Dialgebraic Logics
Extended Abstract

Horst Reichel
Institute: Theoretical Computer Science
Dresden University of Technology
D-01062 Dresden, Germany

Abstract
We extend the notion of a first-order signature in such a way that the type constructors used to define domain and codomain of the fundamental operations are taken to be a constituent part of the signature. Using the generative power of the type constructors and the fundamental types and operations we obtain a general construction of a category of typed terms which will be called syntactic category. Functors into the category of set from a syntactic category preserving the used type constructors represent models and terms with the constant type $\text{Bool}$ as codomain represent properties.

We demonstrate that in this way propositional logic and modal logic can be generated by a uniform constructions which differ only in the used type constructors of the corresponding syntactic categories. This observation leads to the conjecture that logics can be classified by the type constructors used on the syntactic level.

1 Introduction
In the formal foundations of systems specification a great diversity of formalisms is in use. Tools and approaches that stem from algebraic specifications of abstract data types are presented at length in [AKKB98]. Another approach is based on hidden algebras [GM96] which extends the algebraic specification of abstract data types by the concept of observability, first introduced in [GGM76]. In [JR97] coalgebras are suggested as suitable for the specification of state-based systems. There is a long tradition to use transition systems [Mil89] and modal and temporal logics [Gol92] for systems specifications. In [Hag87] dialgebras are introduced to unify algebraic and coalgebraic structures in order to deal in a uniform manner with data types and non-terminating processes.

By this list of references, which is far from being complete, we only want to illustrate the claim that a growing variety of structures and formalisms is used to deal with systems. To overcome this diversity we are looking for a

©1998 Published by Elsevier Science B. V.
general construction of a suitable logic for a given type of structures. Thus the aim of the paper is comparable with that of [Mos97], where Moss suggests the derivation of a modal logic from the type of coalgebras.

The construction of a suitable logic for a class of structures is based on a generalization of the notion of a signature or similarity type of algebraic structures in such a way that the types necessary for the definitions of the domains and codomains of the fundamental operations are considered as constituent part of the signature.

In case of first-order logics there is no need to do so, since always the finite product type and the constant type Bool of truth values is used. Therefore a signature does only say what the arity of a fundamental operation is, i.e. defining the domain, and if the fundamental operation is a function or a predicate. In the first case the codomain is one of the basic types and in the second case it is the type Bool.

In case of coalgebras the codomain of a fundamental (co-)operation is a composed type which is not only a sum of basic types. In case of dialgebras both the domain and the codomain of a fundamental operation may be a composed type. Many different classes of structures may be classified by the types used for the definitions of the domains and codomains of the fundamental operations.

The class of image finite transition systems may be classified by a next-state operation

$$\textup{next} : \textup{In} \times \textup{St} \to \textup{Out} \times \textup{Pfin}(\textup{St}),$$

where $\textup{Pfin}(\textup{St})$ denotes the parameterized data type of finite subsets of $\textup{St}$. The finite set of possible successor states depends on a set of input values and the next-state operations produces an output value of basic type $\textup{Out}$ and the set of possible successor states. This fundamental operation needs beside the atomic types $\textup{In}, \textup{St}, \textup{Out}$ the product type and the parameterized data type $\textup{Pfin}(\cdot)$, and if one wants to deal with properties of states, the data type $\textup{Bool}$ has also to be taken into account. Thus, a corresponding extended signature of image finite transition systems is given as follows:

**type constructors**: $\textup{Bool} : \textup{types},$

$\textup{unit} : \textup{types}$

$\_ \times \_ : \textup{types}, \textup{types} \to \textup{types},$

$\textup{Pfin} : \textup{types} \to \textup{types};$

**basic types**: $\textup{In}, \textup{St}, \textup{Out};$

**basic operations**: $\textup{next} : \textup{In} \times \textup{St} \to \textup{Out} \times \textup{Pfin}(\textup{St}),$

$p : \textup{St} \to \textup{Bool};$

$\_ \to \_ \to \_.$

2
This signature does not define the semantics of the type constructors. It is assumed that the semantics of the type constructors is defined before they are used in an extended signature.

2 Strong Categorical Type Constructors

We assume that the type constructors are defined within the framework of strong categorical datatypes as introduced by Cockett and Spencer, see [CS 92] and [CS 95].

In the following we will only illustrate this approach by the type constructors used in following sections.

unit denotes the terminal object in a category and \( !_C : C \to \text{unit} \) denotes the unique morphism induced by unit and the object \( C \).

\( A \times B \) denotes the cartesian product or the projective limit of the diagram consisting of the two object \( A \) and \( B \). The projections will be denoted by \( p_A^{AB} : A \times B \to A \) and \( p_B^{AB} : A \times B \to B \) and the unique morphism, induced by pairing of \( f : C \to A \) and \( g : C \to A \) is denoted by \( \langle f, g \rangle : C \to A \times B \).

\( \text{Bool} \) denotes a strong bool-object in a category with finite products, i.e., it is characterized by the existence of two morphisms

\[
\text{true}, \text{false} : \text{unit} \to \text{Bool}
\]

such that for each object \( A \) and any morphisms

\[
f, g : A \times \text{unit} \to C
\]

there is exactly one morphism

\[
\text{fold}_{\text{Bool}}(f, g) : A \times \text{Bool} \to C
\]

such that

\[
(id_A \times \text{true}); \text{fold}_{\text{Bool}}(f, g) = f
\]

and

\[
(id_A \times \text{false}); \text{fold}_{\text{Bool}}(f, g) = g,
\]

where \( h; g \) denotes the composition of morphisms in diagrammatic order.

Dually to the product we will also use the sum \( A + B \), where \( \text{in}_A^{AB} : A \to A + B \) und \( \text{in}_B^{AB} : B \to A + B \) denote the injections and \( [f, g] : A + B \to C \) denotes the unique morphism induced by \( f : A \to C \) and \( g : B \to C \).

The finite power set constructor \( P_{\text{fin}}(B) \) is characterized by the existence of the two constructors

\[
\text{empty} : \text{unit} \to P_{\text{fin}}(B), \text{add} : B \times P_{\text{fin}}(B) \to P_{\text{fin}}(B)
\]

such that

\[
\text{add}(x, \text{add}(x, s)) = \text{add}(x, s)
\]
and

\[ add(x, add(y, s)) = add(y, add(x, s)) \]

for \( x, y : unit \rightarrow B \) and \( s : unit \rightarrow P_{fin}(B) \). In the equations representing the idempocy and commutativity of the constructor \( add \) we used a more traditional functional notation. The corresponding universal property of \( P_{fin}(B) \) implies for any two morphisms

\[ f : A \times unit \rightarrow C, \ g : A \times B \times C \rightarrow C \]

which satisfy

\[ g(x, y, g(x, y, z)) = g(x, y, z) \]
\[ g(x, y_1, g(x, y_2, z)) = g(x, y_2, g(x, y_1, z)) \]

the existence of a unique morphism

\[ fold_{P_{fin}}(f, g) : A \times P_{fin}(B) \rightarrow C \]

such that

\[ (fold_{P_{fin}}(f, g))(x, empty) = f(x) \]
\[ (fold_{P_{fin}}(f, g))(x, add(y, s)) = g(x, y, (fold_{P_{fin}}(f, g))(x, s)) \]

where \( x : unit \rightarrow A, y : unit \rightarrow B \) and \( s : unit \rightarrow P_{fin}(B) \).

The finite power set constructor is an example of a type constructor which cannot be expressed in Charity, see [CF 92], since defining equations for the constructors are necessary. Without the defining equations the type constructor defines finite lists and not finite subsets (in the category of sets).

An alternative way to define these type constructions within categories is the representation by sketches or by means of equationally partial operations, see [Rei 87]. Thus, small categories with finite products, with terminal object, with a strong bool–object, and with finitary power set objects form a sketchable category. By lack of room we are not able to present the details of such an equational formalization of type constructors. But one important consequence which we will use in the following is the fact, that there are categories with systems of type constructors which are freely generated by sets of objects and morphisms. Algebraic theories in the sense of Lawvere [Law63] are categories with unit and product freely generated by an ordinary signature of algebraic structures.

3 Syntactic Categories

Having sketched our understanding of type constructors we can introduce the basic notions.
Definition 3.1 An extended signature is a triple

$$\mathcal{E}X\_SIG = (\text{TYPES}, \text{B\_TYPES}, \text{B\_OP})$$

consisting of a finite set $\text{TYPES}$ of type constructors, a finite set $\text{B\_TYPES}$ of type names (names of basic types), and a finite set $\text{B\_OP}$ of typed fundamental operations such that domain type and codomain type of each fundamental operation is a composed type built up of basis types in $\text{B\_TYPES}$ and type constructors in $\text{TYPES}$.

Definition 3.2 The syntactic category $\mathcal{C}_{\mathcal{E}X\_SIG}$ is the category freely generated by $\text{B\_TYPES}$, $\text{B\_OP}$ in the class of small categories closed under the type constructors of $\text{TYPES}$.

An $\mathcal{E}X\_SIG$-structure is a functor

$$M : \mathcal{C}_{\mathcal{E}X\_SIG} \to \text{Set}$$

which preserves the type constructors.

In order to illustrate the introduced notions, let $\mathcal{E}X\_SIG_1$ be the extended signature with $\text{TYPES}_1 = \{\times, \text{Bool}, \text{unit}\}$, $\text{B\_TYPES}_1 = \emptyset$ and $\text{B\_OP}_1 = \{a_1, \ldots, a_n\}$, i.e., there are no basic types and each fundamental operation is a constant of type $\text{Bool}$. The objects of $\mathcal{C}_{\mathcal{E}X\_SIG}$ are the finite products of $\text{Bool}$. What are the morphisms? First, one can easily construct the usual operations on truth values like $\text{neg} : \text{Bool} \to \text{Bool}$, and $\text{imp} : \text{Bool} \times \text{Bool} \to \text{Bool}$ or $\text{imp} : \text{Bool} \times \text{Bool} \to \text{Bool}$ by the $\text{fold}_{\text{Bool}}$-construction. Thus,

$$\text{neg} = \text{fold}_{\text{Bool}}(\text{false}, \text{true}) : \text{Bool} \to \text{Bool},$$

$$\text{and} = \text{fold}_{\text{Bool}}(p_{\text{Bool}}^{\text{unit}}, (p_{\text{unit}}^{\text{Bool}}, \text{false})) : \text{Bool} \times \text{Bool} \to \text{Bool}$$

$$\text{imp} = \text{fold}_{\text{Bool}}((p_{\text{unit}}^{\text{Bool}}, \text{true}), (p_{\text{unit}}^{\text{Bool}}, \text{false})) : \text{Bool} \times \text{Bool} \to \text{Bool}.$$
We will show in the following section, that the same approach can be applied to dialgebras, provided the corresponding type constructors are taken into account.

4 The syntactic category of image finite transition systems - Modal Logics

As described in the Introduction, image finite transition systems may be seen as models of extended signatures containing the finitary power set constructor.

Which additional morphisms are generated by the corresponding universal property of the new type constructor?

Let $E X_{SIG_{ML}}$ be an extended signatur such that

$$TYPES_{ML} = \{\times, \text{unit}, \text{Bool}, P_{\text{fin}}(\_\)\},$$

$In, St, Out \in B_{TYPES_{ML}}$, $(next : In \times St \rightarrow Out \times P_{\text{fin}}(St)) \in B_{OP_{ML}}$ and $(p : St \rightarrow \text{Bool}) \in B_{OP_{ML}}$. There may be more basic operations depending on the specific application area. We assume that there are enough basic operations given such that there is at least one term $t : \text{unit} \rightarrow In$ in the corresponding syntactic category.

By means of the $fold_{P_{\text{fin}}}$-construction for any property $p : St \rightarrow \text{Bool}$ of states one can construct in the syntactic category a morphism

$$all_p : P_{\text{fin}}(St) \rightarrow \text{Bool}$$

which, interpreted in the category of sets, associates to a finite subset of states the truth value $true$ if and only if all elements of the given finite subset have the property $p$.

For the construction of the morphism $all_p$ one can choose $A = \text{unit}$, $f = true : \text{unit} \rightarrow \text{Bool}$ and $g = (p \times \text{id}_{\text{Bool}}) ; \text{and} : St \times \text{Bool} \rightarrow \text{Bool}$ and obtains

$$all_p = fold_{P_{\text{fin}}}(true, ((p \times \text{id}_{\text{Bool}}) ; \text{and})).$$

To apply the $fold_{P_{\text{fin}}}$-construction in that way one has to check that the instantiation of the parameter $g$ satisfies the required condition of idempoicy and commutativity. But, this is a direct consequence of the corresponding properties of and : $\text{Bool} \times \text{Bool} \rightarrow \text{Bool}$.

For each term $t : \text{unit} \rightarrow In$ one gets a morphism $next_t : St \rightarrow P_{\text{fin}}$ by $next_t = ((!_{St} ; t), \text{id}_{St}) ; next$. The composition

$$next_t ; all_p : St \rightarrow \text{Bool}$$

represents an abstract $\text{Bool}$-valued observation on states such that, again interpreted in the category $Set$,

$$s ; next_t ; all_p = true$$
Reichel

for \( s : \text{unit} \to \text{St} \), if all successor states of \( s \), i.e., all states \( s' \) in \( s; \text{next}_t : \text{unit} \to P_{\text{fin}}(\text{St}) \) have the property \( p : \text{St} \to \text{Bool} \). This means

\[
\text{next}_t; \text{all}_p \text{ represents the multi-modal formula } [t]p.
\]

Analogously to \( \text{all}_p \) one can construct in the syntactic category a morphism

\[
\text{ex}_p : P_{\text{fin}}(\text{St}) \to \text{Bool}
\]

by

\[
\text{ex}_p = \text{fold}_{P_{\text{fin}}} (\text{false}, ((p \times \text{id}_{\text{Bool}}); \text{or}))
\]

and gets a morphism

\[
\text{next}_t; \text{ex}_p \text{ which represents the multi-modal formula } \langle t \rangle p.
\]

Using derived properties of states instead of \( p \) one can for instance construct a morphism which represents the multi-modal formula \([t_1](t_2)p\) by

\[
\text{next}_t; \text{all}_{\text{next}_t; \text{ex}_p}.
\]

In a multi-modal logic only one of the both formulae \([t]p, \langle t \rangle p\) has to be introduced, since one of them can be expressed by the other one using negation, for instance by \( \langle t \rangle p = \neg([t] \neg p)\).

Do

\[
\text{next}_t; \text{all}_p; \text{neg}; \text{neg} : \text{St} \to \text{Bool}
\]

and

\[
\text{next}_t; \text{ex}_p : \text{St} \to \text{Bool}
\]

represent in the syntactic category different morphisms? It is not hard to see that the uniqueness condition of the universal property of \( \text{fold}_{P_{\text{fin}}} \) implies the equality

\[
\text{all}_p; \text{neg}; \text{neg} = \text{ex}_p
\]

for each \( p : \text{St} \to \text{Bool} \).

This observation shows that in the syntactic category some tautological equivalences can be proved. It is a problem that has to be investigated, if all tautological equivalences of multi-modal logics are reflected by equalities in the syntactic category, or if there are tautological equivalences that are induced by properties of the category \( \text{Set} \) being the domain of interpretations of multi-modal models.

The preceding discussion proves that that the universal properties of the used type constructors are powerful enough to generate all the usual formulae of multi-modal logics.

Are there morphisms in the syntactic category that do represent properties of states not expressible in multi-modal logics? The author guesses that the answer is NO. But, a proof has still to be given.
Finally we discuss a different type of processes which may be seen as deterministic partial automata with a parameterless state transition function. These kind of automata can be described as models of the extended signature $E \Sigma G_{Aut}$ with $T Y P E S_{Aut} = \{ \times, \text{unit}, \text{Bool}, + \}$, $B_{OP_{Aut}} = \{ \text{St}, \text{Out} \}$ and $B_{OP} = \{ \text{next} : \text{St} \to (\text{Out} \times \text{St}) + \text{unit}, \ldots \}$.

A state $s : \text{unit} \to \text{St}$ is a terminal state if
\[ s; \text{next} = !\text{St}; \text{in}_{\text{unit}}^{B \times \text{St}, \text{unit}}, \]
otherwise there exists an output value $x : \text{unit} \to \text{Out}$ and a successor state $s' : \text{unit} \to \text{St}$ with
\[ s; \text{next} = \langle x, s' \rangle; \text{in}_{\text{Out} \times \text{St}, \text{unit}}^{\text{Out} \times \text{St}, \text{unit}}. \]

Using the sum structure of the codomain we can define the state predicate
\[ \text{stop} = \text{next}; [!\text{Out} \times \text{St}; \text{false}, \text{true}] : \text{St} \to \text{Bool} \]
which characterizes the one-step behavior, i.e., for $s : \text{unit} \to \text{St}$ the equation
\[ s; \text{stop} = \text{true} \]
holds if and only if the is no successor state for $s : \text{unit} \to \text{St}$.

Is it possible to construct other state predicates? Yes, one can express for each natural number whether the process stops (not later than) after $n$ steps. This can be done as follows. First we derive from the state transition function $\text{next} : \text{St} \to (\text{Out} \times \text{St}) + \text{unit}$ the morphism
\[ \text{next}^n = [(p_{\text{Out}, \text{St}}^{\text{Out} \times \text{St}}, \text{next}), \text{id}_{\text{unit}}] : (\text{Out} \times \text{St}) + \text{unit} \to (\text{Out} \times \text{St}) + \text{unit}. \]
and obtain the required state predicate by
\[ \underbrace{\text{next}; \text{next}^*; \ldots; \text{next}^*}_{n\text{-times}}; \text{stop} : \text{St} \to \text{Bool}. \]

The given extended signature does not allow to construct a predicate which distinguishes output elements. Therefore we can only check how many steps a process can proceed and cannot check what the result is, i.e., which output value has been produced. The observability of output values can be achieved by adding predicates to the extended signature with the basic type $\text{Out}$ as domain.

5 Discussion

In the preceding sections we presented the observation that type constructors can be used to classify logics. The classifying types of a logic can be derived from the domains and codomains of the basic operations used. Till now we do not know who general this observation is. Is it for instance possible to find
classifying type constructors for the $\mu$-calculus or for CTL, temporal logics that allow to formulate safety and security properties, not expressible in the modal logic discussed above?

References

[AKKB98] E. Astesiano, H.-J. Kreowski, and B. Krieg-Brückner. *Algebraic Foundations of Systems Specification*. Springer, 1998.

[CS 92] Cockett, J.R.B. and Spencer, D.: Strong categorical data types I. In R.A.G. Seely (Editor) International Meeting on Category Theory 1991, Canadian Mathematical Society Proceedings, AMS, Montreal, 1992.

[CS 95] Cockett, J.R.B. and Spencer, D.: Strong categorical data types II : A term logic for categorical programming. Theoretical computer science, 139, 69–113, 1995.

[CF 92] Cockett, J.R.B. and Fukushima, T.: About Charity. Department of Computer Science, University of Calgary, Technical Report 92/480/18, 1992.

[GGM76] V. Giarrantana, F. Gimona, and U. Montanari. Observability concepts in abstract data specifications. In A. Mazurkiewicz, editor, *Mathematical Foundations of Computer Science*, number 45 in Lecture Notes in Computer Science, pages 576–587. Springer, 1976.

[GM96] J.A. Goguen and G. Malcolm. An extended abstract of a hidden agenda. In J. Meystel, A. Meystel, and R. Quintero, editors, *Proceedings of the Conference on Intelligent Systems: A Semiotic Perspective*, pages 159–167. Nat. Inst. Stand. & Techn., 1996.

[Gol92] R. Goldblatt. *Logics of Time and Computation, Second Edition*. Number 7 in CSLI Lecture Notes. Center for the Study of Language and Information, 1992.

[Hag87] T. Hagino. *A categorical programming language*. PhD thesis, Edinburgh University, 1987.

[JR97] Bart Jacobs and Jan Rutten. A tutorial on (co)algebras and (co)induction. *Bulletin of the EATCS*, (62):222–259, June 1997.

[Law63] F.W. Lawvere. *Functorial Semantics of Algebraic Theories*. Proc. National Academy of Science 50, 1963, pp.869-872.

[Law70] F.W. Lawvere. *Quantifiers and sheaves*. In: Actes du Congrès International des Mathématiciens. Tome 1, pp. 329-334.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[Mos97] Lawrence Moss. Coalgebraic logic. unpublished, 1997.

[Rei 87] Reichel, H,: Initial computability, algebraic specifications, and partial algebras, Clarendon Press, Oxford, 1987.