Towards Coalgebraic Specification

Franz Lichtenberger

Research Institute for Symbolic Computation
Johannes Kepler University, Linz

16 June 2012

Outline

Motivation

Modelling

data and **processes**

in one uniform (formal) framework.

Motivation

Modelling

data and **processes**

in one uniform (formal) framework.

Historical remarks (1)

- **Computability:** Turing machines etc.
  Important results before computers existed!

- **Processes:** modelled by
  - (various types of) automata,
  - finite/abstract state machines,
  - Petri nets,
  - (labeled) transition systems,
  - ...

Historical remarks (1)

- **Computability:** Turing machines etc.
  Important results before computers existed!

- **Processes:** modelled by
  - (various types of) automata,
  - finite/abstract state machines,
  - Petri nets,
  - (labeled) transition systems,
  - ...

Modeling data

- For basic results only the data type "natural numbers" was necessary.
- For real life applications, advanced algorithms, ... richer data structures have to be specified.
- For modeling *Abstract Data Types (ADTs)* we need:
  - Base sets (sorts of objects)
  - Basic functions and predicates
  - Constraints (axioms)
- Thesis: ADTs are (classes of) **algebras**
- Gave raise to "Algebraic Specification" (of ADTs).

Modeling data

- For basic results only the data type "natural numbers" was necessary.
- For real life applications, advanced algorithms, ... richer data structures have to be specified.
- For modeling *Abstract Data Types (ADTs)* we need:
  - Base sets (sorts of objects)
  - Basic functions and predicates
  - Constraints (axioms)
- Thesis: ADTs are (classes of) **algebras**
- Gave raise to "Algebraic Specification" (of ADTs).

Modeling data

- For basic results only the data type "natural numbers" was necessary.
- For real life applications, advanced algorithms, . . . richer data structures have to be specified.
- For modeling *Abstract Data Types (ADTs)* we need:
  - Base sets (sorts of objects)
  - Basic functions and predicates
  - Constraints (axioms)
- Thesis: ADTs are (classes of) **algebras**
- Gave raise to "Algebraic Specification" (of ADTs).

Modeling data

- For basic results only the data type "natural numbers" was necessary.
- For real life applications, advanced algorithms, . . . richer data structures have to be specified.
- For modeling *Abstract Data Types (ADTs)* we need:
  - Base sets (sorts of objects)
  - Basic functions and predicates
  - Constraints (axioms)
- Thesis: ADTs are (classes of) **algebras**
- Gave raise to "Algebraic Specification" (of ADTs).

Algebraic Specification (of ADTs)

- 1972: First paper by Sir C.A.R: Hoare
- 1974-76: ADJ-group, Guttag/Horning, Liskov/Zilles, . . .
- 1983: B. Kutzler, F. Lichtenberger:
  "Bibliography of Abstract Data Types"
  More than 500 references!
- Several AlgSpec languages developed:
  OBJ3, ASL, ACT ONE/TWO, Larch, . . .
- AlgSpec concepts used in CA-Systems: Scratchpad, Axiom, Magma, (Reduce 4), . . .
- CoFI: Common Framework for Algebraic Specification and Development, EU-Project, started 1995.
- 2003: CASL - Common Algebraic Specification Language

Semantics of ADT specifications

Concepts from **category theory** are ubiquitous

- Loose, initial, final, . . . semantics
- Free algebras, galois connexion for (equational) definability
- Free functors, natural transformations for parametrized specifications
- Pushout, pullback in parameter passing
- . . .

Categories

Definition

A category C consists of

- a class of objects,
- a set of morphisms (arrows) between each of two objects,
- a composition of morphisms which is associative,
- an identity morphism for each object.

Remark

Many notions can be defined on this "categorical level", like products and coproducts, mono-, epi-, isomorphisms, initial and final objects, pullbacks and pushouts, limits and colimits, etc. etc.

These notions come "in pairs", i.e. are dual to each other.

Categories

Definition

A category C consists of

- a class of objects,
- a set of morphisms (arrows) between each of two objects,
- a composition of morphisms which is associative,
- an identity morphism for each object.

Remark

Many notions can be defined on this "categorical level", like products and coproducts, mono-, epi-, isomorphisms, initial and final objects, pullbacks and pushouts, limits and colimits, etc. etc.

These notions come "in pairs", i.e. are dual to each other.

Initial and final objects

Definition

- An object $\mathbf{1}$ is called **final** in a category C, iff
  for every object X there
  exists a unique morphism $X \rightarrow \mathbf{1}$
- An object $\mathbf{0}$ is called **initial** in a category C, iff
  for every object X there
  exists a unique morphism $\mathbf{0} \rightarrow X$

Remark

These notions are **dual**, i.e.

- *initial* is *co-final*
- *final* is *co-initial*

Initial and final objects

Definition

- An object **1** is called **final** in a category C, iff
  for every object X there
  exists a unique morphism $X \to \mathbf{1}$
- An object **0** is called **initial** in a category C, iff
  for every object X there
  exists a unique morphism $\mathbf{0} \to X$

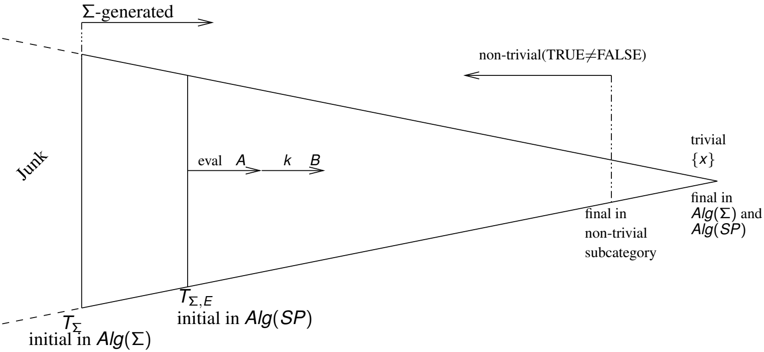
Remark

These notions are **dual**, i.e.

- *initial* is *co-final*
- *final* is *co-initial*

The structure of $Alg(\Sigma)$ and $Alg(SP)$

- The structure of $Alg(\Sigma)$ and $Alg(SP)$, $\Sigma = (S, OP)$ a signature, $SP = (\Sigma, E)$ a specification



Initial-algebra semantics: no junk, no confusion (Rod Burstall)

Functors

Remark

A *functor* is a "Homomorphism between categories".

Definition

Let C and D categories.
A **functor** $F : C \to D$ is a pair of maps, i. e. it maps

- objects of C to objects of D, and
- morphisms $f : X \to X'$ (for $X, X' \in C$) to morphisms $F(f) : F(X) \to F(X')$ in D,

such that

$$F(g \circ f) = F(g) \circ F(f)$$

and

$$F(id_X) = id_{F(X)}$$

Functors

Remark

A *functor* is a "Homomorphism between categories".

Definition

Let C and D categories.
A **functor** $F : C \to D$ is a pair of maps, i. e. it maps

- objects of C to objects of D, and
- morphisms $f : X \to X'$ (for $X, X' \in C$) to morphisms $F(f) : F(X) \to F(X')$ in D,

such that

$$F(g \circ f) = F(g) \circ F(f)$$

and

$$F(id_X) = id_{F(X)}$$

The category SET

- We work in the category SET (objects are sets, morphisms are (total) functions, usual composition of functions).
- We use the following operations on sets:
  - Product:
  $$X \times Y = \{ (x, y) \mid x \in X, y \in Y \}$$
  - Coproduct (direct sum):
  $$X + Y = \{ \langle 0, x\rangle \mid x \in X \} \cup \{ \langle 1, y\rangle \mid y \in Y \}$$
  - Powerset:
  $$\mathcal{P}(X) = \{ Y \mid Y \subseteq X \}$$
  - Function space:
  $$X^Y = \{ f \mid f : Y \to X \}$$

Remark

These operations are **functorial**, i.e., can be lifted from sets to functions between sets, thus forming functors from SET to SET.

Initial and final sets

We write $1 = \{*\}$ for a singleton set (with typical elem. $*$).
There is exactly one function $X \to 1$ for any set X.
Thus 1 is final (or terminal) in SET.

We write 0 for the empty set.
There is exactly one function $0 \to X$ for any set X.
Thus 0 is initial in SET.

Some useful isomorphisms in SET

$$
\begin{array}{cc}
X \times Y \cong Y \times X & X + Y \cong Y + X \\
1 \times X \cong X & 0 + X \cong X \\
X \times (Y \times Z) \cong (X \times Y) \times Z & X + (Y + Z) \cong (X + Y) + Z \\
X \times 0 \cong 0 & X \times (Y + Z) \cong (X \times Y) + (X \times Z)
\end{array}
$$

We shall usually work “up to” these isomorphisms, so we can simply write for n-ary products:

$$X_1 \times X_2 \times \cdots \times X_n$$

without bothering about bracketing.

Polynomial Functors (1)

Remark

We use two trivial functors as well:

1. id : SET $\to$ SET (the identity functor)
2. For a constant set C we have the functorial operation $X \mapsto C$; a function $f : X \to X'$ is mapped to the identity function $id_C : C \to C$.

We will often say things like: consider the functor

$$T(X) = X + (C \times X),$$

i.e., we give the action only on sets, here

$$X \mapsto X + (C \times X).$$

Polynomial Functors (2)

Since all operations are functorial, the action on a function $f : X \to X'$ is derived:

$$T(f) : T(X) \to T(X')$$

explicitly, $T(f)$ in our example is

$$f + (id_C \times f) : X + (C \times X) \to X' + (C \times X')$$

given by:

$$w \mapsto \begin{cases} \langle 0, f(x) \rangle & \text{if } w = \langle 0, x \rangle \\ \langle 1, (c, f(x)) \rangle & \text{if } w = \langle 1, (c, x) \rangle \end{cases}$$

In the sequel we shall use only such **polynomial** functors built up with constants, identity functor, products, coproducts, and -later- also (finite) powersets and function spaces X^A (for constant sets A).

Polynomial Functors (2)

Since all operations are functorial, the action on a function $f : X \to X'$ is derived:

$$T(f) : T(X) \to T(X')$$

explicitly, $T(f)$ in our example is

$$f + (id_C \times f) : X + (C \times X) \to X' + (C \times X')$$

given by:

$$w \mapsto \begin{cases} \langle 0, f(x) \rangle & \text{if } w = \langle 0, x \rangle \\ \langle 1, (c, f(x)) \rangle & \text{if } w = \langle 1, (c, x) \rangle \end{cases}$$

In the sequel we shall use only such **polynomial** functors built up with constants, identity functor, products, coproducts, and -later- also (finite) powersets and function spaces X^A (for constant sets A).

Algebras of a functor

Example

Let T be the polynomial functor

$$T(X) = 1 + X + (X \times X).$$

For a set U, a function $a : T(U) \to U$ is a 3-cotuple $[a_1, a_2, a_3]$ of maps
$a_1 : 1 \to U$,
$a_2 : U \to U$,
$a_3 : U \times U \to U$.
The shape of the functor T determines a **signature**, here of a group:
For a carrier set G and unit element $e : 1 \to G$, inverse $i : G \to G$, multiplication $m : G \times G \to G$,
we get by cotupelling an **algebra of** T:

$$(G, [e, i, m] : T(G) \to G).$$

Example

Similarly, the algebras of the functor

$$T(X) = 1 + X \times X$$

have a **monoid** signature.

Definition

Let T be a functor. An **algebra** of T (or T-algebra) is a pair consisting of set U and a function $a : T(U) \to U$.
We call U the **carrier set**, a the **algebra structure** or **operation** of the algebra.

Example

Natural numbers $\mathbb{N}$

zero	and	successor functions
$0 : 1 \to \mathbb{N}$		$S : \mathbb{N} \to \mathbb{N}$

form an algebra $(\mathbb{N}, [0, S] : 1 + \mathbb{N} \to \mathbb{N})$ of the functor $T(X) = 1 + X$.

Definition

Let T be a functor. An **algebra** of T (or T-algebra) is a pair consisting of set U and a function $a : T(U) \to U$.
We call U the **carrier set**, a the **algebra structure** or **operation** of the algebra.

Example

Natural numbers $\mathbb{N}$

zero	and	successor functions
$0 : 1 \to \mathbb{N}$		$S : \mathbb{N} \to \mathbb{N}$

form an algebra $(\mathbb{N}, [0, S] : 1 + \mathbb{N} \to \mathbb{N})$ of the functor $T(X) = 1 + X$.

Example

A-labelled binary trees: $Tree(A)$

operations:

$$\text{nil} : 1 \to Tree(A)$$

$$\text{node} : Tree(A) \times A \times Tree(A) \to Tree(A)$$

form an algebra of the polynomial functor

$$T(X) = 1 + (X \times A \times X)$$

isomorphic to

$$T(X) = 1 + A \times X^2$$

Remark

This polynomial (functor) is a precise and very concise specification of A-labelled binary trees. The *semantics* is the initial algebra of the category of T- algebras and homomorphisms between T-algebras.

Example

A-labelled binary trees: $Tree(A)$

operations:

$$\text{nil} : 1 \rightarrow Tree(A)$$

$$\text{node} : Tree(A) \times A \times Tree(A) \rightarrow Tree(A)$$

form an algebra of the polynomial functor

$$T(X) = 1 + (X \times A \times X)$$

isomorphic to

$$T(X) = 1 + A \times X^2$$

Remark

This polynomial (functor) is a precise and very concise specification of A-labelled binary trees. The *semantics* is the initial algebra of the category of T- algebras and homomorphisms between T-algebras.

Historical remarks (2)

Later developments in Algebraic Specification:

- Modules
- Objects and components
- Concurrency
- etc. etc. . . .
- Specification of entire software systems

DEAD END STREET!

because some of the phenomena are intrinsically non-algebraic

Historical remarks (2)

Later developments in Algebraic Specification:

- Modules
- Objects and components
- Concurrency
- etc. etc. ...

- Specification of entire software systems

DEAD END STREET!

because some of the phenomena are intrinsically non-algebraic

Historical remarks (2)

Later developments in Algebraic Specification:

- Modules
- Objects and components
- Concurrency
- etc. etc. ...

- Specification of entire software systems

DEAD END STREET!

because some of the phenomena are intrinsically non-algebraic

Historical remarks (2)

Later developments in Algebraic Specification:

- Modules
- Objects and components
- Concurrency
- etc. etc. ...

- Specification of entire software systems

DEAD END STREET!

because some of the phenomena are intrinsically non-algebraic

Coalgebras

Definition

For a functor T, a **coalgebra** (or a T-coalgebra) is a pair (U, c) consisting of a set U and a function $c : U \to T(U)$.

Like for algebras, we call U the **carrier** and c the **structure** or **operation** of (U, c). U is often called the **state space**.

Compare:

algebra: $T(U) \to U$
operation **into** the carrier U, describes **construction** of elements of U.

coalgebra: $U \to T(U)$
operation **out of** the carrier U, describes **observations** about elements of U.

Coalgebras

Definition

For a functor T, a **coalgebra** (or a T-coalgebra) is a pair (U, c) consisting of a set U and a function $c : U \to T(U)$.

Like for algebras, we call U the **carrier** and c the **structure** or **operation** of (U, c). U is often called the **state space**.

Compare:

algebra: $T(U) \to U$
operation **into** the carrier U, describes **construction** of elements of U.

coalgebra: $U \to T(U)$
operation **out of** the carrier U, describes **observations** about elements of U.

Coalgebraic phenomena (1)

Example (Black-box machine with one button and one light.)

- performs an action only if button is pressed
- light goes on only if the machine stops operating (i.e., has reached a final state)

X - the (unknown) "state space"
Describe the machine by a function:

$$\text{button} : X \to \{*\} \cup X,$$

where $* \notin X$ is a new symbol.

- The pair $(X, \text{button} : X \to \{*\} \cup X)$ is an example of a coalgebra.
- Observable behavior: an element of $\bar{\mathbb{N}} := \mathbb{N} \cup \{\infty\}$ describing the number of times the button has to be pressed until the light goes on.
- Mathematically, $\text{beh} : X \to \bar{\mathbb{N}}$, turns out to be a **final coalgebra** of the functor $T(X) = 1 + X$.

Coalgebraic phenomena (1)

Example (Black-box machine with one button and one light.)

- performs an action only if button is pressed
- light goes on only if the machine stops operating (i.e., has reached a final state)

X - the (unknown) "state space"
Describe the machine by a function:

$$\text{button} : X \to \{*\} \cup X,$$

where $* \notin X$ is a new symbol.

- The pair $(X, \text{button} : X \to \{*\} \cup X)$ is an example of a coalgebra.
- Observable behavior: an element of $\bar{\mathbb{N}} := \mathbb{N} \cup \{\infty\}$ describing the number of times the button has to be pressed until the light goes on.
- Mathematically, $\text{beh} : X \to \bar{\mathbb{N}}$, turns out to be a **final coalgebra** of the functor $T(X) = 1 + X$.

Coalgebraic phenomena (1)

Example (Black-box machine with one button and one light.)

- performs an action only if button is pressed
- light goes on only if the machine stops operating (i.e., has reached a final state)

X - the (unknown) "state space"
Describe the machine by a function:

$$\text{button} : X \to \{*\} \cup X,$$

where $* \notin X$ is a new symbol.

- The pair $(X, \text{button} : X \to \{*\} \cup X)$ is an example of a coalgebra.
- Observable behavior: an element of $\bar{\mathbb{N}} := \mathbb{N} \cup \{\infty\}$ describing the number of times the button has to be pressed until the light goes on.
- Mathematically, $\text{beh} : X \to \bar{\mathbb{N}}$, turns out to be a **final coalgebra** of the functor $T(X) = 1 + X$.

Coalgebraic phenomena (2)

Example (Similar machine with two buttons **value** and **next**)

Described by a coalgebra:

$$(X, \langle \mathrm{value}, \mathrm{next} \rangle : X \to A \times X),$$

where A is the set of observable values.

- Observable behavior: an infinite sequence

  $$(a_0, a_1, a_2, \dots) \in A^{\mathbb{N}},$$

  where a_i is the **value** after processing **next** i-times.
- Observing this behavior for every state $s \in X$ gives a function

  $$\mathrm{beh} : X \to A^{\mathbb{N}}$$

  which is the final coalgebra of $T(X) = A \times X$.

Coalgebraic phenomena (3)

Example (transition systems)

Machine with two buttons (second example)

$$\langle \text{value}, \text{next} \rangle : X \to A \times X$$

can be understood as **deterministic** transition system. We write

$$s \xrightarrow{a} s' \text{ iff } \text{value}(s) = a \text{ and } \text{next}(s) = s'.$$

The trace $\text{Tr}(s)$ of observations of state $s \in X$:

$$\text{Tr}(s) = (a_1, a_2, \dots) \text{ where } s \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \cdots,$$

which is the observable behavior $\text{beh}(s) \in A^{\mathbb{N}}$.

Example (transition systems)

A **non deterministic** transition system

$$(X, A, \rightarrow), \text{ where } \rightarrow \subseteq X \times A \times X$$

can be described in coalgebra form as a function using a powerset:

$$\alpha : X \rightarrow \mathcal{P}(A \times X),$$

where $\alpha(s)$ is the "successor set" of $s \in X$.

Remark

Finding the right domain for the observable behavior is non-trivial here.

Summing up the examples of coalgebras:

- We have a state space X about which we make no assumptions.
- On X a function is defined (often consisting of different components) which allows us
  - to observe some aspect directly or
  - move on to next states.
- We can describe just the **behavior** by making successive observations.
- This **behavior** typically is the **final coalgebra** of a (polynomial) functor.
- This also leads to the notion of **bisimilarity**, i.e., two states (which need not be equal as elements of X) cannot be distinguished via the operations at our disposal, i.e., are "equal as far as we can see."
- Bisimilarity is an important and typically coalgebraic concept.

Data and Processes

- Data are modeled by algebras
  Semantics: initial algebra
- Processes are modeled by coalgebras
  Semantics: final coalgebra

They are dual to each other:

- data are co-processes
- processes are co-data

Data and Processes

- Data are modeled by algebras
  Semantics: initial algebra
- Processes are modeled by coalgebras
  Semantics: final coalgebra

They are dual to each other:

- data are co-processes
- processes are co-data

Bialgebraic Modeling

In practice, algebraic and coalgebraic aspects interact on different hierarchical layers, for example

- start with algebraically specifying ones application domain
- describe dynamical systems (processes) as coalgebras, using the algebras above as codomains of observer functions,
- such coalgebraic systems may exist in an algebra of processes.

Remark

P. Padawitz (Dortmund) calls that "Swinging Types".

Bialgebraic Modeling

In practice, algebraic and coalgebraic aspects interact on different hierarchical layers, for example

- start with algebraically specifying ones application domain
- describe dynamical systems (processes) as coalgebras, using the algebras above as codomains of observer functions,
- such coalgebraic systems may exist in an algebra of processes.

Remark

P. Padawitz (Dortmund) calls that "Swinging Types".

Topics for discussion

From theoretical . . .

- Adjoint functors
- Monads/ strong monads/ comonads
- Kleisli categories
- Non-wellfounded sets
- equivalence in algebras vs bisimulation in coalgebras

- ’Added value’ of using coalgebras instead of, say, ASMs
- Monads and Kleisli Triples in functional programming
- Continuation Monad (and other monads)
- Final coalgebra semantics of specification languages
- Paper: "A Coalgebra as an Intrusion Detection System"
- Practical proof schemes for coinduction
- Proof by coinduction and bisimulation

. . . to practical