Adding recursive constructs to bialgebraic semantics

Bartek Klin

BRICS, Department of Computer Science, University of Aarhus
Ny Munkegade bldg. 540, DK-8000 Aarhus C, Denmark[1]

Abstract

This paper aims at fitting a general class of recursive equations into the framework of 'well-behaved' structural operational semantics, formalized as bialgebraic semantics by Turi and Plotkin. Rather than interpreting recursive constructs by means of operational rules, separate recursive equations are added to semantic descriptions of languages. The equations, together with the remaining rules, are then interpreted in a suitable category and merged by means of certain fixpoint constructions. For a class of recursive equations called *regular unfolding rules*, this construction yields distributive laws as analyzed by Turi and Plotkin. This means that regular unfolding rules can be merged seamlessly with abstract operational rules.

1 Introduction

Structural operational semantics ([13], [2]) is one of the fundamental methods of describing behaviour of programs. It is often presented in the form of a transition relation, which in turn is defined by a set of operational rules describing the behaviour of all constructs of the language. Given a set of rules, a transition system — the intended operational model of the language — is derived inductively.

It is often the case that transition relations can be viewed as B-coalgebras for a suitably chosen endofunctor B, called the behaviour functor in this context. If final B-coalgebras exist, then the unique coalgebra homomorphism from the intended operational model to the final coalgebra is called the final coalgebra semantics [16]. The final semantics is always adequate, i.e., it preserves the behavioural distinctions induced by the intended operational model.

Email address: klin@brics.dk (Bartek Klin).

[1] Basic Research in Computer Science (www.brics.dk),
funded by the Danish National Research Foundation.

The signature of a language without variable binding may often be viewed as an endofunctor, usually denoted Σ. A denotational model of a language is then a semantic domain equipped with an operation corresponding to every construct of the language, i.e., it is a Σ-algebra. Closed terms of the language with the associated straightforward operations form an initial Σ-algebra, and the unique algebra homomorphism from the initial algebra to a given denotational model is called the initial algebra semantics [9]. The initial algebra semantics is always compositional, i.e., the meaning of a compound term depends only on the meanings of its sub-terms.

If, for a given language, the final coalgebra semantics induced by the intended operational model coincides with the initial algebra semantics induced by some denotational model, then the final semantics is compositional, and the denotational model gives adequate denotational semantics [16].

In [17], [19] a general categorical framework for 'well-behaved' operational semantics was introduced, called then *bialgebraic semantics*. It is an abstraction of the GSOS format for operational rules [5]. Given a signature endofunctor Σ and a behaviour endofunctor B, (abstract) operational rules are defined to be natural transformations (in a suitably chosen category with enough structure)

$$\rho : \Sigma(Id \times B) \to BT$$

where T is the monad freely generated by Σ, and is called the syntactic monad of the language.

From operational rules of this type, one can derive the intended operational model of the language by induction on the structure of terms. One derives also a canonical denotational model which is always adequate, and under certain conditions on B it is also fully abstract with respect to the behavioural equivalence associated to B.

This mathematically elegant framework covers many interesting examples of process languages ([19], [18]). However, it is not immediately clear how to extend it to cover variable binding and/or recursive constructs. The problem with variable binding is to some extent syntactical: signatures with variable binding constructs are not as easily expressed as endofunctors as ordinary basic constructs of process languages. As was shown in [8], [7], this can be remedied by interpreting operational rules in a suitable presheaf category. At the expense of making the framework considerably more complex, the authors were able to formalize the operational semantics for name-passing and value-passing constructs.

The problem with recursive constructs is of somewhat different nature. Even when variable binding is not around, it is not clear how to interpret such constructs in the bialgebraic framework. As an example, consider an unary language construct `loop` designed to repeat some computation infinitely. Traditionally, there are two ways of writing operational rules for this construct.

One possibility is to write (in the presence of a sequential composition operator '`;`' in the language)

$$\frac{}{\texttt{loop}\ t \rightarrow t; \texttt{loop}\ t}$$

This rule is structural and accurately describes the intended meaning of the looping construct. However in certain situations, e.g., where the intended operational model of the language should be a labelled transition system, and not an unlabelled one, this rule is a source of problems. To merge this rule with other rules that generate a labelled transition system, one has to change the behaviour functor of the intended operational model. More importantly, the final semantics of the derived model does not ignore the 'silent' transitions associated with the above rule. As a result, programs like `loop a` and `a;loop a` are mapped differently by the final semantics, which contradicts the intended meaning of the looping construct.

To remedy this problem on an abstract level, a general coalgebraic theory of weak bisimulation is needed. Some work in this direction has been done ([15], [14]), but no satisfactorily abstract results have been obtained yet. This means that the above rule is hard to fit into the bialgebraic framework so far.

Another option to describe the looping construct is to give the rule

$$\frac{t; \texttt{loop}\ t \xrightarrow{a} t'}{\texttt{loop}\ t \xrightarrow{a} t'}$$

This rule does not introduce any unwelcome silent transitions. However, it is not structural: to compute the intended operational model for a language with this rule, one uses a fixpoint construction rather than induction. This makes it impossible to fit this rule directly into the bialgebraic framework.

In this paper, another approach has been considered. Here recursive constructs are treated separately from the recursion-free ones, and described not with operational rules, but with recursive equations. The equations are then merged with the bialgebraic models obtained for the recursion-free fragment of the language.

This general approach was first used in [18] for the case of guarded recursion. However, unguarded equations like

$$\texttt{loop}\ t = t; \texttt{loop}\ t$$

have not been treated so far.

In this paper, the approach introduced in [18] is modified and extended to deal with a wide range of recursive equations, including many unguarded ones. However, to keep things simple only constructs without variable binding are considered, to avoid moving to a sophisticated presheaf category.

Since unguarded recursive equations may be a source of partiality (divergence), it is convenient to interpret the operational rules and the recursive equations in a suitable category of domains rather than in the category of sets. The examples aimed to explain the constructions introduced in the paper use the category $\mathbf{Cppo}_\perp$ of pointed cpos and continuous, strict functions. However, all the constructions used can be interpreted in any **Cppo**-enriched category equipped with the usual structure needed to interpret the bialgebraic framework of [17], [19].

The structure of the paper is as follows. Section 2 introduces some categorical preliminaries together with certain fixpoint constructions used in the following. In Section 3, the basics of bialgebraic semantics are recalled, illustrated by two examples, and the developments of the remaining sections are introduced on informal level. The examples shown use $\mathbf{Cppo}_\perp$ as the underlying category. One of the examples deals with a looping construct `loop`, and another one — with a more sophisticated construct `unfolding`, which corresponds roughly to the general recursive construct μ restricted to a single recursive variable.

In Section 4 recursive equations are formalized abstractly as unfolding rules, and in Sections 5-7 it is shown how to merge unfolding rules with the bialgebraic framework.

In the developments presented in Sections 5-7 it repeatedly shows that the recursive equation for the construct `unfolding` considered in one of the examples is far less structured than the one for the construct `loop`. This motivates the definition of a regular unfolding rule in Section 8, which is satisfied by the latter but not by the former. Regular unfolding rules have some useful properties: in particular, they allow to construct a distributive law (abstract operational rules) for the full language from a distributive law for the recursion-free fragment.

In Section 9, it is hinted how parts of the present framework could be moved from **Cppo**-enriched categories to the simpler category of sets. It is also mentioned why moving the entire framework to sets seems to be a difficult task. Finally, main directions of future work are described.

Acknowledgements. The author is very grateful to Daniele Turi for introducing him to the subject and for guidance and many fruitful discussions during the initial stage of this work. Gordon Plotkin's insightful remarks motivated the developments presented in Section 8. Thanks are also due to Peter D. Mosses and Paweł Sobociński for useful suggestions. Detailed comments from anonymous referees helped to improve the presentation and remove several mistakes from an initial version of this paper.

2 Preliminaries

This section contains standard definitions of monads and distributive laws, taken largely from [10], [20] and [19], as well as some preliminary fixpoint constructions used in the following sections. For unexplained categorical notions, refer to [11].

A *pointed endofunctor* (T, η) on a category C is an endofunctor T on C together with a natural transformation $\eta : Id \to T$.

A *monad* (T, η, μ) is a pointed endofunctor (T, η) together with a natural transformation $\mu : TT \to T$ such that

$$\mu \circ T\eta = \mu \circ \eta T = id$$

$$\mu \circ T\mu = \mu \circ \mu T$$

We will often omit the names of the above natural transformations and speak of a pointed endofunctor T, or of a monad T.

An *algebra* for an endofunctor T is a morphism $h : TX \to X$. X is then called the *carrier* of h. An algebra for a pointed endofunctor T is an algebra $h : TX \to X$ for the endofunctor T such that $h \circ \eta_X = id_X$. An algebra for a monad T is an algebra $h : TX \to X$ for the pointed endofunctor T such that $h \circ Th = h \circ \mu_X$.

If T is a monad (a pointed endofunctor) then by a T-algebra we will mean an algebra for the monad (the pointed endofunctor), unless otherwise stated.

If the category C has all ω-limits, then every ω-cocontinuous endofunctor Σ on C freely generates a monad T. Then TX is the carrier of the initial $(X + \Sigma)$-algebra, it is isomorphic to $X + \Sigma TX$, and Σ-algebras are in one-to-one correspondence with T-algebras.

A *copointed endofunctor* (H, π) is an endofunctor H together with a natural transformation $\pi : H \to Id$.

A *coalgebra* for an endofunctor B is a morphism $k : X \to BX$. A coalgebra for a copointed endofunctor (H, π) is a coalgebra $k : X \to HX$ for the endofunctor H such that $\pi_X \circ k = id_X$.

Any endofunctor B on a category with binary products cofreely generates a copointed endofunctor $H = Id \times B$ with $\pi = \pi_1$. Obviously then H-coalgebras are in one-to-one correspondence with B-coalgebras.

A *distributive law* of a pointed endofunctor T over a copointed endofunctor H is a natural transformation $\lambda : TH \to HT$ such that

$$\lambda \circ \eta H = H\eta$$

$$T\pi = \pi T \circ \lambda$$

If, moreover, T is a monad and

$$\lambda \circ \mu H = H\mu \circ \lambda T \circ T\lambda$$

then λ is called a distributive law of the monad T over the copointed endofunctor H.

If $H = Id \times B$ is the copointed endofunctor cofreely generated by an endofunctor B, then distributive laws of the pointed endofunctor (monad) T over H are equivalent to natural transformations $\lambda : TH \to BT$ such that $\lambda \circ \eta H = B\eta \circ \pi_2$ (and respectively, $\lambda \circ \mu H = B\mu \circ \lambda T \circ T(T\pi_1, \lambda)$). With a slight abuse of the language, such natural transformations will also be called distributive laws of the copointed endofunctor (monad) T over H.

For a distributive law $\lambda : TH \to BT$ of the pointed endofunctor T over the copointed endofunctor H, a λ-*model* with carrier X is a pair $TX \xrightarrow{h} X \xrightarrow{k} BX$ of a T-algebra and B-coalgebra, satisfying the 'pentagonal law':

$$k \circ h = Bh \circ \lambda_X \circ T(id, k)$$

If, additionally, T is a monad and λ is a distributive law of the monad over H, then the λ-model is called a λ-*bialgebra*.

A category is called **Cppo**-enriched, if its homsets are cppos ((ω)-complete partial orders with bottom elements) and if composition is continuous in both arguments. A morphism f in a **Cppo**-enriched category will be called *strict*, if both operations $_ \circ f$ and $f \circ _$ preserve bottom elements. If all morphisms in a category C are strict, then the collection of bottom elements of homsets may be viewed as a natural transformation $\perp$ between any two given functors.

In the following sections many morphisms in a **Cppo**-enriched category will be defined as fixpoints of certain continuous functions. In many applications the least fixpoints are the ones of the most interest. However, for our reasons it will be more useful to consider only fixpoints that satisfy certain conditions (for example, that factorize through some given morphism) and then take the least fixpoint from this class. This section presents some techniques aimed at defining and reasoning about such morphisms.

First, a straightforward generalization of Tarski's Fixpoint Theorem:

Theorem 1 (Tarski) *Consider a cppo V and a continuous function $\Phi : V \to V$. For every $f \in V$ such that $\Phi f \geq f$, there exists the least fixpoint of Φ greater or equal to f, denoted by $\Phi^*(f)$. Moreover,*

$$\Phi^*(f) = \bigsqcup_{n \in N} \Phi^n f$$

□

Note that for any continuous $\Phi : V \to V$, the set V_Φ of those elements $f \in V$ for which $\Phi f \geq f$, is a sub-cppo of V, i.e., it contains the bottom element $\bot$ and it is closed under lubs. It is easy to observe that $\Phi^* : V_\Phi \to V$ is a continuous operation (hence the notation).

In the following sections we shall prove many properties of morphisms defined as fixpoints. Most of these properties have a form of a commuting diagram involving two such morphisms. In particular, this pattern appears when proving naturality of families of morphisms defined as fixpoints. To prove such properties the following two lemmas will be used.

Lemma 2 *Consider the following shape in a* ***Cppo****-enriched category C:*

$$\begin{array}{ccccc} X & \xrightarrow{k_1} & Y_1 & \xrightarrow{f_1} & Y_1' \\ {\scriptstyle k_2}\downarrow & & & & \downarrow{\scriptstyle l_1} \\ Y_2 & \xrightarrow[f_2]{} & Y_2' & \xrightarrow[l_2]{} & Z \end{array}$$

and two continuous functions

$$\Phi_1 : C(Y_1, Y_1') \to C(Y_1, Y_1') \qquad \Phi_2 : C(Y_2, Y_2') \to C(Y_2, Y_2')$$

such that $\Phi_1(f_1) \geq f_1$ and $\Phi_2(f_2) \geq f_2$. If

- $l_1 \circ f_1 \circ k_1 = l_2 \circ f_2 \circ k_2$
- $l_1 \circ x \circ k_1 = l_2 \circ y \circ k_2$ *implies* $l_1 \circ \Phi_1(x) \circ k_1 = l_2 \circ \Phi_2(y) \circ k_2$

then $l_1 \circ \Phi_1^*(f_1) \circ k_1 = l_2 \circ \Phi_2^*(f_2) \circ k_2$.

Proof: First show by induction that for any n, $l_1 \circ \Phi_1^n(f_1) \circ k_1 = l_2 \circ \Phi_2^n(f_2) \circ k_2$. Then use continuity of composition. □

In other cases, another proof principle will be more useful:

Lemma 3 *In the setting like in Lemma 2, if*

- $l_1 \circ f_1 \circ k_1 \leq l_2 \circ \Phi_2^*(f_2) \circ k_2$
- $l_1 \circ x \circ k_1 \leq l_2 \circ y \circ k_2$ *implies* $l_1 \circ \Phi_1(x) \circ k_1 \leq l_2 \circ \Phi_2(y) \circ k_2$
- $l_1 \circ \Phi_1^*(f_1) \circ k_1 \geq l_2 \circ f_2 \circ k_2$
- $l_1 \circ x \circ k_1 \geq l_2 \circ y \circ k_2$ *implies* $l_1 \circ \Phi_1(x) \circ k_1 \geq l_2 \circ \Phi_2(y) \circ k_2$

then $l_1 \circ \Phi_1^*(f_1) \circ k_1 = l_2 \circ \Phi_2^*(f_2) \circ k_2$.

Proof: To prove $l_1 \circ \Phi_1^*(f_1) \circ k_1 \leq l_2 \circ \Phi_2^*(f_2) \circ k_2$, show by induction from the first two assumptions that for any n, $l_1 \circ \Phi_1^n(f_1) \circ k_1 \leq l_2 \circ \Phi_2^*(f_2) \circ k_2$. Then use continuity of composition.

The proof of $l_1 \circ \Phi_1^*(f_1) \circ k_1 \geq l_2 \circ \Phi_2^*(f_2) \circ k_2$ proceeds analogously using the last two assumptions. □

The above lemmas can be used to prove naturality of families of morphisms defined as fixpoints, by taking both k_1 and l_2 to be identities. This method of proving naturality will be called *square commutation by fixpoint induction.*

Finally, a special class of monomorphisms will be considered in any **Cppo**-enriched category:

Definition 4 *A morphism $f : A \to B$ in a **Cppo**-enriched category is an upper section, if there exists a morphism $g : B \to A$ (called left inverse to f) such that*

$$g \circ f = id_A \qquad \text{and} \qquad f \circ g \leq id_B$$

To prove various properties involving upper sections, the following easy lemma will be used:

Lemma 5 *Assume $f : A \to B$ is an upper section with left inverse g. For any $h : B \to C$, $k : A \to C$, if $k \circ g \circ f \leq h \circ f$ (or, equivalently, $k \leq h \circ f$), then $k \circ g \leq h$.*

Proof: $k \circ g = k \circ (g \circ f) \circ g \leq h \circ f \circ g \leq h$ □

For our purposes, especially interesting examples of upper sections will be units of some monads:

Definition 6 *A variable classifier for a monad (T, η, μ) on a **Cppo**-enriched category C is a natural transformation $\upsilon : T \to Id$ such that*

- *all η_X are upper sections with υ_X left inverses,*
- *all υ_X are algebras for the monad T.*

In particular, if T is freely generated by some endofunctor Σ, and if all morphisms in C are strict, then T can be equipped with a variable classifier:

$$\upsilon = [id, \bot] \circ \iota$$

where $\iota : T \to Id + \Sigma T$ is the isomorphism given by the structure of freely generated monad.

3 Bialgebraic semantics — motivating examples

Consider a language with the following simple syntax:

$$t ::= \texttt{nil} \mid a \mid t;t \mid t+t$$

(where a ranges over some fixed set of actions A), equipped with the following standard small-step operational semantics:

$$\frac{}{a \xrightarrow{a} \mathtt{nil}} \qquad \frac{x \xrightarrow{a} x'}{x;y \xrightarrow{a} x';y} \qquad \frac{x \not\rightarrow \quad y \xrightarrow{a} y'}{x;y \xrightarrow{a} y'}$$

$$\frac{x \xrightarrow{a} x'}{x+y \xrightarrow{a} x'} \qquad \frac{y \xrightarrow{a} y'}{x+y \xrightarrow{a} y'}$$

In [18] it was shown how such simple languages can be interpreted in the bialgebraic framework in an arbitrary category with enough structure. For the purposes of the example here, the category $\mathbf{Cppo}_\perp$ of cppos and strict continuous functions will be used.

The syntax of the simple language mentioned above corresponds to an endofunctor Σ on $\mathbf{Cppo}_\perp$:

$$\Sigma X = 1_\perp \oplus A_\perp \oplus X_\perp \otimes X_\perp \oplus X_\perp \otimes X_\perp = (1 + A + X \times X + X \times X)_\perp$$

where $\oplus$ and $\otimes$ are the coalesced sum and the smash product of cppos, $+$ and $\times$ are the disjoint sum and the Cartesian product of cpos, and $(_)_\perp$ is the lifting operation on cpos. This functor freely generates a monad T on $\mathbf{Cppo}_\perp$. Elements of TX are (possibly infinite) terms built over (a cppo of variables) X, with some sub-terms replaced by the bottom element $\perp$, and with the ordering induced from the ordering on X, with the remark that $\perp$ is smaller than any other term.

One possible behaviour endofunctor B for interpreting the operational rules shown above is

$$BX = \mathcal{P}(A_\perp \otimes X_\perp) = \mathcal{P}((A \times X)_\perp)$$

where $\mathcal{P}$ is the Plotkin powerdomain with the empty set adjoined (see e.g., [1]). As it turns out, the above operational semantics corresponds to a natural transformation

$$\rho : \Sigma(Id \times B) \to BT$$

defined by cases as follows:

$$\begin{aligned}
\rho_X(\iota_1(\mathtt{nil})) &= \emptyset \\
\rho_X(\iota_2(a)) &= \{(a, \mathtt{nil})\} \\
\rho_X(\iota_3(x_1, b_1, x_2, b_2)) &= \begin{cases} \{(a, \iota_3(x_1', x_2)) \mid (a, x_1') \in b_1\} \text{ if } b_1 \neq \emptyset \\ b_2 \text{ otherwise} \end{cases} \\
\rho_X(\iota_4(x_1, b_1, x_2, b_2)) &= b_1 \cup b_2
\end{aligned}$$

where X is a cppo (of variables), x and b range over X and BX respectively, and the ι_i are the coproduct injections to ΣX. It is easy to see that ρ is continuous and, when suitably extended to act on bottom elements, it is also strict. Obviously, ρ is also natural in X.

In [10] it was shown that a natural transformation ρ of the type as above is equivalent to a distributive law $\lambda : TH \to BT$ of the monad T freely generated by Σ over the copointed endofunctor H cofreely generated by B. As shown in [17], [19], ρ also induces a lifting T_λ of the monad T to the category of B-coalgebras, i.e., an endofunctor on the category of B-coalgebras such that for every B-coalgebra $k : X \to BX$ the following diagram commutes:

$$\begin{array}{ccccc} X & \xrightarrow{\eta_X} & TX & \xleftarrow{\mu_X} & TTX \\ {\scriptstyle k}\downarrow & & {\scriptstyle T_\lambda k}\downarrow & & \downarrow{\scriptstyle T_\lambda T_\lambda k} \\ BX & \xrightarrow[B\eta_X]{} & BTX & \xleftarrow[B\mu_X]{} & BTTX \end{array}$$

Moreover, for any B-coalgebra $k : X \to BX$, there is

$$T_\lambda k = \lambda_X \circ T(id, k)$$

When used in diagrams as above, in certain situations the notation $T_\lambda k$ might lead to some confusion. One way to read it is to silently convert a morphism $k \in C(X, BX)$ to an object in the category of B-coalgebras, apply the object part of the functor T_λ to it and silently convert the result back to a morphism in C. Another way might be to treat k as a morphism between some B-coalgebras and apply the morphism part of the functor T_λ to it. In this paper, only the former interpretation is used.

In [17], [19] it was shown that formalizing operational rules in the way shown above gives certain advantages: it allows to derive a canonical denotational model which is adequate with respect to the intended operational model, and ensures that the final semantics of the intended operational model is compositional.

Let us extend the example shown above in two independent ways, adding some simple recursive constructs to it.

Example A: The unary looping construct `loop`. This is done by extending the syntax as follows:

$$t ::= \ldots \mid \texttt{loop}\ t$$

with the intended meaning captured by the recursive equation

$$\texttt{loop}\ t = t; \texttt{loop}\ t$$

Example B: The unary construct `unfolding`, inspired by a similar construct from Mosses's action semantics [12]. This is a version of the general recursive

construct μ, restricted to only one fixed recursive variable, which is hence treated as a constant (to keep the language small, we use the constant `nil` for this purpose). The appropriate syntax extension is

$$t ::= \ldots \mid \texttt{unfolding}\ t$$

with the intended meaning captured by the recursive equation

$$\texttt{unfolding}\ t = t[\texttt{nil} \mapsto \texttt{unfolding}\ t]$$

where $t[r \mapsto s]$ is the standard notation for substitution on terms.

Note that the syntactic extensions in both examples correspond to the same extension of the endofunctor Σ to a new signature endofunctor Σ':

$$\Sigma' X = \Sigma X \oplus X_\perp$$

The syntactic monad freely generated by Σ' will be called T'.

Note that neither of the recursive equations mentioned here are guarded, and any of the above constructs is a potential source of divergence. Indeed, both terms `loop nil` and `unfolding nil` are immediately diverging. This explains the decision to interpret the language in a category of cpos, rather that in the category of sets.

The remainder of this section is devoted to an informal presentation of the technical developments of the following sections. This is to provide the reader with some intuition about the results presented in this paper.

In both examples above, the recursive equation given might be seen as a natural transformation

$$r : T' \to TT'$$

which, given a term in $T'X$, performs one step of syntactic unfolding so that some recursion-free constructs appear on top of it, and splits the resulting term in two 'layers'. For instance, in example A,

$$\texttt{loop}\ a \rightsquigarrow (_;_)[a, \texttt{loop}\ a]$$

where $\rightsquigarrow$ describes the action of r on terms. The notation $(t)[t_1, \ldots, t_n]$ (where t is recursion-free and may contain some place-holders) is to represent a term split in two 'layers', i.e., an element of $TT'X$.

In example B, one can have for instance

$\texttt{unfolding}(a;\texttt{nil}) \rightsquigarrow (_;_)[a, \texttt{unfolding}(a;\texttt{nil})]$

$\texttt{unfolding}\ a \rightsquigarrow (a)[]$

$\texttt{unfolding nil} \rightsquigarrow (_)[\texttt{unfolding nil}]$

In a sense, natural transformations $r : T' \to TT'$ may be viewed as recursive specifications (see e.g., [4]), i.e., sets of recursive equations with terms from $T'X$ playing the role of recursion variables.

A question arises how r should act on constructs from the recursion-free fragment of the language. Several options are available here. Until the end of Section 7, it will not be specified if, for example

$a;(b;c) \rightsquigarrow (_;_)[a, b;c]$, or

$a;(b;c) \rightsquigarrow (a;(b;_))[c]$, or even

$a;(b;c) \rightsquigarrow (a;(b;c))[]$

It will only be required (in the definition of a decomposition structure in Section 4) that after repeating many steps of such 'unfolding' of a recursion-free term, eventually the entire term will appear in the top 'layer'. In other words, the option $a;(b;c) \rightsquigarrow (_)[a;(b;c)]$ will be excluded.

In Section 8, when considering *regular* unfolding rules, the first of the above options (as small a portion of syntax is unfolded as possible) will be specifically required.

However, all the above options have something in common: all recursion-free constructs are left unchanged when unfolded. This will be properly formalized in the definition of an unfolding rule in Section 4.

It is reasonable to expect that if a term is built of some variables, then 'unfolding' a variable leaves it intact.

Given an 'unfolding rule' r, one can repeat its action many times. In the limit, all recursive constructs are wiped out from a given term. This construction is formalized as a natural transformation $\bar{r} : T' \to T$ in Section 5. This transformation replaces recursive constructs with their infinite expansions, and leaves the remaining constructs unchanged.

Given operational rules for the recursion-free fragment of the language (formalized as a distributive law $\lambda : TH \to BT$), and an unfolding rule $r : T' \to TT'$, one can define operational rules for the full language (formalized as a distributive law $\lambda^r : T'H \to BT'$ in Section 6). Somewhat informally speaking, the new operational rules are defined by the following fixpoint construction:

if $t \rightsquigarrow (s)[s_1, \ldots, s_n]$ and if

$$\frac{\dfrac{\Gamma_1}{s_1 \overset{a_1}{\to} s_1'} \cdots \dfrac{\Gamma_n}{s_n \overset{a_n}{\to} s_n'} \quad \vdots \quad \Gamma}{s[s_1, \ldots, s_n] \overset{a}{\to} t'}$$

(where Γ, Γ_i are some sets of premises) is a correct derivation (where the vertical dots represent a derivation using only the rules for the recursion-free fragment), then

$$\frac{\Gamma \cup \Gamma_1 \cup \cdots \cup \Gamma_n}{t \overset{a}{\to} t'}$$

is a valid rule.

Note the difference between this approach (formalized in Section 6) and the second traditional approach shown in the Introduction. Here the burden of a fixpoint construction is shifted to the definition of operational rules, but the rules themselves are structural, and the intended operational model can be derived from them inductively. This will allow to merge recursive equations with the bialgebraic framework.

The intended operational model is derived from T'_{λ^r}, the lifting of the pointed endofunctor T' to B-algebras which is associated with the distributive law λ^r. Intuitively, the operational model unfolds a given term according to the rule r, until the resulting term exhibits some behaviour according to the rules λ. For instance, in the example A above, the intended operational model $T'_{\lambda^r}(0)$ maps a term $\texttt{loop}(a;b)$ to a term $b; \texttt{loop}(a;b)$ together with the action a.

In the remainder of this paper the constructions hinted above are defined formally.

4 Unfolding rules

The ideas presented in the preceding section can be formalized in an arbitrary **Cppo**-enriched category C with products, ω-colimits and with all morphisms being strict. The latter assumption allows to treat the collection of the bottom elements in the homsets as a natural transformation between arbitrary two endofunctors.

From now on assume that all endofunctors on C introduced here are locally continuous.

Assume we are given bialgebraic operational semantics for (the recursion-free fragment of) a language, in the sense of [17], [19], i.e.,

- A syntactic monad T on C,

- A behaviour endofunctor $B : C \to C$, with the copointed endofunctor $H = Id \times B$ cofreely generated by it,
- A set of abstract GSOS rules, i.e., a distributive law of the monad T over the copointed endofunctor H:

$$\lambda : TH \to BT$$

To introduce recursive equations, consider additionally

- A monad T' (the full language),
- A monad morphism $t : T \to T'$.

Assume moreover, that monads T and T' come with variable classifiers υ and υ' respectively, and that $\upsilon' \circ t = \upsilon$.

For any morphism $\gamma_X : TX \to TTX$, one can define a function between cppos $\tilde{\Phi}_X : C(TX, TX) \to C(TX, TX)$ as follows:

$$\tilde{\Phi}_X \frac{TX \xrightarrow{f} TX}{TX \xrightarrow{\gamma_X} TTX \xrightarrow{Tf} TTX \xrightarrow{\mu_X} TX}$$

It is easy to see that $\tilde{\Phi}_X$ is continuous on the cppo $C(TX, TX)$, since T is locally continuous and C is **Cppo**-enriched.

Assume moreover that $\gamma_X \circ \eta_X = T\eta_X \circ \eta_X$, and define $\tilde{e}_X = \eta_X \circ \upsilon_X$. It is easily checked that $\tilde{e}_X \circ \eta_X = \tilde{\Phi}_X(\tilde{e}_X) \circ \eta_X$, hence, by Lemma 5, $\tilde{e}_X \leq \tilde{\Phi}_X(\tilde{e}_X)$ and an 'infinite decomposition' map $\bar{\gamma}_X : TX \to TX$ can be defined as follows:

$$\bar{\gamma}_X = \tilde{\Phi}^*_X(\tilde{e}_X)$$

Definition 7 *A decomposition structure on T is a natural transformation $\gamma : T \to TT$ such that*

- $\gamma \circ \eta = T\eta \circ \eta$
- $\mu \circ \gamma = id$
- $\bar{\gamma}_X = id_{TX}$ *for any object X*

Examples ctd. A decomposition structure $\gamma : T \to TT$ for the simple syntax T shown in Section 3 can be derived from a natural transformation $\gamma^0 : \Sigma \to TT$, defined by cases as follows:

$$\begin{aligned}
\gamma^0_X(\texttt{nil}) &= (\texttt{nil})[] \\
\gamma^0_X(\texttt{a}) &= (\texttt{a})[] \\
\gamma^0_X(x_1;x_2) &= (_;_)[x_1, x_2] \\
\gamma^0_X(x_1+x_2) &= (_+_)[x_1, x_2]
\end{aligned}$$

Then $\gamma : T \to TT$ defined as

$$\gamma_X = [T\eta_X \circ \eta_X, T\mu_X \circ \gamma^0_{TX}] \circ \psi_X$$

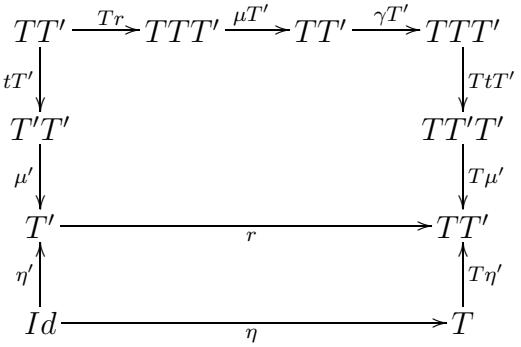
where $\psi_X : TX \to X + \Sigma TX$ is the isomorphism arising from the free monad structure of T) is a decomposition structure for T.

As the informal discusion in Section 3 suggests, an 'unfolding rule' from the 'full' language T' to its 'recursion-free fragment' T might be viewed as a natural transformation $r : T' \to TT'$. This is formalized as follows.

Definition 8 *An unfolding rule from T' to T (based on a decomposition structure γ on T) is a natural transformation*

$$r : T' \to TT'$$

such that the following diagram commutes:

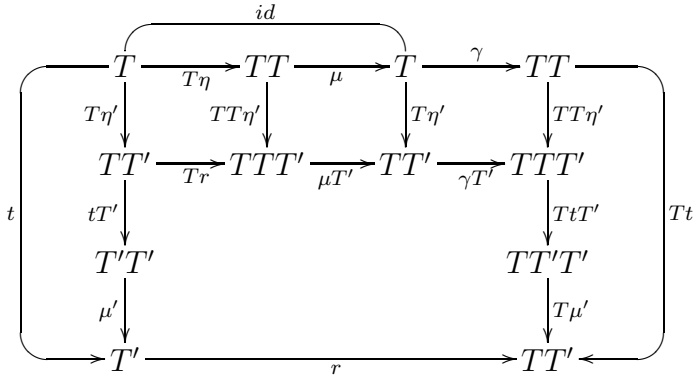


Intuitively, the bottom part of the above diagram means that variables are unfolded trivially. The top part means that the action of r on parts of terms built from syntax T is defined by γ. In particular, the following result holds:

Lemma 9 *For any unfolding rule $r : T' \to TT'$ based on $\gamma : T \to TT$,*

$$r \circ t = Tt \circ \gamma$$

Proof: Everything in the following diagram commutes:



Example A ctd. To define an unfolding rule $r : T' \to TT'$, one only needs to define its action on terms where the top construct is not present in the

recursion-free syntax T. The action on other terms can be defined canonically, using the decomposition structure γ on T. Having defined γ from γ^0 as above, this is particularly easy: take $r_X(\iota_2(t)) = T\mu'_X \circ \gamma^0_{T'X}(t)$, where $\iota_2 : \Sigma T'X \to T'X$. For other terms, one can take e.g.,

$$r_X(\texttt{loop}\ t) = (_;_)[t, \texttt{loop}\ t]$$

Example B ctd. As before, one only has to define r on terms with new constructs at the top. Here take e.g.,

$$r_X(\texttt{unfolding}\ t) = (_)[t[\texttt{nil} \mapsto \texttt{unfolding}\ t]]$$

5 Infinite unfolding

Given an unfolding rule $r : T' \to TT'$ one can translate any term built over the monad T' to a term over the monad T using a certain fixpoint construction, which intuitively corresponds to performing infinitely many unfolding steps.

Formally, given r, for any fixed object X in C, one can define a function $\Phi_X : C(T'X, TX) \to C(T'X, TX)$ acting as follows:

$$\Phi_X \frac{T'X \xrightarrow{f} TX}{T'X \xrightarrow{r_X} TT'X \xrightarrow{Tf} TTX \xrightarrow{\mu_X} TX}$$

It is easy to see that Φ_X is continuous on the cppo $C(T'X, TX)$, since T is locally continuous and C is **Cppo**-enriched.

Given an object X, the 'infinite unfolding' map $\bar{r}_X : T'X \to TX$ is defined as follows:

$$\bar{r}_X = \Phi^*_X(e_X)$$

where $e_X = \eta_X \circ \upsilon'_X$

Intuitively speaking, this map unfolds a term from $T'X$ performing infinitely many steps of unfolding as defined by r, and leaving the variables from X unchanged.

Note that $\bar{r}_X$ is properly defined, since $\Phi_X(e_X) \geq e_X$. This follows from Lemma 5, since it is easily checked that $\Phi_X(e_X) \circ \eta'_X = e_X \circ \eta'_X$.

Example A ctd. With definition of r as shown at the end of Section 4, one has for example

$$\begin{aligned} \bar{r}_0(\texttt{loop a}) &= \texttt{a}; \texttt{a}; \texttt{a}; \texttt{a}; \texttt{a}; \ldots \\ \bar{r}_0(\texttt{loop nil}) &= \bot \end{aligned}$$

Example B ctd. With definition of r as shown at the end of Section 4, here one has

$$\begin{aligned} \bar{r}_0(\texttt{unfolding a}) &= \texttt{a} \\ \bar{r}_0(\texttt{unfolding}(\texttt{a};\texttt{nil})) &= \texttt{a};\texttt{a};\texttt{a};\texttt{a};\texttt{a};\ldots \\ \bar{r}_0(\texttt{unfolding nil}) &= \bot \end{aligned}$$

The maps $\bar{r}_X$ are in fact natural in X:

Lemma 10 *The family $(\bar{r}_X : T'X \to TX)_{X \in C}$ derived from $r : T' \to TT'$ forms a natural transformation $\bar{r} : T' \to T$.*

Proof: (using square commutation by fixpoint induction)

Base case: The family $(e_X : T'X \to TX)_{X \in C}$ is a natural transformation.

Induction step: Assume that the family $(f_X : T'X \to TX)_{X \in C}$ is a natural transformation. Then everything in the definition of $\Phi_X(f_X)$ is natural in X, so the family $(\Phi_X(f_X) : T'X \to TX)_{X \in C}$ is also a natural transformation.

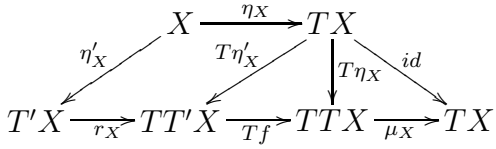
The lemma follows by square commutation by fixpoint induction (Lemma 2). □

Lemma 11 *For any natural transformation $r : T' \to TT'$, $\bar{r} \circ \eta' = \eta$.*

Proof: (componentwise by fixpoint induction on Φ_X)

Base case: Obvious by definition of e_X.

Induction step: Assume that for some $f : T'X \to TX$, $f \circ \eta'_X = \eta_X$. Then also $\Phi_X(f) \circ \eta'_X = \eta_X$, since everything in the following diagram commutes.



□

Remark 12 The latter result shows that $\bar{r}$ satisfies one of the axioms of a monad morphism from T' to T. However, in general $\bar{r}$ is not a monad morphism: it is not the case that

$$\bar{r} \circ \mu' = \mu \circ T\bar{r} \circ \bar{r}T'$$

Indeed, recall the example B presented in Section 3 (with definition of r given

in Section 4) and consider a term

$$s = (\texttt{unfolding_})[a; \texttt{nil}] \in T'T'0$$

Then $\mu_0(T\bar{r}_0(\bar{r}_{T'0}(s))) = a; \texttt{nil}$, but $\bar{r}_0(\mu'_0(s))$ is the infinite term $a;a;\ldots$.

This problem will be addressed in Section 8.

For any unfolding rule r, the unfolding transformation $\bar{r} : T' \to T$ is a left inverse to $t : T \to T'$:

Lemma 13 *For any unfolding rule r, $\bar{r} \circ t = id$.*

Proof: (componentwise) Recall that r is based on some decomposition structure γ on T, and that $id = \bar{\gamma}_X = \tilde{\Phi}^*_X(\tilde{e}_X)$.

To prove that $\Phi^*_X(e_X) \circ t_X = \tilde{\Phi}^*_X(\tilde{e}_X)$, proceed by simultaneous fixpoint induction on Φ_X and $\tilde{\Phi}_X$.

Base case: $e_X \circ t_X = \eta_X \circ \upsilon'_X \circ t_X = \eta_X \circ \upsilon_X = \tilde{e}_X$

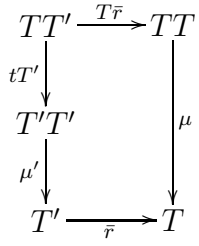
Induction step: Assume $f \circ t_X = g$ for some $f : T'X \to TX$, $g : TX \to TX$. Then

$$\begin{aligned}\Phi_X(f) \circ t_X = \mu_X \circ Tf \circ r_X \circ t_X = &\text{(Lemma 9)}\\ \mu_X \circ Tf \circ Tt_X \circ \gamma = &\\ \mu_X \circ Tg \circ \gamma_X = \tilde{\Phi}_X(g)&\end{aligned}$$

The lemma follows from Lemma 2. □

In the following sections, the following technical lemma will be often used:

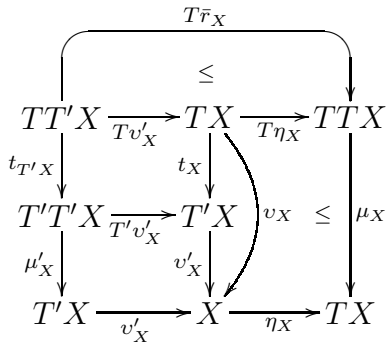
Lemma 14 *For any unfolding rule r, the diagram*



commutes.

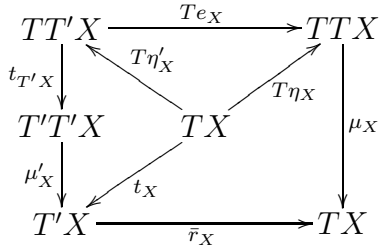
Proof: Proceed componentwise using Lemma 3.

Base case 1: To see that $e_X \circ \mu'_X \circ t_{T'X} \leq \mu_X \circ T\bar{r}_X$, chase the following diagram:



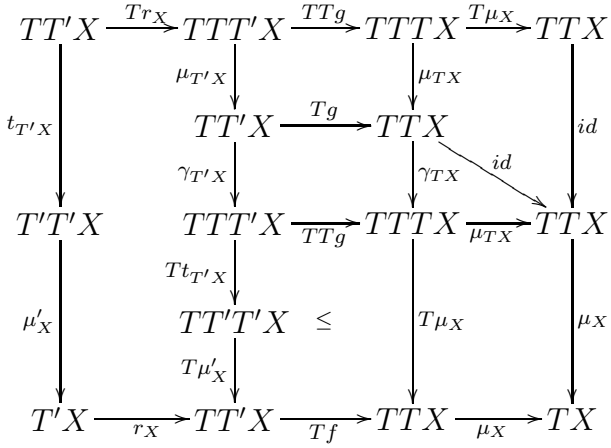
In particular, the lower left square commutes since υ'_X is a T'-algebra, and the upper region — by definition of $\bar{r}$.

Base case 2: Note that locally continuous functors preserve upper sections and their left inverses. This means that $T\eta'_X$ is an upper section, and to check that $\bar{r}_X \circ \mu'_X \circ t_{T'X} \geq \mu_X \circ Te_X$, by Lemma 5 it is enough to check that $\bar{r}_X \circ \mu'_X \circ t_{T'X} \circ T\eta'_X = \mu_X \circ Te_X \circ T\eta'_X$. To see this, chase the diagram



In particular, the lower right square commutes by Lemma 13.

Induction step 1: Consider two morphisms $f : T'X \to TX$, $g : T'X \to TX$ such that $f \circ \mu'_X \circ t_{T'X} \leq \mu_X \circ Tg$. Then the following diagram commutes:



(the upper right square commutes only when postcomposed with μ_X).

Induction step 2: Proceed like in Induction step 1, changing all $\leq$ to $\geq$.

The lemma follows from Lemma 3.

6 Merging unfolding rules with distributive laws

Operational semantics formalized as a distributive law $\lambda : TH \to BT$ of a monad T over a copointed endofunctor H gives rise to an operational interpretation of the language T. Given an unfolding rule $r : T' \to TT'$, one can extend λ to a distributive law $\lambda^r : T'H \to BT'$, thus providing an operational interpretation of terms build from syntax T'.

More formally, for a given distributive law $\lambda : TH \to BT$ and an unfolding rule $r : T' \to TT'$, one defines the function $\Psi_X : C(T'HX, BT'X) \to C(T'HX, BT'X)$ as follows:

$$\Psi_X \frac{T'HX \xrightarrow{f} BT'X}{T'HX \xrightarrow{r_{HX}} TT'HX \xrightarrow{T(T'\pi_1, f)} THT'X \xrightarrow{\lambda_{T'X}} BTT'X \xrightarrow{Bt_{T'X}} BT'T'X \xrightarrow{B\mu'_X} BT'X}$$

It is easy to see that Ψ_X is continuous, since T and B are locally continuous, and C is **Cppo**-enriched.

Based on this function, the morphism $\lambda^r_X : T'HX \to BT'X$ is defined as follows:

$$\lambda^r_X = \Psi^*_X(d_X)$$

where $d_X = B\eta'_X \circ \pi_2 \circ \upsilon'_{HX}$. Note that λ^r_X is well defined, because

$$\Psi_X(d_X) \geq d_X$$

This follows from Lemma 5, since (as is easily checked) $\Psi_X(d_X) \circ \eta'_{HX} = d_X \circ \eta'_{HX}$.

Example A ctd. The map λ^r_X, given a term (together with the behaviour of all variables from X, if applicable), unfolds it using rule r until the resulting term can show some behaviour according to the distributive law λ. With definition of r as shown at the end of Section 4, one has for example

$$\begin{aligned} \lambda^r_0(\texttt{loop a}) &= \{(a, \texttt{loop a})\} \\ \lambda^r_0(\texttt{loop(a + b)}) &= \{(a, \texttt{loop(a + b)}), (b, \texttt{loop(a + b)})\} \\ \lambda^r_0(\texttt{loop nil}) &= \bot \end{aligned}$$

Example B ctd. With definition of r as shown at the end of Section 4, here one has

$$\begin{aligned} \lambda^r_0(\texttt{unfolding(a; a)}) &= \{(a, \texttt{a})\} \\ \lambda^r_0(\texttt{unfolding(a; nil)}) &= \{(a, \texttt{unfolding(a; nil)})\} \\ \lambda^r_0(\texttt{unfolding nil}) &= \bot \end{aligned}$$

Lemma 15 *The family* $(\lambda^r_X : T'HX \to BT'X)_{X\in C}$ *is a natural transformation from* $T'H$ *to* BT'*. Moreover, it is a distributive law of the pointed endofunctor* T' *over the copointed endofunctor* H*.*

Proof: Naturality is shown using square commutation by fixpoint induction (Lemma 2). For the base case, d is a natural transformation. For the induction step, assume a family $(f_X : T'HX \to BT'X)_{X\in C}$ to be natural in X. Then everything in the definition of $\Psi_X(f)$ is natural in X, so the family $(\Psi_X(f) : T'HX \to BT'X)_{X\in C}$ is also natural in X.

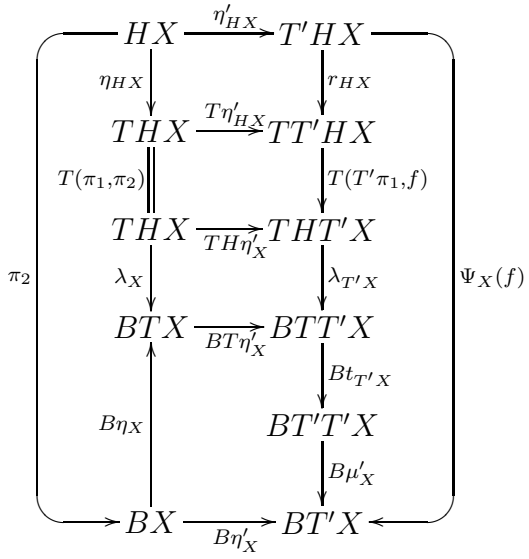
The distributive law axiom

$$\lambda^r_X \circ \eta'_{HX} = B\eta'_X \circ \pi_2$$

is shown by ordinary fixpoint induction on Ψ_X. The base case follows directly from definition of d_X. For the induction step, take a morphism $f : T'HX \to BT'X$, such that

$$f \circ \eta'_{HX} = B\eta'_X \circ \pi_2$$

Then everything in the diagram



commutes. □

Remark 16 In general λ^r is not a distributive law of the monad T over the copointed endofunctor H (for a counterexample, see Remark 12).

The natural transformation λ^r induces a mapping T'_{λ^r} on B-coalgebras. For a B-coalgebra $k : X \to BX$, define

$$T'_{\lambda^r}(k) = \lambda^r_X \circ T'(id, k)$$

Lemma 17 *T'_{λ^r} is an endofunctor lifting the pointed endofunctor T' to B-coalgebras.*

Proof: Functoriality follows easily from naturality of λ^r. The structure of pointed endofunctor follows from Lemma 15. □

The functor T'_{λ^r} may be also defined independently. For any object X, define the function $\Upsilon_X : C(T'X, BT'X) \to C(T'X, BT'X)$ as follows:

$$\Upsilon_X \frac{T'X \xrightarrow{f} BT'X}{T'X \xrightarrow[r_x]{} TT'X \xrightarrow[T_\lambda f]{} BTT'X \xrightarrow[Bt_{T'X}]{} BT'T'X \xrightarrow[B\mu'_X]{} BT'X}$$

It is easy to see that Υ_X is continuous on the cppo $C(T'X, BT'X)$, since T_λ acts as a continuous function on $C(T'X, BT'X)$ and C is **Cppo**-enriched.

For a given H-coalgebra $k : X \to BX$, define $k^\downarrow : T'X \to BT'X$

$$k^\downarrow = d_X \circ T'(id, k)$$

The following result gives an independent characterization of T'_{λ^r}:

Theorem 18 *For any B-coalgebra $k : X \to BX$,*

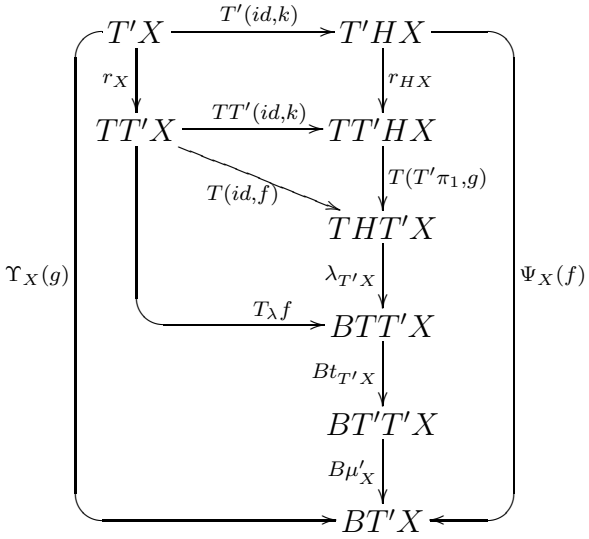
$$T'_{\lambda^r}(k) = \Upsilon^*_X(k^\downarrow)$$

Proof: By simultaneous fixpoint induction on Ψ_X and Υ_X. For the base case, one has to show that

$$d_X \circ T'(id, k) = k^\downarrow$$

which is just the definition of $k^\downarrow$.

For the induction step, consider two maps $f : TH'X \to BT'X$, $g : T'X \to BT'X$ such that $f \circ T'(id, k) = g$. Then the following diagram commutes:



□

Intuitively, for a given B-coalgebra $k : X \to BX$, the coalgebra $T'_{\lambda^r}k : T'X \to BT'X$ unfolds a given term from $T'X$ according to the unfolding map r, until the resulting term exhibits some behaviour according to the distributive law λ, assuming that variables in the terms show behaviour as defined by k.

Remark 19 From Remark 16 it follows that in general T'_{λ^r} does not lift the monad T' to the category of B-coalgebras: the diagram

$$\begin{array}{ccc} T'T'X & \xrightarrow{\mu'_X} & T'X \\ {\scriptstyle T'_{\lambda^r}T'_{\lambda^r}k}\downarrow & & \downarrow{\scriptstyle T'_{\lambda^r}k} \\ BT'T'X & \xrightarrow[B\mu'_X]{} & BT'X \end{array}$$

does not commute in general. This problem will be addressed in Section 8.

7 From unfolding rules to models

In the preceding sections it was shown how, given a distributive law λ of a monad T over a copointed endofunctor H, a translation $t : T \to T'$, and an unfolding rule $r : T' \to TT'$, one can derive the infinite unfolding $\bar{r} : T' \to T$ and the distributive law λ^r of the pointed endofunctor T' over the copointed endofunctor H. This section shows how the two latter notions are related, and how they allow to construct λ^r-models from λ-bialgebras and vice versa.

First, two theorems showing how λ, λ^r, t and $\bar{r}$ relate:

Theorem 20 *The diagram*

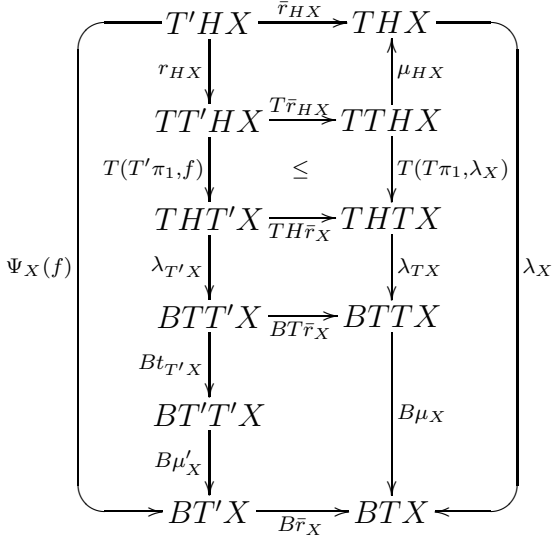
$$\begin{array}{ccc} T'H & \xrightarrow{\lambda^r} & BT' \\ {\scriptstyle \bar{r}H}\downarrow & & \downarrow{\scriptstyle B\bar{r}} \\ TH & \xrightarrow[\lambda]{} & BT \end{array}$$

commutes.

Proof: : First, prove that $B\bar{r}_X \circ \lambda^r_X \leq \lambda_X \circ \bar{r}_{HX}$ by fixpoint induction on Ψ_X.

Base case: $B\bar{r}_X \circ d_X \leq \lambda_X \circ \bar{r}_{HX}$ follows from definition of d_X and from Lemma 5, since (as is easily checked) $B\bar{r}_X \circ d_X \circ \eta'_{HX} = \lambda_X \circ \bar{r}_{HX} \circ \eta'_{HX}$.

Induction step: Assume $B\bar{r}_X \circ f \leq \lambda_X \circ \bar{r}_{HX}$ for some $f : T'HX \to BT'X$. Then the same inequality holds with f replaced by $\Psi_X(f)$, since the following diagram commutes:

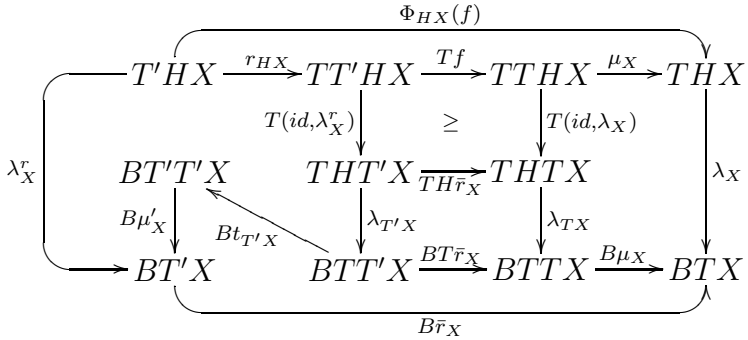


(The four regions in the middle of the diagram commute, counting from the top: by definition of $\bar{r}_X$, by the inductive assumption, by naturality of λ and by Lemma 14. The region to the right commutes since λ is a distributive law of the monad T over H.)

Then proceed to prove $B\bar{r}_X \circ \lambda^r_X \geq \lambda_X \circ \bar{r}_{HX}$ by fixpoint induction on Φ_{HX}.

Base case: $B\bar{r}_X \circ \lambda^r_X \geq \lambda_X \circ e_{HX}$ follows from definition of e_X and from Lemma 5, since (as is easily checked) $B\bar{r}_X \circ \lambda^r_X \circ \eta'_{HX} \geq \lambda_X \circ e_{HX} \circ \eta'_{HX}$

Induction step: Assume $B\bar{r}_X \circ \lambda^r_X \geq \lambda_X \circ f$ for some $f : T'X \to TX$. Then the same inequality holds with f replaced by $\Phi_{HX}(f)$. To prove this, consider the diagram



Here everything commutes (in particular, the large region to the left commutes by definition of λ^r, and the bottom region commutes by Lemma 14).

This completes the proof of Theorem 20. □

Theorem 21 *The diagram*

$$\begin{array}{ccc} TH & \xrightarrow{\lambda} & BT \\ {\scriptstyle tH}\downarrow & & \downarrow{\scriptstyle Bt} \\ T'H & \xrightarrow[\lambda^r]{} & BT' \end{array}$$

commutes.

Proof: To prove this, it is convenient to represent $\lambda_X : HTX \to BTX$ as a fixpoint of a certain function, resembling the function Ψ_X used to define λ^r_X in Section 6. The appropriate function $\tilde{\Psi}_X : C(THX, BTX) \to C(THX, BTX)$ acts as follows:

$$\tilde{\Psi}_X \frac{THX \xrightarrow{f} BTX}{THX \xrightarrow{\gamma_{HX}} TTHX \xrightarrow{T(T\pi_1, f)} THTX \xrightarrow{\lambda_{TX}} BTTX \xrightarrow{B\mu_X} BTX}$$

It is easy to see that $\tilde{\Psi}_X$ is continuous, since T is locally continuous, and C is **Cppo**-enriched.

Having defined $\tilde{d}_X = B\eta_X \circ \pi_2 \circ \upsilon_{HX}$, the following lemma holds:

Lemma 22 $\tilde{\Psi}^*_X(\tilde{d}_X) = \lambda_X$.

Proof: First, prove that $\tilde{\Psi}^*_X(\tilde{d}_X) \leq \lambda_X$ by fixpoint induction on $\tilde{\Psi}_X$.

Base case: $\tilde{d}_X \leq \lambda_X$ follows from definition of $\tilde{d}_X$ and from Lemma 5, since (as is easily checked) $\tilde{d}_X \circ \eta_{HX} = \lambda_X \circ \eta_{HX}$.

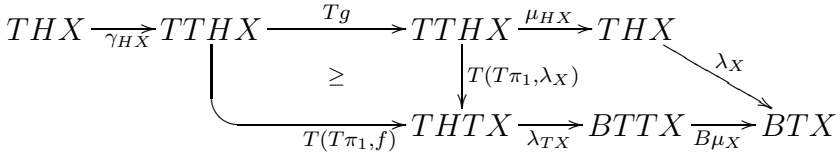
Induction step: Assume $f \leq \lambda_X$ for some $f : THX \to BTX$. Then also $\tilde{\Psi}_X(f) \leq \lambda_X$. To prove this, chase the diagram

$$\begin{array}{ccccc} THX & & & & \\ {\scriptstyle \gamma_{HX}}\downarrow & & & & \\ TTHX & \xrightarrow{\mu_{HX}} & THX & & \\ {\scriptstyle T(T\pi_1,f)}\downarrow & \leq & {\scriptstyle T(T\pi_1,\lambda_X)}\downarrow & \searrow^{\lambda_X} & \\ THTX & \xrightarrow[\lambda_{TX}]{} & BTTX & \xrightarrow[B\mu_X]{} & BTX \end{array}$$

To prove that $\tilde{\Psi}^*_X(\tilde{d}_X) \geq \lambda_X$, recall from Section 4 that $\tilde{\Phi}^*_X(\tilde{e}_X) = id_{TX}$ for any object X and prove $\tilde{\Psi}^*_X(\tilde{d}_X) \geq \lambda_X \circ \tilde{\Phi}^*_{HX}(\tilde{e}_{HX})$ by simultanous induction on $\tilde{\Psi}_X$ and $\tilde{\Phi}_{HX}$.

Base case: Immediate by definition of $\tilde{e}_{HX}$ and $\tilde{d}_X$.

Induction step: Take two morphisms $f : THX \to BTX$, $g : THX \to THX$ such that $f \geq \lambda_X \circ g$. Without any loss of generality, assume that $g \leq \tilde{\Phi}^*_{HX}(\tilde{e}_{HX}) = id_{THX}$. Then chase the diagram

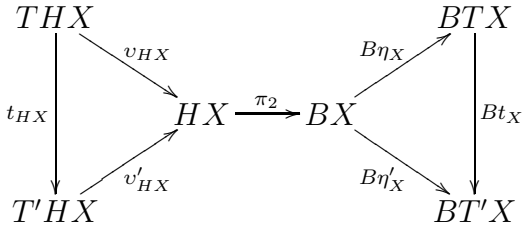


To show the middle triangle commute use the fact that $g \leq id_{THX}$, hence $T\pi_1 \circ g \leq T\pi_1$.

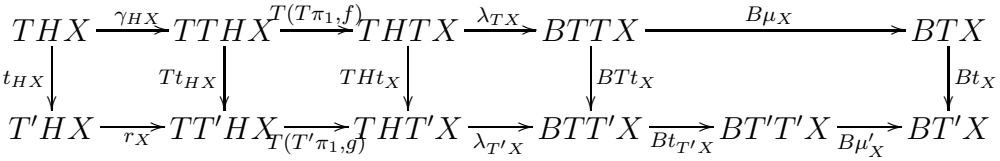
The lemma follows from Lemma 2. □

The proof of Theorem 21 proceeds componentwise by induction on $\tilde{\Psi}_X$ and Ψ_X.

Base case: See definitions of d_X and $\tilde{d}_X$, and chase the diagram



Induction step: Take two morphisms $f : THX \to BTX$, $g : T'HX \to BT'X$ such that $g \circ t_{HX} = Bt_X \circ f$. Then the following diagram commutes:

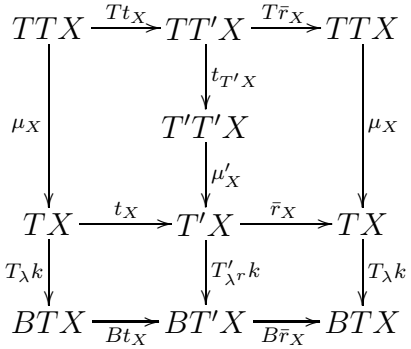


In particular, the leftmost square commutes by Lemma 9.

This, by Lemma 2, completes the proof of Theorem 21. □

From Theorems 20 and 21, a few important corollaries can be drawn:

Corollary 23 *For any B-coalgebra $k : X \to BX$, the following diagram commutes:*



Proof: The bottom left square commutes by Theorem 21, the bottom right

square — by Theorem 20, the top right region — by Lemma 14, and the top left region — since t is a monad morphism. □

The coalgebraic part of Corollary 23 means that interpreting a term from $T'X$ by unfolding it infinitely with $\bar{r}_X$, and then by any λ-bialgebra morphism from TX, is adequate with respect to the operational model $T'_{\lambda^r}k$, which unfolds a given term only until it exhibits some behaviour. The algebraic part of the corollary means that the map $\bar{r}_X$ is compositional with respect to syntax T (it is a T-algebra morphism).

Remark 24 Note that $\bar{r}_X$ is not fully compositional in general (for a counterexample, see Remark 12). This problem will be addressed in Section 8.

Remark 25 The structure

$$TT'X \xrightarrow{t_{T'X}} T'T'X \xrightarrow{\mu'_X} T'X \xrightarrow{T'_{\lambda^r}k} BT'X$$

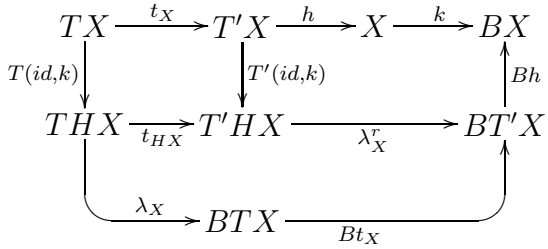
is not a λ-bialgebra in general. Therefore Corollary 23 cannot be read as showing that t_X and $\bar{r}_X$ are λ-bialgebra morphisms.

Corollary 26 *For any λ^r-model* $T'X \xrightarrow{h} X \xrightarrow{k} BX$,

$$TX \xrightarrow{t_X} T'X \xrightarrow{h} X \xrightarrow{k} BX$$

is a λ-bialgebra.

Proof: The diagram



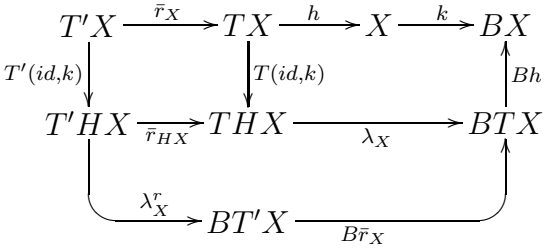
commutes by Theorem 21, and by naturality of t. □

Corollary 27 *For any λ-bialgebra* $TX \xrightarrow{h} X \xrightarrow{k} BX$,

$$T'X \xrightarrow{\bar{r}_X} TX \xrightarrow{h} X \xrightarrow{k} BX$$

is a λ^r-model.

Proof: The diagram



commutes by Theorem 20 and by naturality of $\bar{r}$.

Corollaries 26 and 27 mean that any model of the recursion-free fragment of a language can be used to interpret the full language, and vice versa, along syntactic translations of $\bar{r}$ and t respectively.

The fact that λ^r is not a distributive law of the monad T' over H is the only reason that prevents t and $\bar{r}$ from being morphisms of distributive laws in the sense of [20].

8 Regular unfolding rules

In the preceding sections, recursive constructs were added to languages by means of unfolding rules $r : T' \to TT'$. This notion is quite general, covering rather strange examples, e.g., the `unfolding` construct. This generality also led to some technical problems:

- $\lambda^r : T'H \to BT'$ is not a distributive law of the monad T' over the copointed endofunctor H,
- $\bar{r} : T' \to T$ is not a monad morphism,
- the maps $\bar{r}_X$ are not compositional.

In this section these problems will be remedied by considering a special class of *regular unfolding rules*. These exclude the example B shown throughout this paper, but still cover many examples of recursive equations, including many unguarded ones (in particular, the example A).

For this purpose, the general framework described so far is specialized with the following additional assumptions:

- T is the monad freely generated by an endofunctor Σ
- T' is the monad freely generated by an endofunctor Σ'
- $t : T \to T'$ is the monad morphism generated by a natural transformation $t : \Sigma \to \Sigma'$
- $r : T' \to TT'$ is induced from a natural transformation

  $$r^0 : \Sigma' \to TT'$$

  such that

  $$r^0 \circ t = T\eta' \circ \phi$$

  where $\phi : \Sigma \to T$ is the inclusion arising from the free monad structure of T.

  Given such r^0, the transformation r is derived according to the formula

  $$r = [(\eta T' \circ \eta'), (T\mu' \circ r^0 T')] \circ \iota'$$

  where $\iota' : T' \to Id + \Sigma' T'$ is the isomorphism arising from the free monad structure of T'.

It is easy to check that r defined this way satisfies the definition of an unfolding rule, presented in Section 4, and is based on the decomposition structure $\gamma : T \to TT$ defined as

$$\gamma = [\eta T \circ \eta, \phi T] \circ \iota$$

Unfolding rules r derived as above will be called *regular*.

Note that since T' is freely generated by Σ', its unit η' is an upper section and it can be canonically equipped with a left inverse $\upsilon' : T' \to Id$:

$$\upsilon' = [id, \bot] \circ \iota'$$

The same holds for T.

Example A ctd.: Definition of an appropriate r^0 is immediately derived from definitions of γ^0 and r given in example A in Section 4.

Example B ctd.: Note that the intended recursive equation for the `unfolding` construct cannot be captured as a regular unfolding rule. Indeed, consider two terms t_1, t_2. If an unfolding rule r is derived from a natural transformation $r^0 : \Sigma' \to TT'$ as above, then the term $r(\texttt{unfolding}\ t_2)$ should be obtained from $r(\texttt{unfolding}\ t_1)$ just by replacing every copy of t_1 with t_2. This is, however, not the intended behaviour of `unfolding`: as a counterexample, consider $t_1 = \texttt{a};\texttt{nil}$, $t_2 = \texttt{a};\texttt{a}$, where $r(\texttt{unfolding}\ t_1) = \texttt{a};\texttt{unfolding}\ t_1$ and $r(\texttt{unfolding}\ t_2) = t_2$.

To remedy the technical problems encountered in previous sections, some technical lemmas are needed:

Lemma 28 *The following diagram commutes:*

$$\begin{array}{ccc} \Sigma'T' & \xrightarrow{r^0T'} & TT'T' \\ {\scriptstyle\psi'}\downarrow & & \downarrow{\scriptstyle T\mu'} \\ T' & \xrightarrow[r]{} & TT' \end{array}$$

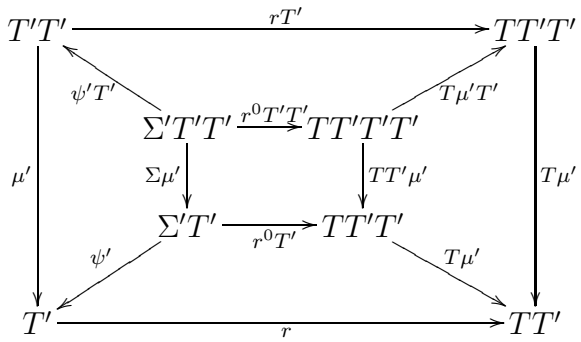
where $\psi' : \Sigma'T' \to T'$ is the inclusion arising from the free monad structure of T'.

Proof: Immediate by definition of r from r^0. □

Lemma 29 *For a regular r, the square as in the following diagram is made commute by the map $\psi'T'$:*

$$\begin{array}{ccccc} \Sigma'T'T' & \xrightarrow{\psi'T'} & T'T' & \xrightarrow{rT'} & TT'T' \\ & & {\scriptstyle\mu'}\downarrow & & \downarrow{\scriptstyle T\mu'} \\ & & T' & \xrightarrow[r]{} & TT' \end{array}$$

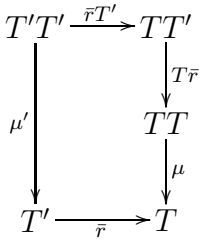
Proof: Everything in the following diagram commutes.



The top and the bottom squares commute by Lemma 28, the middle square — by naturality of r^0, and the left square — by the inductive definition of μ' in the freely generated monad T'. □

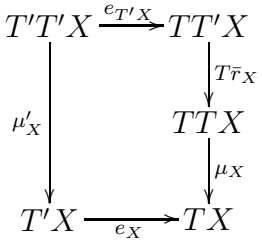
Theorem 30 *For a regular r, the transformation $\bar{r}$ is a monad morphism from T' to T.*

Proof: One of the laws for the monad morphism for $\bar{r}$ was proved to hold in Lemma 11. The remaining law is



To prove this, proceed componentwise using square commutation by fixpoint induction (Lemma 2).

For the base case, it must be shown that that the diagram



commutes. Given that T' is freely generated by Σ', it is enough to show that the diagram commutes when precomposed with $\eta'_{T'X} : T'X \to T'T'X$ or with $\psi'_{T'X} : \Sigma'T'T'X \to T'T'X$. Both cases are checked easily (in the latter case, both sides of the diagram are equal to the bottom element of the homset $C(\Sigma'T'T'X, TX)$).

For the induction step, take two morphisms $f : T'T'X \to TT'X$, $g : T'X \to TX$ and assume that the diagram

$$\begin{array}{ccc} T'T'X & \xrightarrow{f} & TT'X \\ \downarrow{\scriptstyle \mu'_X} & & \downarrow{\scriptstyle T\bar{r}_X} \\ T'X & & TTX \\ & \searrow_{g} & \downarrow{\scriptstyle \mu_X} \\ & & TX \end{array}$$

commutes. Then the diagram

$$\begin{array}{ccccccc} T'T'X & \xrightarrow{r_{T'X}} & TT'T'X & \xrightarrow{Tf} & TTT'X & \xrightarrow{\mu_{T'X}} & TT'X \\ \downarrow{\scriptstyle \mu'_X} & & \downarrow{\scriptstyle T\mu'_X} & & \downarrow{\scriptstyle TT\bar{r}_X} & & \downarrow{\scriptstyle T\bar{r}_X} \\ T'X & \xrightarrow[r_X]{} & TT'X & \xrightarrow[Tg]{} & TTX & \xrightarrow[\mu_X]{} & TX \\ & & & & \downarrow{\scriptstyle T\mu_X} & & \downarrow{\scriptstyle \mu_X} \\ & & & & TTX & \xrightarrow[\mu_{TX}]{} & TX \end{array}$$

commutes when precomposed with $\eta'_{T'X} : T'X \to T'T'X$ or with $\psi'_{T'X} : \Sigma'T'T'X \to T'T'X$. In particular, the left square commutes when precomposed with $\psi'_{T'X} : \Sigma'T'T'X \to T'T'X$ by Lemma 29.

This completes the proof of Theorem 30. □

Theorem 31 *For a regular r, λ^r is a distributive law of the monad T' over the copointed endofunctor H.*

Proof: One of the laws for the distributive law for λ^r was proved to hold in Lemma 15. The remaining law is

$$\begin{array}{ccccc} T'T'H & \xrightarrow{T'(T'\pi_1,\lambda^r)} & T'HT' & \xrightarrow{\lambda^r T'} & BT'T' \\ \downarrow{\scriptstyle \mu'H} & & & & \downarrow{\scriptstyle B\mu'} \\ T'H & & \xrightarrow[\lambda^r]{} & & BT' \end{array}$$

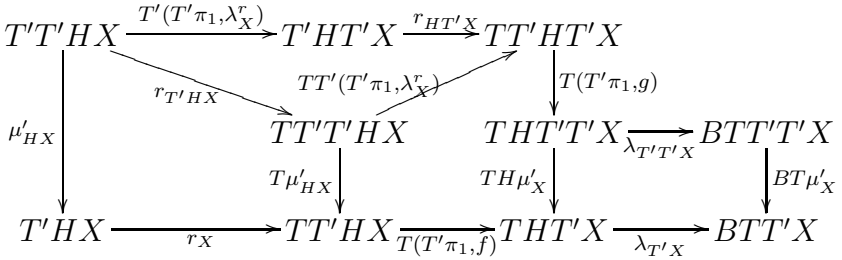
To prove this, proceed componentwise by induction on $\Psi_{T'X}$ and Ψ_X using Lemma 2.

Base case: It is enough to show that the following diagram commutes:

$$\begin{array}{ccccc} T'T'HX & \xrightarrow{T'(T'\pi_1,\lambda^r_X)} & T'HT'X & \xrightarrow{d_{T'X}} & BT'T'X \\ \downarrow{\scriptstyle \mu'_{HX}} & & & & \downarrow{\scriptstyle B\mu'_X} \\ T'HX & & \xrightarrow[d_X]{} & & BT'X \end{array}$$

As in Theorem 30, it is enough to show that this diagram commutes when precomposed with $\eta'_{T'HX}$ or with $\psi'_{T'HX}$. Both cases are checked easily, by definition of d.

Induction step: Take two morphisms $f : T'HX \to BTX$ and $g : T'HT'X \to BT'T'X$ such that $f \circ \mu'_{HX} = B\mu'_X \circ g \circ T'(T'\pi_1, \lambda^r_X)$. Then chase the diagram



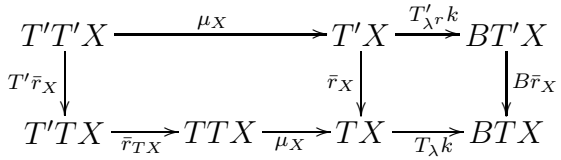
precomposing it with $\eta'_{T'HX} : T'HX \to T'T'HX$ and with $\psi'_{T'HX} : \Sigma'T'T'HX \to T'T'HX$. In particular, the bottom left square precomposed with $\psi'_{T'HX}$ commutes by Lemma 29. The middle pentagon commutes by the inductive assumption, and the upper square — by naturality of r.

This, by Lemma 2, completes the proof. □

Theorems 30 and 31 allow to rephrase many results from previous sections in a more structured fashion, provided the unfolding rule r is regular. Theorems 21 and 20 say now that t and $\bar{r}$ are morphisms of distributive laws in the sense of [20]. In Corollaries 26 and 27 one can replace λ^r-models with λ^r-bialgebras. Also the functor T'_{λ^r} lifts the monad structure of T' to the category of B-coalgebras, as an easy corollary from Theorem 31.

Moreover, for regular r the map $\bar{r}$ is fully compositional:

Corollary 32 *If r is regular, then for any B-coalgebra $k : X \to BX$, the map $\bar{r}_X$ is a λ^r-bialgebra morphism as shown in the diagram:*



Proof: The algebraic part follows from Theorem 30, and the coalgebraic part — from Corollary 23. □

9 Concluding remarks and future work

We have seen how to fit a rather general class of recursive constructs into the bialgebraic semantic framework developed by Turi and Plotkin. The behaviour of recursive constructs is not modelled with operational rules, but with separate recursive equations, formalised as certain natural transformations called unfolding rules. These equations are then merged, using certain fixpoint constructions, with natural transformations corresponding to the operational semantics of the recursion-free fragment of the language in question.

The result is particularly well-structured if the original unfolding rules satisfy an additional property, called regularity. Regular unfolding rules, when merged with distributive laws emerging from operational rules, yield new distributive laws, giving bialgebraic semantics to the language extended with recursive constructs.

The framework presented here is quite general and covers many examples of recursive equations, including many unguarded ones. As such equations can be a source of partiality (divergence), it is convenient to interpret them in a suitable **Cppo**-enriched category. This allows to perform the fixpoint constructions needed to merge the equations with recursion-free operational rules.

Working in **Cppo**-enriched categories makes dealing with recursive constructs simpler, but there is a valid question whether and to what extent a similar framework can be realized in the category of sets and functions, without imposing any additional (and always unwelcome) structure on the programs considered. A particularly interesting option is to interpret recursive equations using completely iterative monads generated by syntactic functors [3], rather than usual freely generated monads. In this framework, one can find solutions to recursive equations, under the mild assumption that the equations are ideal. This makes it rather straightforward, given an ideal recursive equation formalized as an unfolding rule $r : T' \to TT'$, to construct the infinite unfolding map $\bar{r}$ using universal properties of final coalgebras rather than existence of fixpoints. Therefore it seems that the framework presented in Section 5 can be realized in the category of sets, using completely iterative monads.

However, it is less clear how to apply the same idea to the developments presented in Sections 6 and 7. This is due to divergence that may arise from unguarded recursive equations. Indeed, consider a pathological looping construct `loop2` with the intended behaviour captured by the (ideal) recursive equation

$$\texttt{loop2}\ t = \texttt{loop2}\ t;\ t$$

formalised as an appropriate unfolding rule r. Using properties of iterative monads, one can then define the infinite unfolding map $\bar{r}$ and, e.g.,

$$\bar{r}_0(\texttt{loop2 a}) = (((\ldots;\texttt{a});\texttt{a});\texttt{a});\texttt{a})$$

However, it is much harder to define the action of the distributive law λ_0^r on the term `loop2 a`, since this term can never show any real behaviour. It is an open problem whether this drawback can be avoided and whether all developments presented in this paper can be realized without resorting to **Cppo**-enriched categories.

Another line of future work is to extend known applications of the bialgebraic framework to cover recursive constructs formalized as regular unfolding rules.

For example, in a recent paper [6] we showed how to use bialgebraic methods to obtain syntactic formats for structural operational semantics that guarantee congruence properties for various known operational preorders for processes. It will be interesting to see how recursive equations can be incorporated in such formats.

Finally, the framework presented here must be merged with presheaf methods, developed by Fiore, Plotkin and Turi and aimed at describing variable-binding syntax. In particular, it must be checked whether the non-regular unfolding rule for the construct `unfolding` considered in this paper becomes regular when interpreted in an appropriate presheaf category. If this is true, the next step will be to consider the general recursive construct μ.

References

[1] S. Abramsky, *A domain equation for bisimulation*, Inform. and Comput. **92** (1991), 161-218.

[2] L. Aceto, W.J. Fokkink and C. Verhoef, *Structural operational semantics*, Handbook of Process Algebra, J.A. Bergstra, A. Ponse and S.A. Smolka, eds, Elsevier, Amsterdam (2001), 197-292.

[3] P. Aczel, J. Adámek and J. Velebil, *A coalgebraic view of infinite trees and iteration*, Proc. CMCS'01, ENTCS **44.1** (2001).

[4] J.A. Bergstra, W.J. Fokkink and A. Ponse, *Process algebra with recursive operations*, Handbook of Process Algebra, J.A. Bergstra, A. Ponse and S.A. Smolka, eds, Elsevier, Amsterdam (2001), 333-385.

[5] B. Bloom, S. Istrail and A.R. Meyer, *Bisimulation can't be traced*, J. ACM **42** (1995), 232-268.

[6] B. Klin and P. Sobociński, *Congruence formats for free: an abstract approach to process equivalence*, Proc. 14th CONCUR Conf., LNCS 2761, Springer-Verlag (2003).

[7] M. Fiore, G. Plotkin and D. Turi, *Abstract syntax and variable binding*, Proc. 14th LICS Conf., IEEE Computer Society Press (1999), 193-202.

[8] M. Fiore and D. Turi, *Semantics of name and value passing*, Proc. 16th LICS Conf., IEEE Computer Society Press (2001), 93-104.

[9] J.A. Goguen, J.W. Thatcher, E.G. Wagner and J.B. Wright, *Initial algebra semantics and continuous algebras*, J. Assoc. Comput. Mach. **24** (1977), 68-95.

[10] M. Lenisa, J. Power and H. Watanabe, *Distributivity for endofunctors, pointed and co-pointed endofunctors, monads and comonads*, ENTCS **33** (2000).

[11] S. Mac Lane, *Categories for the Working Mathematician*, Springer-Verlag, 1971.

[12] S.B. Lassen, P.D. Mosses and D.A. Watt, *An introduction to AN-2, the proposed new version of Action Notation*, Proc. 3rd Intl. Workshop on Action Semantics, BRICS Note NS-00-6, Aarhus University (2000), 19-36.

[13] G. Plotkin, *A structural approach to operational semantics*, Report DAIMI FN-19, Computer Science Department, Aarhus University (1981).

[14] J. Rothe and D. Mašulović, *Towards weak bisimulation for coalgebras*, ENTCS **68.1**, Elsevier Science (2002).

[15] J. Rutten, *A note on coinduction and weak bisimilarity for while programs*, Report SEN-R9826, CWI, Amsterdam (1998).

[16] J. Rutten and D. Turi, *Initial algebra and final coalgebra semantics for concurrency*, Proc. of the REX workshop A Decade of Concurrency – Reflections and Perspectives, LNCS 803, Springer-Verlag (1994), 530-582.

[17] D. Turi, *Functorial operational semantics and its denotational dual*, Ph.D. Thesis, Vrije Universiteit, Amsterdam (1996).

[18] D. Turi, *Categorical modelling of structural operational rules: case studies*, Proc. 7th CTCS Conf., LNCS 1290, Springer-Verlag (1997), 127-146.

[19] D. Turi and G. Plotkin, *Towards a mathematical operational semantics*, Proc. 12th LICS Conf., IEEE Computer Society Press (1997), 280-291.

[20] H. Watanabe, *Well-behaved translations between structural operational semantics*, ENTCS **65.1**, Elsevier Science (2002).