G.10 Abstract Data Types

by Hans-Dieter Ehrich in Braunschweig, Germany

In computing, a data type is given by a domain of data values and the operations applicable to them. In mathematical terms, this is an algebra. Since data operations are often many-sorted, a data type is a many-sorted (or heterogeneous) algebra as introduced by Birkhoff and Lipson (1970).

An abstract data type is a data type where some details are considered as irrelevant, like internal memory representation. In mathematical terms, this is a class of algebras. Differences between isomorphic data types are always considered irrelevant, so an abstract data type is a class of algebras that is closed under isomorphism. It is called monomorphic if it consists of just one isomorphism class, otherwise it is called polymorphic.

A problem extensively studied in an algebraic setting is abstract data type specification. We concentrate on equational specification using

\[ \text{E.g., the interval } [-l, k] \text{ of integers } (k, l \in \mathbb{N}_0) \text{ of a type with } \Omega_0 := \{0\} \text{ and } \Omega_1 := \{p, s\} \text{ is the free } \mathcal{K} \text{-algebra on } \emptyset, \text{ when } \mathcal{K} \text{ is defined by the ECE-equations } p^0(0) \equiv p'(0), s^k(0) \equiv s^k(0), p(x) \equiv p(x) \Rightarrow s(p(x)) \equiv x \text{ and } s(x) \equiv s(x) \Rightarrow p(s(x)) \equiv x. \]
G.10 Abstract Data Types

initiality, but briefly mention other approaches at the end. We largely follow the presentation in Loeckx et al. (1996, 2000).

A signature \( \Sigma = (S, \Omega) \) is given by sets \( S \) of sorts and \( \Omega \subseteq N \times S^* \times S \) of operation symbols; \( N \) is a set of operation names, and \( S^* \) denotes the set of finite sequences over \( S \) (including the empty sequence). A triple \( (f; s_1, \ldots, s_n, s_0) \in \Omega \) is often written as \( f : s_1 \times \cdots \times s_n \rightarrow s_0 \).

Sorts are names of data domains, for example bool for that of boolean values, nat for natural numbers and natlist for lists of natural numbers. Examples of operation names are \( \lor : \text{bool} \times \text{bool} \rightarrow \text{bool} \) and \( \leq : \text{nat} \times \text{nat} \rightarrow \text{bool} \). An example of a signature \( \Sigma_{\text{bool}} \) is given by the set \( S_{\text{bool}} = \{ \text{bool} \} \) of sorts and the set \( \Omega_{\text{bool}} = \{ \text{false} \rightarrow \text{bool}, \lor : \text{bool} \times \text{bool} \rightarrow \text{bool} \} \) of operation symbols.

A \( \Sigma \)-algebra assigns a set \( A(s) \) to each sort \( s \in S \), and a function \( A(\omega) : A(s_1) \times \cdots \times A(s_n) \rightarrow A(s_0) \) to each operation symbol \( \omega = (f : s_1 \times \cdots \times s_n \rightarrow s_0) \in \Omega \). For example, the usual \( \Sigma_{\text{bool}} \)-algebra \( \text{Bool} \) is given by \( \text{Bool}(\text{bool}) = \{ \text{true}, \text{false} \} \) and \( \text{Bool}(\lor : \text{bool} \times \text{bool} \rightarrow \text{bool}) \) is logical disjunction. Given a signature \( \Sigma \), the class of all \( \Sigma \)-algebras is denoted by \( \text{Alg}(\Sigma) \).

A \( \Sigma \)-algebra morphism \( h : A \rightarrow B \) is a family of maps \( h = (h_s : A(s) \rightarrow B(s))_{s \in S} \) such that, for any operation symbol \( \omega = (f : s_1 \times \cdots \times s_n \rightarrow s_0) \in \Omega \), we have \( h_{s_0}(A(\omega)(a_1, \ldots, a_n)) = B(\omega)(h_{s_1}(a_1), \ldots, h_{s_n}(a_n)) \), for all elements \( a_1 \in A(s_1), \ldots, a_n \in A(s_n) \).

Good candidates for monomorphic abstract data types are isomorphism classes of initial algebras in certain subclasses of \( \text{Alg}(\Sigma) \).

G.10.1 Definition Let \( \mathcal{C} \subseteq \text{Alg}(\Sigma) \) be a class of \( \Sigma \)-algebras. \( A \in \mathcal{C} \) is called initial in \( \mathcal{C} \) iff there is exactly one \( \Sigma \)-algebra morphism \( h : A \rightarrow B \) from \( A \) to any other algebra \( B \) in \( \mathcal{C} \) (cf. H.2.1).

The definition of many-sorted terms is a straightforward generalization of the classical one, cf. Definition G.1.4. \( T_\Sigma(X) = (T_\Sigma(X),s)_{s \in S} \) denotes the \( S \)-indexed set family of terms over \( \Sigma \) and variables \( X = (X_s)_{s \in S} \). For the ground terms over \( \Sigma \) (i.e., \( X = \varnothing = \{ \emptyset_s \}_{s \in S} \)), we write \( T_\Sigma = \{ T_\Sigma, s \}_{s \in S} \). \( T_\Sigma(X) \) can be made a \( \Sigma \)-algebra \( T(\Sigma(X)) \) by defining the operations as term constructors: \( T(\Sigma(X))(\omega)(t_1, \ldots, t_n) = f(t_1, \ldots, t_n) \) for all operation symbols \( \omega = (f : s_1 \times \cdots \times s_n \rightarrow s_0) \in \Omega \) and all terms \( t_1 \in T_{\Sigma(X),s_1}, \ldots, t_n \in T_{\Sigma(X),s_n} \).

For \( A \in \text{Alg}(\Sigma) \), any variable assignment \( \alpha : X \rightarrow A \) can be uniquely extended to a \( \Sigma \)-algebra morphism \( A(\alpha) : T(\Sigma(X)) \rightarrow A \), defining term evaluation. \( T(\Sigma(X)) \) is a free \( \Sigma \)-algebra over \( X \), cf. Section G.4. If \( X = \varnothing \), then there is just one variable assignment \( \emptyset : \emptyset \rightarrow A \) to every
\( \Sigma \)-algebra \( A \). Its unique extension \( A(\varnothing) : T(\Sigma) \to A \) is the one and only morphism from \( T(\Sigma) \) to \( A \). Thus, \( T(\Sigma) \) is initial in \( \text{Alg}(\Sigma) \). We write \( A(t) \) for \( A(\varnothing)(t) \).

Given a class \( \mathcal{C} \subseteq \text{Alg}(\Sigma) \), the **congruence relation** of \( \mathcal{C} \), denoted by \( \equiv_c \), is defined as \( (\equiv_c, s)_{s \in S} \) where \( \equiv_c, s = \{ (t, u) | t, u \in T_{\Sigma, s} \text{ and } A(t) = A(u) \text{ for each } A \in \mathcal{C} \} \). \( T(\Sigma, \mathcal{C}) = T(\Sigma)/\equiv_c \) is called the **quotient term algebra** of the class \( \mathcal{C} \). \( [t]_c \) denotes the congruence class of \( t \) in \( T(\Sigma, \mathcal{C}) \).

**G.10.2 Theorem** Let \( \Sigma \) be a signature and \( \mathcal{C} \subseteq \text{Alg}(\Sigma) \). There is a unique \( \Sigma \)-algebra morphism \( h : T(\Sigma, \mathcal{C}) \to A \) to each algebra \( A \in \mathcal{C} \), namely \( h([t]_c) = A(t) \) for each ground term \( t \in T_\Sigma \).

**G.10.3 Corollary** \( T(\Sigma, \mathcal{C}) \) is initial in \( \mathcal{C} \) iff \( T(\Sigma, \mathcal{C}) \in \mathcal{C} \).

The equational logic over a signature \( \Sigma \) is given by the set \( \mathcal{E}L(\Sigma) \) of \( \Sigma \)-equations of the form \( \forall X.t = u \) where \( X \) is a set of variables for \( \Sigma \) and \( t, u \in T_{\Sigma(\mathcal{X}), s} \) for some sort \( s \) of \( \Sigma \). An equation \( \varphi \in \mathcal{E}L(\Sigma) \) is **satisfied in a \( \Sigma \)-algebra** \( A \in \text{Alg}(\Sigma) \), denoted by \( A \models_\Sigma \varphi \), iff \( A(\alpha)(t) = A(\alpha)(u) \) for every assignment \( \alpha : X \to A \).

If \( \Phi \subseteq \mathcal{E}L(\Sigma) \), \( A \models_\Sigma \Phi \) is defined to hold iff \( A \models_\Sigma \varphi \) for every \( \varphi \in \Phi \).

In this case, we say that \( A \) is a **model** of \( \Phi \). \( \text{Mod}_\Sigma(\Phi) \) denotes the class of all models of \( \Phi \) in \( \text{Alg}(\Sigma) \).

For any set \( \Phi \subseteq \mathcal{E}L(\Sigma) \), \( \text{Mod}_\Sigma(\Phi) \) is an abstract data type. For practical purposes, however, abstract data types of this kind are corrupted by too much polymorphism: they contain the trivial algebras with one element per sort which are hardly ever acceptable as representatives of intended data types ‘up to irrelevant details’. Thus, equational logic alone is not powerful enough for specifying useful abstract data types, especially monomorphic ones.

Adding initiality as a further specification concept helps. Let \( T(\Sigma, \Phi) = T(\Sigma, \text{Mod}_\Sigma(\Phi)) \).

**G.10.4 Theorem** Let \( \Sigma \) be a signature and \( \Phi \subseteq \mathcal{E}L(\Sigma) \) a set of equations. Then \( T(\Sigma, \Phi) \) is initial in \( \text{Mod}_\Sigma(\Phi) \).

According to corollary G.10.3, all what has to be proved is that \( T(\Sigma, \Phi) \in \text{Mod}_\Sigma(\Phi) \), cf. Loeckx et al. (1996, theorem 7.2). Since \( \text{Mod}_\Sigma(\Phi) \) therfore has initial algebras, the following definition makes sense.
G.10.5 Definition An initial abstract data type specification in equational logic, or initial specification for short, consists of (i) the abstract syntax: an equational specification \( D = (\Sigma, \Phi) \) where \( \Sigma \) is a signature and \( \Phi \subseteq \text{EL}(\Sigma) \) is a set of equations, and (ii) the semantics or meaning: the class \( \mathcal{M}(D) \) of initial algebras in \( \text{Mod}_\Sigma(\Phi) \).

For any equational specification \( D, \mathcal{M}(D) \) is a monomorphic abstract data type. \( T(\Sigma, \Phi) \) is an obvious representative. A salient feature of initial semantics is that, in most cases of practical interest, there is a compatible operational semantics, namely term rewriting (cf. Section G.7).

A reduction system is a pair \( (R, \rightarrow) \) where \( R \) is a set and \( \rightarrow \) a binary relation on \( R \). A reduction sequence of \( (R, \rightarrow) \) is a possibly infinite sequence \( r_1, \ldots, r_k, \ldots \) of elements of \( R \) such that \( r_i \rightarrow r_{i+1} \) for each \( i \geq 1 \); for any \( k \geq 1 \) we write \( r_1 \rightarrow^* r_k \). A normal form of \( r \) is an element \( s \in R \) such that \( r \rightarrow^* s \) and there is no \( t \in R \) such that \( s \rightarrow t \). An equivalence sequence is defined like a reduction sequence but with \( r_i \rightarrow r_{i+1} \text{ or } r_{i+1} \rightarrow r_i \) for each \( i \geq 1 \); for each \( k \geq 1 \), we write \( r_1 \simeq r_k \). It is easy to see that \( \simeq \) is an equivalence relation. A reduction system is called Noetherian if it possesses no infinite reduction sequences; it is called confluent if for all \( r, s, t \in R \) the following holds: if \( r \rightarrow^* s \) and \( r \rightarrow^* t \), then there exists an element \( u \in R \) such that \( s \rightarrow^* u \) and \( t \rightarrow^* u \). If \( (R, \rightarrow) \) is Noetherian and confluent, then each element of \( R \) has exactly one normal form, and for any two elements \( r, s \in R \), \( r \simeq s \) holds iff \( r \) and \( s \) have the same normal form.

G.10.6 Definition The term rewriting system for an initial specification \( D = (\Sigma, \Phi) \) is a reduction system \( (T_\Sigma, \rightarrow) \) where \( \rightarrow \) is inductively defined by (i) \( u \sigma \rightarrow w \sigma \) for each equation \( \forall X. w = w \in \Phi \) and for each substitution \( \sigma : X \rightarrow T_\Sigma \), and (ii) if \( t \rightarrow u \), then \( s[t/y] \rightarrow s[u/y] \) for all terms \( s \in T_{\Sigma(\{y\})} \) containing at least one occurrence of the variable \( y \).

G.10.7 Theorem Let \( (T_\Sigma, \rightarrow) \) be the term rewriting system of an equational specification \( D = (\Sigma, \Phi) \). Let \( v, w \in T_\Sigma \) be two ground terms of the same sort. Then \( \Phi \models v = w \) iff \( v \simeq w \).

Noetherian and confluent term rewriting systems provide a useful operational semantics: in order to prove that \( v \simeq w \), it is sufficient to prove that \( u \) and \( v \) have the same normal form. Both properties, however, are undecidable. There is a sufficient condition for being Noetherian (see, e.g., Loeckx et al. (1996, subsection 7.5.5)). Confluence may often be achieved by the Knuth-Bendix completion algorithm (see,
e.g., Klop (1992)) which, where applicable, transforms a specification
with a Noetherian but nonconfluent term rewriting system into another
specification with the same initial semantics but with a Noetherian and
confluent term rewriting system.

Computation by term rewriting does not precisely take place in
\( T(\Sigma, \Phi) \) where the carrier elements are congruence classes of terms, but
in a characteristic term algebra \( C(\Sigma, \Phi) \) for \( T(\Sigma, \Phi) \) where the carrier
elements are the normal forms of \( (T_\Sigma, \rightarrow) \). \( C(\Sigma, \Phi) \) is isomorphic to
\( T(\Sigma, \Phi) \).

Equational initial specification may be generalized to conditional
equations (cf. Section G.9) of the form \( \forall X. t_1 = u_1 \wedge \cdots \wedge t_n = u_n \Rightarrow
\ t_{k+1} = u_{k+1} \). Most results go through, but the operational semantics of
term rewriting is far more complex.

Among other approaches to abstract data type specification, we men-
tion loose specification \( D = (\Sigma, \Phi) \) using first-order logic, defining
just the model class \( \text{Mod}_\Sigma(\Phi) \) as its semantics. While the degenerate
models of equational logic can be avoided in first-order logic, the disad-
vantage is that there are non-generated models. This may be remedied
by loose specifications with free constructors where a subset of sorts
and operations may be specified with the intended meaning that the car-
riers of these sorts are (freely) generated. All these approaches enjoy a
clean mathematical semantics but do not have an equivalent of the opera-
tional semantics of the initial approach. The latter is remedied (at the
expense of mathematical semantics) by constructive specifications.
These are particular cases of loose or initial specifications that have an
abstract programming flavor, they allow rapid prototyping.

So far, we considered specification-in-the-small, i.e., how to create a
specification by giving a signature and axioms. There is also a body
of theory dealing with specification-in-the-large, i.e., how to derive a
new specification from a given one by renaming, extending, forgetting
or restricting sorts and operations; the algebraic essence is to handle
relationships between algebras with different signatures. Modulariza-
tion and parameterization concepts deal with possibly generic specifi-
cation fragments and how to put them together. Further topics deal
with behavioral abstraction, implementation, ordered sorts, exceptions,
dynamic data types and objects. Loeckx et al. (1996) gives an in-depth
treatment of these topics and references for further reading.