Modular Grammars and Splitting of Catamorphisms

Eric Badouel, Rodrigue Djeumen

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Modular Grammars and Splitting of Catamorphisms

Éric Badouel — Rodrigue Djeumen

N° 6313

October 2007

Thème COM

ISSN 0249-6399 ISRN INRIA/RR--6313--FR+ENG

Modular Grammars and Splitting of Catamorphisms

Éric Badouel*, Rodrigue Djeumen†

Thème COM — Systèmes communicants
Projets S4

Rapport de recherche n° 6313 — October 2007 — 16 pages

Abstract: An abstract context-free grammar can be viewed as a system of polynomial functors. The initial algebra of this functor coincides with its least fixed-point; and this fixed-point can be computed by a method of substitution using Bekìc theorem. By doing so the system of polynomial functors is transformed into a related system of regular functors. We introduce a splitting operation on algebras producing an algebra for the resulting system of regular functors from an algebra of the original system of polynomial functors. This transformation preserves the interpretation function (catamorphism). The end result is a class of (extended) abstract context-free grammars, associated with regular functors. This class seems to be well-adapted to the modular design of domain-specific embedded languages.

Key-words: Context-Free Grammars, Catamorphisms, Bekìc Theorem, Modularity

* ebadouel@irisa.fr
† Rodrigue.Djeumen@irisa.fr

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 Rennes Cedex (France)
Téléphone : +33 2 99 84 71 00 — Télécopie : +33 2 99 84 71 71

Grammaires modulaires et scindage de catamorphismes

Résumé : Une grammaire algébrique abstraite peut être identifiée à un système de foncteurs polynomiaux. L'algèbre initiale de ce système coïncide avec son plus petit point fixe, et celui-ci peut être calculé par une méthode de substitution en utilisant le théorème de Bekìc . Ce système est alors transformé en un système associé de foncteurs réguliers. Nous introduisons une opération de scindage sur les algèbres qui reflète la méthode de résolution par substitution. Cette transformation préserve les fonctions d'interprétation (catamorphismes). Il en résulte une classe de grammaires algébriques abstraites étendues associée à la classe des foncteurs réguliers. Cette classe semble adaptée pour la conception modulaire de langages dédiés.

Mots-clés : Grammaires algébriques, catamorphismes, théorème de Bekìc, modularité

1 Introduction

Recent works on language oriented programming and generative programming [9, 4, 6, 3, 11] advocate the use of an intentional representation of a program, dissociated from its more or less partial concrete views, that can be manipulated by metaprogramming tools in order to edit, navigate, transform or extract information from this abstract representation. The core of such intensional representations are abstract syntax trees decorated with attributes. We can generate such trees, for instance through parsing or interactive edition, using anamorphisms associated with coalgebras. Symmetrically we can extract information from such an abstract syntax tree by interpretation in some algebra. For example the evaluation of attributes for a corresponding attribute grammar may be given by the catamorphism for an algebra canonically associated with the set of semantic rules [5]. Language oriented programming put also emphasis on the use of domain-specific languages (DSL), each of which is dedicated with specific aspects of an application domain. Combining such languages requires considering a global grammar such that each DSL is associated with some subgrammar. That global grammar need not be constructed explicitly but we should be able to generate (respectively to evaluate) its abstract syntax trees by combining anamorphisms (resp. catamorphisms) of the corresponding subgrammars.

In this paper we address this problem by introducing the so-called modular grammars. Since we are not interested in concrete syntaxes, what we term grammars will correspond to abstract context-free grammars. Such a grammar can be viewed as a system of polynomial functors. The initial algebra of this functor coincides with its least fixed-point; and this fixed-point can be computed by a method of substitution using Bekìc theorem [2]. By doing so the system of polynomial functors is transformed into a related system of regular functors. We introduce a splitting operation on algebras producing an algebra for the resulting system of regular functors from an algebra of the original system of polynomial functors. This transformation preserves the interpretation function (catamorphism). By duality we can derive a similar result for anamorphisms (generating functions).

We can illustrate our splitting operation on algebras with the following example that will be used as running example in this paper. We consider the type of trees given in Haskell by the following type definition

```
module Tree where
data Tree a = Node{val :: a, forest :: Forest a}
data Forest a = Nil | Cons{head :: Tree a, tail :: Forest a}
```

The related class of algebras and the corresponding evaluation function are given as follows

```
data Alg x a b = Alg{node :: x -> b -> a
                     , nil :: b
                     , cons:: a -> b -> b}
eval :: Alg x a b -> Tree x -> a
eval alg = (f,g) where
  f(Node x ys) = node alg x (g ys)
  g Nil = nil alg
  g (Cons y ys) = cons alg (f y) (g ys)
```

Let us write $([nil, cons, node])_{Tree} = Tree \cdot eval\ (Tree.Alg\ node\ nil\ cons)$ the corresponding evaluation function (catamorphism). A forest is a list of trees where the structure of lists can independently be given as:

```
module List where
data List a = Nil | Cons{head:: a, tail:: List a}
data Alg a b = Alg{nil:: b, cons:: a->b->b}
listmap :: (a->b) -> List a -> List b
listmap f = g where
  g Nil = Nil
  g (Cons a xs) = Cons (f a)(g xs)
instance Functor List where fmap = listmap
eval :: Alg a b -> List a -> b
eval alg = f
 where f Nil = nil alg
       f (Cons a xs) = cons alg a (f xs)
```

Here again we let $([nil, cons])_{List} = List \cdot eval\ (List \cdot Alg\ nil\ cons)$ denote the corresponding evaluation function. Then we can alternatively present trees as the so-called Rose trees given by the following Haskell module:

```
module Rose where
import List
data Rose a = Node{label::a, succ::Forest a}
type Forest a = List (Rose a)
rosemap :: (a->b) -> Rose a -> Rose b
rosemap f = g where
 g(Node x xs) = Node (f x)(List.fmap g xs)
instance Functor Rose where fmap = rosemap
data Alg a b = Alg{node::a-> List b -> b}
eval :: Rose.Alg a b -> Rose a -> b
eval alg (Node x ts) = Rose.node alg x (List.fmap (Rose.eval alg) ts)
```

Let $([node])_{Rose} = Rose \cdot eval\ (Rose \cdot Alg\ node)$. If *Rose.Alg node* is a Rose algebra then *Tree.Alg Nil Cons node* is a Tree algebra and

$$([Nil, Cons, node])_{Tree} = (([node])_{Rose}, List \cdot fmap\ ([node])_{Rose}) \qquad (1)$$

Indeed for any Tree algebra *Tree.Alg nil cons node* we notice that if $([nil, cons, node])_{Tree} = (f, g)$ then $g = ([nil, cons])_{List} \cdot (List \cdot fmap\ f)$ and $([Nil, Cons])_{List}$ is the identity function on lists. Thus any Rose catamorphism can be simulated by a Tree catamorphism. For the converse direction we associate any Tree algebra *Tree.Alg nil cons node* with the Rose algebra *Rose.Alg node′* where

$$node'\ x\ ys = node\ x\ (([nil, cons])_{List}\ ys)$$

and we show that

$$([nil, cons, node])_{Tree} = \left(([node'])_{Rose}, ([nil, cons])_{List} \cdot List \cdot fmap\ ([node'])_{Rose}\right) \qquad (2)$$

Notice that equation (1) appears as an instance of equation (2). We prove the latter by induction. At each induction step it is sufficient to establish that $([node'])_{Rose} = f$ where $([nil, cons, node])_{Tree} = (f, g)$ since as already noticed $g = ([nil, cons])_{List} \cdot (List \cdot fmap\ f)$. On the one hand

$$\begin{array}{rcl} ([node'])_{Rose}\,(Node\,x\,Nil) & = & node'\ x\ (List \cdot fmap\ ([node'])_{Rose}\ Nil) \\ & = & node'\ x\ Nil \\ & = & node\ x\ (([nil, cons])_{List}\ Nil) \\ & = & node\ x\ nil \\ & = & f\ (Node\ x\ Nil) \end{array}$$

On the other hand

$$\begin{array}{l} ([node'])_{Rose}\,(Node\ x\ (Cons\ y\ ys)) \\ = node'\ x\ (List \cdot fmap\ ([node'])_{Rose}\,(Cons\ y\ ys)) \\ = node'\ x\ (Cons\ (([node'])_{Rose}\ y)\ (List \cdot fmap\ ([node'])_{Rose}\ ys)) \\ = node\ x\ (([nil, cons])_{List}\ (Cons\ (([node'])_{Rose}\ y)\ (List \cdot fmap\ ([node'])_{Rose}\ ys))) \\ = node\ x\ cons\ ((([node'])_{Rose}\ y)\ (([nil, cons])_{List}\ (List \cdot fmap\ ([node'])_{Rose}\ ys))) \\ = node\ x\ cons\ (f\ y)\ (g\ ys) \\ = f\ (Node\ x\ (Cons\ y\ ys)) \end{array}$$

We have thus established that the modules Tree and Rose define the same class of catamorphisms (evaluation functions). By duality the same result may be established for anamorphisms (generating functions).

In this paper we generalize on this example and provide an application of this result to the definition of a modular family of grammars.

2 Abstract syntax : polynomial and regular functors

2.1 Abstract context-free grammars

Data types are usually defined as a fixed point of a system of mutually recursive equations. In a first stage we limit ourselves to polynomial systems of equations presented as abstract context-free grmmars.

Definition 1 An abstract context-free grammar $\mathbb{G} = (\mathcal{S}, \mathcal{P})$ consists of a finite set S of grammatical symbols (associated with the involved syntactic categories), and a finite set $\mathcal{P} \subseteq \mathcal{S} \times \mathcal{S}^*$of production rules . A production rule $P = \left(X_{P(0)}, X_{P(1)} \cdots X_{P(n)}\right)$ is written as $P : X_{P(0)} \rightarrow X_{P(1)} \cdots X_{P(n)}$; and we let $|P| = n$ denote the length of the right-hand side of the rule. A symbol $X \in S$ that does not appear as a left-hand side of any rule is a parameter of the grammar, the other grammatical symbols are said to be defined by the grammar.

For instance the grammar $\mathbb{T}$ defining trees has $\mathcal{S} = \{Tree, Forest, Label\}$ as set of grammatical symbols, and the following three production rules (*Label* is the only parameter of the grammar):

$$\begin{array}{lcll} Node & : & Tree & \rightarrow\ Label\quad Forest \\ Nil & : & Forest & \rightarrow\ () \\ Cons & : & Forest & \rightarrow\ Tree\quad Forest \end{array}$$

2.2 Functors associated with abstract context-free grammars

We let $\mathcal{C}$ stand for the category $p\mathcal{C}\mathcal{P}\mathcal{O}_\perp$ of pointed CPOs and strict continuous functions. Then the grammar $\mathbb{T}$ of trees can be associated with the polynomial functor $F_\mathbb{T} : \mathcal{C}^3 \to \mathcal{C}^2$ given by $F = \left\langle F_\mathbb{T}^{(Tree)}, F_\mathbb{T}^{(Forest)} \right\rangle$ and

$$\begin{array}{lcl} F_\mathbb{T}^{(Tree)}(A_{Label}, A_{Tree}, A_{Forest}) & = & (A_{Label} \times A_{Forest})_\perp \\ F_\mathbb{T}^{(Forest)}(A_{Label}, A_{Tree}, A_{Forest}) & = & ()_\perp \oplus (A_{Tree} \times A_{Forest})_\perp \end{array}$$

where $\times$ stands for the cartesian product of CPOs (which is also the categorical product), the lifting operator $(-)_\perp$ consists in adding a new least element to a given CPO: $A_\perp = A \uplus \{\perp\}$, and the coalesced sum $A \oplus B$ of two CPOs (the categorical coproduct in $\mathcal{C}$) is obtained from their disjoint union by identifying their respective least elements: $\perp_{A\oplus B} = \perp_A = \perp_B$. We shall let $\sum_{1\leq i\leq n} A_i = (A_1)_\perp \oplus \cdots \oplus (A_n)_\perp$ stand for the sum of n CPOs given alternatively by $\sum_{1\leq i\leq n} A_i = (A_1 \uplus \cdots \uplus A_n)_\perp$. When this sum has only two operands it will be written with an infix notation: $A + B = (A \uplus B)_\perp$. However we will pay attention to the fact that this binary operation is not associative and that the corresponding n-ary operation cannot be presented as an iterated application of the binary one: we rather have a family of operators indexed by non-negative integers. The unary sum coincides with the lifting operator and the nullary sum gives the CPO: $1 = ()_\perp = \{\perp, ()\}$. With these notations the above functors can be presented as:

$$\begin{array}{lcl} F_\mathbb{T}^{(Tree)}(A_{Label}, A_{Tree}, A_{Forest}) & = & (A_{Label} \times A_{Forest})_\perp \\ F_\mathbb{T}^{(Forest)}(A_{Label}, A_{Tree}, A_{Forest}) & = & 1 + (A_{Tree} \times A_{Forest}) \end{array}$$

More generally, we associate a grammar $\mathbb{G}$ with functor $F_\mathbb{G} = \left\langle F_\mathbb{G}^{(1)}, \ldots, F_\mathbb{G}^{(n)} \right\rangle$ where $F_\mathbb{G}^{(i)} : \mathcal{C}^{p+n} \to \mathcal{C}$ is the polynomial functor (sum of products) associated with the grammatical symbol X_{p+i} (we suppose that $X_1, \ldots, X_p$ are the parameters):

$$F_\mathbb{G}^{(i)}(A_1, \ldots, A_{p+n}) = \sum \left\{ A_{P(1)} \times \cdots \times A_{P(|P|)} \mid P \in \mathcal{P} \ s.t. \ P(0) = X_{p+i} \right\}$$

2.3 Initial algebra and catamorphisms

Without going into the technicality of the theory of recursive domain equations, to which we refer to [8, 1], we just recall the fact that a polynomial functor is locally continuous from which it follows that it admits a unique (parametric) fixed point $F^\dagger$. If $F : \mathcal{C}^{p+n} \to \mathcal{C}^n$ is a locally continuous functor and $\zeta \in |\mathcal{C}|^p$ is a domain of interpretation for the parameters, we let $in_{F,\zeta} : F\zeta\left(F^\dagger\zeta\right) \to F^\dagger\zeta$ and $out_{F,\zeta} : F^\dagger\zeta \to F\zeta\left(F^\dagger\zeta\right)$ stand for the inverse bijections associated with this fixpoint isomorphism. $\left(F^\dagger\zeta, in_{F,\zeta}\right)$ is the initial $F\zeta$-algebra which means that for any algebra $\varphi : F\zeta\alpha \to \alpha$, where $\alpha \in |\mathcal{C}|^n$ is a vector of interpretation domains for the variables of the system, there exists a unique morphism of $F\zeta$-algebras $([\varphi])_{F,\zeta} : \left(F^\dagger\zeta, in_{F,\zeta}\right) \to (\alpha, \varphi)$, called the *catamorphism* associated with φ . It is therefore the unique strict continuous map $([\varphi])_{F,\zeta} : F^\dagger\zeta \to \alpha$ such that $([\varphi])_{F,\zeta} \circ in_{F,\zeta} = \varphi \circ F\zeta([\varphi])_{F,\zeta}$, it is also the least fixed-point of the operator $\lambda h \cdot \left(\varphi \circ F\zeta h \circ out_{F,\zeta}\right)$.

By definition of functor $F_\mathbb{G}$, since $\oplus$ is the categorical coproduct of $\mathcal{C} = p\mathcal{C}\mathcal{P}\mathcal{O}_\perp$ and by observing that $p\mathcal{C}\mathcal{P}\mathcal{O}_\perp(A_\perp, B) \approx p\mathcal{C}\mathcal{P}\mathcal{O}(A, B)$ where $p\mathcal{C}\mathcal{P}\mathcal{O}$ is the category of pointed CPOs and continuous functions, it comes that an algebra $\varphi : F_\mathbb{G}\zeta\alpha \to \alpha$

boils down to the following data: a domain of interpretation $X_i^{(\varphi)}$ for each grammatical symbol X_i ($X_i^{(\varphi)} = \zeta(i)$ for parameters X_i $1 \leq i \leq p$ and $X_{p+i}^{(\varphi)} = \alpha(i)$ for variables X_{p+i} $1 \leq i \leq n$) and a continuous function $P^\varphi : X_{P(1)}^{(\varphi)} \times \cdots \times X_{P(|P|)}^{(\varphi)} \to X_{P(0)}^{(\varphi)}$ associated with each production rule P. We associate the grammar $\mathbb{G}$ with a multi-sorted signature $\Sigma_\mathbb{G}$ whose sorts are the grammatical symbols and whose operators are the production rules where rule $P : X_{P(0)} \to X_{P(1)} \cdots X_{P(n)}$ is interpreted as an operator of arity $X_{P(1)} \times \cdots \times X_{P(n)} \to X_{P(0)}$. Then an $F_\mathbb{G}$-algebra is nothing but a continuous $\Sigma_\mathbb{G}$-algebra in the sense that all interpretation domains are pointed CPOs and interpretation functions are continuous functions, and the elements of the free algebra $F^\dagger\zeta$ are the finite or infinite terms built upon the signature $\Sigma_\mathbb{G}$, whose operators are canonically associated with the production rules of the grammar, together with their approximants.

Definition 2 *We let* $T(\Sigma_\mathbb{G}, X, \zeta) = \pi_i^n \circ F_\mathbb{G}^\dagger$ *denote the set of (finite and infinite) terms of type* $X = X_{p+i}$.

2.4 Type functors and regular functors

If $f : \zeta_1 \to \zeta_2$ is a change of parameters we let $F^\dagger(f)$ be defined as the catamorphism $F^\dagger(f) = \left(\left[in_{F,\zeta_2} \circ Ff\left(F^\dagger\zeta_2\right)\right]\right)_{F,\zeta_1}$. We easily check that with this definition $F^\dagger$ is a functor from $\mathcal{C}^p$ to $\mathcal{C}^n$. This functor, called *type functor*, is locally continuous. We can then iterate the construction and define the class of *regular functors* as the least family of functors from $\mathcal{C}^n$ to $\mathcal{C}^m$ that contains the projections and is closed by sum, product, composition and the formation of type functors. Any regular functors is thus also locally continuous.

3 Decomposition of catamorphisms

3.1 Modular grammars

We now address the problem of modular decomposition of abstract context-free grammars. Let us again consider the grammar $\mathbb{T}$ of trees

$$\begin{array}{lcll} Node & : & Tree & \to & Label & Forest \\ Nil & : & Forest & \to & () & \\ Cons & : & Forest & \to & Tree & Forest \end{array}$$

If we take in isolation the last two production rules we obtain, after a renaming of the grammatical symbols, the grammar $\mathbb{L}$ of lists

$$\begin{array}{lcll} Nil & : & List & \to & () & \\ Cons & : & List & \to & Elet & List \end{array}$$

with $Elet$ as a parameter. The theory of list presents sufficient interest in itself so that we wish to make a separate module out of it that can be imported by various other modules including the module of Trees. This module of lists will export the List functor type $List = F_\mathbb{L}^\dagger$. Using this import the module of Trees can be presented as follows (the so-called "Rose trees"):

$$\begin{array}{rcl} Node & : & Tree \to Label \quad Forest \\ Forest & = & List(Tree) \end{array}$$

This is a new context-free grammar extended with a local definition. The grammatical symbols defined by a local definition are termed local variables. Therefore *Forest* is a local variable. *Label* which is defined neither by a production rule nor by a local definition is a parameter.

Definition 3 *A modular grammar is a structure $\mathbb{G} = (\mathcal{S}, \mathcal{P}, \mathcal{D})$ that contains an abstract context-free grammar $(\mathcal{S}, \mathcal{P})$ whose set of grammatical symbols is divided into three disjoint sets $\mathcal{S} = \mathcal{S}_p \uplus \mathcal{S}_d \uplus \mathcal{S}_\ell$. Symbols in $\mathcal{S}_p = \{X_1, \ldots, X_p\}$ are the parameters of the grammar, symbols in $\mathcal{S}_d = \{X_{p+1}, \ldots, X_{p+n}\}$ are the defined variables and symbols in $\mathcal{S}_\ell = \{X_{p+n+1}, \ldots, X_{p+n+m}\}$ are the local variables. The defined variables are those that appear as the left-hand side of some production rule. Finally the set $\mathcal{D}$ of external definitions associates each local variable X_{p+n+i}, with $1 \leq i \leq m$, to a defining equation of the form $X_{p+n+i} = R^{(i)}(X_1, \ldots, X_{p+n})$ where $R^{(i)} : \mathcal{C}^{p+n} \to \mathcal{C}$ is a regular functor. Thus the parameters are those grammatical symbols that are defined neither by a set of production rules nor by a local definition. We let $F_{\mathcal{P}}^{(i)} : \mathcal{C}^{p+n+m} \to \mathcal{C}$ denote the polynomial functor associated with local variable X_{p+i} according to the set of production rules; it is given as*

$$F_{\mathcal{P}}^{(i)}(A_1, \ldots, A_{p+n+m}) = \sum \left\{ A_{P(1)} \times \cdots \times A_{P(|P|)} \mid P \in \mathcal{P} \text{ s.t. } P(0) = X_{p+i} \right\}$$

We let $F_{\mathcal{P}} = \left\langle F_{\mathcal{P}}^{(1)}, \ldots, F_{\mathcal{P}}^{(n)} \right\rangle : \mathcal{C}^{p+n+m} \to \mathcal{C}^n$ denote the functor associated with the set of productions $\mathcal{P}$, and similarly $F_{\mathcal{D}} = \left\langle R^{(1)}, \ldots, R^{(m)} \right\rangle : \mathcal{C}^{p+n} \to \mathcal{C}^m$ the functor associated with the set of local definitions $\mathcal{D}$. The modular grammar $\mathbb{G}$ is then associated with the regular functor

$$F_{\mathbb{G}} = F_{\mathcal{P}} \circ \langle id_{p+n}, F_{\mathcal{D}} \rangle : \mathcal{C}^{p+n} \to \mathcal{C}^n$$

We say that the modular grammar $\mathbb{G}$ imports the m regular functors $R^{(i)} : \mathcal{C}^{p+n} \to \mathcal{C}$ and exports the n type functors $F_{\mathbb{G}}^{\dagger(i)} = \pi_i^n \circ F_{\mathbb{G}}^{\dagger} : \mathcal{C}^p \to \mathcal{C}$.

For instance the modular grammar $\mathbb{R}$ of "Rose trees" is such that $\mathcal{S}_p = \{Label\}$, $\mathcal{S}_d = \{Tree\}$, $\mathcal{S}_l = \{Forest\}$, $\mathcal{P}$ consists of the unique production rule

$$Node \;:\; Tree \;\to\; Label \quad Forest$$

and $\mathcal{D}$ contains the following definition

$$Forest \;=\; List\ Tree$$

The functor associated with this modular grammar is the regular functor

$$F_{\mathbb{R}}(D_{Label}, D_{Tree}) \;=\; D_{label} \times List\,(D_{Tree})$$

It imports the definition of *List*, the type functor $List = F_{\mathbb{L}}^{\dagger}$ exported by the grammar $\mathbb{L}$:

$$\begin{array}{rcl} Nil & : & List \;\to\; () \\ Cons & : & List \;\to\; Elet \quad List \end{array}$$

Remark 4 *An $F_{\mathbb{G}}$-algebra $\varphi \in Alg_{F_{\mathbb{G}},\zeta}(\alpha)$ of parameter $\zeta \in (|\mathcal{C}|)^p$ and domain $\alpha \in (|\mathcal{C}|)^n$, i.e. a morphism $\varphi : F_{\mathbb{G}}\zeta\alpha \to \alpha$, consists of a domain of interpretation $X_i^{(\varphi)} \in |\mathcal{C}|$ associated with each grammatical symbols $X_i \in \mathcal{S}_p \cup \mathcal{S}_d$ (i.e. $1 \leq i \leq p+n$) from which*

domains $X_i^{(\varphi)} = R^{(j)}\left[X_k^{(\varphi)}/X_k\right]_{1\le k\le p+n}$ *for* $X_i \in \mathcal{S}_\ell$ *(i.e.* $i = p+n+j$ *and* $1 \le j \le m$*) can be defined, together with a continuous mapping* $P^\varphi : X_{P(1)}^{(\varphi)} \times \cdots \times X_{P(|P|)}^{(\varphi)} \to X_{P(0)}^{(\varphi)}$ *associated with each production rule* P*.*

As for Definition 2 we let

Definition 5 *We let* $T(\Sigma_\mathbb{G}, F_\mathcal{D}, X, \zeta) = \pi_i^n \circ F_\mathbb{G}^\dagger$ *denote the set of (finite and infinite) macro terms of type* $X = X_i$*.*

The elements of the free algebra $in_{F_\mathbb{G},\zeta}$ can then be presented as follows.

$$X_i^{in_{F_\mathbb{G},\zeta}} = \begin{cases} \zeta_i & 1 \le i \le p \\ T(\Sigma_\mathbb{G}, F_\mathcal{D}, X_j, \zeta) & i = p+j \qquad 1 \le j \le n \\ R^{(j)}\left[X_k^{(in_{F_\mathbb{G},\zeta})}/X_k\right]_{1\le k\le p+n} & i = p+n+j \qquad 1 \le j \le m \end{cases}$$

We have coined the term "macro term" to designate an element in $T(\Sigma_\mathbb{G}, F_\mathcal{D}, X, \zeta)$ let us explain its structure. Such an element is of the form $P(v_1, \ldots, v_{|P|})$ where P is some production having X in the left-hand side and $v_i \in X_{P(i)}^{in_{F_\mathbb{G},\zeta}}$. Thus it can been seen as a tree whose root is labeled by production P and has $|P|$ subtrees given by $v_1, \ldots, v_{|P|}$. These successor nodes fall into three different categories: if $1 \le P(i) \le p$ then v_i is a parameter node labeled with the corresponding value $v_i \in \zeta_{P(i)}$; if $P(i) = p+j$ with $1 \le j \le n$, then it is an ordinary node labeled by some production rule attached to grammatical symbol $X_{P(i)}$; finally if $P(i) = p+n+j$ with $1 \le j \le m$, then it is a macro node labeled by an element in $R^{(j)}[X_k; 1 \le k \le p+n]$ where each occurrence of X_k represents a placeholder associated with the corresponding grammatical symbol. The arity of a macro node is given by the number of such placeholders and the subtree associated with a placeholder of type X_k is given by an element $v \in X_k^{in_{F_\mathbb{G},\zeta}}$.

According to the above representation of algebras, the evaluation function $([\varphi])\,\zeta : F_\mathbb{G}^\dagger \to \alpha$, characterized by the identity

$$([\varphi]) \circ in_{F_\mathbb{G},\zeta} = \varphi \circ F_\mathbb{G}\zeta\,([\varphi])$$

is given by a family of evaluation functions associated with the variables $X = X_i$ of the modular grammar $\mathbb{G}$ $(p+1 \le i \le p+n)$

$$eval_{X_i}^\varphi : T(\Sigma_\mathbb{G}, F_\mathcal{D}, X_i, \zeta) \to X_i^{(\varphi)}$$

the definition of $eval_X^\varphi$ has one clause associated with each production rule P having X in left-hand side:

$$\begin{array}{lll} eval_X^\varphi\, P(x_1, \ldots, x_{|P|}) = P^\varphi(v_1, \ldots, v_{|P|}) & & \\ \quad where & v_i = x_i & \text{if } 1 \le P(i) \le p \\ & v_i = eval_{X_{P(i)}}^\varphi\, x_i & \text{if } p+1 \le P(i) \le p+n \\ & v_i = map_{R^{(j)}}\left(\prod_{k=1}^{p+n} eval_{X_k}^\varphi\right) x_i & \text{if } P(i) = p+n+j \qquad 1 \le j \le m \end{array}$$

We associate each variable $X = X_i$ of the modular grammar $\mathbb{G}$ (i.e. $p+1 \le i \le p+n$) and each change of parameters $f = f_1 \times \cdots \times f_p : \zeta \to \zeta'$ with the function

$$map_X\ (f_1 \times \cdots \times f_p) : T(\Sigma_\mathbb{G}, F_\mathcal{D}, X, \zeta) \to T(\Sigma_\mathbb{G}, F_\mathcal{D}, X, \zeta')$$

whose definition has one clause associated with each production rule P having X as left-hand side:

$$\begin{array}{lll} map_X\ (f_1 \times \cdots \times f_p)\ P\left(x_1,\ldots,x_{|P|}\right) = P\ \left(v_1,\ldots,v_{|P|}\right) & & \\ \quad where & v_i = f_{P(i)}\ x_i & \text{if } 1 \le P(i) \le p \\ & v_i = map_{X_{P(i)}}\ f\ x_i & \text{if } p+1 \le P(i) \le p+n \\ & v_i = map_{R^{(j)}}\left(\prod_{k=1}^{p+n} map_{X_k}\right) x_i & \text{if } P(i) = p+n+j \qquad 1 \le j \le m \end{array}$$

For instance an $F_{\mathbb{R}}$-algebra consists of a pair of pointed CPOs A_{Label} and A_{Tree} (the domains of interpretation of parameter *Label* and defined variable *Tree* respectively) and of a continuous map $node : A_{Label} \times List\ A_{Tree} \to A_{tree}$. If *Tree* is the functor type $Tree = F_{\mathbb{R}^\dagger}$ and $in_{F_{\mathbb{R}},A_{Label}} : A_{Label} \times List\ (Tree\ A_{Label}) \to Tree\ A_{Label}$ the corresponding initial algebra, the catamorphism $([node]) : Tree\ A_{Label} \to A_{Tree}$ is characterized by the equation: $([node]) \circ in_{F_{\mathbb{R},A_{Label}}} = node \circ \left(id_{A_{Label}} \times List\ ([node])\right)$.

3.2 Splitting of modular grammars

We have split the set of productions of the grammar of trees into two parts leading us to two derived modular grammars. First, the grammar of lists was obtained by extraction of one of these subsets of production rules giving rise to an antonomous external module of lists. Second, the grammar of rose trees was obtained by replacing the extracted rules by local definitions (mainly the type functor) from the module of lists. We can generalize this operation as follows. Suppose $\mathbb{G} = (\mathcal{S}, \mathcal{P}, \mathcal{D})$ is a modular grammar. Up to isomorphism, this grammar is characterized by the system $F_{\mathcal{P}}(\mathbb{G}) : \mathcal{C}^{p+n+m} \to \mathcal{C}^n$ of polynomial functors associated with set $\mathcal{P}$ of productions of $\mathbb{G}$, and $F_{\mathcal{D}}(\mathbb{G}) : \mathcal{C}^{p+n} \to \mathcal{C}^m$ the (regular) functor associated with the set of local definitions $\mathcal{D}$ of $\mathbb{G}$. If $n = n_1 + n_2$ we let $\mathbb{G}_2 = \pi_2^{(n_1,n_2)}\mathbb{G}$ denote the modular grammar obtained by dropping all production rules corresponding to the first n_1 grammatical symbols which therefore become extra parameters. i.e. $\mathbb{G}_2$ has $p_2 = p + n_1$ parameters, n_2 defined variables and m local variables. That grammar is given by

$$\begin{array}{lll} F_{\mathcal{P}}(\mathbb{G}_2) = \pi_2^{(n_1,n_2)} \circ F_{\mathcal{P}}(\mathbb{G}) & : & \mathcal{C}^{(p+n_1)+n_2+m} \to \mathcal{C}^{n_2} \\ F_{\mathcal{D}}(\mathbb{G}_2) = F_{\mathcal{D}}(\mathbb{G}) & : & \mathcal{C}^{(p+n_1)+n_2} \to \mathcal{C}^m \end{array}$$

it is associated with the functor

$$F_{\mathbb{G}_2} = \left\langle F_{\mathcal{P}}^{(n_1+1)},\ldots,F_{\mathcal{P}}^{(n_1+n_2)}\right\rangle \circ \left\langle id_{p+n}, \left\langle R^{(1)},\ldots,R^{(m)}\right\rangle\right\rangle : \mathcal{C}^{p+n_1+n_2} \to \mathcal{C}^{n_2}$$

where $F_{\mathcal{P}}(\mathbb{G}) = \left\langle F_{\mathcal{P}}^{(1)},\ldots,F_{\mathcal{P}}^{(n)}\right\rangle$ and $F_{\mathcal{D}}(\mathbb{G}) = \left\langle R^{(1)},\ldots,R^{(m)}\right\rangle$. i.e. $F_{\mathbb{G}_2} = \pi_2^{(n_1,n_2)} \circ F_{\mathbb{G}}$. Now we let $\mathbb{G}/\mathbb{G}_2$ denote the grammar obtained from $\mathbb{G}$ when subgrammar $\mathbb{G}_2$ has been extracted, the corresponding variables have become new local variables and the set of local definitions have been adapted consequently. In detail $\mathbb{G}/\mathbb{G}_2$ has p parameters, n_1 defined variables and $m_2 = n_2 + m$ local variables. It is given by

$$\begin{array}{lll} F_{\mathcal{P}}(\mathbb{G}/\mathbb{G}_2) = \pi_1^{(n_1,n_2)} F_{\mathcal{P}}(\mathbb{G}) & : & \mathcal{C}^{p+n_1+(n_2+m)} \to \mathcal{C}^{n_1} \\ F_{\mathcal{D}}(\mathbb{G}/\mathbb{G}_2) = F_{\mathbb{G}_2}^{\dagger} \rtimes F_{\mathcal{D}}(\mathbb{G}) & : & \mathcal{C}^{p+n_1} \to \mathcal{C}^{n_2+m} \end{array}$$

where operation $\rtimes$ is given by

Definition 6 *The semidirect product (or cascaded composition) of functors $H : \mathcal{C}^p \to \mathcal{C}^n$ and $T : \mathcal{C}^{p+n} \to \mathcal{C}^m$ is given by*

$$H \rtimes T = \langle H, T \circ \langle id_p, H \rangle \rangle : \mathcal{C}^p \to \mathcal{C}^{n+m}$$

Thus the new local definitions are $X_{p+n_1+i} = R'^{(i)}(X_1, \ldots, X_{p+n_1})$ for $1 \leq i \leq n_2 + m$ where

$$\begin{array}{lcll} R'^{(i)}(X_1, \ldots, X_{p+n_1}) & = & S^{(i)}(X_1, \ldots, X_{p+n_1}) & 1 \leq i \leq n_2 \\ R'^{(n_2+i)}(X_1, \ldots, X_{p+n_1}) & = & R^{(i)}(X_1, \ldots, X_{p+n_1}) & \\ & & , S^{(1)}(X_1, \ldots, X_{p+n_1}) & \\ & & , \cdots & \\ & & , S^{(n_2)}(X_1, \ldots, X_{p+n_1})\Big) & 1 \leq i \leq m \end{array}$$

and $F^{\dagger}_{\mathbb{G}_2} = \left\langle S^{(1)}, \ldots, S^{(n_2)} \right\rangle$. The functor associated with this modular grammar is

$$F_{\mathbb{G}/\mathbb{G}_2} = \left\langle F^{(1)}_{\mathbb{G}}, \ldots, F^{(n_1)}_{\mathbb{G}} \right\rangle \circ \left\langle id_{p+n_1}, \left\langle R'^{(1)}, \ldots, R'^{(n_2+m)} \right\rangle \right\rangle : \mathcal{C}^{p+n_1} \to \mathcal{C}^{n_1}$$

Proposition 7 $F_{\mathbb{G}/\mathbb{G}_2} = F_{\mathbb{G}_1} \circ \left\langle id_{p+n_1}, F^{\dagger}_{\mathbb{G}_2} \right\rangle : \mathcal{C}^{p+n_1} \to \mathcal{C}^{n_1}$ *where*

$$F_{\mathbb{G}_1} = \left\langle F^1_{\mathcal{P}}, \ldots, F^{(n_1)}_{\mathcal{P}} \right\rangle \circ \left\langle id_{p+n}, \left\langle R^{(1)}, \ldots, R^{(m)} \right\rangle \right\rangle : \mathcal{C}^{p+n_1+n_2} \to \mathcal{C}^{n_1}$$

i.e. $F_{\mathbb{G}_1} = \pi_1^{(n_1,n_2)} \circ F_{\mathbb{G}}$.

The grammar of rose trees is thus the residual of the grammar of trees by the grammar of lists: $\mathbb{R} = \mathbb{T}/\mathbb{L}$. The equivalence between the functors $\left(F^{\dagger}_{\mathbb{T}}\right)^{(Tree)}$ and $\left(F^{\dagger}_{\mathbb{R}}\right)^{(Tree)}$ follows from Bekìc theorem [2] which we now recall.

3.3 Bekìc theorem

Definition 8 *A locally continuous functor $F : \mathcal{C}^{p+n} \to \mathcal{C}^n$ with $n = n_1 + n_2$ can be decomposed on the form $F = \langle F_1, F_2 \rangle$ where $F_1 = \pi_1^{(n_1,n_2)} \circ F : \mathcal{C}^{p+n} \to \mathcal{C}^{n_1}$ and $F_2 = \pi_2^{(n_1,n_2)} \circ F : \mathcal{C}^{p+n} \to \mathcal{C}^{n_2}$ where functors $\pi_1^{(n_1,n_2)} : \mathcal{C}^n \to \mathcal{C}^{n_1}$ and $\pi_2^{(n_1,n_2)} : \mathcal{C}^n \to \mathcal{C}^{n_2}$ are the two canonical projections. We successively let*

$$\begin{array}{lcll} F/F_2 & = & F_1 \circ \left\langle id_{p+n_1}, F^{\dagger}_2 \right\rangle & : \quad \mathcal{C}^{p+n_1} \to \mathcal{C}^{n_1} \\ H & = & (F/F_2)^{\dagger} & : \quad \mathcal{C}^p \to \mathcal{C}^{n_1} \\ F'_2 & = & F_2 \circ (\langle id_p, H \rangle \times id_{n_2}) & : \quad \mathcal{C}^{p+n_2} \to \mathcal{C}^{n_2} \\ K & = & F'^{\dagger}_2 & : \quad \mathcal{C}^p \to \mathcal{C}^{n_2} \end{array}$$

where $id_\ell : \mathcal{C}^\ell \to \mathcal{C}^\ell$ stands for the identity functor of $\mathcal{C}^\ell$.

Bekìc theorem asserts the following isomorphism:

$$F^{\dagger}\,\zeta = H\,\zeta \times K\,\zeta$$

It corresponds to the classical method of resolution by substitution. Indeed let $\mathbf{y}$, $\mathbf{x_1}$ and $\mathbf{x_2}$ be variables ranging respectively over $|\mathcal{C}|^p$, $|\mathcal{C}|^{n_1}$ and $|\mathcal{C}|^{n_2}$. Variable $\mathbf{x_1}$ of system F

becomes a parameter for its subsystem F_2. By solving the latter we obtain a parametric solution $F_2^\dagger : \mathcal{C}^{p+n_1} \to \mathcal{C}^{n_1}$. We substitute this solution to variable $\mathbf{x_2}$ in the system F_1 thus leading to a new system $F/F_2 = F_1 \circ \left\langle id_{p+n_1}, F_2^\dagger \right\rangle : \mathcal{C}^{p+n_1} \to \mathcal{C}^{n_1}$ in which variable $\mathbf{x_2}$ no longer appears. Solving this new system provide us with the $\mathbf{x_1}$ component of the solution of the original system thus given by $H = (F/F_2)^\dagger : \mathcal{C}^p \to \mathcal{C}^{n_1}$. We can substitute that value into F_2 in order to derive the system $F_2' = F_2 \circ (\langle id_p, H\rangle \times id_{n_1}) : \mathcal{C}^{p+n_2} \to \mathcal{C}^{n_2}$ whose resolution gives the $\mathbf{x_2}$ component of the solution of the original system. The following lemma says that the $\mathbf{x_2}$ component of the solution of the original system can alternatively be obtained by substituting the $\mathbf{x_1}$ component of the solution of the original system (given by H) in the parametric solution $F_2^\dagger : \mathcal{C}^{p+n_1} \to \mathcal{C}^{n_1}$. The condition expressed by this lemma appears in several axiomatizations of parametric fixed-point operators [10], and in particular in the theory of traced monoidal categories [7].

Lemma 9 $K\,\zeta \simeq F_2^\dagger\,\zeta\,(H\,\zeta)$

Proof. First notice that $F_2'\zeta\,(K\zeta) = F_2\zeta\,(H\zeta)\,(K\zeta)$. The initial F_2', ζ-algebra

$$in_{F_2',\zeta} : F_2\zeta\,(H\zeta)\,(K\zeta) \to K\zeta$$

is thus an F_2-algebra with parameters $\zeta \times H\zeta$. We let

$$\iota_1 = \left(\left[in_{F_2',\zeta}\right]\right)_{F_2,\zeta\times H\zeta} : F_2^\dagger\zeta\,(H\,\zeta) \to K\zeta$$

be the corresponding catamorphism which, by definition, satisfies

$$\iota_1 \circ in_{F_2,\zeta\times H\zeta} = in_{F_2',\zeta} \circ F_2\zeta\,(H\zeta)\,\iota_1$$

Symmetrically, since $F_2\zeta(H\zeta)\left(F_2^\dagger\zeta\,(H\zeta)\right) = F_2'\zeta\left(F_2^\dagger\zeta\,(H\zeta)\right)$, we deduce that the initial $F_2, \zeta\times H\zeta$-algebra $in_{F_2,\zeta\times H\zeta} : F_2\zeta(H\zeta)\left(F_2^\dagger\zeta\,(H\zeta)\right) \to F_2^\dagger\zeta\,(H\zeta)$ is an F_2', ζ-algebra. Let $\iota_2 = \left(\left[in_{F_2,\zeta\times H\zeta}\right]\right)_{F_2',\zeta} : K\zeta \to F_2^\dagger\zeta\,(H\zeta)$ denote the corresponding catamorphism which, by definition, satisfies $\iota_2 \circ in_{F_2',\zeta} = in_{F_2,\zeta\times H\zeta} \circ F_2\zeta(H\zeta)\iota_2$. On the one hand it follows

$$\begin{array}{rcl} \iota_1 \circ \iota_2 \circ in_{F_2',\zeta} & = & \iota_1 \circ in_{F_2,\zeta\times H\zeta} \circ F_2\zeta(H\zeta)\iota_2 \\ & = & in_{F_2',\zeta} \circ F_2\zeta\,(H\zeta)\,\iota_1 \circ F_2\zeta(H\zeta)\iota_2 \\ & = & in_{F_2',\zeta} \circ F_2\zeta\,(H\zeta)\,(\iota_1 \circ \iota_2) \\ & = & in_{F_2',\zeta} \circ F_2'\zeta\,(\iota_1 \circ \iota_2) \end{array}$$

and thus $\iota_1 \circ \iota_2 = \left(\left[in_{F_2',\zeta}\right]\right)_{F_2',\zeta} = id_{K\zeta}$. On the other hand

$$\begin{array}{rcl} \iota_2 \circ \iota_1 \circ in_{F_2,\zeta\times H\zeta} & = & \iota_2 \circ in_{F_2',\zeta} \circ F_2\zeta\,(H\zeta)\,\iota_1 \\ & = & in_{F_2,\zeta\times H\zeta} \circ F_2\zeta(H\zeta)\iota_2 \circ F_2\zeta\,(H\zeta)\,\iota_1 \\ & = & n_{F_2,\zeta\times H\zeta} \circ F_2\zeta(H\zeta)\,(\iota_2 \circ \iota_1) \end{array}$$

and thus $\iota_2 \circ \iota_1 = \left(\left[in_{F_2,\zeta\times H\zeta}\right]\right)_{F_2,\zeta\times H\zeta} = id_{F^\dagger\zeta(H\zeta)}$. The pair of morphisms $\iota_1 : F_2^\dagger\zeta\,(H\,\zeta) \to K\zeta$ and $\iota_2 : K\zeta \to F_2^\dagger\zeta\,(H\zeta)$ thus constitutes the required isomorphism $K\,\zeta \simeq F_2^\dagger\,\zeta\,(H\,\zeta)$. □

Corollary 10 $F^\dagger = (F/F_2)^\dagger \rtimes F_2^\dagger$

By Prop.7 one has $F_{\mathbb{G}/\mathbb{G}_2} = F_{\mathbb{G}}/F_{\mathbb{G}_2}$ where $\mathbb{G}$ is a modular grammar with p parameters, $n = n_1 + n_2$ defined variables, and m local variables, and $\mathbb{G}_2 = \pi_2^{(n_1,n_2)}\,\mathbb{G}$.

Corollary 11 $F_{\mathbb{G}}^\dagger = \left(F_{\mathbb{G}/\mathbb{G}_2}^\dagger\right) \rtimes F_{\mathbb{G}_2}^\dagger$

In particular the type of trees and rose trees are isomorphic: $\left(F_{\mathbb{T}}^\dagger\right)^{(Tree)} = \left(F_{\mathbb{R}}^\dagger\right)^{(Tree)}$.

3.4 Decomposition of algebras

Using Bekìc theorem we now define a decomposition of algebras.

Definition 12 *As in Def.8 we let $F : \mathcal{C}^{p+n} \to \mathcal{C}^n$ be a locally continuous functor with $n = n_1 + n_2$. Let moreover $\Phi : F\zeta\alpha_1\alpha_2 \to \alpha_1 \times \alpha_2$ be an $F\zeta$-algebra ($\zeta \in |\mathcal{C}^p|$) on the domain $\alpha = \alpha_1 \times \alpha_2$ ($\alpha_1 \in |\mathcal{C}^{n_1}|$, and $\alpha_2 \in |\mathcal{C}^{n_2}|$). Φ can be decomposed into*

$$\varphi_1 = \pi_1^{(n_1,n_2)}(\Phi) : F_1\ \zeta\ \alpha_1\ \alpha_2 \to \alpha_1 \quad \textit{and} \quad \varphi_2 = \pi_2^{(n_1,n_2)}(\Phi) : F_2\ \zeta\ \alpha_1\ \alpha_2 \to \alpha_2$$

The (n_1,n_2)-splitting of Φ is the pair consisting of the $(F/F_2)\,\zeta$-algebra of domain α_1

$$\pi_{F/F_2}\Phi \triangleq \varphi_1 \circ \left(F_1\ \zeta\ \alpha_1\ \left(|\varphi_2|\right)_{F_2,\zeta\times\alpha_1}\right) :\ F_1\ \zeta\ \alpha_1\ \left(F_2^\dagger\ \zeta\ \alpha_1\right) \to \alpha_1$$

together with the $F_2\,(\zeta \times \alpha_1)$-algebra of domain α_2

$$\pi_{F_2}\Phi \triangleq \varphi_2 : F_2\ \zeta\ \alpha_1\ \alpha_2 \to \alpha_2$$

The operation of decomposition of algebras is thus given as:

$$\begin{array}{l} \mathit{Split}^{(n.m)} :\ \mathit{Alg}_{F,\zeta}(\alpha_1 \times \alpha_2) \to \left(\mathit{Alg}_{F/F_2,\zeta}(\alpha_1)\right) \times \left(\mathit{Alg}_{F_2,\zeta\times\alpha_1}(\alpha_2)\right) \\ \mathit{Split}^{(n_1,n_2)}\ \Phi\ =\ \left(\pi_{F/F_2}\Phi, \pi_{F_2}\Phi\right) \end{array}$$

Thus an algebra $\Phi = \varphi_1 \times \varphi_2 : F\zeta\alpha_1\alpha_2 \to \alpha_1 \times \alpha_2$ is decomposed into an algebra $\pi_{F_2}\Phi = \varphi_2 : F_2\ \zeta\ \alpha_1\ \alpha_2 \to \alpha_2$ for the "subsystem" F_2 together with an algebra $\pi_{F/F_2}\Phi : F/F_2\zeta\alpha_1 \to \alpha_1$ for the "residual system" F/F_2. The following result shows that the catamorphism (evaluation function) associated with the algebra Φ for the overall system can be reconstructed from the catamorphisms associated respectively with $\pi_{F_2}\Phi$ and $\pi_{F/F_2}\Phi$ using some kind of semidirect product which we first introduce. In Definition 6 we defined the semidirect product of two functors $H :\ \mathcal{C}^p \to \mathcal{C}^n$ and $T :\ \mathcal{C}^{p+n} \to \mathcal{C}^m$ as

$$H \rtimes T = \langle H, T \circ \langle id_p, H\rangle\rangle :\ \mathcal{C}^p \to \mathcal{C}^{n+m}$$

By functoriality of the product and composition we deduce a related operation of semidirect product of natural transformations $\eta : H \xrightarrow{\bullet} H'$ and $\tau : T \xrightarrow{\bullet} T'$ where $H, H' :\ \mathcal{C}^p \to \mathcal{C}^n$ and $T, T' :\ \mathcal{C}^{p+n} \to \mathcal{C}^m$ given by

$$(\eta \rtimes \tau)_\zeta = \eta_\zeta \times \left(\tau_{\zeta,H'\zeta} \circ T\zeta\eta_\zeta\right) = \eta_\zeta \times \left(T'\zeta\eta_\zeta \circ \tau_{\zeta,H\zeta}\right)$$

Considering the special case where the target functors H' and T' are constant functors leads us to the following definition

Definition 13 *The semidirect composition of two maps* $f : H\zeta \to \alpha$ *and* $g : T\zeta\alpha \to \beta$ *where* $H : \mathcal{C}^p \to \mathcal{C}^n$ *and* $T : \mathcal{C}^{p+n} \to \mathcal{C}^m$ *is the map* $f \rtimes g : (H \rtimes T)\,\zeta \to \alpha \times \beta$ *given by* $(f \rtimes g) = f \times (g \circ T\zeta f)$.

Using this operation we can now state

Theorem 14 *Up to the isomorphisms* $F^\dagger\zeta = H\zeta \times K\zeta$ *and* $K\zeta = F_2^\dagger\zeta\,(H\zeta)$

$$([\Phi])_{F,\zeta} = \left([\pi_{F/F_2}\Phi]\right)_{F/F_2,\zeta} \rtimes ([\pi_{F_2}\Phi])_{F_2,\zeta\times\alpha_1}$$

With $F = F_{\mathbb{T}}$ and $F_2 = F_{\mathbb{L}}$ it comes that

$$([nil, cons, node])_{Tree} = \left(([node'])_{Rose}, ([nil, cons])_{List} \cdot List \cdot fmap\ ([node'])_{Rose}\right)$$

where

$$node'\ x\ ys\ =\ node\ x\ (([nil, cons])_{List}\ ys)$$

and thus the above result is a generalization of formula 2 discussed in the introduction.

Lemma 15 *Up to the isomorphism* $F^\dagger\zeta = H\zeta \times K\zeta$ *the initial algebra* $in_{F,\zeta} : F\zeta\left(F^\dagger\zeta\right) \to F^\dagger\zeta$ *decomposes on the form* $in_{F,\zeta} = in_{H,\zeta} \times in_{K,\zeta}$ *where* $in_{H,\zeta} : F_1\zeta(H\zeta)(K\zeta) \to H\zeta$ *and* $in_{K,\zeta} : F_2\zeta(H\zeta)(K\zeta) \to K\zeta$ *are respectively given by* $in_{H,\zeta} = in_{F/F_2,\zeta} \circ (F_1\zeta(H\zeta)\iota_2)$ *and* $in_{K,\zeta} = in_{F_2',\zeta}$.

Proof. The initial algebra is an isomorphism and the converse also holds true (any algebra which is an isomorphism is initial) when we have unicity of fixed-point (up to isomorphism) which is indeed the case here.

$$in_{H,\zeta} = in_{F/F_2,\zeta} \circ (F_1\zeta(H\zeta)\iota_2) : F_1\zeta(H\zeta)(K\zeta) \to H\zeta$$

and $in_{K,\zeta} = in_{F_2',\zeta} : F_2\zeta(H\zeta)(K\zeta) \to K\zeta$ are isomorphisms and thus

$$in_{H,\zeta} \times in_{K,\zeta} : F\zeta(H\zeta)(K\zeta) \to H\zeta \times K\zeta$$

is the initial algebra of functor F. □

Corollary 16 *Up to the isomorphism* $F^\dagger\zeta = H\zeta \times K\zeta$*, the two parts* $f : H\zeta \to \alpha_1$ *and* $g : K\zeta \to \alpha_2$ *of catamorphism* $([\Phi])_{F,\zeta} = f \times g$ *are characterized by* $f \circ in_{H,\zeta} = \varphi_1 \circ F\zeta fg$ *and* $g \circ in_{K,\zeta} = \varphi_2 \circ F_2\zeta fg$.

Lemma 17 *For any morphism* $f : H\zeta \to \alpha_1$ *one has*

$$([\varphi_2 \circ F_2\zeta f\alpha_2])_{F_2',\zeta} = ([\varphi_2])_{F_2,\zeta\times\alpha_1} \circ \left(F_2^\dagger\zeta f\right) \circ \iota_2 : K\zeta \to \alpha_2$$

and that morphism $g(f)$ *satisfies* $g(f) \circ in_{K,\zeta} = \varphi_2 \circ (F_2\,\zeta\,f\,g(f))$.

Proof. By definition $F_2^\dagger\zeta f = \left(\left[in_{F_2,\zeta\times\alpha_1} \circ \left(F_2\,\zeta\,f\,\left(F_2^\dagger\zeta\alpha_1\right)\right)\right]\right)_{F_2,\zeta\times H\zeta}$ and that morphism satisfies

$$F_2^\dagger\zeta f \circ in_{F_2,\zeta\times H\zeta} = in_{F_2,\zeta\times\alpha_1} \circ \left(F_2\,\zeta\,f\,\left(F_2^\dagger\zeta\alpha_1\right)\right) \circ F_2\zeta(H\zeta)\left(F_2^\dagger\zeta f\right)$$

It follows that

$$\begin{aligned}
& ([\varphi_2])_{F_2,\zeta\times\alpha_1} \circ \left(F_2^\dagger \zeta f\right) \circ \iota_2 \circ in_{F_2',\zeta} \\
= \;& ([\varphi_2])_{F_2,\zeta\times\alpha_1} \circ \left(F_2^\dagger \zeta f\right) \circ in_{F_2,\zeta\times H\zeta} \circ F_2\zeta(H\zeta)\iota_2 \\
= \;& ([\varphi_2])_{F_2,\zeta\times\alpha_1} \circ in_{F_2,\zeta\times\alpha_1} \circ \left(F_2\,\zeta\, f\, \left(F_2^\dagger \zeta\alpha_1\right)\right) \circ F_2\zeta(H\zeta) \left(F_2^\dagger \zeta f\right) \circ F_2\zeta(H\zeta)\iota_2 \\
= \;& \varphi_2 \circ F_2\zeta\alpha_1 ([\varphi_2])_{F_2,\zeta\times\alpha_1} \circ \left(F_2\,\zeta\, f\, \left(F_2^\dagger \zeta\alpha_1\right)\right) \circ F_2\zeta(H\zeta) \left(F_2^\dagger \zeta f \circ \iota_2\right) \\
= \;& \varphi_2 \circ F_2\zeta f\alpha_2 \circ F_2\zeta(H\zeta) ([\varphi_2])_{F_2,\zeta\times\alpha_1} \circ F_2\zeta(H\zeta) \left(F_2^\dagger \zeta f \circ \iota_2\right) \\
= \;& (\varphi_2 \circ F_2\zeta f\alpha_2) \circ F_2\zeta(H\zeta) \left(([\varphi_2])_{F_2,\zeta\times\alpha_1} \circ F_2^\dagger \zeta f \circ \iota_2\right)
\end{aligned}$$

and thus $([\varphi_2 \circ F_2\zeta f\alpha_2])_{F_2',\zeta} = ([\varphi_2])_{F_2,\zeta\times\alpha_1} \circ \left(F_2^\dagger \zeta f\right) \circ \iota_2$. If we let $g(f) \triangleq ([\varphi_2])_{F_2,\zeta\times\alpha_1} \circ \left(F_2^\dagger \zeta f\right) \circ \iota_2$ denote this morphism, we deduce $g(f) \circ in_{K,\zeta} = \varphi_2 \circ F_2\zeta f\alpha_2 \circ F_2\zeta(H\zeta)\, g(f)$ = $\varphi_2 \circ F_2\,\zeta\, f\, g(f)$ because $in_{K,\zeta} = in_{F_2',\zeta}$. □

Lemma 18 *If $f : H\zeta \to \alpha_1$ and $g : K\zeta \to \alpha_2$ are, up the isomorphism $F^\dagger\zeta = H\zeta \times K\zeta$, the two parts of catamorphism $([\Phi])_{F,\zeta} = f \times g$ then $f = \left(\left[\varphi_1 \circ F_1\zeta\alpha_1 ([\varphi_2])_{F_2,\zeta\times\alpha_1}\right]\right)_{F/F_2,\zeta}$ and $g = ([\varphi_2 \circ F_2\zeta f\alpha_2])_{F_2',\zeta}$.*

Proof. By Corollary 16 the two parts $f : H\zeta \to \alpha_1$ and $g : K\zeta \to \alpha_2$ of the catamorphism $([\Phi])_{F,\zeta} = f \times g$ are characterized by $f \circ in_{H,\zeta} = \varphi_1 \circ F\zeta fg$ and $g \circ in_{K,\zeta} = \varphi_2 \circ F_2\zeta fg$. Set $f' = \left(\left[\varphi_1 \circ F_1\zeta\alpha_1 ([\varphi_2])_{F_2,\zeta\times\alpha_1}\right]\right)_{F/F_2,\zeta}$ and $g' = g(f') = ([\varphi_2 \circ F_2\zeta f'\alpha_2])_{F_2',\zeta}$. By the preceding lemma $g' \circ in_{K,\zeta} = \varphi_2 \circ F_2\,\zeta\, f'\, g'$, moreover

$$\begin{aligned}
& f' \circ in_{H,\zeta} \\
= \;& f' \circ in_{F/F_2,\zeta} \circ F_1\zeta(H\zeta)\iota_2 \\
= \;& \varphi_1 \circ F_1\zeta\alpha_1 ([\varphi_2])_{F_2,\zeta\times\alpha_1} \circ F_1\zeta f' \left(F_2^\dagger \zeta f'\right) \circ F_1\zeta(H\zeta)\iota_2 \\
= \;& \varphi_1 \circ F_1\zeta\alpha_1 ([\varphi_2])_{F_2,\zeta\times\alpha_1} \circ F_1\zeta\alpha_1 \left(F_2^\dagger \zeta f'\right) \circ F_1\zeta f' \left(F_2^\dagger \zeta(H\zeta)\right) \circ F_1\zeta(H\zeta)\iota_2 \\
= \;& \varphi_1 \circ F_1\zeta\alpha_1 ([\varphi_2])_{F_2,\zeta\times\alpha_1} \circ F_1\zeta\alpha_1 \left(F_2^\dagger \zeta f'\right) \circ F_1\zeta\alpha_1\iota_2 \circ F_1\zeta f'(K\zeta) \\
= \;& \varphi_1 \circ F_1\zeta\alpha_1 \left(([\varphi_2])_{F_2,\zeta\times\alpha_1} \circ F_2^\dagger \zeta f' \circ \iota_2\right) \circ F_1\zeta f'(K\zeta) \\
= \;& \varphi_1 \circ F_1\zeta\alpha_1 g' \circ F_1\zeta f'(K\zeta) \\
= \;& \varphi_1 \circ F_1\zeta f' g'
\end{aligned}$$

From which it follows that $f' = f$ and $g' = g$. □

Theorem 14 follows from Lemma 17 and Lemma 18.

4 Conclusion

As mentioned in the introduction the global grammar would normally be left implicit. Our result allows to represent it as a cascaded composition of its constituent subgrammars. And this representation preserves catamorphisms (and also by duality the anamorphisms). We know that an attribute grammar provides an executable specification of a DSL and thus appears as a formalism adapted to the prototyping of such languages. We can then adopt an incremental approach consisting in growing such a

DSL by cascaded composition of modular attribute grammars using the results presented here. We intend to apply also the corresponding result on anamorphisms to the modular design of parsers, syntax-directed translators and iteractive editors.

References

[1] S. Abramsky and A. Jung. Domain Theory. Hanbook of Logic in Computer Science, vol. III, 1994.

[2] Hans Bekìc. Definable operations in general algebras, and the theory of automata and flowcharts. Lecture Notes in Computer Science vol. 177:30-55, 1984.

[3] Krzysztof Czarnecki, Ulrich W. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison Wesley, 2000.

[4] Sergey Dmitriev. Language Oriented Programming: The Next Paradigm. http://www.onboard.jetbrains.com/articles/04/10/lop/index.html

[5] M. Fokkinga, J. Jeuring, L. Meertens, E. Meijer. A translation from Attribute Grammars to Catamorphisms. *The Squiggolist*, 2(1):20-26, 1991.

[6] Martin Fowler. Language Workbenches: The Killer-App for Domain Specific Languages. http://www.martinfowler.com/articles/languageWorkbench.html

[7] A. Joyal, R. Street, D. Verity. Traced monoidal categories. *Mathematical Proceedings of the Cambridge Philosophical Society* 119 (3) pp. 447-468, 1996.

[8] G. Plotkin. Post-graduate lectures notes in advanced domain theory (incorporating the "Pisa Notes"'. Dept. of Computer Science, Univ. of Edinburg, 1981.

[9] Charles Simonyi. The Death of Computer Languages, the Birth of Intentional Programming. In B. Randell (Ed.) The future of Software, Proceeding of the joint International Computer Limited. University of Newcastle seminar (Also : Technical Report MSR-TR-95-52, Microsoft Research, redmond), 1995.

[10] A.K. Simpson, G. Plotkin. Complete axioms for categorical fixed-point operators. In. Proc. Logic in Computer Science (LICS'2000), 30-41, 2000.

[11] Eric Van Wyk, Oege de Moor, Ganesh Sittampalam, Ivan Sanabria Piretti, Kevin Backhouse, Paul Kwiatkowski. Intentional Programming: A Host of Language Features. Oxford University Computing Laboratory, PRG-RR-01-21, 2001.

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifi que
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399