Chapter 1

Introduction

The purpose of this thesis is to investigate the semantics of fixpoint equations, which is used to assign a meaning to syntactic objects like computer programs, and to use these semantic insights to obtain rules for program transformations and refinement. We shall prove that these rules are complete in a certain sense.

In section 1.1 an introduction of fixpoint semantics is given. Section 1.2 contains a short description of each chapter. In section 1.3 we sum up the mathematical preliminaries and some notations. In section 1.4 we give all the definitions and results that we need of category theory and $\mathcal{F}$-(co)algebras.

1.1 Fixpoint semantics

In formal semantics one wants to assign to each element of a given set of expressions, called the syntactic domain, a meaning in a set of values, called the semantic domain. Usually, for each syntactic domain, there is a class of semantic domains, representing all the different meanings that each expression can have. Once a definition of meaning is established, a central problem that one tries to solve is: which pairs of expressions have the same meaning in each semantic domain of a given class? This is called the equivalence problem, and is mostly solved by giving a universal semantic domain for this class, that is, a semantic domain in which two expressions have the same meaning iff they have the same meaning in each semantic domain of the given class. Often, this universal semantic domain is constructed from the syntactic domain. In this thesis, syntactic and semantic domains are described with the use of (co)algebras of a given signature, and we give a rough description of three different approaches in the case where the signature only contains operators of a certain arity.

Initial algebra approach: suppose a set of laws, like associativity and commutativity are given. The syntactic domain is the initial algebra, which consists of the expressions that are generated by the operators, and where two expressions are equal iff they are constructed in the same way. The expressions of the initial algebra represent non-recursive equations. The class of semantic domains consists of algebras, which interpret the operators as functions that satisfy the laws. Initiality gives us a unique homomorphism from the syntactic domain to the
semantic domain, which is defined as the meaning. A universal semantic domain is constructed from the syntactic domain by identifying expressions that can be proved equal by using the laws. A treatment for universal algebra is first established by Birkhoff [Bir35]. This is later generalized to many-sorted algebras by Higgins [Hig63] and Birkhoff and Lipson [BL70], and a complete axiomatization for this case can be found in [MG85].

Final coalgebra approach: the syntactic domain consists of coalgebras, which can be seen as sets of (possibly recursive) equations that use the operators. There is only one semantic domain: the final coalgebra. Finality gives us a unique cohomomorphism from every coalgebra to the final coalgebra, which is defined as the meaning. Because we only have one semantic domain, the equality in the final coalgebra solves the equivalence problem, and two expressions have the same meaning in the final coalgebra iff they are related by a so-called bisimulation. This approach originated from the work of Milner [Mil89] and Hoare [Hoa85] on the semantics of concurrent programs. A treatment for categorical algebras is given by Rutten [RT93] and Aczel [Acz93].

Program scheme approach: the syntactic domain consists of so-called program schemes, which are certain tree-structures that use expressions of the initial algebra. As in the case of the final coalgebra approach, these program schemes describe recursive equations. The class of semantic domains consists of algebras equipped with a complete order such that the interpretations of the operators (which are functions) fulfill a certain continuity condition. It is easy to formulate a fixpoint condition that one expects to hold for the meaning, and the least fixpoint is defined as the meaning. A universal semantic domain consists of infinite trees, representing symbolic computations. Laws can be incorporated in the same way as in the initial algebra approach. This approach originated from denotational semantics, introduced by Scott [SS71]. A treatment for algebras can be found in [ANN85, Gue85], and for many-sorted algebras in [Cou90, Gal81, WBT85].

We use elements of the program scheme and coalgebra approach, and we now explain and motivate our approach. This thesis started as an investigation of the semantics of imperative sequential programs, especially programs that are defined by (recursive) procedure declarations. We started in the context of universal algebra.

A procedure declaration can be represented by a program scheme, and in the first instance, the elements of the syntactic domain were (a special kind of) program schemes. The monotonic predicate transformers form a semantic domain, and as in the program scheme approach, the meaning of a program is defined as the least fixpoint of a certain function (see [Hes92]). However, in contrary to the program scheme approach, the interpretation of the sequential operator in the semantic domain is not continuous. We want to include this domain in the theory, and our class of semantic domains consists of ordered algebras such that the interpretations of the operators are monotonic functions. We proved the associated equivalence problem (a bit reformulated: see below), in which an order on program schemes occurred, which appeared to be similar to bisimulation. After studying some general results about bisimulation, especially the final coalgebra connection, we concluded that program schemes (forming the syntactic domain) should be replaced by coalgebras.

This change in the theory initiated a generalization from the context of universal algebra to \(F\)-(co)algebras in the category of sets. The syntactic domain consists of \(F\)-coalgebras. The class of semantic domains consists of ordered \(F\)-algebras that satisfy a monotonicity condition, which is formulated in terms of bisimulation. The least fixpoint definition of
meaning easily generalizes, and is also noted by Meijer [Mei92] and Fokkinga [Fok92] where meaning is called hylomorphism. We want to keep the class of semantic domains as large as possible, and we do not demand that the meaning always exists. It turned out that one half of the equivalence result exactly corresponds to the construction of the final coalgebra, and from the generality of this construction one half of the equivalence result was generalized. To our surprise, the other half of the equivalence result also could be generalized.

**Example 1.1.1.** For a specific signature, we give two elements of the syntactic domain that have equal meanings in every semantic domain. We leave out some technical conditions on semantic domains.

A semantic domain consists of an ordered set $A$ together with two functions $f, g : A^2 \to A$, which are monotonic in both arguments. We consider the following two pairs of mutually recursive equations.

\[
\begin{align*}
  x &= f(x, y) \\
  y &= g(x, y)
\end{align*}
\]

and

\[
\begin{align*}
  x &= f(x, g(x, y)) \\
  y &= g(f(x, y), y)
\end{align*}
\]

For both pairs of equations, the form corresponds to a coalgebra, which is an element of the syntactic domain. We represent these coalgebras pictorially: see figure 1.1. Meaning assigns to each coalgebra a value in a semantic domain, and in these two cases meaning is the least solution in variable $x$ of the corresponding equations. We can prove that both equations have equal least solutions in $x$, that is, the two coalgebras have equal meanings in every semantic domain. The equivalence result states that two coalgebras have equal meanings in every semantic domain iff the coalgebras are related by bisimulation. We will see that this is indeed the case for the coalgebras of figure 1.1.

*(End of example)*
We will see that for a special kind of program schemes, the semantics is included in our approach. The final coalgebra and initial algebra semantics are both included. Because we do not formally treat initial algebra semantics, a proof of this inclusion is not given. However, by using the result, proved by de Bruin [Bru95], that the initial algebra is contained in the final coalgebra, this easily follows.

We reformulate the equivalence problem in the following way. In the semantics of programming languages, one is not only interested in equality of the meaning of two programs, but also in the ordering of the meanings. For example, in the case of imperative sequential programs, the refinement relation on programs is an order on the meanings of two programs, which states that “one program implements the other”. Because the elements of the semantic domain are ordered algebras, we can consider the statement: the meaning of one program is less than or equal to the meaning of the other. Unfortunately, the refinement relation itself does not support a least fixpoint definition of the semantics. The equivalence problem now becomes: which pairs of expressions are such that in every semantic domain the meaning of the first expression is less than or equal to the meaning of the second? This problem is called the ordering problem, and is solved by giving a universal semantic domain. It is well-known that the final coalgebra is also an algebra, and we construct a universal semantic domain that is similar to this algebra. The fixpoint condition of meaning in this case instantiates exactly to the cohomomorphism property of meaning, and with finality we get a unique fixpoint, which defines meaning.

We restrict our attention to \( F \)-(co)algebras in the category of sets. With more general definitions of (co)algebras, one can add more structure to the syntactic and semantic domain. For example, in our case the elements of a domain are all of the same sort, which is not the case for many-sorted algebras. However, even with this restriction the theory gets complicated enough.

We mentioned that a coalgebra can be seen as a set of equations. We introduce a specific functor \( Q \), such that the \( Q \)-coalgebras describe equations that contain \( n \)-ary operators and a choice operator. We show that the \( Q \)-coalgebras cover the equations that correspond to context-free grammars, finite state machines, and procedure declarations of an imperative sequential programming language. Moreover, we also show that the corresponding semantics of generated strings, accepted strings, and weakest (liberal) precondition are instances of the least fixpoint semantics. The functor \( Q \) is more closely investigated, especially the associated ordering problem with respect to classes of semantic domains that satisfy extra properties like laws and continuity.

One of the main results in this thesis is the general ordering result (corollary 6.3.1), which establishes a connection between bisimulation and least fixpoint semantics. In words, this result reads: bisimulation exactly determines all the “transformations” of equations (coalgebras) that are respected by every least fixpoint semantics. From this it follows that if we are dealing with a least fixpoint semantics, we do not have to consider every coalgebra, but only representatives of bisimulation equivalence classes. In [Mil84], Milner treats \( Q \)-coalgebras that only contain some unary and constant operators. He gives a simple language for bisimulation equivalence classes of these specific \( Q \)-coalgebras, together with a complete deduction system for determining bisimulation. The generalization of this to arbitrary \( Q \)-coalgebras (containing \( n \)-ary operators) is also a main result (theorem 8.2.4, theorem 8.2.7, and corollary 8.2.3). Together with the ordering result, it follows that if we are dealing with
a least fixpoint semantics then an arbitrary \( Q \)-coalgebra (equation) can be represented by an expression of this simple language. For example, a language for procedure declarations emerges. Moreover, the deduction system describes which manipulations of equations always can be applied (without changing the semantics).

The ordering problem with respect to classes of semantics domains that satisfy laws is not solved. However, we show (in theorem 8.2.10) that by adding rules for laws to the deduction system for bisimulation, we get a sound deduction system for the meaning of coalgebra equations.

### 1.2 Overview

We now give a short description of the following chapters. In chapter 2 we introduce relators, which we use both on the syntactical and semantical side of the theory. In this quite technical chapter we prove all the results that we need in the sequel about relators.

In chapter 3 we introduce schemes, formulae, and simulation. Schemes are the elements of the syntactic domain, and formulae are pairs of schemes and represent the “less than or equal” statement with which the ordering problem deals. Simulation is a preorder on schemes, and is defined as the union of all relations that satisfy a certain simulation property.

Semantic domains are introduced in chapter 4, and are called models. We define the fixpoint semantics of a scheme in a model, and the truth value of a formula in a model. We fix the class of models with which the ordering problem is formulated.

In chapter 5 we prove that if two schemes are related by the simulation relation then the meaning of the first scheme is less than or equal to the meaning of the second (equivalently: the formula that corresponds to the two schemes is true) in each model. This proves one implication of the equivalence in the ordering problem, and this implication is called soundness.

The reverse of the soundness implication is called completeness, and is proved in chapter 6. This is done by constructing a universal model for the class of chapter 4. The elements of the universal model are simulation-equivalence classes of schemes, where the schemes are elements of a so-called universe. Universes are defined as sets that satisfy certain closure properties, and are needed to ensure that the elements of the universal model together form a proper set.

The previous chapters all dealt with an (almost) arbitrary functor on the category of sets, and the rest of the chapters are devoted to one specific functor \( Q \). Examples of schemes of this signature are context-free grammars, finite state machines, and procedure declarations. In chapter 7 we restrict the class of considered models to so-called regular models. Regular models are defined as models that have a separate interpretation of choice and operators, where the interpretation of choice is required to be idempotent and associative. We prove that the universal model is a regular model.

In chapter 8 we give a simple language for schemes (of signature \( Q \)) and give complete deduction systems for two kinds of simulation. One of these simulations is bisimulation as defined originally in [Par81], and its deduction system is a generalization of the one given in [Mil84]. We also give a sound deduction system for classes of regular models for which laws hold.
In chapter 9 we give two well-known regular models for imperative sequential programs, and show that they are equivalent in the sense that they have the same set of true formulae.

A continuous model is a special kind of regular model, and is defined in chapter 10. We give the universal model for the class of continuous models that respect the laws. This universal model is a generalization of the model of languages (sets of strings) in which a context-free grammar is interpreted.

1.3 Prerequisites and notations

This thesis is mainly self-contained, only some knowledge of basic set-theory, cardinal number calculus, and predicate calculus is required.

Because we reserve the symbol “=” for a different use, we use “≡” to denote the universal equality.

For a relation $R \subseteq A \times B$ we often write $aRb$ instead of $(a, b) \in R$. For sets $A$, $B$, and $C$, relations $R \subseteq A \times B$ and $S \subseteq B \times C$ we have the following definitions.

\[
\begin{align*}
R \circ S & \equiv \{(a, c) \in A \times C \mid \exists b \in B (aRb \land bSc)\} \quad \text{(composition)} \\
R^- & \equiv \{(b, a) \in B \times A \mid aRb\} \quad \text{(transposition)} \\
1_A & \equiv \{(a, a) \mid a \in A\} \quad \text{(identity)}
\end{align*}
\]

A preorder is a reflexive and transitive relation, an order is an anti-symmetric preorder, and an equivalence is a symmetric preorder. We use the symbol “≤” for preorders, and denote the transposed relation “≤” by “≥”. In chapter 2 we will see that some care has to be taken in using this notation.

Let a preordered set $(A, \leq)$ be given. The relation $\leq \cap \geq$ is an equivalence on $A$, and we denote this equivalence by “≡”. The set of “≡”-equivalence classes is denoted by $A/\equiv$. We denote the elements of $A/\equiv$ by representants (elements of $A$). The order on $A/\equiv$ that is induced by a preorder $\leq$ on $A$ is denoted by the same symbol “≤”. Furthermore, if we have a function with domain $A$ that respects the equivalence “≡”, then the induced function on equivalence classes is denoted by the same symbol. We call an ordered set complete iff for every subset a supremum (and hence an infimum) exists. For a preordered set we cannot uniquely characterize the supremum of a subset, it can only be characterized up to equivalence (induced by the preorder). An element of this equivalence class is said to have the supremum property for that subset.

Function application is denoted by means of a dot: for a function $f : A \rightarrow B$, and $a \in A$ we have $f(a) \in B$. This operator binds from left to right, to allow currying. We sometimes define a function by means of a pair of simple expressions and the symbol “⇒”, where the first expression ranges over the domain. For example, the function $(a, b) \mapsto a$, where $a$ and $b$ range over sets $A$ and $B$, respectively, defines the projection from $A \times B$ to $A$. So for a function $f : A \rightarrow B$ we have $f \equiv a \mapsto f(a)$. Function composition is denoted by the infix operator “◦”: for functions $f : A \rightarrow B$ and $g : B \rightarrow C$ we have $g \circ f : A \rightarrow C$ defined by $g \circ f \equiv a \mapsto g(f(a))$. We let typing decide whether function composition or relation composition is meant. The identity function on a set $A$ is denoted by $id_A$. We use the notation $f[A] \equiv \{f(a) \mid a \in A\}$. The set of functions from $A$ to $B$ is denoted by $B^A$, so $f : A \rightarrow B \iff f \in B^A$. 


For a relation \( R \subseteq A \times B \) and a set \( C \), we denote the relation lifted to \( A^C \times B^C \) by the same symbol as the relation: for \( f : C \to A \) and \( g : C \to B \) we have \( f R g \Leftrightarrow \forall c \in C ((f(c)) R (g(c))) \). In case \( A \equiv B \) and \( R \) is an order, we get the lifted order on the set of functions \( C \to A \). If the ordered set \((A, \leq)\) is complete then, for any set \( B \), the ordered set \((B^A, \leq)\) is also complete (with respect to the lifted order). A function \( f \) is defined to be isotonic from \((A, \leq)\) to \((B, \leq)\) (two preordered sets) iff, for every \( a, a' \in A \), we have \( a \leq a' \Rightarrow f(a) \leq f(a') \).

We need some cardinal calculus (see [TZ71]). The class of cardinal numbers is denoted by \( \text{Card} \), and the smallest infinite cardinal number by \( \omega \). The cardinality of a set \( A \) is denoted by \( \#A \). The sets of natural numbers and real numbers are denoted by \( \mathbb{N} \) and \( \mathbb{R} \), respectively. Let \( n \in \mathbb{N}, A \equiv \{a_i \mid 1 \leq i \leq n\} \) with \( i \mapsto a_i \) injective, and let \( B \) be a set. The function from \( A \) to \( B \) that assigns \( b_i \in B \) to \( a_i \) for \( i \leq n \), can be defined by the notation \((a_1, \cdots, a_n) \mapsto (b_1, \cdots, b_n)\). For example, \((1, 4, 2) \mapsto (1, 1, 0)\) defines the function from \( \{1, 2, 4\} \) to \( \mathbb{N} \) that assigns 1 to 1, 0 to 2, and 1 to 4.

Let \( A \) be a set and \((f.A)_{a \in A}\) a family of sets. The disjoint sum and Cartesian product of this family are defined as follows.

- \( \sum_{a \in A}(f.A) \equiv \{(a, b) \mid a \in A \land b \in f.A\} \)
- \( \prod_{a \in A}(f.A) \equiv \{g : A \to \bigcup \{f.A \mid a \in A\} \mid \forall a \ (g.a \in f.A)\} \)

For each \( x \in A \) we have the injection \( \sigma_x : f.x \to \sum_{a \in A}(f.A) \) and the projection \( \pi_x : \prod_{a \in A}(f.A) \to f.x \), which are defined by \( \sigma_x \equiv b \mapsto (x, b) \) and \( \pi_x \equiv g \mapsto gx \). Let \((g.A)_{a \in A}\) be a family of sets and \((r.A)_{a \in A}\) a family of functions such that, for \( a \in A \), we have \( r.a : f.A \to g.A \), then we denote the product of this family of functions by \( \prod_{a \in A}(r.A) : \prod_{a \in A}(f.A) \to \prod_{a \in A}(g.A) \). If \( B \) is a set and \((r.A)_{a \in A}\) a family of functions with \( r.a : B \to f.A \) for \( a \in A \), then we denote the unique function from \( B \) to \( \prod_{a \in A}(f.A) \) by \( \langle r.A \rangle_{a \in A} \).

For the powerset notation we do not use the usual notation \( \mathcal{P} \), because we use this for an almost identical notion. Instead we use the following notation. For a set \( A \) we have \( \text{Pow}(A) \equiv \{B \mid B \subseteq A\} \).

In the sequel, all the knowledge stated in this chapter is referred to as “calculus”.

## 1.4 Categories and algebras

We only need the basic concepts of category theory [ML71] and \( F \)-algebra theory [BW90]. We summarize everything we need from these theories, together with some concrete categories, functors, and (co)algebras.

A category consists of a class of objects, which are abstractions of sets, and a class of arrows, which are abstractions of functions. Furthermore, a category has a domain and codomain assignment (from arrows to objects), identity arrows, and an associative composition of arrows.

**Definition 1.4.1** A category \( \mathbf{A} \) consists of the following.

- A class of objects \( \text{obj}(\mathbf{A}) \) and a class of arrows \( \text{arr}(\mathbf{A}) \).
A domain function and a codomain function from \( \text{arr}(A) \) to \( \text{obj}(A) \). If to \( f \in \text{arr}(A) \) domain \( A \) and codomain \( B \) are assigned then we denote this by \( f : A \to B \) in \( A \).

For each \( A \in \text{obj}(A) \) an arrow \( \text{id}_A : A \to A \).

For each \( f : A \to B \) and \( g : B \to C \) an arrow \( g \circ f : A \to C \).

Furthermore, the following properties hold.

- For \( f : A \to B \) we have \( \text{id}_B \circ f \equiv f \circ \text{id}_A \equiv f \).
- For \( f : A \to B, g : B \to C, \) and \( h : C \to D \) we have \( h \circ (g \circ f) \equiv (h \circ g) \circ f \).

Many mathematical structures can be described as categories. We only use the category of sets and the category of preordered sets.

- The category \( \text{Set} \), with sets as objects and (total) functions as arrows.
- The category \( \text{Prs} \), with preordered sets as objects and monotonic functions as arrows.

A functor between two categories assigns objects to objects and arrows to arrows, in such a way that it preserves composition and the identity.

**Definition 1.4.2** Let \( A \) and \( B \) be categories. A functor \( F : A \to B \) consists of a function from \( \text{obj}(A) \) to \( \text{obj}(B) \) and a function from \( \text{arr}(A) \) to \( \text{arr}(B) \) such that the following holds.

- If \( f : A \to B \) in \( A \) then \( F.f : F.A \to F.B \) in \( B \).
- \( \forall f \in \text{arr}(A) \quad F.(\text{id}_A) \equiv \text{id}_{F.(A)} \).
- \( \forall f \in \text{arr}(A) \quad F.(g \circ f) \equiv (F.g) \circ (F.f) \).

From the above definition it easily follows that the composition of two functors is again a functor. A functor from a category to itself is called an endofunctor (on that category).

In category \( \text{Set} \), an arrow (function) \( f \) is represented by a relation \( \{(a, f.a) \mid a \in A\} \), together with a domain and codomain. For example, for a set \( A \), the identity arrow \( \text{id}_A \) is represented by the relation \( 1_A \), domain \( A \), and codomain \( A \). If \( A \) is a subset of \( B \) then the inclusion function from \( A \) to \( B \) is also represented by the relation \( 1_A \), but in general this function is different from \( \text{id}_A \) (unless \( A \equiv B \)). In this thesis, all statements about functions depend only on the representing relations (not on the domain or codomain), and we want to identify functions that are represented by the same relation. For this reason we have to demand that the functors that occur in this thesis, respect this identification. We formalize this as follows.

For sets \( A, B \) with \( A \subseteq B \), let \( i_{A,B} \) denote the inclusion from \( A \) to \( B \). In the sequel we implicitly assume that, for any introduced endofunctor \( F \) on \( \text{Set} \), the following holds. For every set \( A \) and \( B \) such that \( A \subseteq B \) we have

\[
F.(i_{A,B}) \equiv i_{F.(A),F.(B)}.
\]

As an immediate consequence, we see that \( A \subseteq B \) implies \( F.A \subseteq F.B \). We now prove that, under the above condition for a functor \( F \), two functions represented by equal relations are mapped by \( F \) to functions, which again are represented by equal relations. Let \( f : A \to B \)
and \( g : A' \to C \) be such that \( \{(a, f, a) \mid a \in A\} \) equals \( \{(a, g, a) \mid a \in A'\} \). Then obviously \( A \equiv A' \), and if we put \( D \equiv B \cup C \) then we get \( i_{B,D} \circ f \equiv f \equiv i_{C,D} \circ g \). If we apply \( F \) to both sides then we get \( F(i_{B,D} \circ f) \equiv F(f) \equiv F(i_{C,D} \circ g) \), and because \( F \) is a functor, the above condition on \( F \) yields \( i_{(F,B),(F,D)} \circ (Ff) \equiv i_{(F,C),(F,D)} \circ (Fg) \). This implies that the functions \( Ff \) and \( Fg \) have equal relations. So we can identify functions with equal relations. For example, for sets \( A, B \) with \( A \subseteq B \), we write for the inclusion function from \( A \to B \) simply \( i_A \).

For the rest of this thesis we fix three constants: A set \( \text{Op} \), a function \( \eta : \text{Op} \to \text{Card} \), and an infinite cardinal \( \rho \). We call \( \text{Op} \) the set of operator names, \( \eta \) the arity assignment, and \( \rho \) the cardinality bound. These names are explained later on. The cardinal number \( \omega \) is the smallest valid cardinality bound. From cardinal calculus it is known that, for an infinite set \( A \), the Cartesian product \( A \times A \) has the same cardinality as the set \( A \). Because \( \rho \) is an infinite cardinal, we have \( \rho^2 = \rho \). This is actually the only property of \( \rho \) that is needed, and we assume it to be known in the sequel.

We give three endofunctors on the category \( \text{Set} \), which are dependent on the constants \( \text{Op}, \eta, \) and \( \rho \). It is easily checked that the above condition on endofunctors on \( \text{Set} \) is satisfied by these three functors. First we define the endofunctor \( O \) on \( \text{Set} \), which depends on the pair \( (\text{Op}, \eta) \).

**Definition 1.4.3** Define \( O : \text{Set} \to \text{Set} \) by the following.

- \( O.A \equiv \sum_{p \in \text{Op}} A^{(\eta, p)} \) for a set \( A \).
- \( O.f \equiv (p, f) \mapsto (p, f \circ p I) \) for sets \( A, B \) and \( f : A \to B \).

We now define the endofunctor \( P \) on \( \text{Set} \), which is dependent on the cardinality bound \( \rho \). As the name already indicates, the number \( \rho \) bounds the cardinality of the subsets of a given set. We also define the endofunctor \( Q \) on \( \text{Set} \) as the composition of \( O \) and \( P \), and we see that \( Q \) depends on all three constants.

**Definition 1.4.4**

1. Define \( P : \text{Set} \to \text{Set} \) by the following.
   - \( P.A \equiv \{B \subseteq A \mid \#B < \rho\} \) for a set \( A \).
   - \( P.f \equiv U \mapsto f[U] \) for sets \( A, B \) and \( f : A \to B \).

2. Define \( Q : \text{Set} \to \text{Set} \) as \( Q \equiv P \circ O \).

In case \( \rho \equiv \omega \), we see that \( P.A \) only contains the finite subsets of set \( A \). Note that for \( P \) to be a functor, we have to check that, for a set \( U \) with cardinality less than \( \rho \), the cardinality of the set \( P.f.U \) is less than \( \rho \). This follows easily from the definition.

For a functor \( F \) we define \( F \)-algebras and \( F \)-coalgebras. These two definitions are so-called dual to each other (a concept of category theory). Duality refers to the reversing of arrows in a category, which results in the opposite category. With this opposite category defined, the definitions of \( F \)-algebra and \( F \)-coalgebra are in fact two instances of one definition. We give the two definitions explicitly, but the duality is immediately spotted.

**Definition 1.4.5** Let \( A \) be a category and \( F \) an endofunctor on \( A \).
1. An $F$-algebra in $A$ is a pair $(A, \alpha)$ with $A \in \text{obj}(A)$ and $\alpha : F.A \to A$ in $A$.

2. An $F$-coalgebra in $A$ is a pair $(A, \alpha)$ with $A \in \text{obj}(A)$ and $\alpha : A \to F.A$ in $A$.

Usually we omit the words “in $A$” if it is clear which category is meant. We call $F$ the signature of an $F$-(co)algebra.

We illustrate the view that we have of algebras and coalgebras by giving some concrete examples for both cases.

**Example 1.4.1.**

1. The traditional $\Sigma$-algebras, which are sets with operations of finite arity (see [Bir35]) are $O$-algebras. In fact, if $(A, \alpha)$ is an $O$-algebra then $\alpha$ is a function from $O.A$ to $A$, and can be seen as a family of functions over $O.p$: for $p \in O.p$ we have $\alpha.p : A^{\eta.p} \to A$. Therefore, the elements $p \in O.p$ can be regarded as operator names with arities $\eta.p$. In this way, an $O$-algebra interprets the operator names as actual operators on a set, where the arities of the operators are determined by the function $\eta$.

2. If $(A, \alpha)$ is a $P$-algebra then $\alpha$ is a function from $P.A$ to $A$, that is, $\alpha$ assigns to a subset (of bounded cardinality) of $A$ an element of $A$. For example, if $A \equiv N$ and $\rho \equiv \omega$ then taking the maximum, minimum, sum, and product are examples of such functions.

*(End of example)*

For the following example we assume the formal definitions of grammars and finite state machines to be known (see [HU79]). We show that productions (of a certain form) of grammars, and transition functions together with a set of final states of machines, are both coalgebras. The variables that we use in the next example for the components of a coalgebra, are the ones that we use in the next chapters. This is done because the example is frequently referred to in the sequel, and we prefer to have the same variables.

**Example 1.4.2.** Let $T$ be a finite set.

1. Assume $O.p \equiv \{se\} \cup T$ (“se” stands for sequencing), $\eta.se \equiv 2$, $\eta.t \equiv 0$ for $t \in T$, and $\rho \equiv \omega$. The elements of the set $T$ are regarded as terminal symbols. Let $(D, \phi)$ be a $Q$-coalgebra with $D$ finite. The elements of $D$ are regarded as non-terminals. From definition 1.4.3 it follows that $O.D$ is finite, and so $Q.D \equiv P.(O.D) \equiv Pow(O.D)$ because $\rho \equiv \omega$. The function $\phi : D \to Q.D$ assigns to each element of $D$ a set of elements of $O.D$, which are of the form $(se, (d, d'))$ or $(t, ())$ with $d, d' \in D$ and $t \in T$. Thus, the coalgebra $(D, \phi)$ corresponds to a context-free grammar of a restricted form, where the set of productions $P$ is defined by

- $d \to d'd' \in P \iff (se, (d, d')) \in \phi.d$ for $d, d', d'' \in D$.
- $d \to t \iff (t, ()) \in \phi.d$ for $d \in D$ and $t \in T$.

The above form of (the productions of) the grammar is called the Chomsky normal form (see [HU79]). It is known that every context-free grammar can be rewritten (without changing the semantics) to a grammar of the Chomsky normal form. We see that the grammars that correspond to coalgebras cover all context-free grammars.
2. Assume $\mathcal{O}p \equiv \{\text{st}\} \cup T$ (“st” stands for stop), $\eta \cdot \text{st} \equiv 0, \eta \cdot t \equiv 1$ for $t \in T$, and $\rho \equiv \omega$. Let $(D, \phi)$ be a $Q$-coalgebra with $D$ finite. As in part 1, we have $\mathcal{Q}.D \equiv \text{Pow}(\mathcal{O}.D)$. The function $\phi : D \rightarrow \mathcal{Q}.D$ assigns to each element of $D$ a set of elements of $\mathcal{O}.D$, which are of the form $(\text{st},())$ or $(t,(d))$ for $t \in T$ and $d \in D$. With some calculus it follows that $\phi$ corresponds to a pair $(D_0, \delta)$, with the set $D_0 \subseteq D$ and the function $\delta : D \times T \rightarrow \text{Pow}(D)$ defined by

- $d \in D_0 \iff (\text{st},()) \in \phi \cdot d$ for $d \in D$.
- $d' \in \delta(d,t) \iff (t,(d')) \in \phi \cdot d$ for $d, d' \in D$ and $t \in T$.

If we regard $T$ as the set of input symbols and $D$ as the set of states then $\phi$ corresponds to a set of final states $D_0$ together with a transition function $\delta$. So in this case, the coalgebra $(D, \phi)$ corresponds to a non-deterministic finite state machine.

(End of example)

From the above two examples we see that an algebra can be regarded as an object that describes an interpretation of “composed objects over a set” as elements of that set. Dually, we see that a coalgebra describes a way to break up an element of a set into components. Summarizing, we can regard an algebra as an interpreting object, and a coalgebra as a defining object. In chapter 3 and chapter 4 this view will become more concrete.

Let two endofunctors on a category be given. From an algebra arrow for one functor and an algebra arrow for the other functor, both on the same object, one can form an algebra arrow of the composition of the two functors.

**Theorem 1.4.1** Let $A$ be a category and $F,G$ two endofunctors on $A$. If $(A, \alpha)$ is an $F$-algebra and $(A, \beta)$ is a $G$-algebra then $(A, \beta \circ (G \alpha))$ is a $(G \circ F)$-algebra.

Proof. Indeed, $\beta \circ (G \alpha) : G.(F.A) \rightarrow A$. \qed

The functor $Q$ is the composition of the functors $\mathcal{O}$ and $\mathcal{P}$, and from the above theorem we see that from an $\mathcal{O}$-algebra and a $\mathcal{P}$-algebra we can form a $Q$-algebra. We define an abbreviation for this situation.

**Definition 1.4.6** Let $(A, \alpha)$ be an $\mathcal{O}$-algebra in $\text{Set}$ and $(A, \beta)$ a $\mathcal{P}$-algebra in $\text{Set}$. Define $[\alpha, \beta] \equiv \beta \circ (\mathcal{P} \alpha) : \mathcal{Q}.A \rightarrow A$.

So from theorem 1.4.1 and definition 1.4.4.2 we see that, given an $\mathcal{O}$-algebra $(A, \alpha)$ and a $\mathcal{P}$-algebra $(A, \beta)$, the pair $(A, [\alpha, \beta])$ is a $Q$-algebra. The following question arises: can every $Q$-algebra be formed from an $\mathcal{O}$-algebra and a $\mathcal{P}$-algebra? The following example shows that this is not the case.

**Example** Assume $\mathcal{O}p \equiv \{\text{se}\}$, $\eta \equiv (\text{se}) \rightarrow (2)$, and $\rho \equiv \omega$. Let $A$ be a set. We prove that there exists a $Q$-algebra $(A, \gamma)$ such that there do not exist an $\mathcal{O}$-algebra $(A, \alpha)$ and a $\mathcal{P}$-algebra $(A, \beta)$ with $\gamma \equiv [\alpha, \beta]$. This is done by counting the possibilities for $\alpha$, $\beta$, and $\gamma$. We assume that $A$ is finite, and because $\rho \equiv \omega$, we have $\mathcal{P}.A \equiv \text{Pow}(A)$ (see definition 1.4.4.1). The functions $\alpha$, $\beta$, and $\gamma$ are arbitrary elements of $A^2 \rightarrow A$, $\text{Pow}(A) \rightarrow A$, and $\text{Pow}(A^2) \rightarrow A$, respectively. Put $n \equiv \#A$. With some calculus we get that $\#(A^2 \rightarrow A) \equiv n^{(n^2)}, \#(\text{Pow}(A) \rightarrow A) \equiv n^{(2^n)}$, and $\#(\text{Pow}(A^2) \rightarrow A) \equiv n^{(2(n^2))}$. If
we take \( n_A \equiv 2 \) then there are \( 2^4 \) possibilities for \( \alpha \), \( 2^4 \) for \( \beta \), and \( 2^{16} \) for \( \gamma \). So there are maximally \( 2^8 \) different combinations of \( \alpha \) and \( \beta \). We see that less than one percent of the \( \mathbb{Q} \)-algebras can be formed from an \( O \)-algebra and a \( P \)-algebra.  

(End of example)

In a (co)algebra the arrow determines some structure, and an arrow that preserves this structure is called a (co)hom, which is an abbreviation of (co)homomorphism. Again, the definitions of hom and cohom are each others dual.

**Definition 1.4.7** Let \( A \) be a category and \( F \) an endofunctor on \( A \).

1. \( f \) is an \( F \)-hom from \( F \)-algebra \((A, \alpha)\) to \( F \)-algebra \((B, \beta)\) iff \( f : A \to B \) in \( A \) and \( \beta \circ (F.f) \equiv f \circ \alpha : FA \to B \).

2. \( f \) is an \( F \)-cohom from \( F \)-coalgebra \((A, \alpha)\) to \( F \)-coalgebra \((B, \beta)\) iff \( f : A \to B \) in \( A \) and \( (F.f) \circ \alpha \equiv \beta \circ f : A \to FB \).

If we state that a certain function is a hom from an object to another object, then we implicitly assume that the domain and codomain objects are algebras. It is easily checked that the composition of two \( F \)-(co)homs is again an \( F \)-(co)hom. We show that an \( O \)-hom is exactly the familiar algebra homomorphism.

**Example** Let \((A, \alpha), (B, \beta)\) be \( O \)-algebras and \( f : A \to B \). From the above definition, definition 1.4.3, and some calculus we see that \( f \) is an \( O \)-hom iff

\[
\forall p \in Op (\beta.(p \circ I) \equiv f.(\alpha.(p \circ I))).
\]

(End of example)

If an arrow is a hom with respect to two endofunctors then it is also a hom with respect to the composition of the two functors.

**Theorem 1.4.2** Let \( A \) be a category and \( F, F' \) two endofunctors on \( A \). If \( f \) is an \( F \)-hom from \((A, \alpha)\) to \((B, \beta)\) and an \( F' \)-hom from \((A, \alpha')\) to \((B, \beta')\) then \( f \) is an \((F' \circ F)\)-hom from \((A, \alpha' \circ (F' \circ \alpha))\) to \((B, \beta' \circ (F' \circ \beta))\).

**Proof.**

\[
(\beta' \circ (F' \circ \beta)) \circ ((F' \circ F).f)
\]

\[
= \{ F' \text{ is a functor, calculus } \}
\]

\[
\beta' \circ (F'.(\beta \circ (F.f)))
\]

\[
= \{ f \text{ is an } F \text{-hom } \}
\]

\[
\beta' \circ (F'.(f \circ \alpha))
\]

\[
= \{ F' \text{ is a functor, } f \text{ is an } F' \text{-hom } \}
\]

\[
f \circ (\alpha' \circ (F' \circ \alpha))
\]

\[ \square \]

The above theorem immediately implies the following.
Example 1.4.5. Let \( f \) be an \( \mathcal{O} \)-hom from \((A, \alpha)\) to \((B, \beta)\) and a \( \mathcal{P} \)-hom from \((A, \alpha')\) to \((B, \beta')\). Then \( f \) is a \( \mathcal{Q} \)-hom from \((A, [\alpha, \alpha'])\) to \((B, [\beta, \beta'])\).

Proof. This follows directly from theorem 1.4.2 with \( F := \mathcal{O} \) and \( F' := \mathcal{P} \), and definition 1.4.6.

(End of example)

In the category \( \text{Set} \) we need, next to the concept of a hom, also the weaker concept of a semi-hom between two algebras. For defining this, the codomain object has to be equipped with a preorder.

Definition 1.4.8. Let \( F \) be an endofunctor on \( \text{Set} \), \((A, \alpha), (B, \beta)\) two \( F \)-algebras, and \( \leq_B \) a preorder on \( B \). Then \( f : A \rightarrow B \) is an \( F \)-semi-hom from \((A, \alpha)\) to \((B, \leq_B, \beta)\) iff \( \beta \circ (F \cdot f) \leq_B f \circ \alpha \).

As in the case of homs, if we state that \( f \) is a semi-hom from an object to an object then we implicitly assume that the domain object is an algebra and the codomain object an algebra equipped with a preorder. It is easily checked that a hom from an \((A, \alpha)\) to \((B, \beta)\) is a semi-hom from \((A, \alpha)\) to \((B, 1_B, \beta)\).

To allow composition of semi-homs (and for the sake of symmetry), we also could have required the domain object to have a preorder. If to the definition of a semi-hom we then add the condition that the semi-hom is a monotonic function, the composition of two semi-homs is again a semi-hom. This follows from theorem 1.4.3 below. Because compositions of semi-homs rarely occur in this thesis and the above definition has other advantages, we prefer the given definition.

Theorem 1.4.3. Let \( F \) be an endofunctor on \( \text{Set} \), let \( f \) be an \( F \)-semi-hom from \((A, \alpha)\) to \((B, \leq_B, \beta)\), and \( g \) an \( F \)-semi-hom from \((B, \beta)\) to \((C, \leq_C, \gamma)\). If \( g \) is monotonic from \((B, \leq_B)\) to \((C, \leq_C)\) then \( g \circ f \) is an \( F \)-semi-hom from \((A, \alpha)\) to \((C, \leq_C, \gamma)\).

Proof.

\[
\begin{align*}
\gamma \circ (F \circ (g \circ f)) & \leq \{ F \text{ is a functor, } g \text{ is an } F \text{-semi-hom} \} \\
& \leq \{ f \text{ is an } F \text{-semi-hom, } g \text{ is monotonic} \} \\
& \leq (g \circ f) \circ \alpha
\end{align*}
\]

We need the analogy of theorem 1.4.2 for semi-homs. However, from the proof of theorem 1.4.2 it follows that some kind of monotony of \( \beta' \) and \( F \) is needed. For formulating these monotony conditions, we need the theory of the next chapter, and the analogy of theorem 1.4.2 for semi-homs is postponed.