Correctness of full first-order specifications

Wolfgang Reif

Angaben zur Veröffentlichung / Publication details:

Reif, Wolfgang. 2003. "Correctness of full first-order specifications." In *Proceedings Fourth International Conference on Software Engineering and Knowledge Engineering, 15-20 June 1992, Capri, Italy*, 276–83. Piscataway, NJ: IEEE. https://doi.org/10.1109/seke.1992.227918.

Correctness of Full First-Order Specifications

Wolfgang Reif

Institut für Logik, Komplexität und Deduktionssysteme
Universität Karlsruhe, FRG, reif@ira.uka.de

Abstract

The paper investigates an algebraic specification method for abstract data types which is designed for the application in formal software development. The specification language is full first-order logic, and the semantics of a specification is the class of its generated models. Full first-order specifications are more flexible than Horn clause specifications and exhibit better deductive properties. We present criteria for the correctness of full first-order specifications and for the incremental development of large specifications from smaller ones.

1 Introduction

Since the early nineteen-seventies a number of methods for the algebraic specification of abstract data types have been designed. Most of these apporaches consider a specific class of algebras as the semantics of a specification. Candidates for this class are the initial models ([LZ 74], [Gu 76], [GTW 78], [TWW82], [Ga 83], [Pa 83], [Pa 85], [EM 85], [Pa 90], [EM 90]) or terminal models ([Wa 79], [HR 80], [Ka 83]). All these approaches restrict the specification language in order to guarantee that every specification has a model of this particular kind. In the case of the initial semantics of abstract data types the specification language is restricted to universal equations or Horn clauses (for the extension by constraints see [Pa 90], [EM 90]).

In this paper we argue that from the view point of a specific application, equational and Horn clause specifications suffer from two defects which motivate the need for a more flexible specification method. The application refers to the automated verification and development of formally specified computer software ([HRS 88], [HRS 89], [HRS 90]). In this context, Horn clause specifications exhibit a "deductive weakness" when we try to verify properties of a specification or properties of (abstract) programs over a specification. Furthermore, the syntactic corset of universal Horn clauses leads to a considerable loss of flexibility in practical applications. In section 3 we illustrate that this restriction may lead to unnecessarily complicated solutions even in simple examples.

In order to avoid these defects we prefer another specification method. We adopt an approach of Giarratana et al. ([GGM 76]) and the Munic CIP-group ([BDPPW 79], [BW 82], [WPPDB 83], [BWP 84], [CIP 87]). This approach allows full first-order specifications and considers the class of all generated models of a specification as its semantics. This semantics is somehow intermediate between initial and loose semantics. In contrast to Horn clause specifications full first-order specifications do not always have a generated model. This means that the classical methods of universal algebra and category theory are not directly applicable. Therefore, we get two groups of questions:

1) What can be said about the structure of generated models of a given specification? Under which conditions do exist initial and terminal models?

2) Is it possible to design a specification method with this semantics, which allows the unrestricted use of full first-order logic, but preserves at the same time the nice properties of the classical algebraic specification method? This refers to a correctness criterion for specifications as well as to the posibility to develop large correct specifications by union and enrichment.

The first group of questions were investigated by Broy, Wirsing et al. ([BDPPW 79], [BW 82], [WPPDB 83], [BWP 84]) for the case of hierarchical data types and were solved for a subclass of first-order formulas. In this paper we investigate the second group of questions for the unparameterized case and propose a practically applicable method to develop correct first-order specifications avoiding the weaknesses of Horn clause specifications. We present syntactic criteria both for the correctness of a specification and for the union and enrichment of correct specifications.

This research was partly supported by the BSI project VSE and the BMFT project KORSO

Section 2 recalls some basic definitions. In section 3 we illustrate our criticism of equational and Horn clause specifications by two examples. Section 4 presents the syntax and the semantics of full first-order specifications. In section 5 we develop a criterion for monomorphic and correct specifications, and section 6 deals with the incremental development of larger specifications from smaller ones. Finally, in section 7, we draw some conclusions.

2 Basic notions

We consider many-sorted *signatures* SIG = (S, OP) with a set of sorts S and a set OP of operations over S. S^* are the finite words over S with empty word λ. OP is the disjoint union of sets OP_w with $w \in S^*$ and $OP_{w,s}$ with $w \in S^*$ and $s \in S$. For $w = s_1 .. s_n$ OP_w is the set of n-ary predicate symbols with the indicated typing, and $OP_{w,s}$ is the set of function symbols with argument sorts w and target sort s. For every sort $s \in S$ there is a countably infinite set X_s of variables. X is the system of all these variable sets. For SIG = (S, OP) and X, $T_{OP,s}(X)$ denotes the *terms* of sort s over OP and X. $T_{OP,s}$ is the corresponding set of ground terms (= $T_{OP,s}(\varnothing)$). $T_{OP}(X)$ and T_{OP} are the sets of all terms and ground terms, respectively. A signature SIG is called *strict* if there exists a ground term for every sort. Henceforth, we will only consider strict signatures. *L(SIG, X)* is the set of first-order formulas over SIG and X. *$\forall$L(SIG, X)* is the set of universal formulas. If X is clear we write L(SIG). Universal Horn clauses are formulas of the shape $Cl\forall(\varphi_1 \wedge .. \wedge \varphi_n \rightarrow \varphi)$ where φ_1 , .., φ_n, φ are atomic formulas and $Cl\forall$ is the universal closure. Universal equations take the form $Cl\forall(t_1 = t_2)$. Equational or Horn clause specifications are meant to be sets of formulas of the corresponding kind. For convenience we mostly drop the universal quantifiers.

Formulas are interpreted over SIG-algebras. For SIG = (S, OP) a *SIG-algebra* $\mathcal{A} = ((A_s)_{s\in S}, (f_{\mathcal{A}})_{f\in OP})$ consists of non-empty carrier sets A_s and the interpretations $f_{\mathcal{A}}$ for the symbols from OP. The interpretation $f_{\mathcal{A}}$ is a total function or a predicate depending on what kind of symbol f is. We use a relaxed notation for algebras by just enumerating the components, for example, ($\mathbb{N}$, 0, succ, <) for the natural numbers with zero, successor and the less-than predicate. The class of SIG-algebras is abbreviated by Alg(SIG). If SIG' = (S', OP') is a supersignature of SIG = (S, OP) (i.e. $S' \supseteq S$, $OP' \supseteq OP$) and $\mathcal{A}$ a SIG-algebra, then $\mathcal{B} \in Alg(SIG')$ is an *expansion* of $\mathcal{A}$, if $A_s = B_s$ for all $s\in S$ and $f_{\mathcal{A}} = f_{\mathcal{B}}$ for all $f \in OP$. $\mathcal{A}$ is called the *reduct* of $\mathcal{B}$ wrt. SIG, $\mathcal{B}|_{SIG} = \mathcal{A}$. An algebra is called *generated* wrt. SIG if every carrier element can be denoted by a ground term over SIG. For strict signatures every SIG-algebra $\mathcal{A}$ has a *minimal subalgebra*, $\mathcal{A}\downarrow$, which is a generated subalgebra of $\mathcal{A}$ which in turn has no proper subalgebras. We write $\mathcal{A} \models \varphi$ if $\mathcal{A}$ is a model of the formula φ, $T \models \varphi$ if every model of the formula set T is a model of φ. Alg(T), Gen(T) are the classes of all models and all generated models of T, rsp.

Statements about generated algebras can be proved by structural induction. IND(SIG) denotes the set of first order instances of a schema for *structural induction* over SIG = (S, OP). IND(SIG) is defined as follows: Let $\phi = <\varphi_s : s\in S>$ a vector of formulas φ_s and $x = <x_s: s \in S>$ a vector of induction variables x_s of sort $s\in S$. We set $IND(SIG) = \{ (Basis(x, \phi) \wedge Hyp(x, \phi)) \rightarrow Concl(x, \phi) : \phi = \langle\varphi_s : s\in S\rangle, x = <x_s: s\in S>, \varphi_s \in L(SIG)\}$ with:

$Basis(x, \phi) := \wedge_{c\in OP_{\lambda,s}} \varphi_s(c)$
$Concl(x, \phi) := \wedge_{s\in S} \forall x_s.\varphi_s$
$Hyp(x, \phi) := \wedge_{f\in OP_{w,s}} Step(f, x, \phi)$
For $f \in OP_{w,s}$ with $w = s_1..s_r$, and new variables $x_1, .., x_r$, $Step(f, x, \phi)$ is the formula:
$\forall x_1,.., x_r. (\wedge_{j=1..r} \varphi_{s_j}(x_j)) \rightarrow \varphi_s(f(x_1, .., x_r))$.

Finally, we define the *initial model* of a Horn clause specification. Let SPEC a set of Horn clauses over a strict signature SIG = (S, OP). Then we define a SIG-algebra $\mathcal{H} = ((H_s)_{s\in S}, (f_{\mathcal{H}})_{s\in S})$ as follows: $H_s = T_{OP,s}$ for $s\in S$. For $f \in OP_{w,s}$, $w = s_1...s_n$ we set $f_{\mathcal{H}}(t_1, .., t_n) := f(t_1, .., t_n)$, and for $q \in OP_w$, $w = s_1...s_n$ we define $(t_1, .., t_n) \in q_{\mathcal{H}} \Leftrightarrow SPEC \models q(t_1, .., t_n)$. Furthermore, we define a family of relations $\equiv := (\equiv_s)_{s\in S}$ with $\equiv_s \subseteq H_s \times H_s$ by $t_1 \equiv_s t_2 \Leftrightarrow SPEC \models t_1 = t_2$. The relation $\equiv$ is a congruence on $\mathcal{H}$. Now we define $\mathcal{T}_{SPEC}$ to be the quotient of $\mathcal{H}$ by $\equiv$: $\mathcal{T}_{SPEC} := \mathcal{H}/\equiv$. $\mathcal{T}_{SPEC}$ is called the initial model of SPEC. SPEC is *initial correct* for $\mathcal{A} \in Alg(SIG)$ if $\mathcal{A}$ is isomorphic to $\mathcal{T}_{SPEC}$.

3 Criticism of Horn specifications

In this section we demonstrate two weaknesses of equational- and Horn clause specifications which motivate the need for a more flexible specification method. The first one refers to a certain "deductive weakness" in situations where the formal derivation of a property of the specified model requires more knowledge than is given explicitly in the specification. The second weakness refers to a lack of flexibility due to the strong syntactical restriction to universal equations or Horn formulas. This restriction imposes a constructive (executable) style of specification even in cases where this is rather inadequate. The following two examples illustrate these phenomena.

3.1 Example (numbers, program verification)

Consider the algebra of natural numbers $\mathcal{N}at$ = ($\mathbb{N}$, 0, succ, pred, +, *) with zero, successor (succ), predecessor

(pred, pred(0) = 0), addition and multiplication. The following specification $SPEC_1$ is initial correct for $\mathcal{N}at$.

$SPEC_1$ =

$$\{ \begin{array}{ll} pred(0) = 0 & pred(succ(x)) = x \\ x + 0 = x & x + succ(y) = succ(x+y) \\ x * 0 = 0 & x * succ(y) = (x * y) + x \end{array} \}$$

As usual, the formulas of $SPEC_1$ have to be read with tacit universal quantification. Furthermore, we do not distinguish between the symbols of the signature (in $SPEC_1$) and their meanings (in $\mathcal{N}at$). Now we consider the following very simple program α over the signature of $\mathcal{N}at$ (**abort** denotes the never halting program):

$$\alpha = (\textbf{if } x = 0 \textbf{ then abort else } x := succ(0) \textbf{ fi}).$$

If x = succ(succ(0)) holds then the test of α is false, α terminates and yields x = succ(0) as a result. Formulated as a total correctness assertion (in terms of Dynamic Logic, [Go 82]) we get:

$$\mathcal{N}at \models x = succ(succ(0)) \rightarrow \langle\alpha\rangle x = succ(0)$$

Although $SPEC_1$ is initial correct, this statement does not follow from the formulas in $SPEC_1$ (even if enriched by induction axioms):

$$SPEC_1 \not\models x = succ(succ(0)) \rightarrow \langle\alpha\rangle x = succ(0)$$

The reason is that the above statement follows from $SPEC_1$ if and only if $SPEC_1$ entails the inequality $\neg\ 0 = succ(succ(0))$. The inequality $\neg\ 0 = succ(succ(0))$, however, does not follow from $SPEC_1$.

Equational specifications generally reflect explicitly only knowledge about the equality of expressions. The fact that all ground terms, which can not be proved equal, denote different objects, follows implicitly from the semantical restriction to initial algebras. Therefore, in a sense, specification with initial semantics and classical deduction to not fit together.

The situation changes if one adds to the specification explicit information about the inequality of expressions:

$$SPEC_2 = SPEC_1 \cup \{\neg\ 0 = succ(x)\}$$

Up to isomorphism $\mathcal{N}at$ is the only generated model of $SPEC_2$. For $SPEC_2$ the following holds:

$$SPEC_2 \models \neg\ \ 0 = succ(succ(0))$$
$$SPEC_2 \models x = succ(succ(0)) \rightarrow \langle\alpha\rangle x = succ(0)$$

This example illustrates, that equational specifications exhibit a certain deductive weakness. The example suggests to allow the use of explicit negation in specifications, in order to improve their deductive properties, and to consider the class of generated models as the semantics of the specification.

3.2 Example (non constructive specifications)

Let us consider again $SPEC_1$. We extend the signature and the algebra $\mathcal{N}at$ by the binary predicate *nd*, which holds for two numbers x and y if x is not a divisor of y: $\mathcal{N}at'$ = ($\mathbb{N}$, 0, succ, pred, +, *, nd). We are looking for an extension $SPEC_3$ of $SPEC_1$ such that $SPEC_3$ is initial correct for $\mathcal{N}at'$. In order to specify the *nd* predicate some auxiliary operations are needed: remainder (mod), subtraction (-) and the ordering (>) on numbers. The following Horn clause specification $SPEC_3$ describes $\mathcal{N}at''$ = ($\mathbb{N}$, 0, succ, pred, +, *, >, - , mod, nd):

$SPEC_3 = SPEC_1 \cup$

$$\{ \begin{array}{l} succ(k) + x = y \rightarrow y > x \\ x - 0 = x \\ x - succ(y) = pred(x - y) \\ (y \bmod 0) = y \\ (y \bmod y) = 0 \\ x > 0 \wedge y > x \rightarrow (y \bmod x) = ((y - x) \bmod x) \\ x > y \rightarrow (y \bmod x) = y \\ (y \bmod x) > 0 \rightarrow nd(x, y) \end{array} \}$$

The last axiom specifies nd(x, y), while the remaining seven axioms describe the auxiliary operations. $SPEC_3$ is indeed initial correct for $\mathcal{N}at''$. The criticism of this specification is obvious: In order to describe one new predicate we had to introduce three new operations, and the length of the specification has more than doubled. The main argument, however, is that the specification method imposes an algorithmic description of the nd predicate, which is rather an implementation of a non-divisibility test than an explanation of what non-divisibility means. As a consequence the specification is more detailed than necessary and gets incomprehensible. This phenomenon is a general problem of Horn specifications with initial semantics.

The situation changes if we start out with $SPEC_2$ and allow the use of full first-order logic. In this context the extension $\mathcal{N}at'$ = ($\mathbb{N}$, 0, succ, pred, +, *, nd) of $\mathcal{N}at$ can be specified very easily:

$$NAT' = SPEC_2 \cup \{nd(x, y) \leftrightarrow \neg\ \exists k.\ x * k = y\}$$

NAT' seems to be the shortest and most natural specification for $\mathcal{N}at'$. It does not need any auxiliary operations and is a unique description of $\mathcal{N}at'$: $\mathcal{N}at'$ is the only generated model of NAT' up to isomorphism.

The examples suggest to use full first-order logic to specify abstract data types and to consider the class of ge-

nerated models as the semantics of such a specification. This approach is adopted in the following section.

4 First-order specifications

4.1 Definition (syntax)

A *specification* SPEC = (S, OP, X, T) consists of a signatur SIG = (S, OP), a set X of variables and a set T of first-order formulas over SIG and X. A specification $SPEC_1 = (S_1, OP_1, X_1, T_1)$ is a *subspecification* of $SPEC_2 = (S_2, OP_2, X_2, T_2)$, $SPEC_1 \subseteq SPEC_2$, if $S_1 \subseteq S_2$, $OP_1 \subseteq OP_2$, $X_1 \subseteq X_2$, $T_1 \subseteq T_2$. In this case we also write $SPEC_2 = SPEC_1 \cup (S_2 - S_1, OP_2 - OP_1, X_2 - X_1, T_2 - T_1)$ or just $SPEC_2 = SPEC_1 \cup (T_2 - T_1)$.

The semantics SEM(SPEC) of a specification SPEC is the class of all generated models of SPEC, Gen(SPEC). Every algebra $\mathcal{A}$ from Gen(SPEC) is called a data type which satisfies SPEC. The whole class Gen(SPEC) of data types is considered an *abstract data type.*

4.2 Definition (semantics, satisfaction)

For a specification SPEC = (S, OP, T, X) we set *SEM(SPEC)* = Gen(SPEC) = $\{\mathcal{A} : \mathcal{A} \models T, \mathcal{A}$ generated$\}$. An algebra $\mathcal{A}$ *satisfies* SPEC, if $\mathcal{A} \in$ SEM(SPEC). SPEC is *satisfiable*, if there exists an $\mathcal{A}$ which satisfies SPEC.

A minimal requirement for the semantics of abstract data types is the closure under isomorphisms. This is fulfilled by SEM(SPEC). Since we allow full first-order logic there is no guarantee that every specification has a model. SEM(SPEC) may be empty. For example, SEM(SPEC) is empty if SPEC is inconsistent. But even if SPEC is consistent SEM(SPEC) may be empty. This is illustrated by the following example: Let SPEC = (S, OP, X, T) with S = {s}, OP = {c: $\rightarrow$ s} and T = {$\exists$x. $\neg$x=c}. SPEC (resp.T) is consistent but has no generated model.

If we consider particular specifications the situation changes. For universal specifications SPEC = (S, OP, X, T) over a strict signature with $T \subseteq \forall L(SIG)$, SEM(SPEC) $\neq \emptyset$ if and only if SPEC is consistent.

From a pragmatic point of view the lack of a guaranteed model does not cause any severe drawbacks. In our opinion the central aim of a specification method is not primarily that every specification has a model but to specify given algebras up to isomorphism or to capture at least some relevant properties. From this perspective writing a specification means to accumulate properties of existing generated models. Therefore, SEM(SPEC) is not empty by construction. SEM(SPEC) contains at least the intended model or the intended collection of models. The question how to detect that an algebra is captured completely by the accumulated properties is answered in section 5. In general SEM(SPEC) may contain non-isomorphic generated models. $SEM(SPEC_1)$ from section 3, for example, contains two different classes of isomorphic generated models, namely the class containing $\mathcal{N}at'$ and the class of the trivial one-element model. This means that first-order specifications are somehow intermediate between the initial and the loose approach, allowing for both complete and incomplete specifications.

5 Monomorphicity and correctness

In this section we deal with the problem to find for a given generated algebra $\mathcal{A}$ a specification SPEC which determines $\mathcal{A}$ as precise as possible by its axioms. A specification SPEC which determines $\mathcal{A}$ uniquely up to isomorphism among the models satisfying SPEC, is called correct for $\mathcal{A}$. Correct specifications have the property that any two generated models of SPEC are isomorphic. This property is called monomorphicity, and will play a key role. A specification SPEC is correct for $\mathcal{A}$ if $\mathcal{A}$ satisfies SPEC, and SPEC is monomorphic. In this section we analyze the conditions under which specifications are monomorphic.

5.1 Definition (monomorphicity)

A specification is *monomorphic* if any two models of SEM(SPEC) are isomorphic.

The following analysis of monomorphic specifications is based on the simple fact that generated models are determined (up to isomorphism) by their diagram (the set of valid closed atomic and negated atomic formulas). The idea is to examine a specification whether it determines a diagram. To this end we need the notion of ground literals

5.2 Definition (ground literals)

Let SIG = (S, OP) a signature. *Ground literals* are formulas of the shape $t_1 = t_2$, $\neg t_1 = t_2$, $q(t_1,..,t_n)$ and $\neg q(t_1,..,t_n)$, where $t_1, t_2,.., t_n \in T_{OP}$ and $q \in OP_w$. Non-negated ground literals are called *positive.*

The following lemma shows that a specification is monomorphic if any two generated models coincide with respect to the validity of the positive ground literals.

5.3 Lemma (monomorphicity, ground literals)

SPEC is monomorphic if and only if for all $\mathcal{A}, \mathcal{B} \in$ SEM(SPEC) and all positive ground literals φ the equivalence $\mathcal{A} \models \varphi \Leftrightarrow \mathcal{B} \models \varphi$ holds.

We rely on the observation that in many examples a specification SPEC determines the validity of the ground literals not only for all generated models but for all models of SPEC which additionally satisfy structural induction. In this case SPEC enriched by IND(SIG) is complete for positive ground literals.

5.4 Definition (SPEC fixes the ground literals)

SPEC *fixes the ground literals* if for all positive ground literals φ, either SPEC $\cup$ IND(SIG) $\models \varphi$ or SPEC $\cup$ IND(SIG) $\models \neg \varphi$ holds.

The condition that SPEC fixes the ground literals is sufficient for monomorphicity, and necessary under certain restrictions.

5.5 Theorem (criterion for monomorphicity)

SPEC is monomorphic if it fixes the ground literals. If SPEC is closed under minimal subalgebras, then it is monomorphic if and only if SPEC fixes the ground literals.

Proof:

When SPEC fixes the ground literals any two generated models of SPEC coincide on the positive ground literals. Hence SPEC is monomorphic. If SPEC is closed under minimal subalgebras we argue indirectly as follows for the "only-if-part": If for a positive ground literal φ neiter SPEC $\cup$ IND(SIG) $\models \varphi$ nor SPEC $\cup$ IND(SIG) $\models \neg \varphi$ holds then there exist two models $\mathcal{A}, \mathcal{B} \in$ Alg(SPEC $\cup$ IND(SIG)) which differ wrt. φ. Since both SPEC $\cup \{\varphi\}$ and SPEC $\cup \{\neg\varphi\}$ are closed under minimal subalgebras, the minimal subalgebras $\mathcal{A}\downarrow, \mathcal{B}\downarrow$ differ wrt. φ. On the other hand $\mathcal{A}\downarrow, \mathcal{B}\downarrow$ are generated models of the monomorphic specification SPEC. Therefore they are isomorphic. This is a contradiction.

Theorem 5.5 provides a practically applicable criterion for a specification to be monomorphic. It suffices to show that SPEC enriched by structural induction is complete for positive ground literals. For specifications which additionally are closed under minimal subalgebras the criterion is both necessary and sufficient. This is true, for example, for universal specifications SPEC $\subseteq \forall$L(SIG) over a strict signature SIG. We are now going to define the correctness of a specification.

5.6 Definition (correctness of specifications)

SPEC is *correct* for $\mathcal{A}$, if $\mathcal{A}$ is generated, $\mathcal{A} \models$ SPEC and SPEC is monomorphic.

5.7 Corollary (correctness of specifications)

A specification SPEC is correct for $\mathcal{A}$, if

1) $\mathcal{A}$ is generated
2) $\mathcal{A} \models$ SPEC
3) for all positive ground literals φ : either SPEC $\cup$ IND(SIG) $\models \varphi$ or SPEC $\cup$ IND(SIG) $\models \neg \varphi$ holds.

Instead of 3) the following can be used equivalently:

4) for all ground literals φ:
   $\mathcal{A} \models \varphi \Rightarrow$ SPEC $\cup$ IND(SIG) $\models \varphi$

Corollary 5.7 is based on 5.5 and can be used to prove that a specification SPEC is correct for an algebra $\mathcal{A}$. In the rest of the section we deal with two examples which illustrate the application of 5.7. The second one motivates why we enriched SPEC by the axioms for structural induction in 5.4.

5.8 Example (binary words)

We want to specify the one-sorted algebra $\mathcal{B}in0$ = ({zero, one}$^+$, zero, one, s0, s1) of the (non-empty) finite binary words over {zero, one}. SIG$_{\mathcal{B}in0}$ = ({bin}, {zero, one : $\rightarrow$ bin, s0, s1 : bin $\rightarrow$ bin}). The sort bin is the only sort, zero and one are the two elementary binary words, s0, s1 are unary constructors which attach zero or one, respectively to the end of a word w. Again, we do not distinguish between the symbols of the signature and their meanings. The following specification BIN0 is correct for $\mathcal{B}in0$.

BIN0 =

{ 1) $\neg$ zero = one 2) $\neg$ zero = s0(w)
  3) $\neg$ zero = s1(w) 4) $\neg$ one = s0(w)
  5) $\neg$ one = s1(w) 6) $\neg$ s0(w) = s1(w')
  7) s0(w) = s0(w') $\leftrightarrow$ w = w'
  8) s1(w) = s1(w') $\leftrightarrow$ w = w' }

We prove the correctness of BIN0 with corollary 5.6. Obviously, $\mathcal{B}in0$ is generated and $\mathcal{B}in0 \models$ BIN0. It remains to prove that BIN0 $\cup$ IND(SIG$_{\mathcal{B}in0}$) is complete for positive ground literals. We will see that this is already true for BIN0. In SIG$_{\mathcal{B}in0}$ non-negated equations are the only positive ground literals. We show that for all ground terms t_1 and t_2 BIN0 $\models t_1 = t_2$ or BIN0 $\models \neg\, t_1 = t_2$ hold, by induction (on the meta level) on the structure of t_1:

t_1 = zero:

If t_2 is zero, too, BIN0 $\models t_1 = t_2$ holds vacuously. If t_2 is one, BIN0 $\models \neg\, t_1 = t_2$ follows from axiom 1). Otherwise, t_2 starts either with s0 or with s1. In both cases BIN0 $\models \neg\, t_1 = t_2$ follows from the axioms 2) and 3), rsp.

t_1 = one:

As in the case before with axioms 1), 4) and 5).

t_1 = s0(t_3):

If t_2 is zero or one, or starts with s1, then BIN0 $\models \neg\ t_1 = t_2$ follows with 2), 4) and 6) respectively. If $t_2 = s0(t_4)$, then we get from the induction hypothesis BIN0 $\models\ t_3 = t_4$ or BIN0 $\models \neg\ t_3 = t_4$. Axiom 7) yields BIN0 $\models\ t_1 = t_2$ or BIN0 $\models \neg\ t_1 = t_2$.

$t_1 = s1(t_3)$:

As in the case before with axioms 3), 5) 6) and 8).

This example did not make use of the induction axioms $IND(SIG_{Bin0})$. This is a frequent situation. However, it may happen that the induction axioms are needed.

5.9 Example (use of IND(SIG))

Let us consider a signature with a single sort s and two constants c_1 and c_2. Let SPEC = $\{\exists x.\ \neg x = c_1\}$. The ground term algebra $\mathcal{T} = (\{c_1, c_2\}, c_1, c_2)$ is a generated model of SPEC. Since, on the other hand, all generated models of SPEC have two elements, and c_1 and c_2 have to be interpreted differently, it follows that any two models from SEM(SPEC) are isomorphic. Therefore, SPEC is monomorphic and by 5.6 correct for $\mathcal{T}$. Let us try to prove this fact by 5.7: There are only four positive ground literals: $c_1 = c_1$, $c_2 = c_2$, $c_1 = c_2$, $c_2 = c_1$. The first two follow from SPEC. Neither the remaining two nor their negations follow from SPEC. This means that without adding IND(SIG) the ground literals would not be fixed. With IND(SIG), however, we get SPEC $\cup$ IND(SIG) $\models \neg\ c_1 = c_2$ and SPEC $\cup$ IND(SIG) $\models \neg\ c_2 = c_1$. In this case we deduce the correctness by 5.7.

6 Incremental development

The correctness criterion in section 5 is suitable to prove that a given specification describes a given data type correctly. For large specifications, however, this proof is very extensive and designing the specification requires a lot of intuition. The reason is, that for large specifications over a rich signature, there exist many ground literals of different shapes. It becomes hard to keep track of all the case distinctions necessary to prove the specification monomorphic. These problems can be avoided, if one starts out with a small fragment of the data type with only a small subsignature which can be correctly specified, using the criterion from section 5. Then this fragment is extended incrementally in a correctness preserving manner. The means for extending a correct specification are designed in a way that the correctness proofs done for the components need not be repeated. This leads to an incremental and tractable way of developing large correct specifications.

The central methods for the extension are the disjoint union and the enrichment of data types and specifications. The extension by disjoint union is used to provide new sorts, the enrichment serves to add new operations.

6.1 Definition (disjoint union)

Two specifications SPEC (over SIG) and SPEC' (over SIG') are disjoint if all the components are disjoint. With *SPEC+SPEC'* (over the disjoint union SIG+SIG') we denote the component-wise union of the two disjoint specifications. If SIG and SIG' are disjoint, and $\mathcal{A}$ and $\mathcal{A}'$ are SIG- and SIG'-algebras, respectively, then the disjoint union $\mathcal{A}+\mathcal{A}'$ results from glueing together $\mathcal{A}$ and $\mathcal{A}'$.

6.2 Theorem (disjoint union and correctness)

Let SPEC, SPEC' be disjoint. SPEC+SPEC' is correct for $\mathcal{A}+\mathcal{B}$ if SPEC is correct for $\mathcal{A}$ and SPEC' correct for $\mathcal{B}$.

We now turn our attention to the enrichment of data types by new functions and predicates. This requires to extend a given signature SIG = (S, OP) by new function symbols $f_1, .., f_n$ and new predicate symbols $q_1, .., q_k$. We just write OP $\cup \{f_1, .., f_n, q_1, .., q_k\}$ and assume that the typing of the symbols is clear.

6.3 Definition (enrichment of data types)

Let SIG = (S, OP), SIG' = (S, OP $\cup \{f_1, .., f_n, q_1, .., q_k\}$) and $\mathcal{A} \in$ Alg(SIG).We denote the *enrichment* of $\mathcal{A}$ by functions $\mathbf{f_1}, .., \mathbf{f_n}$ and predicates $\mathbf{q_1}, .., \mathbf{q_k}$ to a SIG'-algebra by $\mathcal{A}+\{\mathbf{f_1}, .., \mathbf{f_n}, \mathbf{q_1}, .., \mathbf{q_k}\} := \mathcal{A}'$, where $A_s' = A_s$ for $s \in S$, $f_{\mathcal{A}'} = f_{\mathcal{A}}$ for $f \in$ OP, $f_{\mathcal{A}'} = \mathbf{f}$ for $f \in \{f_1, .., f_n, q_1, .., q_k\}$.

Our aim is to get a correct specification for an enrichment of a data type from a correct specification of the original data type by adding new axioms describing the new symbols. The following criterion states that the correctness of the original specification carries over to the new specification for the enriched data type if the latter satisfies the additional axioms, and if the new symbols can be eliminated from ground terms and atomic formas using the original specification and the additional axioms. Criteria of this kind play an important role in many specification methods (cf. Gu 76], [WPPDB 83] "sufficient completeness" or [EM 85] "complete extension").

6.4 Theorem (enrichment and correctness)

Let SPEC a specification over SIG = (S, OP) and Ax axioms over SIG' = SIG $\cup$ ($\varnothing$, $\{f_1, .., f_n, q_1, .., q_k\}$). SPEC $\cup$ Ax is correct for $\mathcal{A}+\{\mathbf{f_1}, .., \mathbf{f_n}, \mathbf{q_1}, .., \mathbf{q_k}\}$ if

1) SPEC correct for $\mathcal{A}$
2) $\mathcal{A}+\{\mathbf{f_1}, .., \mathbf{f_n}, \mathbf{q_1}, .., \mathbf{q_k}\} \models$ Ax

3) for all $t \in T_{OP \cup \{f_1, .., f_n\}}$ there exists $t' \in T_{OP}$, such that SPEC $\cup$ IND(SIG') $\cup$ Ax $\models t = t'$

4) for all $q(t_1, .., t_m)$ with $q \in \{q_1, .., q_k\}$, $t_1, .., t_m \in T_{OP}$ exists $\varphi \in L(SIG, X)$, such that: SPEC $\cup$ IND(SIG) $\cup$ Ax $\models q(t_1, .., t_m) \leftrightarrow \varphi$

Proof:

We show that $\mathcal{A}' := \mathcal{A} + \{f_1, .., f_n, q_1, .., q_k\}$ is in SEM(SPEC $\cup$ Ax) and that SPEC $\cup$ Ax is monomorphic.

$\mathcal{A}' \in$ SEM(SPEC $\cup$ Ax):

From 1) and 2) we get $\mathcal{A}' \models$ SPEC $\cup$ Ax. Since $\mathcal{A}$ is generated and $\mathcal{A}'$ has the same carriers as $\mathcal{A}$, $\mathcal{A}'$ is generated, too, consequently $\mathcal{A}' \in$ SEM(SPEC $\cup$ Ax).

Monomorphicity of SEM(SPEC $\cup$ Ax):

Before we start we check the following fact (*): With $\mathcal{B} \in$ SEM(SPEC $\cup$ Ax) we also get $\mathcal{B}|_{SIG} \in$ SEM(SPEC). Let $\mathcal{B} \in$ SEM(SPEC $\cup$ Ax). Then $\mathcal{B}$ is generated with respect to SIG' and $\mathcal{B} \models$ SPEC $\cup$ Ax $\cup$ IND(SIG'). Because of 3) $\mathcal{B}$ is even generated with respect to SIG and consequently $\mathcal{B}|_{SIG} \in$ SEM(SPEC), especially $\mathcal{B} \models$ IND(SIG).

Let now $\mathcal{B}_1, \mathcal{B}_2 \in$ SEM(SPEC $\cup$ Ax). We show that $\mathcal{B}_1$ and $\mathcal{B}_2$ coincide on the positive ground literals.

We start with ground equations $t_1 = t_2$. Because of 3) there exist SIG-terms t_1', t_2' with SPEC $\cup$ IND(SIG') $\cup$ Ax $\models t_1 = t_1'$ and SPEC $\cup$ IND(SIG') $\cup$ Ax $\models t_2 = t_2'$. Since $\mathcal{B}_1 \models$ SPEC $\cup$ IND(SIG') $\cup$ Ax and $t_1', t_2' \in T_{OP}$ we get: $\mathcal{B}_1 \models t_1 = t_2 \Leftrightarrow \mathcal{B}_1 \models t_1' = t_2' \Leftrightarrow \mathcal{B}_1|_{SIG} \models t_1' = t_2'$. Since $\mathcal{B}_1|_{SIG}$ and $\mathcal{B}_2|_{SIG}$ are in SEM(SPEC) according to (*), it follows by the monomorphism of SPEC (premise 1) $\mathcal{B}_1|_{SIG} \models t_1' = t_2' \Leftrightarrow \mathcal{B}_2|_{SIG} \models t_1' = t_2' \Leftrightarrow \mathcal{B}_2 \models t_1' = t_2'$. Since $\mathcal{B}_2 \models$ SPEC $\cup$ IND(SIG') $\cup$ Ax we get, according to the choice of t_1' and t_2', $\mathcal{B}_2 \models t_1' = t_2' \Leftrightarrow \mathcal{B}_2 \models t_1 = t_2$.

Let us now consider the ground literals of the shape $q(t_1, .., t_r)$. Let $\mathcal{B}_1 \models q(t_1, .., t_r)$. According to 3) there exist terms $t_1', .., t_r' \in T_{OP}$ with SPEC $\cup$ IND(SIG') $\cup$ Ax $\models t_i = t_i'$ for $i \in \{1, .., r\}$. This yields $\mathcal{B}_1 \models q(t_1', .., t_r')$. We set $\varphi := q(t_1', .., t_r')$, if $q \in OP$. Then $q(t_1', .., t_r')$ is already in L(SIG, X). If $q \in \{q_1, .., q_k\}$, let φ according to premise 4) a formula from L(SIG, X) with SPEC $\cup$ IND(SIG) $\cup$ Ax $\models q(t_1', .., t_r') \leftrightarrow \varphi$. Because of $\mathcal{B}_1 \models$ SPEC $\cup$ IND(SIG') $\cup$ Ax we get by (*) $\mathcal{B}_1 \models$ IND(SIG). This yields $\mathcal{B}_1 \models \varphi$ and $\mathcal{B}_1|_{SIG} \models \varphi$. According to (*) $\mathcal{B}_1|_{SIG}$ and $\mathcal{B}_2|_{SIG}$ are in SEM(SPEC). Because of the monomorphism of SPEC we get $\mathcal{B}_2|_{SIG} \models \varphi$ and $\mathcal{B}_2 \models \varphi$. From $\mathcal{B}_2 \models$ SPEC $\cup$ IND(SIG') $\cup$ Ax and $\mathcal{B}_2 \models$ IND(SIG) we deduce $\mathcal{B}_2 \models q(t_1', .., t_r')$, and, with the choice of $t_1', .., t_r'$ in mind, $\mathcal{B}_2 \models q(t_1, .., t_r)$. The fact that $\mathcal{B}_1 \models q(t_1, .., t_r)$ follows from $\mathcal{B}_2 \models q(t_1, .., t_r)$ is shown by symmetric arguments.

Therefore we may conclude that SPEC $\cup$ Ax is monomorphic and correct for $\mathcal{A}'$.

6.5 Example (natural numbers)

We return to the example from section 3. Due to space limitations we only sketch the correctness proof of NAT' for the algebra $\mathcal{N}at' = (\mathbb{N}, 0, succ, pred, +, *, nd)$:

NAT' =

$\{ \neg\, 0 = succ(x) \quad pred(0) = 0$

$pred(succ(x)) = x \quad x * 0 = 0$

$x * succ(y) = (x * y) + x \quad x + 0 = x$

$x + succ(y) = succ(x+y) \quad nd(x, y) \leftrightarrow \neg\exists k.\ x * k = y\}$

We start with a fragment of $\mathcal{N}at'$, namely $\mathcal{N}at_0 = (\mathbb{N}, 0, succ)$, and enrich it by the other functions and the nd predicate: Let NAT0 = $\{\neg\, 0 = succ(x), succ(x) = succ(y) \rightarrow x = y\}$. With the help of 5.7 it is proved that NAT0 is correct for $\mathcal{N}at_0$. Then we enrich $\mathcal{N}at_0$ by pred, +, * and get $\mathcal{N}at_3 = (\mathbb{N}, 0, succ, pred, +, *)$. We set Ax := NAT' - $\{\neg\, 0 = succ(x), nd(x, y) \leftrightarrow \neg\exists k.\ x * k = y\}$ and NAT1 := NAT0 $\cup$ Ax. Using 6.4 we prove that NAT1 is correct for $\mathcal{N}at_3$. Now the injectivity of succ can be dropped from NAT1 since it is derivable. Next $\mathcal{N}at_3$ is enriched by nd and we get $\mathcal{N}at'$. We set Ax := $\{nd(x, y) \leftrightarrow \neg\exists k.\ x * k = y\}$ and NAT' := NAT1 $\cup$ Ax. Again, 6.4 is applicable, and we deduce that NAT' is correct for $\mathcal{N}at'$. In the last case the application of 6.4 is easy since for all ground terms t_1, t_2 over 0, pred, succ, +, *, there exists trivially a formula φ in the signature of $\mathcal{N}at_3$ such that NAT' $\models nd(t_1, t_2) \leftrightarrow \varphi$, namely $\neg\exists k.\ t_1 * k = t_2$.

7 Conclusion

We have investigated an algebraic specification method for (non-parameterized) abstract data types which allows to specify in full first-order logic. It was motivated by the restricted flexibility and the deductive weaknesses of Horn clause specifications in practical applications. Although the methods of universal algebra and category theory are not applicable in this context there exists a simple criterion for the monomorphicity and the correctness of a specification. Furthermore, we have shown that the "classical" criteria for the compositional development of correct specifications can be carried over to the more general framework. In this sense, the method avoids the drawback of restricted flexibility of Horn clause specifications while preserving their nice methodological aspects. Concerning the deductive properties, correct full first-order specifications (in the sense of 5.6) are (with restrictions) complete for ground formulas while this is not true for Horn clause specifications. In both approaches the central proof technique is structural induction. But more facts can be deduced by

induction from a correct first-order specification than from a initial correct Horn clause specification. And, finally, many typical computer science data types have decidable fragments. In these cases the specification can be designed in such a way that it is complete for the fragment. This is, in general, not possible for Horn clause specifications.

In this paper we did not deal with parameterized full first-order specifications. They can be treated along the same lines (see [Re 91]). Another aspect not treated in this paper is error handling. One might introduce partiality (see [BW 82]) or use a technique with error elements (see [SA 91]). A very useful next step would be to provide machine support for proving the correctness criteria.

References

[BDPPW 79] Broy, M., Dosch, H., Partsch, H., Pepper, P., Wirsing, M., Existential Quantifiers in Abstract Data Types, Proc. 6th ICALP, Graz 1979, Springer LNCS 71, pp. 73-87

[BW 82] Broy, M., Wirsing, M., Partial Abstract Types, Acta Informatica 18, 1982, pp. 47-64

[BWP 84] Broy, M., Wirsing, M., Pair, C., A Systematic Study of Models of Abstract Data Types, Theoretical Computer Science 33, 2 and 3 (1984), pp. 139-174

[CIP 87] CIP-Group, The Munich Project CIP, Vol. 2, The Program Transformation System CIP-S, Springer LNCS 292 (1987)

[EM 85] Ehrig, H., Mahr, B., Fundamentals of Algebraic Specification 1, Equations and Initial Semantics, EATCS Monographs on Theoretical Computer Science, Vol. 6, Springer 1985

[EM 90] Ehrig, H., Mahr, B., Fundamentals of Algebraic Specification 2, Module Specifications and Constraints, EATCS Monographs on Theoretical Computer Science, Vol. 21, Springer 1990

[Ga 83] Ganzinger, H., Parameterized Specifications: Parameter Passing and Implementation with Respect to Observability, ACM TOPLAS, Vol. 5, No. 3, July 1983, 318-354

[GGM 76] Giarratana, V., Gimona, F., Montanari, U., Observability Concepts in Abstract Data Type Specifications, 5th Symposium Math. Foundations of Computer Science (1976), Springer LNCS 45, 576-587

[GTW 78] Goguen, J., Thatcher, J., Wagner, E., An Initial Algebra Approach to the Specification, Correctness and Implementation of Abstract Data Types, Current Trends in Programming Methodology IV, Yeh, R. (Ed.), Prentice-Hall, Englewood Cliffs, 1978, pp. 80-149

[Go 82] Goldblatt, R., Axiomatising the Logic of Computer Programming, Springer LNCS 130

[Gu 76] Guttag, J.V., Abstract Data Types and The Development of Data Structures, Supplement to Proc. Conference on Data Abstraction, Definition, and Structure, SIGPLAN Notices 8 (1976)

[HR 80] Hornung, G., Raulefs, P., Terminal Algebra Semantics and Retractions for Abstract Data Types, DeBakker, van Leuwen, 7th ICALP, Springer LNCS 85, pp. 310-323 (1980)

HRS 88] Heisel, M., Reif, W., Stephan, W., Implementing Verification Strategies in the KIV-System, Proc. 9th International Conference on Automated Deduction, E. Lusk, R. Overbeek (eds), Springer LNCS 310 (1988), pp. 131-140

[HRS 89] Heisel, M., Reif, W., Stephan, W., Formal Software Development in the KIV-System, Proc. Workshop on Automating Software Design, IJCAI-89, Kestrel Institute, Palo Alto (1989), pp. 115 - 124, and AAAI press 1991.

[HRS 90] Heisel, M., Reif, W., Stephan, W., Tactical Theorem Proving in Program Verification, 10th International Conference on Automated Deduction, Kaiserslautern, FRG, July 1990, Springer LNCS 449, pp. 117-131

[Ka 83] Kamin, S., Final Data Types and Their Specification, ACM TOPLAS 5,1 1983, pp. 97-123

[LZ 74] Liskov, B.H., Zilles, S.N., Programming with Abstract Data Types, SIGPLAN Notices 6 (1974), pp. 50-59

[Pa 83] Padawitz, P., Correctness, Completeness, and Consistency of Equational Data Type Specifications, Technische Universität Berlin, Fachbereich 20 (Informatik), Bericht Nr. 83-15

[Pa 85] Padawitz, P., Parameter Preserving Data Type Specifications, Journal of Computer and System Sciences 34, 179-209

[Pa 90] Padawitz, P., Horn Logic and Rewriting for Functional and Logic Program Design, Universität Passau, Fakultät für Mathematik und Informatik, MIP-9002, 3/1990

[Re 91] Reif, W., Correctness of Specifications and Generic Modules, Dissertation, University of Karlsruhe, 1991 (in German)

[SA 91] Sperschneider, V., Antoniou, G., Logic : A Foundation for Computer Science, Addison Wesley, 1991.

[TWW 82] Thatcher, J., Wagner, E., Wright, J., Data Type Specification: Parametrization and the Power of Specification Techniques, ACM TOPLAS 4,4 1982, pp. 711-732

[Wa 79] Wand, M., Final Algebra Semantics and Data Type Extensions, Journal of Computer and System Sciences 19, 1, 1979, pp. 27-44

[WPPDB 83] Wirsing, M., Pepper, P., Partsch, H., Dosch, W., Broy, M., On Hierarchies of Abstract Data Types, Acta Informatica 20 (1983), pp. 1-33