sigma\in\Sigma} X^{\mathrm{ar}(\sigma)}$$

For example, the endofunctor corresponding to the above signature $\Sigma = \mathsf{Act} \cup \{\mathsf{nil},\ ;\ \}$ is

$$\Sigma X \;=\; 1 + (\coprod_{Act} 1) + X\times X \;\cong\; 1 + \mathsf{Act} + X\times X$$

The category of algebras of a signature is then an instance of the following more general notion.

> Let $\Sigma : \mathbf{Set} \to \mathbf{Set}$ be *any* endofunctor on **Set**. The **category of Σ-algebras**, denoted by $\mathbf{Set}^{\Sigma}$, has as objects pairs $\langle X,h\rangle$, with X a set and $h : \Sigma X \to X$ a function. The arrows of the category are functions between the underlying sets preserving the algebra structure, that is, making the following diagram commute.

$$\begin{tikzcd}
\Sigma X \arrow[r, "\Sigma f"] \arrow[d, "h"'] & \Sigma Y \arrow[d, "k"] \\
X \arrow[r, "f"'] & Y
\end{tikzcd}$$

That is,

$$f \circ h = k \circ \Sigma f$$

Even more generality can be achieved by considering also algebras of endofunctors on categories $\mathbf{C}$ other than $\mathbf{Set}$. For instance, since any endofunctor corresponding to a signature Σ extends to the category $\mathbf{pSet}$ of sets and partial functions, the category $\mathbf{pSet}^\Sigma$ can be considered: this is the same as the category of *partial* Σ-algebras mentioned above.

The initial object in the category of algebras of an arbitrary endofunctor Σ, ie the **initial Σ-algebra**, does not always exists, but if it does, then its structure is an isomorphism:

> **Initial algebras are isomorphisms.** (*Lambek's Lemma.*) Let $\langle \overline{\Sigma}, \psi \rangle$ be the initial algebra of an arbitrary endofunctor Σ. Then the algebra structure $\psi : \Sigma\overline{\Sigma} \to \overline{\Sigma}$ is always an isomorphism
>
> $$\psi : \Sigma\overline{\Sigma} \cong \overline{\Sigma} \qquad \text{(initial } \Sigma\text{-algebra)}$$
>
> (To prove this notice that the initial algebra structure ψ is also a Σ -algebra arrow from $\langle \Sigma\overline{\Sigma}, \Sigma\psi \rangle$ to $\langle \overline{\Sigma}, \psi \rangle$.)

As mentioned in the introduction, initial algebras give a very useful *induction principle*. Indeed, every algebra structure $h : \Sigma X \to X$ of an arbitrary endofunctor Σ with initial algebra $\Sigma\overline{\Sigma} \cong \overline{\Sigma}$ can be inductively extended to an arrow $h^\# : \overline{\Sigma} \to X$ by taking the unique algebra arrow from the initial algebra to the algebra $\langle X, h \rangle$:

Inductive Extension

$$\begin{tikzcd}
\Sigma\overline{\Sigma} \arrow[r, "\Sigma h^\#"] \arrow[d, "\cong"'] & \Sigma X \arrow[d, "h"] \\
\overline{\Sigma} \arrow[r, dashed, "h^\#"'] & X
\end{tikzcd}$$

Notice this is a definition which holds in any category of algebras, thus, for instance, also for *partial* Σ-algebras.

Next, consider the construction of initial algebras. In the general setting where the endofunctor Σ might not stem from a signature, the initial Σ-algebra does not always arise from an inductive construction and might even fail to exist. But for the so-called $\boldsymbol{\omega}$-cocontinuous endofunctors, like those corresponding to signatures, the construction of the initial algebra is inductive indeed. Here $\boldsymbol{\omega}$ is the category having natural numbers as objects and arrows $n \to m$ iff $n \leq m$; that is, $\boldsymbol{\omega} = \{0 \to 1 \to 2 \to \cdots\}$. An **$\boldsymbol{\omega}$-cocontinuous functor** $F : \mathbf{C} \to \mathbf{D}$ is then a functor which preserves the colimits of functors $J : \boldsymbol{\omega} \to \mathbf{C}$, that is, $F\mathrm{Colim}J \cong \mathrm{Colim}FJ$. (The categories $\mathbf{C}$ and $\mathbf{D}$ are thus supposed to have these colimits.) Notice that a functor $J : \boldsymbol{\omega} \to \mathbf{C}$ is a diagram in $\mathbf{C}$ of the form $\{C_0 \xrightarrow{f_0} C_1 \xrightarrow{f_1} C_2 \xrightarrow{f_2} \cdots\}$.

The construction of the initial algebra of an $\boldsymbol{\omega}$-cocontinuous endofunctor is the functorial generalization of the least fixed point construction of an endofunction f in a partial order, namely as the least upper bound $\bigsqcup_{n<\omega} f^n\bot$. (This works if the partial order has a least element $\bot$ and the desired lub, and the function preserves that lub.) A partial order is a category with at most one arrow from one object to another. For such a category, the initial object is the least element, an endofunctor is a monotone endofunction, and $\boldsymbol{\omega}$-cocompleteness amounts to chain-completeness, ie, to the existence of least upper bounds of ω-chains. An $\boldsymbol{\omega}$-cocontinuous functor is thus a monotone function which preserves lubs of ω-chains. Finally, an algebra is a pre-fixed point $fx \leq x$ and the initial algebra is the least (pre-)fixed point.

Let Σ be an $\boldsymbol{\omega}$-cocontinuous endofunctor on $\mathbf{Set}$. Consider the unique function, say $0_{\Sigma 0}$, from the initial object in $\mathbf{Set}$ (the empty set – denoted by 0) to the set $\Sigma 0$. Next, consider the diagram D obtained by the iterative application of the endofunctor Σ to the initial function $0_{\Sigma 0}$; that is, for every n in ω, map the arrow $n \to n+1$ of $\boldsymbol{\omega}$ into $\Sigma^n 0_{\Sigma 0}$:

$$0 \xrightarrow{0_{\Sigma 0}} \Sigma 0 \xrightarrow{\Sigma 0_{\Sigma 0}} \Sigma^2 0 \xrightarrow{\Sigma^2 0_{\Sigma 0}} \cdots$$

Let Σ^ω be the colimit of this diagram D. Then, since the endofunctor Σ is $\boldsymbol{\omega}$-cocontinuous, $\Sigma\Sigma^\omega$ is the colimit of the diagram ΣD (which is simply D without the first arrow). Next, consider the colimiting cocone $\nu : D \Rightarrow \Sigma^\omega$:

$$\begin{tikzcd}
0 \arrow[r, "0_{\Sigma 0}"] \arrow[rrrr, bend right=60, "\nu_0"'] & \Sigma 0 \arrow[r, "\Sigma 0_{\Sigma 0}"] \arrow[rrr, bend right=45, "\nu_1"'] & \Sigma^2 0 \arrow[r, "\Sigma^2 0_{\Sigma 0}"] \arrow[rr, bend right=30, "\nu_2"'] & \cdots & \Sigma^\omega
\end{tikzcd}$$

Without the first component ν_0 this is also a cocone from ΣD to Σ^ω. Then:

> In the above construction, the mediating arrow from the colimit $\Sigma\Sigma^\omega$ of ΣD into Σ^ω gives the initial Σ-algebra structure. This can be proved by noticing that, for any algebra Σ-algebra $\langle X, h\rangle$, a cocone from D to X can be obtained as illustrated in the diagram below and then the inductive extension of the algebra structure $h : \Sigma X \to X$ is given by the

corresponding mediating arrow.

$$\begin{tikzcd}
0 \arrow[r, "0_{\Sigma 0}"] \arrow[d, "0_X"'] & \Sigma 0 \arrow[r, "\Sigma 0_{\Sigma 0}"] \arrow[d, "\Sigma 0_X"'] & \Sigma^2 0 \arrow[r, "\Sigma^2 0_{\Sigma 0}"] \arrow[d, "\Sigma^2 0_X"'] & \cdots \\
X & \Sigma X \arrow[l, "h"] & \Sigma^2 X \arrow[l, "\Sigma h"] & \cdots \arrow[l, "\Sigma^2 h"]
\end{tikzcd}$$

(This is the "Basic Lemma" from [SP82].)

Notice that the above construction applies to *any* category with initial object and $\boldsymbol{\omega}$-colimits. Thus, for instance, it can be applied also to $\boldsymbol{\omega}$-cocontinuous endofunctors on **pSet**.

Evident $\boldsymbol{\omega}$-cocontinuous endofunctors are identity and constant functors, as well as colimit functors (because of the standard "interchange of colimits") like coproducts. In **Set**, also finite products are $\boldsymbol{\omega}$-cocontinuous (see, eg, [Mac71, Theorem IX.2.1]), hence, since $\boldsymbol{\omega}$-cocontinuousness is preserved by composition, the endofunctors corresponding to signatures are $\boldsymbol{\omega}$-cocontinuous. Similarly, for every signature Σ and every set X, the endofunctor

$$(X+\Sigma) : \mathbf{Set} \to \mathbf{Set}$$

with action $Y \mapsto X + \Sigma Y$, is $\boldsymbol{\omega}$-cocontinuous, hence its initial algebra exists: it is the **algebra freely generated by** Σ **on** X, with as carrier TX, the set of terms with variables $x \in X$. Since initial algebras are isomorphisms

$$X + \Sigma TX \cong TX$$

the set TX is a coproduct and its algebra structure is the copair of the injections

$$\mathsf{inl}_X : X \to TX \qquad \mathsf{inr}_X : \Sigma TX \to TX$$

The left injection is the usual insertion of variables $x \in X$ into the terms $t \in TX$, which is usually left implicit. Formally, x is simply an element of the set X and it is only after applying inl_X to it that one obtains a variable. This variable-making function is usually written as

$$\eta_X = \mathsf{inl}_X : X \to TX$$

The other injection $\mathsf{inr}_X : \Sigma TX \to TX$ is the operation which permits to construct a new term given any n-ary operator σ and terms $t_1, \ldots, t_n$; also the right injection is usually left implicitly and one writes simply $\sigma(t_1, \ldots, t_n)$ for the resulting term.

Like $T0$, also TX, being an initial algebra, comes with an induction principle. and, since it is a coproduct, one can rephrase the principle as follows. For every

Σ-algebra structure $h : \Sigma Z \to Z$ and every '*valuation*' function $f : X \to Z$ of the variables in X as elements of the algebra $\langle Z, h \rangle$, there exists a unique function $f^\sharp : TX \to Z$ making

$$\begin{tikzcd}
X \arrow[r, "\eta_X = \mathsf{inl}_X"] \arrow[dr, "f"'] & TX \arrow[d, dashed, "f^\sharp"] & \Sigma TX \arrow[l, "\mathsf{inr}_X"'] \arrow[d, "\Sigma f^\sharp"] \\
 & Z & \Sigma Z \arrow[l, "h"]
\end{tikzcd}$$

commute. Omitting the injections,

$$f^\sharp(x) = f(x) \quad \text{and} \quad f^\sharp(\sigma(t_1, \ldots, t_n)) = h(\sigma(f^\sharp(t_1), \ldots, f^\sharp(t_n)))$$

This **inductive extension of** h **along the** valuation **function** f is, formally, the inductive extension $[f, h]^\#$ of the $(X+\Sigma)$-algebra structure on Z given by the copair

$$\begin{tikzcd}
X \arrow[r, tail] \arrow[dr, "f"'] & X+\Sigma Z \arrow[d, dashed, "{[f,h]}"] & \Sigma Z \arrow[l, tail] \arrow[dl, "h"] \\
 & Z &
\end{tikzcd}$$

For instance, this induction principle can be used to show that the operator T inductively extends to a functor $T : \mathbf{Set} \to \mathbf{Set}$. Indeed, to define its action Tf on a function $f : X \to Y$, take the inductive extension of $\mathsf{inr}_Y : \Sigma TY \to TY$ along the composite $\mathsf{inl}_Y \circ f$:

$$\begin{tikzcd}
X \arrow[r, "\eta_X = \mathsf{inl}_X"] \arrow[d, "f"'] & TX \arrow[d, dashed, "Tf = (\eta_Y \circ f)^\sharp"] & \Sigma TX \arrow[l, "\mathsf{inr}_X"'] \arrow[d, "\Sigma Tf"] \\
Y \arrow[r, "\eta_Y = \mathsf{inl}_Y"'] & TY & \Sigma TY \arrow[l, "\mathsf{inr}_Y"]
\end{tikzcd}$$

To prove that this definition is functorial, ie $T(\mathsf{id}_X) = \mathsf{id}_{TX}$ and $T(g \circ f) = Tg \circ Tf$, for $g : Y \to Z$, one exploits the uniqueness of inductive extensions: the function id_{TX} fits as $(\eta_X \circ \mathsf{id}_X)^\sharp = (\eta_X)^\sharp$ and $Tg \circ Tf$ fits as $(\eta_Z \circ g \circ f)^\sharp$.

Notice that a function $f : X \to Y$ can be seen as a '*renaming*' of variables and then the function $Tf : TX \to TY$ is the inductive extension of such a renaming from simple variables to complex terms with variables.

Another example is the definition of the operation $\mu_X : TTX \to TX$ inductively extending $\mathsf{inr}_X : \Sigma TX \to TX$ along the identity on TX:

$$
\begin{tikzcd}
TX \arrow[r, "\eta_{TX}"] \arrow[dr, equal] & T^2X \arrow[d, dashed, "\mu_X = (\mathsf{id}_{TX})^\sharp"] & \Sigma T^2X \arrow[l, "\mathsf{inr}_{TX}"'] \arrow[d, "\Sigma\mu_X"] \\
 & TX & \Sigma TX \arrow[l, "\mathsf{inr}_X"]
\end{tikzcd}
$$

This permits to form terms from *any* operator derivable from the signature. For instance, for the above sample language, consider the derived (unary) operator '$a\ ;\ (\_)$': given any term $t \in TX$, one can form the term $a\,;t$ by first applying $a\ ;\ (\_)$ to t and then μ_X:

$$a\ ; t = \mu_X(a\ ;\ (t))$$

Derived operators can also be seen as *contexts* and then the operation μ_X is formally needed to remove brackets after plugging terms in the holes of a context.

Notes. For a comprehensive survey on the use of Σ-algebras in semantics see [MT92].

2 Terms, Algebras and Monads

The inductive definition of the syntax of a language as a free algebra on a signature Σ defines a '*syntactical monad*' T. In general, every algebraic theory $\langle \Sigma, E \rangle$ defines a monad T and, 'conversely', every monad is defined by its algebras in a categorical, more abstract sense.

Let I be the identity functor. By definition, the insertion-of-variables function $\eta_X = \mathsf{inl}_X : X \to TX$ introduced in the previous section is natural in X:

$$\eta : I \Rightarrow T$$

Similarly, the brackets-removing function $\mu_X : T^2 X \to TX$ is natural in X, because it is the inductive extension of a natural transformation (the right injection $\mathsf{inr} : \Sigma T \Rightarrow T$) along the identity. The triple

$$T = < T, \eta, \mu >$$

is a 'monad' on **Set**.

> A **monad** in a category $\mathbf{C}$ is like a monoid in $\mathbf{C}^{\mathbf{C}}$ – the category having as objects endofunctors on $\mathbf{C}$ and as arrows natural transformations between them: it is a triple $< T, \eta, \mu >$ consisting of an object $T : \mathbf{C} \to \mathbf{C}$, an associative **multiplication** $\mu : T^2 \Rightarrow T$, and a **unit** $\eta : I \Rightarrow T$ for this multiplication. Notice that $T^2 = T \circ T$, thus the composition of functors is used in this definition rather than their product. Diagrammatically, the associativity and the (left and right) unit laws are expressed as follows.

Monad Laws

$$\begin{tikzcd}
T^3 \arrow[r, Rightarrow, "T\mu"] \arrow[d, Rightarrow, "\mu T"'] & T^2 \arrow[d, Rightarrow, "\mu"] \\
T^2 \arrow[r, Rightarrow, "\mu"'] & T
\end{tikzcd}
\qquad
\begin{tikzcd}
IT \arrow[r, Rightarrow, "\eta T"] \arrow[dr, equal] & T^2 \arrow[d, Rightarrow, "\mu"] & TI \arrow[l, Rightarrow, "T\eta"'] \arrow[dl, equal] \\
 & T &
\end{tikzcd}$$

In order to prove that the free Σ-algebra functor T, together with the left injection $\eta = \mathsf{inl} : I \Rightarrow T$ as unit and the inductive extension of the right injection $\mathsf{inr} : \Sigma T \Rightarrow T$ along the identity as multiplication $\mu : T^2 \Rightarrow T$, is a monad on **Set**, recall the definition of μ:

$$\begin{tikzcd}
TX \arrow[r, "\eta_{TX}"] \arrow[dr, equal] & T^2X \arrow[d, "\mu_X"] & \Sigma T^2X \arrow[l, "\mathsf{inr}_{TX}"'] \arrow[d, "\Sigma\mu_X"] \\
& TX & \Sigma TX \arrow[l, "\mathsf{inr}_X"]
\end{tikzcd}$$

The commutativity of the triangle on the left shows that η and μ satisfy the left unit law. As for the right unit law, exploit the uniqueness of inductive extensions, noticing that both the identity on TX and the composite $\mu_X \circ T\eta_X$ fit as the (unique!) inductive extension ${\eta_X}^\sharp$ of inr_X along η_X:

$$\begin{tikzcd}
X \arrow[rr, "\eta_X"] \arrow[dr, "\eta_X"'] \arrow[dd, equal] & & TX \arrow[d, "T\eta_X"] & \Sigma TX \arrow[l, "\mathsf{inr}_X"'] \arrow[d, "\Sigma T\eta_X"] \\
& TX \arrow[r, "\eta_{TX}"] \arrow[dr, equal] & T^2X \arrow[d, "\mu_X"] & \Sigma T^2X \arrow[l, "\mathsf{inr}_{TX}"'] \arrow[d, "\Sigma\mu_X"] \\
X \arrow[rr, "\eta_X"'] & & TX & \Sigma TX \arrow[l, "\mathsf{inr}_X"]
\end{tikzcd}$$

Indeed, everything in sight in the above diagram commutes, either by definition or by naturality (of η and inr). Similarly, one can prove the associativity law by noticing that both composites $\mu_{TX} \circ \mu_X$ and $T\mu_X \circ \mu_X$ fit as the inductive extension ${\mu_X}^\sharp$ of inr_X along μ_X.

From adjunctions to monads. A source of monads is to be found in adjunctions:

Every adjunction from a category $\mathbf{C}$ *to a category* $\mathbf{D}$

$$\begin{tikzcd} \mathbf{D} \arrow[d, bend left, "G"] \\ \mathbf{C} \arrow[u, bend left, "F"] \arrow[u, phantom, "\dashv"] \end{tikzcd} \qquad \begin{array}{l} \text{counit} = \varepsilon : FG \Rightarrow I \\ \\ \\ \text{unit} = \eta : I \Rightarrow GF \end{array}$$

gives rise to a monad $T =< GF, \eta, G\varepsilon F >$ *on* $\mathbf{C}$.

For a proof of this fact see, eg, [Mac71, §VI.1]; here, as an example, consider again the above term monad. Firstly, notice that the property that every Σ-algebra structure $h : \Sigma Z \to Z$ can be inductively extended along any function $f : X \to Z$ to a function $f^\sharp : TX \to Z$ amounts to the fact that the forgetful functor $U^\Sigma : \mathbf{Set}^\Sigma \to \mathbf{Set}$, mapping Σ-algebras to their carriers, has a left adjoint, namely the functor

$$F^\Sigma : \mathbf{Set} \to \mathbf{Set}^\Sigma \qquad\qquad X \mapsto (\mathsf{inr}_X : \Sigma TX \to TX)$$

Indeed, the diagram defining $f^\sharp$

$$\begin{tikzcd} X \arrow[r, "\eta_X"] \arrow[dr, "f"'] & TX \arrow[d, dashed, "f^\sharp"] & \Sigma TX \arrow[l, "\mathsf{inr}_X"'] \arrow[d, "\Sigma f^\sharp"] \\ & Z & \Sigma Z \arrow[l, "h"] \end{tikzcd}$$

can be decomposed into

$$\begin{tikzcd} X \arrow[r, "\eta_X"] \arrow[dr, "f"'] & TX = U^\Sigma\langle TX, \mathsf{inr}_X\rangle \arrow[d, "f^\sharp = U^\Sigma f^\sharp"] \\ & Z = U^\Sigma\langle Z, h\rangle \end{tikzcd} \qquad \begin{tikzcd} TX \arrow[d, dashed, "f^\sharp"] & \Sigma TX \arrow[l, "\mathsf{inr}_X"'] \arrow[d, "\Sigma f^\sharp"] \\ Z & \Sigma Z \arrow[l, "h"] \end{tikzcd}$$

which shows that F^Σ is the left adjoint of U^Σ and, moreover, that η is the unit of the adjunction.

Next, the counit ε of the adjunction is $\mathsf{id}^\sharp$, ie, for every Σ-algebra structure $h : \Sigma X \to X$,

$$\varepsilon_h : TX \to X$$

is the inductive extension of h along the identity on X. Then, indeed, from $F^\Sigma \dashv U^\Sigma$, one gets the above monad as follows.

$$T = U^\Sigma F^\Sigma \qquad \eta = \eta \qquad \mu = \varepsilon_{\mathsf{inr}} = \varepsilon_{F^\Sigma} = U^\Sigma \varepsilon_{F^\Sigma}$$

From monads to adjunctions

Not only every adjunction gives rise to a monad, but also, conversely, every monad splits into an adjunction. In general, there are many categories **D** such that a monad in **C** splits into an adjunction from **C** to **D**, but there are two canonical ones, namely the initial and the final ones in a suitable sense. Consider the final one; it is defined by adding some extra conditions on the objects of the category of algebras of an endofunctor:

> Let $T =< T, \eta, \mu >$ be a monad in a category **C**. The **category of** T**-algebras**, denoted by $\mathbf{C}^T$, has as objects pairs $\langle X, h\rangle$, with X an object of **C** and $h : TX \to X$ an arrow of **C** such that the following diagrams commute.
>
> **T-Algebra Laws**
>
> $$\begin{tikzcd} T^2X \arrow[r, "Th"] \arrow[d, "\mu_X"'] & TX \arrow[d, "h"] \\ TX \arrow[r, "h"'] & X \end{tikzcd} \qquad \begin{tikzcd} X \arrow[r, "\eta_X"] \arrow[dr, equal] & TX \arrow[d, "h"] \\ & X \end{tikzcd}$$
>
> The arrows of the category are those arrows of the category **C** which preserve the algebra structure, that is, making the following diagram commute.
>
> $$\begin{tikzcd} TX \arrow[r, "Tf"] \arrow[d, "h"'] & TY \arrow[d, "k"] \\ X \arrow[r, "f"'] & Y \end{tikzcd}$$
>
> (This category is also called the *Eilenberg-Moore category* of the monad.)

Notice that, in particular, $\langle TX, \mu_X\rangle$ is a T-algebra for every object X in **C**. Therefore, also $\langle T^2X, \mu_{TX}\rangle$ is a T-algebra and μ_X is an algebra arrow between them.

Another example of a T-algebra structure is given by the above inductive extension $\varepsilon_h : TX \to X$ of a Σ-algebra structure $h : \Sigma X \to X$ along the identity on X. Indeed, the law $\varepsilon_h \circ \eta_X = \mathsf{id}_X$ holds by definition, while the other law holds because both composites $\varepsilon_h \circ \mu_X$ and $\varepsilon_h \circ T\varepsilon_h$ fit as the inductive extension ${\varepsilon_h}^\sharp$ of h along ε_h

$$\begin{tikzcd}
TX \arrow[r, "\eta_{TX}"] \arrow[d, "\varepsilon_h"'] & T^2X \arrow[dd, dashed, "{\varepsilon_h}^\sharp"] & \Sigma T^2 X \arrow[l, "\mathsf{inr}_{TX}"'] \arrow[dd, "\Sigma{\varepsilon_h}^\sharp"] \\
X \arrow[dr, equal] & & \\
& X & \Sigma X \arrow[l, "h"]
\end{tikzcd}$$

as shown by the commutativity of the following two diagrams.

$$\begin{tikzcd}
TX \arrow[r, "\eta_{TX}"] \arrow[d, "\varepsilon_h"'] & T^2X \arrow[d, "T\varepsilon_h"] & \Sigma T^2 X \arrow[l, "\mathsf{inr}_{TX}"'] \arrow[d, "\Sigma T\varepsilon_h"] \\
X \arrow[r, "\eta_X"] \arrow[dr, equal] & TX \arrow[d, "\varepsilon_h"] & \Sigma TX \arrow[l, "\mathsf{inr}_X"'] \arrow[d, "\Sigma\varepsilon_h"] \\
& X & \Sigma X \arrow[l, "h"]
\end{tikzcd}
\qquad
\begin{tikzcd}
TX \arrow[r, "\eta_{TX}"] \arrow[d, "\varepsilon_h"'] \arrow[dr, equal] & T^2X \arrow[d, "\mu_X"] & \Sigma T^2 X \arrow[l, "\mathsf{inr}_{TX}"'] \arrow[d, "\Sigma \mu_X"] \\
X \arrow[dr, equal] & TX \arrow[d, "\varepsilon_h"] & \Sigma TX \arrow[l, "\mathsf{inr}_X"'] \arrow[d, "\Sigma\varepsilon_h"] \\
& X & \Sigma X \arrow[l, "h"]
\end{tikzcd}$$

Σ-algebras are T-algebras. The above mapping

$$(h : \Sigma X \to X) \mapsto (\varepsilon_h : TX \to X)$$

taking a Σ-algebra structure on X into its coinductive extension along the identity on X

$$\begin{tikzcd}
X \arrow[r, "\eta_X"] \arrow[dr, equal] & TX \arrow[d, dashed, "\varepsilon_h"] & \Sigma TX \arrow[l, "\mathsf{inr}_X"'] \arrow[d, "\Sigma\varepsilon_h"] \\
& X & \Sigma X \arrow[l, "h"]
\end{tikzcd}$$

is an isomorphism between the category of Σ-algebras and the algebras of its corresponding monad T.

For the inverse of this mapping from Σ- to T-algebras, precompose each T-algebra $\langle X, h\rangle$ first with the right injection $\mathsf{inr}_X : \Sigma TX \to TX$ and then with $\Sigma\eta_X : \Sigma X \to \Sigma TX$

$$\langle X, h\rangle \mapsto \langle X, h \circ \mathsf{inr}_X \circ \Sigma\eta_X\rangle$$

One half of this isomorphism is illustrated by the following diagram, which commutes 'almost' by definition.

$$\begin{tikzcd}
X \arrow[r, "\eta_X"] \arrow[dr, equal] & TX \arrow[d, "\varepsilon_h"] & \Sigma TX \arrow[l, "\mathsf{inr}_X"'] \arrow[d, "\Sigma\varepsilon_h"'] & \Sigma X \arrow[l, "\Sigma\eta_X"'] \arrow[dl, equal] \\
& X & \Sigma X \arrow[l, "h"] &
\end{tikzcd}$$

The other half of the isomorphism, namely the commutativity of

$$\begin{tikzcd}
X \arrow[r, "\eta_X"] \arrow[dr, equal] & TX \arrow[d, "h"] & & & \Sigma TX \arrow[lll, "\mathsf{inr}_X"'] \arrow[d, "\Sigma h"] \\
& X & TX \arrow[l, "h"] & \Sigma TX \arrow[l, "\mathsf{inr}_X"] & \Sigma X \arrow[l, "\Sigma\eta_X"]
\end{tikzcd}$$

is more complex. To prove it, fill the above diagram with subdiagrams which commute either by the T-algebras laws (for the algebra $\langle X, k\rangle$) or by naturality (of the right injection inr and of the unit η), or by the 'identity law' for the monad T:

$$\begin{tikzcd}
X \arrow[r, "\eta_X"] \arrow[dddr, equal] & TX \arrow[ddd, "h"] & & & \Sigma TX \arrow[lll, "\mathsf{inr}_X"'] \arrow[ddl, "\Sigma\eta_{TX}"] \arrow[ddd, "\Sigma h"] \arrow[dll, phantom, "="] \\
& & \Sigma TX \arrow[ul, "\mathsf{inr}_X"] & & \\
& & T^2X \arrow[uul, "\mu_X"] \arrow[d, "Th"] & \Sigma T^2X \arrow[l, "\mathsf{inr}_{TX}"] \arrow[ul, "\Sigma\mu_X"'] \arrow[d, "\Sigma Th"] & \\
& X & TX \arrow[l, "h"] & \Sigma TX \arrow[l, "\mathsf{inr}_X"] & \Sigma X \arrow[l, "\Sigma\eta_X"]
\end{tikzcd}$$

This concludes the proof of the isomorphism between Σ- and T-algebras.

Under the above isomorphism, the free Σ-algebra structure $\mathsf{inr}_X : \Sigma TX \to TX$ over X corresponds to the T-algebra structure $\mu_X = \varepsilon_{\mathsf{inr}} : T^2X \to TX$. (See the concrete description of these two operations given in the previous section.) Recall that the forgetful functor $U^\Sigma : \mathbf{Set}^\Sigma \to \mathbf{Set}$ from the Σ-algebras has a left adjoint $F^\Sigma X = \langle X, \mathsf{inr}_X : \Sigma TX \cong TX\rangle$. Correspondingly, also the evident forgetful functor

$U^T : \mathbf{Set}^T \to \mathbf{Set}$ from the T-algebras has a left adjoint namely $F^T X = \langle X, \mu_X : T^2X \to TX\rangle$ and the following two diagrams commute.

$$\begin{tikzcd} \mathbf{Set}^\Sigma \arrow[r, phantom, "\cong"] \arrow[dr, "U^\Sigma"'] & \mathbf{Set}^T \arrow[d, "U^T"] \\ & \mathbf{Set} \end{tikzcd} \qquad \begin{tikzcd} \mathbf{Set}^\Sigma \arrow[r, phantom, "\cong"] & \mathbf{Set}^T \\ & \mathbf{Set} \arrow[ul, "F^\Sigma"] \arrow[u, "F^T"'] \end{tikzcd}$$

In general, the above adjunction $F^T \dashv U^T$ holds for algebras of monads on *any* category $\mathbf{C}$:

> **The adjunction $F^T \dashv U^T$ splitting the monad T.** The functor
>
> $$F^T : \mathbf{C} \to \mathbf{C}^T \qquad\qquad X \mapsto \langle X, \mu_X : T^2X \to TX\rangle$$
>
> is the left adjoint of the forgetful functor $U^T : \mathbf{C}^T \to \mathbf{C}$ mapping T-algebras to their carriers. The unit of this adjunction is the unit η of the monad. As for the counit $\varepsilon : F^T U^T \Rightarrow I$, this is simply
>
> $$\varepsilon_{\langle X,h\rangle} = h : F^T U^T \langle X, h\rangle = \langle TX, \mu_X\rangle \to \langle X, h\rangle$$
>
> which is a T-algebra arrow from $\langle TX, \mu_X\rangle$ to $\langle X, h\rangle$ because of the very definition of T-algebra structure. The right unit law of the monad and the T-algebra law for the unit are then the two triangular equalities which prove the adjunction $F^T \dashv U^T$.

The monad arising from this adjunction is the original monad T:

$$T \ = < T, \eta, \mu > = < U^T F^T,\ \eta\ ,\ U^T F^T \varepsilon >$$

Therefore:

Every monad is defined by its algebras.

Moreover, the adjunction $F^T \dashv U^T$ is the 'final' one defining the monad T; that is, from any adjunction

$$\begin{tikzcd} \mathbf{C} \arrow[r, shift left=1.5, "F"] \arrow[r, phantom, "\perp"] & \mathbf{D} \arrow[l, shift left=1.5, "U"] \end{tikzcd}$$

giving rise to the monad T there exists a unique 'comparison' functor $K : \mathbf{D} \to \mathbf{C}^T$ such that the following two diagrams commute.

$$\begin{tikzcd} \mathbf{D} \arrow[rr, dashed, "K"] \arrow[dr, "U"'] & & \mathbf{C}^T \arrow[dl, "U^T"] \\ & \mathbf{C} & \end{tikzcd} \qquad \begin{tikzcd} \mathbf{D} \arrow[rr, dashed, "K"] & & \mathbf{C}^T \\ & \mathbf{C} \arrow[ul, "F"] \arrow[ur, "F^T"'] & \end{tikzcd}$$

If $\varepsilon : FU \Rightarrow I$ is the counit of the adjunction $F \dashv U$, then, for every object D of $\mathbf{D}$,

$$KD = \langle UD, U\varepsilon_D : UFUD = TUD \to UD \rangle$$

When this comparison functor K is an isomorphism, then the functor $U : \mathbf{D} \to \mathbf{C}$ is called **monadic**. Thus, for instance, the forgetful functor $U^\Sigma : \mathbf{Set}^\Sigma \to \mathbf{Set}$ is monadic.

In general, to prove that a functor is monadic, one can use *Beck's theorem* (see, eg, [Mac71]) stating that a functor is monadic if and only if it 'creates' suitable coequalizers. In particular, this can be used to prove the following generalization of the above correspondence between Σ- and T-algebras.

> **Algebras are T-algebras.** Given a signature Σ and a set E of equations on the (derived) operators of the signature, consider the corresponding category $\mathbf{Set}^{\langle \Sigma, E \rangle}$ of Σ-algebras validating the equations in E and having as arrows functions which preserve the operators. Then, the evident forgetful functor from $\mathbf{Set}^{\langle \Sigma, E \rangle}$ to $\mathbf{Set}$ has a left adjoint and, moreover, it is monadic. Therefore, the category of algebras of the monad T corresponding to this adjunction is isomorphic to the category $\mathbf{Set}^{\langle \Sigma, E \rangle}$.

This shows that the notion of algebras of monads encompasses the standard notion of algebras as *varieties*, that is, as sets with operations from a signature Σ which validate a set of equations E. (Eg, monoids, groups, semi-lattices, etc.)

Notice that one might want to describe the programs of a language as a free $\langle \Sigma, E \rangle$-algebra rather than a free Σ-algebra. For instance, the behaviour of the sequential composition operator is intended to be associative thus one can axiomatize this directly in the syntax by adding the equation

$$x \,;\, (y \,;\, z) = (x \,;\, y) \,;\, z$$

Then, there will be no distinction in the syntax anymore between the program $t \,;\, (u \,;\, v)$ and the program $(t \,;\, u) \,;\, v$, ie they will represent the same program. (Another example is in Section 10, where the semi-lattice laws are imposed on the 'non-deterministic choice' operator 'or'.)

Equations can also be used to describe the behaviour of new operators algebraically. For instance, one can define a 'replication' operator '$!$' in terms of sequential composition by means of the equation

$$!x = x \,;\, (!x)$$

Thus, in general, the programs of a language might be terms of a signature Σ quotiented by (the smallest congruence generated by) a set of equations E. In the sequel, monads T corresponding to $\langle \Sigma, E \rangle$-algebras describing the programs of a language will be called **syntactical monads**.

Finally, notice that the fact that Σ-algebras are T-algebras holds also for arbitrary endofunctors $\Sigma : \mathbf{C} \to \mathbf{C}$ which have an initial $(X+\Sigma)$-algebra $TX \cong X+\Sigma TX$ for every object X in the category $\mathbf{C}$. That is, the forgetful functor $U^\Sigma : \mathbf{C}^\Sigma \to \mathbf{C}$ has a left adjoint $X \mapsto \langle TX, \mathsf{inr}_X : \Sigma TX \to TX \rangle$ and, moreover, it is monadic. Thus the isomorphism of categories

$$\mathbf{Set}^\Sigma \cong \mathbf{Set}^T$$

is not only an instance of

$$\mathbf{Set}^{\langle \Sigma, E \rangle} \cong \mathbf{Set}^T$$

but also of

$$\mathbf{C}^\Sigma \cong \mathbf{C}^T$$

3 Operational Semantics, Transition Systems and Coalgebras

Operational models like transition systems can be seen as '*coalgebras*' of suitable '*behaviour*' endofunctors.

The **operational semantics** of a language defines *how* programs are to be executed and what their observable effect is. More specifically, the operational semantics considered here aims at specifying the actions that programs can perform, like changing a state, and their subsequent *transitions* into new programs, usually the part of the code still remaining to be executed. The result is thus a relation of type

$$\mathsf{Programs} \times \mathsf{Actions} \times \mathsf{Programs}$$

usually denoted element-wise as a labelled arrow of type

$$\mathsf{program} \xrightarrow{\mathsf{act}} \mathsf{program}$$

Relations of this kind are called 'labelled transition systems' as they specify the (labelled) transitions between programs.

In general, a **transition system** with labels $a \in A$ is given by a set X of *states* and a family $\{\xrightarrow{a}\}_A$ of *transition relations* labelled by $a \in A$:

$$\langle X, \{\xrightarrow{a}\}_A \rangle$$

One reads

$$x \xrightarrow{a} x'$$

as 'from the state x the system can perform an action a and reach the state x''. Equivalently, a labelled transition system is a labelled directed graph: nodes = states, labelled arcs = transitions.

The **inert** states of a transition system are those from which no action can be performed. It is convenient to introduce an explicit predicate '$\downarrow *$' on states to express that one can *observe* that a state is inert:

$$x \downarrow * \iff x \text{ is inert}$$

Thus a transition system is a triple

$$\langle X, \{\xrightarrow{a}\}_A \,, \downarrow* \rangle$$

In general, given an operational semantics, it might not be easy to prove things about the behaviour of programs, like, for instance, to see whether a program is deterministic. In order to facilitate reasoning about programs, it is convenient that the operational semantics be **structured**, that is, the transition system should be defined by induction on the program constructs (**structural induction**). For example, the intended operational semantics for the simple imperative language

$$t ::= x \mid a \mid \mathsf{nil} \mid (t \mathbin{;} t)$$

could be specified by induction on the program constructs as follows.

Consider first the constant nil: its intended meaning is that it is the basic *inert* program, that is, a program which cannot perform any action. The only rule for it is then

$$\mathsf{nil} \downarrow *$$

Next, every constant a in Act is an atomic program which can perform the corresponding action a and then become inert:

$$a \xrightarrow{a} \mathsf{nil}$$

Finally, for the sequential composition operator there are three cases to be considered: (*i*) the first component can perform a transition; (*ii*) the first component is inert but the second component can perform a transition; (*iii*) both components are inert. That is, using also the meta-variables u, v, etc, to range over the programs of the language,

$$\frac{u \xrightarrow{a} u'}{u \mathbin{;} v \xrightarrow{a} u' \mathbin{;} v} \qquad \frac{u \downarrow * \quad v \xrightarrow{a} v'}{u \mathbin{;} v \xrightarrow{a} v'} \qquad \frac{u \downarrow * \quad v \downarrow *}{u \mathbin{;} v \downarrow *}$$

Let us denote the above set of rules by $\mathcal{R}$. All rules of $\mathcal{R}$ are *well-founded*, hence the least transition system closed under $\mathcal{R}$ does exist: this is the *intended model* for $\mathcal{R}$. Moreover, the rules of $\mathcal{R}$ are *finitary*, hence every transition in the intended model can be proved in a finite number of steps.

By structural induction, one can prove that the set of states of the intended model is the set $T0$ of closed programs. Indeed, there are axioms for all constants and if two programs u and v belong to the states of the model then also $u \mathbin{;} v$ does. Thus the intended model is of the form

$$\langle T0, \{\xrightarrow{a}\}_{\mathsf{Act}}, \downarrow * \rangle$$

Another property of the above transition system which can be proved by structural induction is that it is **deterministic**: there is only one rule for each constant and the three rules for sequential composition have, by induction, disjoint hypotheses; thus every program can perform at most one action.

A similar argument shows that every program can either perform an action or being inert; that is, for every closed program t, either there exists a unique action

a and a unique program t' such that $t \xrightarrow{a} t'$ or, otherwise, $t \downarrow *$. Therefore, this transition system (ie, the transition *relation* together with the predicate $\downarrow *$) can then be regarded as a single total *function*

$$[\![\text{-}]\!]_{\mathcal{R}} : T0 \to 1 + \mathsf{Act} \times T0$$

For this, put

$$[\![t]\!]_{\mathcal{R}} = * \iff t \downarrow * \qquad \text{and} \qquad [\![t]\!]_{\mathcal{R}} = <a, t'> \iff t \xrightarrow{a} t'$$

where, recall, '$*$' denotes the unique element of the final object 1 in **Set**. In general, this defines a one-to-one correspondence between deterministic transition systems and '*co*-algebras' of the endofunctor $BX = 1 + \mathsf{Act} \times X$ on **Set**.

> Given an endofunctor $B : \mathbf{C} \to \mathbf{C}$ on a category $\mathbf{C}$, the **category of B-coalgebras**, denoted by $\mathbf{C}_B$, has as objects pairs $\langle X, k \rangle$, with X an object of $\mathbf{C}$ and $k : X \to BX$ an arrow of $\mathbf{C}$. The arrows $f : \langle X, k \rangle \to \langle X', k' \rangle$ of $\mathbf{C}_B$ are the arrows $f : X \to X'$ of $\mathbf{C}$ which preserve the coalgebra structure:
>
> $$\begin{tikzcd} X \arrow[r, "f"] \arrow[d, "k"'] & X' \arrow[d, "k'"] \\ BX \arrow[r, "Bf"'] & BX' \end{tikzcd}$$
>
> (Cf Σ-algebras in Section 1.)

Thus a coalgebra of the endofunctor $BX = 1 + \mathsf{Act} \times X$ is a pair $\langle X, k \rangle$, with X a set and k a function

$$k : X \to 1 + \mathsf{Act} \times X$$

This can be seen as a deterministic transition system

$$\langle X, \{\xrightarrow{a}\}_{\mathsf{Act}}, \downarrow * \rangle$$

because of the correspondence

$$x \downarrow * \iff k(x) = * \qquad \text{and} \qquad x \xrightarrow{a} x' \iff k(x) = <a, x>$$

Notions of behaviour and endofunctors. The above correspondence between deterministic transition systems and coalgebras of the 'behaviour' endofunctor $BX = 1+\mathsf{Act}\times X$ generalizes to several forms of *non-deterministic* transition systems. More generally, the claim is that *coalgebras* are suitable to modelling the *operational behaviour* of the programs of a language. The corresponding endofunctors are called **behaviour endofunctors**.

Consider transition systems without the inert predicate $\downarrow\!*$. Take the endofunctor

$$BX = \mathcal{P}(\mathsf{Act} \times X)$$

where $\mathcal{P} : \mathbf{Set} \to \mathbf{Set}$ is the (covariant) power-set endofunctor: for every set X and function $f : X \to Y$

$$\mathcal{P}X = \{X' \mid X' \subseteq X\} \qquad (\mathcal{P}f)(X') = \{fx \mid x \in X'\}$$

Then, a one-to-one correspondence between coalgebras

$$k : X \to \mathcal{P}(\mathsf{Act} \times X)$$

and transition systems

$$\langle X, \{\xrightarrow{a}\}_{\mathsf{Act}}\rangle$$

is obtained by putting

$$<a, x'> \in k(x) \iff x \xrightarrow{a} x'$$

Another example is obtained by restricting the above behaviour to

$$BX = \mathcal{P}_{\mathrm{fi}}(\mathsf{Act} \times X)$$

where $\mathcal{P}_{\mathrm{fi}} : \mathbf{Set} \to \mathbf{Set}$ is the *finite power-set endofunctor*. Its coalgebras correspond to '*finitely branching* transition systems', that is transition systems which can, at each state, choose among a *finite* set of transitions rather than among an arbitrarily large one.

Notice in the two examples above that a state x is mapped by the coalgebra structure k to the empty set 0 if and only if the corresponding transition system cannot perform any transition from x. Alternatively, one can use the isomorphism

$$\mathcal{P}_{\mathrm{fi}}(\mathsf{Act} \times X) \cong 1 + \check{\mathcal{P}}(\mathsf{Act} \times X)$$

where $\check{\mathcal{P}}$ is the 'relevant' part of the (finite) power-set functor, mapping a set to the set of its (finite) and *non-empty* subsets. The coalgebras of the behaviour $BX = 1+\check{\mathcal{P}}(\mathsf{Act}\times X)$ are then finitely branching transition system with the explicit inert predicate $\downarrow\!*$. Omitting the injections into the coproduct $1 + \check{\mathcal{P}}(\mathsf{Act} \times X)$, the correspondence is as follows.

$$k(x) = * \iff x \downarrow * \qquad \text{and} \qquad <a, x'> \in k(x) \iff x \xrightarrow{a} x'$$

Here the transition relation and the inert predicate are disjoint: if a state can become inert then it cannot choose to perform an action. If, instead, one wants to consider transition systems with states in which both choices are allowed the following behaviour is to be used.

$$BX = \check{\mathcal{P}}(1 + \mathsf{Act} \times X)$$

Omitting the injections, one has the following correspondence.

$$* \in k(x) \iff x \downarrow * \qquad \text{and} \qquad <a, x'> \in k(x) \iff x \xrightarrow{a} x'$$

One step further is to consider the same behaviours as above but taken in **pSets** – the category of sets and partial functions – rather than in **Set**. This corresponds to considering *partial transition systems*, ie transition systems with states whose behaviour might be undefined.

> It should be stressed that the coalgebras of the above behaviours correspond only as *objects* to transition systems: the *arrows* are quite different. Consider the case of transition systems without the predicate $\downarrow *$. Then, following the definition of transition systems as relations (or as graphs) the natural definition of an arrow
>
> $$f : \langle X, \{\xrightarrow{a}_X\}_{\mathsf{Act}}\rangle \to \langle Y, \{\xrightarrow{a}_Y\}_{\mathsf{Act}}\rangle$$
>
> between transition systems with the same labels is as a function $f : X \to Y$ between their states such that if $x \xrightarrow{a}_X x'$ then $f(x) \xrightarrow{a}_Y f(x')$. Instead, regarding a transition system as a coalgebra, one has the extra condition that the function f must be such that if $f(x) \xrightarrow{a}_Y y$ for some state $y \in Y$, then there exists a state $x' \in X$ such that $x \xrightarrow{a}_X x'$.
>
> Therefore, a category of transition systems is different from the category of coalgebras of the corresponding behaviour. In particular, the universals in the two categories will be different. For instance, while the product of two transition system always exists, the product of two coalgebras does not necessarily exist. Also, the final transition system is different from the final coalgebra. (The latter is an object which enjoys very important semantical properties – cf Section 5.)

The above behaviours, whose coalgebras correspond to various forms of labelled transition systems, are suitable for modelling imperative and concurrent languages. Instead, for modelling applicative languages, one needs behaviours involving some form of function space functor. An example is the endofunctor

$$BX = 1 + X^Y$$

The 'exponent' X^Y is the set of functions from Y to X. In order to avoid the usual 'mixed variance' problems, Y is here treated as a parameter. By putting $Y = X$ one obtains that the corresponding coalgebras are the *quasi-applicative transition systems* defined in [Abr90]. The 'exception' 1 in the above behaviour can be used to encode non-termination.

For example, for X and Y both equal to the set Λ of closed λ-*terms*, one can define a coalgebra structure

$$ev : \Lambda \to 1 + \Lambda^{\Lambda}$$

by putting, for every λ-term $M \in \Lambda$,

$$ev(M) = P \mapsto N[P/x]$$

if M converges to 'principal weak head normal form' $\lambda x\,.N$, and

$$ev(M) = *$$

otherwise.

Back now to deterministic transition systems and the corresponding behaviour $BX = 1 + \mathsf{Act} \times X$. Recall that the rules $\mathcal{R}$ for the above sample language induce a coalgebra

$$[\![\text{-}]\!]_{\mathcal{R}} : T0 \to BT0$$

This can be seen as a special case of a general construction which, starting from a coalgebra (ie deterministic transition system) structure $k : X \to BX$, yields a new coalgebra structure

$$[\![\text{-}]\!]^{k}_{\mathcal{R}} : TX \to BTX$$

with the set of terms TX as carrier and which 'conservatively extends' the original structure k.

Indeed, one can add, for every $x \in X$, the value of $k(x)$ as an axiom to the rules in $\mathcal{R}$ that is, if $k(x) = < a, x >$ then add $x \xrightarrow{a} x'$ to $\mathcal{R}$ and if $k(x) = *$ then add $x \downarrow *$. The least transition system induced by these extended rules will have then TX as set of states and be deterministic, hence it can be regarded as a coalgebra with structure $[\![\text{-}]\!]^{k}_{\mathcal{R}} : TX \to 1 + \mathsf{Act} \times TX$. By structural induction, one can prove that this induced transition system/coalgebra **conservatively extends** the coalgebra/transition system $\langle X, k\rangle$ in the sense that, for every $x \in X$,

$$k(x) = [\![x]\!]^{k}_{\mathcal{R}}$$

Formally, recalling that $\eta_X : X \to TX$ is the insertion-of-variables function which permits to see the elements $x \in X$ as variable terms in TX, the above conservative extension property amounts to the commutativity of the following diagram.

$$\begin{tikzcd}
X \arrow[r, "\eta_X"] \arrow[d, "k"'] & TX \arrow[d, "{[\![\text{-}]\!]^{k}_{\mathcal{R}}}"] \\
BX \arrow[r, "B\eta_X"'] & BTX
\end{tikzcd}$$

That is, the function $\eta_X : X \to TX$ 'lifts' to a coalgebra arrow

$$\eta_X : \langle X, k\rangle \to \langle TX, [\![\text{-}]\!]^{k}_{\mathcal{R}}\rangle$$

for every coalgebras structure k on X.

Notes. The importance of the correspondence between labelled transition systems and coalgebras of the behaviour $BX = \mathcal{P}(\mathsf{Act} \times X)$ has been stressed by Peter Aczel in [Acz88]. (But see also [Ken87] and [Hes88].) For a comprehensive categorical (but not coalgebraic!) treatment of labelled transition systems see [WN95].

As mentioned in the introduction, it would be interesting to sort out the relationship between the present notion of *behaviour* as an *endofunctor* whose coalgebras are operational models and Eugenio Moggi's notion of *computation* as a *monad* [Mog91]. The examples of *computational monads* given in [Mog91] (partiality, non-determinism, side-effects, exceptions, etc) all qualify as behaviours, and the corresponding monadic operations could play an important rôle in further developments. (The operations of the (finite) non-determinism monad $\mathcal{P}_{\mathrm{fi}}$ are already used in Sections 10 and 11.)

4 Functorial Operational Semantics

In this section, a new approach to operational semantics, based on categorical notions of *syntax* and *behaviour*, is introduced: an operational semantics is *functorial* when it is a 'lifting' of the syntactical monad T to the coalgebras of the behaviour endofunctor B.

Inductively, this can be obtained by defining an 'action' of the program constructs on the composite functor BT; as an instance, the operational rules of a simple deterministic language are shown to define such an action. More generally, a functorial operational semantics can be obtained by defining a 'distributive law' of the syntactical monad T over the behaviour functor B.

Given a syntactical monad T and a behaviour endofunctor B on the same category, a **functorial operational semantics** wrt T and B is a 'lifting' of the monad T to the B- coalgebras.

In general, let $U : \mathbf{C}_B \to \mathbf{C}$ be the forgetful functor mapping coalgebras $\langle X, k\rangle$ to their carriers X. Then, a **lifting** of a monad $T = <T, \eta, \mu>$ to the coalgebras of an endofunctor B on the same category $\mathbf{C}$ is a monad Φ such that the diagram

$$\begin{tikzcd}
\mathbf{C}_B \arrow[r, "\Phi"] \arrow[d, "U"'] & \mathbf{C}_B \arrow[d, "U"] \\
\mathbf{C} \arrow[r, "T"'] & \mathbf{C}
\end{tikzcd}$$

commutes, making $U : \mathbf{C}_B \to \mathbf{C}$ a 'map of monads'. That is, Φ is a triple $<\Phi, \tilde{\eta}, \tilde{\mu}>$ such that

$$\begin{array}{rcl}
U\Phi & = & TU : \mathbf{C}_B \to \mathbf{C} \\
U\tilde{\eta} & = & \eta_U : U \Rightarrow TU \\
U\tilde{\mu} & = & \mu_U : T^2U \Rightarrow TU
\end{array}$$

The second and third equation imply that the unit $\tilde{\eta}$ and multiplication $\tilde{\mu}$ of Φ are the same as the unit η and multiplication μ of $T = <T, \eta, \mu>$, because of the very definition of coalgebra arrows. Therefore:

$$\Phi = <\Phi, \eta, \mu>$$

One can check that the three equations and the fact that the triple $T = < T, \eta, \mu >$ is a monad imply that also the triple $\Phi = < \Phi, \eta, \mu >$ is a monad.

Let us now look at the endofunctor Φ. The equation $U\Phi = TU$ implies that Φ is completely determined by its action on the structure of coalgebras, that is, on the arrow $k : X \to BX$ in a coalgebra $\langle X, k \rangle$:

$$\frac{X \xrightarrow{\ k\ } BX}{TX \xrightarrow[\ \Phi k\ ]{} BTX}$$

Indeed, by the definition of coalgebra arrows, the action of Φ on arrows is the same as the one of T:

$$\begin{tikzcd}
X \arrow[r, "f"] \arrow[d, "k"'] & X' \arrow[d, "k'"] & & TX \arrow[r, "Tf"] \arrow[d, "\Phi k"'] & TX' \arrow[d, "\Phi k'"] \\
BX \arrow[r, "Bf"'] & BX' & \arrow[r, mapsto, "\Phi"] & BTX \arrow[r, "BTf"'] & BTX'
\end{tikzcd}$$

Rewriting the above action as

$$\langle X, k \rangle \xrightarrow{\ f\ } \langle X', k' \rangle$$

$$\begin{tikzcd}
TU\langle X, k \rangle \arrow[r, "TUf"] \arrow[d, "\Phi\langle X, k \rangle"'] & TU\langle X', k' \rangle \arrow[d, "\Phi\langle X', k' \rangle"] \\
BTU\langle X, k \rangle \arrow[r, "BTUf"'] & BTU\langle X', k' \rangle
\end{tikzcd}$$

shows the following correspondence.

> **Liftings as Coactions.** A lifting of an *endofunctor* T to the B-coalgebras, that is, an *endofunctor* Φ such that $U\Phi = TU$, is the same as a **coaction** of B on the composite functor $TU : \mathbf{C}_B \to \mathbf{C}$, that is, a natural transformation
>
> $$TU \Rightarrow BTU$$

Finally, the conditions $U\widetilde{\eta} = \eta_U$ and $U\widetilde{\mu} = \mu_U$ amount to say that η and μ lift to natural transformations in the B-coalgebras. That is, for every coalgebra $\langle X, k \rangle$, the two squares in the following diagram commute.

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

Inductive Functorial Operational Semantics

An *inductive* way of defining a functorial operational semantics is by specifying the **action** of the program constructs Σ on the 'observables' BT of the language, that is, by giving a natural transformation

$$\phi : \Sigma BT \Rightarrow BT$$

Indeed, for every B-coalgebra $\langle X, k\rangle$, the Σ-algebra structure $\phi_X = \phi_{U\langle X,h\rangle}$: $\Sigma(BTX) \to BTX$ on BTX can be inductively extended along the composite $B\eta_X \circ k$ to a coalgebra structure $\widehat{\phi}(k) : TX \to BTX$

$$\begin{tikzcd}
X \arrow[r, "\eta_X"] \arrow[d, "k"'] & TX \arrow[d, dashed, "\widehat{\phi}(k)\ ="', "(B\eta_X \circ k)^\sharp"] & \Sigma TX \arrow[l, "\mathsf{inr}_X"'] \arrow[d, "\Sigma\widehat{\phi}(k)"] \\
BX \arrow[r, "B\eta_X"'] & BTX & \Sigma BTX \arrow[l, "\phi_X"]
\end{tikzcd}$$

By the naturality of ϕ, this definition is natural in $\langle X, k\rangle$, that is,

$$\widehat{\phi} : TU \Rightarrow BTU$$

thus $\widehat{\phi}$ can be seen as an endofunctor (with the same name) on the B-coalgebras.

Moreover, the triple $<\widehat{\phi}, \eta, \mu>$ – where, recall, η and μ are the unit and multiplication of the term monad T – is a monad on the B-coalgebras, that is, the two squares in

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

commute. Indeed, the square corresponding to the unit η commutes by definition, while the one corresponding to the multiplication μ commutes because both composites $\widehat{\phi}(k) \circ \mu_X$ and $B\mu_X \circ \widehat{\phi}^2(k)$ fit as the (unique!) inductive extension of ϕ_X along $\widehat{\phi}(k)$

$$\begin{tikzcd}
TX \arrow[r, "\eta_{TX}"] \arrow[d, "\widehat{\phi}(k)"'] & T^2X \arrow[dd, dashed, "\widehat{\phi}(k)^\sharp"] & \Sigma T^2X \arrow[l, "\mathsf{inr}_{TX}"'] \arrow[dd, "\Sigma\widehat{\phi}(k)^\sharp"] \\
BTX \arrow[dr, equal] & & \\
& BTX & \Sigma BTX \arrow[l, "\phi_X"]
\end{tikzcd}$$

because

$$\begin{tikzcd}
TX \arrow[r, "\eta_{TX}"] \arrow[d, "\widehat{\phi}(k)"'] & T^2X \arrow[d, "\widehat{\phi}^2(k)"] & \Sigma T^2X \arrow[l, "\mathsf{inr}_{TX}"'] \arrow[d, "\Sigma\widehat{\phi}^2(k)"] \\
BTX \arrow[r, "B\eta_{TX}"] \arrow[dr, equal] & BT^2X \arrow[d, "B\mu_X"] & \Sigma BT^2X \arrow[l, "B\mathsf{inr}_{TX}"'] \arrow[d, "\Sigma B\mu_X"] \\
& BTX & \Sigma BTX \arrow[l, "\phi_X"]
\end{tikzcd}
\qquad
\begin{tikzcd}
TX \arrow[r, "\eta_{TX}"] \arrow[d, "\widehat{\phi}(k)"'] \arrow[dr, equal] & T^2X \arrow[d, "\mu_X"] & \Sigma T^2X \arrow[l, "\mathsf{inr}_{TX}"'] \arrow[d, "\Sigma\mu_X"] \\
BTX \arrow[dr, equal] & TX \arrow[d, "\widehat{\phi}(k)"] & \Sigma TX \arrow[l, "\mathsf{inr}_X"'] \arrow[d, "\Sigma\widehat{\phi}(k)"] \\
& BTX & \Sigma BTX \arrow[l, "\phi_X"]
\end{tikzcd}$$

Some terminology: in the sequel, a functorial operational semantics Φ is also called the **operational monad** Φ and the natural transformation $\phi : \Sigma BT \Rightarrow BT$ inducing the operational monad $\widehat{\phi}$ is called the **germ** of $\widehat{\phi}$.

Operational Rules and Inductive Functorial Operational Semantics

Now the claim is that the operational rules $\mathcal{R}$ given in the previous section for the simple deterministic language $t ::= x \mid a \mid \mathsf{nil} \mid (t\ ;\ t)$ can be regarded as a natural transformation

$$\lceil \mathcal{R} \rceil : \Sigma B \Rightarrow BT$$

Moreover, by taking the composite $B\mu \circ \lceil \mathcal{R} \rceil_T : \Sigma BT \Rightarrow BT$ one obtains the germ of an inductive functorial operational semantics which is 'observationally equivalent' to the operational semantics induced by the rules $\mathcal{R}$. (This result is generalized in Section 11 to the large class of 'GSOS' operational rules, which are suitable to model most of imperative and concurrent programming languages.)

Recall that the algebras of the signature $\Sigma = \mathsf{Act} \cup \{\mathsf{nil},\ ;\ \}$ for the above language are the same as the algebras of the endofunctor

$$\Sigma X = 1 + \mathsf{Act} + X \times X$$

on **Set** and that the programs t are the elements of TX, the carrier of the free Σ-algebra on X. Also, recall that the operational semantics induced by the rules $\mathcal{R}$ of the language is a deterministic transition system and that there is a one-to-one correspondence between deterministic transition systems and coalgebras of the endofunctor

$$BX = 1 + \mathsf{Act} \times X$$

This correspondence says that a transition $x \xrightarrow{a} x'$ of a deterministic transition system can be seen as the action $x \mapsto\ <a, x'>$ of a coalgebra structure $X \to BX$; similarly, the action $x \mapsto *$ corresponds to the fact that $x \downarrow *$ holds. Thus the operational rules $\mathcal{R}$ given in the previous section can be written as follows.

$$\mathsf{nil} \mapsto * \qquad\qquad a \mapsto\ <a, \mathsf{nil}>$$

$$\frac{u \mapsto\ <a, u'>}{u\ ;\ v \mapsto\ <a, u'\ ;\ v>} \qquad \frac{u \mapsto * \quad v \mapsto\ <a, v'>}{u\ ;\ v \mapsto\ <a, v'>} \qquad \frac{u \mapsto * \quad v \mapsto *}{u\ ;\ v \mapsto *}$$

Next, let us define the natural transformation $\lceil \mathcal{R} \rceil : \Sigma B \Rightarrow BT$. Let r and s be meta-variables ranging over elements of $BX = 1 + \mathsf{Act} \times X$, for arbitrary sets of variables X. One has to define the value of $\lceil \mathcal{R} \rceil_X$ at nil, at a, and at $r\ ;\ s$, for all r, s. Omitting the subscript X, put

$$\lceil \mathcal{R} \rceil(\mathsf{nil}) = * \qquad \text{and} \qquad \lceil \mathcal{R} \rceil(a) = \ <a, \mathsf{nil}>$$

For sequential composition there are three cases to be considered, namely

1. $r = \ <a, x>$
2. $r = *$ and $s = \ <a, y>$

3. $r = *$ and $s = *$

In the second and third case one can follow the definition of $\mathcal{R}$ and put $<a, y>$ and $*$, respectively, for the value of $\lceil\mathcal{R}\rceil$ at $r\ ;\ s$. Instead, in the first case, one cannot put simply $< a, x\ ;\ s >$ because $x\ ;\ s$ is not of type T. The problem is that s is of type B rather than T. But notice that B can be *embedded* in T:

> **The embedding γ of the behaviour into the syntax.** The action
>
> $$* \mapsto \mathsf{nil} \qquad <a, x> \mapsto a\ ;\ x$$
>
> defines an *injective* function from BX to TX, for every set X. It is manifestly natural in X; call it
>
> $$\gamma : B \Rightarrow T$$

One can then put

$$\lceil\mathcal{R}\rceil(r\ ;\ s) \;=\; \begin{cases} <a, x\ ;\ \gamma s> & \text{if } r = <a, x> \\ <a, y> & \text{if } r = * \text{ and } s = <a, y> \\ * & \text{if } r = * = s \end{cases}$$

Altogether, in a more suggestive notation:

$$\frac{\mathsf{nil} \overset{\lceil\mathcal{R}\rceil}{\longmapsto} * \qquad r = <a, x>}{r\ ;\ s \overset{\lceil\mathcal{R}\rceil}{\longmapsto} <a, x\ ;\ \gamma s>} \qquad \frac{a \overset{\lceil\mathcal{R}\rceil}{\longmapsto} <a, \mathsf{nil}> \qquad r = * \quad s = <a, y>}{r\ ;\ s \overset{\lceil\mathcal{R}\rceil}{\longmapsto} <a, y>} \qquad \frac{r = * \quad s = *}{r\ ;\ s \overset{\lceil\mathcal{R}\rceil}{\longmapsto} *}$$

This definition yields a natural transformation

$$\lceil\mathcal{R}\rceil : \Sigma B \Rightarrow BT$$

> Indeed, the only problematic clause for the naturality of $\lceil\mathcal{R}\rceil$ is $\lceil\mathcal{R}\rceil(r\ ;\ s)$ for $r = <a, x>$. One has to show that, for every 'renaming' $f : X \to Y$, the following holds.
>
> $$\begin{tikzcd} <a, x> ;\ s \arrow[r, mapsto, "\Sigma B f"] \arrow[d, mapsto, "\lceil\mathcal{R}\rceil_X"'] & <a, fx> ;\ (Bf)(s) \arrow[d, mapsto, "\lceil\mathcal{R}\rceil_Y"] \\ <a, x\ ;\ \gamma_X s> \arrow[r, mapsto, "BTf"'] & <a, (Tf)(x\ ;\ \gamma_X s)> = <a, fx\ ;\ \gamma_Y (Bf)(s)> \end{tikzcd}$$
>
> That is,
>
> $$(Tf)(\gamma_X s) = \gamma_Y (Bf)(s)$$
>
> But this is immediate from the fact that γ is a natural transformation from B to T.

(As shown in Section 11, the argument in the above proof generalizes to any (possibly non-deterministic) rule in the 'GSOS-format'.)

Next, consider the germ of the functorial operational semantics corresponding to $\mathcal{R}$. It is essentially the same as $\lceil\mathcal{R}\rceil$, only it is applied to terms, hence the multiplication μ of the syntactical monad T is needed in order to remove brackets from the resulting terms of terms to yield simple terms. Thus $\phi^{\mathcal{R}} = B\mu \circ \lceil\mathcal{R}\rceil_T : \Sigma BT \Rightarrow BT$, that is,

$$\begin{tikzcd} \Sigma BT \arrow[r, Rightarrow, "\lceil\mathcal{R}\rceil_T"] \arrow[dr, Rightarrow, "\phi^{\mathcal{R}}"'] & BT^2 \arrow[d, Rightarrow, "B\mu"] \\ & BT \end{tikzcd}$$

Therefore:

$$\mathsf{nil} \stackrel{\phi^{\mathcal{R}}}{\longmapsto} * \qquad a \stackrel{\phi^{\mathcal{R}}}{\longmapsto} <a, \mathsf{nil}>$$

$$\frac{r = <a,t>}{r\ ;\ s \stackrel{\phi^{\mathcal{R}}}{\longmapsto} <a, t\ ;\ \gamma s>} \qquad \frac{r = * \quad s = <a,t>}{r\ ;\ s \stackrel{\phi^{\mathcal{R}}}{\longmapsto} <a,t>} \qquad \frac{r = * \quad s = *}{r\ ;\ s \stackrel{\phi^{\mathcal{R}}}{\longmapsto} *}$$

The resulting $\phi^{\mathcal{R}} : \Sigma BT \Rightarrow BT$ is the germ of a functorial operational semantics. In particular, consider the case of closed terms $T0$ and write

$$[\![\text{-}]\!]_{\lceil\mathcal{R}\rceil} : T0 \to BT0$$

for the operational model obtained by taking the inductive extension of $\phi_0^{\mathcal{R}} : \Sigma BT0 \to BT0$

$$\begin{tikzcd} \Sigma T0 \arrow[r, "\Sigma\, [\![\text{-}]\!]_{\lceil\mathcal{R}\rceil}"] \arrow[d, "\cong"'] & \Sigma BT0 \arrow[d, "\phi_0^{\mathcal{R}}"] \\ T0 \arrow[r, dashed, "{[\![\text{-}]\!]_{\lceil\mathcal{R}\rceil}}"'] & BT0 \end{tikzcd}$$

Then, by definition,

$$[\![u\ ;\ v]\!]_{\lceil\mathcal{R}\rceil} = \begin{cases} <a, u'\ ;\ \gamma[\![v]\!]_{\lceil\mathcal{R}\rceil}> & \text{if } [\![u]\!]_{\lceil\mathcal{R}\rceil} = <a, u'> \\ <a, v'> & \text{if } [\![u]\!]_{\lceil\mathcal{R}\rceil} = * \text{ and } [\![v]\!]_{\lceil\mathcal{R}\rceil} = <a, v'> \\ * & \text{if } [\![u]\!]_{\lceil\mathcal{R}\rceil} = * = [\![v]\!]_{\lceil\mathcal{R}\rceil} \end{cases}$$

Contrast this with the operational model

$$[\![\text{-}]\!]_{\mathcal{R}} : T0 \to BT0$$

'directly' induced by the rules $\mathcal{R}$ on the closed terms: they are the same, except for

$$[\![u\,;\,v]\!]_{\mathcal{R}} = <a, u'\,;\,v> \qquad \text{if } [\![u]\!]_{\mathcal{R}} = <a, u'>$$

In Section 6 it is shown that for every term v, the term $\gamma[\![v]\!]_{\lceil\mathcal{R}\rceil}$ exhibits the same 'observable behaviour' as v, under any context. Therefore, the two models $[\![\text{-}]\!]_{\mathcal{R}}$ and $[\![\text{-}]\!]_{\lceil\mathcal{R}\rceil}$ are 'observationally equivalent'. This is based on the fact that the above natural transformation $\gamma : B \Rightarrow T$ is a 'retraction' for the operational semantics induced by $\mathcal{R}$. More precisely, for every coalgebra structure $k : X \to BX$, the composite arrow $\mu_X \circ \gamma_{TX} : BTX \to TX$ is a right inverse for the operational model $[\![\text{-}]\!]^k_{\mathcal{R}} : TX \to BTX$ induced by $\mathcal{R}$. Indeed, omitting, as usual, the multiplication μ,

$$\gamma_{TX}(*) \;=\; \mathsf{nil} \overset{\mathcal{R}}{\longmapsto} * \qquad \text{and} \qquad \gamma_{TX}(<a, t>) \;=\; a\,;\,t \overset{\mathcal{R}}{\longmapsto} <a, t>$$

hence

$$[\![\gamma_{TX}(*)]\!]^k_{\mathcal{R}} = * \qquad \text{and} \qquad [\![\gamma_{TX}(<a, t>)]\!]^k_{\mathcal{R}} = <a, t>$$

Semantics as a Distributive Law

The germ $\phi : \Sigma BT \Rightarrow BT$ of an inductive functorial operational semantics $\widehat{\phi}$ defines a 'distributive law' $\phi^\# : TB \Rightarrow BT$ of the syntactical monad T over the behaviour B. The operational monad $\widehat{\phi}$ can be then decomposed in terms of this distributive law and of T itself. In turn, every distributive law $\lambda : TB \Rightarrow BT$ defines a lifting of the monad T to the B-coalgebras.

In general, a **distributive law** of a monad $T = <T, \eta, \mu>$ over an endofunctor B (on the same category) is a natural transformation

$$\lambda : TB \Rightarrow BT$$

such that the following two diagrams commute.

$$\begin{tikzcd} & B \arrow[dl, Rightarrow, "\eta_B"'] \arrow[dr, Rightarrow, "B\eta"] & \\ TB \arrow[rr, Rightarrow, "\lambda"'] & & BT \end{tikzcd} \qquad \begin{tikzcd} T^2B \arrow[r, Rightarrow, "T\lambda"] \arrow[d, Rightarrow, "\mu_B"'] & TBT \arrow[r, Rightarrow, "\lambda_T"] & BT^2 \arrow[d, Rightarrow, "B\mu"] \\ TB \arrow[rr, Rightarrow, "\lambda"'] & & BT \end{tikzcd}$$

Every distributive law $\lambda : TB \Rightarrow BT$ defines an endofunctor lifting T to the B-coalgebras by mapping a coalgebra $\langle X, k\rangle$ to the coalgebra $\langle TX, \lambda \circ Tk\rangle$:

$$\frac{X \xrightarrow{k} BX}{TX \xrightarrow[Tk]{} TBX \xrightarrow[\lambda_X]{} BTX}$$

Moreover, this is a lifting of the whole monad $T = <T, \eta, \mu>$ to the B-coalgebras, because everything in sight in the following diagram commutes (either by the naturality of η and μ or by distributivity).

$$\begin{tikzcd} X \arrow[r, "\eta_X"] \arrow[dddd, "k"'] & TX \arrow[dd, "Tk"'] & T^2X \arrow[l, "\mu_X"'] \arrow[d, "T^2k"] \\ & & T^2BX \arrow[dl, "\mu_{BX}"'] \arrow[dd, "T\lambda_X"] \\ & TBX \arrow[dd, "\lambda_X"'] & \\ & & TBTX \arrow[d, "\lambda_{TX}"] \\ BX \arrow[r, "B\eta_X"'] \arrow[uur, "\eta_{BX}"] & BTX & BT^2X \arrow[l, "B\mu_X"] \end{tikzcd}$$

A distributive laws can be defined from a germ $\phi : \Sigma BT \Rightarrow BT$ by taking the inductive extension

$$\phi^\# = (B\eta)^\sharp : TB \Rightarrow BT$$

of the germ ϕ along the natural transformation $B\eta : B \Rightarrow BT$.

$$\begin{tikzcd}
B \arrow[r, Rightarrow, "\eta_B"] \arrow[dr, Rightarrow, "B\eta"'] & TB \arrow[d, Rightarrow, dashed, "\phi^{\#} = (B\eta)^{\sharp}"] & \Sigma TB \arrow[l, Rightarrow, "\mathsf{inr}_B"'] \arrow[d, Rightarrow, "\Sigma\phi^{\#}"] \\
& BT & \Sigma BT \arrow[l, Rightarrow, "\phi"]
\end{tikzcd}$$

Indeed, the left triangle shows that $\phi^{\#}$ satisfies the first of the two conditions for being a distributive law. To prove the second, one can show that both composites $\phi^{\#} \circ \mu_B$ and $B\mu \circ \phi^{\#}_T \circ T\phi^{\#}$ fit as the unique inductive extension of ϕ along $\phi^{\#}$. (This is very much the same as the above proof that μ lifts to a multiplication for the inductive functorial operational semantics $\widehat{\phi}$.)

Notice that then the action of the inductive operational monad $\widehat{\phi}$ on a coalgebra $\langle X, k\rangle$ can be decomposed into the action of the syntactical monad T on the structure k, followed by the distributive law $\phi^{\#}$ at the carrier X:

$$\widehat{\phi}\,\langle X, k\rangle = \phi^{\#}_X \circ Tk$$

Notes. The notion of a distributive law of a monad over an endofunctor is derived from the more familiar notion of a distributive law of a monad T_1 over another monad T_2 introduced in [Bec69]. In that paper, the equivalence is proved between distributive laws of the monad T_1 over the monad T_2, liftings of the monad T_2 to the T_1-algebras, and actions of the monad T_1 over the functor $T_2U^{T_1}$. (See also [BW85], Chapter 9.) Here monads are lifted to coalgebras (of a functor) rather than to algebras (of a monad) and this gives a slightly different situation, with a monad distributing over a functor (and with distributive laws implying liftings but not vice versa). More symmetry is gained in Section 7 by considering the *comonad* D cofreely generated by the behaviour B.

5 Recursive Behaviours, Final Coalgebras and Coinduction

The rôle of final coalgebras is dual to the one played by initial algebras, and dual are their properties and constructions. For instance, as initial algebras account for induction, final coalgebras account for the dual notion of 'coinduction', which is useful to deal with the behaviour of recursive programs. Also, as the programs of a language may be described as the initial algebra of a signature Σ, the abstract global behaviours – the 'processes' – of a language may be described as the final coalgebra of a behaviour B.

Let B be an endofunctor which has a **final coalgebra** (ie the final object in the corresponding category of coalgebras) and let $\widehat{B}$ denote the carrier of this final coalgebra. The structure of a final coalgebra is, like that of an initial algebra, an isomorphism, because the notion of isomorphism is 'self-dual'. Thus:

$$\widehat{B} \cong B\widehat{B} \qquad\qquad \text{(final } B\text{-coalgebra)}$$

Any coalgebra structure $k : X \to BX$ can be 'coinductively' extended to an arrow $k^@ : X \to \widehat{B}$ by taking the unique coalgebra arrow from the coalgebra $\langle X, k\rangle$ to the final coalgebra:

Coinductive Extension

$$\begin{tikzcd} X \arrow[r, dashed, "k^@"] \arrow[d, "k"'] & \widehat{B} \arrow[d, "\cong"] \\ BX \arrow[r, "Bk^@"'] & B\widehat{B} \end{tikzcd}$$

Of particular interest are the coinductive extensions of operational models. In order to illustrate this, let us consider languages, like the one in Section 3, which have an operational semantics yielding deterministic transition systems, that is, coalgebras of the behaviour endofunctor

$$BX = 1 + \mathsf{Act} \times X$$

on **Set**. Thus, if T is the syntactical monad for the language, an operational model is a coalgebra with structure

$$[\![\text{-}]\!] : TX \to BTX$$

where X is the set of variables of the language. Then, under the assumption that the final B-coalgebra exists, the coinductive extension of this coalgebra structure yields the

Final Coalgebra Semantics

$$\begin{tikzcd} TX \arrow[r, dashed, "[\![\text{-}]\!]^{@}"] \arrow[d, "[\![\text{-}]\!]"'] & \widehat{B} \arrow[d, "\cong"] \\ BTX \arrow[r, "B[\![\text{-}]\!]^{@}"'] & B\widehat{B} \end{tikzcd}$$

of the language. Since $BTX = 1 + \mathsf{Act} \times TX$, this yields, for any term t, the following definition.

$$[\![t]\!]^{@} = \begin{cases} * & \text{if } [\![t]\!] = * \\ <a, [\![t']\!]^{@}> & \text{if } [\![t]\!] = <a, t'> \end{cases}$$

(Notice that the isomorphism $\widehat{B} \cong B\widehat{B}$ has been treated as an equality in order to simplify the notation.) Thus, for instance, wrt the operational model $[\![\text{-}]\!]$ given in Section 3, the programs $a\ ;\ b$ and $a\ ;\ \mathsf{nil}\ ;\ b$ have the same final coalgebra semantics:

$$[\![a\ ;\ b]\!]^{@} = <a, b, *> = [\![a\ ;\ \mathsf{nil}\ ;\ b]\!]^{@}$$

In general, under this final coalgebra semantics, a program is mapped into the stream of actions that it can perform.

Next, consider the construction of the final coalgebra for the above endofunctor $BX = 1 + \mathsf{Act} \times X$. This is an $\boldsymbol{\omega}^{\mathrm{op}}$-continuous endofunctor, that is, it preserves limits of functors from $\boldsymbol{\omega}^{\mathrm{op}} = \{0 \leftarrow 1 \leftarrow 2 \leftarrow \cdots\}$. Indeed, it is made of constants, a product, and a coproduct: constants and products (like all limit functors) are $\boldsymbol{\omega}^{\mathrm{op}}$-continuous in every category; finite coproducts are $\boldsymbol{\omega}^{\mathrm{op}}$-continuous in **Set**, by the dual of a theorem [Mac71, Theorem IX.2.1] mentioned in Section 1. By further dual considerations, the carrier of the final coalgebra of an $\boldsymbol{\omega}^{\mathrm{op}}$-continuous endofunctor B is the limit $\widehat{B}$ of the following diagram obtained by iterative applications of B to the unique function from $B1$ to the singleton set 1, the final object in **Set**.

$$1 \xleftarrow{1_{B1}} B1 \xleftarrow{B1_{B1}} B^2 1 \xleftarrow{B^2 1_{B1}} \cdots$$

The isomorphism $\varphi : \widehat{B} \cong B\widehat{B}$ giving the coalgebra structure is obtained as a mediating arrow just like in the initial algebra construction.

This general construction of final coalgebras of $\boldsymbol{\omega}^{\mathrm{op}}$-continuous endofunctors yields, in the particular case considered here, the final coalgebra with carrier the set

$$\mathsf{Act}^{\infty} = \coprod_{\alpha \leq \omega} \mathsf{Act}^{\alpha}$$

of finite ($\alpha = n$) and infinite ($\alpha = \omega$) streams of actions generated by Act, and with structure the isomorphism

$$\varphi : \mathsf{Act}^{\infty} \cong 1 + \mathsf{Act} \times \mathsf{Act}^{\infty}$$

This isomorphism is an operation which allows one to explore the streams $w \in \mathsf{Act}^{\infty}$: if $w = \epsilon$, the empty stream, then $\varphi(w) = *$, otherwise $w = a \cdot w'$ and $\varphi(w) = <a, w'>$, that is, φ applied to a non-empty stream returns the first element of the stream plus its continuation. Also notice that its inverse $\varphi^{-1} : 1 + \mathsf{Act} \times \mathsf{Act}^{\infty} \cong \mathsf{Act}^{\infty}$ is a *B-algebra* structure; it gives the empty stream constant $\epsilon = \varphi^{-1}(*)$ and the prefixing operators $a \cdot \text{-} = \varphi^{-1}(a, \text{-})$, and the identity $a \cdot \epsilon = a$ follows from the fact that $\mathsf{Act} \times 1 \cong \mathsf{Act}$.

Next, the unique coalgebra arrow from a B-coalgebra $\langle X, k\rangle$ to $\langle \mathsf{Act}^{\infty}, \varphi\rangle$ is defined as follows. Let $\langle X, \{\xrightarrow{a}\}_{Act}\rangle$ be the deterministic transition system corresponding to the coalgebra $\langle X, k\rangle$. (Cf Section 3.) Then, for every $x \in X$, consider the **global behaviour** of the state x in the transition system: there are three possibilities, namely either (i) the state x is inert, or (ii) the system performs a finite sequence

$$x \xrightarrow{a_1} x_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} x_n$$

of transitions starting from the state x and then reaches an inert state x_n, or (iii) the system performs an infinite sequence

$$x \xrightarrow{a_1} x_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} x_n \xrightarrow{a_{n+1}} \cdots$$

of transitions, never reaching an inert state. Correspondingly, define the function $k^{@} : X \to \mathsf{Act}^{\infty}$ by putting, for every $x \in X$,

$$k^{@}(x) \;=\; \begin{cases} * & \text{if } \; (i) \\ <a_1, a_2, \ldots, a_n, *> & \text{if } \; (ii) \\ <a_1, a_2, \ldots, a_n, a_{n+1}, \ldots> & \text{if } \; (iii) \end{cases}$$

One can check this is the desired unique coalgebra arrow from $\langle X, k\rangle$ to $\langle \mathsf{Act}^{\infty}, \varphi\rangle$.

Thus the coinductive extension of a coalgebra structure is defined in terms of the global behaviours in the corresponding transition system. The carrier of the final coalgebra itself is the set of all possible '*abstract global behaviours*' wrt B, in which the name of the states is irrelevant. In other words, streams are global behaviours of deterministic transition systems with a single state.

Notice that, taking the behaviour $BX = 1 + \mathsf{Act} \times X$ in the category **pSet** of sets and partial functions rather than in **Set**, the (carrier of the) final B-coalgebra in **pSet** does not contain infinite steams but only the finite ones. Indeed, using partial functions, the coinductive extension of a state having an infinite global behaviour can be left undefined.

Now, the set of finite streams is the carrier of the *initial B-algebra*, both in **Set** and **pSet**. Similarly, the set of natural numbers $N \cong 1 + N$ is both the carrier of the initial algebra and of the final coalgebra of the endofunctor $X \mapsto 1 + X$ on **pSet**, while in **Set** the final coalgebra needs an extra infinity point ∞. This fact generalizes to all functors $X \mapsto \coprod_{\sigma \in \Sigma} X^{\mathrm{ar}(\sigma)}$ corresponding to signatures Σ.

Guarded Recursion. So far, the operational interpretation of the sample language

$$t ::= x \mid a \mid \mathsf{nil} \mid (t\ ;\ t)$$

yields global behaviours which are always *finite*. In order to obtain *infinite* global behaviours, let us use the variables $x \in X$ of the language and define *recursive programs* as solutions of 'term-equations' like

$$x = a\ ;\ x$$

Intuitively, the solution of the above equation should be a program having as abstract global behaviour the infinite stream a^ω.

In general, not all term-equations have solutions which can be interpreted as streams. For instance, the equation

$$x = x\ ;\ x$$

should have as solution a program which keeps on unfolding itself

$$x \longrightarrow x\ ;\ x \longrightarrow x\ ;\ x\ ;\ x\ ;\ x \longrightarrow \dots$$

never performing any action. In order to rule out this kind of equation one usually considers only recursive definitions which are 'guarded', that is, equations $x = t$ in which t is of the form $a\ ;\ t'$.

Operationally, the above presentation of recursive programs can be made formal by introducing a *fixed point* binding operator fix: given a variable x and a 'guarded' term $t = a\ ; t'$, the expression $\mathsf{fix}x.t$ is then a term with operational behaviour described by the rule

$$\frac{t[\mathsf{fix}x.t/x] \xrightarrow{a} u}{\mathsf{fix}x.t \xrightarrow{a} u}$$

in which the expression $t[\mathsf{fix}x.t/x]$ stands for the term obtained by substituting the term $\mathsf{fix}x.t$ for every occurrence of x in t.

One of the advantages and novelties of the present functorial approach to operational semantics is that it allows for an elegant operational description of recursive programs which, quite surprisingly, does not require the introduction of a binding operator like the above fix (at least for 'top-level' recursive definitions). Moreover, it allows for a general formal description of guarded recursion, independent of the use of actions and transitions.

Firstly, every *system* of term-equations

$$\begin{array}{rcl} x_1 & = & t_1 \\ x_2 & = & t_2 \\ & \vdots & \end{array}$$

with $x_i \in X$ and $t_i \in TX$, can be seen as a *coalgebra* of the syntax T having as carrier the set $X = \{x_1, x_2, \ldots\}$ of variables appearing in the system and as structure the function

$$k : X \to TX \qquad\qquad x_i \mapsto t_i$$

The generalization of allowing for systems of equations, rather than single equations amounts to allowing for *mutually recursive definitions* like, eg,

$$\begin{array}{rcl} x & = & a \,;\, y \\ y & = & b \,;\, c \,;\, y \end{array}$$

Next, recall the embedding $\gamma : B \Rightarrow T$ of the behaviour into the syntax, mapping $*$ to nil and $< a, x >$ to $a \,;\, x$. Then, a system of (mutually) recursive definitions $k : X \to TX$ is **guarded** if it factorizes through a coalgebra

$$g : X \to BTX$$

of the composite endofunctor BT in the sense that

$$\begin{tikzcd} X \arrow[rr, "k"] \arrow[dr, "g"'] & & TX \\ & BTX \arrow[r, "\gamma_{TX}"'] & T^2X \arrow[u, "\mu_X"'] \end{tikzcd}$$

commutes, that is, $k = \mu_X \circ \gamma_{TX} \circ g : X \to TX$, where $\mu : T^2 \Rightarrow T$ is the multiplication of the syntactical monad T (cf Section 2). For instance, the above system is guarded because the corresponding T-coalgebra factorizes through

$$g(x) = < a, y > \qquad g(y) = < b,\, c \,;\, y >$$

Next, given the germ

$$\phi : \Sigma BT \Rightarrow BT$$

of an inductive functorial operational semantics, write

$$[\![\text{-}]\!]_g : TX \to BTX$$

for the *inductive* extension $g^\sharp$ of the Σ-algebra structure $\phi_X : \Sigma BTX \to BTX$ along a system

$$g : X \to BTX$$

of guarded recursive definitions:

$$\begin{tikzcd}
X \arrow[r, "\eta_X"] \arrow[dr, "g"'] & TX \arrow[d, dashed, "\llbracket\text{-}\rrbracket_g = g^\sharp"] & \Sigma TX \arrow[l, "\mathsf{inr}_X"'] \arrow[d, "\Sigma\llbracket\text{-}\rrbracket_g"] \\
 & BTX & \Sigma BTX \arrow[l, "\phi_X"]
\end{tikzcd}$$

Notice the left triangle tells that, up to the insertion-of-variables η_X,

$$\llbracket x \rrbracket_g = g(x)$$

for every x in X. In this way the variables $x \in X$ can be seen as states of a transition systems whose behaviour is described by the semantics ϕ. For instance, in the above example, $x \xrightarrow{a} y$ and $y \xrightarrow{b} c\ ;\ y$.

Then, the desired interpretation of g as a recursive process is obtained by taking the corresponding final coalgebra semantics $(g^\sharp)^@ = \llbracket\text{-}\rrbracket_g^@ : TX \to \widehat{B}$ precomposed with the insertion-of-variables $\eta_X : X \to TX$. Write, abusing the notation, $g^@$ for this function:

$$\begin{tikzcd}
X \arrow[r, "\eta_X"] \arrow[rr, bend left, "g^@"] \arrow[dr, "g"'] & TX \arrow[r, dashed, "\llbracket\text{-}\rrbracket_g^@"] \arrow[d, "\llbracket\text{-}\rrbracket_g = g^\sharp"] & \widehat{B} \arrow[d, "\cong"] \\
 & BTX \arrow[r, "B\llbracket\text{-}\rrbracket_g^@"'] & B\widehat{B}
\end{tikzcd}$$

Thus, for the above example, one has, omitting, as usual, both the insertion-of-variables η_X and the final coalgebra isomorphism φ,

$$\begin{array}{rcl} g^@(x) & = & <a, g^@(y)> \\ g^@(y) & = & <b,\ \llbracket c\ ;\ y \rrbracket_g^@> \quad = \quad <b, c, g^@(y)> \end{array}$$

that is, $g^@(x) = a(bc)^\omega$ and $g^@(y) = (bc)^\omega$. (Cf the above final coalgebra semantics.)

To be formal, the functorial operational semantics of the previous section gives

$$\llbracket c\ ;\ y \rrbracket_g = <c, \gamma_{TX} \llbracket y \rrbracket_g>$$

hence $g^@(y) = <b, c, \llbracket \gamma_{TX} \llbracket y \rrbracket_g \rrbracket_g^@>$. However, by 'unfolding' $\llbracket \gamma_{TX} \llbracket y \rrbracket_g \rrbracket_g^@$ by one step

$$\llbracket \gamma_{TX} \llbracket y \rrbracket_g \rrbracket_g^@ = \varphi^{-1} \circ B\llbracket\text{-}\rrbracket_g^@ \circ \llbracket \gamma_{TX} \llbracket y \rrbracket_g \rrbracket_g$$

and by using the fact that γ_{TX} is a retraction for the operational model

$$\llbracket \gamma_{TX} \llbracket y \rrbracket_g \rrbracket_g = \llbracket y \rrbracket_g$$

one obtains

$$g^{@}(y) = \llbracket \gamma_{TX} \llbracket y \rrbracket_g \rrbracket_g^{@}$$

Therefore, the equation

$$g^{@}(y) = \langle b, c, g^{@}(y) \rangle$$

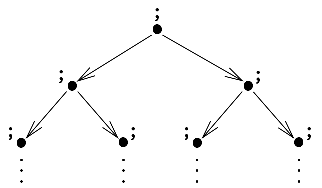
does hold.

Notice that the above recursive definition is automatically well-defined because of the coinduction principle given by finality. In general, final coalgebras allow recursive constructs to be interpreted also in categories where there is no structure to ensure the existence of (canonical) fixed points of functions. In other words, the above interpretation of recursion by final coalgebras encompasses the traditional methods using least fixed points in complete partial orders, or unique fixed points in complete metric spaces, or, more recently, the solution lemma in non-well-founded sets (see Part V), and it permits to interpret recursion in any category, including the ordinary category of (standard) sets.

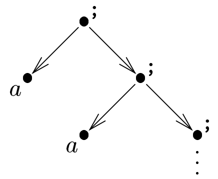
Unguarded Recursion. An alternative approach to recursive programs is obtained by regarding them as (possibly) infinite terms. Representing a term as a tree whose root is labelled by the outermost constructor of the term, one has, for instance, that the solution of the equation

$$x = x \; ; \; x$$

is an infinite tree with no leaf and all nodes labelled by ‘ ; ’:



Similarly, the solution of $x = a \; ; \; x$ is the infinite term represented by the following tree



The advantage of this approach is that it can be applied also to unguarded definitions, but, in order for an infinite term to be given an operational meaning, one needs to shift from the category of ordinary sets to categories with more structured objects like cpos or complete metric spaces.

Coalgebraically, the idea is that, while the initial Σ-algebra is the set of finite terms in Σ, the final Σ-coalgebra contains also the infinite terms. The argument is similar to the one above showing that the final coalgebra of the behaviour $X \mapsto 1+\mathsf{Act}\times X$ contains both finite and infinite streams, while its initial algebra only the finite ones. Now, apart from 'meaningless' equations like $x = x$ (or, more generally, $x = y$, $y = x$) every (possibly unguarded) system of term-equations can be seen as a coalgebra of the composite endofunctor ΣT, that is, as a function

$$k : X \to \Sigma T X$$

This can be made into a Σ-coalgebra with carrier TX by 'copairing' k with the identity on ΣTX using the fact that TX, since it is the carrier of the initial $(X{+}\Sigma)$-algebra, is a coproduct $TX \cong X + \Sigma TX$

$$\begin{tikzcd}
X \arrow[r, tail] \arrow[dr, "k"'] & TX \arrow[d, dashed, "{[k, \mathsf{id}_{\Sigma TX}]}"] & \Sigma TX \arrow[l, tail'] \arrow[dl, equal] \\
& \Sigma TX &
\end{tikzcd}$$

(By definition, the value of this coalgebra structure at a variable x is the same as the value of k at x.) Abusing the notation, write

$$k^{@} : X \to \widehat{\Sigma}$$

for the composition of the insertion-of-variables $\eta_X : X \to TX$ with the coinductive extension of the copair $[k, \mathsf{id}_{\Sigma TX}] : TX \to \Sigma TX$

$$\begin{tikzcd}
X \arrow[r, "\eta_X"] \arrow[rr, bend left, "k^{@}"] \arrow[dr, "k"'] & TX \arrow[r, dashed, "{[k, \mathsf{id}_{\Sigma TX}]^{@}}"] \arrow[d, "{[k, \mathsf{id}_{\Sigma TX}]}"] & \widehat{\Sigma} \arrow[d, "\cong"] \\
& \Sigma TX \arrow[r, "{\Sigma[k, \mathsf{id}_{\Sigma TX}]^{@}}"'] & \Sigma\widehat{\Sigma}
\end{tikzcd}$$

Thus, for the coalgebra structure k corresponding to the equation $x = x \,;\, x$ one has, omitting, as usual, the final coalgebra isomorphism,

$$k^{@}x = (k^{@}x) \,;\, (k^{@}x)$$

which is the desired infinite term.

Once infinite terms are introduced in the syntax, the problem arises of how to interpret them operationally. One possible solution is to consider categories in which initial algebras and final coalgebras coincide. Indeed, if the inverse of the initial Σ-*algebra* isomorphism $\Sigma\overline{\Sigma} \cong \overline{\Sigma}$ is the final Σ-*coalgebra* isomorphism $\widehat{\Sigma} \cong \Sigma\widehat{\Sigma}$ and, hence,

$$\overline{\Sigma} = T0 = \widehat{\Sigma}$$

then the interpretation of a recursive definition k is the composition of the above $k^{@} : X \to T0 = \widehat{\Sigma}$ with the coinductive extension $[\![-]\!]^{@} : T0 = \overline{\Sigma} \to \widehat{B}$ of the operational model $[\![-]\!] : T0 \to BT0$.

$$\begin{tikzcd}
& X \arrow[rr, "k^{@}"] \arrow[dl, "k"'] & & \widehat{\Sigma} = T0 \arrow[rr, "[\![-]\!]^{@}"] \arrow[dl, "\cong"] \arrow[dd, "[\![-]\!]"'] & & \widehat{B} \arrow[dd, "\cong"] \\
\Sigma T X \arrow[rr, "\Sigma[k, \mathsf{id}_{\Sigma TX}]^{@}"'] & & \Sigma\widehat{\Sigma} & & & \\
& & & BT0 \arrow[rr, "B[\![-]\!]^{@}"'] & & B\widehat{B}
\end{tikzcd}$$

As mentioned in Section 1, a category where the initial Σ-algebra is also the final Σ-coalgebra is **pSet**, the category of sets and partial functions. However, like in **Set**, also in **pSet** the object $\overline{\Sigma} = T0$ is the set of finite terms only: the arrow $k^{@} : X \to T0$ is thus a partial function mapping to 'undefined' *every* variable whose intended solution is an infinite term. Thus, in particular, both $x = x \,;\, x$ and $x = a \,;\, x$ would be interpreted as undefined, which is not what one expects.

To obtain both infinite terms as elements of an initial algebra and the coincidence of initial algebra and final coalgebras one can move from **pSets** to **pCpo**, the category having as objects complete partial orders (possibly without a bottom element) and as arrows *partial Scott-continuous* functions. The signature $\Sigma X = \coprod_{\sigma} X^{\mathsf{ar}(\sigma)}$ extends to **pCpo** but its initial algebra is the same as the one in **pSets**. In order to obtain infinite terms, one needs to modify Σ by applying to every element of the coproduct $\coprod_{\sigma\in\Sigma} X^{\mathrm{ar}(\sigma)}$ the *lifting monad* $X \mapsto X_{\perp}$ adding a new bottom element to a cpo. That is, take

$$\Sigma X = \coprod_{\sigma\in\Sigma} (X^{\mathrm{ar}(\sigma)})_{\perp}$$

In this way, the syntax will contain both *partial terms* of the form $\perp \,;\, (a \,;\, \perp)$ and infinite terms obtained as limit of finite terms. (Cf [Plo81a]: "Syntax considered as a cpo".)

Notice that the behaviour $BX = 1 + \mathsf{Act} \times X$ also extends to **pCpo** but, in general, the problem remains of how to extend a functorial operational semantics from sets to cpos. This is not treated in the present study and left to future work. It shows anyway the importance of the generality of the formulation of functorial operational semantics, where the base category **C** is not necessarily **Set**.

Notes. The standard solution of *domain equations* in **pCpo** [SP82, Plo85] has long been known to be a final coalgebra, but this was obscured by the fact that initial algebras and final coalgebras of ('locally continuous') endofunctors on **pCpo** do coincide in the sense that they are 'canonically isomorphic'. (And the same holds for 'locally contracting' endofunctors on complete metric spaces – cf [AR89, RT93].) Correspondingly, the availability of a coinduction principle was obscured by the use of induction and by 'internal' properties, like the existence of least (respectively unique) fixed points of continuous (respectively contracting) functions.

It has been Peter Aczel's work on 'non-well-founded sets' [Acz88] which has brought to light the main semantic properties of final coalgebras. (But see also [Ole82] for an early example of coinductive definitions by means of final coalgebras.) In [RT93], a first attempt is made towards systematizing these properties and the term 'final (coalgebra) semantics' is introduced. Examples of final coalgebra semantics appear in [RT94] (both with ordinary sets and with semi-lattices), [Acz94, Bal94, HL95, Har96] (with non-well-founded sets), [Fio93] (with complete partial orders), and [TJ93] (both with complete partial orders and with semi-lattices).

The above coalgebraic/functorial approach to the operational semantics of recursive programs deals neatly with top-level, mutually recursive definitions, but it ignores some aspects of the expressivity of the 'fix' operation, like the ability of dealing with local definitions and parameterized definitions: this is left to future work.