New approaches in functional programming using algebras and coalgebras

Viliam Slodičák, Pavol Macko

Department of Computers and Informatics
Faculty of Electrical Engineering and Informatics
Technical University of Košice

ETAPS - Workshop on generative technologies

Saarbrücken, 27.03.2011

Basic Concepts: Category theory

- Program is defined as data structures and algorithms. In developing large scale programs we always have to apply several mathematical theories.
- The goal of programming is then to formulate a solution over these theories.
- Mathematical machines (i.e. computers) are able to make logical reasoning over representations of types.
- Categories are useful in computer science, where we often use more complex structures not expressible by sets.
- The relations between objects are expressed by morphisms.

Basic Concepts: Category theory

Category

- $Ob(\mathscr{C})$, objects of category $\mathscr{C}$, e.g. $A, B, \ldots$;
- $Morph(\mathscr{C})$, morphisms of category $\mathscr{C}$, e.g. $f : A \rightarrow B$;
- identity morphism for each object of $\mathscr{C}$, $id_A : A \rightarrow A$;
- composition of morphisms: for $f : A \rightarrow B$ and $g : B \rightarrow C$ there is $f \circ g : A \rightarrow C$.

Functor

- is a morphism between categories, $F : \mathscr{C} \rightarrow \mathscr{D}$;
- maps objects of $\mathscr{C}$ to objects of $\mathscr{D}$;
- maps morphism $C_1 \rightarrow C_2$ in $\mathscr{C}$ to morphism $FC_1 \rightarrow FC_2$ in $\mathscr{D}$.

Recursion versus corecursion

- Recursion in computer programming is exemplified when a function is defined in terms of simpler, often smaller versions of itself.
- Recursion in computer programming is exemplified when a function is defined in terms of simpler, often smaller versions of itself.
- Dual notion to recursion is corecursion.
- Corecursion can produce both finite and infinite data structures as result, and may employ self-referential data structures.

Coalgebras and algebras

Coalgebras and algebras

For the category $\mathscr{C}$ and polynomial endofunctors $F, G : \mathscr{S}et \to \mathscr{S}et$:

- G-coalgebra is a pair (U, φ) and

$$\varphi = \langle destr_1, \ldots, destr_n \rangle = U \to G(U)$$

  is coalgebraic structure providing observable properties of a program system

- F-algebra is a pair (A, α) and

$$\alpha = [cons_1, \ldots, cons_n] = F(A) \to A$$

  is algebraic structure describing internal structure of a program system.

Category of Algebras

Algebra homomorphism

$$f^* : (A, \alpha) \to (B, \beta)$$

$$\begin{tikzcd} F(A) \arrow[r, "F(f)"] \arrow[d, "\alpha"'] & F(B) \arrow[d, "\beta"] \\ A \arrow[r, "f"'] & B \end{tikzcd}$$

Category of algebras

$\mathscr{A}lg(F)$

- objects - algebras: $(A, a), (B, b), \ldots$;
- morphisms - algebra homomorphisms: $f^* : (A, a) \to (B, b)$;
- identity - for each algebra: $id_{(A,a)} : (A, a) \to (A, a)$;
- morphisms are composable.

Initial algebra

- Initial algebra is the initial object of the category $\mathscr{A}lg(F)$

$$(\mu F, in_F)$$

- algebra operation in_F is defined:

$$in_F : F(\mu F) \to F;$$

- The morphism from initial algebra into any algebra we call catamorphism: for $\alpha : F(A) \to A$ is

$$cata\ \alpha : \mu F \to A$$

Initial algebra

$(\mu F, in_F)$

It holds for initial algebra:

$$\begin{tikzcd} F\mu F \arrow[r, "in_F"] \arrow[d, dashed, "F(cata\ \alpha)"'] & \mu F \arrow[d, "cata\ \alpha"] \\ FA \arrow[r, "\alpha"'] & A \end{tikzcd}$$

$$in_F \circ cata\ \alpha = F(cata\ \alpha) \circ \alpha$$

Category of coalgebras

Coalgebra homomorphism

$$f_* : (U, \varphi) \to (V, \psi)$$

$$\begin{tikzcd} U \arrow[r, "f"] \arrow[d, "\varphi"'] & V \arrow[d, "\psi"] \\ F(U) \arrow[r, "F(f)"'] & F(V) \end{tikzcd}$$

Category of coalgebras

$\mathscr{C}oalg(F)$

- objects - coalgebras: $(U, \varphi), (V, \psi), \ldots$;
- morphisms - coalgebra homomorphisms: $f_* : (U, \varphi) \to (V, \psi)$;
- identity - for each coalgebra: $id_{(U,\varphi)} : (U, \varphi) \to (U, \varphi)$;
- morphisms are composable.

Final coalgebra

- Final coalgebra is the final object of the category $\mathscr{C}oalg(F)$

$$(\nu F, out_F)$$

- coalgebra dynamics *out* is defined:

$$out_F : F \to F(\nu F);$$

- The morphism from any coalgebra into the final coalgebra we call anamorphism:
  for $\varphi : U \to F(U)$ is

$$ana\ \alpha : U \to \nu F$$

Final coalgebra

$(\nu F, out_F)$

It holds for final coalgebra:

$$\begin{tikzcd} U \arrow[r, "\varphi"] \arrow[d, dashed, "ana\ \varphi"'] & FU \arrow[d, "F(ana\ \varphi)"] \\ \nu F \arrow[r, "out_F"'] & F\nu F \end{tikzcd}$$

$$ana\ \varphi \circ out_F = \varphi \circ F f$$

Recursive Coalgebra

Definition

A coalgebra (U, φ) is called recursive if for every algebra (A, α) there exists a unique coalgebra-to-algebra morphism $f : U \to A$

$$\begin{tikzcd} FU \arrow[d, "Ff"'] & U \arrow[l, "\varphi"'] \arrow[d, "f"] \\ FA \arrow[r, "\alpha"'] & A \end{tikzcd}$$

It holds that

$$f = \varphi \circ Ff \circ \alpha.$$

Hylomorphism

Definition

A coalgebra (U, φ) is called recursive if for every algebra (A, α) there exists a unique coalgebra-to-algebra morphism $f : U \to A$

$$\begin{tikzcd} FU \arrow[d, "Ff"'] & U \arrow[l, "\varphi"'] \arrow[d, "hylo(\varphi,\alpha)_F"] \\ FA \arrow[r, "\alpha"'] & A \end{tikzcd}$$

It holds that

$$hylo(\varphi, \alpha)_F = \varphi \circ Ff \circ \alpha.$$

Moreover, the hylomorphism is a composition of anamorphism and catamorphism

$$hylo(\varphi, \alpha)_F = (cata\ \alpha)_F \circ (ana\ \varphi)_F$$

Data Structure: Stack

Stack(σ):

$$
\begin{array}{ll}
Stack(\sigma): & \\
\quad new: & \rightarrow Stack(\sigma) \\
\quad push: & Stack(\sigma), \sigma \rightarrow Stack(\sigma) \\
\quad top: & Stack(\sigma) \rightarrow \sigma \\
\quad is\_empty: & Stack(\sigma) \rightarrow bool \\
\quad pop: & Stack(\sigma) \rightarrow Stack(\sigma)
\end{array}
$$

Polynomial endofunctor

$$F(S) = 1 + (S \times I)$$

The F-algebra (S, a), where $a = [new, push]$ is defined by

$$[new, push](w) = \begin{cases} new, & \text{if } w = (1, (s, \varepsilon)) \\ push(s, i) & \text{if } w = (2, (s, i)) \end{cases}$$

and $w \in 1 + (S \times I)$.

Constructor Operations on Stack

Constructor operations on Stack:

$$new : \to S \qquad push : S \times I \to S$$

where

- I is the representation of the type σ;
- I^*, the Kleene closure over I contains the sequences of stack values;
- S is the representation of the type $Stack(\sigma)$.

The initial algebra

$$(I^*, [new, push])$$

$$\begin{tikzcd}
1 + (I^* \times I) \arrow[r, "id + (fill \times id)"] \arrow[d, "in_F = {[new, push]}"', "\cong"] \arrow[dr, phantom, "\circledcirc"] & 1 + (S \times I) \arrow[d, "\alpha = {[new, \pi_1]}"] \\
I^* \arrow[r, dashed, "fill = cata\ \alpha"'] & S
\end{tikzcd}$$

Coalgebraic definition of constructors

Combinig the operations *pop* a *top* we construct the operation

$$next : S \to 1 + (S \times I)$$

where

- I is the representation of the type σ;
- I^*, the Kleene closure over I contains the sequences of stack values;
- S is the state space.

The final coalgebra

$$(I^*, next)$$

where

$$next : I^* \to 1 + (I^* \times I)$$
$$next(s) = \begin{cases} \kappa_1(*) & \text{if } s \text{ is empty} \\ \kappa_2(s', i) & \text{if } s = push(s', i) \end{cases}$$

Coalgebraic definition of constructors

$$\begin{tikzcd}
1 \arrow[rr, dashed, "new"] \arrow[d, "\kappa_1"'] & & I^* \arrow[d, "\cong"', "next"] \\
1 + (1 \times I) \arrow[r, phantom, ""] \arrow[rr, "id + (new \times id)"'] & \circledcirc & 1 + (I^* \times I)
\end{tikzcd}$$

$$\begin{tikzcd}
I^* \times I \arrow[rr, dashed, "ana\ \varphi = push"] \arrow[d, "\varphi = \kappa_2"'] & & I^* \arrow[d, "\cong"', "next = out_F"] \\
1 + ((I^* \times I) \times I) \arrow[rr, "id + (push \times id)"'] & \circledcirc & 1 + (I^* \times I)
\end{tikzcd}$$

Combining the algebra and coalgebra

$$\begin{tikzcd}
1 + (S \times I) \arrow[r, "{[new, \pi_1]}"] & S \\
1 + (I^* \times I) \arrow[u, "id + (fill \times id)"] \arrow[r, "{[new, push]}"] & I^* \arrow[u, "fill"'] \arrow[d, "next"'] \\
1 + ((I^* \times I) \times I) \arrow[u, "id + (push \times id)"] \arrow[r, "id + (push \times id)"'] & 1 + (I^* \times I)
\end{tikzcd}$$

Recursive coalgebra for Stack

Recursive coalgebra for Stack

$$\begin{tikzcd} I^* \arrow[r, "next"] \arrow[d, "fill"'] \arrow[dr, phantom, "\circledcirc"] & 1 + (I^* \times I) \arrow[d, "F(fill)"] \\ S & 1 + (S \times I) \arrow[l, "{[new, \pi_1]}"] \end{tikzcd}$$

It holds

$$fill = next \circ F(fill) \circ [new, \pi_1]$$

where

$$fill : I^* \to S$$

Recursive coalgebra for Stack

Recursive coalgebra for Stack

The coalgebra-to-algebra morphism

$$fill : I^* \to S$$

is defined:

$$fill(i) = \begin{cases} s & \text{if } card(i) = length(s) \\ \perp & \text{otherwise} \end{cases}$$

for $i \in I^*$, $s \in S$.

$$(I^*, next) \to (S, [push, new])$$

Implementation of the anamorphism

Anamorphism

- it represents the corecursive function

$$int \rightarrow intList$$

```
let rec ana n =
match n with
| 0 -> []
| 1 -> [1]
| n -> n :: ana (n-1);;
```

Implementation of the catamorphism

Catamorphism

- it represents the recursive function

$$intList \rightarrow int$$

```
let rec cata list =
match list with
| [] -> 1
| head :: tail -> head * (cata tail);;
```

Implementation of the catamorphism

Hylomorphism

- defined as the composition of anamorphism and catamorphism;
- it represents the function that corecursively generates the list and then it recursively treat with it
$$int \rightarrow int;$$
- the function *ana* generates the list of natural numbers from n to 1;
- the function *cata* eliminates the generated list of natural numbers;

```
let fact x =
cata (ana x);;
```

Execution of the function $fact$

```
# fact 4;;
- : int = 24
```

Implementation of the hylomorphism

Factorial by hylomorphism execution

fact 4 =

cata (ana 4) =

4 cata (ana 3) =

12 cata (ana 2) =

24 cata (ana 1) =

24 id =

24

Thank You for Your attention

Conferrence INFORMATICS'2011

Rožňava, 16th-18th Nov 2011

Topics

1. Computer Architectures
2. Computer Networks
3. Theoretical Informatics
4. Programming Paradigms, Programming Languages
5. Software Engineering
6. Distributed Systems
7. Computer Graphics and Virtual Reality
8. Artificial Intelligence
9. Knowledge Management
10. Information System Research
11. Applied Informatics and Simulation

Conferrence INFORMATICS'2011

Rožňava, 16th-18th Nov 2011

Topics

- Abstract submission: April 30th 2011
- Abstract acceptance: May 15th 2011
- Full paper submission: September 2th 2011
- Notification of acceptance: September 30th 2011
- Registration, payment, conference program: October 10th 2011
- Conference: November 16th-18th 2011

Abstracts should be sent electronically to the address: submission@kpi.fei.tuke.sk. Abstracts should be no more than 15 lines with name, affiliation of authors and topic.