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

- \( \text{Ob}(\mathcal{C}) \), objects of category \( \mathcal{C} \), e.g. \( A, B, \ldots \);
- \( \text{Morph}(\mathcal{C}) \), morphisms of category \( \mathcal{C} \), e.g. \( f : A \to B \);
- identity morphism for each object of \( \mathcal{C} \), \( \text{id}_A : A \to A \);
- composition of morphisms: for \( f : A \to B \) and \( g : B \to C \) there is \( f \circ g : A \to C \).

Functor

- is a morphism between categories, \( F : \mathcal{C} \to \mathcal{D} \);
- maps objects of \( \mathcal{C} \) to objects of \( \mathcal{D} \);
- maps morphism \( C_1 \to C_2 \) in \( \mathcal{C} \) to morphism \( FC_1 \to FC_2 \) in \( \mathcal{D} \).
Recursion versus corecursion

- 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

For the category $\mathcal{C}$ and polynomial endofunctors $F, G : \mathcal{I}et \rightarrow \mathcal{I}et$:

- $G$-coalgebra is a pair $(U, \varphi)$ and
  \[
  \varphi = \langle \text{destr}_1, \ldots, \text{destr}_n \rangle = U \rightarrow G(U)
  \]
  is coalgebraic structure providing observable properties of a program system

- $F$-algebra is a pair $(A, \alpha)$ and
  \[
  \alpha = [\text{cons}_1, \ldots, \text{cons}_n] = F(A) \rightarrow A
  \]
  is algebraic structure describing internal structure of a program system.
Category of Algebras

Algebra homomorphism

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

\[ F(A) \xrightarrow{F(f)} F(B) \]

\[ \alpha \]

\[ A \xrightarrow{f} B \]

Category of algebras

\[ \mathcal{Alg}(F) \]

- 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 $\mathcal{Alg}(F)$

$$(\mu F, \text{in}_F)$$

- Algebra operation $\text{in}_F$ is defined:

$$\text{in}_F : F(\mu F) \to F;$$

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

$$\text{cata } \alpha : \mu F \to A$$
Initial algebra

It holds for initial algebra:

\[(\mu F, \text{in}_F)\]

\[\begin{array}{c}
F\mu F \xrightarrow{\text{in}_F} \mu F \\
\downarrow \phantom{\alpha} & \phantom{\alpha} \\
F(cata \alpha) \xrightarrow{cata \alpha} \phantom{\alpha} & \phantom{\alpha} \\
\downarrow \phantom{\alpha} & \phantom{\alpha} \\
FA \xrightarrow{\alpha} A
\end{array}\]

\[\text{in}_F \circ cata \alpha = F(cata \alpha) \circ \alpha\]
Category of coalgebras

Coalgebra homomorphism

\[ f_* : (U, \varphi) \rightarrow (V, \psi) \]

Category of coalgebras

\[ \text{Coalg}(F) \]

- objects - coalgebras: \((U, \varphi), (V, \psi), \ldots\);
- morphisms - coalgebra homomorphisms: \(f_* : (U, \varphi) \rightarrow (V, \psi)\);
- identity - for each coalgebra: \(\text{id}_{(U, \varphi)} : (U, \varphi) \rightarrow (U, \varphi)\);
- morphisms are composable.
Final coalgebra

- Final coalgebra is the final object of the category $\text{Coalg}(F)$
  
  $$(\nu F, \text{out}_F)$$

- Coalgebra dynamics $\text{out}$ is defined:
  
  $$\text{out}_F : F \rightarrow F(\nu F);$$

- The morphism from any coalgebra into the final coalgebra we call anamorphism:
  
  for $\varphi : U \rightarrow F(U)$ is
  
  $$\text{ana} \ \alpha : U \rightarrow \nu F$$
Final coalgebra

$$(\nu F, \text{out}_F)$$

It holds for final coalgebra:

$$U \xrightarrow{\varphi} FU$$

$$\xrightarrow{\text{ana } \varphi} F\text{ana } \varphi$$

$$\xrightarrow{\text{out}_F} F\nu F$$

$$\text{ana } \varphi \circ \text{out}_F = \varphi \circ Ff$$
Recursive Coalgebra

**Definition**

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

\[
\begin{array}{ccc}
FU & \xrightarrow{\varphi} & U \\
\downarrow \phi & & \downarrow f \\
FA & \xrightarrow{\alpha} & A
\end{array}
\]

It holds that

\[ f = \varphi \circ Ff \circ \alpha. \]
Hylomorphism

**Definition**

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

\[
\begin{align*}
FU & \xleftarrow{\varphi} U \\
Ff & \downarrow \quad hyl\circ(\varphi, \alpha)_F \\
FA & \xrightarrow{\alpha} A
\end{align*}
\]

It holds that

\[hyl\circ(\varphi, \alpha)_F = \varphi \circ Ff \circ \alpha.\]

Moreover, the hylomorphism is a composition of anamorphism and catamorphism

\[hyl\circ(\varphi, \alpha)_F = (cata \alpha)_F \circ (ana \varphi)_F\]
### Data Structure: Stack

**Stack(\(\sigma\)):**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Invariant Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>new</code></td>
<td>Stack((\sigma))</td>
</tr>
<tr>
<td><code>push</code></td>
<td>Stack((\sigma)), (\sigma) (\rightarrow) Stack((\sigma))</td>
</tr>
<tr>
<td><code>top</code></td>
<td>Stack((\sigma)) (\rightarrow) (\sigma)</td>
</tr>
<tr>
<td><code>is_empty</code></td>
<td>Stack((\sigma)) (\rightarrow) bool</td>
</tr>
<tr>
<td><code>pop</code></td>
<td>Stack((\sigma)) (\rightarrow) Stack((\sigma))</td>
</tr>
</tbody>
</table>

#### Polynomial endofunctor

\[
F(S) = 1 + (S \times I)
\]

The \(F\)-algebra \((S, a)\), where \(a = [\text{new}, \text{push}]\) is defined by

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

and \(w \in 1 + (S \times I)\).
Constructor Operations on Stack

Constructor operations on Stack:

\[ new : \rightarrow S \quad push : S \times I \rightarrow S \]

where

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

The initial algebra

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

\[ \begin{align*}
1 + (I^* \times I) & \xrightarrow{id + (fill \times id)} 1 + (S \times I) \\
\downarrow \text{in}_F = [new, push] & \cong \quad \odot \\
I^* & \xrightarrow{\text{fill} = \text{cata } \alpha} S \\
\end{align*} \]
Coalgebraic definition of constructors

Combining the operations \textit{pop} a \textit{top} we construct the operation

\[
next : S \rightarrow 1 + (S \times I)
\]

where

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

The final coalgebra

\[ (I^*, next) \]

where

\[
next : I^* \rightarrow 1 + (I^* \times I)
\]

\[
next(s) = \begin{cases} 
\kappa_1(*) & \text{if } s \text{ is empty} \\
\kappa_2(s', i) & \text{if } s = \text{push}(s', i)
\end{cases}
\]
Coalgebraic definition of constructors

\[ 1 \xrightarrow{\new} I^* \]

\[ \kappa_1 \xrightarrow{\circ} \equiv \xrightarrow{\text{next}} \]

\[ 1 + (1 \times I) \xrightarrow{id + (\new \times id)} 1 + (I^* \times I) \]

\[ I^* \times I \xrightarrow{\text{ana } \varphi = \text{push}} I^* \]

\[ \varphi = \kappa_2 \xrightarrow{\circ} \equiv \xrightarrow{\text{next } = \text{out}_F} \]

\[ 1 + ((I^* \times I) \times I) \xrightarrow{id + (\text{push} \times id)} 1 + (I^* \times I) \]
Combining the algebra and coalgebra

\[
1 + (S \times I) \xrightarrow{\text{[new, } \pi_1]} S
\]

\[
1 + (I^* \times I) \xrightarrow{\text{[new, push]}} I^*
\]

\[
1 + ((I^* \times I) \times I) \xrightarrow{id + (push \times id)} 1 + (I^* \times I)
\]
Recursive coalgebra for Stack

It holds

\[ \text{fill} = \text{next} \circ F(\text{fill}) \circ [\text{new}, \pi_1] \]

where

\[ \text{fill} : I^* \rightarrow S \]
Recursive coalgebra for Stack

The coalgebra-to-algebra morphism

\[ fill : I^* \rightarrow S \]

is defined:

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

for \( i \in I^* \), \( s \in S \).

\[(I^*, \text{next}) \rightarrow (S, [\text{push}, \text{new}])\]
Implementation of the anamorphism

**Anamorphism**

- it represents the corecursive function

\[
int \rightarrow intList
\]

```ocaml
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

\[ \text{intList} \rightarrow \text{int} \]

```
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

\[ \text{int} \rightarrow \text{int}; \]

- the function \textit{ana} generates the list of natural numbers from \( n \) to 1;
- the function \textit{cata} eliminates the generated list of natural numbers;

\[
\text{let fact x = cata (ana x)};;
\]

Execution of the function \textit{fact}

\[
\text{# fact 4};;
\]

\[
- : \text{int} = 24
\]
Implementation of the hylomorphism

Factorial by hylomorphism execution

\[
\text{fact } 4 = \\
\text{cata } (\text{ana } 4) = \\
4 \text{ cata } (\text{ana } 3) = \\
12 \text{ cata } (\text{ana } 2) = \\
24 \text{ cata } (\text{ana } 1) = \\
24 \text{ id } = \\
24
\]
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
8. Artificial Intelligence
9. Knowledge Management
10. Information System Research
11. Applied Informatics and Simulation
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.